Analyse und Verifikation

LVA 185.276, VU 2.0, ECTS 3.0
SS 2021

(Stand: 09.06.2021)

Jens Knoop

Technische Universität Wien
Information Systems Engineering
Compilers and Languages

Inhaltsverzeichnis

Inhaltsverzeichnis (1)

Teil I: Motivation

Inhaltsverzeichnis (2)

Teil II: Semantik

Inhaltsverzeichnis (3)

Teil III: Verifikation

Inhaltsverzeichnis (4)

Inhaltsverzeichnis (5)

Teil IV: Analyse

Inhaltsverzeichnis (6)

Inhaltsverzeichnis (7)

Inhaltsverzeichnis (8)

Inhaltsverzeichnis (9)

Inhaltsverzeichnis (10)

Inhaltsverzeichnis (11)

Inhaltsverzeichnis (12)

Inhaltsverzeichnis (13)

Inhaltsverzeichnis (14)

Inhaltsverzeichnis (15)

Inhaltsverzeichnis (16)

Inhaltsverzeichnis (17)

Inhaltsverzeichnis (18)

Inhaltsverzeichnis (19)

Teil VI: Abstrakte Interpretation und Modellprüfung

Inhaltsverzeichnis (20)

Inhaltsverzeichnis (21)

Anhänge

Inhaltsverzeichnis (22)

Inhaltsverzeichnis (23)

This software comes “without warranty of any kind, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose.”

Teil I

Motivation

Kapitel 1

Einführung

Kapitel 1.1

Syntax, Semantik

Syntax und Semantik

...legen Form und Bedeutung von Programmiersprachen fest.

- Syntax: Regelwerk zur präzisen Beschreibung wohlgeformter Programme.
- Semantik: Regelwerk zur präzisen Beschreibung der Bedeutung oder des Verhaltens wohlgeformter Programme oder Programmteile (aber auch von Hardware).

Vorteilhaft: Syntax und Semantik durch

- formale Regelwerke

festzulegen.

Vorteil

...formaler Regelwerke: Rigorosität!

Die (mathematische) Rigorosität formaler Regelwerke für Syntax und Semantik von Programmiersprachen

- erlaubt Mehrdeutigkeiten, Über- und Unterspezifikationen natürlichsprachlicher Beschreibungen in Syntax und Semantik aufzudecken und aufzulösen.
- schafft die Grundlage für vertrauenswürdige Implementierungen der Programmiersprache, für die Analyse, Verifikation und Transformation von Programmen.

Als Programmiersprache werden wir anfänglich die Modellsprache WHILE betrachten, anhand derer wir die Festlegung von Syntax und Semantik mit formalen Regelwerken beispielhaft illustrieren und demonstrieren.

Kapitel 1.2

Modellsprache WHILE

While

...der sog. 'while'-Kern imperativer Programmiersprachen, besitzt:

- Zuweisungen (einschließlich der leeren Anweisung)
- Fallunterscheidungen
- while-Schleifen (namensgebend für die Sprache)
- Sequentielle Komposition

Beachte: While ist 'schlank', doch Turing-mächtig!

Syntax von WHILE

...festgelegt durch folgende Backus-Naur-Regel (BNF), die die Menge wohlgeformter WHILE -Programme beschreibt:

$$
\begin{array}{lll}
\pi & ::= & x := a \qquad \text{(Zuweisung)} \\
& | & \mathit{skip} \qquad \text{(Leere Anweisung)} \\
& | & \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi} \qquad \text{(Fallunterscheidung)} \\
& | & \text{while } b \text{ do } \pi_1 \text{ od} \qquad \text{(while-Schleife)} \\
& | & \pi_1; \pi_2 \qquad \text{(Sequentielle Komposition)} \\
& | & (\pi_1) \qquad \text{(Klammerung)}
\end{array}
$$

wobei

- a für arithmetische Ausdrücke über Numeralen, Variablen und arithmetischen Operatoren
- b für Wahrheitswertausdrücke über Booleschen Konstantensymbolen, arithmetischen Relatoren und logischen Operatoren

stehen.

Syntax von Numeralen und Ausdrücken

...beschrieben durch folgende BNF-Regeln.

Numerale (Zahlwörter)

$$
\begin{array}{lcll}
z & ::= & 0 \mid 1 \mid 2 \mid \ldots \mid 9 & \text{(Ziffer)} \\
n & ::= & z \mid n\,z & \text{(Numeral)}
\end{array}
$$

Arithmetische Ausdrücke

$$
\begin{array}{lcll}
a & ::= & n & \text{(Numeral)} \\
 & & \mid x & \text{(Variable)} \\
 & & \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \mid a_1/a_2 \mid \ldots &
\end{array}
$$

Wahrheitswertausdrücke (Boolesche Ausdrücke)

$$
\begin{array}{lcll}
b & ::= & \mathit{true} & \text{(Konstantensymbol)} \\
 & & \mid \mathit{false} & \text{(Konstantensymbol)} \\
 & & \mid a_1 = a_2 \mid a_1 \neq a_2 & \\
 & & \mid a_1 < a_2 \mid a_1 \leq a_2 \mid \ldots & \\
 & & \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b_1 &
\end{array}
$$

Bezeichnungen

- **Var**, Menge der Variablen, $x \in$ **Var**
- **Num**, Menge der Zahlwörter, $n \in$ **Num**
- **Aexpr**, Menge arithmetischer Ausdrücke, $a \in$ **Aexpr**
- **Bexpr**, Menge Boolescher Ausdrücke, $b \in$ **Bexpr**
- **Prg**, Menge aller WHILE -Programme, $\pi \in$ **Prg**

Die Semantikfestlegung für WHILE

...stützt sich auf die Bedeutung (oder: Semantik) von

- Zahlwörtern
- arithmetische Ausdrücken
- Wahrheitswertausdrücken

und den Begriff von

- Speicherzuständen

Wir legen deshalb zunächst diese Begriffe fest.

Kapitel 1.3

Semantik von Numeralen

Semantik von Numeralen (oder Zahlwörtern)

...gegeben durch eine induktiv definierte totale Abbildung

$$[\![\, . \,]\!]_N : \mathbf{Num} \to \mathbb{Z}$$

definiert durch:

$$\begin{array}{lll} [\![\, 0 \,]\!]_N & =_{df} & \mathbf{0} \\ \ldots & \ldots & \ldots \\ [\![\, 9 \,]\!]_N & =_{df} & \mathbf{9} \\ [\![\, n\,i \,]\!]_N & =_{df} & ([\![\, n \,]\!]_N \ \mathbf{mal\ 10})\ \mathbf{plus}\ [\![\, i \,]\!]_N,\ i \in \{0, \ldots, 9\} \\ [\![\, -n \,]\!]_N & =_{df} & \mathbf{0\ minus}\ [\![\, n \,]\!]_N \end{array}$$

wobei

$$\begin{array}{lll} \mathbf{plus} & : & \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \quad \text{(Addition auf } \mathbb{Z}\text{)} \\ \mathbf{mal} & : & \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \quad \text{(Multiplikation auf } \mathbb{Z}\text{)} \\ \mathbf{minus} & : & \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \quad \text{(Subtraktion auf } \mathbb{Z}\text{)} \end{array}$$

Bemerkung: – Zeichen $=_{df}$ steht für 'definitionsgemäß gleich'.
– Operationen **plus**, **mal**,... werden infix-verwendet.

Bemerkung: Syntakt. vs. Semant. Entitäten

Syntaktische Entitäten

- 0, 1, 2, ... bezeichnen syntaktische Entitäten, Darstellungen von Zahlen.
- $-$ bezeichnet eine syntaktische Entität, die Darstellung eines (syntaktischen) Operators (dem als Semantik der 'Vorzeichenwechsel' zugeordnet wird).

Semantische Entitäten

- $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, ... bezeichnen semantische Entitäten, hier ganze Zahlen: $\mathbb{Z} =_{df} \{\ldots, \mathbf{-2}, \mathbf{-1}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$.
- **plus**, **mal**, **minus** : $\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, bezeichnen (semantische) Operationen, hier die übliche Addition, Multiplikation und Subtraktion auf $\mathbb{Z}$.

Beachte: Die Semantik von Numeralen ist zustandsunabhängig.

Kapitel 1.4

Semantik arithmetischer Ausdrücke

Semantik arithmetischer Ausdrücke (1)

...gegeben durch eine induktiv definierte totale Abbildung

$$[\![\, . \,]\!]_A : \mathbf{Aexpr} \to (\Sigma \hookrightarrow \mathbb{Z})$$

mit

- $\mathbb{Z} =_{df} \{\ldots, \mathbf{-2}, \mathbf{-1}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$ Menge ganzer Zahlen
- $\Sigma =_{df} \{ \sigma \mid \sigma : \mathbf{Var} \to \mathbb{Z} \}$ Menge der Zustände (oder: Speicherzustände) über $\mathbb{Z}$

Notationelle Konvention: Der gerade Pfeil $\to$ bezeichnet totale Funktionen, der Hakenpfeil $\hookrightarrow$ partielle Funktionen.

Semantik arithmetischer Ausdrücke (2)

...wobei $[\![.]\!]_A : \textbf{Aexpr} \to (\Sigma \hookrightarrow \mathbb{Z})$ definiert ist durch:

$$
\begin{array}{lll}
[\![n]\!]_A & =_{df} & \lambda\sigma.\ [\![n]\!]_N \quad \text{(Wert von } n \text{ zustandsunabhängig!)} \\
[\![x]\!]_A & =_{df} & \lambda\sigma.\ \sigma(x) \quad \text{(Wert von } x \text{ zustandsabhängig!)} \\
[\![a_1 + a_2]\!]_A & =_{df} & \lambda\sigma.\ [\![a_1]\!]_A(\sigma)\ \textbf{plus}\ [\![a_2]\!]_A(\sigma) \\
[\![a_1 * a_2]\!]_A & =_{df} & \lambda\sigma.\ [\![a_1]\!]_A(\sigma)\ \textbf{mal}\ [\![a_2]\!]_A(\sigma) \\
[\![a_1 - a_2]\!]_A & =_{df} & \lambda\sigma.\ [\![a_1]\!]_A(\sigma)\ \textbf{minus}\ [\![a_2]\!]_A(\sigma)
\end{array}
$$

...weitere arithmetische Operatoren ($/$, *mod*, $\hat{}$, ...) analog.

Beachte auch hier wieder den Unterschied zwischen syntaktischen und semantischen Entitäten:

- $+$, $*$, $-$ bezeichnen syntaktische Entitäten: Operatoren.
- **plus**, **mal**, **minus** bezeichnen semantische Entitäten: Operationen (hier auf $\mathbb{Z}$).

Kapitel 1.5

Semantik Boolescher Ausdrücke

Semantik Boolescher Ausdrücke (1)

...gegeben durch eine induktiv definierte totale Abbildung

$$[\![\, . \,]\!]_B : \mathbf{Bexpr} \to (\Sigma \hookrightarrow \mathbb{B})$$

mit

- $\mathbb{B} =_{df} \{\mathbf{wahr}, \mathbf{falsch}\}$ Menge der Wahrheitswerte
- $\mathbb{Z} =_{df} \{\ldots, \mathbf{-2}, \mathbf{-1}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$ Menge ganzer Zahlen
- $\Sigma =_{df} \{\, \sigma \mid \sigma : \mathbf{Var} \to \mathbb{Z} \,\}$ Menge der Zustände (oder Speicherzustände) über $\mathbb{Z}$

Semantik Boolescher Ausdrücke (2)

...wobei $[\![\, . \,]\!]_B : \mathbf{Bexpr} \to (\Sigma \hookrightarrow \mathbb{B})$ definiert ist durch:

$$
\begin{array}{lll}
[\![\mathit{wahr}]\!]_B & =_{df} & \lambda\sigma.\ \mathbf{wahr} \\
[\![\mathit{falsch}]\!]_B & =_{df} & \lambda\sigma.\ \mathbf{falsch} \\
[\![a_1 = a_2]\!]_B & =_{df} & \lambda\sigma.\ \begin{cases} \mathbf{wahr} & \text{falls } [\![a_1]\!]_A(\sigma) = [\![a_2]\!]_A(\sigma) \\ \mathbf{falsch} & \text{sonst} \end{cases} \\
[\![a_1 \neq a_2]\!]_B & =_{df} & \lambda\sigma.\ \begin{cases} \mathbf{wahr} & \text{falls } [\![a_1]\!]_A(\sigma) \neq [\![a_2]\!]_A(\sigma) \\ \mathbf{falsch} & \text{sonst} \end{cases}
\end{array}
$$

...weitere arithmetische Relatoren $(>, \geq, <, \leq, \ldots)$ analog.

$$
\begin{array}{lll}
[\![b_1 \wedge b_2]\!]_B & =_{df} & \lambda\sigma.\ [\![b_1]\!]_B(\sigma)\ \mathbf{und}\ [\![b_2]\!]_B(\sigma) \\
[\![b_1 \vee b_2]\!]_B & =_{df} & \lambda\sigma.\ [\![b_1]\!]_B(\sigma)\ \mathbf{oder}\ [\![b_2]\!]_B(\sigma) \\
[\![\neg b]\!]_B & =_{df} & \lambda\sigma.\ \mathbf{nicht}\,([\![b]\!]_B(\sigma)) \\
[\![b_1 = b_2]\!]_B & =_{df} & \lambda\sigma.\ \begin{cases} \mathbf{wahr} & \text{falls } [\![b_1]\!]_B(\sigma) = [\![b_2]\!]_B(\sigma) \\ \mathbf{falsch} & \text{sonst} \end{cases} \\
[\![b_1 \neq b_2]\!]_B & =_{df} & \lambda\sigma.\ \mathbf{nicht}\,([\![b_1 = b_2]\!]_B(\sigma))
\end{array}
$$

Semantik Boolescher Ausdrücke (3)

Dabei bezeichnen:

und	:	$\mathbb{B} \times \mathbb{B} \to \mathbb{B}$	(Logische Konjunktion)
oder	:	$\mathbb{B} \times \mathbb{B} \to \mathbb{B}$	(Logische Disjunktion)
nicht	:	$\mathbb{B} \to \mathbb{B}$	(Logische Negation)
$=$	:	$\mathbb{D} \times \mathbb{D} \to \mathbb{B}, \mathbb{D} \in \{\mathbb{Z}, \mathbb{B}\}$	(Gleichheitsrelation auf $\mathbb{Z}, \mathbb{B}$)

Bemerkung:

- Beachte die Überladung der
  - Relatorsymbole $=$ und $\neq$.
  - Relationssymbole $\mathbf{=}$ und $\mathbf{\neq}$.
- Statt $\mathbf{=}, \mathbf{\neq}$ werden wir später oft einfacher $=$, $\neq$ schreiben; der Kontext macht jeweils deutlich, ob z.B. $=$ als
  - Relatorsymbol wie in $[\![a_1 = a_2]\!]_B(\sigma)$
  - Relationssymbol wie in $[\![a_1]\!]_A(\sigma) \mathbf{=} [\![a_2]\!]_A(\sigma)$

  verwendet wird.
- Operationen **und**, **oder** und Relationen $\mathbf{=}, \mathbf{\neq}$,... werden infix-verwendet.

Semantik Boolescher Ausdrücke (4)

Beachte wieder den Unterschied zwischen syntaktischen und semantischen Entitäten:

Syntaktische Entitäten:

- *wahr*, *falsch* bezeichnen syntaktische Entitäten: Konstantensymbole.
- $=$, $\neq$, $>$, etc. bezeichnen syntaktische Entitäten: Relatoren.
- $\wedge$, $\vee$, $\neg$ bezeichnen syntaktische Entitäten: Operatoren.

Semantische Entitäten:

- **wahr**, **falsch** bezeichnen semantische Entitäten: Wahrheitswerte (aus $\mathbb{B}$).
- $=$, $\neq$, etc. bezeichnen semantische Entitäten: Relationen (auf $\mathbb{Z}$ bzw. $\mathbb{B}$).
- **und**, **oder**, **nicht** bezeichnen semantische Entitäten: Operationen (auf $\mathbb{B}$).

Kapitel 1.6

Eigenschaften von $[\![\ ]\!]_N$, $[\![\ ]\!]_A$, $[\![\ ]\!]_B$

Freie Variablen

...arithmetischer Ausdrücke:

$$\begin{aligned} FV(n) &= \emptyset \\ FV(x) &= \{x\} \\ FV(a_1 + a_2) &= FV(a_1) \cup FV(a_2) \\ &\dots \end{aligned}$$

...Boolescher Ausdrücke:

$$\begin{aligned} FV(true) &= \emptyset \\ FV(false) &= \emptyset \\ FV(a_1 = a_2) &= FV(a_1) \cup FV(a_2) \\ &\dots \\ FV(b_1 \wedge b_2) &= FV(b_1) \cup FV(b_2) \\ FV(b_1 \vee b_2) &= FV(b_1) \cup FV(b_2) \\ FV(\neg b_1) &= FV(b_1) \end{aligned}$$

Eigenschaften von $[\![\ ]\!]_N$, $[\![\ ]\!]_A$, $[\![\ ]\!]_B$

Lemma 1.6.1

Sei $n \in \mathbf{Num}$ und $\sigma, \sigma' \in \Sigma$. Dann gilt:
$[\![n]\!]_A(\sigma) = [\![n]\!]_A(\sigma') = [\![n]\!]_N$.

Lemma 1.6.2

Sei $a \in \mathbf{Aexpr}$ und $\sigma, \sigma' \in \Sigma$ mit $\sigma(x) = \sigma'(x)$ für alle $x \in FV(a)$. Dann gilt: $[\![a]\!]_A(\sigma) = [\![a]\!]_A(\sigma')$.

Lemma 1.6.3

Sei $b \in \mathbf{Bexpr}$ und $\sigma, \sigma' \in \Sigma$ mit $\sigma(x) = \sigma'(x)$ für alle $x \in FV(b)$. Dann gilt: $[\![b]\!]_B(\sigma) = [\![b]\!]_B(\sigma')$.

Beachte: Die Gleichheitsrelation $=$ auf $\mathbb{Z}$ und $\mathbb{B}$ wird hier bereits einfacher mit dem überladenen Symbol $=$ bezeichnet.

Übungsaufgabe 1.6.4

Was besagen

1. Lemma 1.6.1
2. Lemma 1.6.2
3. Lemma 1.6.3

anschaulich?

Kapitel 1.7

Syntaktische und semantische Substitution

Substitution

...ein Begriff von zentraler Bedeutung in zwei Varianten:

- Syntaktische Substitution
- Semantische Substitution

Das

- Substitutionslemma 1.7.3

beschreibt den Zusammenhang zwischen syntaktischer und semantischer Substitution.

Syntaktische Substitution

...für arithmetische Audrücke.

Definition 1.7.1 (Syntaktische Substitution)

Die syntaktische Substitution für arithmetische Ausdrücke ist eine dreistellige Abbildung

$$\cdot[\cdot/\cdot] : \mathbf{Aexpr} \times \mathbf{Aexpr} \times \mathbf{Var} \to \mathbf{Aexpr}$$

die induktiv definiert ist durch:

$\forall\, a, a' \in \mathbf{Aexpr}.\ \forall\, x \in \mathbf{Var}.$

$$\begin{array}{lcll} n[a'/x] & =_{df} & n & \text{falls } a = n \in \mathbf{Num} \\ y[a'/x] & =_{df} & \begin{cases} a' & \text{falls } y = x \\ y & \text{sonst} \end{cases} & \text{falls } a = y \in \mathbf{Var} \\ (a_1\ op\ a_2)[a'/x] & =_{df} & (a_1[a'/x]\ op\ a_2[a'/x]) & \text{falls } a = (a_1\, op\, a_2), \\ & & & op \in \{+, *, -, \ldots\} \end{array}$$

Semantische Substitution

...für arithmetische Ausdrücke.

Definition 1.7.2 (Semantische Substitution)

Die semantische Substitution für arithmetische Ausdrücke ist eine dreistellige Abbildung

$$\cdot[\cdot/\cdot] : \Sigma \times \mathbb{Z} \times \mathbf{Var} \to \Sigma$$

die definiert ist durch:

$$\forall\, \sigma \in \Sigma.\, \forall\, \mathbf{z} \in \mathbb{Z}.\, \forall\, x \in \mathbf{Var}.\, \sigma[\mathbf{z}/x](y) =_{df} \begin{cases} \mathbf{z} & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

Substitutionslemma für arithmet. Ausdrücke

...Zusammenhang syntaktischer und semantischer Substitution für arithmetische Ausdrücke:

Lemma 1.7.3 (Substitutionslemma für $[\![\ ]\!]_A$)

$$\forall\, a, a' \in \mathbf{Aexpr}.\ \forall\, \sigma \in \Sigma.\ \underbrace{[\![\, a[a'/x] \,]\!]_A(\sigma)}_{\text{Substituierter Ausdruck}} = [\![\, a \,]\!]_A(\underbrace{\sigma[[\![\, a' \,]\!]_A(\sigma)/x]}_{\text{Substituierter Zustand}})$$

wobei

- $[\, a'/x \,]$ die syntaktische Substitution
- $[\, [\![\, a' \,]\!]_A(\sigma)/x \,]$ die semantische Substitution

bezeichnen.

Substitutionslemma für Boolesche Ausdrücke

...die Begriffe syntaktischer und semantischer Substitution lassen sich analog für Boolesche Ausdrücke definieren.

...für den Zusammenhang syntaktischer und semantischer Substitution für Boolesche Ausdrücke erhalten wir:

Lemma 1.7.4 (Substitutionslemma für $[\![\ ]\!]_B$)

$$\forall\, b \in \mathbf{Bexpr}.\ \forall\, a' \in \mathbf{Aexpr}.\ \forall\, \sigma \in \Sigma.$$

$$\underbrace{[\![\, b[a'/x] \,]\!]_B(\sigma)}_{\substack{\text{Substituierter} \\ \text{Ausdruck}}} = [\![\, b \,]\!]_B(\underbrace{\sigma[\,[\![\, a' \,]\!]_A(\sigma)/x\,]}_{\substack{\text{Substituierter} \\ \text{Zustand}}})$$

Kapitel 1.8

Induktive Beweisprinzipien

Grundlegende induktive Beweisprinzipien

...für den Beweis von Eigenschaften und Aussagen wie in Kapitel 1.6 und 1.7:

- Vollständige Induktion
- Verallgemeinerte Induktion
- Strukturelle Induktion

Chapter 1.8.1

Vollständige Induktion

Das Prinzip vollständiger Induktion

Sei IN die Menge natürlicher Zahlen und E eine Eigenschaft natürlicher Zahlen.

Das Prinzip vollständiger Induktion:

$$\underbrace{E(1)}_{\text{Induktionsanfang}} \wedge [\overbrace{\forall n \in \mathbb{N}.\ \underbrace{E(n)}_{\text{Induktionshypothese}} \Rightarrow E(n+1)}^{\text{Induktiver Fall}}] \Rightarrow \underbrace{\forall\, n \in \mathbb{N}.\ E(n)}_{\text{Folgerung}}$$

Induktionsschritt

Beispiel: Illustration vollständiger Induktion

Lemma 1.8.1.1

$$\forall\, n \in \mathbb{N}.\ \sum_{k=1}^{n}(2k-1) = n^2$$

Beweis durch vollständige Induktion.

Beweis von Lemma 1.8.1.1 (1)

Induktionsanfang: Sei $n = 1$. In diesem Fall erhalten wir die Gleichheit von linker und rechter Seite der Aussage des Lemmas wie folgt:

$$\begin{aligned}
\sum_{k=1}^{n}(2k-1) &= \sum_{k=1}^{1}(2k-1) \\
&= 2*1-1 \\
&= 2-1 \\
&= 1 \\
&= 1^2 \\
&= n^2
\end{aligned}$$

Beweis von Lemma 1.8.1.1 (2)

Induktionsschritt: Sei $n \in \mathbb{N}$. Aufgrund der Induktionshypothese (IH) können wir die Gleichheit $\sum_{k=1}^{n}(2k-1) = n^2$ annehmen. Damit können wir den Beweis wie folgt vervollständigen:

$$\begin{aligned}
\sum_{k=1}^{n+1}(2k-1) &= 2(n+1) - 1 + \sum_{k=1}^{n}(2k-1) \\
\overset{(IH)}{} &= 2(n+1) - 1 + n^2 \\
&= 2n + 2 - 1 + n^2 \\
&= 2n + 1 + n^2 \\
&= n^2 + 2n + 1 \\
&= n^2 + n + n + 1 \\
&= (n+1)(n+1) \\
&= (n+1)^2
\end{aligned}$$

□

Übungsaufgabe 1.8.1.2

Beweise durch vollständige Induktion:

Lemma 1.8.1.3

1. $$\forall\, n \in \mathbb{N}.\ \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

2. $$\forall\, n \in \mathbb{N}.\ \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

3. $$\forall\, n \in \mathbb{N}.\ \sum_{k=1}^{n} k^3 = \left(\frac{n(n+1)}{2}\right)^2$$

Chapter 1.8.2

Verallgemeinerte Induktion

Das Prinzip verallgemeinerter Induktion

Sei IN die Menge natürlicher Zahlen und E eine Eigenschaft natürlicher Zahlen.

Das Prinzip verallgemeinerter Induktion:

$$\forall n \in \mathbb{N}.\ [\overbrace{(\forall m < n.\ E(m))}^{\text{Induktions-hypothese}} \underbrace{\Rightarrow E(n)}_{\text{Induktions-schritt}}] \quad \Rightarrow \quad \underbrace{\forall\, n \in \mathbb{N}.\ E(n)}_{\text{Folgerung}}$$

(Induktiver) Fall

Beachte: Für die kleinste natürliche Zahl $\hat{n}$ ($\mathbb{N}_0$ vs. $\mathbb{N}_1$) reduziert sich die Induktionshypothese auf 'wahr', d.h. $E(\hat{n})$ muss ohne Rückgriff auf besondere Voraussetzungen bewiesen werden.

Bsp: Illustration verallgemeinerter Induktion

Die Fibonacci-Funktion $fib : \mathbb{N}_0 \to \mathbb{N}_0$ ist definiert durch:

$$\forall\, n \in \mathbb{N}_0.\ fib(n) =_{df} \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ fib(n-1) + fib(n-2) & \text{falls } n \geq 2 \end{cases}$$

Lemma 1.8.2.1

$$\forall\, n \in \mathbb{N}_0.\ fib(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Beweis durch verallgemeinerte Induktion.

Schlüssel z. Beweis v. Lemma 1.8.2.1 f. $n \geq 2$

...ist es, gemäß der Induktionshypothese (IH) für $m = n - 1$ und $m = n - 2$ die Gleichheit

$$fib(m) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^m - \left(\frac{1-\sqrt{5}}{2}\right)^m}{\sqrt{5}}$$

auszunutzen.

(Beachte: Für $n \geq 2$ könnten wir diese Gleichheit aufgrund der Induktionshypothese sogar für alle $m < n$ ausnutzen (statt nur für $m = n - 1$ und $m = n - 2$), was aber nicht erforderlich ist, um den Beweis erfolgreich abzuschließen.)

Beweis von Lemma 1.8.2.1 (1)

Fall 1: Sei $n = 0$. Wir erhalten wie gewünscht:

$$fib(0) = 0 = \frac{0}{\sqrt{5}} = \frac{1-1}{\sqrt{5}} = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^0 - \left(\frac{1-\sqrt{5}}{2}\right)^0}{\sqrt{5}}$$

(Beachte: Für den Beweis von Fall 1 liefert uns die Induktionshypothese nichts über die Gültigkeit der Aussage des Lemmas; glücklicherweise wird auch nichts benötigt.)

Fall 2: Sei $n = 1$. Wir erhalten auch hier wie gewünscht:

$$fib(1) = 1 = \frac{\sqrt{5}}{\sqrt{5}} = \frac{\frac{1}{2} + \frac{\sqrt{5}}{2} - \left(\frac{1}{2} - \frac{\sqrt{5}}{2}\right)}{\sqrt{5}} = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^1 - \left(\frac{1-\sqrt{5}}{2}\right)^1}{\sqrt{5}}$$

(Beachte: Für den Beweis von Fall 2 hätten wir aufgrund der Induktionshypothese die Aussage des Lemmas für $n = 0$ ausnutzen können; das ist aber nicht erforderlich.)

Beweis von Lemma 1.8.2.1 (2)

Fall 3: Sei $n \geq 2$. Mithilfe der Ind.-Hypothese f. $n-2$, $n-1$ erhalten wir:

$$
\begin{aligned}
fib(n) &= fib(n-2) + fib(n-1) \\
\text{(2x IH)} &= \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{n-2} - \left(\frac{1-\sqrt{5}}{2}\right)^{n-2}}{\sqrt{5}} + \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{n-1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n-1}}{\sqrt{5}} \\
&= \frac{\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n-2} + \left(\frac{1+\sqrt{5}}{2}\right)^{n-1}\right] - \left[\left(\frac{1-\sqrt{5}}{2}\right)^{n-2} + \left(\frac{1-\sqrt{5}}{2}\right)^{n-1}\right]}{\sqrt{5}} \\
&= \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{n-2}\left[1 + \frac{1+\sqrt{5}}{2}\right] - \left(\frac{1-\sqrt{5}}{2}\right)^{n-2}\left[1 + \frac{1-\sqrt{5}}{2}\right]}{\sqrt{5}} \\
(*) &= \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{n-2}\left(\frac{1+\sqrt{5}}{2}\right)^{2} - \left(\frac{1-\sqrt{5}}{2}\right)^{n-2}\left(\frac{1-\sqrt{5}}{2}\right)^{2}}{\sqrt{5}} \\
&= \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{n} - \left(\frac{1-\sqrt{5}}{2}\right)^{n}}{\sqrt{5}}
\end{aligned}
$$

□

Beweis von $(*)$

Gleichheit $(*)$ gilt aufgrund der Gleichheiten (1) and (2):

$$\left(\frac{1+\sqrt{5}}{2}\right)^2 = 1+\frac{1+\sqrt{5}}{2} \tag{1}$$

$$\left(\frac{1-\sqrt{5}}{2}\right)^2 = 1+\frac{1-\sqrt{5}}{2} \tag{2}$$

die sich mithilfe der Binomialformeln (BF)zeigen lassen.

$$\left(\frac{1+\sqrt{5}}{2}\right)^2 \overset{(BF)}{=} \frac{1+2\sqrt{5}+5}{4} = \frac{6+2\sqrt{5}}{4} = \frac{3+\sqrt{5}}{2} = 1+\frac{1+\sqrt{5}}{2}$$

$$\left(\frac{1-\sqrt{5}}{2}\right)^2 \overset{(BF)}{=} \frac{1-2\sqrt{5}+5}{4} = \frac{6-2\sqrt{5}}{4} = \frac{3-\sqrt{5}}{2} = 1+\frac{1-\sqrt{5}}{2}$$

Übungsaufgabe 1.8.2.2

Sei Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ definiert durch:

$$\forall\, n \in \mathbb{N}_0.\ f(n) =_{df} \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ \sum_{k=0}^{n-1} f(k) & \text{falls } n \geq 2 \end{cases}$$

Beweise durch vollständige Induktion:

Lemma 1.8.2.3

$$(\forall\, n \in \mathbb{N}.\ n \geq 3).\ \sum_{k=0}^{n-3} 2^k = 2^{n-2} - 1$$

Beweise mit verallgemeinerter Induktion (und Lemma 1.8.2.3):

Lemma 1.8.2.4

$$(\forall\, n \in \mathbb{N}.\ n \geq 2).\ f(n) = 2^{n-2}$$

Kapitel 1.8.3

Strukturelle Induktion

Das Prinzip struktureller Induktion

Seien A und O eine Menge von Atomen und Operatoren; sei M die Menge der aus Elementen von A und O induktiv konstruierbaren Elemente. Bezeichne $sub(m) \subseteq M$, $m \in M$, die Menge der Elemente, aus denen m konstruiert ist, und sei E eine Eigenschaft der Elemente von M.

Das Prinzip struktureller Induktion:

$$\forall\, m \in M.\, [\overbrace{\underbrace{(\forall\, m' \in sub(m).\, E(m'))}_{\text{Induktions-hypothese}} \underbrace{\Rightarrow E(m)}_{\text{Induktions-schritt}}}^{\text{(Induktiver) Fall}}] \Rightarrow \underbrace{\forall\, m \in M.\, E(m)}_{\text{Folgerung}}$$

Beachte: Für die Atome $\hat{m}$ aus M, die 'einfachsten' Elemente von M, gilt $sub(\hat{m}) = \emptyset$. Für diese Elemente reduziert sich die Induktionshypothese auf 'wahr', d.h. $E(\hat{m})$ muss ohne besondere Voraussetzungen bewiesen werden.

Beispiel: Lemma 1.6.2

...zur Bequemlichkeit hier wiederholt:

Lemma 1.6.2

Sei $a \in$ **Aexpr** und $\sigma, \sigma' \in \Sigma$ mit $\sigma(x) = \sigma'(x)$ für alle $x \in FV(a)$. Dann gilt:

$$[\![a]\!]_A(\sigma) = [\![a]\!]_A(\sigma')$$

Beweis durch strukturelle Induktion (über den induktiven Aufbau arithmetischer Ausdrücke).

Beweis von Lemma 1.6.2 (1)

Sei $a \in \mathbf{Aexpr}$ und seien $\sigma, \sigma' \in \Sigma$ mit $\sigma(x) = \sigma'(x)$ für alle $x \in FV(a)$.

Fall 1: Sei $a = n$, $n \in \mathbf{Num}$. Mithilfe der Definitionen von $[\![\ ]\!]_A$ und $[\![\ ]\!]_N$ erhalten wir unmittelbar die Gleichheit der rechten und linken Seite der Aussage des Lemmas:

$$[\![a]\!]_A(\sigma) = [\![n]\!]_A(\sigma) = [\![n]\!]_N = [\![n]\!]_A(\sigma') = [\![a]\!]_A(\sigma')$$

Fall 2: Sei $a = x$, $x \in \mathbf{Var}$. Wie in Fall 1) erhalten wir auch hier mithilfe der Definition von $[\![\ ]\!]_A$ die geforderte Gleichheit von rechter und linker Seite der Aussage des Lemmas:

$$[\![a]\!]_A(\sigma) = [\![x]\!]_A(\sigma) = \sigma(x) = \sigma'(x) = [\![x]\!]_A(\sigma') = [\![a]\!]_A(\sigma')$$

Beweis von Lemma 1.6.2 (2)

Fall 3: Sei $a = a_1 + a_2$, $a_1, a_2 \in$ **Aexpr**. Aufgrund der Induktionshypothese (IH) dürfen wir die Gleichheiten $[\![a_1]\!]_A(\sigma) = [\![a_1]\!]_A(\sigma')$ und $[\![a_2]\!]_A(\sigma) = [\![a_2]\!]_A(\sigma')$ annehmen. Das erlaubt uns, den Beweis wie folgt abzuschließen:

$$
\begin{array}{rcl}
 & & [\![a]\!]_A(\sigma) \\
\text{(Wahl von } a\text{)} & = & [\![a_1 + a_2]\!]_A(\sigma) \\
\text{(Def. von } [\![\ ]\!]_A\text{)} & = & [\![a_1]\!]_A(\sigma) \ \textbf{plus} \ [\![a_2]\!]_A(\sigma) \\
\text{(IH für } a_1, a_2\text{)} & = & [\![a_1]\!]_A(\sigma') \ \textbf{plus} \ [\![a_2]\!]_A(\sigma') \\
\text{(Def. von } [\![\ ]\!]_A\text{)} & = & [\![a_1 + a_2]\!]_A(\sigma') \\
\text{(Wahl von } a\text{)} & = & [\![a]\!]_A(\sigma')
\end{array}
$$

Fälle für weitere arithmetische Operatoren: Analog. □

Übungsaufgabe 1.8.3.1

Beweise durch strukturelle Induktion (über den induktiven Aufbau Boolescher Ausdrücke):

Lemma 1.6.3

Sei $b \in \textbf{Bexpr}$ und $\sigma, \sigma' \in \Sigma$ mit $\sigma(x) = \sigma'(x)$ für alle $x \in FV(b)$. Dann gilt:

$$[\![b]\!]_B(\sigma) = [\![b]\!]_B(\sigma')$$

Übungsaufgabe 1.8.3.2

Beweise durch strukturelle Induktion über den induktiven Aufbau arithmetischer Ausdrücke):

Lemma 1.7.3 (Substitutionslemma für $[\![\ ]\!]_A$)

$$\forall\, a, a' \in \mathbf{Aexpr}.\ \forall\, \sigma \in \Sigma.\ [\![\ a[a'/x]\ ]\!]_A(\sigma) = [\![\ a\ ]\!]_A(\sigma[[\![\ a'\ ]\!]_A(\sigma)/x])$$

Beweise durch strukturelle Induktion über den induktiven Aufbau Boolescher Ausdrücke):

Lemma 1.7.4 (Substitutionslemma für $[\![\ ]\!]_B$)

$$\forall\, b \in \mathbf{Bexpr}.\ \forall\, a' \in \mathbf{Aexpr}.\ \forall\, \sigma \in \Sigma.$$
$$[\![\ b[a'/x]\ ]\!]_B(\sigma) = [\![\ b\ ]\!]_B(\sigma[[\![\ a'\ ]\!]_A(\sigma)/x])$$

Übungsaufgabe 1.8.3.3

Was besagen

1. Substitutionslemma 1.7.3
2. Substitutionslemma 1.7.4

anschaulich?

Kapitel 1.8.4

Gleichwertigkeit

Die Prinzipien

...vollständiger Induktion:

$$E(1) \wedge [\forall n \in \mathbb{N}.\, E(n) \Rightarrow E(n+1)] \quad \Rightarrow \quad \forall\, n \in \mathbb{N}.\, E(n)$$

...verallgemeinerter Induktion:

$$\forall\, n \in \mathbb{N}.\, [(\forall\, m < n.\, E(m)) \Rightarrow E(n)] \quad \Rightarrow \quad \forall\, n \in \mathbb{N}.\, E(n)$$

...struktureller Induktion:

$$\forall\, m \in M.\, [(\forall\, m' \in \mathit{sub}(m).\, E(m')) \Rightarrow E(m)] \Rightarrow \forall\, m \in M.\, E(m)$$

sind gleich mächtig, gleich ausdruckskräftig.

Abhängig

...vom Anwendungsfall ist meist eines

- der Induktionsprinzipien unmittelbarer, einfacher und deshalb zweckmäßiger anwendbar.

Zum Beweis von Aussagen oder Eigenschaften

- über induktiv definierten Datenstrukturen ist i.a. das Prinzip struktureller Induktion

am zweckmäßigsten.

Kapitel 1.9

Ausblick

Die Semantik von WHILE

...als Bedeutung von WHILE werden wir ein Semantikfunktional

$$[\![\ ]\!] : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

festlegen, das jedem WHILE -Programm π eine (partielle) Zustandstransformation

$$[\![\ \pi\ ]\!] : \Sigma \hookrightarrow \Sigma$$

auf der Menge der Zustände über der Variablenmenge **Var** und einem geeigneten Datenbereich ID

$$\Sigma =_{df} \{\sigma \mid \sigma : \mathbf{Var} \to \mathrm{ID}\}$$

als Bedeutung zuordnet; für ID werden wir meist die Menge ganzer Zahlen $\mathbb{Z}$ betrachten.

Semantikdefinitionsstile (1)

...die Semantik einer Programmiersprache lässt sich wie deren Syntax auf verschiedene Weise festlegen. Man spricht von unterschiedlichen Definitionsstilen, die eine unterschiedliche Sicht auf die Bedeutung der Sprache gewähren und sich implizit an unterschiedliche Adressaten richten.

Von besonderer Wichtigkeit sind hier der

- operationelle (s. Kap. 2)
- denotationelle (s. Kap. 3)

und mit abweichendem Fokus

- axiomatische (s. Kap. 4)

Semantikdefinitionsstil.

Semantikdefinitionsstile (2)

- **Operationelle Semantik**

  Die Bedeutung eines (programmiersprachlichen) Konstrukts ist durch die Berechnung beschrieben, die es bei seiner Ausführung auf der Maschine induziert. Wichtig ist insbesondere, wie der Effekt der Berechnung erzeugt wird.

- **Denotationelle Semantik**

  Die Bedeutung eines Konstrukts wird durch mathematische Objekte modelliert, die den Effekt der Ausführung der Konstrukte repräsentieren. Wichtig ist einzig der Effekt, nicht wie er bewirkt wird.

- **Axiomatische Semantik**

  Bestimmte Eigenschaften des Effekts der Ausführung eines Konstrukts werden in Form von Zusicherungen ausgedrückt. Nicht relevante andere Aspekte der Ausführung werden dabei i.a. ignoriert.

Semantikdefinitionsstile (3)

...und ihre Adressaten bzw. besondere Eignung für:

Sprachimplementierersicht/Sprachimplementierung:

- Operationelle Semantik
  - Natürliche Semantik (Großschrittsemantik)
  - Strukturell operationelle Semantik (Kleinschrittsemantik)

Sprachentwicklersicht/Sprachdesign:

- Denotationelle Semantik

Verifizierersicht/Anwendungsprogrammierung

- Axiomatische Semantik
  - Beweiskalküle für partielle und totale Korrektheit
  - Korrektheit, Vollständigkeit

Zurück zur Semantik von WHILE (1)

...wir werden ein Semantikfunktional für WHILE in jedem dieser Stile einführen:

1. Operationelle Semantik (Kap. 2)
   - 1.1 Natürliche Semantik (Kap. 2.1)
     $[\![ ]\!]_{ns} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$
   - 1.2 Strukturell operationelle Semantik (Kap. 2.2)
     $[\![ ]\!]_{sos} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$
2. Denotationelle Semantik (Kap. 3)
   $[\![ ]\!]_{ds} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$

Als Hauptergebnis wird sich herausstellen, dass die eingeführten Semantiken allesamt gleich sind, d.h.:

$$[\![ ]\!]_{ns} = [\![ ]\!]_{sos} = [\![ ]\!]_{ds}$$

Zurück zur Semantik von While (2)

Die Gleichheit von $[\![\ ]\!]_{ns}$, $[\![\ ]\!]_{sos}$ und $[\![\ ]\!]_{ds}$ erlaubt es, von **der** Semantik von While zu sprechen und den Index an den Funktionalen fallen zu lassen:

$$[\![\ ]\!] : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

Bei Bedarf können wir stets die für eine Anwendung jeweils zweckmäßigste Variante wählen.

Im Anschluss an die Semantikdefinition von While werden wir die

3. Axiomatische Semantik (Kap. 4)

betrachten, die einen abweichenden Fokus auf Programmverifikation legt.

Literaturhinweise für Kapitel 1 bis 5

...als Textbücher verwendbar:

- Hanne R. Nielson, Flemming Nielson. Semantics with Applications: An Appetizer. Springer-V., 2007.
- Hanne R. Nielson, Flemming Nielson. Semantics with Applications: A Formal Introduction. Wiley Professional Computing, Wiley, 1992.

  Bem.: Eine (überarbeitete) Version ist frei erhältlich auf:
  `www.daimi.au.dk/~bra8130/Wiley_book/wiley.html`

Verwandt, aber nicht austauschbar:

- Flemming Nielson, Hanne R. Nielson. Formal Methods: An Appetizer. Springer-V., 2019.

Literaturhinweise besonders für Kapitel 4 (1)

...ein Lehrbuchklassiker:

- Jacques Loeckx, Kurt Sieber. The Foundations of Program Verification. Wiley, 1984.

...zwei Überblicksklassiker:

- Krzysztof R. Apt. Ten Years of Hoare's Logic: A Survey – Part 1. ACM Transactions on Programming Languages and Systems 3(4):431-483, 1981.
- Krzysztof R. Apt. Ten Years of Hoare's Logic: A Survey – Part II: Nondeterminism. Theoretical Computer Science 28(1-2):83-109, 1984.

Literaturhinweise besonders für Kapitel 4 (2)

...ergänzend, weiterführend, vertiefend:

- Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. Verification of Sequential and Concurrent Programs. 3. Auflage, Springer-V., 2009.
- Krzysztof R. Apt, Ernst-Rüdiger Olderog. Programmverifikation – Sequentielle, parallele und verteilte Programme. Springer-V., 1994.
- Ernst-Rüdiger Olderog, Bernhard Steffen. Formale Semantik und Programmverifikation. In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 145-166, 2006.

Kapitel 1.10

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (1)

Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 2. Auflage, Springer-V., 2001. (Chapter 9.2, Semantics of Programming Languages)

Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis and Verification of Aerospace Software by Abstract Interpretation*. In Proceedings AIAA Infotech@Aerospace (AIAA I@A 2010), AIAA-2010-3385, American Institute of Aeronautics and Astronautics, 1-38, April 2010.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (2)

Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis by Abstract Interpretation of Embedded Critical Software*. ACM Software Engineering Notes 36(1):1-8, 2011.

Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler. *A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World*. Communications of the ACM 53(2):66-75, 2010.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (3)

Gilles Dowek. *Principles of Programming Languages*. Springer-V, 2009. (Chapter 1, Imperative Core; Chapter 1.1, Five Constructs)

Gerhard Goos, Wolf Zimmermann. *Programmiersprachen*. In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 515-562, 2006. (Kapitel 2.2, Elemente von Programmiersprachen: Syntax, Semantik und Pragmatik, Syntaktische Eigenschaften, Semantische Eigenschaften)

Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, 1992.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (4)


Steve P. Miller, Michael W. Whalen, Darren D. Cofer. *Software Model Checking Takes Off*. Communications of the ACM 53(2):58-64, 2010.


Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019.


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 1, Introduction)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 1, Introduction)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (5)

Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, Ciera Jaspan. *Lessons from Building Static Analysis Tools at Google*. Communications of the ACM 61(4):58-66, 2018.

Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993. (Chapter 3, Some principles of induction; Chapter 4, Inductive definitions)

Teil II

Semantik

Kapitel 2

Operationelle Semantik von WHILE

Operationelle Semantik von WHILE

...die Bedeutung eines programmiersprachlichen Konstrukts ist durch die Berechnung beschrieben, die es bei seiner Ausführung auf der Maschine induziert. Wichtig ist, **wie** *der Effekt der Berechnung erzeugt wird.*

Kapitel 2.1

Strukturell operationelle Semantik (SOS)

Strukturell operationelle Semantik

...beschreibt den Ablauf jedes einzelnen Berechnungsschritts, der stattfindet; daher auch die Bezeichnung Kleinschritt-Semantik.

WHILE : Strukturell operationelle Semantik

...das Funktional der strukturell operationellen Semantik (kurz: SO-Semantik) von WHILE :

$$[\![\, . \,]\!]_{sos} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation zu:

$$[\![\, \pi \,]\!]_{sos} : \Sigma \hookrightarrow \Sigma$$

die wir in der Folge definieren.

Die SO-Semantik von WHILE

...beschreibt den Berechnungsvorgang von WHILE -Programmen als Folge elementarer

- Speicherzustandsübergänge

in Form von

- Konfigurationen, Paaren von (Rest-) Programm und (Zwischen-) Zustand bzw. Endzuständen.

Konfigurationen

Definition 2.1.1 (Konfigurationen)

- Paare $\langle \pi, \sigma \rangle$ oder Zustände σ mit $\pi \in \mathbf{Prg}\ \mathrm{WHILE}$ -Programm und $\sigma \in \Sigma$ Zustand, heißen Konfiguration.
- $\Gamma =_{df} (\mathbf{Prg} \times \Sigma) \cup \Sigma$ bezeichnet die Menge aller Konfigurationen, γ eine einzelne Konfiguration.

Definition 2.1.2 (Nichtterminale, terminale Konfig.)

- Konfigurationen der Form $\langle \pi, \sigma \rangle$ heißen nichtterminal (oder: Zwischenkonfiguration):

  ...das (Rest-) Programm π ist auf den (Zwischen-) Zustand σ anzuwenden.

- Konfigurationen der Form σ heißen terminal (oder: final):

  ...Zustand σ ist der nach Ende einer (regulär terminierenden) Programmausführung erreichte Zustand.

Das SOS-Regelwerk für WHILE : Axiome (1)

$$[\text{skip}_{sos}] \quad \frac{}{\langle \mathit{skip}, \sigma \rangle \ggg \sigma}$$

$$[\text{ass}_{sos}] \quad \frac{}{\langle x{:=}t, \sigma \rangle \ggg \sigma[[\![t]\!]_A(\sigma)/x]}$$

$$[\text{if}^{tt}_{sos}] \quad \frac{}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \ggg \langle \pi_1, \sigma \rangle} \qquad [\![b]\!]_B(\sigma) = \textbf{wahr}$$

$$[\text{if}^{ff}_{sos}] \quad \frac{}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \ggg \langle \pi_2, \sigma \rangle} \qquad [\![b]\!]_B(\sigma) = \textbf{falsch}$$

$$[\text{while}_{sos}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \ggg \langle \text{if } b \text{ then } \pi;\ \text{while } b \text{ do } \pi \text{ od else } \mathit{skip} \text{ fi}, \sigma \rangle}$$

Das SOS-Regelwerk für WHILE : Regeln (2)

$$[\text{comp}^1_{sos}] \quad \frac{\langle \pi_1, \sigma \rangle \;\; \ggg \;\; \langle \pi_1', \sigma' \rangle}{\langle \pi_1;\pi_2, \sigma \rangle \;\; \ggg \;\; \langle \pi_1';\pi_2, \sigma' \rangle}$$

$$[\text{comp}^2_{sos}] \quad \frac{\langle \pi_1, \sigma \rangle \;\; \ggg \;\; \sigma'}{\langle \pi_1;\pi_2, \sigma \rangle \;\; \ggg \;\; \langle \pi_2, \sigma' \rangle}$$

Regelwerke: Axiome und Regeln

...wir unterscheiden:

- Prämissenlose Regeln, sog. Axiome, der Form:

$$[\mathit{Axiomsname}] \quad \frac{}{\text{Konklusion}} \quad [\mathit{Randbedingung}(\mathit{en})]$$

- Prämissenbehaftete Regeln, sog. (echte) Regeln, der Form:

$$[\mathit{Regelname}] \quad \frac{\text{Prämisse(n)}}{\text{Konklusion}} \quad [\mathit{Randbedingung}(\mathit{en})]$$

jeweils mit optionalen Randbedingungen (oder: Seitenbedingungen) wie z.B. im Axiom $[\mathrm{if}^{ff}_{sos}]$ in Form von

$$[\![\, b \,]\!]_B(\sigma) = \textbf{falsch}$$

Das SOS-Regelwerk von WHILE

...besteht aus:

- 5 Axiomen
  ...eins für die leere Anweisung, eins für die Zuweisung, zwei für die Fallunterscheidung, eins für die while-Schleife.
- 2 Regeln
  ...für die sequentielle Komposition.

Beachte: Mit Ausnahme der Axiome [skip_{sos}] und [ass_{sos}] sind alle Axiome und Regeln von der Form

$$\frac{\ldots}{\langle \pi, \sigma \rangle \Rightarrow\!\!> \langle \pi' \sigma' \rangle}$$

wobei die Konklusion einen SOS-Transitionsübergang in eine Zwischenkonfiguration zeigt, in der ein Restprogramm π' auf einen Zwischenzustand σ' anzuwenden bleibt, deshalb Kleinschrittsemantik.

Berechnungsschritt, Berechnungsfolge

Definition 2.1.3 (Berechnungsschritt)

Ein (SOS-) Berechnungsschritt ist von der Form

- $\langle \pi, \sigma \rangle \ggg \gamma \quad$ mit $\quad \gamma \in \Gamma =_{df} (\mathbf{Prg} \times \Sigma) \cup \Sigma$

Mit $\ggg^*$ bezeichnen wir die reflexiv-transitive Hülle von $\ggg$.

Definition 2.1.4 (Berechnungsfolge)

Eine (SOS-) Berechnungsfolge eines Programms $\pi \in \mathbf{Prg}$ angesetzt auf einen (Anfangs-) Zustand $\sigma \in \Sigma$ ist eine

- endliche Folge $\gamma_0, \ldots, \gamma_k$ von Konfigurationen mit:
  $\gamma_0 = \langle \pi, \sigma \rangle$ und $\gamma_i \ggg \gamma_{i+1}$ für alle $i \in \{0, \ldots, k-1\}$

oder eine

- unendliche Folge $\gamma_0, \gamma_1, \gamma_2, \ldots$ von Konfigurationen mit:
  $\gamma_0 = \langle \pi, \sigma \rangle$ und $\gamma_i \ggg \gamma_{i+1}$ für alle $i \in \mathbb{N}_0$.

Terminierung und Divergenz

Definition 2.1.5 (Terminierung und Divergenz)

Eine maximale (d.h. nicht mehr verlängerbare) Berechnungsfolge heißt

- regulär terminierend, wenn sie endlich ist und die letzte Konfiguration aus Σ ist.
- divergierend, wenn sie unendlich ist.
- irregulär terminierend sonst.

  (z.B. wegen Nichtauswertbarkeit der Bedingung einer Fallunterscheidung aufgrund einer Division durch $\mathbf{0}$:
  $\langle \text{if } a/0 = 42 \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle$ hat keine Folgekonfiguration: Weder $[if^{tt}_{sos}]$ noch $[if^{ff}_{sos}]$ ist anwendbar, da für beide Axiome die Auswertung der Randbedingung scheitert.)

Beispiel: Illustration der SO-Semantik (1)

Gegeben:

- Programm $\pi \in \mathbf{Prg}$:
  $\pi \equiv y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$
  (Bemerkung: $\equiv$ steht für 'syntaktisch ident')
- Anfangszustand $\sigma \in \Sigma$:

$$\sigma(z) = \begin{cases} \mathbf{3} & \text{falls } z = x \\ \mathbf{z} & \mathbf{z} \in \mathbb{Z} \text{ beliebig, falls } z \neq x \end{cases}$$

Gesucht: Die von der Anfangskonfiguration $\langle \pi, \sigma \rangle$ (lies: 'π angesetzt auf σ'):

$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma \rangle$$

ausgehende Berechnungsfolge.

Behauptung:

$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od},\ \sigma \rangle \Longrightarrow^* \sigma[\mathbf{6}/y][\mathbf{3}/x]$$

Beispiel: Illustration der SO-Semantik (2)

...die die Behauptung beweisende Berechungsfolge gemäß des SOS-Regelwerks:

$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma\rangle$$

$$\Rightarrow\!\!\!> \langle \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{1}/y]\rangle$$

$$\Rightarrow\!\!\!> \langle \text{if } x \neq 1 \text{ then } y := y * x;\ x := x - 1; \text{ while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od else } \textit{skip} \text{ fi}, \sigma[\mathbf{1}/y]\rangle$$

$$\Rightarrow\!\!\!> \langle y := y * x;\ x := x - 1; \text{ while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{1}/y]\rangle$$

$$\Rightarrow\!\!\!> \langle x := x - 1; \text{ while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, (\sigma[\mathbf{1}/y])[\mathbf{3}/y]\rangle$$

$$(\widehat{=} \quad \langle x := x - 1; \text{ while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{3}/y])\rangle\ )$$

Beispiel: Illustration der SO-Semantik (3)

$$
\begin{aligned}
\Rightarrow\!\!\!\!\Rightarrow\ & \langle \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, (\sigma[\mathbf{3}/y])[\mathbf{2}/x] \rangle \\
\Rightarrow\!\!\!\!\Rightarrow\ & \langle \text{if } x \neq 1 \\
& \quad \text{then } y := y * x;\ x := x - 1; \\
& \qquad \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od} \\
& \quad \text{else } \mathit{skip} \text{ fi}, (\sigma[\mathbf{3}/y])[\mathbf{2}/x] \rangle \\
\Rightarrow\!\!\!\!\Rightarrow\ & \langle y := y * x;\ x := x - 1; \\
& \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, (\sigma[\mathbf{3}/y])[\mathbf{2}/x] \rangle \\
\Rightarrow\!\!\!\!\Rightarrow\ & \langle x := x - 1; \\
& \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, (\sigma[\mathbf{6}/y])[\mathbf{2}/x] \rangle \\
\Rightarrow\!\!\!\!\Rightarrow\ & \langle \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, (\sigma[\mathbf{6}/y])[\mathbf{1}/x] \rangle
\end{aligned}
$$

Beispiel: Illustration der SO-Semantik (4)

$$
\begin{aligned}
\Rightarrow\!\!\!\!\gg \quad & \langle \text{if } x \neq 1 \\
& \quad \text{then } y := y * x;\ x := x - 1; \\
& \qquad \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od} \\
& \quad \text{else } \mathit{skip} \text{ fi}, (\sigma[\mathbf{6}/y])[\mathbf{1}/x] \rangle \\
\Rightarrow\!\!\!\!\gg \quad & \langle \mathit{skip}, (\sigma[\mathbf{6}/y])[\mathbf{1}/x] \rangle \\
\Rightarrow\!\!\!\!\gg \quad & (\sigma[\mathbf{6}/y])[\mathbf{1}/x] \\
& = \sigma[\mathbf{6}/y][\mathbf{1}/x]
\end{aligned}
$$

Korrektheitsargument voll aufgebrochen (1)

Der Berechnungsschritt:

$$[\text{while}_{sos}] \quad \Longrightarrow\!\!\!\!\Longrightarrow \quad \begin{array}{l} \langle \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{1}/y] \rangle \\ \langle \text{if } x \neq 1 \\ \quad \text{then } y := y * x;\ x := x - 1; \\ \quad\quad \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od} \\ \quad \text{else } \mathit{skip} \text{ fi}, \sigma[\mathbf{1}/y] \rangle \end{array}$$

... mit angegebener angewendeter Regel steht abkürzend und vereinfachend für den Ableitungbaum:

$$\frac{}{\langle \texttt{while } x \neq 1 \texttt{ do } y := y*x;\ x := x-1 \texttt{ od},\ \sigma\,[\mathbf{1}/y] \rangle \Longrightarrow\!\!\!\!\Longrightarrow} [\text{while}_{sos}]$$

$$\langle \texttt{if } x \neq 1 \texttt{ then } y := y*x;\ x := x-1; \texttt{ while } x <> 1 \texttt{ do } y := y*x;\ x := x-1 \texttt{ od else skip fi}, \sigma\,[\mathbf{1}/y] \rangle$$

Korrektheitsargument voll aufgebrochen (2)

Der Berechnungsschritt:

$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma\rangle$$

$[\text{ass}_{sos}]$,

$[\text{comp}^2_{sos}] \quad \ggg \langle \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{1}/y]\rangle$

...mit angegebenen angewendeten Regeln steht abkürzend und vereinfachend für den Ableitungbaum:

$$[\text{comp}^2_{sos}]\ \frac{[\text{ass}_{sos}]\ \dfrac{}{\langle y := 1,\ \sigma\rangle \ggg \sigma\,[\mathbf{1}/y]}}{\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\rangle \ggg \langle \text{while } x \neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\,[\mathbf{1}/y]\rangle}$$

Korrektheitsargument voll aufgebrochen (3)

Der Berechnungsschritt:

$$\langle (y := y * x;\ x := x - 1);$$
$$\text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{1}/y]\rangle$$

$[\text{ass}_{sos}]$,
$[\text{comp}^2_{sos}]$,
$[\text{comp}^1_{sos}]$ $\Longrightarrow\!\!\!>$

$$\langle x := x - 1;$$
$$\text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od},$$
$$(\sigma[\mathbf{1}/y])[\mathbf{3}/y]\rangle$$

...mit angegebenen angewendeten Regeln steht abkürzend und vereinfachend für den Ableitungbaum:

$$\dfrac{\dfrac{\dfrac{}{\langle y := y*x,\ \sigma[\mathbf{1}/y]\rangle \Longrightarrow\!\!\!> (\sigma[\mathbf{1}/y])[\mathbf{3}/y]\rangle}[\text{ass}_{sos}]}{\langle y := y*x;\ x := x-1,\ \sigma[\mathbf{1}/y]\rangle \Longrightarrow\!\!\!> \langle x := x-1,\ (\sigma[\mathbf{1}/y])[\mathbf{3}/y]\rangle}[\text{comp}^2_{sos}]}{\langle (y := y*x;\ x := x-1);\ \text{while } x \neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od},\ \sigma[\mathbf{1}/y]\rangle \Longrightarrow\!\!\!> \langle x := x-1;\ \text{while } x \neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od},\ (\sigma[\mathbf{1}/y])[\mathbf{3}/y]\rangle}[\text{comp}^1_{sos}]$$

SOS-Regeln: Determiniertheit, Determinismus

Lemma 2.1.6

$\forall\, \pi \in \mathbf{Prg}.\ \forall\, \sigma \in \Sigma.\ \forall\, \gamma, \gamma' \in \Gamma.$

$$\langle \pi, \sigma \rangle \Rrightarrow \gamma \ \wedge \langle \pi, \sigma \rangle \Rrightarrow \gamma' \ \Rightarrow \ \gamma = \gamma'$$

Korollar 2.1.7

Die vom SOS-Regelwerk für eine Konfiguration induzierte Berechnungsfolge ist eindeutig bestimmt, d.h. determiniert.

Salopper: Die SO-Semantik von WHILE ist deterministisch!

Das Semantikfunktional $[\![\ ]\!]_{sos}$

...dank Korollar 2.1.7 ist folgende Festlegung sinnvoll:

Definition 2.1.8 (SO-Semantik von WHILE)

Die strukturell operationelle Semantik (oder: SO-Semantik) von WHILE ist durch das in folgender Weise definierte Funktional:

$$[\![\ .\ ]\!]_{sos} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

gegeben:

$$\forall\, \pi \in \mathbf{Prg}.\ [\![\ \pi\ ]\!]_{sos} =_{df} \lambda\, \sigma. \begin{cases} \sigma' & \text{falls } \langle \pi, \sigma \rangle \Rrightarrow^* \sigma' \\ \mathit{undef} & \text{sonst} \end{cases}$$

(Anm.: $\Rrightarrow^*$ bezeichnet die reflexiv-transitive Hülle von $\Rrightarrow$)

Induktion über Längen von Berechnungsfolgen

...als Variante induktiver Beweisführung.

Induktion über die Länge von Berechnungsfolgen:

- Induktionsanfang
  - Beweise, dass Eigenschaft E für Berechnungsfolgen der Länge 0 gilt.
- Induktionsschritt
  - Beweise unter der Annahme, dass E für Berechnungsfolgen der Länge kleiner k gilt (Induktionshypothese!), dass E auch für Berechnungsfolgen der Länge k gilt.

Induktive Beweisführung

...über die Länge von Berechnungsfolgen ist typisch für den Nachweis von Aussagen oder Eigenschaften im Zusammenhang mit strukturell operationeller Semantik.

Ein typisches Beispiel ist der Beweis von:

Lemma 2.1.9

$$\forall \pi, \pi' \in \mathbf{Prg}.\ \forall \sigma, \sigma'' \in \Sigma.\ \forall k \in \mathbb{N}.\ (\langle \pi_1; \pi_2, \sigma \rangle \Longrightarrow^{k} \sigma'') \Rightarrow$$
$$\exists \sigma' \in \Sigma.\ \exists k_1, k_2 \in \mathbb{N}.\ (k_1 + k_2 = k \wedge \langle \pi_1, \sigma \rangle \Longrightarrow^{k_1} \sigma' \wedge \langle \pi_2, \sigma' \rangle \Longrightarrow^{k_2} \sigma'')$$

Übungsaufgabe 2.1.10

Beweise mithilfe einer Induktion über die Länge von Berechnungsfolgen die Aussage von Lemma 2.1.9.

Kapitel 2.2

Natürliche Semantik (NS)

Natürliche Semantik

...beschreibt, wie sich das Gesamtergebnis der Programmausführung ergibt; daher auch die Bezeichnung Großschritt-Semantik.

WHILE : Natürliche Semantik

...das Funktional der natürlichen Semantik (kurz: N-Semantik) von WHILE :

$$[\![.]\!]_{ns} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation zu:

$$[\![\pi]\!]_{ns} : \Sigma \hookrightarrow \Sigma$$

die wir in der Folge definieren.

Die N-Semantik von WHILE

...beschreibt den Berechnungsvorgang von WHILE -Programmen unmittelbar durch den Zusammenhang zwischen

- initialem (oder: Anfangszustand)
- finalem (oder: Endzustand)

Speicherzustand der Berechnung eines Programms.

Auch hier ist der bereits von der SO-Semantik bekannte Begriff der

- Konfigurationen (s. Definition 2.1.1)

zentral.

Das NS-Regelwerk für WHILE : Axiome (1)

$$[\text{skip}_{ns}] \quad \frac{}{\langle \mathit{skip}, \sigma \rangle \rightarrow \sigma}$$

$$[\text{ass}_{ns}] \quad \frac{}{\langle x{:=}t, \sigma \rangle \rightarrow \sigma[[\![t]\!]_A(\sigma)/x]}$$

$$[\text{while}^{ff}_{ns}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow \sigma} \qquad [\![b]\!]_B(\sigma) = \textbf{falsch}$$

Das NS-Regelwerk für WHILE : Regeln (2)

$$[\text{while}^{tt}_{ns}] \quad \frac{\langle \pi, \sigma \rangle \longrightarrow \sigma', \ \langle \text{while } b \text{ do } \pi \text{ od}, \sigma' \rangle \longrightarrow \sigma''}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \longrightarrow \sigma''} \quad [\![b]\!]_B(\sigma) = \textbf{wahr}$$

$$[\text{if}^{tt}_{ns}] \quad \frac{\langle \pi_1, \sigma \rangle \longrightarrow \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \longrightarrow \sigma'} \quad [\![b]\!]_B(\sigma) = \textbf{wahr}$$

$$[\text{if}^{ff}_{ns}] \quad \frac{\langle \pi_2, \sigma \rangle \longrightarrow \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \longrightarrow \sigma'} \quad [\![b]\!]_B(\sigma) = \textbf{falsch}$$

$$[\text{comp}_{ns}] \quad \frac{\langle \pi_1, \sigma \rangle \longrightarrow \sigma', \ \langle \pi_2, \sigma' \rangle \longrightarrow \sigma''}{\langle \pi_1 ; \pi_2, \sigma \rangle \longrightarrow \sigma''}$$

Das NS-Regelwerk von WHILE

...besteht aus:

- 3 Axiomen
  ...eins für die leere Anweisung, eins für die Zuweisung, eins für die while-Schleife.
- 4 Regeln
  ...eins für die while Schleife, zwei für die Fallunterscheidung, eins für die sequentielle Komposition.

Beachte: Alle Axiome und Regeln sind von der Form

$$\frac{\dots}{\langle \pi, \sigma \rangle \to \sigma'}$$

wobei die Konklusion einen NS-Transitionsübergang in einem Schritt in eine finale Konfiguration σ' zeigt, deshalb Großschrittsemantik.

Beispiel: Illustration der N-Semantik (1)

Gegeben:

- Programm $\pi \in \mathbf{Prg}$:
  $\pi \equiv y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$
  (Bemerkung: $\equiv$ steht für 'syntaktisch ident')
- Anfangszustand $\sigma \in \Sigma$:

$$\sigma(z) = \begin{cases} \mathbf{3} & \text{falls } z = x \\ \mathbf{z} & \mathbf{z} \in \mathbb{Z} \text{ beliebig, falls } y \neq x \end{cases}$$

Gesucht: Der von der Anfangskonfiguration $\langle \pi, \sigma \rangle$ (lies: 'π angesetzt auf σ'):

$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma \rangle$$

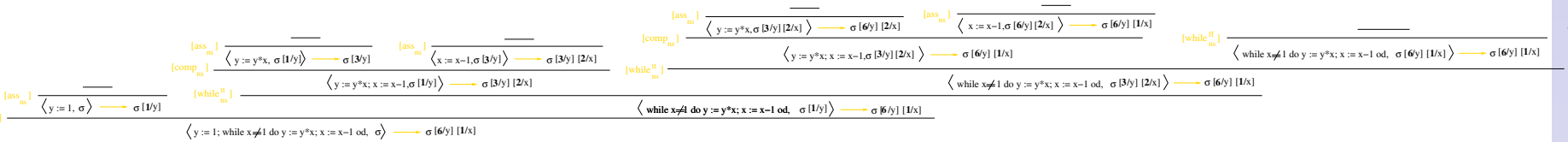
ausgehend erreichte finale Zustand.

Behauptung:

$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma \rangle \longrightarrow \sigma[\mathbf{6}/y][\mathbf{3}/x]$$

Beispiel: Illustration der N-Semantik (2)

...der die Behauptung beweisende Ableitungsbaum gemäß des NS-Regelwerks:



...der gleiche Ableitungsbaum in geringfügig lesefreundlicherer Form durch Herausziehen der benannten Teilbäume T1, T2:

$$[\text{while}^{\text{ff}}_{\text{ns}}]\ \frac{}{\langle \text{while } x\neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od},\ \ \sigma\,[6/y]\,[1/x]\rangle \longrightarrow \sigma\,[6/y]\,[1/x]} \equiv T1$$

$$[\text{while}^{\text{tt}}_{\text{ns}}]\ \frac{[\text{ass}_{\text{ns}}]\ \frac{}{\langle y := y*x, \sigma\,[3/y]\,[2/x]\rangle \longrightarrow \sigma\,[6/y]\,[2/x]}\quad [\text{ass}_{\text{ns}}]\ \frac{}{\langle x := x-1, \sigma\,[6/y]\,[2/x]\rangle \longrightarrow \sigma\,[6/y]\,[1/x]}}{[\text{comp}_{\text{ns}}]\ \frac{}{\langle y := y*x;\ x := x-1, \sigma\,[3/y]\,[2/x]\rangle \longrightarrow \sigma\,[6/y]\,[1/x]}\quad T1}$$

$$\frac{}{\langle \text{while } x\neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od},\ \ \sigma\,[3/y]\,[2/x]\rangle \longrightarrow \sigma\,[6/y]\,[1/x]} \equiv T2$$

$$[\text{comp}_{\text{ns}}]\ \frac{[\text{ass}_{\text{ns}}]\ \frac{}{\langle y := y*x, \sigma\,[1/y]\rangle \longrightarrow \sigma\,[3/y]}\quad [\text{ass}_{\text{ns}}]\ \frac{}{\langle x := x-1, \sigma\,[3/y]\rangle \longrightarrow \sigma\,[3/y]\,[2/x]}}{\langle y := y*x;\ x := x-1, \sigma\,[1/y]\rangle \longrightarrow \sigma\,[3/y]\,[2/x]}\quad T2$$

$$[\text{comp}_{\text{ns}}]\ \frac{[\text{ass}_{\text{ns}}]\ \frac{}{\langle y := 1, \sigma\rangle \longrightarrow \sigma\,[1/y]}\quad [\text{while}^{\text{tt}}_{\text{ns}}]\ \frac{}{\langle \text{while } x\neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od},\ \ \sigma\,[1/y]\rangle \longrightarrow \sigma\,[6/y]\,[1/x]}}{\langle y := 1;\ \text{while } x\neq 1 \text{ do } y := y*x;\ x := x-1 \text{ od},\ \ \sigma\rangle \longrightarrow \sigma\,[6/y]\,[1/x]}$$

NS-Regeln: Determiniertheit, Determinismus

Lemma 2.2.1

$$\forall\, \pi \in \mathbf{Prg}.\, \forall\, \sigma \in \Sigma.\, \forall\, \gamma, \gamma' \in \Gamma.\, \langle \pi, \sigma \rangle \to \gamma \wedge \langle \pi, \sigma \rangle \to \gamma' \Rightarrow \gamma = \gamma'$$

Korollar 2.2.2

Die vom NS-Regelwerk für eine Konfiguration induzierte finale Konfiguration ist (sofern definiert) eindeutig bestimmt, d.h. determiniert.

Salopper: Die N-Semantik von WHILE ist deterministisch!

Das Semantikfunktional $[\![\ ]\!]_{ns}$

...dank Korollar 2.2.2 ist folgende Festlegung sinnvoll:

Definition 2.2.3 (N-Semantik von WHILE)

Die natürliche Semantik (oder: N-Semantik) von WHILE ist durch das in folgender Weise definierte Funktional:

$$[\![\ .\ ]\!]_{ns} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

gegeben:

$$\forall\, \pi \in \mathbf{Prg}.\ [\![\ \pi\ ]\!]_{ns} =_{df} \lambda\, \sigma. \begin{cases} \sigma' & \text{falls } \langle \pi, \sigma \rangle \to \sigma' \\ \mathit{undef} & \text{sonst} \end{cases}$$

Induktion über Ableitungsbäume

....als Variante induktiver Beweisführung.

Induktion über den Aufbau von Ableitungsbäumen:

- Induktionsanfang
  - Beweise, dass Eigenschaft E für die Axiome des Regelwerks gilt (und somit für alle nichtzusammengesetzten Ableitungsbäume).
- Induktionsschritt
  - Beweise für jede echte Regel des Regelwerks unter der Annahme, dass E für jede Prämisse dieser Regel gilt (Induktionshypothese!), dass E auch für die Konklusion dieser Regel gilt, sofern die (optional vorhandenen) Randbedingungen der Regel erfüllt sind.

Induktive Beweisführung

...über den Aufbau von Ableitungsbäumen ist typisch für den Nachweis von Aussagen oder Eigenschaften im Zusammenhang mit natürlicher Semantik.

Ein typisches Beispiel ist der Beweis von Lemma 2.2.1, das wir nachstehend wiederholen:

Lemma 2.2.1

$$\forall\, \pi \in \mathbf{Prg}.\, \forall\, \sigma \in \Sigma.\, \forall\, \gamma, \gamma' \in \Gamma.\, \langle \pi, \sigma \rangle \to \gamma \wedge \langle \pi, \sigma \rangle \to \gamma' \Rightarrow \gamma = \gamma'$$

Übungsaufgabe 2.2.4

Beweise induktiv über den Aufbau von Ableitungsbäumen die Aussage von Lemma 2.2.1.

Kapitel 2.3

Äquivalenz von SO- und N-Semantik

Zusammenhang von $[\![\ ]\!]_{sos}$ und $[\![\ ]\!]_{ns}$

Lemma 2.3.1

$$\forall\, \pi \in \mathbf{Prg}.\ \forall\, \sigma, \sigma' \in \Sigma.\ \langle \pi, \sigma \rangle \to \sigma' \Rightarrow \langle \pi, \sigma \rangle \Longrightarrow^{*} \sigma'$$

Beweis durch strukturelle Induktion über den Aufbau des Ableitungsbaums für $\langle \pi, \sigma \rangle \to \sigma'$.

Lemma 2.3.2

$$\forall\, \pi \in \mathbf{Prg}.\ \forall\, \sigma, \sigma' \in \Sigma.\ \forall\, k \in \mathbb{N}.\ \langle \pi, \sigma \rangle \Longrightarrow^{k} \sigma' \Rightarrow \langle \pi, \sigma \rangle \to \sigma'$$

Beweis durch Induktion über die Länge der Berechnungsfolge $\langle \pi, \sigma \rangle \Longrightarrow^{k} \sigma'$, d.h. durch vollständige Induktion über k.

Äquivalenz von SO- und N-Semantik

...aus Lemma 2.3.1 und Lemma 2.3.2 folgt sofort:

Theorem 2.3.3 (Gleichheit von $[\![\pi]\!]_{sos}$ und $[\![\pi]\!]_{ns}$)

$$\forall\, \pi \in \mathbf{Prg}.\ [\![\pi]\!]_{sos} = [\![\pi]\!]_{ns}$$

...und somit Gleichheit der SO- und N-Semantikfunktionale:

Theorem 2.3.4 (Gleichheit von $[\![\ ]\!]_{sos}$ und $[\![\ ]\!]_{ns}$)

$$[\![\cdot]\!]_{sos} = [\![\cdot]\!]_{ns}$$

Kapitel 2.4

Vergleich von SO- und N-Semantik

Der Fokus strukturell operationeller Semantik

...liegt auf den

- individuellen Schritten einer Berechnungsfolge (der Ausführung von Zuweisungen und Tests) beschrieben durch Transitionen.

Intuitiv beschreibt eine Transition $\langle \pi, \sigma \rangle \Rrightarrow \gamma$

- den ersten Schritt der von π angesetzt auf σ ausgehenden Berechnungsfolge.

γ kann dabei von einer von zwei Formen sein:

- $\langle \pi', \sigma' \rangle$: Die Abarbeitung von π ist nicht vollständig; das Restprogramm π' ist auf σ' anzusetzen. Ist von $\langle \pi', \sigma' \rangle$ kein Transitionsübergang möglich (z.B. Division durch $\mathbf{0}$), so terminiert die Abarbeitung von π in $\langle \pi', \sigma' \rangle$ irregulär.
- σ': Die Abarbeitung von π ist vollständig; π angesetzt auf σ terminiert in einem Schritt in σ' regulär.

Der Fokus natürlicher Semantik

...liegt auf dem

- Zusammenhang von initialem und finalem Zustand einer Berechnungsfolge.

Intuitiv hat eine Transition $\langle \pi, \sigma \rangle \rightarrow \sigma'$ folgende Bedeutung:

- π angesetzt auf den initialen Zustand σ terminiert im finalen Zustand σ'.
- Existiert ein solches σ' nicht, so ist die N-Semantik für den initialen Zustand σ undefiniert.

Nichtdefiniertsein der N-Semantik für eine Anfangskonfiguration $\langle \pi, \sigma \rangle$ entspricht Nichtterminierung oder irregulärer Terminierung der SO-Semantik für $\langle \pi, \sigma \rangle$ und umgekehrt.

Grundlegende Arbeiten

...zur strukturell operationellen Semantik:

- Gordon D. Plotkin. A Structural Approach to Operational Semantics. Lecture notes, DAIMI FN-19, Aarhus University, Dänemark, 1981 (auch als Nachdruck von 1991).
  (Die 'Ursprungsreferenz' für strukturell operationelle Semantik)
- Gordon D. Plotkin. An Operational Semantics for CSP. In Proceedings of the TC-2 Working Conference on Formal Description of Programming Concepts II, Dines Bjørner (Hrsg.), North-Holland, Amsterdam, 199-226, 1982.

In konsolidierter Form:

- Gordon D. Plotkin. The Origins of Structural Operational Semantics. Journal of Logic and Algebraic Programming 60-61:3-15, 2004.
- Gordon D. Plotkin. A Structural Approach to Operational Semantics. Journal of Logic and Algebraic Programming 60-61:17-139, 2004. (i.w. Überarbeitung von DAIMI FN-19)

Grundlegende Arbeiten

...zur natürlichen Semantik:

- Dominique Clément, Joëlle Despeyroux, Thierry Despeyroux, Gilles Kahn. A Simple Applicative Language: Mini-ML. In Proceedings of the Int. ACM Conference on Lisp and Functional Programming (LFP'86), 13-27, 1986.

  This paper presents a formal description of the central part of the ML language in Natural Semantics...

Konsolidierter:

- Gilles Kahn. Natural Semantics. In Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS'87), Springer-V., LNCS 247, 22-39, 1987.

  ...many researchers have begun to present semantic specifications in a style [...] advocated by Plotkin. The purpose of this paper is to introduce in an intuitive manner the essential ideas of the method that we call now Natural Semantics...

Kapitel 2.5

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 2 (1)


Dominique Clément, Joëlle Despeyroux, Thierry Despeyroux, L. Hascoet, Gilles Kahn. *Natural Semantics on the Computer*. INRIA Research Report RR 416, INRIA, Sophia-Antipolis, June 1985.


Dominique Clément, Joëlle Despeyroux, Thierry Despeyroux, Gilles Kahn. *A Simple Applicative Language: Mini-ML*. In Proceedings of the International ACM Conference on Lisp and Functional Programming (LFP'86), 13-27, 1986.


Joëlle Despeyroux. *Proof of Translation in Natural Semantics*. In Proceedings of the 2nd International IEEE Symposium on Logic in Computer Science (LICS'86), 193-205, 1986.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 2 (2)


Matthew Hennessey. *The Semantics of Programming Languages: An Elementary Introduction using Structural Operational Semantics*. Wiley, 1991.


Gilles Kahn. *Natural Semantics*. In Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS'87), Springer-V., LNCS 247, 22-39, 1987.


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 2, Operational Semantics)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 2, Operational Semantics)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 2 (3)


Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Lecture notes, DAIMI FN-19, Aarhus University, Dänemark, 1981, (als Nachdruck von 1991).


Gordon D. Plotkin. *An Operational Semantics for CSP*. In Proceedings of the TC-2 Working Conference on Formal Description of Programming Concepts II, Dines Bjørner (Hrsg.), North-Holland, Amsterdam, 199-226, 1982.


Gordon D. Plotkin. *The Origins of Structural Operational Semantics*. Journal of Logic and Algebraic Programming 60-61:3-15, 2004.


Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Journal of Logic and Algebraic Programming 60-61:17-139, 2004.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 2 (4)

Thierry Despeyroux. *Typol: A Formalism to Implement Natural Semantics*. INRIA Research Report 94, Roquencourt, France, 1988.

Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993. (Chapter 2, Introduction to operational semantics)

Kapitel 3

Denotationelle Semantik von WHILE

Denotationelle Semantik

...die Bedeutung eines programmiersprachlichen Konstrukts wird durch mathematische Objekte, Abbildungen, modelliert, die den Effekt der Ausführung der Konstrukte beschreiben. Wichtig ist **einzig** *der Effekt, nicht wie er bewirkt wird.*

Vergleich operationelle, denotationelle Semantik

...der Fokus (strukturell) operationeller Semantik liegt darauf,

- wie ein Programm angesetzt auf einen (einzelnen) Zustand ausgeführt wird.
- Der Gesamteffekt des Programms, seine Bedeutung als Zustandstransformation (im Sinn einer Abbildung) bleibt im Dunkeln.

...der Fokus denotationeller Semantik liegt auf dem

- Gesamteffekt eines Programms, seiner Bedeutung als Zustandstransformation (im Sinn einer Abbildung).
- Wie das Programm angesetzt auf einen Zustand diese Abbildung erreicht, bleibt im Dunkeln.

Beispiel

...ist π ein Programm zur Berechnung der Fakultätsfunktion, so macht es eine

- (strukturell) operationelle Semantik für π
  - einfach zu erkennen, wie das Programm angesetzt auf einen Zustand das Resultat berechnet.
  - schwierig zu erkennen, welche Bedeutung einzelne Programmteile haben und dass insgesamt die Fakultätsfunktion berechnet wird.
- denotationelle Semantik für π
  - einfach zu erkennen, welche Bedeutung einzelne Programmteile haben und dass insgesamt die Fakultätsfunktion berechnet wird.
  - schwierig zu erkennen, wie dies für einen Zustand tatsächlich erreicht wird.

Der denotationelle Semantikdefinitionsstil

...erreicht das, indem für jedes syntaktische Konstrukt eine semantische Funktion festgelegt wird, die dem syntaktischen Konstrukt ein mathematisches Objekt, eine Abbildung als Bedeutung zuweist; eine Abbildung, die den Effekt der Ausführung des Konstrukts beschreibt (nicht jedoch, wie dieser Effekt erreicht wird oder erreicht werden kann).

Zusammengefasst: Für jedes

- elementare syntaktische Konstrukt gibt es eine semantische Funktion, eine Zustandstransformation, die seinen Effekt, seine Bedeutung beschreibt.
- zusammengesetzte syntaktische Konstrukt gibt es eine semantische Funktion, die kompositionell über die semantischen Funktionen seiner Komponenten definiert ist, und seinen Effekt, seine Bedeutung beschreibt.
- Zentral: Kompositionalität der semantischen Funktionen.

Kapitel 3.1

Denotationelle Semantik (DS)

WHILE : Denotationelle Semantik

...das Funktional der denotationellen Semantik (kurz: D-Semantik) von WHILE :

$$[\![\, . \,]\!]_{ds} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation zu:

$$[\![\, \pi \,]\!]_{ds} : \Sigma \hookrightarrow \Sigma$$

die wir in der Folge definieren.

Die D-Semantik von While

...beschreibt die Bedeutung von While -Programmen unmittelbar in Form einer Zustrandstransformation(sfunktion), die sich

- kompositionell (!) aus den Bedeutungsfunktionen (d.h. den Zustandstransformationsfunktionen) der Programmteile ergibt.

Konfigurationen wie bei der SO- und N-Semantik spielen keine Rolle, ebensowenig der konkrete Ausführungsprozess.

Das D-Regelwerk v. WHILE: Defin. Gleichungen

$$[\![\mathit{skip}]\!]_{ds} = \mathit{id} \qquad \text{(Identität)}$$

$$[\![x := t]\!]_{ds} = \lambda\sigma.\ \sigma[[\![t]\!]_A(\sigma)/x] \qquad \text{(Zustandssubstitution)}$$

$$[\![\pi_1;\ \pi_2]\!]_{ds} = [\![\pi_2]\!]_{ds} \circ [\![\pi_1]\!]_{ds} \qquad \text{(Kompositionalität!)}$$

$$[\![\text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}]\!]_{ds} = \mathit{cond}\,([\![b]\!]_B, [\![\pi_1]\!]_{ds}, [\![\pi_2]\!]_{ds})$$

$$[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} = \mathit{FIX}\ F$$

$$\text{mit } F\ g = \mathit{cond}\,([\![b]\!]_B, g \circ [\![\pi]\!]_{ds}, \mathit{id})$$

mit:

- $\mathit{id} = \lambda\sigma.\ \sigma$: Identische Zustandstransformation.
- cond: Fallunterscheidungsfunktional.
- F: Zustandstransformationsfunktional.
- FIX: Fixpunktzustandstransformationsfunktional
  (kurz: Fixpunktfunktional)

Die Funktionale *cond*, F und *FIX* im Detail

- $cond : (\Sigma \hookrightarrow \mathbb{B}) \times (\Sigma \hookrightarrow \Sigma) \times (\Sigma \hookrightarrow \Sigma) \to (\Sigma \hookrightarrow \Sigma)$

  ...Fallunterscheidungsfunktional definiert durch:

$$cond\,(p, g_1, g_2) =_{df} \lambda\sigma. \begin{cases} g_1(\sigma) & \text{falls } p(\sigma) = \textbf{wahr} \\ g_2(\sigma) & \text{falls } p(\sigma) = \textbf{falsch} \\ undef & \text{sonst} \end{cases}$$

- $F : (\Sigma \hookrightarrow \Sigma) \to (\Sigma \hookrightarrow \Sigma)$

  ...Zustandstransformationsfunktional; einige Zustandstransf. werden von F ident abgebildet, sog. Fixpunkte.

- $FIX : ((\Sigma \hookrightarrow \Sigma) \to (\Sigma \hookrightarrow \Sigma)) \to (\Sigma \hookrightarrow \Sigma)$

  ...Fixpunkzustandstransformationsfunktional, das den kleinsten Fixpunkt des Argumentfunktionals liefert.

  Intuitiv: Sind $f : (\Sigma \hookrightarrow \Sigma) \to (\Sigma \hookrightarrow \Sigma)$ und $g : (\Sigma \hookrightarrow \Sigma)$ zwei Funktionen, so gilt:

$$FIX\ f = g \iff \underbrace{f\ g = g}_{g \text{ Fixpunkt von } f} \quad \wedge \quad \underbrace{\forall g'.\ f\ g' = g' \Rightarrow g \sqsubseteq g'}_{g \text{ kleinster Fixpunkt von } f}$$

Die D-Semantik der Fallunterscheidung

...ist im D-Regelwerk festgelegt durch:

$$[\![\text{ if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi }]\!]_{ds} = \mathit{cond}([\![b]\!]_B, [\![\pi_1]\!]_{ds}, [\![\pi_2]\!]_{ds})$$

Expandieren wir *cond* in dieser Gleichung erhalten wir:

$$[\![\text{ if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi }]\!]_{ds}$$

$$= \lambda\sigma. \begin{cases} [\![\pi_1]\!]_{ds}(\sigma) & \text{falls } ([\![b]\!]_B(\sigma) = \textbf{wahr} \\ [\![\pi_2]\!]_{ds}(\sigma) & \text{falls } ([\![b]\!]_B(\sigma) = \textbf{falsch} \\ \mathit{undef} & \text{falls } [\![b]\!]_B(\sigma) = \mathit{undef} \end{cases}$$

was unserer Erwartung an die Bedeutung der Fallunterscheidung entspricht.

Erinnerung: $[\![.]\!]_B$ und $[\![.]\!]_A$ sind partiell definiert (z.B. Division durch $\mathbf{0}$, vgl. Kapitel 1.4 und 1.5).

Die D-Semantik der while-Schleife

...ist im D-Regelwerk festgelegt durch:

$$[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} = FIX\ F$$

wobei das Argumentfunktional:

$$F : (\Sigma \hookrightarrow \Sigma) \to (\Sigma \hookrightarrow \Sigma)$$

des Fixpunktfunktionals definiert ist durch:

$$\begin{aligned} F\ g &= cond\,([\![b]\!]_B, g \circ [\![\pi]\!]_{ds}, id) \\ &= \lambda\sigma. \begin{cases} (g \circ [\![\pi]\!]_{ds})(\sigma) & \text{falls } [\![b]\!]_B(\sigma) = \textbf{wahr} \\ \sigma & \text{falls } [\![b]\!]_B(\sigma) = \textbf{falsch} \\ undef & \text{falls } [\![b]\!]_B(\sigma) = undef \end{cases} \end{aligned}$$

was auch unserer Erwartung an die Bedeutung der while-Schleife entspricht.

Veranschaulichung (1)

...wir zeigen, dass bei 'vernünftiger' Festlegung der D-Semantik die Bedeutung der while-Schleife ein Fixpunkt des Funktionals F sein muss.

Wir haben die Erwartung, dass bei 'vernünftiger' Festlegung der D-Semantik von Fallunterscheidung und while-Schleife gilt:

Die D-Semantiken der Anweisungen while b do π od und if b then $(\pi;$ while b do π od$)$ else *skip* fi stimmen überein:

A) $[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} =$

$$[\![\text{if } b \text{ then } (\pi; \text{while } b \text{ do } \pi \text{ od}) \text{ else } skip \text{ fi}]\!]_{ds}$$

Gleichheit A) liefert zusammen mit den D-Semantiken von Fallunterscheidung, sequentieller Komposition und *skip*-Anweisung Gleichheit B):

B) $[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} =$

$$cond\,([\![b]\!]_B, [\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} \circ [\![\pi]\!]_{ds}, id)$$

Veranschaulichung (2)

Die Definition v. Funktional F liefert zusätzlich Gleichheit C):

C) $F\ [\![\ \text{while } b \text{ do } \pi \text{ od}\ ]\!]_{ds} =_{df}$

$$cond\ ([\![\ b\ ]\!]_B, [\![\ \text{while } b \text{ do } \pi \text{ od}\ ]\!]_{ds} \circ [\![\ \pi\ ]\!]_{ds}, id)$$

Aus B) und C) folgt Gleichheit D):

D) $[\![\ \text{while } b \text{ do } \pi \text{ od}\ ]\!]_{ds} = F\ [\![\ \text{while } b \text{ do } \pi \text{ od}\ ]\!]_{ds}$

...was wie angekündigt bedeutet: Die D-Semantik der while-Schleife ist Fixpunkt, *ein* Fixpunkt des Funktionals F (i.a. gibt es mehr als einen Fixpunkt).

Das D-Regelwerk legt die D-Semantik der while-Schleife genauer als den eindeutig bestimmten kleinsten Fixpunkt von F fest (s. Kapitel 3.2):

E) $[\![\ \text{while } b \text{ do } \pi \text{ od}\ ]\!]_{ds} =_{df} FIX\ F$

D-Regeln: Determiniertheit, Determinismus

...unter Vorwegnahme der Ergebnisse aus Kapitel 3.2 zur Wohldefiniertheit von *FIX F* gilt, dass die Festlegungen des D-Regelwerks 'vernünftig' sind:

Theorem 3.1.1 (Determiniertheit, Determinismus)

Das D-Regelwerk von WHILE legt für jedes WHILE -Programm $\pi \in \mathbf{Prg}$ in eindeutiger Weise eine partielle Zustandstransformation

$$z_\pi : \Sigma \hookrightarrow \Sigma$$

(als Bedeutung) fest.

Das Semantikfunktional $[\![\ ]\!]_{ds}$

...dank Theorem 3.1.1 ist folgende Festlegung sinnvoll:

Definition 3.1.2 (D-Semantik von WHILE)

Die denotationelle Semantik (oder: D-Semantik) von WHILE ist gegeben durch das Funktional

$$[\![\ .\ ]\!]_{ds} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

wobei für alle $\pi \in \mathbf{Prg}$

$$[\![\ \pi\ ]\!]_{ds} : \Sigma \hookrightarrow \Sigma$$

die gemäß Theorem 3.1.1 gegebene eindeutig bestimmte partielle Zustandstransformation z_π ist:

$$[\![\ \pi\ ]\!]_{ds} = z_\pi$$

Kapitel 3.2

Wohldefiniertheit des Fixpunktfunktionals

Vorgehens- und Beweisskizze

Wir zeigen:

1. Die Menge $\mathcal{Z}$ der partiellen Zustandstransformationen

$$\mathcal{Z} =_{df} [\Sigma \hookrightarrow \Sigma]$$

   ist durch die Ordnung $\sqsubseteq_{\mathcal{Z}}$ auf $\mathcal{Z}$ vollständig partiell geordnet, d.h. $\sqsubseteq_{\mathcal{Z}} \subseteq \mathcal{Z} \times \mathcal{Z}$, $(\mathcal{Z}, \sqsubseteq_{\mathcal{Z}})$ ist vollständige partielle Ordnung (VPO) (Definition 3.2.3, Lemma 3.2.13).
2. Das Funktional $F : \mathcal{Z} \to \mathcal{Z}$ auf $\mathcal{Z}$ mit $F\ g = \mathit{cond}\,(\llbracket\, b \,\rrbracket_B, g \circ \llbracket\, \pi \,\rrbracket_{ds}, \mathit{id})$ ist stetig (Lemma 3.2.14, Lemma 3.2.15).
3. Stetige Funktionen auf VPOs besitzen nach Fixpunktsatz 3.2.12 von Knaster, Tarski und Kleene einen eindeutig bestimmten kleinsten Fixpunkt als kleinste obere Schranke der von $\bot$ ausgehenden Kleene-Kette.

Aus 1.), 2.) und 3.) folgt die Wohldefiniertheit von $\mathit{FIX}\ F$ im Anwendungsfall; damit Theorem 3.1.1 u. die Wohldefiniertheit des denot. Semantikfunktionals: $\llbracket\ \rrbracket_{ds} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$.

Der Graph von Funktionen

..bezeichne $\mathcal{F}$ die Menge aller (partiellen und totalen) Funktionen von M nach N.

Definition 3.2.1 (Graph einer Funktion)

Der Graph einer

1. totalen Funktion $f \in \mathcal{F}$ ist die Relation *graph* auf $M \times N$ definiert durch:
$$graph(f) =_{df} \{\langle m, n\rangle \mid f(m) = n\} \subseteq M \times N$$
2. partiellen Funktion $f \in \mathcal{F}$ mit Definitionsbereich $M_f \subseteq M$ ist die Relation *graph* auf $M_f \times N$ definiert durch:
$$graph(f) =_{df} \{\langle m, n\rangle \mid m \in M_f \wedge f(m) = n\} \subseteq M_f \times N$$

Vereinbarung: Für $f \in \mathcal{F}$, $M_f \subseteq M$, schreiben wir:

$$f(m) = \begin{cases} n & \text{falls } \langle m, n\rangle \in graph(f) \\ undef & \text{falls } m \notin M_f \end{cases}$$

Eigenschaften des Graphen einer Funktion

Lemma 3.2.2 (Funktionsgrapheigenschaften)

Der Graph einer

1. totalen Funktion $f \in \mathcal{F}$ ist:
   - 1.1 rechtseindeutig, d.h. $\forall\, m \in M.\ \forall\, n, n' \in N.\ \langle m, n\rangle \in \mathit{graph}(f) \ \wedge\ \langle m, n'\rangle \in \mathit{graph}(f) \Rightarrow n = n'$.
   - 1.2 linkstotal, d.h. $\forall\, m \in M.\ \exists\, n \in N.\ \langle m, n\rangle \in \mathit{graph}(f)$.
2. (echt) partiellen Funktion $f : M \hookrightarrow N$ mit $M_f \subset M$ ist rechtseindeutig, aber nicht linkstotal.

Bemerkung: Funktionen $f \in \mathcal{F}$ können mit ihrem Graphen $\mathit{graph}(f)$ identifiziert werden und umgekehrt.

Partielle Ordnung auf $\mathcal{F}$

Definition 3.2.3 (Ordnung auf $\mathcal{F}$)

$$\forall f, g \in \mathcal{F}.\ f \sqsubseteq_{\mathcal{F}} g \iff_{df} \mathit{graph}(f) \subseteq \mathit{graph}(g)$$

Lemma 3.2.4

$$\forall f, g \in \mathcal{F}.\ f \sqsubseteq_{\mathcal{F}} g \iff$$
$$\forall m \in \mathrm{M}.\ f(m)\ \mathit{definiert} = n \Rightarrow g(m)\ \mathit{definiert} = n$$

Lemma 3.2.5 (Part. Ordnung m. kleinstem Element)

Das Paar $(\mathcal{F}, \sqsubseteq_{\mathcal{F}})$ ist eine partielle Ordnung mit kleinstem Element $\bot_{\mathcal{F}} \in \mathcal{F}$, $\bot_{\mathcal{F}} =_{df} \lambda m.\ \mathit{undef}$, der total undefinierten Funktion.

Ketten und Schranken in $\mathcal{F}$

Definition 3.2.6 (Kette in $\mathcal{F}$)

Sei $F =_{df} \{f_1, f_2, f_3, \ldots\} \subseteq \mathcal{F}$. $\mathcal{F}$ heißt Kette (in $\mathcal{F}$), wenn F bezüglich $\sqsubseteq_{\mathcal{F}}$ total geordnet ist, d.h. die Elemente von F sich anordnen lassen in der Form:

$$f_1 \sqsubseteq_{\mathcal{F}} f_2 \sqsubseteq_{\mathcal{F}} f_3 \sqsubseteq_{\mathcal{F}} \ldots$$

Definition 3.2.7 ((Kleinste) obere Schranke in $\mathcal{F}$)

Sei $F \subseteq \mathcal{F}$, $g \in \mathcal{F}$. g heißt

1. obere Schranke von F, wenn gilt: $\forall f \in F.\ f \sqsubseteq_{\mathcal{F}} g$.
2. kleinste obere Schranke von F, wenn gilt:
   - 2.1 g ist obere Schranke von F.
   - 2.2 $\forall f \in \mathcal{F}.\ f$ obere Schranke von $F \Rightarrow g \sqsubseteq_{\mathcal{F}} f$.

Schranken von Ketten (partieller) Funktionen

Lemma 3.2.8

Sei $F \subseteq \mathcal{F}$ eine Kette (partieller oder totaler) Funktionen. Dann gilt: Die kleinste obere Schranke von F, in Zeichen: $\bigsqcup F$, existiert und ist gegeben durch:

$$\mathit{graph}(\bigsqcup F) \text{ existiert } = \bigcup \{\mathit{graph}(f) \mid g \in F\}$$

Identifizieren wir $\mathit{graph}(\bigsqcup F)$ mit der Funktion $\bigsqcup F$ selbst, erhalten wir:

$$\forall\, m \in M.\ (\bigsqcup F)(m) = n \iff \exists\, f \in F.\ f(m) = n$$

Vollständige partielle Ordnungen

Definition 3.2.9 (Vollständige partielle Ordnung)

Eine partielle Ordnung $(P, \sqsubseteq)$ heißt vollständige partielle Ordnung (VPO), falls jede (aufsteigende) Kette $K \subseteq P$ eine kleinste obere Schranke $\bigsqcup K$ in P besitzt, d.h. $\bigsqcup K$ *existiert* $\in P$.

Lemma 3.2.10 (VPO $(\mathcal{F}, \sqsubseteq_{\mathcal{F}})$)

$(\mathcal{F}, \sqsubseteq_{\mathcal{F}})$ ist eine vollständige partielle Ordnung (VPO) (mit kleinstem Element $\bot_{\mathcal{F}}$).

Monotonie und Stetigkeit

...von Funktionen auf vollständigen partiellen Ordnungen.

Definition 3.2.11 (Monotonie, Stetigkeit)

Seien $(C, \sqsubseteq_C)$ und $(D, \sqsubseteq_D)$ zwei VPOs und f eine Funktion von C nach D. Dann heißt f

1. monoton gdw. $\forall\, c, c' \in C.\ c \sqsubseteq_C c' \Rightarrow f(c) \sqsubseteq_D f(c')$
   (Erhalt der Ordnung der Elemente)
2. stetig gdw. (i) f monoton
   (ii) $\forall\, C' \subseteq C.\ f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$
   (Erhalt der kleinsten oberen Schranken)

Existenz kleinster Fixpunkte stetiger Funktionen

Fixpunkttheorem 3.2.12 (Knaster, Tarski, Kleene)

Sei $(C, \sqsubseteq)$ eine VPO und $f \in [C \overset{stet}{\to} C]$ eine stetige Funktion auf C. Dann gilt:

f hat einen eindeutig bestimmten kleinsten Fixpunkt $\mu f \in C$, der durch das Supremum der (sog.) Kleene-Kette $\{\bot, f(\bot), f^2(\bot), f^3(\bot), \ldots\}$ gegeben ist, d.h.:

$$\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) = \bigsqcup \{\bot, f(\bot), f^2(\bot), \ldots\}$$

Erinnerung: $f^0 =_{df} Id_C$; $f^i =_{df} f \circ f^{i-1}$, $i > 0$.

Zustandstransformationen als VPO

...bezeichne $\mathcal{Z} =_{df} [\Sigma \hookrightarrow \Sigma]$ die Menge partieller Zustandstransformationen.

Die Ordnung $\sqsubseteq_{\mathcal{F}}$ auf $\mathcal{F}$ überträgt sich für $M = N$ und $\Sigma = M$ unmittelbar auf $\mathcal{Z}$:

$$\forall z_1, z_2 \in \mathcal{Z}.\ z_1 \sqsubseteq_{\mathcal{Z}} z_2 \iff_{df} z_1 \sqsubseteq_{\mathcal{F}} z_2 \iff$$
$$\forall \sigma \in \Sigma.\ z_1(\sigma)\ \mathit{definiert} = \sigma' \Rightarrow z_2(\sigma)\ \mathit{definiert} = \sigma'$$

Lemma 3.2.13 (VPO $(\mathcal{Z}, \sqsubseteq_{\mathcal{Z}})$)

$(\mathcal{Z}, \sqsubseteq_{\mathcal{Z}})$ ist eine vollständige partielle Ordnung (VPO) (mit kleinstem Element $\bot_{\mathcal{Z}} \in \mathcal{Z}$, $\bot_{\mathcal{Z}} =_{df} \lambda\sigma.\ \mathit{undef}$, der total undefinierten Zustandstransformation).

Stetigkeitsresultate für Zustandstransformat.

Lemma 3.2.14

Sei $p \in [\Sigma \hookrightarrow \mathbb{B}]$, $g, g_0 \in [\Sigma \hookrightarrow \Sigma]$ und F definiert durch:

$$F\ g = \mathit{cond}\ (p, g, g_0)$$

Dann gilt: F ist stetig.

Lemma 3.2.15

Sei $g, g_0 \in [\Sigma \hookrightarrow \Sigma]$ und F definiert durch:

$$F\ g = g \circ g_0$$

Dann gilt: F ist stetig.

Zusammen

...mit:

Lemma 3.2.16

Das D-Regelwerk von WHILE definiert eine totale Funktion:

$$[\![\ ]\!]_{ds} \in [\mathbf{Prg} \to \mathcal{Z}]$$

...sind wir durch. Wir erhalten:

Theorem 3.2.17 (Wohldefiniertheit von $[\![\ ]\!]_{ds}$)

$[\![\ ]\!]_{ds} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$ ist wohldefiniert.

Grundlegende Darstellung

...der Theorie denotationeller Semantik:

- Joseph E. Stoy. Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press, 1981.

Kapitel 3.3

Äquivalenz denotationeller und operationeller Semantik

Äquivalenz von $[\![\ ]\!]_{ds}$, $[\![\ ]\!]_{sos}$: (i) $[\![\ ]\!]_{ds} \sqsubseteq [\![\ ]\!]_{sos}$

Lemma 3.3.1

$$\forall\, \pi \ \in\ \mathbf{Prg}.\ [\![\, \pi \,]\!]_{ds} \ \sqsubseteq\ [\![\, \pi \,]\!]_{sos}$$

Beweis durch strukturelle Induktion über den induktiven Aufbau von π.

Äquivalenz von $[\![\ ]\!]_{ds}$, $[\![\ ]\!]_{sos}$: (ii) $[\![\ ]\!]_{sos} \sqsubseteq [\![\ ]\!]_{ds}$

Lemma 3.3.2

$$\forall\, \pi \ \in\ \mathbf{Prg}.\ [\![\, \pi \,]\!]_{sos} \ \sqsubseteq\ [\![\, \pi \,]\!]_{ds}$$

Beweis von

$$\forall\, \pi \in \mathbf{Prg}.\ \forall\, \sigma, \sigma' \in \Sigma.\ \langle \pi, \sigma \rangle \ \Longrightarrow^{*}\ \sigma' \ \Rightarrow\ [\![\, \pi \,]\!]_{ds}(\sigma) = \sigma'$$

durch Induktion über die Länge k der Berechnungsfolge $\langle \pi, \sigma \rangle \ \Longrightarrow^{k}\ \langle \pi', \sigma' \rangle$ unter Benutzung von 1.) und 2.), dass:

$\forall\, \pi \in \mathbf{Prg}.\ \forall\, \sigma, \sigma' \in \Sigma.$

1. $\langle \pi, \sigma \rangle \ \Longrightarrow\ \sigma' \quad \Rightarrow \quad [\![\, \pi \,]\!]_{ds}(\sigma) = \sigma'$
2. $\langle \pi, \sigma \rangle \ \Longrightarrow\ \langle \pi', \sigma' \rangle \quad \Rightarrow \quad [\![\, \pi \,]\!]_{ds}(\sigma) = [\![\, \pi' \,]\!]_{ds}(\sigma')$

...die durch Induktion über den Aufbau des Ableitungsbaums für $\langle \pi, \sigma \rangle \ \Longrightarrow\ \sigma'$ bzw. $\langle \pi, \sigma \rangle \ \Longrightarrow\ \langle \pi', \sigma' \rangle$ gezeigt werden.

Äquivalenz von $[\![\ ]\!]_{ds}$, $[\![\ ]\!]_{sos}$, $[\![\ ]\!]_{ns}$

Lemma 3.3.1 und Lemma 3.3.2 liefern:

Theorem 3.3.3 (Äquivalenz von $[\![\ ]\!]_{ds}$ und $[\![\ ]\!]_{sos}$)

$$\forall\, \pi \ \in\ \mathbf{Prg}.\ [\![\, \pi \,]\!]_{ds} = [\![\, \pi \,]\!]_{sos}$$

Aus Theorem 3.3.3 und Theorem 2.3.3 folgt weitergehend:

Theorem 3.3.4 (Äquivalenz von $[\![\ ]\!]_{ds}$, $[\![\ ]\!]_{sos}$, $[\![\ ]\!]_{ns}$)

$$\forall\, \pi \ \in\ \mathbf{Prg}.\ [\![\, \pi \,]\!]_{ds} = [\![\, \pi \,]\!]_{sos} = [\![\, \pi \,]\!]_{ns}$$

Kapitel 3.4

Eindeutigkeit der Semantik von WHILE

Eindeutigkeit der Semantik von WHILE

...die Äquivalenz der strukturell operationellen, natürlichen und denotationellen Semantik von WHILE erlaubt, den semantik-angebenden Index in der Folge fortzulassen und vereinfachend von $[\![\ ]\!]_{\text{WHILE}}$ als **der** Semantik von WHILE zu sprechen:

$$[\![\ ]\!]_{\text{WHILE}} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$$

definiert (z.B.) für den

- Sprachentwickler in Form von: $[\![\ ]\!]_{\text{WHILE}} =_{df} [\![\ ]\!]_{ds}$
- Sprachimplementierer in Fom von: $[\![\ ]\!]_{\text{WHILE}} =_{df} [\![\ ]\!]_{sos}$

Da wir keine andere Sprache als WHILE betrachten, können wir statt $[\![\ ]\!]_{\text{WHILE}}$ noch einfacher $[\![\ ]\!]$ schreiben.

Kapitel 3.5

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 3 (1)

Michael J.C. Gordon. *The Denotational Description of Programming Languages*. Springer-V., 1979.

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 4, Denotational Semantics)

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 5, Denotational Semantics; Chapter 6, More on Denotational Semantics)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 3 (2)

- David A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn & Bacon, 1986.
- Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1981.
- Robert D. Tennent. *Semantics of Programming Languages*. Prentice Hall, 1991.
- Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993. (Chapter 5, The denotational semantics of IMP; Chapter 8, Introduction to domain theory)

Teil III

Verifikation

Kapitel 4

Axiomatische Semantik von WHILE

Axiomatische Semantik von WHILE

...bestimmte Eigenschaften des Effekts der Ausführung eines Konstrukts werden als Zusicherungen ausgedrückt; nicht dafür relevante andere Aspekte der Ausführung werden ignoriert. Die Korrektheit oder Gültigkeit der Zusicherungen wird bewiesen; man spricht von Programmverifikation.

Kapitel 4.1

Korrektheitsbegriffe, Programmverifikation

Programmverifikation

...beschäftigt sich damit zu beweisen, dass ein Programm bestimmte

- Eigenschaften

erfüllt (oder besitzt oder für diese Eigenschaften korrekt ist).

Dabei lässt sich zwischen

- partiellen
- totalen

Korrektheitseigenschaften unterscheiden.

Partielle, totale Korrektheitseigenschaften

...für ein Programm π.

Partielle Korrektheitseigenschaften von π garantieren:

- Wird π angesetzt auf einen Zustand σ und terminiert π regulär in einem Zustand σ', dann stehen die Werte der Variablen von σ und σ' in einer bestimmten Beziehung E zueinander; π heißt dann partiell korrekt für E.

Totale Korrektheitseigenschaften von π garantieren:

- Wird π angesetzt auf einen Zustand σ, dann terminiert π regulär in einem Zustand σ' und die Werte der Variablen von σ und σ' stehen in einer bestimmten Beziehung E zueinander. π heißt dann total korrekt für E.

Informell: Totale Korrektheit "gleich"
Partielle Korrektheit "plus" Reguläre Termination

Beispiel: Programm und Programmeigenschaft

Sei π das (kanonische) Fakultätsprogramm:

$$\pi \equiv x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

Betrachte Eigenschaft E:

$$E : \Sigma \times \Sigma \to \mathbb{B}$$

$$\forall\, \sigma, \sigma' \in \Sigma.\ E(\sigma, \sigma') \iff_{df} \sigma'(y) = \sigma(a)!$$

wobei $! : \mathbb{N}_0 \to \mathbb{N}_1$ die Fakultätsfunktion bezeichnet:

$$\forall\, n \in \mathbb{N}_0.\ n! =_{df} \begin{cases} 1 & \text{falls } n = 0 \\ n * (n-1)! & \text{sonst} \end{cases}$$

Partielle, totale Korrektheit von π bzgl. E

Lemma 4.1.1 (Partielle Korrektheit von π bzgl. E)

π ist partiell korrekt für E, d.h. wird π angesetzt auf einen Zustand $\sigma \in \Sigma$ und terminiert π regulär in einem Zustand $\sigma' \in \Sigma$, dann gilt $E(\sigma, \sigma')$, d.h.: $\sigma'(y) = \sigma(a)$!

Lemma 4.1.2 (Totale Korrektheit von π bzgl. E)

π ist total korrekt für E für jeden Anfangszustand σ mit $\sigma(a) \geq 0$, d.h. wird π angesetzt auf einen Zustand $\sigma \in \Sigma$ mit $\sigma(a) \geq 0$, dann terminiert π regulär in einem Zustand $\sigma' \in \Sigma$ und es gilt $E(\sigma, \sigma')$, d.h.: $\sigma'(y) = \sigma(a)$!

Aufgabe der Programmverifikation

...ist es nun, die Gültigkeit von Lemma 4.1.1 und Lemma 4.1.2 zu beweisen.

Das ist auf verschiedene Weise möglich, z.B. durch:

- direkte Programmverifikation.

Kapitel 4.2

Direkte Programmverifikation

Direkte Verifikation

...wegen $[\![\ ]\!]_{sos} = [\![\ ]\!]_{ns} = [\![\ ]\!]_{ds}$ (s. Äquivalenztheorem 3.3.4) haben wir Wahlfreiheit, die Gültigkeit von Lemma 4.1.1 und Lemma 4.1.2 zu zeigen bzgl. der

- strukturell operationellen
- natürlichen
- denotationellen

Semantik von π.

Beweisskizzen für Lemma 4.1.1 (1)

...bei Wahl der

- strukturell operationellen Semantik ist zu zeigen:

  $$\forall\, \sigma, \sigma' \in \Sigma.\ \langle \pi, \sigma \rangle \Rightarrow\!\!\!>^{*} \sigma' \Rightarrow \sigma'(y) = \sigma(a)! \wedge \sigma(a) \geq \mathbf{0} \quad (*)$$

  wobei $(*)$ durch vollständige Induktion über die Länge der Ableitungsfolge von

  $$\langle \pi, \sigma \rangle \Rightarrow\!\!\!>^{*} \sigma'$$

  bewiesen wird.

- natürlichen Semantik ist zu zeigen:

  $$\forall\, \sigma, \sigma' \in \Sigma.\ \langle \pi, \sigma \rangle \rightarrow \sigma' \Rightarrow \sigma'(y) = \sigma(a)! \wedge \sigma(a) \geq \mathbf{0} \quad (**)$$

  wobei $(**)$ durch strukturelle Induktion über den Aufbau des Ableitungsbaums von

  $$\langle \pi, \sigma \rangle \rightarrow \sigma'$$

  bewiesen wird.

Beweisskizzen für Lemma 4.1.1 (2)

...bei Wahl der

– denotationellen Semantik ist zu zeigen:

$$\psi([\![\,\pi\,]\!]_{ds}) = \textbf{wahr} \quad (***)$$

wobei $\psi : [\,(\Sigma \hookrightarrow \Sigma) \to \mathbb{B}\,]$ ein Prädikat auf der Menge der Zustandstransformationen ist, das definiert ist durch:

$$\forall\, g \in [\Sigma \hookrightarrow \Sigma].\ \psi(g) = \textbf{wahr} \iff_{df}$$
$$\forall\, \sigma, \sigma' \in \Sigma.\ g(\sigma) = \sigma' \Rightarrow \sigma'(y) = \sigma(a)! \wedge \sigma(a) \geq \mathbf{0}$$

Nach Beweis der Zulässigkeit von ψ (s. Definition A.6.1) wird $(***)$ mittels Fixpunktinduktion (s. Anhang A.6) bewiesen.

Übungsaufgabe 4.2.1

1. Vervollständige die Beweisskizzen für Lemma 4.1.1 mit Wahl der
   - strukturell operationellen
   - natürlichen
   - denotationellen

   Semantik für π.
2. Beweise in gleicher Weise die Gültigkeit von Lemma 4.1.2.

Bobachtung

...anhand der (hier nicht) im Detail ausgeführten Beweise von Lemma 4.1.1 und Lemma 4.1.2:

Unabhängig von der Wahl

- strukturell operationeller
- natürlicher
- denotationeller

Semantik für die Beweisausführung, erscheinen die Beweise durch die enge Kopplung an die volle Semantik von WHILE detaillierter und kleinteiliger als es für den Beweis von 'lediglich' Eigenschaft E nötig erscheint.

Erleichterung schafft deshalb der Übergang von direkter zu axiomatischer Programmverifikation, die von 'unwesentlichen' Details der Programmiersprachensemantik abstrahiert und deshalb 'einfachere' Beweise sog. Zusicherungen erlaubt.

Kapitel 4.3

Axiomatische Programmverifikation

Axiomatische Programmverifikation

...beschäftigt sich mit dem Beweis partieller und totaler Korrektheitseigenschaften von Programmen in Form sog. Zusicherungen, deren (semantischer) Gültigkeit und (syntaktischer) Ableitbarkeit mithilfe von Kalkülen.

Grundlegende Begriffe:

- Hoare-Tripel (syntaktische Sicht) bzw. Korrektheitsformeln (semantische Sicht) der Form

$$\{p\}\ \pi\ \{q\} \quad \text{bzw.} \quad [p]\ \pi\ [q]$$

- Gültigkeit von Korrektheitsformeln im Sinn
  - partieller ($\{p\}\ \pi\ \{q\}$)
  - totaler Korrektheit ($[p]\ \pi\ [q]$).
- Ableitungskalküle (oder: Beweiskalküle) für partielle, totale Korrektheit
  - (Kalkül-) Korrektheit
  - (Kalkül-) Vollständigkeit

Zentral für uns

...die Einführung von Begriff und Bedeutung von

- (Kalkül-) Korrektheit
- (Kalkül-) Vollständigkeit

am Beispiel von Ableitungskalkülen für

- partielle, totale Korrektheit

von Zusicherungen.

Aufbau von Zusicherungen

Zusicherungen sind von einer der Formen:

$$\{p\}\ \pi\ \{q\} \quad \text{oder} \quad [p]\ \pi\ [q]$$

mit

- π Programm
- p Prädikat oder logische Formel, sog. Vorbedingung
- q Prädikat oder logische Formel, sog. Nachbedingung
- Vor- und Nachbedingung
  - geschweift geklammert: partielle Korrektheitszusicherung
  - eckig geklammert: totale Korrektheitszusicherung

Je nach Wahl der Grundmenge für Vor- und Nachbedingungen führt dies auf extensionale und intensionale Ansätze axiomatischer Programmverifikation.

Extensionale vs. intensionale Ansätze

...axiomatischer Programmverifikation.

Extensionale Ansätze: Prädikate als Grundmenge

- p, q Prädikate auf Zuständen, d.h. $p, q \in [\Sigma \hookrightarrow \mathbb{B}]$.

Intensionale Ansätze: Logische Formeln als Grundmenge

- p, q Formeln einer Logik $\mathcal{L}$, einer sog. Zusicherungssprache:
  - Aussagenlogik
  - Prädikatenlogik (1. Stufe)
  - Prädikatenlogik 2. Stufe
  - ...

  deren Semantik gegeben ist durch ein Funktional:

$$[\![.]\!]_{\mathcal{L}} : \mathcal{L} \to \Sigma \hookrightarrow \mathbb{B}$$

  d.h. $[\![p]\!]_{\mathcal{L}}, [\![q]\!]_{\mathcal{L}} \in [\Sigma \to \mathbb{B}]$.

  Beispiel: $\mathcal{L} =_{df}$ **Bexpr** mit $[\![.]\!]_{\mathcal{L}} =_{df} [\![.]\!]_{B}$.

Zur 'Erfüllt in'-Sprechweise

...für Prädikate und logische Formeln relativ zu einem Zustand.

Sei $\sigma \in \Sigma$ ein Zustand.

- Extensional: Sei $p : \Sigma \to \mathbb{B}$ ein Prädikat.

  $$p \text{ heißt erfüllt in } \sigma \iff_{df} p(\sigma) = \textbf{wahr} \quad (\iff p(\sigma))$$

- Intensional: Sei $p \in \mathcal{L}$ eine logische Formel.

  $$p \text{ heißt erfüllt in } \sigma \iff_{df} [\![p]\!]_{\mathcal{L}}(\sigma) = \textbf{wahr} \quad (\iff [\![p]\!]_{\mathcal{L}}(\sigma))$$

Zwei wegbereitende klassische Arbeiten

...zu Programmverifikation im Stil axiomatischer Semantik:

- Robert W. Floyd. Assigning Meaning to Programs. Proceedings of Symposium on Applied Mathematics, Mathematical Aspects of Computer Science, American Mathematical Society, New York, Vol. 19, 19-32, 1967.
- Charles A.R. Hoare. An Axiomatic Basis for Computer Programming. Communications of the ACM 12:576-580, 583, 1969.

Kapitel 4.3.1

Partielle und totale Korrektheit

Partielle Korrektheit

Sei π ein WHILE -Programm, p, q zwei logische Formeln oder Prädikate:

Definition 4.3.1.1 (Partielle Korrektheit)

Eine Hoaresche Zusicherung (oder: Korrektheitsformel)

$$\{p\}\ \pi\ \{q\}$$

heißt gültig im Sinn partieller Korrektheit (oder: partiell korrekt) (in Zeichen: $\models_{pk} \{p\}\ \pi\ \{q\}$) gdw für jeden Zustand $\sigma \in \Sigma$ gilt:

Ist die Vorbedingung p in σ erfüllt und terminiert die Berechnung von π angesetzt auf σ regulär im Endzustand σ', dann ist die Nachbedingung q in σ' erfüllt.

Totale Korrektheit

Sei π ein WHILE -Programm, p, q zwei logische Formeln oder Prädikate:

Definition 4.3.1.2 (Totale Korrektheit)

Eine Hoaresche Zusicherung (oder: Korrektheitsformel)

$$[p]\ \pi\ [q]$$

heißt gültig im Sinn totaler Korrektheit (oder: total korrekt) (in Zeichen: $\models_{tk}\ [p]\ \pi\ [q]$) gdw für jeden Zustand $\sigma \in \Sigma$ gilt:

Ist die Vorbedingung p in σ erfüllt, dann terminiert die Berechnung von π angesetzt auf σ regulär im Endzustand σ' und die Nachbedingung q ist in σ' erfüllt.

Erinnerung: Terminierung, Divergenz

Ein WHILE -Programm π angesetzt auf einen Zustand $\sigma \in \Sigma$ terminiert

- regulär gdw π endet nach endlich vielen Schritten in einer finalen Konfiguration, d.h. in einem Zustand $\sigma' \in \Sigma$.
- irregulär gdw π endet nach endlich vielen Schritten in einer Zwischenkonfiguration $\langle \pi', \sigma' \rangle$, die keine Folgekonfiguration besitzt (die Berechnung bleibt in $\langle \pi', \sigma' \rangle$ stecken).

Ein WHILE -Programm π angesetzt auf einen Zustand $\sigma \in \Sigma$

- divergiert gdw π terminiert nicht (weder regulär noch irregulär).

Ein WHILE -Programm π heißt

- divergent gdw π divergiert für jeden Zustand $\sigma \in \Sigma$.

Wichtig: WHILE -Programme sind deterministisch!

Zusammenhang

...von partieller und totaler Korrektheit:

Lemma 4.3.1.3

Für WHILE -Programme π gilt:

$$\models_{tk} [p]\ \pi\ [q] \quad \Longrightarrow \quad \models_{pk} \{p\}\ \pi\ \{q\}$$

...d.h. totale Korrektheit eines WHILE -Programms bzgl. eines Paars aus Vor- und Nachbedingung impliziert auch partielle Korrektheit bzgl. dieses Vor- und Nachbedingungspaars.

Informell

Totale Korrektheit "gleich"

Partielle Korrektheit "plus" Reguläre Termination

Charakterisierung logischer Formeln, Prädikate

...durch erfüllende Zustandsmengen.

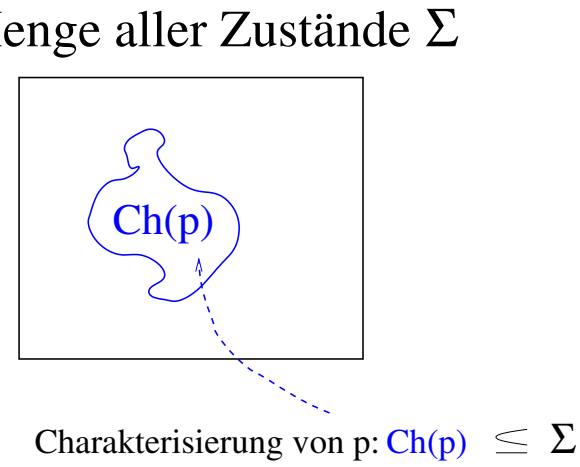
Definition 4.3.1.4 (Charakterisierung)

Die Zustandsmenge

1. $Ch(p) =_{df} \{\sigma \in \Sigma \mid [\![p]\!]_B(\sigma) = \mathbf{wahr}\}$, p logische Formel
2. $Ch(p) =_{df} \{\sigma \in \Sigma \mid p(\sigma) = \mathbf{wahr}\}$, p Prädikat

heißt Charakterisierung von p.

Veranschaulichung:



Partielle und totale Korrektheit

...ausgedrückt mithilfe der Charakterisierung von Vor- und Nachbedingungen von Korrektheitsformeln.

Lemma 4.3.1.5 (Charakterisierungslemma)

Eine Korrektheitsformel $\{p\}\ \pi\ \{q\}$ ist

1. partiell korrekt ($\models_{pk} \{p\}\ \pi\ \{q\}$), falls gilt:

$$[\![\pi]\!](Ch(p)) \subseteq Ch(q)$$

   mit $[\![\pi]\!](Ch(p)) =_{df} \{[\![\pi]\!](\sigma) \mid \sigma \in Ch(p)\}$.

2. total korrekt ($\models_{tk} [p]\ \pi\ [q]$), falls gilt:

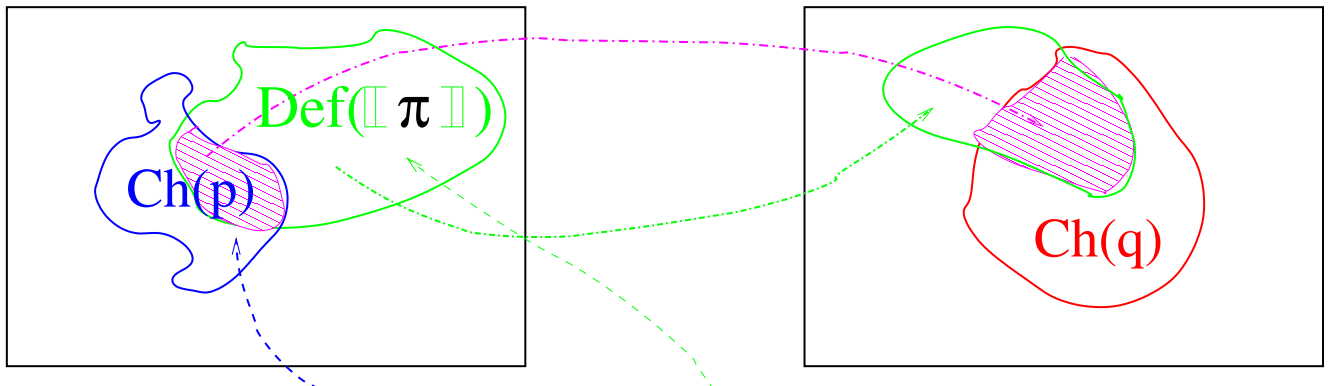
$$\{p\}\ \pi\ \{q\} \text{ partiell korrekt } \wedge\ Ch(p) \subseteq Def([\![\pi]\!])$$

   mit $Def([\![\pi]\!]) =_{df} \{\sigma \mid \pi$ angesetzt auf $\sigma, \sigma \in \Sigma,$ terminiert regulär$\}$ Definitionsbereich v. π.

Veranschaulichung von Lemma 4.3.1.5(1)

...Gültigkeit von $\{p\}\ \pi\ \{q\}$ im Sinn partieller Korrektheit:

Menge aller Zustände Σ



Charakterisierung von p: $Ch(p) \leq \Sigma$

Definitionsbereich von π : $Def(\llbracket \pi \rrbracket) \leq \Sigma$

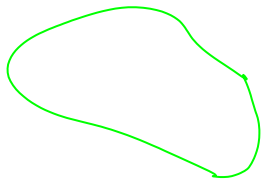


Bild von $\llbracket \pi \rrbracket$ für $Def(\llbracket \pi \rrbracket)$

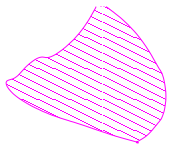
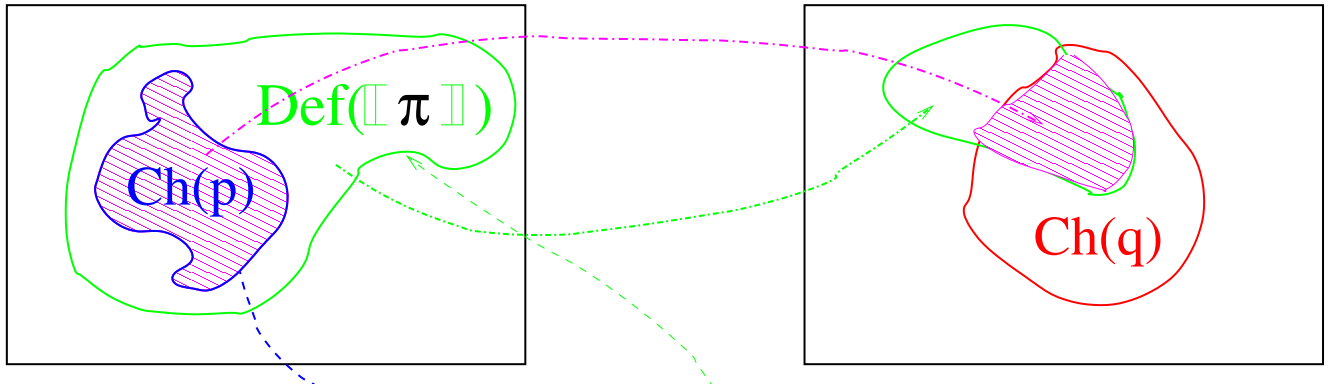


Bild von $\llbracket \pi \rrbracket$ für $Def(\llbracket \pi \rrbracket) \wedge Ch(p)$

Veranschaulichung von Lemma 4.3.1.5(2)

...Gültigkeit von $[p]\ \pi\ [q]$ im Sinn totaler Korrektheit:

Menge aller Zustände Σ



Charakterisierung von p: $\mathrm{Ch}(p) \leq \Sigma$

Definitionsbereich von π : $\mathrm{Def}(\llbracket \pi \rrbracket) \leq \Sigma$

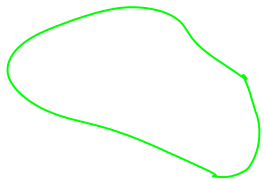


Bild von $\llbracket \pi \rrbracket$ für $\mathrm{Def}(\llbracket \pi \rrbracket)$

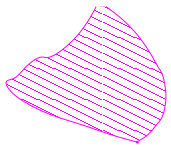


Bild von $\llbracket \pi \rrbracket$ für $\mathrm{Def}(\llbracket \pi \rrbracket) \wedge \mathrm{Ch}(p)$

Beispiele: Hoaresche Zusicherungen

...für das WHILE -Programm zur Berechnung der Fakultäts-funktion.

- Partielle Korrektheit:

$$\{true\}$$
$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$\{y = a!\}$$

- Totale Korrektheit:

$$[a \geq 0]$$
$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$[y = a!]$$

Intensionales, extensionales Verständnis

...von Vor- und Nachbedingungen.

$$\{true\} \;/\; [a \geq 0]$$
$$x := a;\; y := 1;\; \text{while } x \neq 1 \text{ do } y := y * x;\; x := x - 1 \text{ od}$$
$$\{y = a!\} \;/\; [y = a!]$$

Wenn nicht ausdrücklich festgelegt, können die Vorbedingungen (im Bsp. $true$, $a \geq 0$) und die Nachbedingung (im Bsp. $y = a!$) verstanden werden als:

- Logische Ausdrücke, hier Boolesche Ausdr. (intensional):
  $true, (a \geq 0), (y = a!) \in \mathbf{Bexpr}$
- Prädikate (extensional):
  $true, (a \geq 0), (y = a!) \in [\Sigma \to \mathbb{B}]$ definiert durch
  - $\forall \sigma \in \Sigma.\ (true)(\sigma) =_{df}$ **wahr**
  - $\forall \sigma \in \Sigma.\ (a \geq 0)(\sigma) =_{df}\ \sigma(a) \stackrel{\geq}{=} \mathbf{0}$
  - $\forall \sigma \in \Sigma.\ (y = a!)(\sigma) =_{df}\ \sigma(y) = (\sigma(a))!$

Für die Lesart von Vor- und Nachbedingungen

...als Prädikate werden diese als Prädikatkurzschreibweisen gemäß folgender Vereinbarungen angesehen:

$p_1 \wedge p_2$	kurz für	p	def. durch	$p(\sigma) =_{df} p_1(\sigma)$ **und** $p_2(\sigma)$
$p_1 \vee p_2$	kurz für	p	def. durch	$p(\sigma) =_{df} p_1(\sigma)$ **oder** $p_2(\sigma)$
$\neg p$	kurz für	p'	def. durch	$p'(\sigma) =_{df}$ **nicht** $p(\sigma)$
$p[a/x]$	kurz für	p'	def. durch	$p'(\sigma) =_{df} p(\sigma[[\![a]\!]_A(\sigma)/x)$
$p_1 \Rightarrow p_2$	kurz für	p	def. durch	$p(\sigma) =_{df} p_1(\sigma)$ **impl** $p_2(\sigma)$
$a_1 = a_2$	kurz für	p	def. durch	$p(\sigma) =_{df} [\![a_1]\!]_A(\sigma) = [\![a_2]\!]_A(\sigma)$
$a_1 \neq a_2$	kurz für	p	def. durch	$p(\sigma) =_{df} [\![a_1]\!]_A(\sigma) \neq [\![a_2]\!]_A(\sigma)$
...	...	...	...	...

Programmvariablen vs. logische Variablen

...in Zusicherungen $\{p\}\ \pi\ \{q\}$, $[p]\ \pi\ [q]$ wird unterschieden zwischen:

- Programmvariablen
  ...Variablen, die in π vorkommen (und deren Wert möglicherweise durch π verändert wird).
- logischen Variablen
  ...Variablen, auf die von π nicht (oder höchstens lesend) zugegriffen wird (und deren Wert deshalb von π nicht verändert werden kann).

Logische Variablen erlauben es

- sich Werte von Programmvariablen von vor Ausführung eines Programm(stücks) zu 'merken' und sich in Nachbedingungen darauf zu beziehen.

Veranschaulichung anhand zweier Beispiele

$$[x = n \wedge n \geq 0]$$
$$y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$[y = n!]$$

...ist gültiges Hoare-Tripel: Die Nachbedingung trifft eine Aussage über den Zusammenhang des Werts der logischen Variable n (vor und nach Ausführung des Programms) und des Werts von Programmvariable y nach Ausführung des Programms.

$$[x = n \wedge n \geq 0]$$
$$y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$[y = x!]$$

...ist weder gültiges noch sinnvolles Hoare-Tripel: Die Nachbedingung trifft eine (i.a. falsche) Aussage über den Zusammenhang der Werte der Programmvariablen x und y nach Ausführung des Programms; ein Zusammenhang zum Wert v. n fehlt.

Zusammenfassung

Hoaresche Zusicherungen (oder: Korrektheitsformeln) sind Tripel der Form

- $\{p\}\ \pi\ \{q\}$ (für partielle Korrektheit)
- $[p]\ \pi\ [q]$ (für totale Korrektheit)

Die Sprechweisen

- Hoaresches Tripel (oder: Hoare-Tripel)
- Hoaresche Zusicherung (oder: Korrektsheitsformel)

betonen jeweils die

- syntaktische ($\vdash_{pk} \{p\}\ \pi\ \{q\}$, $\vdash_{tk} [p]\ \pi\ [q]$)
- semantische ($\models_{pk} \{p\}\ \pi\ \{q\}$, $\models_{tk} [p]\ \pi\ [q]$)

Sicht auf die Tripel $\{p\}\ \pi\ \{q\}$ bzw. $[p]\ \pi\ [q]$.

Kapitel 4.3.2

Stärkste Nachbedingungen, schwächste und schwächste liberale Vorbedingungen

Formeln, Prädikate stärker/schwächer als

Definition 4.3.2.1 (stärker als, schwächer als)

1. Seien p, q logische Formeln.
   - 1.1 p heißt stärker als q gdw $p \Rightarrow q$.
   - 1.2 p heißt schwächer als q gdw $q \Rightarrow p$.
2. Seien p, q Prädikate.
   - 2.1 p heißt stärker als q gdw $\forall\, \sigma \in \Sigma.\ p(\sigma) \Rightarrow q(\sigma)$.
   - 2.2 p heißt schwächer als q gdw $\forall\, \sigma \in \Sigma.\ q(\sigma) \Rightarrow p(\sigma)$.

Seien p, q logische Formeln oder Prädikate.

Lemma 4.3.2.2

$$
\begin{aligned}
p \text{ stärker als } q &\iff Ch(p) \subseteq Ch(q) \\
&\iff Ch(q) \supseteq Ch(p) \\
&\iff q \text{ schwächer als } p
\end{aligned}
$$

Schwächste Vorbedingungen

Definition 4.3.2.3 (Schwächste Vorbedingung)

Sei π ein Programm, q eine Formel oder Prädikat. Dann heißt $wp(\pi, q)$ schwächste Vorbedingung von π bezüglich Nachbedingung q, wenn gilt:

1. Die Zusicherung $[wp(\pi, q)]\ \pi\ [q]$ ist total korrekt.
2. Für alle p mit p Formel bzw. Prädikat und $[p]\ \pi\ [q]$ total korrekt, gilt: $wp(\pi, q)$ ist schwächer als p, d.h.:

$$p \Rightarrow wp(\pi, q) \ \wedge\ Ch(wp(\pi, q)) \supseteq Ch(p)$$

Schwächste liberale Vorbedingungen

Definition 4.3.2.4 (Schwächste liberale Vorbedingung)

Sei π ein Programm, q eine Formel oder Prädikat. Dann heißt $wlp(\pi, q)$ schwächste liberale Vorbedingung von π bezüglich Nachbedingung q, wenn gilt:

1. Die Zusicherung $\{wlp(\pi, q)\}\ \pi\ \{q\}$ ist partiell korrekt.
2. Für alle p mit p Formel bzw. Prädikat und $\{p\}\ \pi\ \{q\}$ partiell korrekt, gilt: $wlp(\pi, q)$ ist schwächer als p, d.h.:

$$p \Rightarrow wlp(\pi, q) \ \wedge\ Ch(wlp(\pi, q)) \supseteq Ch(p)$$

Stärkste Nachbedingungen

Definition 4.3.2.5 (Stärkste Nachbedingung)

Sei π ein Programm, p eine Formel oder Prädikat. Dann heißt $sp(p, \pi)$ stärkste Nachbedingung von π bezüglich Vorbedingung p, wenn gilt:

1. Die Zusicherung $\{p\}\ \pi\ \{sp(p, \pi)\}$ ist partiell korrekt.
2. Für alle q mit q Formel bzw. Prädikat und $\{p\}\ \pi\ \{q\}$ partiell korrekt, gilt: $sp(p, \pi)$ ist stärker als q, d.h.:

$$sp(p, \pi) \Rightarrow q \ \wedge\ Ch(sp(p, \pi)) \subseteq Ch(q)$$

Stärkste Nach-, schwächste Vorbedingungen

...als Charakterisierung(smeng)en von Formeln, Prädikaten.

Mit den Bezeichnungen $[\![\pi]\!]^{-1}$ und $\mathcal{C}$:

- $\forall \Sigma' \subseteq \Sigma.\ [\![\pi]\!]^{-1}(\Sigma') =_{df} \{\sigma \in \Sigma \mid [\![\pi]\!](\sigma) \text{ def.} \in \Sigma'\}$
- $\mathcal{C}$ Mengenkomplementoperator (zur Grundmenge Σ), d.h.: $\forall \Sigma' \subseteq \Sigma.\ \mathcal{C}(\Sigma') =_{df} \Sigma \backslash \Sigma'$

gilt:

Lemma 4.3.2.6 (Charakterisierungsmengen)

Sei π ein WHILE -Programm, p, q zwei logische Formeln oder Prädikate. Dann gilt:

1. $Ch(wp(\pi, q)) = [\![\pi]\!]^{-1}(Ch(q))$
2. $Ch(wlp(\pi, q)) = [\![\pi]\!]^{-1}(Ch(q)) \cup \mathcal{C}(Def([\![\pi]\!]))$
3. $Ch(sp(p, \pi)) = [\![\pi]\!](Ch(p))$

Zusammenhang

...der Charakterisierungen von Vor- und Nachbedingungen:

Lemma 4.3.2.7

Sei π ein WHILE -Programm, p, q zwei logische Formeln oder Prädikate. Dann gilt:

$$[\![\pi]\!](Ch(p)) \subseteq Ch(q) \iff [\![\pi]\!]^{-1}(Ch(q)) \supseteq Ch(p)$$

Beweis: Übungsaufgabe 4.3.2.8.

Bemerkung

Die Definitionen

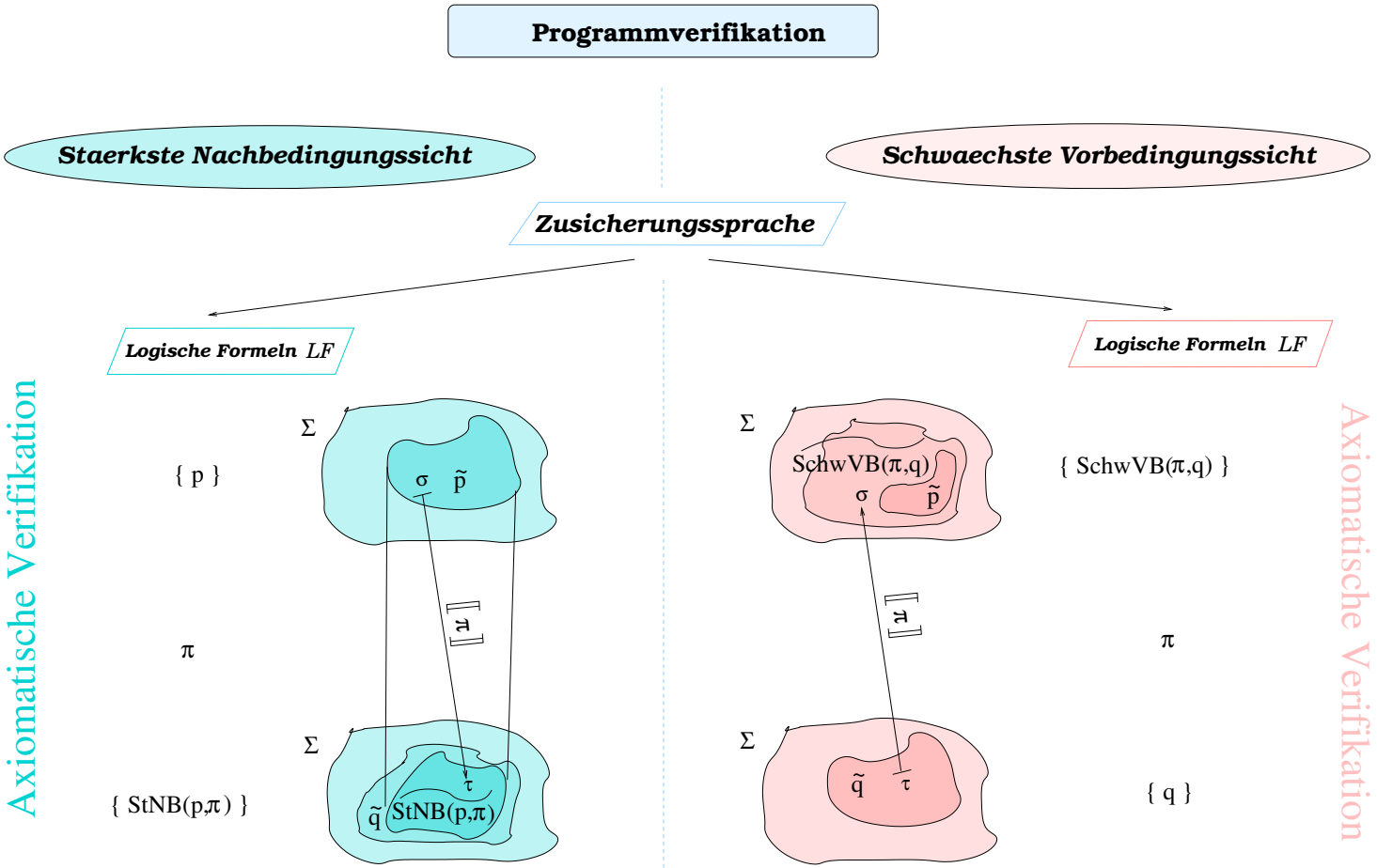
- schwächster/schwächster liberaler Vorbedingungen
- stärkster Nachbedingungen

sind für Prädikate und Formeln in gleicher Weise getroffen.

Allerdings ist nicht gesichert, dass jede Charakterisierungsmenge schwächster Vor-/stärkster Nachbedingungen (i.S.v. Lemma 4.3.2.6) stets durch eine passende Formel der Zusicherungssprache beschrieben werden kann, d.h. deren Charakterisierung mit der Charakterisierungsmenge der entsprechenden schwächsten Vor-/stärksten Nachbedingung übereinstimmt.

Die Darstellbarkeit einer Zustandsmenge als Formel ist an die Ausdruckskraft der Formelsprache, d.h. der gewählten Logik, gebunden (s. dazu auch Kap. 4.5 zur Vollständigkeit intensionaler Ableitungskalküle).

Stärkste Nachbed.-, schwächste Vorbed.-Sicht



StNB(p, π) ∈ *LF muss erfüllen:*

(1) $\models_{PV}$ { p } π { StNB(p,π) }

(2) ∀ q ∈ *LF*. $\models_{PV}$ { p } π { q } impliziert StNB(p,π) $\Longrightarrow$ q

SchwVB(π,q) ∈ *LF muss erfüllen:*

(1) $\models_{PV}$ { SchwVB(π,q) } π { q }

(2) ∀ p ∈ *LF*. $\models_{PV}$ { p } π { q } impliziert p $\Longrightarrow$ SchwVB(π,q)

Übungsaufgabe 4.3.2.8

Beweise Lemma 4.3.2.7.

Kapitel 4.3.3

Korrektheit, Vollständigkeit von Ableitungskalkülen

Korrektheits-, Vollständigkeitsfrage

...für Ableitungs- (oder (Beweis-) Kalküle oder Regelwerke) K für partielle oder totale Korrektheit.

Korrektheitsfrage:

- Ist jede mithilfe von K ableitbare (oder: syntaktisch beweisbare) (in Zeichen: $\vdash_K$) Korrektheitsformel partiell/total korrekt?

Vollständigkeitsfrage:

- Ist jede partiell/total korrekte Korrektheitsformel (in Zeichen: $\models_{pk/tk}$) mithilfe von K ableitbar (oder: syntaktisch beweisbar)?

Korrektheit und Vollständigkeit eines Kalküls

...für partielle Korrektheit.

Definition 4.3.3.1 (Korrektheit und Vollständigkeit)

Ein Beweiskalkül K_{pk} für partielle Korrektheit heißt

1. korrekt (engl. sound), falls gilt: Ist eine Korrektheitsformel mit K_{pk} ableitbar ($\vdash_{K_{pk}} \{p\}\ \pi\ \{q\}$), dann ist sie semantisch gültig im Sinn partieller Korrektheit ($\models_{pk} \{p\}\ \pi\ \{q\}$), d.h.:

$$\vdash_{K_{pk}} \{p\}\ \pi\ \{q\} \implies \models_{pk} \{p\}\ \pi\ \{q\}$$

2. vollständig (engl. complete), falls gilt: Ist eine Korrektheitsformel semantisch gültig im Sinn partieller Korrektheit ($\models_{pk} \{p\}\ \pi\ \{q\}$), dann ist sie mit K_{pk} ableitbar ($\vdash_{K_{pk}} \{p\}\ \pi\ \{q\}$), d.h.:

$$\models_{pk} \{p\}\ \pi\ \{q\} \implies \vdash_{K_{pk}} \{p\}\ \pi\ \{q\}$$

Korrektheit und Vollständigkeit eines Kalküls

...für totale Korrektheit.

Definition 4.3.3.2 (Korrektheit und Vollständigkeit)

Ein Beweiskalkül K_{tk} für totale Korrektheit heißt

1. korrekt (engl. sound), falls gilt: Ist eine Korrektheitsformel mit K_{tk} ableitbar ($\vdash_{K_{tk}} [p]\ \pi\ [q]$), dann ist sie auch semantisch gültig im Sinn totaler Korrektheit ($\models_{tk} [p]\ \pi\ [q]$), d.h.:

$$\vdash_{K_{tk}} [p]\ \pi\ [q] \implies \models_{tk} [p]\ \pi\ [q]$$

2. vollständig (engl. complete), falls gilt: Ist eine Korrektheitsformel semantisch gültig im Sinn totaler Korrektheit ($\models_{tk} [p]\ \pi\ [q]$), dann ist sie auch mit K_{tk} ableitbar ($\vdash_{K_{tk}} [p]\ \pi\ [q]$), d.h.:

$$\models_{tk} [p]\ \pi\ [q] \implies \vdash_{K_{tk}} [p]\ \pi\ [q]$$

Kapitel 4.4

Ableitungskalkül HK_{pk} für partielle Korrektheit

Hoare-Kalkül HK_{pk} für partielle Korrektheit

Seien p, q zwei logische Formeln oder Prädikate.

Axiome:

[skip] $$\frac{}{\{p\}\ skip\ \{p\}}$$

[ass] $$\frac{}{\{p[t/x]\}\ x{:=}t\ \{p\}}$$

(Rückwärtssubstitution, Rückwärtsregel)

Regeln:

[comp] $$\frac{\{p\}\ \pi_1\ \{r\},\ \{r\}\ \pi_2\ \{q\}}{\{p\}\ \pi_1;\pi_2\ \{q\}}$$

[ite] $$\frac{\{p \wedge b\}\ \pi_1\ \{q\},\ \{p \wedge \neg b\}\ \pi_2\ \{q\}}{\{p\}\ \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}\ \{q\}}$$

[while$_{pk}$] $$\frac{\{I \wedge b\}\ \pi\ \{I\}}{\{I\}\ \text{while } b \text{ do } \pi \text{ od}\ \{I \wedge \neg b\}}$$

(I Invariante)

[cons] $$\frac{p \Rightarrow p_1,\ \{p_1\}\ \pi\ \{q_1\},\ q_1 \Rightarrow q}{\{p\}\ \pi\ \{q\}}$$

Anmerkungen zur while-Regel [while$_{pk}$]

$$[\text{while}_{pk}] \quad \frac{\{I \wedge b\}\ \pi\ \{I\}}{\{I\}\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ \{I \wedge \neg b\}} \qquad (I \text{ Invariante})$$

Informell:

Die Prämisse der while-Regel [while$_{pk}$] besagt:

- Gelten vor Ausführung des Schleifenrumpfs die Abbruchbedingung b der Schleife und die Formel bzw. das Prädikat I, so gilt I auch nach Ausführung des Schleifenrumpfs.

I wird deshalb als Schleifeninvariante (oder Invariante) der while-Schleife bezeichnet.

Die Konklusion der while-Regel [while$_{pk}$] besagt:

- Die Schleifeninvariante gilt vor Eintritt in und nach Austritt aus der Schleife; zusätzlich gilt nach Austritt aus der Schleife die Negation der Abbruchbedingung b der Schleife, d.h. das Prädikat $\neg\, b$.

Anmerkungen zur Konsequenzregel (1)

[cons] $$\frac{p \Rightarrow p_1,\ \{p_1\}\ \pi\ \{q_1\},\ q_1 \Rightarrow q}{\{p\}\ \pi\ \{q\}}$$

Informell: Die Konsequenzregel ist die

- Schnittstelle zwischen den programmbezogenen Axiomen und Regeln des Beweiskalküls und den logischen Formeln der Zusicherungssprache.

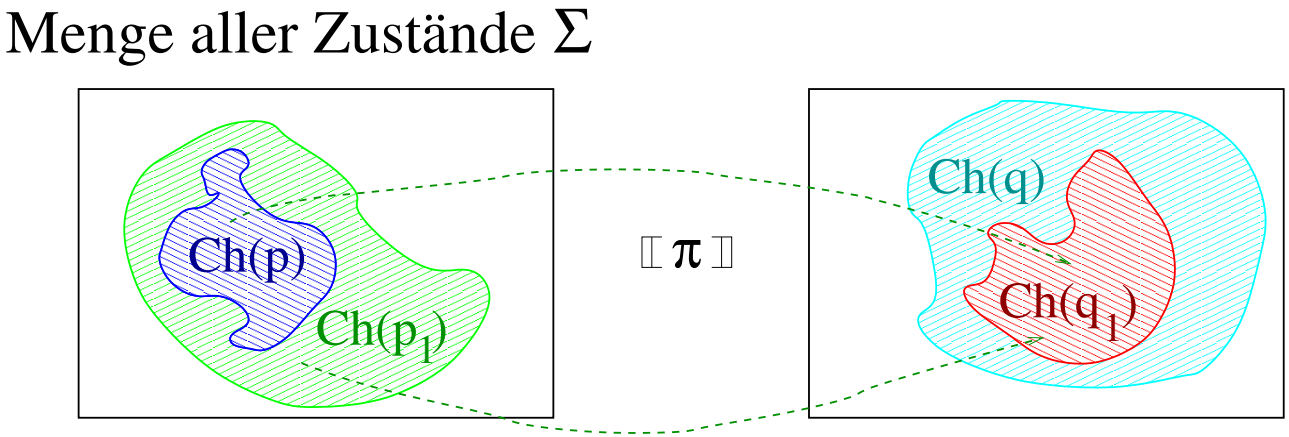
Sie erlaubt

- Vorbedingungen zu verstärken
  ...Übergang von p_1 zu p: Möglich, falls
  $$p \Rightarrow p_1 \ (\ \Leftrightarrow\ \ Ch(p) \subseteq Ch(p_1)\ )$$
- Nachbedingungen abzuschwächen
  ...Übergang von q_1 zu q: Möglich, falls
  $$q_1 \Rightarrow q \ (\ \Leftrightarrow\ \ Ch(q_1) \subseteq Ch(q)\ )$$

um so die Anwendung anderer Beweisregeln zu ermöglichen.

Anmerkungen zur Konsequenzregel (2)

...Veranschaulichung von Verstärkung und Abschwächung:



$$p \Longrightarrow p_1 \qquad \{p_1\}\ \pi\ \{q_1\} \qquad q_1 \Longrightarrow q$$

z.B.: $x>5 \Longrightarrow x>0 \qquad \{\ x>0\ \}\ \pi\ \{\ y>5\ \} \qquad y>5 \Longrightarrow y>0$

Anmerkungen zur Konsequenzregel (3)

Pragmatisch ist es vorteilhaft, zusätzlich zur Konsequenzregel

$$[\text{cons}] \quad \frac{p \Rightarrow p_1,\ \{p_1\}\ \pi\ \{q_1\},\ q_1 \Rightarrow q}{\{p\}\ \pi\ \{q\}}$$

auch folgende Spezialisierungen der Konsequenzregel zum Beweiskalkül hinzuzunehmen:

$$[\text{cons}'] \quad \frac{p \Rightarrow p_1,\ \{p_1\}\ \pi\ \{q\}}{\{p\}\ \pi\ \{q\}}$$

$$[\text{cons}''] \quad \frac{\{p\}\ \pi\ \{q_1\},\ q_1 \Rightarrow q}{\{p\}\ \pi\ \{q\}}$$

um das Mitschleppen der trivialen Implikationen $p \Rightarrow p$ bzw. $q \Rightarrow q$ zu vermeiden

In der Folge gehen wir davon aus, dass HK_{pk} (und später auch HK'_{tk}, HK_{tk}) die Konsequenzregeln [cons], [cons′] und [cons″] enthält.

Diskussion von Vorwärtszuweisungsregeln

Die (korrekte) Vorwärtsregel für die Zuweisung

$$[\text{ass}_{vw}] \quad \frac{}{\{p\}\ x{:=}t\ \{\exists z.\ p[z/x] \wedge x{=}t[z/x]\}} \qquad (\text{'Vorwärtssubstitution'})$$

...mag natürlich erscheinen, ist aber beweistechnisch lästig durch das notwendige Mitschleppen quantifizierter Formeln.

Die möglicherweise naheliegend scheinende quantorfreie Variante der Vorwärtszuweisungsregel:

$$[\text{ass}_{vw\text{-}naiv}] \quad \frac{}{\{p\}\ x{:=}t\ \{p[t/x]\}}$$

...ist nicht korrekt (Beweis: Übungsaufgabe).

Kapitel 4.5

Korrektheit und Vollständigkeit von HK_{pk}

Korrektheit von HK_{pk}

Sei π ein WHILE -Programm, p, q zwei logische Formeln oder Prädikate:

Theorem 4.5.1 (Korrektheit von HK_{pk})

Der Ableitungkalkül HK_{pk} ist korrekt, d.h. jedes mit den Axiomen und Regeln von HK_{pk} ableitbare Hoaresche Tripel (in Zeichen: $\vdash_{HK_{pk}} \{p\}\ \pi\ \{q\}$) ist gültig im Sinn partieller Korrektheit (in Zeichen: $\models_{pk} \{p\}\ \pi\ \{q\}$), d.h.:

$$\vdash_{HK_{pk}} \{p\}\ \pi\ \{q\} \implies \models_{pk} \{p\}\ \pi\ \{q\}$$

Beweis durch strukturelle Induktion über den Aufbau des Ableitungsbaums der Korrektheitsformel $\{p\}\ \pi\ \{q\}$.

Vollständigkeit von HK_{pk}

Sei π ein WHILE -Programm, p, q zwei Prädikate (extensionaler Ansatz):

Theorem 4.5.2 (Vollständigkeit von HK_{pk})

Der Ableitungkalkül HK_{pk} ist vollständig, d.h. jede im Sinn partieller Korrektheit gültige Korrektheitsformel (in Zeichen: $\models_{pk} \{p\}\ \pi\ \{q\}$) ist mit den Axiomen und Regeln von HK_{pk} ableitbar (in Zeichen: $\vdash_{pk} \{p\}\ \pi\ \{q\}$), d.h.:

$$\models_{pk} \{p\}\ \pi\ \{q\} \implies \vdash_{pk} \{p\}\ \pi\ \{q\}$$

Beweis durch strukturelle Induktion über den Aufbau von π.

Für den intensionalen Ansatz

...mit Wahl von **Bexpr** als Logik bzw. Zusicherungssprache und $[\![\ ]\!]_B$ als Semantik gilt Vollständigkeitstheorem 4.5.2 nicht.

Wichtig ist folgende Beobachtung:

Lemma 4.5.3

1. $\forall\, \Sigma' \subseteq \Sigma.\ \exists\, p \in [\Sigma \to \mathbb{B}].\ Ch(p) = \Sigma'$
2. $\exists\, \Sigma' \subseteq \Sigma.\ \forall\, p \in \mathbf{Bexpr}.\ Ch(p) \neq \Sigma'$

Beweis von

1. Sei $\Sigma' \subseteq \Sigma$ beliebig. Definiere $p : \Sigma \to \mathbb{B}$ durch:

$$\forall\, \sigma \in \Sigma.\ p(\sigma) = \mathbf{wahr} \iff_{df} \sigma \in \Sigma'$$

   Wie man leicht sieht, gilt: $Ch(p) = \Sigma'$.

2. Beweis durch Reduktion auf das Halteproblem. □

Die Aussage von Lemma 4.5.3(2)

...informell gedeutet:

- Anders als Prädikate ist **Bexpr** nicht ausdruckskräftig genug, jede Teilmenge Σ' von Σ durch eine Formel p zu beschreiben, d.h. durch eine Formel, deren Charakterisierung $Ch(p)$ gerade Σ' ist.
- Anders als durch Prädikate sind daher insbesondere schwächste oder schwächste liberale Vorbedingungen für Paare aus Programm und Nachbedingung i.a. nicht durch Formeln aus **Bexpr** ausdrückbar.
- Daran scheitern Beweisversuche von Vollständigkeitstheorem 4.5.2 für den intensionalen Ansatz mit **Bexpr** als Zusicherungssprache und $[\![\ ]\!]_B$ als Semantik.

Zu Lem. 4.5.3(2): Halteproblemreduktion (1)

Sei π ein WHILE -Programm mit unentscheidbarem Halteproblem, $\mathcal{L} =_{df}$ **Bexpr** die Zusicherungssprache mit Semantik $[\![\ ]\!]_{\mathcal{L}} =_{df} [\![\ ]\!]_{B}$.

Gemäß Definition 4.3.2.4 und 4.3.1.1 gilt:

Die Korrektheitsformel

$$\{wlp(\pi, false)\}\ \pi\ \{false\} \qquad (a)$$

ist partiell korrekt gdw π terminiert nicht regulär angesetzt auf einen Zustand aus $Ch(wlp(\pi, false))$.

Zu Lem. 4.5.3(2): Halteproblemreduktion (2)

Angenommen, es gibt $f_\pi \in \mathcal{L}$ mit:

$$\forall\, \sigma \in \Sigma.\ [\![\, f_\pi \,]\!]_B(\sigma) = \textbf{wahr} \iff$$
$$\pi \text{ terminiert nicht regulär angesetzt auf } \sigma \quad (b)$$

Mit $f_\pi \in \mathcal{L}$ ist auch $\neg f_\pi \in \mathcal{L}$ mit:

$$\forall\, \sigma \in \Sigma.\ [\![\, \neg f_\pi \,]\!]_B(\sigma) = \textbf{wahr} \iff$$
$$\pi \text{ terminiert regulär angesetzt auf } \sigma \quad (c)$$

Aus (b), (c) folgt zusammen mit (a) und Definition 4.3.2.4:

$$Ch(f_\pi) = Ch(wlp(\pi, false))$$
$$Ch(\neg f_\pi)\ (= \mathcal{C}(Ch(wlp(\pi, false)))) = \Sigma \setminus Ch(wlp(\pi, false))$$

Da $[\![\, f_\pi \,]\!]_B(\sigma)$, $[\![\, \neg f_\pi \,]\!]_B(\sigma)$ für alle $\sigma \in \Sigma$ berechenbar sind (s. Kap. 1.5) und somit $Ch(f_\pi)$, $Ch(\neg f_\pi)$ entscheidbar sind, ergibt sich ein Widerspruch zur Unentscheidbarkeit des Halteproblems für π. Folglich kann die Annahme $f_\pi \in \mathcal{L}$ nicht richtig sein.

Vollständigkeit intensionaler Ansätze

...erfordert den Übergang von **Bexpr** zu

- ausdruckskräftigeren

mächtigeren Logiken als Zusicherungssprachen.

Vollständigkeit intensionaler Ansätze ist deshalb i.a. nur relativ zur Ausdruckskraft der Zusicherungssprache und der Entscheidbarkeit (oder schwächer Aufzählbarkeit) der zugrundeliegenden Theorien (bei uns die Theorie Boolescher Ausdrücke über arithmetischen Ausdrücken und ganzen Zahlen) erreichbar.

Das ermöglicht Beweise

- relativer Vollständigkeit (im Sinn von Cook)

passender Hoarescher Ableitungs- (oder Beweis-) Kalküle.

Die Details relativer Vollständigkeiten sind für uns in der Folge nicht relevant und werden daher nicht näher betrachtet.

Übungsaufgabe 4.5.4

Warum führt der prädikatenbasierte extensionale Ansatz anders als der formelbasierte intensionale Ansatz nicht zum Widerspruch zur Unentscheidbarkeit des Halteproblems nach dem Muster der Überlegungen zu Lemma 4.5.3(2)?

Kapitel 4.6

Partielle Korrektheitsbeweise

Kapitel 4.6.1

Fakultät, ganzzahlige Division mit Rest

Die Programme

1) Fakultät

$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$

2) Ganzzahlige Division mit Rest

$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$

Die Zusicherungen partieller Korrektiheit

Lemma 4.6.1.1 (Fakultät)

Die Hoaresche Zusicherung

$$\{true\}$$
$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$\{y = a!\}$$

ist gültig im Sinn partieller Korrektheit.

Lemma 4.6.1.2 (Ganzzahlige Division mit Rest)

Die Hoaresche Zusicherung

$$\{x \geq 0 \wedge y > 0\}$$
$$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$$
$$\{x = q * y + r \wedge 0 \leq r < y\}$$

ist gültig im Sinn partieller Korrektheit.

Beweisdarstellungen: Baum und lineare Skizze

Aufgabe: Beweis von Lemma 4.6.1.1 und 4.6.1.2.

...d.h., zeigen, dass die Hoareschen Tripel für die Berechnung der Fakultätsfunktion und der ganzzahligen Division mit Rest gültig sind im Sinn partieller Korrektheit.

Wir zeigen die Beweise in zwei notationellen Varianten. Als:

1. Ableitungsbaum (kanonische Variante) (Kap. 4.6.2).
2. lineare Beweisskizze (pragmatische Variante) (Kap. 4.6.3).

Kapitel 4.6.2

Ableitungsbäume

Fakultätsprogramm: Kanonischer Beweis (1)

...Beweis von Lemma 4.6.1.1.

Vorbereitung:

Schritt 1: Wahl einer ausreichend starken Invariante

"Träumen" einer (geeigneten) Invariante, hier:

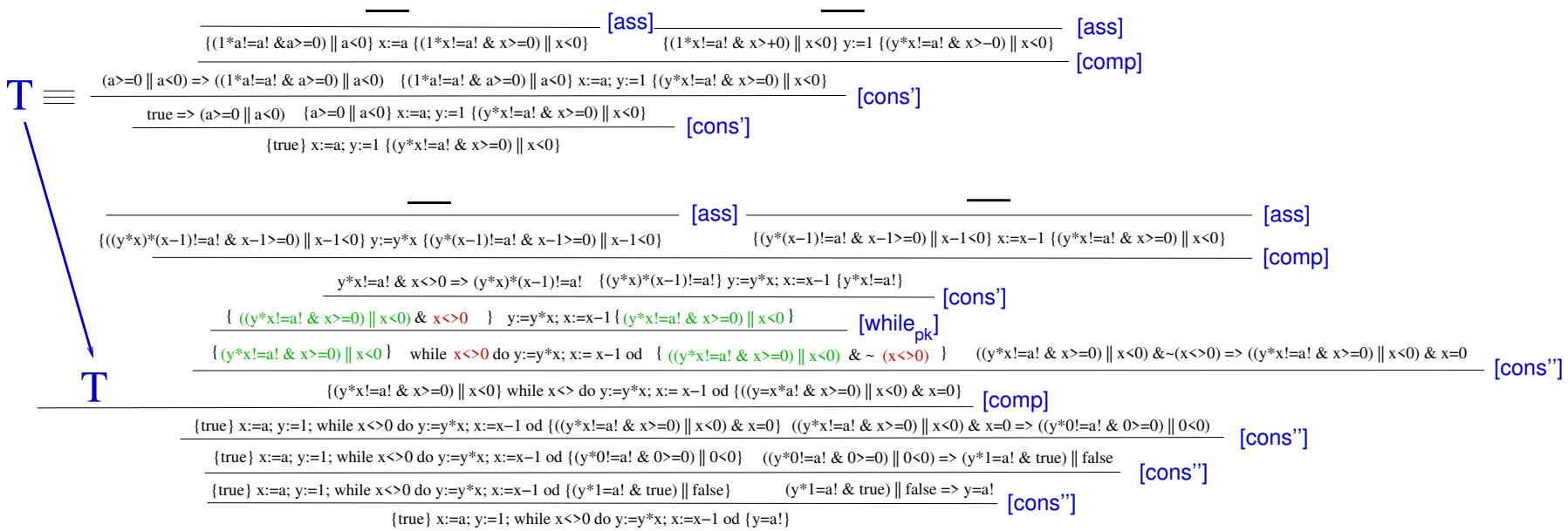
- $I \equiv (y * x! = a! \wedge x \geq 0) \vee x < 0$

um die Regel $[\text{while}_{pk}]$ anwenden und den Beweis erfolgreich abschließen zu können.

Fakultätsprogramm: Kanonischer Beweis (2)

Beweis im engeren Sinn:

Schritt 2: Konstruktion des Ableitungsbaums



& : Logisches und || : Logisches oder ~ : Logisches nicht <> : ungleich–Relator >= : größergleich–Relator

Fakultätsprogramm: Kanonischer Beweis (3)

Zusammenfassung:

Durch Konstruktion des Ableitungsbaums haben wir gezeigt:

Die Hoaresche Zusicherung

$$\{true\}$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$\{y = a!\}$$

ist mit den Axiomen und Regeln von HK_{pk} ableitbar. Gemäß Korrektheitstheorem 4.5.1 ist die Zusicherung damit gültig im Sinn partieller Korrektheit und der Beweis von Lemma 4.6.1.1 somit erbracht.

Divisionsprogramm: Kanonischer Beweis (1)

...Beweis von Lemma 4.6.1.2.

Vorbereitung:

Schritt 1: Wahl einer ausreichend starken Invariante

"Träumen" einer (geeigneten) Invariante, hier:

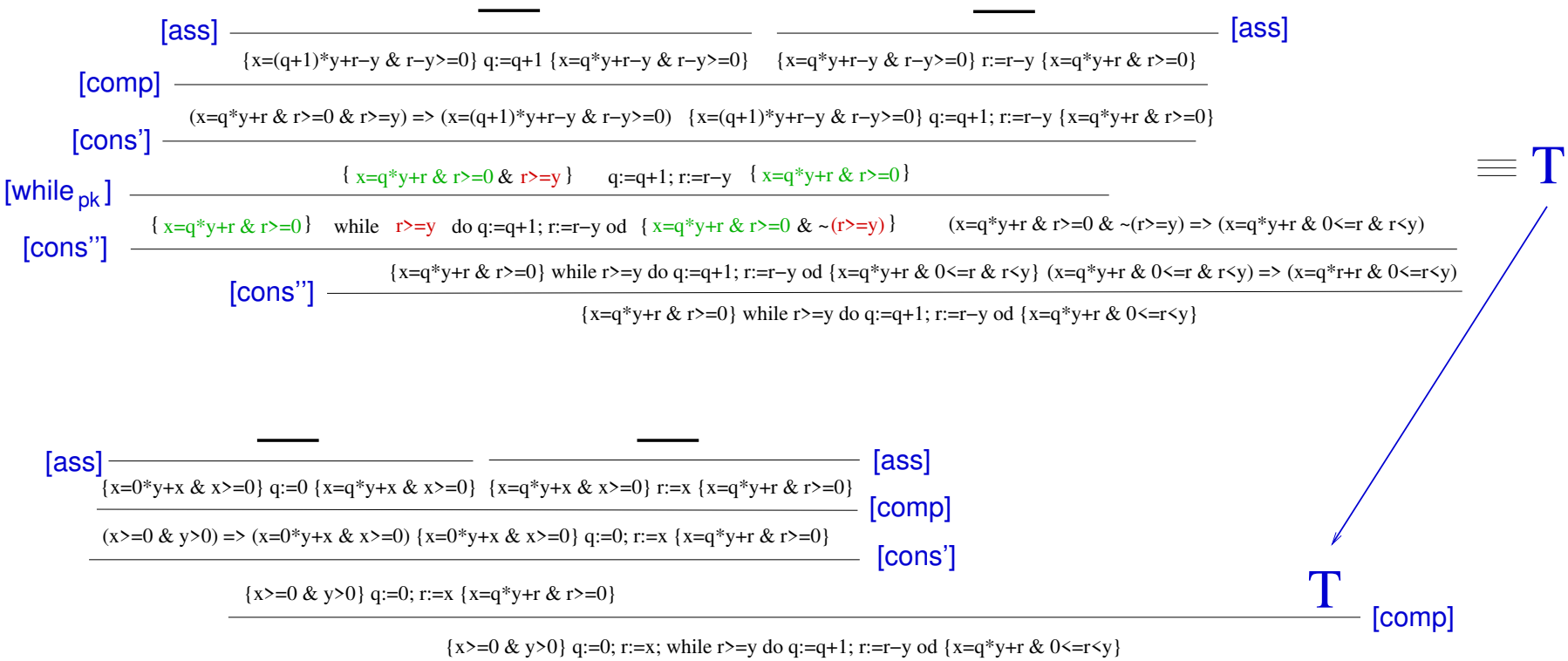
- $I \equiv x = q * y + r \wedge 0 \leq r < y$

um die Regel $[\text{while}_{pk}]$ anwenden und den Beweis erfolgreich abschließen zu können.

Divisionsprogramm: Kanonischer Beweis (2)

Beweis im engeren Sinn:

Schritt 2: Konstruktion des Ableitungsbaums



& : Logisches und ~ : Logisches nicht >= : größergleich–Relator <= : kleinergleich–Relator

Divisionsprogramm: Kanonischer Beweis (3)

Zusammenfassung:

Durch Konstruktion des Ableitungsbaums haben wir gezeigt:

Die Hoaresche Zusicherung

$$\{x \geq 0 \wedge y > 0\}$$
$$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$$
$$\{x = q * y + r \wedge 0 \leq r < y\}$$

ist mit den Axiomen und Regeln von HK_{pk} ableitbar. Gemäß Korrektheitstheorem 4.5.1 ist die Zusicherung damit gültig im Sinn partieller Korrektheit und der Beweis von Lemma 4.6.1.2 somit erbracht.

Lineare Beweise als Ableitungsbaumalternative

Die von den Kalkülregeln induzierte kanonische Darstellung Hoarescher Korrektheitsbeweise in Form von

- Ableitungsbäumen

ist i.a. schwerfällig und unhandlich.

Als Ergänzung hat sich deshalb eine pragmatische notationelle Variante eingebürgert, bei der

- Zusicherungen in Form von Annotationen

in den Programmtext eingestreut werden.

Man spricht von sog.

- linearen Beweisen oder linearen Beweisskizzen

aus denen sich der Ableitungsbaum jederzeit rekonstruieren lässt.

Vorteil

...des linearen gegenüber des baumartigen Notationsstils:

- Keine Redundanz: Kompaktere, knappere Beweise.
- Kein Informationsverlust: Ableitungsbaum jederzeit aus der Beweisskizze herstellbar.

In der Folge demonstrieren wir den Notationsstil

- linearer Beweisskizzen

am Beispiel des Beweises von Lemma 4.6.1.1.

Kapitel 4.6.3

Lineare Beweisskizzen

Fakultätsprogramm: Arbeitsplan

...Beweis von Lemma 4.6.1.1:

Wir zeigen durch Angabe einer linearen Beweisskizze, dass das Hoaresche Tripel

$$\{true\}$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$\{y = a!\}$$

gültig ist im Sinn partieller Korrektheit.

...die lineare Beweisskizze dafür entwickeln wir Schritt für Schritt!

Fakultätsprogramm: Lineare Beweisskizze (1)

Schritt 1: Wahl einer ausreichend starken Invariante.

"Träumen" einer (geeigneten) Invariante, hier:

- $I \equiv x = q * y + r \wedge 0 \leq r < y$

um die Regel $[\text{while}_{pk}]$ anwenden und den Beweis erfolgreich abschließen zu können.

Fakultätsprogramm: Lineare Beweisskizze (2)

Schritt 2: Behandlung des Rumpfs der while-Schleife.

Die Herleitung von

$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$

$$y := y * x;$$

$$x := x - 1;$$

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

erlaubt mithilfe der $[\text{while}_{pk}]$-Regel den Übergang zu:

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$\text{while } x \neq 0 \text{ do}$$

$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$

$$y := y * x;$$

$$x := x - 1;$$

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$\text{od } [\text{while}_{pk}]$$

$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ \neg(x \neq 0)\}$$

Fakultätsprogramm: Lineare Beweisskizze (3)

Behandlung des Rumpfs der while-Schleife im Detail:

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge x \neq 0\}$$

$$y := y * x;$$

$$x := x - 1;$$

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

Fakultätsprogramm: Lineare Beweisskizze (4)

Wegen Rückwärtszuweisungsregel wird der Rumpf der while-Schleife von hinten nach vorne bearbeitet:

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge x \neq 0\}$$

$$y := y * x;$$

$$\{(y * (x-1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$x := x - 1; \text{ [ass]}$$

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

Fakultätsprogramm: Lineare Beweisskizze (5)

Nochmalige Anwendung der [ass]-Regel liefert:

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge x \neq 0\}$$

$$\{((y * x) * (x - 1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$y := y * x; \text{ [ass]}$$

$$\{(y * (x - 1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$x := x - 1; \text{ [ass]}$$

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

...wobei noch eine Beweislücke verbleibt!

Fakultätsprogramm: Lineare Beweisskizze (6)

Schließen der Beweislücke in der zugrundeliegenden Theorie algebraischer und Boolescher Ausdrücke:

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge x \neq 0\}$$

$$\Downarrow \text{[cons']}$$

$$\{((y * x) * (x - 1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$y := y * x; \text{ [ass]}$$

$$\{(y * (x - 1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$x := x - 1; \text{ [ass]}$$

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

Fakultätsprogramm: Lineare Beweisskizze (7)

Anwendung der [while$_{pk}$]-Regel liefert nun wie gewünscht:

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

$$\text{while } x \neq 0 \text{ do}$$

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge x \neq 0\}$$

$$\Downarrow [\text{cons}']$$

$$\{((y * x) * (x - 1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$y := y * x; \ [\text{ass}]$$

$$\{(y * (x - 1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$x := x - 1; \ [\text{ass}]$$

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

$$\text{od } [\text{while}_{pk}]$$

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge \neg(x \neq 0)\}$$

Fakultätsprogramm: Lineare Beweisskizze (8)

Schritt 3: Zur gewünschten Nachbedingung verbleibt ebenfalls eine Beweislücke:

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

$$\text{while } x \neq 0 \text{ do}$$

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge x \neq 0\}$$

$$\Downarrow [\text{cons}']$$

$$\{((y * x) * (x-1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$y := y * x; [\text{ass}]$$

$$\{(y * (x-1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$x := x - 1; [\text{ass}]$$

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

$$\text{od } [\text{while}_{pk}]$$

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge \neg(x \neq 0)\}$$

$$\{y = a!\}$$

Fakultätsprogramm: Lineare Beweisskizze (9)

Schließen der Beweislücke in der zugrundeliegenden Theorie arithmetischer und Boolescher Ausdrücke:

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
$$\text{while } x \neq 0 \text{ do}$$
$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$
$$\Downarrow [\text{cons}']$$
$$\{((y * x) * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$
$$y := y * x; \ [\text{ass}]$$
$$\{(y * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$
$$x := x - 1; \ [\text{ass}]$$
$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
$$\text{od } [\text{while}_{pk}]$$
$$\{((y * x! = a! \ \wedge \ x \geq 0) \vee \ x < 0) \ \wedge \ \neg(x \neq 0)\}$$
$$\Downarrow [\text{cons}'']$$
$$\{((y * x! = a! \ \wedge \ x \geq 0) \vee \ x < 0) \ \wedge \ x = 0\}$$
$$\Downarrow [\text{cons}'']$$
$$\{(y * x! = a! \ \wedge \ x \geq 0 \ \wedge \ x = 0) \ \vee \ (x < 0 \ \wedge \ x = 0)\}$$
$$\Downarrow [\text{cons}'']$$
$$\{(y * 0! = a! \ \wedge \ x = 0) \ \vee \ \mathit{false}\}$$
$$\Downarrow [\text{cons}'']$$
$$\{y * 1 = a! \ \wedge \ x = 0\}$$
$$\Downarrow [\text{cons}'']$$
$$\{y = a!\}$$

Fakultätsprogramm: Lineare Beweisskizze (10)

Aus Platzgründen etwas verkürzt dargestellt:

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

$$\text{while } x \neq 0 \text{ do}$$

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge x \neq 0\}$$

$$\Downarrow [\text{cons}']$$

$$\{((y * x) * (x-1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$y := y * x; \ [\text{ass}]$$

$$\{(y * (x-1)! = a! \wedge x - 1 \geq 0) \vee x - 1 < 0\}$$

$$x := x - 1; \ [\text{ass}]$$

$$\{(y * x! = a! \wedge x \geq 0) \vee x < 0\}$$

$$\text{od } [\text{while}_{pk}]$$

$$\{((y * x! = a! \wedge x \geq 0) \vee x < 0) \wedge \neg(x \neq 0)\}$$

$$\Downarrow 5x \ [\text{cons}'']$$

$$\{y = a!\}$$

Fakultätsprogramm: Lineare Beweisskizze (11)

Schritt 4: Es verbleibt, die Beweislücke zur gewünschten Vorbedingung zu schließen:

$$\{true\}$$

$$x := a;$$

$$y := 1;$$

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$\text{while } x \neq 0 \text{ do}$$

$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$

$$\Downarrow [\text{cons}']$$

$$\{((y * x) * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$

$$y := y * x; \ [\text{ass}]$$

$$\{(y * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$

$$x := x - 1; \ [\text{ass}]$$

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$\text{od } [\text{while}_{pk}]$$

$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ \neg(x \neq 0)\}$$

$$\Downarrow 5x \ [\text{cons}'']$$

$$\{y = a!\}$$

Fakultätsprogramm: Lineare Beweisskizze (12)

Einmalige Anwendung der [ass]-Regel liefert:

$$\{true\}$$

$$x := a;$$

$$\{(1 * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$y := 1; \ [ass]$$

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$\text{while } x \neq 0 \text{ do}$$

$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$

$$\Downarrow [cons']$$

$$\{((y * x) * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$

$$y := y * x; \ [ass]$$

$$\{(y * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$

$$x := x - 1; \ [ass]$$

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$\text{od } [while_{pk}]$$

$$\{((y * x! = a! \ \wedge \ x \geq 0) \vee \ x < 0) \ \wedge \ \neg(x \neq 0)\}$$

$$\Downarrow 5x \ [cons'']$$

$$\{y = a!\}$$

Fakultätsprogramm: Lineare Beweisskizze (13)

Nochmalige Anwendung der [ass]-Regel liefert:

$$\{true\}$$

$$\{(1 * a! = a! \ \wedge \ a \geq 0) \ \vee \ a < 0\}$$

$$x := a; \ [ass]$$

$$\{(1 * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$y := 1; \ [ass]$$

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$\text{while } x \neq 0 \text{ do}$$

$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$

$$\Downarrow [cons']$$

$$\{((y * x) * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$

$$y := y * x; \ [ass]$$

$$\{(y * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$

$$x := x - 1; \ [ass]$$

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$

$$\text{od } [while_{pk}]$$

$$\{((y * x! = a! \ \wedge \ x \geq 0) \vee \ x < 0) \ \wedge \ \neg(x \neq 0)\}$$

$$\Downarrow 5x \ [cons'']$$

$$\{y = a!\}$$

Fakultätsprogramm: Lineare Beweisskizze (14)

Schließen der letzten Beweislücke in der zugrundeliegenden Theorie arithmetischer und Boolescher Ausdrücke:

$$\{true\}$$
$$\Downarrow [cons']$$
$$\{a \geq 0 \ \vee \ a < 0\}$$
$$\Downarrow [cons']$$
$$\{(1 * a! = a! \ \wedge \ a \geq 0) \ \vee \ a < 0\}$$
$$x := a; \ [ass]$$
$$\{(1 * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
$$y := 1; \ [ass]$$
$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
$$\text{while } x \neq 0 \text{ do}$$
$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$
$$\Downarrow [cons']$$
$$\{((y * x) * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$
$$y := y * x; \ [ass]$$
$$\{(y * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$
$$x := x - 1; \ [ass]$$
$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
$$\text{od } [while_{pk}]$$
$$\{((y * x! = a! \ \wedge \ x \geq 0) \vee \ x < 0) \ \wedge \ \neg(x \neq 0)\}$$
$$\Downarrow 5x \ [cons'']$$
$$\{y = a!\}$$

Gesamtskizze

$$\{true\}$$
$$\Downarrow [cons']$$
$$\{a \geq 0 \ \vee \ a < 0\}$$
$$\Downarrow [cons']$$
$$\{(1 * a! = a! \ \wedge \ a \geq 0) \ \vee \ a < 0\}$$
$$x := a; \ [ass]$$
$$\{(1 * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
$$y := 1; \ [ass]$$
$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
$$\text{while } x \neq 0 \text{ do}$$
$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$
$$\Downarrow [cons']$$
$$\{((y * x) * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$
$$y := y * x; \ [ass]$$
$$\{(y * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$
$$x := x - 1; \ [ass]$$
$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
$$\text{od } [while_{pk}]$$
$$\{((y * x! = a! \ \wedge \ x \geq 0) \vee \ x < 0) \ \wedge \ \neg(x \neq 0)\}$$
$$\Downarrow [cons'']$$
$$\{((y * x! = a! \ \wedge \ x \geq 0) \vee \ x < 0) \ \wedge \ x = 0\}$$
$$\Downarrow [cons'']$$
$$\{(y * x! = a! \ \wedge \ x \geq 0 \ \wedge \ x = 0) \ \vee \ (x < 0 \ \wedge \ x = 0)\}$$
$$\Downarrow [cons'']$$
$$\{(y * 0! = a! \ \wedge \ x = 0) \ \vee \ false\}$$
$$\Downarrow [cons'']$$
$$\{y * 1 = a! \ \wedge \ x = 0\}$$
$$\Downarrow [cons'']$$
$$\{y = a!\}$$

Zusammenfassung

Durch Konstruktion der linearen Beweisskizze haben wir gezeigt:

Die Hoaresche Zusicherung

$$\{true\}$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$\{y = a!\}$$

ist mit den Axiomen und Regeln von HK_{pk} ableitbar. Gemäß Korrektheitstheorem 4.5.1 ist die Zusicherung damit gültig im Sinn partieller Korrektheit und der Beweis von Lemma 4.6.1.1 somit erbracht.

Übungsaufgabe 4.6.3.1: Fakultätsprogramm

Zeige durch Konstruktion von

1. Ableitungsbaums
2. linearer Beweisskizze

dass (auch) die Hoaresche Zusicherung

$$\{a \geq 0\}$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$\{y = a!\}$$

gültig ist im Sinn partieller Korrektheit, d.h.:

$$\models_{pk} \{a \geq 0\}\ \pi\ \{y = a!\}$$

3. Lässt sich die Invariante I aus dem Beweis partieller Korrektheit zur Vorbedingung *true* für den zur Vorbedingung $a \geq 0$ zu einer Invariante I' abschwächen? Begründe die Antwort.

Übungsaufgabe 4.6.3.2: Divisionsprogramm

Beweise Lemma 4.6.1.2: Zeige durch Konstruktion einer linearen Beweisskizze, dass die Hoaresche Zusicherung

$$\{x \geq 0 \wedge y > 0\}$$

$$\pi \equiv q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q+1;\ r := r-y \text{ od}$$

$$\{x = q * y + r \wedge 0 \leq r < y\}$$

gültig ist im Sinn partieller Korrektheit, d.h.:

$$\models_{pk} \{x \geq 0 \wedge y > 0\}\ \pi\ \{x = q * y + r \wedge 0 \leq r < y\}$$

Übungsaufgabe 4.6.3.3: Divisionsprogramm (1)

Ist $x \geq 0 \wedge y > 0$ die schwächste liberale Vorbedingung für das Programm

$\pi \equiv q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$

zur ganzzahligen Division mit Rest und Nachbedingung

$$x = q * y + r \wedge 0 \leq r < y \quad ?$$

Falls nein, bestimme eine Vorbedingung *wlp* und beweise, dass *wlp* tatsächlich die gesuchte schwächste liberale Vorbedingung beschreibt, d.h. beweise:

$$wlp \iff wlp(\pi, x = q * y + r \wedge 0 \leq r < y) \quad (*)$$

Übungsaufgabe 4.6.3.3: Divisionsprogramm (2)

Zeige zum Beweis von $(*)$ insbesondere die partielle Korrektheit der Hoareschen Zusicherung

$$\{wlp\}\ \pi\ \{x = q * y + r \wedge 0 \leq r < y\}$$

d.h.:

$$\models_{pk} \{wlp\}\ \pi\ \{x = q * y + r \wedge 0 \leq r < y\}$$

durch Konstruktion von

1. Ableitungsbaum
2. linearer Beweisskizze

Welche Eigenschaften sind darüberhinaus zu zeigen, um $(*)$ und damit die Äquivalenz von *wlp* zur schwächsten liberalen Vorbedingung $wlp(\pi, x = q * y + r \wedge 0 \leq r < y)$ zu zeigen?

3. Beweise die zusätzlichen Eigenschaften.

Kapitel 4.7

Ableitungskalküle HK'_{tk}, HK_{tk} für totale Korrektheit

Die Hoare-Kalküle

...für partielle und totale Korrektheit für WHILE sind

- nahezu ident.

Einziger Unterschied: Die Regel zur Behandlung der while-Schleife

- $[\text{while}_{pk}]$

die beim Übergang von HK_{pk} zu HK_{tk} ersetzt werden muss durch eine terminierungssensitive Regel

- $[\text{while}_{tk}]$.

Hierfür gibt es verschiedene Möglichkeiten, die eine Abwägung treffen zwischen Einfachheit von

- Regel (Variante V1).
- Regelanwendung (Variante V2).

V1: Hoare-Kalkül HK_{tk} für totale Korrektheit

Variante 1: Regeleinfachheit vor Regelanwendungseinfachheit

$$[\text{while}'_{tk}] \quad \frac{I \Rightarrow t{\geq}0,\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]} \qquad (I \text{ Invariante})$$

mit

- t arithmetischer Ausdruck über ganzen Zahlen, sog. Terminierungsterm.
- w ganzzahlige 'frische' logische Variable, d.h. w kommt in I, b, π und t nicht frei vor.

Terminationsordnung ist: $(\mathbb{N}, \textbf{kleiner})$ (bzw. $(\mathbb{N}, <)$ bei überladener Verwendung des Symbols $<$)

Anmerkungen zur while-Regel $[\text{while}'_{tk}]$ (1)

$$[\text{while}'_{tk}] \quad \frac{I \Rightarrow t{\geq}0,\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]} \qquad (I\ \text{Invariante})$$

Informell:

Die linke Prämisse $I \Rightarrow t \geq 0$ von $[\text{while}'_{tk}]$ besagt:

- Vor Ausführung des Schleifenrumpfs (I ist wahr!), gilt:

$$t \in \mathbb{N}_0,\ t \geq 0$$

Das bedeutet, der Wert des Terminierungsterms t ist vor (und nach) jeder Ausführung des Schleifenrumpfs π Element der durch die Relation **kleiner** (bzw. überladen: $<$) Noethersch geordneten Menge $\mathbb{N}_0$.

Anmerkungen zur while-Regel $[\text{while}'_{tk}]$ (2)

Die rechte Prämisse $[I \wedge b \wedge t = w]\ \pi\ [I \wedge t < w]$ von $[\text{while}'_{tk}]$ besagt:

- Wenn t vor Ausführung des Schleifenrumpfs den Wert w hat, so hat t nach Ausführung des Schleifenrumpfs einen echt kleineren Wert, da w als 'frische' logische Variable in π nicht vorkommt und deshalb vor und nach Ausführung von π denselben Wert hat.

Zusammen mit der linken Prämisse folgt daraus, dass der Wert des Terminierungsterms t mit jeder Ausführung des Schleifenrumpfs π echt kleiner wird, d.h. bzgl. der Noetherschen Ordnung **kleiner** (bzw. überladen: $<$) von $\mathbb{N}_0$ echt abnimmt.

Da es in $\mathbb{N}_0$ keine unendlich absteigenden Ketten gibt, kann die linke Prämisse also nur endlich oft wahr sein, was nichts anderes als Terminierung der while-Schleife bedeutet.

V2: Hoare-Kalkül HK_{tk} für totale Korrektheit

Variante 2: Regelanwendungseinfachheit vor Regeleinfachheit

$$[\text{while}_{tk}] \quad \frac{I \wedge b \Rightarrow u[t/v],\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]} \qquad (I \text{ Invariante})$$

mit

- u Boolescher Ausdruck über der Variablen v.
- t arithmetischer Ausdruck über ganzen Zahlen, sog. Terminierungsterm.
- w ganzzahlige 'frische' logische Variable, d.h. w kommt in I, b, π und t nicht frei vor.
- $M =_{df} \{\sigma(v) \mid \sigma \in Ch(u)\}$ bzgl. $\sqsubset$ Noethersch geordnete Menge (oder: Noethersche (Halb-) Ordnung).

Terminationsordnung ist: $(M, \sqsubset)$

Anmerkungen zur while-Regel $[\text{while}_{tk}]$ (1)

$$[\text{while}_{tk}] \quad \frac{I \wedge b \Rightarrow u[t/v],\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]} \qquad (I\ \text{Invariante})$$

Informell:

Die linke Prämisse $I \wedge b \Rightarrow u[t/v]$ von $[\text{while}_{tk}]$ besagt:

- Vor jeder Ausführung des Schleifenrumpfs ($I \wedge b$ wahr!), gilt:
$$u[t/v] \Longleftrightarrow \textbf{wahr}$$

Zusammen mit der Definition von M folgt daraus, dass der Wert des Terminierungsterms t vor jeder Ausführung des Schleifenrumpfs π Element einer Noethersch geordneten Menge ist.

Anmerkungen zur while-Regel [while_{tk}] (2)

Die rechte Prämisse $[I \wedge b \wedge t = w]\ \pi\ [I \wedge t < w]$ von [while_{tk}] besagt:

- Wenn t vor Ausführung des Schleifenrumpfs den Wert w hat, so hat t nach Ausführung des Schleifenrumpfs einen echt kleineren Wert, da w als logische Variable in π nicht vorkommt und deshalb vor und nach Ausführung von π denselben Wert hat.

Zusammen mit der linken Prämisse folgt daraus, dass der Wert des Terminierungsterms t mit jeder Ausführung des Schleifenrumpfs echt kleiner wird, d.h. bzgl. der Noetherschen Ordnung von M echt abnimmt.

Da es in M keine unendlich absteigenden Ketten gibt, kann die linke Prämisse also nur endlich oft wahr sein, woraus die Terminierung der while-Schleife folgt.

V1: Hoare-Kalkül HK'_{tk} für totale Korrektheit

Seien p, q zwei logische Formeln oder Prädikate.

Axiome:

[skip] $\dfrac{}{[p]\ \mathit{skip}\ [p]}$

[ass] $\dfrac{}{[p[t/x]]\ x{:=}t\ [p]}$ (Rückwärtssubstitution, Rückwärtsregel)

Regeln:

[comp] $\dfrac{[p]\ \pi_1\ [r],\ [r]\ \pi_2\ [q]}{[p]\ \pi_1;\pi_2\ [q]}$

[ite] $\dfrac{[p \wedge b]\ \pi_1\ [q],\ [p \wedge \neg b]\ \pi_2\ [q]}{[p]\ \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi } [q]}$

[while$'_{tk}$] $\dfrac{I \Rightarrow t{\geq}0,\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while } b \text{ do } \pi \text{ od } [I \wedge \neg b]}$ (I Invariante)

[cons] $\dfrac{p \Rightarrow p_1,\ [p_1]\ \pi\ [q_1],\ q_1 \Rightarrow q}{[p]\ \pi\ [q]}$

Beachte die überladene Verwendung der eckigen Klammern in [ass].

V2: Hoare-Kalkül HK_{tk} für totale Korrektheit

Seien p, q zwei logische Formeln oder Prädikate.

Axiome:

[skip] $\dfrac{}{[p]\ \mathit{skip}\ [p]}$

[ass] $\dfrac{}{[p[t/x]]\ x{:=}t\ [p]}$ (Rückwärtssubstitution, Rückwärtsregel)

Regeln:

[comp] $\dfrac{[p]\ \pi_1\ [r],\ [r]\ \pi_2\ [q]}{[p]\ \pi_1;\pi_2\ [q]}$

[ite] $\dfrac{[p \wedge b]\ \pi_1\ [q],\ [p \wedge \neg b]\ \pi_2\ [q]}{[p]\ \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi } [q]}$

[while_{tk}] $\dfrac{I \wedge b \Rightarrow u[t/v],\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while } b \text{ do } \pi \text{ od } [I \wedge \neg b]}$ (I Invariante)

[cons] $\dfrac{p \Rightarrow p_1,\ [p_1]\ \pi\ [q_1],\ q_1 \Rightarrow q}{[p]\ \pi\ [q]}$

Beachte die überl. Verw. der eckigen Klammern in [ass], [while_{tk}].

Vergleich von $[\text{while}_{tk}]$ und $[\text{while}'_{tk}]$

...zentraler Unterschied:

- $[\text{while}_{tk}]$: Beliebige Noethersche Ordnung als Terminationsordnung zulässig: $(M, \sqsubset)$.
- $[\text{while}'_{tk}]$: Festlegung auf eine spezielle Noethersche Ordnung als Terminationsordnung, nämlich: $(\mathbb{N}_0, <)$.

Beachte: Oft erfordert die Rückspiegelung einer sich 'natürlich' anbietenden Noetherschen Terminationsordnung auf die spezielle Noethersche Ordnung $(\mathbb{N}_0, <)$ zusätzlichen Modellierungsaufwand.

In diesen Fällen ist $[\text{while}_{tk}]$ pragmatisch vorteilhaft im Vergleich zu $[\text{while}'_{tk}]$.

Hintergrund: Irreflexive partielle Ordnungen

Definition 4.7.1 (Irreflexive partielle Ordnung)

Sei P eine Menge und $\sqsubset$ eine irreflexive und transitive Relation auf P. Dann heißt das Paar $(P, \sqsubset)$ eine irreflexive partielle Ordnung. Gilt $p \sqsubset p'$, $p, p' \in P$, so heißt p kleiner als p' und p' größer als p.

Beispiele: $(\mathbb{Z}, <)$, $(\mathbb{Z}, >)$, $(\mathbb{N}, <)$, $(\mathbb{N}, >)$ sind irreflexive partielle Ordnungen (überladene Verwendung der Symbole $<$, $>$).

Hintergrund: Wohlfundierte Ordnungen

Definition 4.7.2 (Wohlfundierte Ordnung)

Sei $(P, \sqsubset)$ eine irreflexive partielle Ordnung und $W \subseteq P$ eine Teilmenge von P.

1. $\sqsubset$ heißt wohlfundiert auf W, wenn es keine unendlich absteigende Kette

$$\ldots \sqsubset w_2 \sqsubset w_1 \sqsubset w_0$$

   von Elementen $w_i \in W$ gibt.
2. Ist $\sqsubset$ wohlfundiert auf W, heißt das Paar $(W, \sqsubset)$ eine wohlfundierte Struktur (oder Noethersch geordnete Menge oder wohlfundierte oder Noethersche Ordnung).

Beispiele: $(\mathbb{N}, <)$ ist eine Noethersche Ordnung, nicht aber $(\mathbb{Z}, <)$, $(\mathbb{Z}, >)$ und $(\mathbb{N}, >)$.

Hintergrund: Konstruktion wohlfundierter Ord.

...aus gegebenen wohlfundierten Ordnungen:

Lemma 4.7.3

Seien $(W_1, \sqsubset_1)$ und $(W_2, \sqsubset_2)$ zwei wohlfundierte Ordnungen. Dann sind auch

1. $(W_1 \times W_2, \sqsubset_{com})$ mit komponentenweiser Ordnung definiert durch

$$(m_1, m_2) \sqsubset_{com} (n_1, n_2) \iff_{df} m_1 \sqsubset_1 n_1 \wedge m_2 \sqsubset_2 n_2$$

2. $(W_1 \times W_2, \sqsubset_{lex})$ mit lexikographischer Ordnung definiert durch

$$(m_1, m_2) \sqsubset_{lex} (n_1, n_2) \iff_{df} (m_1 \sqsubset_1 n_1) \vee (m_1 = n_1 \wedge m_2 \sqsubset_2 n_2)$$

wohlfundierte Ordnungen.

Vergleich von HK'_{tk}, HK_{tk} und HK_{pk}

...HK'_{tk}, HK_{tk} und HK_{pk} sind bis auf die Prämissen der Schleifenregeln ident:

- Totale Korrektheit: $[\text{while}'_{tk}]$, $[\text{while}_{tk}]$

$$[\text{while}'_{tk}] \quad \frac{I \Rightarrow t \geq 0,\ [I \wedge b \wedge t = w]\ \pi\ [I \wedge t < w]}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]} \qquad (I \text{ Invariante})$$

$$[\text{while}_{tk}] \quad \frac{I \wedge b \Rightarrow u[t/v],\ \{I \wedge b \wedge t = w\}\ \pi\ \{I \wedge t < w\}}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]} \qquad (I \text{ Invariante})$$

- Partielle Korrektheit: $[\text{while}_{pk}]$

$$[\text{while}_{pk}] \quad \frac{\{I \wedge b\}\ \pi\ \{I\}}{\{I\}\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ \{I \wedge \neg b\}} \qquad (I \text{ Invariante})$$

Abschließende beweistechnische Anmerkung

...'zerlegt' man die Prämissen von $[\text{while}'_{tk}]$ wie folgt:

$$[\text{while}''_{tk}] \quad \frac{\{I \wedge b\}\ \pi\ \{I\},\ I \Rightarrow t \geq 0,\ [I \wedge b \wedge t = w]\ \pi\ [t < w]}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]}\ (I \text{ Invariante})$$

wird deutlich: Der Beweis totaler Korrektheit einer Hoareschen Zusicherung besteht aus dem Beweis:

- partieller Korrektheit
- regulärer Terminierung des Programms.

Salopp: Totale Korrektheit "gleich"
Partielle Korrektheit "plus" Reguläre Terminierung

Diese Trennung kann im Beweis explizit vollzogen werden. Der Gesamtbeweis wird dadurch modular; der Terminationsbeweis ist zudem oft einfach.

Bemerkung: Die obige Zerlegung kann in gleicher Weise für die Schleifenregel $[\text{while}_{tk}]$ erfolgen.

Kapitel 4.8

Korrektheit und Vollständigkeit von HK'_{tk}, HK_{tk}

Korrektheit von HK'_{tk} und HK_{tk}

Sei π ein WHILE -Programm, p, q zwei logische Formeln oder Prädikate:

Theorem 4.8.1 (Korrektheit von HK'_{tk} und HK_{tk})

Die Ableitungkalküle HK'_{tk} und HK_{tk} sind korrekt, d.h. jede mit den Axiomen und Regeln von HK'_{tk} und HK_{tk} ableitbare Korrektheitsformel (in Zeichen: $\vdash_{HK'_{tk}/HK_{tk}} [p]\ \pi\ [q]$) ist gültig im Sinne totaler Korrektheit (in Zeichen: $\models_{tk} [p]\ \pi\ [q]$), d.h.:

$$\vdash_{HK'_{tk}/HK_{tk}} [p]\ \pi\ [q] \ \Rightarrow\ \models_{tk} [p]\ \pi\ [q]$$

Beweis durch strukturelle Induktion über den Aufbau des Ableitungsbaums der Korrektheitsformel $[p]\ \pi\ [q]$.

Vollständigkeit von HK'_{tk} und HK_{tk}

Sei π ein WHILE -Programm, p, q zwei Prädikate (extensionaler Ansatz):

Theorem 4.8.2 (Vollständigkeit von HK'_{tk} und HK_{tk})

Die Ableitungkalküle HK'_{tk} und HK_{tk} sind vollständig, d.h. jede im Sinn totaler Korrektheit gültige Korrektheitsformel (in Zeichen: $\models_{tk} [p]\ \pi\ [q]$) ist mit den Axiomen und Regeln von HK'_{tk} und HK_{tk} ableitbar (in Zeichen: $\vdash_{HK'_{tk}/HK_{tk}} [p]\ \pi\ [q]$), d.h.:

$$\models_{tk} [p]\ \pi\ [q] \ \Rightarrow\ \vdash_{HK'_{tk}/HK_{tk}} [p]\ \pi\ [q]$$

Beweis durch strukturelle Induktion über den Aufbau von π.

Kapitel 4.9

Totale Korrektheitsbeweise

Kapitel 4.9.1

Fakultät, ganzzahlige Division mit Rest

Die Programme

1) Fakultät

$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$

2) Ganzzahlige Division mit Rest

$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$

Die Zusicherungen totaler Korrektiheit

Lemma 4.9.1.1 (Fakultät)

Die Hoaresche Zusicherung

$$[a \geq 0]$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[y = a!]$$

ist gültig im Sinn totaler Korrektheit.

Lemma 4.9.1.2 (Ganzzahlige Division mit Rest)

Die Hoaresche Zusicherung

$$[x \geq 0 \wedge y > 0]$$

$$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$$

$$[x = q * y + r \wedge 0 \leq r < y]$$

ist gültig im Sinn totaler Korrektheit.

Beweisdarstellungen: Baum und lineare Skizze

Aufgabe: Beweis von Lemma 4.9.1.1 und 4.9.1.2.

...d.h., zeigen, dass die Hoareschen Tripel für die Berechnung der Fakultätsfunktion und der ganzzahligen Division mit Rest gültig sind im Sinn totaler Korrektheit.

Wir zeigen die Beweise in zwei notationellen Varianten. Als:

1. Ableitungsbaum (kanonische Variante) (Kap. 4.9.2).
2. lineare Beweisskizze (pragmatische Variante) (Kap. 4.9.3).

Kapitel 4.9.2

Ableitungsbäume

Übungsaufgabe 4.9.2.1: Fakultäts-, Div.-prog.

Beweise die Gültigkeit der Hoareschen Zusicherungen aus

- Lemma 4.9.1.1 (Fakultät)
- Lemma 4.9.1.2 (Ganzzahlige Division mit Rest)

mithilfe der Axiome und Regeln von

1. HK'_{tk}
2. HK_{tk}

durch Konstruktion entsprechender Ableitungsbäume.

Kapitel 4.9.3

Lineare Beweisskizzen

Fakultätsprogramm: Arbeitsplan

...Beweis von Lemma 4.9.1.1:

Wir zeigen durch Angabe einer linearen Beweisskizze, dass das Hoaresche Tripel

$$[a \geq 0]$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[y = a!]$$

gültig ist im Sinn totaler Korrektheit.

...die lineare Beweisskizze dafür enwickeln wir Schritt für Schritt!

Fakultätsprogramm: Lineare Beweisskizze (1)

Schritt 1: Wahl einer ausreichend starken Invariante und eines geeigneten Terminierungsterms.

"Träumen" (geeigneter) Invariante und Terminierungsterms, hier:

- $I \equiv y * x! = a! \wedge x \geq 0$ als Invariante
- $t \equiv x$ als Terminierungsterm
- $u \equiv v > 0$ als Boolescher Ausdruck über v

um die Regel $[\text{while}_{tk}]$ anwenden und den Beweis erfolgreich abschließen zu können.

Fakultätsprogramm: Lineare Beweisskizze (2)

...mit Wahl von $u \equiv v > 0$ und $t \equiv x$ gilt für:

- $u[t/v]$: $u[t/v] = (v > 0)[x/v] = x > 0$
- M:

$$
\begin{aligned}
M &= \{\sigma(v) \mid \sigma \in Ch(u)\} \\
&=_{df} \{\sigma(v) \mid \sigma \in Ch(v > 0)\} \\
&= \{\sigma(v) \mid \sigma \in \Sigma \wedge [\![v]\!]_A(\sigma) \textbf{ größer } [\![0]\!]_A(\sigma)\} \\
&= \{\sigma(v) \mid \sigma \in \Sigma \wedge \sigma(v) \textbf{ größer } \mathbf{0}\} \\
&= \mathbb{N}_1
\end{aligned}
$$

Damit gilt: M ist bzgl. der Relation **kleiner** (bzw. überladen: $<$) auf $\mathbb{N}_1$ Noethersch geordnet, d.h.:

$$(M, \sqsubseteq) = (\mathbb{N}_1, \textbf{kleiner}) \text{ (bzw. überladen: } (\mathbb{N}_1, <))$$

ist Noethersche Ordnung.

Fakultätsprogramm: Lineare Beweisskizze (3)

...mit Wahl von $I \equiv y * x! = a! \wedge x \geq 0$ und der Schleifenbedingung $b \equiv x \neq 0$ aus π gilt:

Für alle Zustände $\sigma \in \Sigma$, in denen

- I erfüllt ist, gilt: $\mathbf{0} \leq \sigma(x) \in \mathbb{N}_0$
- I und b erfüllt sind, gilt: $\mathbf{1} \leq \sigma(x) \in M \ (= \mathbb{N}_1)$

Anders ausgedrückt:

- $\forall\, \sigma \in Ch(I).\ \sigma(x) \geq \mathbf{0}$
  d.h.: $\{\sigma(x) \,|\, \sigma \in Ch(I)\} = \mathbb{N}_0$
- $\forall\, \sigma \in Ch(I \wedge b).\ \sigma(x) \geq \mathbf{1}$
  d.h.: $\{\sigma(x) \,|\, \sigma \in Ch(I \wedge b)\} = \mathbb{N}_1$

Insbesondere:

- $\{\sigma(x) \,|\, \sigma \in Ch(I)\}$, $\{\sigma(x) \,|\, \sigma \in Ch(I \wedge b)\}$ Noethersch geordnet.
- $\forall\, \sigma \in Ch(I) \cup Ch(I \wedge b).\ \sigma(x)$ Element Noethersch geordneter Menge.

Fakultätsprogramm: Lineare Beweisskizze (4)

Schritt 2: Behandlung des Rumpfs der while-Schleife.

Die Herleitung von

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$
$$y := y * x;$$
$$x := x - 1;$$
$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

erlaubt mithilfe der $[\text{while}_{tk}]$-Regel den Übergang zu:

$$[y * x! = a! \wedge x \geq 0]$$
$$\text{while } x \neq 0 \text{ do}$$
$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$
$$y := y * x;$$
$$x := x - 1;$$
$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$
$$\text{od } [\text{while}_{tk}]$$
$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

Fakultätsprogramm: Lineare Beweisskizze (5)

Behandlung des Rumpfs der while-Schleife im Detail:

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$y := y * x;$$

$$x := x - 1;$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

Fakultätsprogramm: Lineare Beweisskizze (6)

Wegen Rückwärtszuweisungsregel wird der Rumpf der while-Schleife von hinten nach vorne bearbeitet:

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$y := y * x;$$
$$[y * (x-1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$
$$x := x - 1; \text{ [ass]}$$
$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

Fakultätsprogramm: Lineare Beweisskizze (7)

Nochmalige Anwendung der [ass]-Regel liefert:

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$
$$y := y * x; \text{ [ass]}$$
$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$
$$x := x - 1; \text{ [ass]}$$
$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

...wobei noch eine Beweislücke verbleibt!

Fakultätsprogramm: Lineare Beweisskizze (8)

Schließen der Beweislücke in der zugrundeliegenden Theorie arithmetischer und Boolescher Ausdrücke:

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$

$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

Fakultätsprogramm: Lineare Beweisskizze (9)

Anwendung der [while$_{tk}$]-Regel liefert nun wie gewünscht:

$$[y * x! = a! \wedge x \geq 0]$$

$$\text{while } x \neq 0 \text{ do}$$

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$

$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; \ [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; \ [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

$$\text{od } [\text{while}_{tk}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

Fakultätsprogramm: Lineare Beweisskizze (10)

Schritt 3: Zur gewünschten Nachbedingung verbleibt ebenfalls eine Beweislücke:

$$[y * x! = a! \wedge x \geq 0]$$

while $x \neq 0$ do

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \ \wedge \ x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; \ [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; \ [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

od $[\text{while}_{tk}]$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$[y = a!]$$

Fakultätsprogramm: Lineare Beweisskizze (11)

Schließen der Beweislücke in der zugrundeliegenden Theorie arithmetischer und Boolescher Ausdrücke:

$$[y * x! = a! \wedge x \geq 0]$$

while $x \neq 0$ do

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; \ [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; \ [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

od $[\text{while}_{tk}]$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow [\text{cons}'']$$

$$[y * x! = a! \wedge x \geq 0 \wedge x = 0]$$

$$\Downarrow [\text{cons}'']$$

$$[y * 0! = a!]$$

$$\Downarrow [\text{cons}'']$$

$$[y = a!]$$

Fakultätsprogramm: Lineare Beweisskizze (12)

Aus Platzgründen etwas verkürzt dargestellt:

$$[y * x! = a! \wedge x \geq 0]$$

while $x \neq 0$ do

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; \ [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; \ [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

od $[\text{while}_{tk}]$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow 3x \ [\text{cons}'']$$

$$[y = a!]$$

Fakultätsprogramm: Lineare Beweisskizze (13)

Schritt 4: Es verbleibt, die Beweislücke zur gewünschten Vorbedingung zu schließen:

$$[a \geq 0]$$

$$x := a;$$

$$y := 1;$$

$$[y * x! = a! \wedge x \geq 0]$$

while $x \neq 0$ do

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; \ [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; \ [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

od $[\text{while}_{tk}]$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow 3x \ [\text{cons}'']$$

$$[y = a!]$$

Fakultätsprogramm: Lineare Beweisskizze (14)

Einmalige Anwendung der [ass]-Regel liefert:

$$[a \geq 0]$$

$$x := a;$$

$$[1 * x! = a! \wedge x \geq 0]$$

$$y := 1; \text{ [ass]}$$

$$[y * x! = a! \wedge x \geq 0]$$

$$\text{while } x \neq 0 \text{ do}$$

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; \text{ [ass]}$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; \text{ [ass]}$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

$$\text{od } [\text{while}_{tk}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow 3x \text{ [cons}'']$$

$$[y = a!]$$

Fakultätsprogramm: Lineare Beweisskizze (15)

Nochmalige Anwendung der [ass]-Regel liefert:

$$[a \geq 0]$$

$$[1 * a! = a! \wedge a \geq 0]$$

$$x := a; \text{ [ass]}$$

$$[1 * x! = a! \wedge x \geq 0]$$

$$y := 1; \text{ [ass]}$$

$$[y * x! = a! \wedge x \geq 0]$$

$$\text{while } x \neq 0 \text{ do}$$

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; \text{ [ass]}$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; \text{ [ass]}$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

$$\text{od } [\text{while}_{tk}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow 3x\ [\text{cons}'']$$

$$[y = a!]$$

Fakultätsprogramm: Lineare Beweisskizze (16)

Schließen der letzten Beweislücke in der zugrundeliegenden Theorie arithmetischer und Boolescher Ausdrücke:

$$[a \geq 0]$$

$$\Downarrow [\text{cons}']$$

$$[1 * a! = a! \wedge a \geq 0]$$

$$x := a; [\text{ass}]$$

$$[1 * x! = a! \wedge x \geq 0]$$

$$y := 1; [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0]$$

$$\text{while } x \neq 0 \text{ do}$$

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

$$\text{od } [\text{while}_{tk}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow 3x [\text{cons}'']$$

$$[y = a!]$$

Gesamtskizze

$$[a \geq 0]$$

$$\Downarrow [\text{cons}']$$

$$[1 * a! = a! \wedge a \geq 0]$$

$$x := a; [\text{ass}]$$

$$[1 * x! = a! \wedge x \geq 0]$$

$$y := 1; [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0]$$

while $x \neq 0$ do

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

$$\Downarrow [\text{cons}']$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$y := y * x; [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

$$x := x - 1; [\text{ass}]$$

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

od $[\text{while}_{tk}]$

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow [\text{cons}'']$$

$$[y * x! = a! \wedge x \geq 0 \wedge x = 0]$$

$$\Downarrow [\text{cons}'']$$

$$[y * 0! = a!]$$

$$\Downarrow [\text{cons}'']$$

$$[y = a!]$$

Zusammenfassung

Durch Konstruktion der linearen Beweisskizze haben wir gezeigt:

Die Hoaresche Zusicherung

$$[a \geq 0]$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[y = a!]$$

ist mit den Axiomen und Regeln von HK_{tk} ableitbar. Gemäß Korrektheitstheorem 4.8.1 ist die Zusicherung damit gültig im Sinn totaler Korrektheit und der Beweis von Lemma 4.9.1.1 somit erbracht.

Übungsaufgabe 4.9.3.1: Divisionsprogramm

Beweise Lemma 4.9.1.2: Zeige durch Konstruktion einer linearen Beweisskizze, dass die Hoaresche Zusicherung

$$[x \geq 0 \wedge y > 0]$$
$$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$$
$$[x = q * y + r \wedge 0 \leq r < y]$$

gültig ist im Sinn totaler Korrektheit, d.h.:

$$\models_{tk} [x \geq 0 \wedge y > 0]\ \pi\ [x = q * y + r \wedge 0 \leq r < y]$$

Kapitel 4.10

Ansätze, Werkzeuge für (semi-) automatische axiomatische Programmverifikation

Ansätze, Werkzeuge

...zur (semi-) automatischen Programmverifikation im Hoare-schen Stil.

Unter anderem:

- Theorema, RISC, JKU Linz.
- KeY-Hoare, KIT Karlsruhe, Chalmers University of Technology, TU Darmstadt.
- ...

Theorema-Projekt (1)

...www.theorema.org:

"The Theorema project aims at extending current computer algebra systems by facilities for supporting mathematical proving. The present early-prototype version of the Theorema software system is implemented in Mathematica. The system consists of a general higher-order predicate logic prover and a collection of special provers that call each other depending on the particular proof situations. The individual provers imitate the proof style of human mathematicians and produce human-readable proofs in natural language presented in nested cells. The special provers are intimately connected with the functors that build up the various mathematical domains."

(Exzerpt von http://www.theorema.org)

Theorema-Projekt (2)

"The long-term goal of the project is to produce a complete system which supports the mathematician in creating interactive text- books, i.e. books containing, besides the ordinary passive text, active text representing algorithms in executable format, as well as proofs which can be studied at various levels of detail, and whose routine parts can be automatically generated. This system will provide a uniform (logic and software) framework in which a working mathematician, without leaving the system, can get computer-support while looping through all phases of the mathematical problem solving cycle. [...]"

(Exzerpt von `http://www.theorema.org`)

KeY-Projekt (1)

...www.key-project.org:

Integrated Deductive Software Design

"The KeY System is a formal software development tool that aims to integrate design, implementation, formal specification, and formal verification of object-oriented software as seamlessly as possible. At the core of the system is a novel theorem prover for the first-order Dynamic Logic for Java with a user-friendly graphical interface.

The project was started in November 1998 at the University of Karlsruhe. It is now a joint project of Karlsruhe Institute of Technology and Chalmers University of Technology, Gothenburg, and TU Darmstadt.

The KeY tool is available for down-load. [...]"

(Exzerpt von http://www.key-project.org)

KeY-Projekt (2)

KeY-Hoare (www.key-project.org/download/hoare) unterstützt

- partielle Korrektheitsbeweise
- totale Korrektheitsbeweise und Ausführungszeitkorrektheitsbeweise (Versionen ab 0.1.6)
- ganzzahlige und Boolesche Felder (Versionen ab 0.1.7)

Nützliche Anleitung:

- Reiner Hähnle, Richard Bubel. *A Hoare-Style Calculus with Explicit State Updates*. Handout in a course on Program Verification at the Department of Computer Science at the Chalmers University of Technology on the Hoare Calculus and the usage of the tool KeY-Hoare, 19 pages. http://i12www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf

Kapitel 4.11

Historische Meilensteine der Programmverifikation

Meilensteine der Programmverifikation (1)

Frühe Anfänge

1949 Turings Vision: Korrekte Programme
Beispiel Fakultätsfunktion: Zusicherungen und Terminierungsfunktion

Axiomatische Methode

1967 Floyd: Flussdiagramme
Hoare: while-Programme

Erweiterung der axiomatischen Methode

1971 Hoare: Rekursive Prozeduren
1976/77 Owicki & Gries, Lamport: Parallele Programme
1980/81 Apt, Francez & de Roever, Levin & Gries: Verteilte Programme
1991 de Boer: Parallele, objektorientierte Programme
1977 Pnueli: Temporale Logik für Programme
1979 Clarke: Grenzen der axiomatischen Methode

Meilensteine der Programmverifikation (2)

Automatisierung der Verifikation

1981/82 Emerson & Clarke, Quielle & Sifakis: Modellprüfung
1977 Cousot & Cousot: Abstrakte Interpretation
1979 Deduktion: Interaktive Theorembeweiser
1967 Automatische Terminierungsbeweise

Entwicklung korrekter Programme

1976 Dijkstra: Kalkül der schwächsten Vorbedingung
1997 Meyer: Design-by-Contract
1969 Büchi & Landweber: Automatenbasierte Systeme

Quelle: Ernst-Rüdiger Olderog, Reinhard Wilhelm. Turing und die Verifikation. Informatik Spektrum 35(4):271-279, 2012.

Kapitel 4.12

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (1)

- Krzysztof R. Apt. *Ten Years of Hoare's Logic: A Survey – Part 1*. ACM Transactions on Programming Languages and Systems 3:431-483, 1981.
- Krzysztof R. Apt. *Ten Years of Hoare's Logic: A Survey – Part II: Nondeterminism*. Theoretical Computer Science 28(1-2):83-109, 1984.
- Krzysztof R. Apt, Ernst-Rüdiger Olderog. *Programmverifikation – Sequentielle, parallele und verteilte Programme*. Springer-V., 1994.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (2)

Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Springer-V., 3. Auflage, 2009. (Chapter 3, While Programs; Chapter 3.3, Verification; Chapter 3.4, Proof Outlines – Partial Correctness, Total Correctness; Chapter 3.5, Completeness)

Bernhard Beckert, Reiner Hähnle, Peter H. Schmitt (Hrsg.). *Verification of Object-Oriented Software: The KeYApproach*. LNCS 4334, Springer-V., 2007.

Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 2. Auflage, Springer-V., 2001. (Chapter 9, Programs: Semantics and Verification)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (3)


Ernie Cohen, Dexter Kozen. *A Note on the Complexity of Propositional Hoare Logic*. ACM Transactions on Computational Logic 1(1):171-174, 2000.


Stephen A. Cook. *Soundness and Completeness of an Axiom System for Program Verification*. SIAM Journal on Computing 7(1):70-90, 1978.


Jaco W. De Backer. *Mathematical Theory of Program Correctness*. Prentice-Hall, 1980.


Edmund M. Clarke. *Programming Language Constructs for which it is Impossible to Obtain Good Hoare Axiom Systems*. Journal of the ACM 26(1):129-147, 1979.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (4)


Edmund M. Clarke, Stephen M. German, Joseph Y. Halpern. *Effective Axiomatizations of Hoare Logics*. Journal of the ACM 30(1):612-636, 1983.


Martin Davis. *Computability and Unsolvability*. Dover Publications, 1982.


Robert W. Floyd. *Assigning Meaning to Programs*. In Proceedings of Symposium on Applied Mathematics, Mathematical Aspects of Computer Science, American Mathematical Society, New York, 19:19-32, 1967.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (5)


Emily P. Friedman. *Relationships between Monadic Recursion Schemes and Deterministic Context-free Languages*. In IEEE Conference Record of the 15th Annual Symposium on Switching and Automata Theory (SWAT'74), 43-51, 1974.


Emily P. Friedman. *Equivalence Problems for Deterministic Context-free Languages and Monadic Recursion Schemes*. Journal of Computer and System Sciences 14(3):344-359, 1977.


Stephen J. Garland, David C. Luckham. *Program Schemes, Recursion Schemes, and Formal Languages*. Journal of Computer and System Sciences 7(2):119-160, 1973.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (6)

Seymour Ginsburg, Sheila Greibach. *Deterministic Context Free Languages*. Information and Control 9(6):620-648, 1966.

Reiner Hähnle, Richard Bubel. *A Hoare-Style Calculus with Explicit State Updates*. Handout in the course Program Verification at the Department of Computer Science at the Chalmers University of Technology, 19 Seiten. `http://i12www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf`

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (7)


Charles A.R. Hoare. *An Axiomatic Basis for Computer Programming*. Communications of the ACM 12(10):576-580, 583, 1969.


Charles A.R. Hoare. *The Emperor's Old Clothes*. Communications of the ACM 24(2):75-83, 1981. DOI: 10.1145/358549.358561


Charles A.R. Hoare. *The Ideal of Program Correctness*. The Computer Journal 50(3):254-260, 2007.


Charles A.R. Hoare. *Retrospective: An Axiomatic Basis for Computer Programming*. Communications of the ACM 52(10):30-32, 2009. DOI: 10.1145/1562764.1562779

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (8)

Tudor Jebelean, Laura Kovács, Nikolaj Popov. *Experimental Program Verification in the Theorema System*. In Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004), 92-99, 2004. www.risc.jku.at/publications/download/risc_2243/KoPoJeb.pdf

Laura Kovács, Tudor Jebelean. *Practical Aspects of Imperative Program Verification using Theorema*. In Proceedings of the 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2003), 317-320, 2003. www.risc.jku.at/publications/download/risc_464/synasc03.pdf

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (9)


Laura Kovács, Tudor Jebelean. *Generation of Invariants in Theorema*. In Proceedings of the 10th International Symposium of Mathematics and its Applications, 407-415, 2003. `www.risc.jku.at/publications/download/risc_2053/2003-11-06-A.pdf`


Dexter Kozen, Jerzy Tiuryn. *On the Completeness of Propositional Hoare Logic*. Information Sciences 139(3-4):187-195, 2001.


Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009. (Chapter 1, Introduction: What do we want to know about the Program?, Chapter 2, How to prove a Program Correct: Programs without Loops; Chapter 3, How to prove a Program Correct: Iterative Programs)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (10)


Jacques Loeckx, Kurt Sieber. *The Foundations of Program Verification*. Wiley, 1984.


Konstantinos Mamouras. *On the Hoare Theory of Monadic Recursion Schemes*. In Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (CSL-LICS'14), Article 69, 69.1-69.10, 2014.


Konstantinos Mamouras. *The Hoare Logic of Deterministic and Nondeterministic Monadic Recursion Schemes*. ACM Transactions on Computational Logic 17(2):13.1-13.30, 2016.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (11)

Robert Lover. *Elementary Logic for Software Development*. Springer-V., 2008. (Chapter 19, Program Correctness Proofs; Chapter 19.3, Proofs using Floyd's Method of Invariant Assertions; Chapter 20.2.1, Floyd-Hoare Logic)

Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019. (Chapter 3, Program Verification)

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 6, Axiomatic Program Verification)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (12)

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 9, Axiomatic Program Verification; Chapter 10, More on Axiomatic Program Verification)

David von Oheimb. *Hoare Logic for Java in Isabelle/HOL*. Concurrency and Computation: Practice and Experience 13(13):1173-1214, 2001.

Ernst-Rüdiger Olderog. *Correctness of Programs with Pascal-like Procedures without Global Variables*. Theoretical Computer Science 30(1):49-90, 1984.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (13)


Ernst-Rüdiger Olderog, Bernhard Steffen. *Formale Semantik und Programmverifikation*. In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 145-166, 2006.


Ernst-Rüdiger Olderog, Reinhard Wilhelm. *Turing und die Verifikation*. Informatik Spektrum 35(4):271-279, 2012.


Vaughan R. Pratt. *Semantical Considerations of Floyd-Hoare Logic*. In Proceedings of the 17th IEEE Annual Symposium on Foundations of Computer Science (FOCS'76), 109-121, 1976.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (14)

Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993. (Chapter 6, The axiomatic semantics of IMP; Chapter 7, Completeness of the Hoare rules)

Kapitel 5

Axiomatische Zeitaufwandsanalyse

Kapitel 5.1

Motivation

Übergang

...von axiomatischer Programmverifikation zu axiomatischer Programmanalyse.

...am Beispiel

– axiomatischer asymptotischer Zeitaufwandsanalyse

nach:

– Kapitel 6.5, Assertions for Execution Time.
Hanne Riis Nielson, Flemming Nielson. Semantics with Applications – A Formal Introduction. Wiley, 1992.
– Kapitel 10.2, Assertions for Execution Time.
Hanne Riis Nielson, Flemming Nielson. Semantics with Applications – An Appetizer. Springer-V., 2007.

Ausführungsaufwandsanalyse

Hintergrund: In vielen Anwendungsbereichen sind

- (zumindest) weiche Aussagen über den Ausführungszeitaufwand von Programmen erforderlich, z.B. Antwortzeiten von Buchungsportalen.
- (sogar) harte Aussagen über die Ausführungs- bzw. Antwortzeiten von Programmen unverzichtbar, in jedem Fall für sicherheitskritische (Echtzeit-) Anwendungen (sog. Schlechtester-Fall-Ausführungszeitanalyse (engl. worst-case execution time (WCET) analysis).

Der Nachweis totaler Korrektheit mittels axiomatischer Programmverifikation garantiert zwar

- Terminierung eines Programms

sagt jedoch nichts über den tatsächlichen Ressourcen-, insbesondere Laufzeitbedarf aus.

In diesem Kapitel

...Erweiterung und Adaptierung des Hoare-Kalküls für totale Korrektheit, um beweisbare Aussagen über den

- asymptotischen Zeitaufwand

zu ermöglichen.

Grundlage: Einführung einer ausführungszeitbewussten

- (Nichtstandard-) Semantik für WHILE

aufbauend auf einer ausführungszeitbewussten

- (Nichtstandard-) Semantik für
  - Konstantensymbole (Numerale, Wahrheitswertkonst.)
  - Variablen (-zugriffe)
  - Ausdrücke.

Grundidee: Aufwandszuordnung zu Ausdrücken

- Numerale, Wahrheitswertkonstanten

  ...konstanter Auswertungsaufwand/-zeit, d.h. von Größenordnung $\mathcal{O}(\mathbf{1})$.

- Variablen (-zugriffe)

  ...konstanter Zugriffsaufwand/-zeit (lesen, schreiben), d.h. von Größenordnung $\mathcal{O}(\mathbf{1})$.

- Nichtelementare Ausdrücke

  ...linearer Auswertungsaufwand/-zeit in der Zahl $\mathbf{n}$ der Operatoren und Relatoren von Ausdrücken, d.h. von Größenordnung $\mathcal{O}(\mathbf{n})$.

Grundidee: Aufwandszuord. zu Anweisungen (1)

- Leere Anweisung

  ...konstanter Ausführungsaufwand/-zeit, d.h. von Grössenordnung $\mathcal{O}(\mathbf{1})$.

- Zuweisung

  ...Ausführungsaufwand/-zeit als Auswertungsaufwand/-zeit des rechtsseitigen Zuweisungsausdrucks.

- (Sequentielle) Komposition

  ...Ausführungsaufwand/-zeit als Summe der Ausführungsaufwände/- zeiten der Komponenten.

Grundidee: Aufwandszuord. zu Anweisungen (2)

- Fallunterscheidung

  ...Ausführungsaufwand/-zeit als Summe des Auswertungsaufwands/-zeit von Bedingung und Maximum der Ausführungsaufwände/-zeiten der beiden Fallunterscheidungszweige.

- while-Schleife

  ...Ausführungsaufwand/-zeit als Summe der wiederholten Auswertungsaufwände/-zeiten von Abbruchbedingung und Ausführungsaufwänden/-zeiten des Schleifenrumpfs.

...Verfeinerungen und präzisere Zuordnungen sind möglich.

Formalisierung und Umsetzung

...der Grundidee in drei Schritten:

1. Ausführungszeitbewusste (abstrakte) Ausdruckssemantik:
   ...Einführung einer abstrakten Semantik, die die Auswertungszeit arithmetischer und Boolescher Ausdrücke beschreibt (Kap. 5.2).
2. Ausführungszeitbewusste (abstrakte) Programmsemantik:
   ...Erweiterung und Adaption der natürlichen Semantik von WHILE zu einer ausführungszeitbewussten abstrakten Programmsemantik (Kap. 5.3).
3. Ableitungskalkül für totale Korrektheit mit asymptotischen Ausführungsaufwandsaussagen:
   Erweiterung und Adaption des Ableitungskalküls für totale Korrektheit zur Ableitung von Aussagen zum asymptotischen Ausführungsaufwand von Programmen (Kap. 5.4).

Kapitel 5.2

Zeitaufwandsbewusste Ausdruckssemantik

Zeitaufwandsbewusste Ausdruckssemantik

...Einführung und Definition zeitaufwandsbewusster (abstrakter) Semantiken für arithmetische und Boolesche Ausdrücke:

- $[\![.]\!]_{ZA} : \mathbf{Aexpr} \to \mathbb{N}$
- $[\![.]\!]_{ZB} : \mathbf{Bexpr} \to \mathbb{N}$

die arithmetischen und Booleschen Ausdrücken als Bedeutung ihren Auswertungszeitaufwand zuordnen (in Zeiteinheiten einer hier nicht näher spezifizierten abstrakten Maschine *AM*).

Intuitiv: $[\![a]\!]_{ZA}$, $a \in \mathbf{Aexpr}$, und $[\![b]\!]_{ZB}$, $b \in \mathbf{Bexpr}$, liefern die Anzahl der Zeiteinheiten, die *AM* zur Auswertung von a bzw. b benötigt.

Aufwandsbewusste Semantik arith. Ausdrücke

$[\![.]\!]_{ZA} : \mathbf{Aexpr} \to \mathbb{N}$ induktiv definiert durch:

$$
\begin{aligned}
[\![n]\!]_{ZA} &=_{df} \mathbf{1} \\
[\![x]\!]_{ZA} &=_{df} \mathbf{1} \\
[\![a_1 + a_2]\!]_{ZA} &=_{df} [\![a_1]\!]_{ZA} + [\![a_2]\!]_{ZA} + \mathbf{1} \\
[\![a_1 * a_2]\!]_{ZA} &=_{df} [\![a_1]\!]_{ZA} + [\![a_2]\!]_{ZA} + \mathbf{1} \\
[\![a_1 - a_2]\!]_{ZA} &=_{df} [\![a_1]\!]_{ZA} + [\![a_2]\!]_{ZA} + \mathbf{1} \\
[\![a_1 / a_2]\!]_{ZA} &=_{df} [\![a_1]\!]_{ZA} + [\![a_2]\!]_{ZA} + \mathbf{1}
\end{aligned}
$$

...andere Operatoren analog, ggf. mit operationsspezifischen Auswertungszeiten.

Beachte: $[\![.]\!]_{ZA}$ ist zustandsunabhängig.

Aufwandsbewusste Semantik Bool. Ausdrücke

$[\![.]\!]_{ZB} : \mathbf{Bexpr} \to \mathbb{N}$ induktiv definiert durch:

$$
\begin{array}{rcl}
[\![\mathit{true}]\!]_{ZB} & =_{df} & \mathbf{1} \\
[\![\mathit{false}]\!]_{ZB} & =_{df} & \mathbf{1} \\
[\![a_1 = a_2]\!]_{ZB} & =_{df} & [\![a_1]\!]_{ZA} + [\![a_2]\!]_{ZA} + \mathbf{1} \\
[\![a_1 < a_2]\!]_{ZB} & =_{df} & [\![a_1]\!]_{ZA} + [\![a_2]\!]_{ZA} + \mathbf{1} \\
\ldots & \ldots & \ldots \\
[\![\neg b]\!]_{ZB} & =_{df} & [\![b]\!]_{ZB} + \mathbf{1} \\
[\![b_1 \wedge b_2]\!]_{ZB} & =_{df} & [\![b_1]\!]_{ZB} + [\![b_2]\!]_{ZB} + \mathbf{1} \\
[\![b_1 \vee b_2]\!]_{ZB} & =_{df} & [\![b_1]\!]_{ZB} + [\![b_2]\!]_{ZB} + \mathbf{1}
\end{array}
$$

...andere Relatoren (z.B. $\leq$, ...) analog, ggf. mit operationsspezifischen Auswertungszeiten.

Beachte: $[\![.]\!]_{ZB}$ ist zustandsunabhängig.

Kapitel 5.3

Zeitaufwandsbewusste natürliche Semantik

Erweiterung und Anpassung

...der

- natürlichen Semantik $[\![\ ]\!]_{ns}$

von WHILE zur Bestimmung des

- Ausführungszeitaufwands von WHILE -Programmen

zusätzlich zu ihrer üblichen Bedeutung.

Methode: Ersetzen der Transitionen der Form

$$\langle \pi, \sigma \rangle \to \sigma'$$

der N-Semantik von WHILE durch Transitionen der NZ-Semantik der Form

$$\langle \pi, \sigma \rangle \to^t \sigma'$$

mit der Bedeutung, dass ein Programm π angesetzt auf σ nach t Zeiteinheiten in σ' terminiert.

NZS-Regelwerk von WHILE

Axiome:

$$[\text{skip}_{nzs}] \quad \frac{}{\langle \mathit{skip}, \sigma \rangle \rightarrow^{1} \sigma}$$

$$[\text{ass}_{nzs}] \quad \frac{}{\langle x := t, \sigma \rangle \rightarrow^{[\![t]\!]_{ZA}+1} \sigma[[\![t]\!]_A(\sigma)/x]}$$

$$[\text{while}^{ff}_{nzs}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow^{[\![b]\!]_{ZB}+3} \sigma} \qquad [\![b]\!]_B(\sigma) = \textbf{falsch}$$

Regeln:

$$[\text{while}^{tt}_{nzs}] \quad \frac{\langle \pi, \sigma \rangle \rightarrow^{t} \sigma', \langle \text{while } b \text{ do } \pi \text{ od}, \sigma' \rangle \rightarrow^{t'} \sigma''}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow^{[\![b]\!]_{ZB}+t+t'+2} \sigma''} \qquad [\![b]\!]_B(\sigma) = \textbf{wahr}$$

$$[\text{if}^{tt}_{nzs}] \quad \frac{\langle \pi_1, \sigma \rangle \rightarrow^{t} \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \rightarrow^{[\![b]\!]_{ZB}+t+1} \sigma'} \qquad [\![b]\!]_B(\sigma) = \textbf{wahr}$$

$$[\text{if}^{ff}_{nzs}] \quad \frac{\langle \pi_2, \sigma \rangle \rightarrow^{t} \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \rightarrow^{[\![b]\!]_{ZB}+t+1} \sigma'} \qquad [\![b]\!]_B(\sigma) = \textbf{falsch}$$

$$[\text{comp}_{nzs}] \quad \frac{\langle \pi_1, \sigma \rangle \rightarrow^{t_1} \sigma', \langle \pi_2, \sigma' \rangle \rightarrow^{t_2} \sigma''}{\langle \pi_1; \pi_2, \sigma \rangle \rightarrow^{t_1+t_2} \sigma''}$$

Beispiel: Illustration der NZ-Semantik (1)

...anhand des (kanonischen) Fakultätsprogramms.

Sei $\sigma \in \Sigma$, $\sigma(x) = \mathbf{3}$. Dann gilt: Das Programm

$$y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

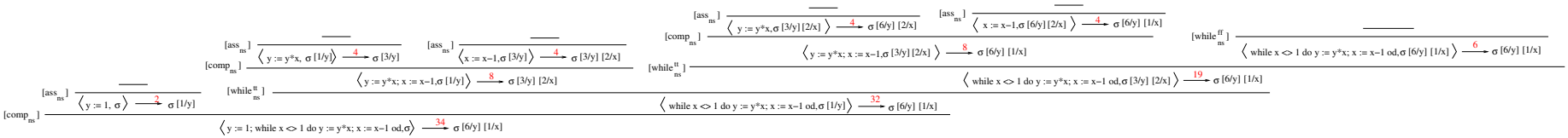
angesetzt auf σ terminiert in 34 Zeiteinheiten im Zustand

$$\sigma[\mathbf{6}/y][\mathbf{1}/x]$$

Das heißt:

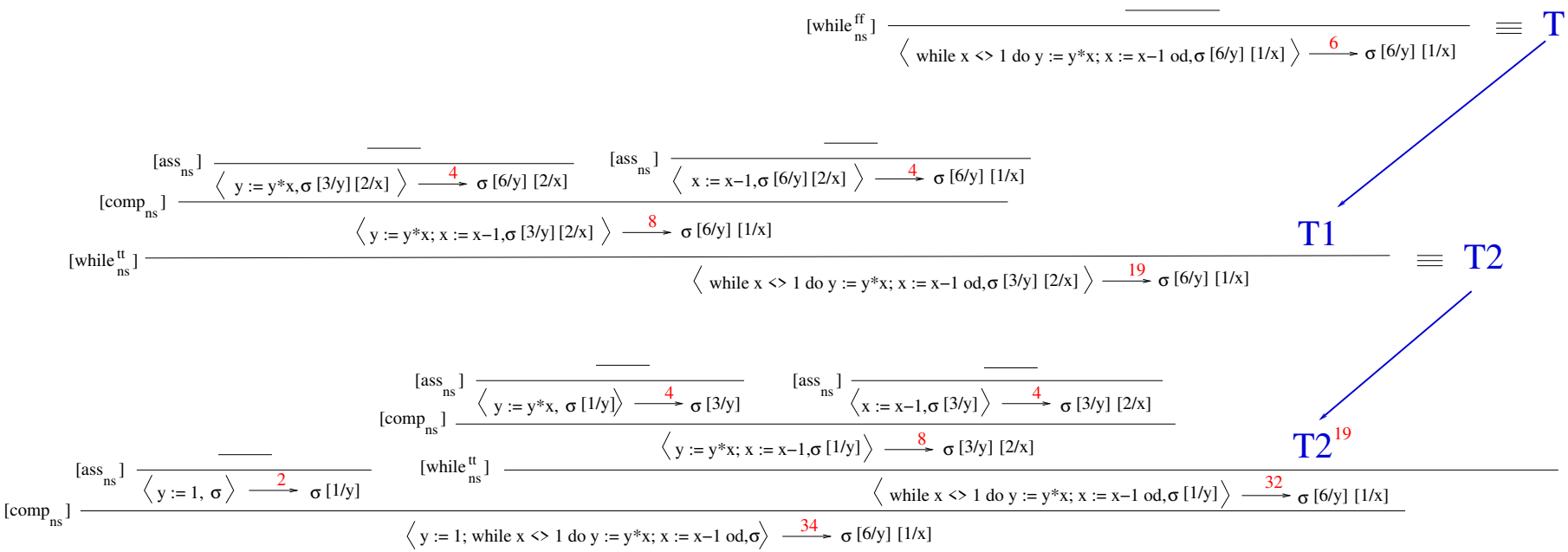
$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma\rangle \to^{34} \sigma[\mathbf{6}/y][\mathbf{1}/x]$$

Zugehöriger Ableitungsbaum:



Beispiel: Illustration der NZ-Semantik (2)

...der gleiche Ableitungsbaum in 'etwas' größerer Darstellung durch Einführung der benannten Teilbäume T1, T2:



Kapitel 5.4

Zeitaufwandsbewusste axiomatische Semantik

Erweiterung und Anpassung

...des

- Ableitungskalküls HK_{tk}

für totale Korrektheit von Programmen um den Aspekt ihres asymptotischen Ausführungzeitsaufwands.

Methode: Übergang von zeitunbewussten Korrektheitstripeln der Form:

$$[p]\ \pi\ [q]$$

zu zeitbewussten Korrektheitstripeln der Form:

$$[p]\ \pi\ [e \Downarrow q]$$

mit:

- π WHILE -Programm.
- p, q logische Formeln oder Prädikate als Vor- und Nachbedingung (wie bisher!).
- $e \in \mathbf{Aexp}$ arithmetischer Ausdruck als Aufwandsabschätzung, d.h. Aufwand von π ist von Größenordnung $\mathcal{O}(e)$.

Semantik aufwandsbewusster Korrektheitstripel

Sei π ein WHILE -Programm, p, q zwei logische Formeln oder Prädikate, e ein arithmetischer Ausdruck.

Definition 5.1.1 (Gültigkeit aufw.b. Korrektheitstr.)

Das zeitaufwandsbewusste Korrektheitstripel

$$[p]\ \pi\ [e \Downarrow q]$$

ist gültig im Sinn zeitaufwandsbewusster totaler Korrektheit (in Zeichen: $\models_{ztk} [p]\ \pi\ [e \Downarrow q]$) gdw. für jeden Zustand $\sigma \in \Sigma$ gilt:

Ist die Vorbedingung p in σ erfüllt, dann terminiert die zugehörige Berechnung von π angesetzt auf σ regulär in einem Endzustand σ' und die Nachbedingung q ist in σ' erfüllt und der benötigte Ausführungszeitaufwand von π ist durch e beschränkt, d.h. von Größenordnung $\mathcal{O}(e)$.

Charakterisierung aufw.b. totaler Korrektheit

Lemma 5.1.2 (Charakterisierung)

Das zeitaufwandsbewusste Korrektheitstripel

$$[p]\ \pi\ [e \Downarrow q]$$

ist gültig (in Zeichen: $\models_{ztk} [p]\ \pi\ [e \Downarrow q]$) gdw es existiert ein $\mathbf{k} \in \mathbb{N}$, so dass für alle Zustände $\sigma \in \Sigma$ gilt:

Ist die Vorbedingung p in σ erfüllt, dann gibt es einen Zustand $\sigma' \in \Sigma$ und eine natürliche Zahl t, so dass gilt:

- π angesetzt auf σ terminiert in t Zeiteinheiten regulär in σ', d.h. $\langle \pi, \sigma \rangle \rightarrow^t \sigma'$.
- Nachbedingung q ist erfüllt in σ'.
- t ist von Größenordnung $\mathcal{O}(e)$, d.h. $t \leq \mathbf{k} * [\![e]\!]_A(\sigma)$ (In anderen Worten: t und $[\![e]\!]_A(\sigma)$ unterscheiden sich nur durch einen konstanten Faktor $\mathbf{k}$).

Beachte

...dass der Ausdruck e zur Größenordnungszeitaufwandsabschätzung im Anfangszustand σ ausgewertet wird, nicht im Endzustand σ':

- $t \leq \mathbf{k} * [\![e]\!]_A(\sigma)$

Diesem sinnvollen Umstand (Übungsaufgabe: Warum?) ist geschuldet, dass die Regeln

- $[while_e]$ und $[comp_e]$

des Hoare-artigen Zeitaufwandsabschätzungskalküls AK_{ztk} komplizierter ausfallen als möglicherweise zunächst vermutet.

Aufwandsabschätzungskalkül AK_{ztk} f. WHILE

Axiome:

$[skip_e]$ $\dfrac{}{\{p\}\ skip\ \{\mathbf{1} \Downarrow p\}}$

$[ass_e]$ $\dfrac{}{\{p[t\backslash x]\}\ x{:=}t\ \{\mathbf{1} \Downarrow p\}}$

Regeln:

$[comp_e]$ $\dfrac{[p \wedge e_2' {=} u]\ \pi_1\ [e_1 \Downarrow r \wedge e_2 {\leq} u],\ [r]\ \pi_2\ [e_2 \Downarrow q]}{[p]\ \pi_1;\pi_2\ [e_1 {+} e_2' \Downarrow q]}$

wobei u frische logische Variable.

$[ite_e]$ $\dfrac{[p \wedge b]\ \pi_1\ [e \Downarrow q],\ [p \wedge \neg b]\ \pi_2\ [e \Downarrow q]}{[p]\ \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi } [e \Downarrow q]}$

$[while_e]$ $\dfrac{[p(z{+}1) \wedge e' {=} u]\ \pi\ [e_1 \Downarrow p(z) \wedge e {\leq} u]}{[\exists z.\ p(z)]\ \text{while } b \text{ do } \pi \text{ od } [e \Downarrow p(0)]}$

wobei: $p(z+1) \Rightarrow (b \wedge e \geq e_1 + e')$,

$p(0) \Rightarrow (\neg\, b \wedge 1 \leq e)$,

$z \in \mathbb{N}_0$, u frische logische Variable.

$[cons_e]$ $\dfrac{p \Rightarrow p_1 \quad [p_1]\ \pi\ [e' \Downarrow q_1] \quad q_1 \Rightarrow q}{[p]\ \pi\ [e \Downarrow q]}$ $\quad \exists\, \mathbf{k} \in \mathbb{N}.\ e' \leq \mathbf{k} * e$

Anmerkungen zur AK_{ztk}-Regel [$comp_e$]

Die Anwendung der [$comp_e$]-Regel verlangt, dass es

- Ableitungen dafür gibt, dass e_1, e_2 die Größenordnung der Zahl der Ausführungsschritte von π_1, π_2 beschreiben.
- e_1 macht dies für π_1 relativ zum Anfangszustand von π_1 aus; e_2 für π_2 relativ zum Anfangszustand von π_2.
- Die Größenordnung der Zahl der Ausführungsschritte der sequentiellen Komposition $\pi_1; \pi_2$ ist deshalb nicht einfach summativ durch $e_1 + e_2$ beschrieben.
- Vielmehr muss für e_2 ein Ausdruck e_2' gefunden werden, so dass e_2 ausgewertet im Anfangszustand von π_2 durch die Größenordnung von e_2' ausgewertet im Anfangszustand von π_1 beschränkt ist.
- Dies wird durch die Erweiterung der Vor- und Nachbedingung von π_1 unter Verwendung der frischen logischen Variable u erreicht.

Anmerkungen zur AK_{ztk}-Regel [$while_e$]

Die Anwendung der [$while_e$]-Regel verlangt, dass es

- eine Ableitung bzw. Nachweis dafür gibt, dass e_1 die Größenordnung der Zahl der Ausführungsschritte des Schleifenrumpfs, e die der gesamten Schleife beschreiben.
- Ähnlich der [$comp_e$]-Regel ist die Größenordnung der Zahl der Ausführungsschritte der gesamten Schleife nicht direkt durch den summativen Ausdruck $e_1 + e$ beschrieben, da e_1 auf den Zustand vor Ausführung des Schleifenrumpfs Bezug nimmt, e hingegen auf den Zustand nach seiner einmaligen Ausführung.
- Deshalb muss ein Ausdruck e' gefunden werden, der ausgewertet vor Ausführung des Schleifenrumpfs Ausdruck e ausgewertet nach seiner Ausführung beschränkt.
- Das erfordert, dass e die Ungleichung $e \geq e_1 + e'$ erfüllt, da e die Ausführungszeit der while-Schleife unabhängig von der Anzahl ihrer Wiederholungen beschränken muss.

Anmerkungen zur AK_{ztk}-Regel $[\text{cons}_e]$

Pragmatisch ist es vorteilhaft, zusätzlich zur Konsequenzregel

$$[\text{cons}_e] \quad \frac{p \Rightarrow p_1 \quad [p_1]\ \pi\ [e' \Downarrow q_1] \quad q_1 \Rightarrow q}{[p]\ \pi\ [e \Downarrow q]} \qquad \exists\, \mathbf{k} \in \mathbb{N}.\ e' \leq \mathbf{k} * e$$

auch folgende Spezialisierungen der Konsequenzregel zum Beweiskalkül hinzuzunehmen:

$$[\text{cons}'_e] \quad \frac{p \Rightarrow p_1 \quad [p_1]\ \pi\ [e' \Downarrow q]}{[p]\ \pi\ [e \Downarrow q]} \qquad \exists\, \mathbf{k} \in \mathbb{N}.\ e' \leq \mathbf{k} * e$$

$$[\text{cons}''_e] \quad \frac{[p]\ \pi\ [e' \Downarrow q_1] \quad q_1 \Rightarrow q}{[p]\ \pi\ [e \Downarrow q]} \qquad \exists\, \mathbf{k} \in \mathbb{N}.\ e' \leq \mathbf{k} * e$$

In der Folge gehen wir davon aus, dass AK_{ztk} neben $[\text{cons}_e]$ auch die Konsequenzregeln $[\text{cons}'_e]$ und $[\text{cons}''_e]$ enthält.

Beispiel 5.1.3: Fakultätsprogramm

Das zeitaufwandsbewusste Korrektheitstripel

$$[a = 3]$$
$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$[1 \Downarrow \mathit{true}]$$

ist gültig im Sinn zeitaufwandsbewusster totaler Korrektheit und beschreibt, dass die Zahl der Ausführungsschritte des Fakultätsprogramms angesetzt auf einen Zustand σ mit $\sigma(a) = \mathbf{3}$ von der Größenordnung $\mathcal{O}(\mathbf{1})$ ist, also durch eine Konstante beschränkt ist.

Beispiel 5.1.4: Fakultätsprogramm

Das zeitaufwandsbewusste Korrektheitstripel

$$[a \geq 0]$$
$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$[a \Downarrow \mathit{true}]$$

ist gültig im Sinn zeitaufwandsbewusster totaler Korrektheit und beschreibt, dass die Zahl der Ausführungsschritte des Fakultätsprogramms angesetzt auf einen Zustand σ mit $\sigma(a) > \mathbf{0}$ von der Größenordnung $\mathcal{O}(a)$ ist, also linear in der Größe von a und damit des Anfangswerts von x ist.

Kapitel 5.5

Zeitaufwandsbewusste totale Korrektheitsbeweise

Das Fakultätsprogramm in variierter Form

...Beweis von Terminierung und Zeitaufwandsabschätzung:

Lemma 5.5.1 (Fakultät)

1. Terminierung: Das aufwandsbewusste Korrektheitstripel

$$[x = 3]$$

$$y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[1 \Downarrow \mathit{true}]$$

   ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

2. Aufwandsabschätzung: Das aufwandsbewusste Korrektheitstripel

$$[x > 0]$$

$$y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[x \Downarrow \mathit{true}]$$

   ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

Lemma 5.5.1(2): Lineare Beweisskizze (1)

$[x > 0]$

$\ldots$

$y := 1;$

$\ldots$

$[\exists\, z \in \mathbb{N}_0.\ (x > 0 \wedge x = z + 1)]$ $(\equiv\ [\exists\, z \in \mathbb{N}_0.\ \mathit{INV}(z)])$

while $x \neq 1$ do

$[(x > 0 \wedge x = z + 1) \wedge x - 1 = u_1]$ $(\equiv\ [\mathit{INV}(z+1) \wedge x - 1 = u_1])$

$[(x > 0 \wedge x = (z + 1) + 1) \wedge x - 1 = u_1]$

$[((x > 0 \wedge x = (z + 1) + 1) \wedge x - 1 = u_1) \wedge 1 = u_2]$

$\Downarrow [\mathrm{cons}'_e]$

$[((x - 1 > 0 \wedge x - 1 = z + 1) \wedge x - 1 \leq u_1) \wedge 1 \leq u_2]$

$y := y * x;\ [\mathrm{ass}_e],\ [\mathrm{comp}_e]$

$[1 \Downarrow ((x - 1 > 0 \wedge x - 1 = z + 1) \wedge x - 1 \leq u_1) \wedge 1 \leq u_2]$

$[(x - 1 > 0 \wedge x - 1 = z + 1) \wedge x - 1 \leq u_1]$

$x := x - 1;\ [\mathrm{ass}_e]$

$[1 \Downarrow (x > 0 \wedge x = z + 1) \wedge x \leq u_1]$

$[1 + 1 \Downarrow (x > 0 \wedge x = z + 1) \wedge x \leq u_1]$

$\Downarrow [\mathrm{cons}''_e]$

$[1 \Downarrow (x > 0 \wedge x = z + 1) \wedge x \leq u_1]$ $(\equiv\ [1 \Downarrow\ \mathit{INV}(z) \wedge x \leq u_1])$

$(x > 0 \wedge x = (z + 1) + 1) \Rightarrow \neg(x = 1) \wedge x \geq 1 + (x - 1)$ $(\equiv \mathit{INV}(z+1) \Rightarrow \neg(x = 1) \wedge x \geq 1 + (x - 1))$

$(x > 0 \wedge x = 0 + 1) \Rightarrow \neg(\neg(x = 1)) \wedge 1 \leq x$ $(\equiv\ \mathit{INV}(0) \Rightarrow \neg(\neg(x = 1)) \wedge 1 \leq x)$

od $[\mathrm{while}_e]$

$[x \Downarrow (x > 0 \wedge x = 0 + 1)]$ $(\equiv\ [x \Downarrow\ \mathit{INV}(0)])$

$\ldots$

$[x \Downarrow \mathit{true}]$

$\mathit{INV}(z) = x > 0 \wedge x = z + 1,\ z \in \mathbb{N}_0$ (d.h. $\forall\, \sigma \in \Sigma.\ \mathit{INV}(z)(\sigma) = \sigma(x) > 0 \wedge \sigma(x) = z + 1$)

Lemma 5.5.1(2): Lineare Beweisskizze (2)

$$[x > 0]$$

$$[x > 0 \wedge 1 = u3]$$

$$\Downarrow [\mathrm{cons}'_e]$$

$$[\exists\, z \in \mathbb{N}_0.\ (x > 0 \wedge x = z + 1) \wedge 1 \leq u_3]$$

$$y := 1;\ [\mathrm{ass}_e],\ [\mathrm{comp}_e]$$

$$[1 \Downarrow \exists\, z \in \mathbb{N}_0.\ (x > 0 \wedge x = z + 1) \wedge 1 \leq u_3]$$

$$[\exists\, z \in \mathbb{N}_0.\ (x > 0 \wedge x = z + 1)] \quad (\equiv\ [\exists\, z \in \mathbb{N}_0.\ \mathit{INV}(z)])$$

while $x \neq 1$ do

$$y := y * x;$$

$$x := x - 1;$$

od $[\mathrm{while}_e]$

$$[x \Downarrow (x > 0 \wedge x = 0 + 1)] \quad (\equiv\ [x \Downarrow \mathit{INV}(0)])$$

$$[1 + x \Downarrow x > 0 \wedge x = z + 1]$$

$$\Downarrow [\mathrm{cons}''_e] \quad (\text{da } (x > 0 \Rightarrow 1 + x \leq 2 * x) \wedge ((x > 0 \wedge x = z + 1) \Rightarrow \mathit{true}))$$

$$[x \Downarrow \mathit{true}]$$

$\mathit{INV}(z) = x > 0 \wedge x = z + 1,\ z \in \mathbb{N}_0$ (d.h. $\forall\, \sigma \in \Sigma.\ \mathit{INV}(z)(\sigma) = \sigma(x) > 0 \wedge \sigma(x) = z + 1$)

Übungsaufgabe 5.5.2: Fakultätsprogramm

Beweise Lemma 5.5.3 (Terminierung, Aufwandsabschätzung):

Lemma 5.5.3 (Fakultät)

1. Terminierung: Das aufwandsbewusste Korrektheitstripel

$$[a = 3]$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[1 \Downarrow \mathit{true}]$$

ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

2. Aufwandsabschätzung: Das aufwandsbewusste Korrektheitstripel

$$[a \geq 0]$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[a \Downarrow \mathit{true}]$$

ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

Übungsaufgabe 5.5.4: Fakultätsprogramm

Beweise Lemma 5.5.5 (Terminierung/Aufwandsabschätzung plus funktionale Korrektheit):

Lemma 5.5.5 (Fakultät)

1. Terminierung + funktionale Korrektheit: Das aufwandsbewusste Korrektheitstripel

$$[a = 3]$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[1 \Downarrow\ y = 6 = a!]$$

ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

2. Aufwandsabschätzung + funktionale Korrektheit: Das aufwandsbewusste Korrektheitstripel

$$[a \geq 0]$$

$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$

$$[a \Downarrow\ y = a!]$$

ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

Übungsaufgabe 5.5.6: Divisionsprogramm

Beweise Lemma 5.5.7 (Terminierung, Aufwandsabschätzung):

Lemma 5.5.7 (Ganzzahlige Division mit Rest)

1. Terminierung: Das aufwandsbewusste Korrektheitstripel

$$[x = 17 \wedge y = 4]$$

$$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$$

$$[1 \Downarrow \mathit{true}]$$

ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

2. Aufwandsabschätzung: Das aufwandsbewusste Korrektheitstripel

$$[x \geq 0 \wedge y > 0]$$

$$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$$

$$[x - y \Downarrow \mathit{true}]$$

ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

Übungsaufgabe 5.5.8: Divisionsprogramm

Beweise Lemma 5.5.9 (Terminierung/Aufwandsabschätzung plus funktionale Korrektheit):

Lemma 5.5.9 (Ganzzahlige Division mit Rest)

1. Terminierung + funktionale Korrektheit: Das aufwandsbewusste Korrektheitstripel

$$[x = 17 \wedge y = 4]$$

$$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$$

$$[1 \Downarrow x = q * y + r \wedge 0 \leq r < y \wedge q = 4 \wedge r = 1]$$

ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

2. Aufwandsabschätzung + funktionale Korrektheit: Das aufwandsbewusste Korrektheitstripel

$$[x \geq 0 \wedge y > 0]$$

$$q := 0;\ r := x;\ \text{while } r \geq y \text{ do } q := q + 1;\ r := r - y \text{ od}$$

$$[x - y \Downarrow x = q * y + r \wedge 0 \leq r < y\}$$

ist gültig im Sinn aufwandsbewusster totaler Korrektheit.

Kapitel 5.6

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (1)

Axiomatische asymptotische Aufwandsabschätzungsanalyse


Hanne Riis Nielson. *Hoare Logic's for Run-time Analysis of Programs*. PhD thesis, Edinburgh University, UK, 1984.


Hanne Riis Nielson. *A Hoare-like Proof System for Run-Time Analysis of Programs*. Science of Computer Programming 9(2):107-136, 1987.


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 6.5, Assertions for Execution Time)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 10.2, Assertions for Execution Time)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (2)

Schlechtester-Fall-Ausführungszeitanalyse: Überblicksarbeiten

Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström. *The Worst-case Execution Time Problem – Overview of Methods and Survey of Tools*. ACM Transactions on Embedded Computing Systems 7(3):36.1-53, 2008.

Reinhard Wilhelm. *Real Time spent on Real Time*. Communications of the ACM 63(10):54-60, 2020.

...the story of the development of a sound, static method for worst-case execution-time analysis.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (3)

Schlechtester-Fall-Ausführungszeitanalyse: Ausgewählte Arbeiten und Werkzeuge

aiT Worst-Case Execution Time Analyzers. Website: `http://www.absint.com/ait`, 2016. [Online; accessed 1-August-2016]

Philip Axer, Rolf Ernst, Heiko Falk, Alain Girault, Daniel Grund, Nan Guan, Bengt Jonsson, Peter Marwedel, Jan Reineke, Christine Rochange, Maurice Sebastian, Reinhard von Hanxleden, Reinhard Wilhelm, Wang Yi. *Building Timing Predictable Embedded Systems*. ACM Transactions on Embedded Computing Systems 13(4):82, 2014.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (4)


Clément Ballabriga, Hugues Cassé, Christine Rochange, Pascal Sainrat. *OTAWA: An Open Toolbox for Adaptive WCET Analysis*. In Proceedings SEUS 2010, Springer-V., 35-46, 2010.


Raimund Kirner, Jens Knoop, Adrian Prantl, Markus Schordan, Albrecht Kadlec. *Beyond Loop Bounds: Comparing Annotation Languages for Worst-Case Execution Time Analysis*. Journal of Software and Systems Modeling 10(3):411-437, Springer-V., 2011.


Armelle Bonenfant, Hugues Cassé, Marianne De Michiel, Jens Knoop, Laura Kovács, Jakob Zwirchmayr. *FFX: A Portable WCET Annotation Language*. In Proceedings of the 20th International Conference on Real-Time and Network Systems (RTNS 2012), ACM, 91-100, 2012.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (5)


Marvin Damschen, Lars Bauer, Jörg Henkel. *Timing Analysis of Tasks on Runtime Reconfigurable Processors*. In IEEE Transactions on Very Large Scale Integration Systems 25(1):294-307 2017.


Stephen A. Edwards, Edward A. Lee. *The Case for the Precision-timed (PRET) Machine*. In Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC'07), 264-265, 2007.


Jan Gustafsson. *Usability Aspects of WCET Analysis*. In Proceedings of the 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2008), 346-352, 2008.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (6)

Jan Gustafsson, Adam Betts, Andreas Ermedahl, Björn Lisper. *The Mälardalen WCET Benchmarks: Past, Present, and Future*. In Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010), 136-146, 2010.

Thomas Leveque, Etienne Borde, Amine Marref, Jan Carlson. *Hierarchical Composition of Parametric WCET in a Component Based Approach*. In Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2011), 261-268, 2011.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (7)


Yau-Tsun Steven Li, Sharad Malik. *Performance Analysis of Embedded Software using Implicit Path Enumeration*. ACM SIGPLAN Notices 30(11):88-98, 1995.


Björn Lisper, Andreas Ermedahl, Dietmar Schreiner, Jens Knoop, Peter Gliwa. *Practical Experiences of Applying Source-level WCET Flow Analysis to Industrial Code*. Journal of Software Tools for Technology Transfer (STTT) 15(1):53-63, Springer-V., 2013.


Greger Ottosson, Mikael Sjödin. *Worst-Case Execution Time Analysis for Modern Hardware Architectures*. In Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems (LCT-RTS'97), 1997.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (8)

Peter Puschner, Raimund Kirner, Robert G. Pettit. *Towards Composable Timing for Real-Time Programs*. Software Technologies for Future Dependable Distributed Systems, 1-5, 2009.

Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, Gernot Gebhard. *The T-CREST Approach of Compiler and WCET-Analysis Integration*. In Proceedings of the 9th International Workshop on Software Technologies for Future Embedded and Ubiquitious Systems (SEUS 2013), 33-40, 2013.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (9)

Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, Bernd Becker. *A Definition and Classification of Timing Anomalies*. In Proceedings of the 6th International Workshop on Worst-Case Execution Time Analysis (WCET 2006), 2006.

Henrik Theiling. *ILP-based Interprocedural Path Analysis*. In Proceedings of the International Workshop on Embedded Software (EMSOFT 2002), Springer-V., LNCS 2491, 349-363, 2002.

Lothar Thiele, Reinhard Wilhelm. *Design for Timing Predictability*. Real-Time Systems 28(2-3):157-177, 2004.

Teil IV

Analyse

Kapitel 6

Programmanalyse

Kapitel 6.1

Motivation, Problem

Motivation, Problem

...in Kapitel 2 und 3 haben wir uns mit verschiedenen Methoden zur Festlegung einer

- konkreten Semantik $[\![\ ]\!]_{\text{While}}$ für die Sprache While

und darauf aufbauend in Kapitel 4 und 5 mit

- axiomatischer Verifikation

zum Beweis von Eigenschaften von While-Programmen relativ zur konkreten Semantik $[\![\ ]\!]_{\text{While}}$ beschäftigt.

In den folgenden Kapiteln

...werden wir uns mit Methoden zur Festlegung verschiedener

- abstrakter Semantiken $[\![\ ]\!]_{absSem}$ für While

beschäftigen und darauf aufbauend mit

- Analyseverfahren

zum Beweis von Eigenschaften eines Programms relativ zu einer abstrakten Semantik $[\![\ ]\!]_{absSem}$, deren Ergebnisse bzgl. der konkreten Semantik $[\![\ ]\!]_{\text{While}}$ von While korrekt sein müssen.

Dabei gilt:

'Interessante' Eigenschaften von Programmen sind fast immer

- unentscheidbar

bzgl. der konkreten wie (auch vieler) abstrakter Programmsemantiken.

Glücklicherweise: Einige interessante Eigenschaften sind

- entscheidbar
- nützlich auch bei unvollständiger Berechenbarkeit
- ermöglichen
  - manuelle/semiautomatische Programmverifikations-
  - semiautomatische/vollautomatische Programmanalyse-

  Verfahren.

Kapitel 6.2

Ausblick

Ausblick mit Lichtblick(en)

...die Entwicklung von Analyse- und Verifikationsverfahren relativ zu einer abstrakten Programmsemantik kann sich zunutze machen, die fast immer konfliktären Ziele von

- Performanz/Effizienz/Skalierbarkeit
- Akkuratheit/Vollständigkeit

gegeneinander abzuwiegen und abzutauschen, unter Umständen sogar gegen

- Korrektheit

solange die erzielten Resultate noch nützlich sind.

Mit der Aufgabe, Verfahren und Methoden zu entwickeln, die im Spannungsdreieck von

- Vollständigkeit, Nützlichkeit, Performanz/Skalierbarkeit

eine gute Balance bewahren, beginnt Informatik.

Ein weiterer Lichtblick

...'interessante' Eigenschaften mögen zwar

- nicht generell
- nicht generell effizient
- nicht generell auch praktisch effizient

entscheidbar sein, aber all dies mag gelten für eingeschränkte Programmklassen, die ihrerseits noch hinreichend (praxis-) relevant und deshalb interessant sind; oft z.B.:

- schleifenfreie oder/und rekursionsfreie Programme
- parallelitätsfreie Programme
- prozedur-, methodenfreie Programme
- Formale prozedur-, methodenaufruffreie Programme
- ...

Auch das Erkennen und Ausnutzen solcher Spezialfälle ist Informatik.

Analyse- und Verifikationsverfahren

...zur Programmanalyse gibt es (deshalb) in verschiedensten Zugängen und Ausprägungen:

- Datenflussanalyse (Kap. 8)
- Reverse Datenflussanalyse (Kap. 10)
- Parallele Datenflussanalyse (Kap. 12)
- Abstrakte Interpretation (Kap. 18)
- Modellprüfung (Kap. 19)
- Symbolische Analyse
- Konkolische Analyse
- ...

von denen wir beginnend mit der Theorie der Datenflussanalyse die farblich hervorgehobenen in den folgenden Kapiteln genauer untersuchen und...

Anwendungen, Wechselbeziehungen, Nützlichk.

...hinsichtlich ihrer Nützlichkeit exemplarisch auch am Beispiel von Anwendungen wie:

- Elimination unnötiger Anweisungen in Programmen (Kap. 15, Kap. 16)
- Ersetzung von Ausdrücken durch ihre Werte (Kap. 17)

und im Hinblick auf Gemeinsamkeiten, Unterschiede, Verbindungen und Wechselbeziehungen zu- und miteinander verglei-chen werden:

- Abstrakte Interpretation und Datenflussanalyse (Kap. 18)
- Modellprüfung und Datenflussanalyse (Kap. 19)
- Modellprüfung und Abstrakte Interpretation (Kap. 20)

Kapitel 6.3

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (1)

Lehrbuchdarstellungen

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley, 2. Auflage, 2007. (Kapitel 1.2, The Structure of a Compiler; Kapitel 1.4, The Science of Building a Compiler; Kapitel 1.4.2, The Science of Code Optimization; Kapitel 9.1, The Principal Sources of Program Optimization)

Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004. (Anhang B.3.1, Graphical Intermediate Representations)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (2)


Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.


Uday P. Khedker, Amitabha Sanyal, Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press, 2009. (Kapitel 3, Theoretical Abstractions in Data Flow Analysis; Kapitel 4, General Data Flow Frameworks; Kapitel 5, Complexity of Iterative Data Flow Analysis)


Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009. (Kapitel 7, What can one tell about a Program without its Execution: Static Analysis)


Robert Morgan. *Building an Optimizing Compiler*. Digital Press, 1998. (Kapitel 2.3, Building the Flow Graph; Kapitel 4.7, Structure of Program Flow Graph)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (3)


Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997. (Kapitel 7, Control-Flow Analysis)


Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019. (Chapter 4, Program Analysis)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Kapitel 5, Static Program Analysis)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Kapitel 7, Program Analysis; Kapitel 8, More on Program Analysis; Anhang B, Implementation of Program Analysis)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (4)

Grundlegende, wegweisende Arbeiten

Frances E. Allen, John A. Cocke. *A Program Data Flow Analysis Procedure*. Communications of the ACM 19(3):137-147, 1976.

Dhananjay M. Dhamdhere, Barry K. Rosen, F. Kenneth Zadeck. *How to Analyze Large Programs Efficiently and Informatively*. In Proceedings of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation (PLDI'92), ACM SIGPLAN Notices 27(7):212-223, 1992.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (5)


Susan Horwitz, Alan J. Demers, Tim Teitelbaum. *An Efficient General Iterative Algorithm for Dataflow Analysis*. Acta Informatica 24(6):679-694, 1987.


John B. Kam, Jeffrey D. Ullman. *Global Data Flow Analysis and Iterative Algorithms*. Journal of the ACM 23:158-171, 1976.


John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.


Gary A. Kildall. *A Unified Approach to Global Program Optimization*. In Conference Record of the 1st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'73), 194-206, 1973.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (6)

Rahmenwerke, Werkzeugkisten

Marion Klein, Jens Knoop, Dirk Koschützki, Bernhard Steffen. *DFA&OPT-METAFrame: A Toolkit for Program Analysis and Optimization*. In Proceedings of the 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96), Springer-V., LNCS 1055, 422-426, 1996.

Jens Knoop. *From DFA-Frameworks to DFA-Generators: A Unifying Multiparadigm Approach*. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Springer-V., LNCS 1579, 360-374, 1999.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (7)


Thomas J. Marlowe, Barbara G. Ryder. *Properties of Data Flow Frameworks*. Acta Informatica 28(2):121-163, 1990.


Florian Martin. *PAG - An Efficient Program Analyzer Generator*. Journal of Software Tools for Technology Transfer 2(1):46-67, 1998.


Stephen P. Masticola, Thomas J. Marlowe, Barbara G. Ryder. *Lattice Frameworks for Multisource and Bidirectional Data Flow Problems*. ACM Transactions on Programming Languages and Systems (TOPLAS) 17(5):777-803, 1995.


Flemming Nielson. *Semantics-directed Program Analysis: A Tool-maker's Perspective*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 2-21, 1996.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (8)


Christian Fecht, Helmut Seidl. *Propagating Differences: An Efficient New Fixpoint Algorithm for Distributive Constraint Systems*. In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 90-104, 1998.


Christian Fecht, Helmut Seidl. *A Faster Solver for General Systems of Equations*. Science of Computer Programming 35(2):137-161, 1999.


Bernhard Steffen, Andreas Claßen, Marion Klein, Jens Knoop, Tiziana Margaria. *The Fixpoint Analysis Machine*. In Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95), Springer-V., LNCS 962, 72-87, 1995.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (9)

Flussgraph-Pragmatik

Larry Carter, Jeanne Ferrante, Clark Thomborson. *Folklore Confirmed: Reducible Flow Graphs are Exponentially Larger*. In Conference Record of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2003), 106-114, 2003.

Jens Knoop, Dirk Koschützki, Bernhard Steffen. *Basic-block Graphs: Living Dinosaurs?* In Proceedings of the 7th International Conference on Compiler Construction (CC'98), Springer-V., LNCS 1383, 65-79, 1998.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (10)

Verschiedenes

Stephen M. Blackburn, Amer Diwan, Matthias Hauswirth, Peter F. Sweeny, José Nelson Amaral, Tim Brecht, Lubomír Bulej, Cliff Click, Lieven Eeckhout, Sebastian Fischmeister, Daniel Frampton, Laurie J. Hendren, Michael Hind, Antony L. Hosking, Richard E. Jones, Tomas Kalibera, Nathan Keynes, Nathaniel Nystrom, Andreas Zeller. *The Truth, The Whole Truth, and Nothing But the Truth: A Pragmatic Guide to Assessing Empirical Evaluations*. ACM Transactions on Programming Languages and Systems 38(4), Article 15:1-20, 2016.

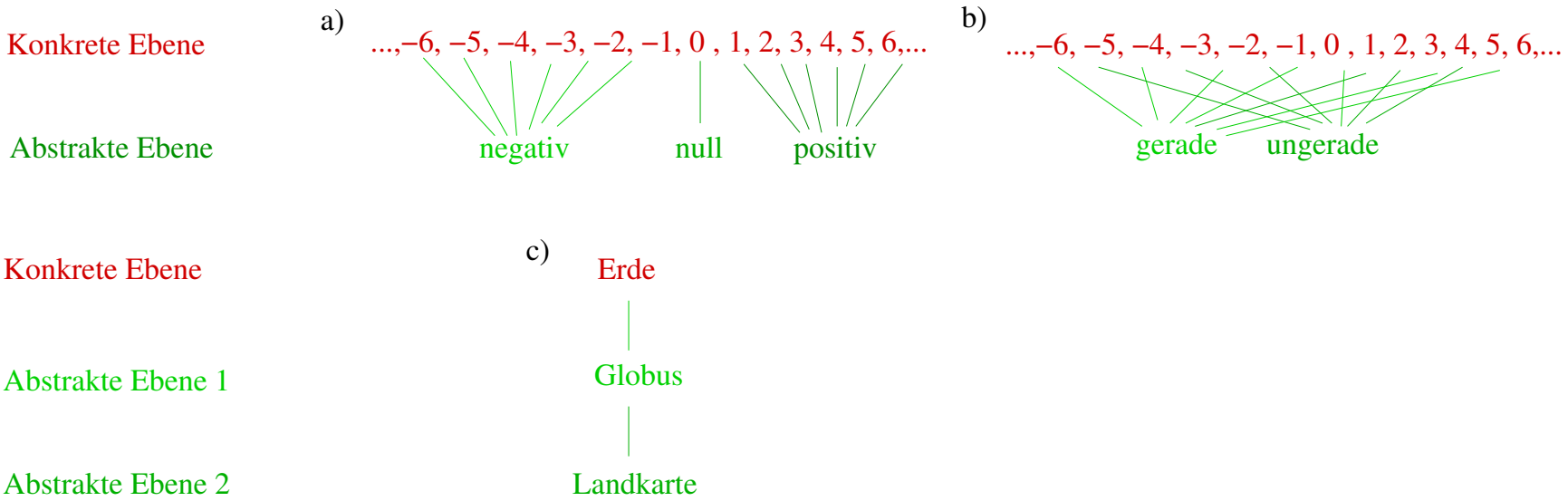
Kapitel 7

Abstrakte Semantiken, Analysesemantiken

Abstraktion: Grundlage für Programmanalyse

Abstraktion

...die Zusammenfassung oder Identifikation auf 'konkreter' Ebene unterschiedener Dinge, Objekte, etc. auf 'abstrakter' Ebene, um Entscheidbarkeit, Skalierbarkeit, etc. zu erreichen.



Wichtige Abstraktionsfragen

...für Programmanalyse.

Wie, wodurch sollen

1. Analyseinformationen
2. Programme

abstrahiert (modelliert, dargestellt) sein?

Viele Antworten sind möglich (und werden in der Praxis gegeben). Unsere Antwort für dieses und die unmittelbar folgenden Kapitel: Durch

1. Elemente vollständiger Verbände (Kap. 7.1)
2. kantenbenannte nichtdet. Flussgraphen (Kap. 7.2)

Kapitel 7.1

Abstrakte Informationsmodellierung: Verbände

Von part. Ordnungen zu vollst. Verbänden

Definition 7.1.1 (Partielle Ordnung)

Ein Paar (M, R) mit M Menge und $R \subseteq M \times M$ Relation auf M ist eine partielle Ordnung (partiell geordnete Menge) (engl. partial order (partially ordered set) gdw R ist reflexiv, transitiv und anti-symmetrisch.

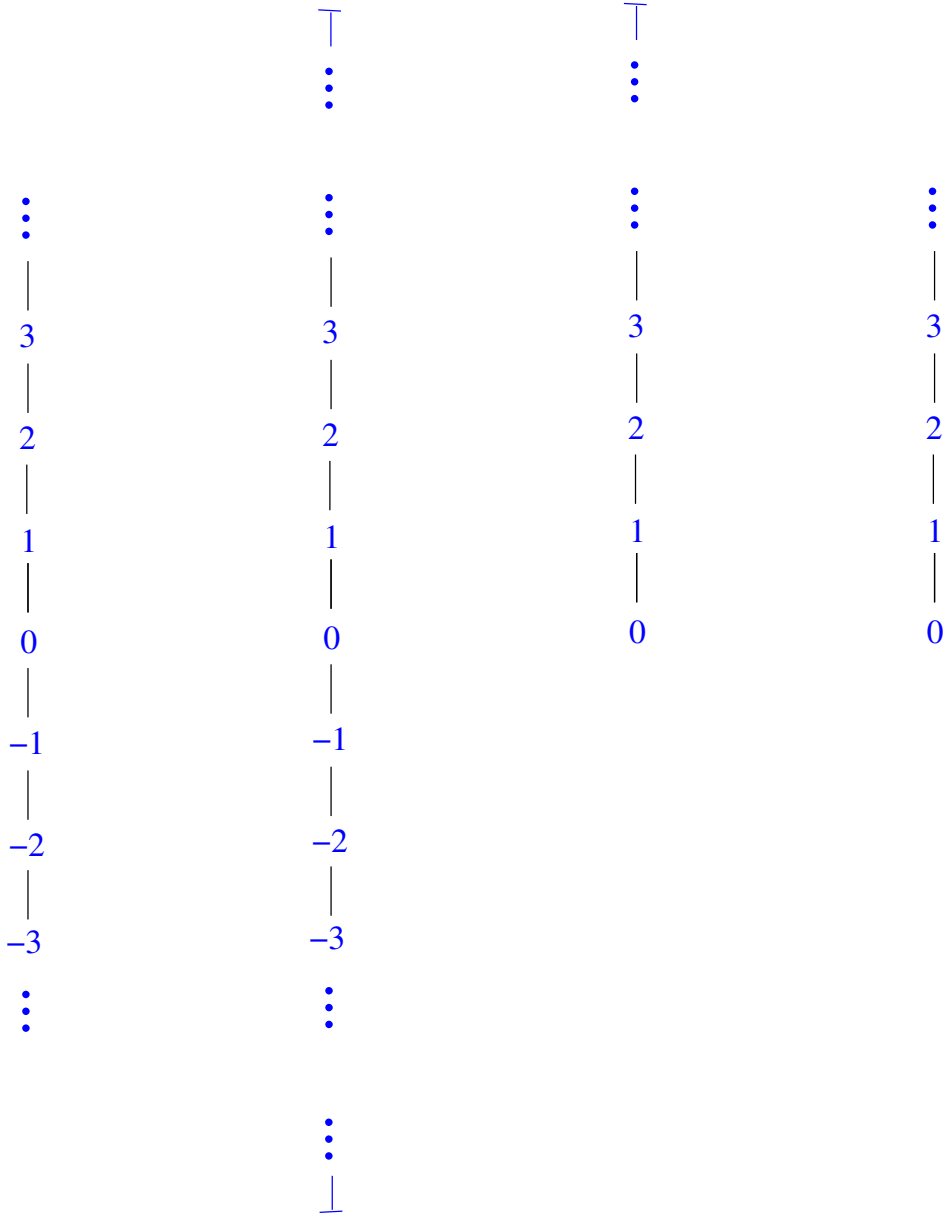
Definition 7.1.2 (Verband, Vollständiger Verband)

Eine partielle Ordnung $(P, \sqsubseteq)$ ist ein

1. Verband (engl. lattice), wenn jede nichtleere endliche Teilmenge P' von P eine kleinste obere und eine größte untere Schranke in P besitzt.
2. vollständiger Verband (engl. complete lattice), wenn jede Teilmenge P' von P eine kleinste obere und eine größte untere Schranke in P besitzt.

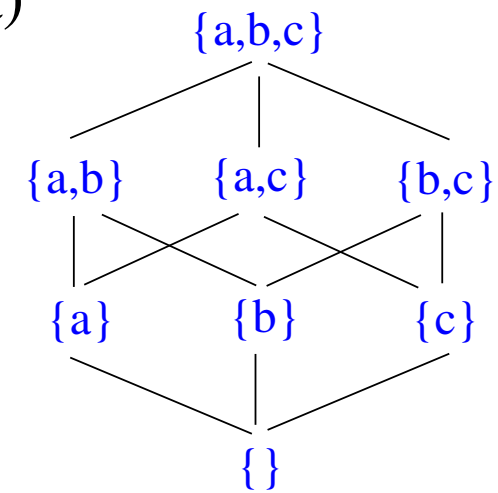
Beispiele partieller Ordnungen und Verbände

a) b) c) d)

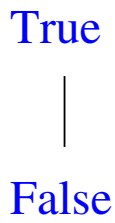


Beispiele vollständiger Verbände

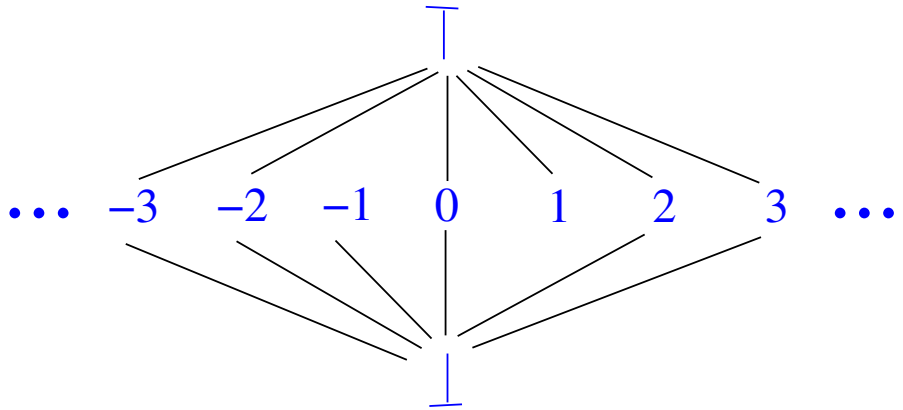
a)



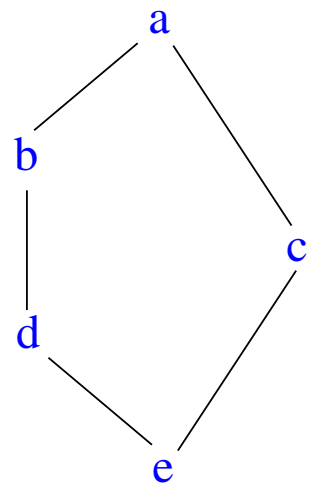
b)



c)



d)



Eigenschaften vollständiger Verbände

Lemma 7.1.3

Für $(\mathcal{C}, \sqsubseteq)$ vollständiger Verband gilt:

1. Jede Teilmenge $C \subseteq \mathcal{C}$ von $\mathcal{C}$ besitzt eine
   - 1.1 größte untere Schranke (Infimum)
   - 1.2 kleinste obere Schranke (Supremum)

   in $\mathcal{C}$.
2. $\mathcal{C}$ besitzt ein
   - 2.1 kleinstes
   - 2.2 größtes

   Element.

Infimum und Supremum von C werden mit $\bigsqcap C$ und $\bigsqcup C$ bezeichnet, kleinstes und größtes Element von $\mathcal{C}$ mit $\bot$ und $\top$.

Charakterisierungen ‘extremer’ Elemente

Lemma 7.1.4

Für $(\mathcal{C}, \sqsubseteq)$ vollständiger Verband gilt:

1. Für jede Teilmenge $C \subseteq \mathcal{C}$ gilt:
   - 1.1 $\bigsqcap C = \bigsqcup\{c \in \mathcal{C} \mid c \sqsubseteq C\}$
   - 1.2 $\bigsqcup C = \bigsqcap\{c \in \mathcal{C} \mid C \sqsubseteq c\}$
2. Für die kleinsten und größten Elemente $\bot$ und $\top$ gilt:
   - 2.1 $\bot = \bigsqcap \mathcal{C} = \bigsqcup \emptyset$
   - 2.2 $\top = \bigsqcup \mathcal{C} = \bigsqcap \emptyset$

Ordnungstheoretische und algebraische Sicht

In Verbänden lassen sich Ordnungsrelation und Schnitt- und Vereinigungsoperation wechselweise aufeinander zurückführen:

- $c_1 \sqcap c_2 = c_1$ und $c_1 \sqcup c_2 = c_2$ gdw $c_1 \sqsubseteq c_2$.
- $c_1 \sqsubseteq c_2$ gdw $c_1 \sqcap c_2 = c_1$ und $c_1 \sqcup c_2 = c_2$.

Zur Betonung spricht man spricht deshalb auch von Verbänden in den zueinander gleichwertigen Sichten als ordnungstheoretische und algebraische Verbände $(\mathcal{C}, \sqsubseteq)$ und $(\mathcal{C}, \sqcap, \sqcup)$ (s.a. Anhang A.4.6).

Schreib- und Sprechweisen für Verbände

Verbände, in denen kleinste und größte Elemente nicht notwendig existieren, werden deshalb oft als Paar, Tripel oder Quadrupel angegeben:

- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq)$ oder $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup)$ oder $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup)$

vollständige Verbände entsprechend als Sechstupel:

- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$

Die Symbole $\sqcap$, $\sqcup$, $\bot$ und $\top$ werden dabei als Schnitt, Vereinigung, 'bottom' und 'top' gelesen.

Schnitt-, Vereinigungsvollständigkeit

...wird Vollständigkeit eines Verbands nur für Schnitt oder Vereinigung benötigt bzw. ausgenutzt, wird das durch die Sprechweisen Schnitt- bzw. Vereinigungsvollständigkeit ($\sqcap$-vollständig, $\sqcup$-vollständig) und schnitt- bzw. vereinigungsvollständige Verbände ausgedrückt.

Es gilt allerdings:

Lemma 7.1.5

Für Verbände $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq)$ mit kleinstem und größtem Element sind äquivalent:

1. $\widehat{\mathcal{C}}$ ist schnittvollständig.
2. $\widehat{\mathcal{C}}$ ist vereinigungsvollständig.

Kapitel 7.2

Abstrakte Programmmodellierung: Flussgraphen

Flussgraphen

...als Abstraktion von als Text (Zeichenreihen) oder konkreter oder abstrakter Syntaxbaum dargestellter WHILE-Programme:

Definition 7.2.1 (Flussgraph)

Ein Flussgraph zu einem WHILE- Programm π ist ein Quadrupel $G = (N, E, \mathbf{s}, \mathbf{e})$ mit:

- N Menge von Knoten
- $E \subseteq N \times N$ Menge von Kanten
- $\mathbf{s}$ ausgezeichneter Startknoten ohne Vorgänger
- $\mathbf{e}$ ausgezeichneter Endknoten ohne Nachfolger

dessen Knoten oder Kanten mit den Anweisungen (Zuweisungen, Tests,...) benannt sind.

ObdA nehmen wir an, dass jeder Knoten auf einem Pfad von $\mathbf{s}$ nach $\mathbf{e}$ liegt.

Deutung von Flussgraphen/-bestandteilen

Für den Flussgraphen $G = (N, E, \mathbf{s}, \mathbf{e})$ eines WHILE-Programms π gilt:

- Knoten von G repräsentieren die Programmpunkte
- Kanten von G repräsentieren die Verzweigungsstruktur

von π.

Sind die Anweisungen (d.h. Zuweisungen, Tests) von π im Stil

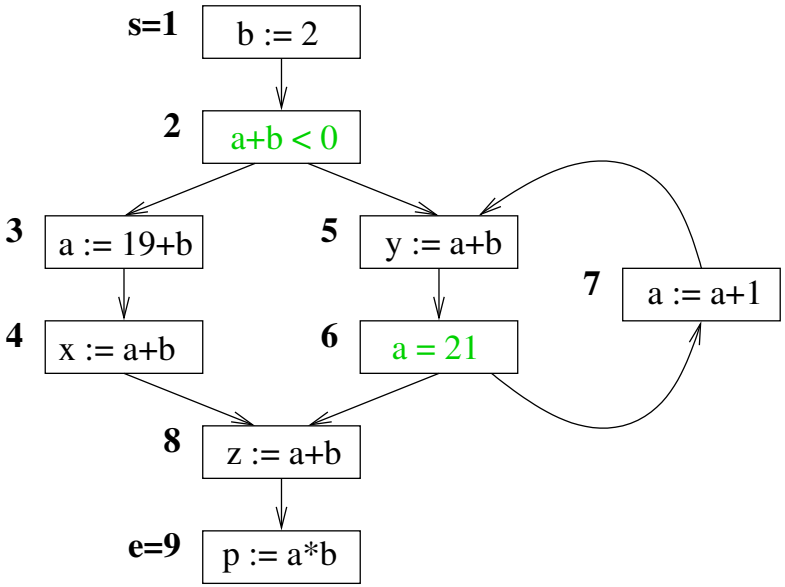
- einer Kripke-Struktur den Knoten
- eines Transitionssystems den Kanten

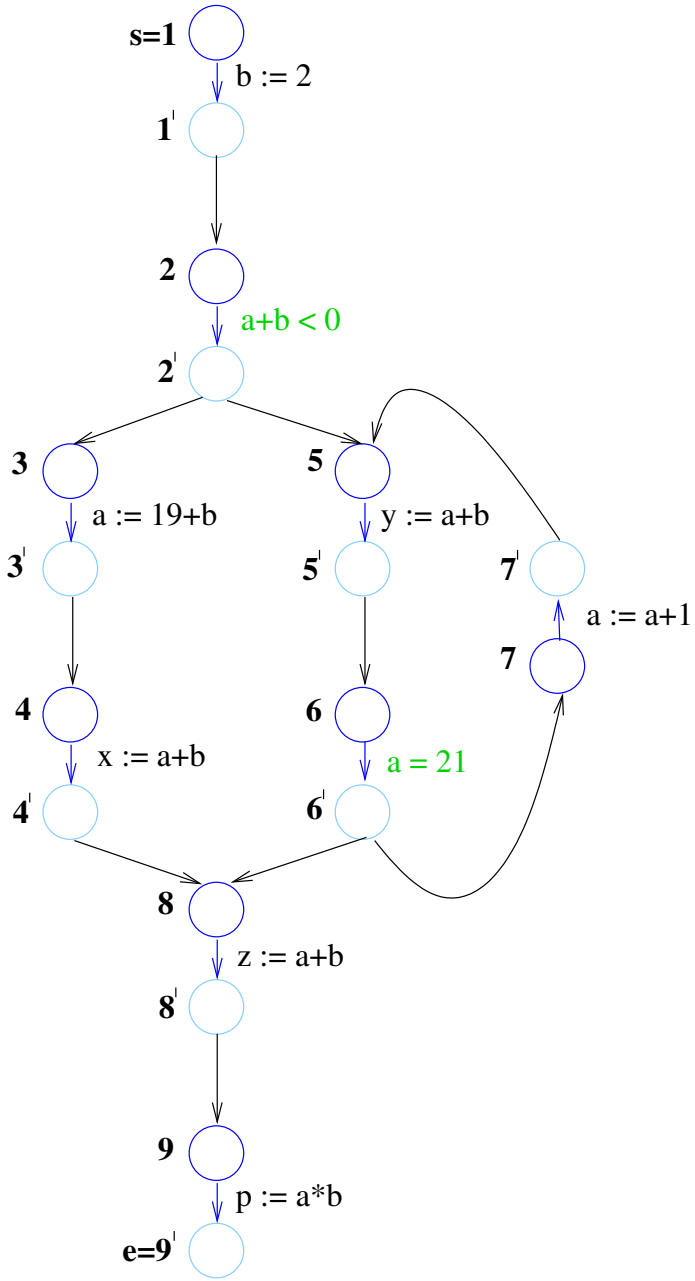
von G zugeordnet, sprechen wir von

- knotenbenannten
- kantenbenannten

Flussgraphen.

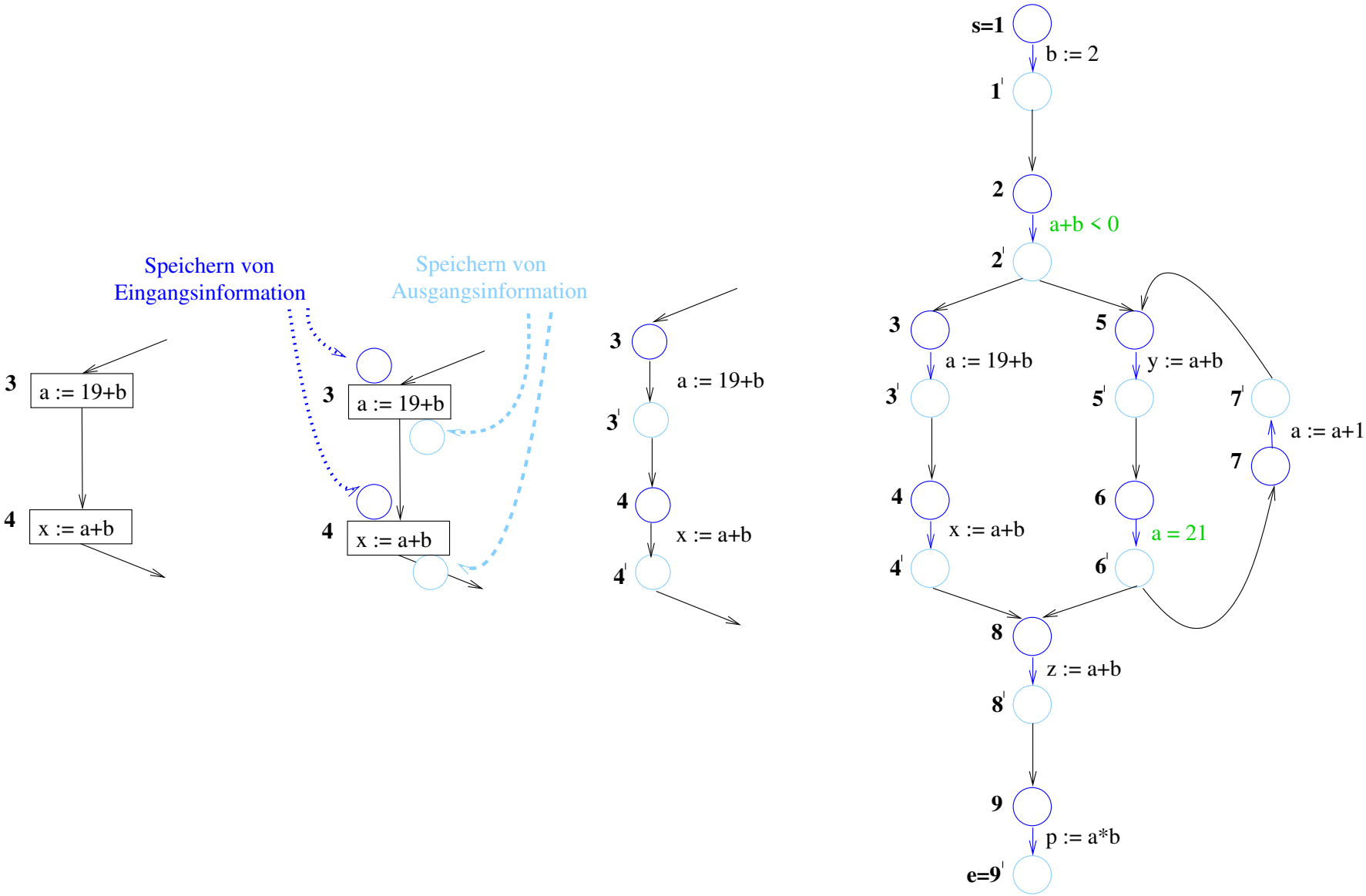
Bsp.: Knoten- u. kantenbenannte Flussgraphen





Schematische Überführung

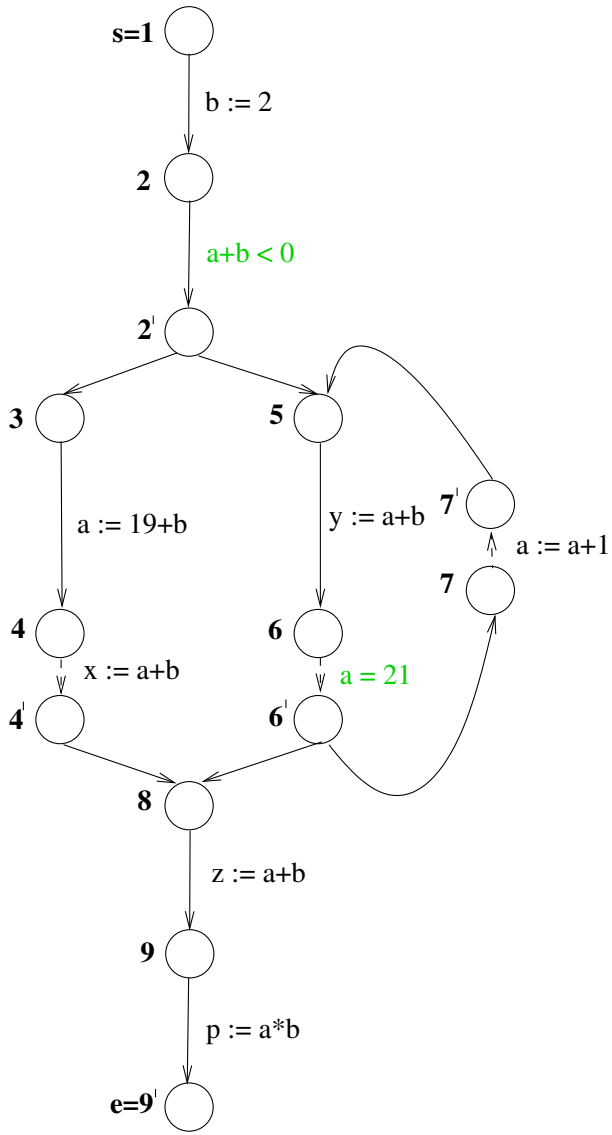
...eines knoten- in einen kantenbenannten Flussgraphen:



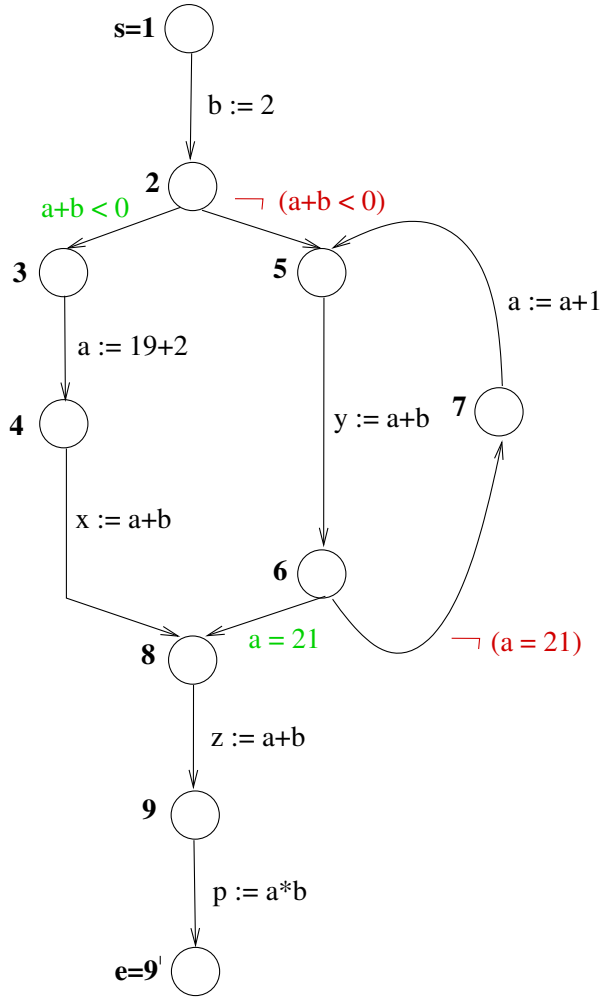
Der schematisch überführte Flussgraph nach

...a) ‘Aufräumen’ und b) aufgelöster Verzweigungsbehandlung:

a)



b)



Bezeichnungen

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Flussgraph, m, n zwei Knoten aus N.

Definition 7.2.2 (Vorgänger-, Nachfolgerknoten)

- $pred(n) =_{df} \{ m \mid (m, n) \in E \}$ bezeichnet die Menge der Vorgängerknoten von n.
- $succ(n) =_{df} \{ m \mid (n, m) \in E \}$ bezeichnet die Menge der Nachfolgerknoten von n.

Definition 7.2.3 (Pfade)

- Eine Folge von Kanten $\langle (n_1, m_1), (n_2, m_2), \ldots, (n_k, m_k) \rangle$, wobei $m_i = n_{i+1}$, $1 \leq i < k$, heißt Pfad von n_1 nach m_k.
- $\mathbf{P}[m, n]$ bezeichnet die Menge aller Pfade von m nach n.

Ist der Flussgraph G aus d. Kontext nicht eindeutig bestimmt, schreiben wir statt $pred$, $succ$, $\mathbf{P}$ genauer $pred_G$, $succ_G$, $\mathbf{P}_G$.

In der Folge

1. wählen wir (meist)
   - kantenbenannte Flussgraphen

   als Programmabstraktion, da sie notationell zu weniger Aufwand führen und aus pragmatischer Sicht deshalb günstiger sind.
2. werten wir Verzweigungsbedingungen in Flussgraphen nicht aus, um (einige) Unentscheidbarkeiten von vornherein zu vermeiden, und sprechen deshalb von
   - nichtdeterministischen (kantenbenannten) Flussgraphen.

Anmerkung: Pragmatische Vor- und Nachteile unterschiedlicher Flussgraphvarianten für Programmanalyse werden in Anhang B 'Flussgraphvarianten' untersucht und diskutiert.

Kapitel 7.3

Lokale abstrakte Semantiken

Lokale abstrakte Semantiken, Instruktionssem.

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein kantenbenannter Flussgraph und $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq)$ ein Verband.

Definition 7.3.1 (Lokale abstrakte Semantik)

Eine lokale abstrakte (Kanten-) Semantik von G ist eine Funktion:

$$[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$$

die jeder Kante von G eine Funktion auf der Elementmenge $\mathcal{C}$ von $\widehat{\mathcal{C}}$ als Bedeutung zuordnet.

In Anwendungen kommt lokalen abstrakten Semantiken die Rolle der auf das Niveau des Verbands abstrahierten Instruktionssemantik zu, da die Kanten kantenbenannter Flussgraphen mit den Anweisungen bzw. Instruktionen des zugrundeliegenden Programms benannt sind (s. Kapitel 8).

Kapitel 7.4

Operationelle globale abstrakte Semantiken

Überblick

...lokale abstrakte Semantiken können kompositionell von Kanten auf Pfade und vollständige Flussgraphen ausgedehnt werden. In einem ersten Schritt führt uns das zu einer operationellen globalen abstrakten Semantik für Flussgraphen, der nichtdeterministischen

1. Aufsammelsemantik (AS) (engl. collecting semantics (*CS*))

Von AS leiten wir anschließend zwei deterministische Varianten ab: Die

2. Schnitt-über-alle-Pfade (*SUP*) Semantik (engl. meet over all paths (*MOP*) semantics)
3. Vereinigung-über-alle-Pfade (*VUP*) Semantik (engl. join over all paths (*JOP*) semantics)

Die *SUP* - und *VUP* -Semantiken werden sich in Kapitel 8 als für Programmanalyse intuitiv gewünscht, aber unentscheidbar herausstellen.

Ausblick

In Kapitel 8 werden wir deshalb zusätzlich zwei (berechenbare) denotationelle Gegenstücke zur *SUP*- und *VUP*-Semantik einführen: Dual zur *VUP*-Semantik die

1. maximale Fixpunktsemantik (*MaxFP*) (engl. maximum fixed point (*MaxFP*) semantics)

und dual zur *VUP*-Semantik die

2. minimale Fixpunktsemantik (*MinFP*) (engl. minimum fixed point (*MinFP*) semantics)

Die *MaxFP*- und *MinFP*-Semantiken induzieren anders als ihre operationellen Gegenstücke iterative Berechnungsverfahren, mit denen sich die *SUP* - und *VUP* -Semantiken unter gewissen Voraussetzungen approximativ und unter in der Praxis oft erfüllten zusätzlichen Voraussetzungen sogar exakt berechnen lassen.

Kapitel 7.4.1

Pfadausdehnung lokaler abstrakter Semantiken

Pfadausdehnung lokaler Semantiken

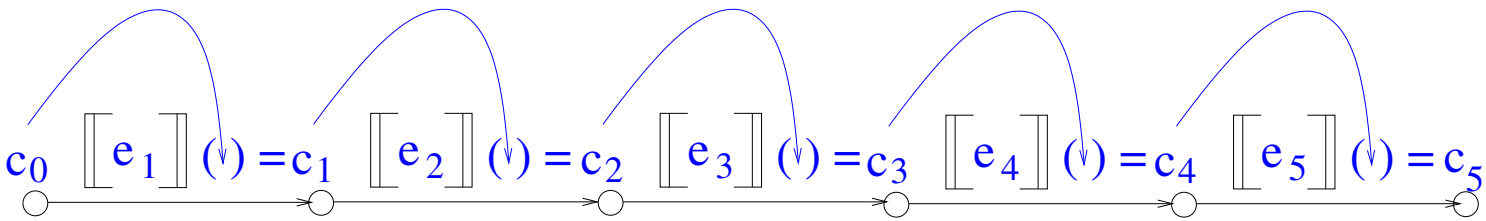
Definition 7.4.1.1 (Pfadausdehnung von $[\![\ ]\!]$)

Die Pfadausdehnung $[\![\ p\ ]\!]$ einer lokalen abstrakten Semantik $[\![\ ]\!]$ auf einen Pfad $p = \langle e_1, e_2, \ldots, e_q \rangle$ ist kompositionell definiert durch:

$$[\![\ p\ ]\!] =_{df} \begin{cases} Id_{\mathcal{C}} & \text{falls } \lambda_p < 1 \\ [\![\ \langle e_2, \ldots, e_q \rangle\ ]\!] \circ [\![\ e_1\ ]\!] & \text{sonst} \end{cases}$$

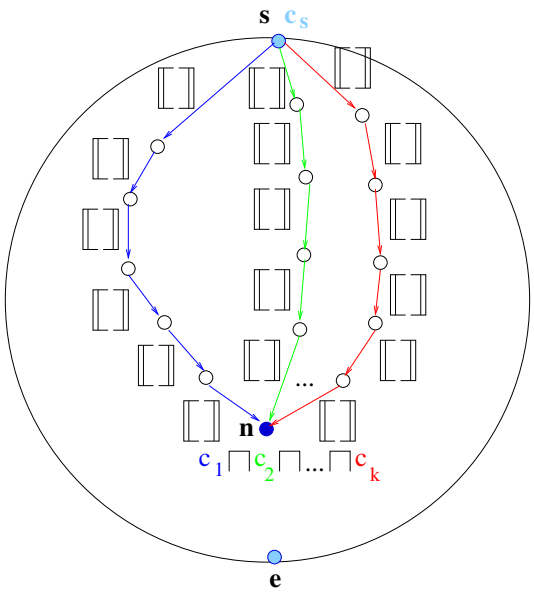
wobei $Id_{\mathcal{C}} = \lambda c \in \mathcal{C}.\, c$ die Identität auf $\mathcal{C}$ bezeichnet.

Veranschaulichung der Pfadausdehnung von $[\![\ ]\!]$:

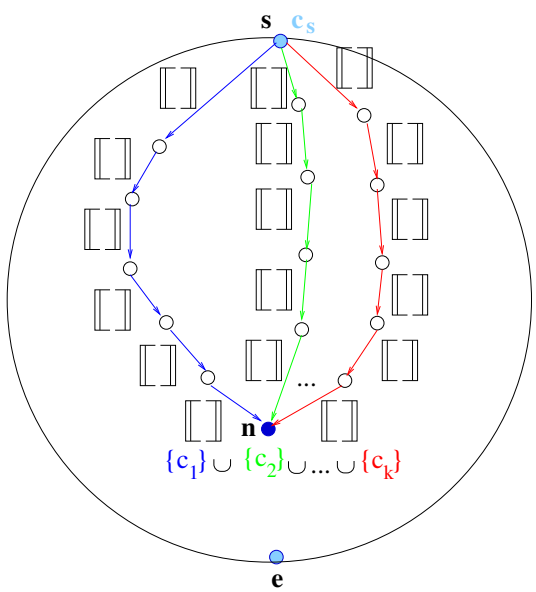


Ausgehend von der Pfadausdehnung

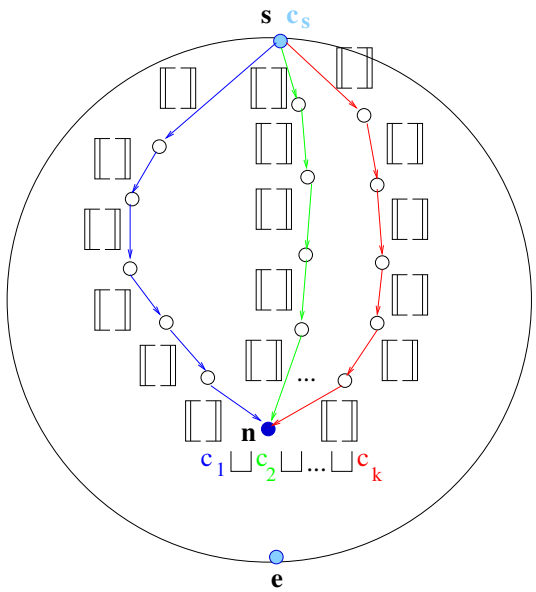
...lokaler abstrakter Semantiken $[\![\ ]\!]$ betrachten wir jetzt drei Globalisierungstrategien zur Ausdehnung von Pfadsemantiken auf Semantiken vollständiger Flussgraphen:



Schnitt–ueber–alle–Pfade–Semantik



Aufsammelsemantik



Vereinigung–ueber–alle–Pfade–Semanti

Sei in der Folge

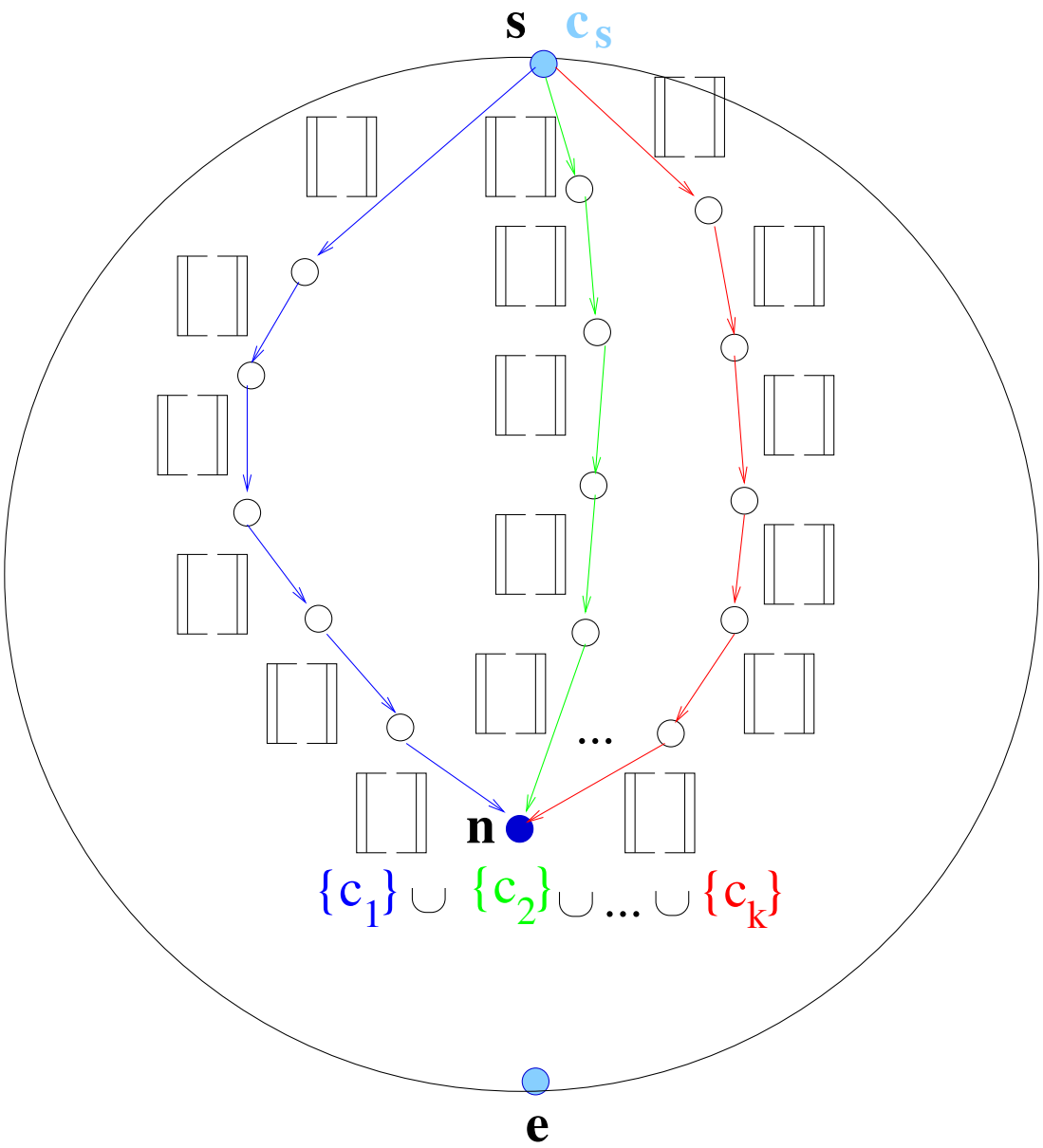
...von Kapitel 7.4:

- $G = (N, E, \mathbf{s}, \mathbf{e})$ ein kantenbenannter Flussgraph
- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq)$ bzw. $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ ein Verband bzw. vollständiger Verband
- $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ eine lokale abstrakte Semantik für G

Kapitel 7.4.2

Aufsammelsemantik

Veranschaulichung der Aufsammelsemantik



Aufsammelsemantik

Definition 7.4.2.1 (Aufsammelsemantik)

Die von $[\![\ ]\!]$ induzierte Aufsammelsemantik (oder: nichtdeterministische globale abstrakte Semantik) (engl. collecting semantics) von G ist definiert durch:

$$[\![\ ]\!]_{AS} : \mathcal{C} \to N \to \mathcal{P}(\mathcal{C})$$

$$[\![\ ]\!]_{AS} =_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \{\, [\![\, p \,]\!](c) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\}$$

wobei $\mathcal{P}$ den Potenzmengenoperator bezeichnet.

Ohne besondere Anforderungen an $\widehat{\mathcal{C}}$ und $[\![\ ]\!]$ gilt:

Lemma 7.4.2.2 (AS-Wohldefiniertheit)

Die A-Semantik $[\![\ ]\!]_{AS}$ von G ist wohldefiniert.

Zusammenhang von $[\![\pi]\!]_{\text{WHILE}}$ und $[\![\mathbf{e}]\!]_{AS}$

...ist π ein WHILE -Programm, G seine Flussgraphdarstellung und $[\![\ ]\!]$ eine lokale abstrakte Semantik für G, dann ist die Aufsammelsemantik am Endknoten $\mathbf{e}$ von G:

$$[\![\mathbf{e}]\!]_{AS}(\mathcal{C}) =_{df} \{ [\![p]\!](c) \mid p \in \mathbf{P}[\mathbf{s}, \mathbf{e}],\ c \in \mathcal{C} \}$$

das abstrakte nichtdeterministische Gegenstück zur deterministischen WHILE -Semantik von π für Σ:

$$[\![\pi]\!]_{\text{WHILE}}(\Sigma) =_{df} \{ [\![\pi]\!]_{\text{WHILE}}(\sigma) \mid \sigma \in \Sigma \}$$

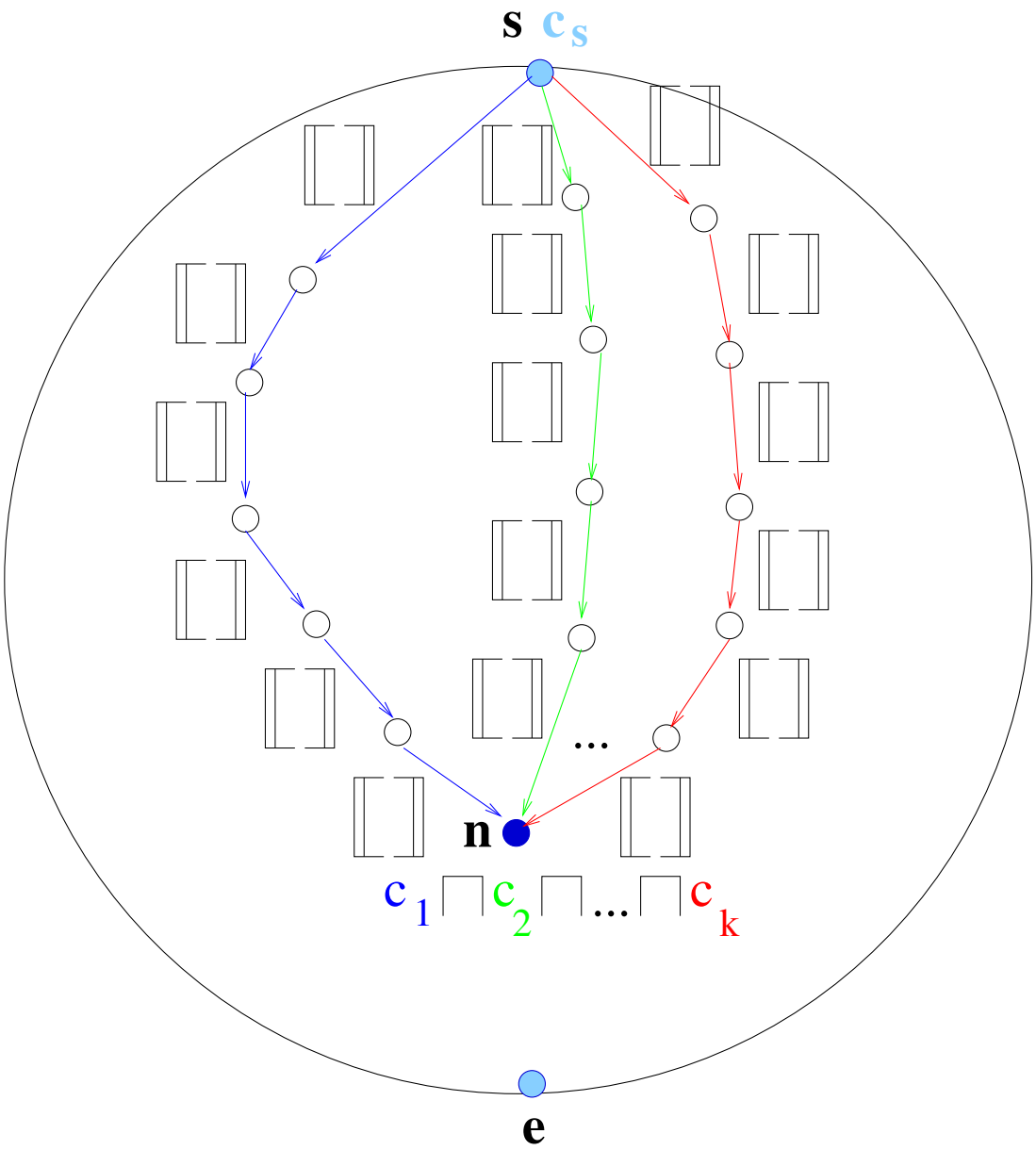
Informell:

$$[\![\pi]\!]_{\text{WHILE}}(\Sigma) \ \widehat{=}\ [\![\mathbf{e}]\!]_{AS}(\mathcal{C})$$

Kapitel 7.4.3

Schnitt-über-alle-Pfade-Semantik

Veranschaulichung der *SUP*-Semantik



Schnitt-über-alle-Pfade (*SUP*) Semantik

Definition 7.4.3.1 (*SUP*-Semantik)

Die von $[\![\ ]\!]$ induzierte *SUP*-Semantik (oder: deterministische globale Schnitt-über-alle-Pfade-Semantik) von G ist definiert durch:

$$[\![\ ]\!]_{SUP} : \mathcal{C} \to N \to \mathcal{C}$$

$$\begin{aligned} [\![\ ]\!]_{SUP} &=_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \bigsqcap [\![\, n \,]\!]_{AS}(c) \\ &= \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \bigsqcap \{ [\![\, p \,]\!](c) \mid p \in \mathbf{P}[\mathbf{s}, n] \} \end{aligned}$$

Es gilt:

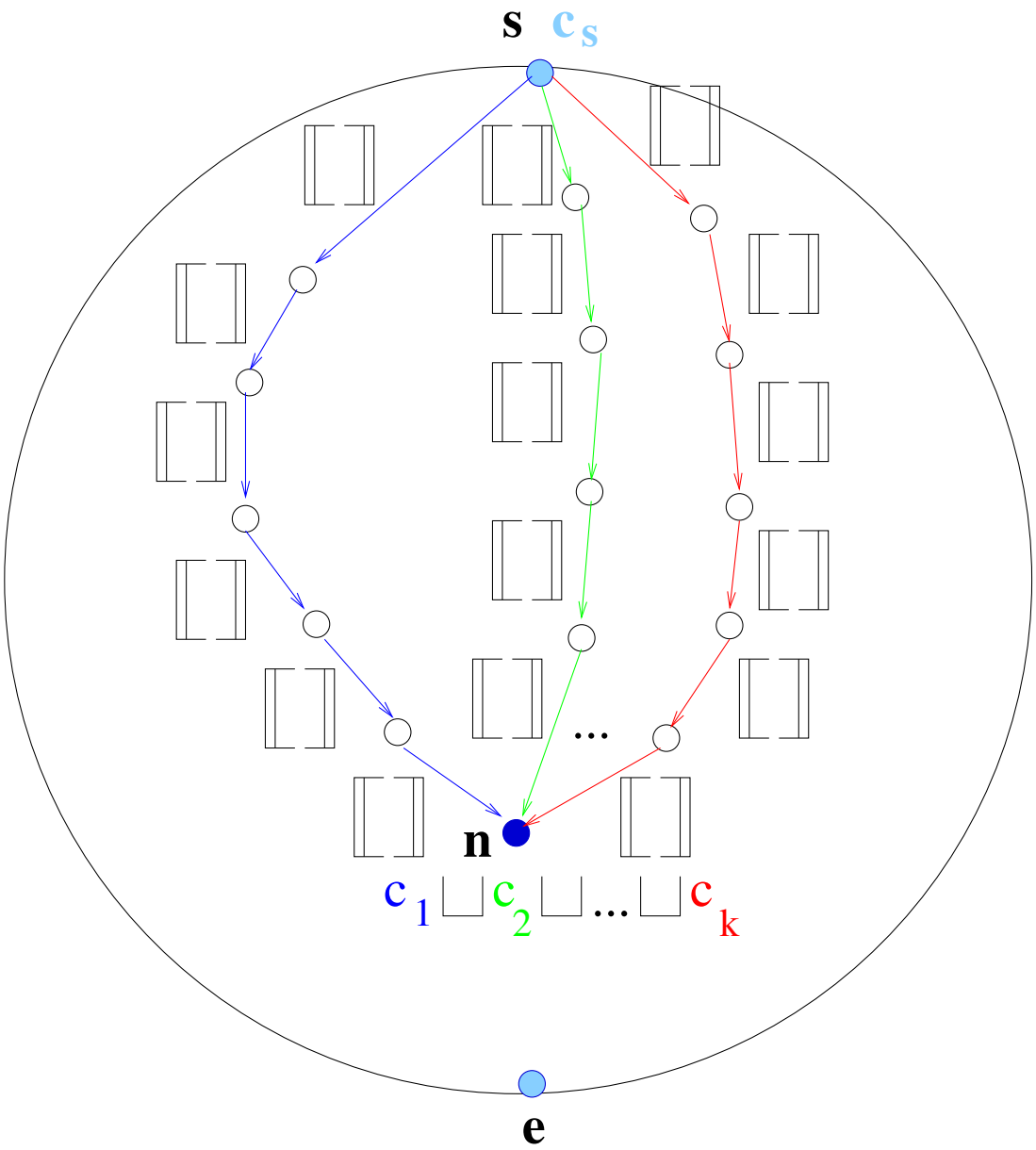
Lemma 7.4.3.2 (*SUP*-Wohldefiniertheit)

Die *SUP*-Semantik $[\![\ ]\!]_{SUP}$ von G ist wohldefiniert, wenn $\widehat{\mathcal{C}}$ $\sqcap$-vollständig ist.

Kapitel 7.4.4

Vereinigung-über-alle-Pfade-Semantik

Veranschaulichung der *VUP*-Semantik



Vereinigung-über-alle-Pfade (*VUP*) Semantik

Definition 7.4.4.1 (*VUP*-Semantik)

Die von $[\![\ ]\!]$ induzierte *VUP*-Semantik (oder: deterministische globale Vereinigung-über-alle-Pfade-Semantik) von G ist definiert durch:

$$[\![\ ]\!]_{VUP} : \mathcal{C} \to N \to \mathcal{C}$$

$$\begin{aligned} [\![\ ]\!]_{VUP} &=_{df} \forall\, c \in \mathcal{C}.\ \forall\, n \in N.\ \bigsqcup [\![\ n\ ]\!]_{AS}(c) \\ &= \forall\, c \in \mathcal{C}.\ \forall\, n \in N.\ \bigsqcup \{\, [\![\ p\ ]\!](c) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\} \end{aligned}$$

Es gilt:

Lemma 7.4.4.2 (*VUP*-Wohldefiniertheit)

Die *VUP*-Semantik $[\![\ ]\!]_{VUP}$ von G ist wohldefiniert, wenn $\widehat{\mathcal{C}}$ $\sqcup$-vollständig ist.

Kapitel 7.5

Zusammenfassung

Zur Definiertheit von $[\![\ ]\!]_{AS}$, $[\![\ ]\!]_{SUP}$, $[\![\ ]\!]_{VUP}$

Die Aufsammelsemantik ist

- stets definiert.

Weder Verband noch lokale abstrakte Semantik müssen dafür besonderen Anforderungen genügen.

Die Schnitt- und Vereinigung-über-alle-Pfade-Semantiken sind

- definiert, wenn der Verband ($\sqcap$/$\sqcup$-) vollständig ist.

Die lokale abstrakte Semantik muss dafür keinen besonderen Anforderungen genügen.

Von ihrem Wesen

...sind

- Aufsammelsemantik
- Schnitt-über-alle-Pfade-Semantik
- Vereinigung-über-alle-Pfade-Semantik

operationell (orientiert an Programmpfaden).

...ist die

- Aufsammelsemantik

nichtdeterministisch (i.S.v.: liefert die Menge möglicher Werte für jeden Programmpunkt).

...sind

- Schnitt-über-alle-Pfade-Semantik
- Vereinigung-über-alle-Pfade-Semantik

deterministisch (i.S.v.: liefern genau einen Wert für jeden Programmpunkt).

Kapitel 7.6

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 7 (1)

- Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004. (Appendix B.3.1, Graphical Intermediate Representations)
- Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.
- John B. Kam, Jeffrey D. Ullman. *Global Data Flow Analysis and Iterative Algorithms*. Journal of the ACM 23:158-171, 1976.
- John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 7 (2)

Uday P. Khedker, Amitabha Sanyal, Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press, 2009. (Chapter 3, Theoretical Abstractions in Data Flow Analysis)

Robert Morgan. *Building an Optimizing Compiler*. Digital Press, 1998. (Chapter 2.3, Building the Flow Graph; Chapter 4.7, Structure of Program Flow Graph)

Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997. (Chapter 7, Control-Flow Analysis)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 7 (3)

Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019. (Chapter 1, Program Graphs)

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 5, Static Program Analysis)

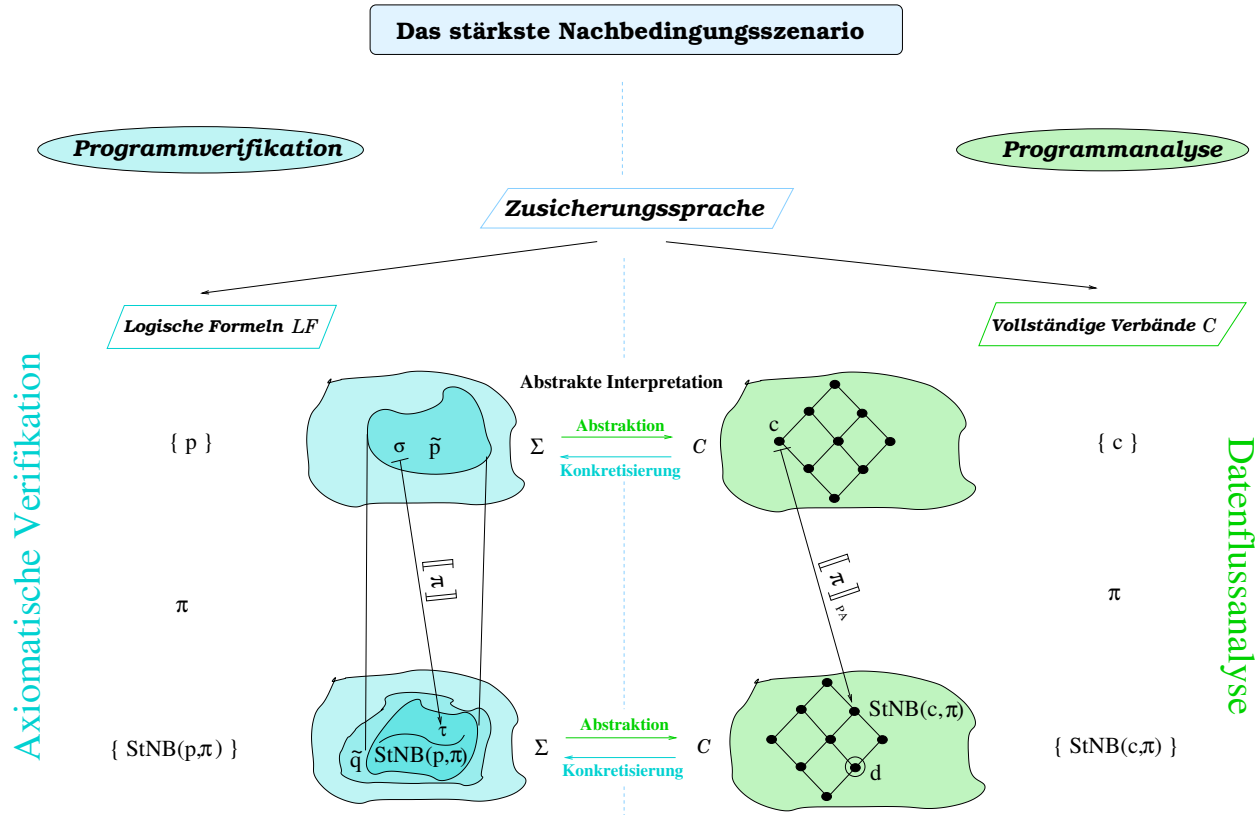
Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 7, Program Analysis; Chapter 8, More on Program Analysis; Appendix B, Implementation of Program Analysis)

Kapitel 8

Datenflussanalyse

Datenflussanalyse: Berechnung stärkster

...Nachbedingungen: Das Analogon zu axiomatischer stärkster Nachbedingungsverfikation (s. Kapitel 4).



StNB(p, π) ∈ *LF* *muss erfüllen:*

(1) $\models_{PV} \{ p \}\pi\{ StNB(p,\pi) \}$

(2) $\forall\, q \in LF.\ \models_{PV} \{ p \}\pi \{ q \}$ impliziert $StNB(p,\pi) \Longrightarrow q$

StNB(c,π) ∈ *C* *muss erfüllen:*

(1) $\models_{PA} \{ c \}\pi \{ StNB(c,\pi) \}$

(2) $\forall\, d \in C.\ \models_{PA} \{ c \}\pi \{ d \}$ impliziert $StNB(c,\pi) \sqsupseteq d$

Datenflussanalyse, Datenflussanalyseprobleme

...vollständig beschrieben als Paar aus

- vollständigem Verband
- lokaler abstrakter Semantik

die die (DFA-) Semantik der Datenflussanalyse festlegen.

Eine zusätzliche

- Anfangsinformation für den Startknoten

legt dann ein konkretes DFA-Problem, eine Probleminstanz der (DFA-) Analyse fest.

Sei in der Folge von Kapitel 8

- $G = (N, E, \mathbf{s}, \mathbf{e})$ der kanntenbenannte Flussgraph

eines WHILE-Programms π.

Kapitel 8.1

DFA-Spezifikationen, DFA-Probleme

DFA-Semantik, Datenflussanalyse

Definition 8.1.1 (DFA-Semantik, Datenflussanalyse)

Ein Paar $(\widehat{\mathcal{C}}, [\![\ ]\!])$ mit:

- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ vollständiger Verband
- $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ lokale abstrakte Semantik für G

definiert die (lokale) (DFA-) Semantik einer Datenflussanalyse für G.

DFA-Spezifikation, DFA-Problem

Definition 8.1.2 (DFA-Spezifikation, DFA-Problem)

Ein Tripel $(\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathsf{s}})$ mit:

- $(\widehat{\mathcal{C}}, [\![\ ]\!])$ (lokale) DFA-Semantik für G
- $c_{\mathsf{s}} \in \mathcal{C}$ initiale Information (oder Anfangszusicherung)

spezifiziert ein konkretes DFA-Problem, die Probleminstanz $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathsf{s}})$.

Für

...eine DFA-Spezifikation $(\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ gilt:

- Die Elemente von $\widehat{\mathcal{C}}$ repräsentieren die interessierende(n) Datenflussinformation(en).
- Die Funktionen $[\![\, e \,]\!]$, $e \in E$, repräsentieren die Instruktionssemantik auf dem durch $\widehat{\mathcal{C}}$ gegebenen abstrakten (Analyse-) Niveau.
- $c_{\mathbf{s}} \in \mathcal{C}$ ist die Datenflussinformation, die die Analyse am Startknoten $\mathbf{s}$ von G als gültig annehmen soll.

Das legt als Sprechweisen nahe:

- $\widehat{\mathcal{C}}$ einen DFA-Verband
- $[\![\ ]\!]$ eine lokale (abstrakte) DFA-Semantik
- $[\![\, e \,]\!]$, $e \in E$, eine lokale (abstrakte) DFA-Semantikfunktion (oder kürzer: DFA-Funktion)
- $c_{\mathbf{s}} \in \mathcal{C}$ eine DFA-Anfangszusicherung

zu nennen.

Historisch getroffene Generalvereinbarung

...für Datenflussanalyse und DFA-Verbände:

- Verbandsmäßig größer

heißt

- bessere, genauere Information!

Beachte:

- In der Theorie abstrakter Interpretationen ist diese Vereinbarung genau andersherum getroffen (s. Kapitel 16.2).
- Beide Festlegungen sind grundsätzlich gleichwertig, müssen nur durchgehalten werden.

Beispiel: Verfügbarkeit eines Terms t (1)

...t heißt verfügbar an Programmpunkt n, wenn t auf jedem Pfad p von **s** nach n berechnet wird, ohne dass ein Operand von t nach der letzten Berechnung von t auf p modifiziert wird.

DFA-Spezifikation für die Verfügbarkeit von Term t:

- DFA-Verband
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $(\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr}) = \widehat{\mathbb{B}}$
- DFA-Semantik
  $[\![\ ]\!]^t_{av} : E \to (\ \mathbb{B} \to \mathbb{B}\ )$ where
  $\forall\, e \in E.\ [\![\ e\ ]\!]^t_{av} =_{df}\ \lambda\, b.\ (b \vee Comp^t_e) \wedge Transp^t_e$
- DFA-Anfangszusicherung: $b_{\mathbf{s}} \in \mathbb{B}$

Insgesamt:

- Verfügbarkeitsspezifikation: $\mathcal{S}^{av,t}_G = (\widehat{\mathbb{B}}, [\![\ ]\!]^t_{av}, b_{\mathbf{s}})$

Beispiel: Verfügbarkeit eines Terms t (2)

Dabei bezeichnet $\widehat{\mathbb{B}}$ den DFA-Verband der Wahrheitswerte:

- $\widehat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr})$

  ...Verband der Wahrheitswerte: kleinstes Element **falsch**, größtes Element **wahr**, **falsch** $\leq$ **wahr**, logisches $\wedge$ und logisches $\vee$ als Schnitt- und Vereinigungsoperation.

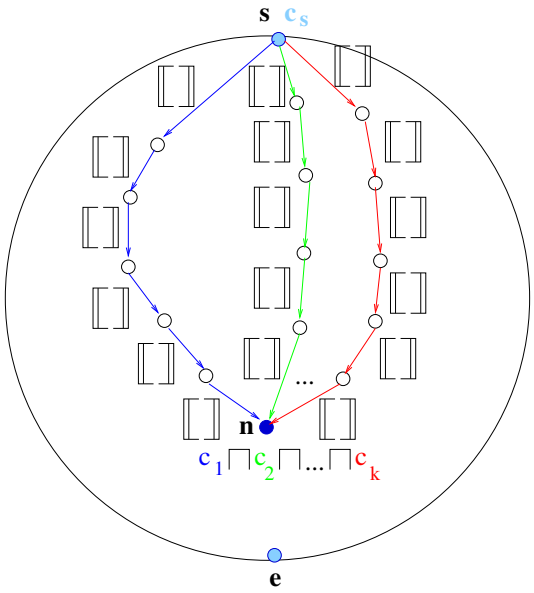
...und $Comp_e^t$ und $Transp_e^t$ zwei mit Kanten und ihren Instruktionen assoziierte lokale Prädikate:

- $Comp_e^t$ ...**wahr**, wenn t bei Ausführung der Instruktion an Kante e berechnet wird, **falsch** sonst.
- $Transp_e^t$ ...**wahr**, wenn e transparent für t ist (d.h., keinem Operanden von t wird bei Ausführung der Instruktion an Kante e ein Wert zugewiesen), **falsch** sonst.

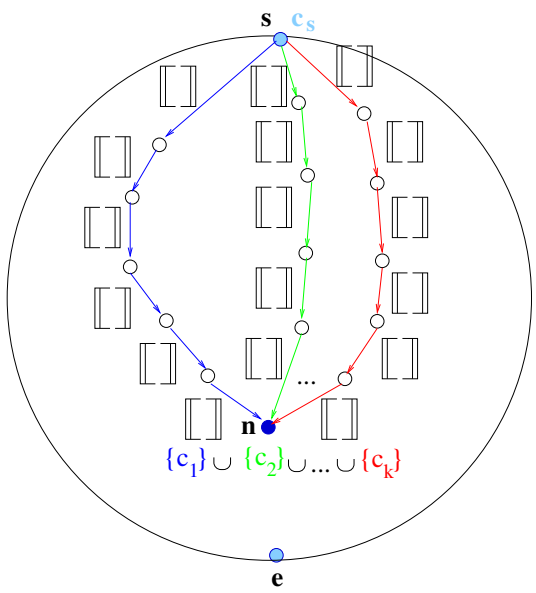
Kapitel 8.2

SUP- und *VUP*-Semantik als zueinander duale spezifizierende DFA-Problemlösungen

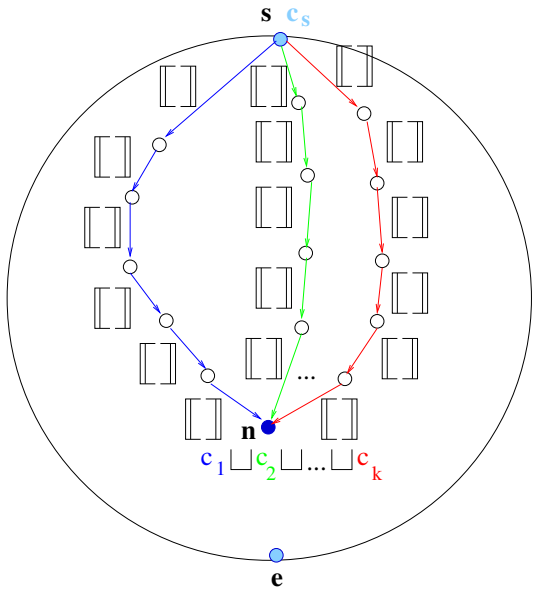
Wie in Abbildung



Schnitt–ueber–alle–Pfade–Semantik



Aufsammelsemantik



Vereinigung–ueber–alle–Pfade–Semanti

...aus Kapitel 7.4 illustriert, grenzen die *SUP*- und *VUP*-Semantik die am Programmpunkt n bzgl. $\mathcal{S}_G$ mögliche DFA-Information in auf folgender Seite gegebenem Sinn ein:

Eingrenzung

...für jeden Pfade $p \in \mathbf{P}[\mathbf{s}, n]$ ist die von p an n garantierte DFA-Information $[\![p]\!](c)$

- mindestens so groß wie die *SUP*-Semantik an n (es kann für keinen Pfad schlechter, höchstens gleich gut oder besser sein):

$$[\![p]\!](c) \sqsupseteq [\![n]\!]_{SUP}(c)$$

- höchstens so groß wie die *VUP*-Semantik an n (es kann für keinen Pfad besser, höchstens gleich gut oder schlechter sein):

$$[\![p]\!](c) \sqsubseteq [\![n]\!]_{VUP}(c)$$

Zusammen gilt:

$$\forall\, p \in \mathbf{P}[\mathbf{s}, n]\ \forall\, c \in \mathcal{C}.\ [\![n]\!]_{SUP}(c) \sqsubseteq [\![p]\!](c_{\mathbf{s}}) \sqsubseteq [\![n]\!]_{VUP}(c)$$

SUP - und *VUP* -Semantik: Intuitiv gewünscht

...*SUP*- und *VUP*-Semantik beschreiben damit für jeden Programmpunkt n die im folgenden Sinn an n bzgl. $\mathcal{S}_G$ bestmöglichen DFA-Informationen und sind damit die zueinander dualen intuitiv gewünschten DFA-Semantiken für G:

- $[\![n]\!]_{SUP}(c_{\mathbf{s}})$ ist die an n mindestens gültige Information (es kann für keinen Pfad schlechter sein, höchstens gleich oder besser): $\forall\, p \in \mathbf{P}[\mathbf{s}, n].\ [\![n]\!]_{SUP}(c_{\mathbf{s}}) \sqsubseteq [\![p]\!](c_{\mathbf{s}})$.
- $[\![n]\!]_{VUP}(c_{\mathbf{s}})$ ist die an n höchstens gültige Information (es kann für keinen Pfad besser sein, höchstens gleich oder schlechter): $\forall\, p \in \mathbf{P}[\mathbf{s}, n].\ [\![n]\!]_{VUP}(c_{\mathbf{s}}) \sqsupseteq [\![p]\!](c_{\mathbf{s}})$.

SUP- und *VUP*-Semantik garantieren damit die Abwesenheit von 'Überraschungen' zur Laufzeit:

$$\forall\, p \in \mathbf{P}[\mathbf{s}, n]\ \forall\, c_{\mathbf{s}} \in \mathcal{C}.\ [\![n]\!]_{SUP}(c_{\mathbf{s}}) \sqsubseteq [\![p]\!](c_{\mathbf{s}}) \sqsubseteq [\![n]\!]_{VUP}(c_{\mathbf{s}})$$

Spezifizierende DFA-Problemlösungen

Das legt folgende Festlegung nahe:

Definition 8.2.1 (Spezifizierende DFA-Problemlsg.)

Die *SUP*- und *VUP*-Semantik eines Flussgraphen definieren zwei zueinander duale spezifizierende Lösungen eines DFA-Problems, seine sog.:

1. *SUP*-Lösung
2. *VUP*-Lösung

Nach dieser Festlegung können wir jetzt definieren, wann ein DFA-Algorithmus zur Lösung von DFA-Problemen

- korrekt ($\Leftrightarrow$ unterapproximierend)
- vollständig ($\Leftrightarrow$ überapproximierend)
- akkurat (oder: optimal) ($\Leftrightarrow$ korrekt und vollständig)

heißen soll.

Kapitel 8.3

Korrektheit, Vollständigkeit, Akkuratheit von DFA-Algorithmen

Korrektheit, Vollständigkeit, Akkuratheit

...von DFA-Algorithmen.

Definition 8.3.1 (Korrekt, vollständig, akkurat)

Ein DFA-Algorithmus A heißt

1. *SUP*-korrekt (*VUP*-korrekt), wenn A für alle DFA-Probleme $\mathcal{S}_G$ mit einer unteren (oberen) Approximation der *SUP*- (*VUP*-) Semantik von G terminiert.
2. *SUP*-vollständig (*VUP*-vollständig), wenn A für alle DFA-Probleme $\mathcal{S}_G$ mit einer oberen (unteren) Approximation der *SUP*- (*VUP*-) Semantik von G terminiert.
3. *SUP*-akkurat (*VUP*-akkurat), wenn A für alle DFA-Probleme $\mathcal{S}_G$ exakt mit der *SUP*- (*VUP*-) Semantik von G terminiert.

Es gilt

Lemma 8.3.2 (Akkuratheit)

Ein DFA-Algorithmus A ist

1. *SUP*-akkurat gdw A ist *SUP*-korrekt und *SUP*-vollständig.
2. *VUP*-akkurat gdw A ist *VUP*-korrekt und *VUP*-vollständig.

Statt von akkurat sprechen wir auch von optimal.

Definition 8.3.3 (Optimalität)

Ein DFA-Algorithmus A heißt *SUP*-optimal (*VUP*-optimal) gdw A ist *SUP*-akkurat (*VUP*-akkurat).

Beachte: Ausgesprochen kommt es bei *SUP*-optimal auf das harte p an!

Zur Existenz

...korrekter, vollständiger, optimaler DFA-Algorithmen.

Lemma 8.3.4 (Existenz)

Ein DFA-Algorithmus A, der für den Startknoten $\mathbf{s}$ stets die Anfangszusicherung $c_{\mathbf{s}}$ und für alle anderen Programmpunkte stets die Information

1. $\bot$ ($\top$) liefert, ist *SUP*-korrekt (*SUP*-vollständig).
2. $\top$ ($\bot$) liefert, ist *VUP*-korrekt (*VUP*-vollständig).

Offenbar sind die DFA-Algorithmen aus Lemma 8.3.4 nutzlos.

Aufgabe der Informatik ist es, nützlich DFA-Algorithmen zu finden. Als erstes stellt sich damit die Frage:

- Gibt es optimale DFA-Algorithmen?

Diese Frage ist nicht trivial!

...die Definitionen der *SUP*- und *VUP*-Semantik induzieren selbst

- keine effektiven Verfahren

zu ihrer Berechnung (Schleifen in einem nichtdeterministisch interpretierten Flussgraphen führen dazu, dass die Zahl der Pfade zu darin bzw. dahinter liegenden Programmpunkten unendlich ist).

Noch schlechter: *SUP*- und *VUP*-Semantik sind

- unentscheidbar!

Damit besteht keine Hoffnung, DFA-Algorithmen zu finden, die für alle DFA-Probleme korrekt und vollständig sind; Hoffnung besteht allenfalls für DFA-Probleme mit gewissen zusätzlichen Eigenschaften.

Kapitel 8.4

Unentscheidbarkeit der *SUP*- und *VUP*-Semantik

Unentscheidbarkeit der *SUP*-Semantik

Theorem 8.4.1 (Unentscheidbarkeit d. *SUP*-Sem.)

Es gibt keinen Algorithmus A mit der Eigenschaft:

- Eingabe für A ist
  - eine DFA-Spezifikation $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$.
  - Algorithmen zur Berechnung von Schnitt, Test auf Gleichheit und Anwendung monotoner Funktionen auf $\widehat{\mathcal{C}}$.
- Ausgabe von A ist die *SUP*-Semantik von $\mathcal{S}_G$.

(John B. Kam, Jeffrey D. Ullman. Monotone Data Flow Analysis Frameworks. Acta Informatica 7:305-317, 1977)

Beweisidee. Reduktion auf das *Modifizierte Postsche Korrespondenzproblem* (*MPKP*).

(Für *MPKP* s. z.B.: John E. Hopcroft, Jeffrey D. Ullman. Introduction to Automata Theory, Languages and Computation. Addison-Wesley, 1979.)

Unentscheidbarkeit der *VUP*-Semantik

Korollar 8.4.2 (Unentscheidbarkeit d. *VUP*-Sem.)

Es gibt keinen Algorithmus A mit der Eigenschaft:

- Eingabe von A ist
  - eine DFA-Spezifikation $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$.
  - Algorithmen zur Berechnung von Vereinigung, Test auf Gleichheit und Anwendung monotoner Funktionen auf $\widehat{\mathcal{C}}$.
- Ausgabe von A ist die *VUP*-Semantik von $\mathcal{S}_G$.

Anm.: Das *MPK*-Problem ist folgendes: Seien A, B Listen mit je k nichtleeren Zeichenreihen s_i, t_i, $0 \leq i \leq k-1$ über dem Alphabet $\{0,1\}$: $A = \langle s_0, s_1, s_2, \ldots, s_{k-1} \rangle$, $B = \langle t_0, t_1, t_2, \ldots, t_{k-1} \rangle$.

Gibt es eine Indexfolge $i_1, i_2, \ldots, i_r$, so dass die Konkatenationen von $s_0, s_{i_1}, s_{i_2}, \ldots s_{i_r}$ und $t_0, t_{i_1}, t_{i_2}, \ldots, t_{i_r}$ übereinstimmen, d.h.: $s_0 s_{i_1} s_{i_2} \ldots s_{i_r} = t_0 t_{i_1} t_{i_2} \ldots t_{i_r}$?

Für r unbeschränkt, ist *MPKP* unentscheidbar.

Ausblick

...um entscheidbare DFA-Semantiken zu erhalten, schränken wir die Mengen zulässiger

- lokaler (abstrakter) DFA-Semantiken
- DFA-Verbände

ein: Einschränkung auf

- monotone lokale (abstrakte) DFA-Semantiken garantiert Wohldefiniertheit
- vollständige Verbände mit absteigender (aufsteigender) Kettenbedingung garantiert effektive Berechenbarkeit und damit Entscheidbarkeit

der zu den operationellen *SUP*- und *VUP*-Semantiken dualen Fixpunktsemantiken, der sog.

- maximalen Fixpunktsemantik (*MaxFP*)
- minimalen Fixpunktsemantik (*MinFP*)

Ausblick (fgs.)

Durch ihre Berechenbarkeit induzieren *MaxFP*- und *MinFP*-Semantik zwei zueinander duale berechenbare DFA-Problemlösungen, die sog.

- maximale Fixpunktlösung (*MaxFP*)
- minimale Fixpunktlösung (*MinFP*)

von DFA-Problemen.

Dazu sind folgende mathematische Erweiterungen nötig...

Kapitel 8.5

Mathematische Erweiterungen

Absteigende, aufsteigende Kettenbedingung

Definition 8.5.1 (Kettenbedingung)

Ein Verband $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq)$ erfüllt die

1. absteigende Kettenbedingung (engl. descending chain condition), wenn jede absteigende Kette schließlich stationär wird, d.h., für jede Kette $c_1 \sqsupseteq c_2 \sqsupseteq \ldots \sqsupseteq c_n \sqsupseteq \ldots$ gibt es einen Index $m \geq 1$ mit $c_m = c_{m+j}$ für alle $j \in \mathbb{N}$.
2. aufsteigende Kettenbedingung (engl. ascending chain condition), wenn jede aufsteigende Kette schließlich stationär wird, d.h., für jede Kette $c_1 \sqsubseteq c_2 \sqsubseteq \ldots \sqsubseteq c_n \sqsubseteq \ldots$ gibt es einen Index $m \geq 1$ with $c_m = c_{m+j}$ für alle $j \in \mathbb{N}$.

Monotonie, Distributivität und Additivität

...wichtige Eigenschaften von Funktionen auf Verbänden.

Definition 8.5.2 (Monotonie)

Eine Funktion $f : \mathcal{C} \to \mathcal{C}$ auf einem vollständigen Verband $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ heißt:

- monoton gdw $\forall\, c, c' \in \mathcal{C}.\ c \sqsubseteq c' \Rightarrow f(c) \sqsubseteq f(c')$
  (Erhalt der Ordnung von Elementen)

Definition 8.5.3 (Distributivität, Additivität)

Eine Funktion $f : \mathcal{C} \to \mathcal{C}$ auf einem vollständigen Verband $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ heißt:

- distributiv gdw $\forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcap C') = \bigsqcap \{f(c) \mid c \in C'\}$
  (Erhalt größter unterer Schranken)
- additiv gdw $\forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ \ f(\bigsqcup C') = \bigsqcup \{f(c) \mid c \in C'\}$
  (Erhalt kleinster oberer Schranken)

Teilweise Ordnungserhaltung bei Monotonie

Monotonie kann äquivalent über den Erhalt größter unterer und kleinster oberer Schranken charakterisiert werden:

Lemma 8.5.4

Für eine Funktion $f : \mathcal{C} \to \mathcal{C}$ auf einem vollständigen Verband $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ sind äquivalent:

1. f ist monoton.
2. $\forall \emptyset \neq C' \subseteq \mathcal{C}.\ \ f(\bigsqcap C') \sqsubseteq \bigsqcap \{f(c) \mid c \in C'\}$
3. $\forall \emptyset \neq C' \subseteq \mathcal{C}.\ \ f(\bigsqcup C') \sqsupseteq \bigsqcup \{f(c) \mid c \in C'\}$

Zshg. v. Monotonie, Distributivität, Additivität

Lemma 8.5.5

Für eine Funktion $f : \mathcal{C} \to \mathcal{C}$ auf einem vollständigen Verband $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ gilt:

1. f distributiv oder additiv $\Longrightarrow$ f monoton
2. Distributivität und Additivität sind unabhängig voneinander; keine Eigenschaft impliziert die andere.

Kapitel 8.6

Monotone, distributive, additive DFA-Probleme

DFA-Probleme

...mit:

- monotonen
- distributiven (additiven)

DFA-Semantiken über DFA-Verbänden mit:

- absteigender (aufsteigender) Kettenbedingung

besitzen entscheidbare Fixpunktsemantiken und sind deshalb lösbar (s. Kap. 8.8, 8.9) und von praktischem Interesse (s. Kap. 8.13).

Wir legen deshalb fest...

DFA-Semantiken u. -Probleme: Eigenschaften

Definition 8.6.1 (DFA-Semantikeigenschaften)

Eine lokale DFA-Semantik $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ ist monoton/distributiv/additiv gdw alle DFA-Funktionen $[\![\ e\ ]\!]$, $e \in E$, monoton/distributiv/additiv sind.

Definition 8.6.2 (DFA-Problemeigenschaften)

Ein DFA-Problem $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$

- ist monoton/distributiv/additiv gdw die lokale DFA-Semantik $[\![\ ]\!]$ von $\mathcal{S}_G$ monoton/distributiv/additiv ist.
- erfüllt die absteigende/aufsteigende Kettenbedingung gdw der DFA-Verband $\widehat{C}$ von $\mathcal{S}_G$ die absteigende/aufsteigende Kettenbedingung erfüllt.

Beispiel: Verfügbarkeit eines Terms t (1)

Lemma 8.6.3 (DFA-Funktionen)

$$[\![\ ]\!]^t_{av} = \lambda e. \begin{cases} Cst_{\mathbf{wahr}} & \text{falls } Comp^t_e \wedge Transp^t_e \\ Id_{\mathbb{B}} & \text{falls } \neg Comp^t_e \wedge Transp^t_e \\ Cst_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

mit

- $Cst_{\mathbf{wahr}}, Cst_{\mathbf{falsch}} : \mathbb{B} \to \mathbb{B}$ (konstante Funktionen auf $\mathbb{B}$)

  $Cst_{\mathbf{wahr}} =_{df} \lambda b.\ \mathbf{wahr}$
  $Cst_{\mathbf{falsch}} =_{df} \lambda b.\ \mathbf{falsch}$

- $Id_{\mathbb{B}} : \mathbb{B} \to \mathbb{B}$ (Identität auf $\mathbb{B}$)

  $Id_{\mathbb{B}} =_{df} \lambda b.\ b$

Beispiel: Verfügbarkeit eines Terms t (2)

Lemma 8.6.4 (Kettenbedingung)

$\widehat{\mathbb{B}}$ erfüllt die absteigende (und aufsteigende) Kettenbedingung.

Lemma 8.6.5 (Distributivität, Additivität)

$[\![e]\!]^t_{av}$, $e \in E$, ist distributiv (und additiv) (und deshalb auch monoton).

Beweis. Unmittelbar mit Lemma 8.6.3.

Korollar 8.6.6 (Verfügbarkeit eines Terms t)

Das durch $\mathcal{S}^{av,t}_G = (\widehat{\mathbb{B}}, [\![\ ]\!]^t_{av}, b_{\mathbf{s}})$ gegebene DFA-Problem ist distributiv (und additiv) und erfüllt die absteigende (und aufsteigende) Kettenbedingung.

Kapitel 8.7

Denotationelle globale DFA-Semantiken: Fixpunktsemantiken

Kapitel 8.7.1

Maximale Fixpunktsemantik

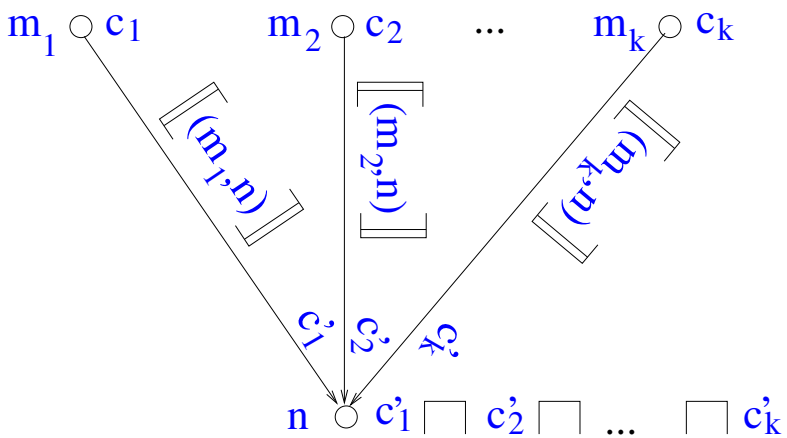
Die maximale Fixpunktsemantik (*MaxFP*)

Sei $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ eine DFA-Spezifikation.

Gleichungssystem 8.7.1.1 (*MaxFP*-Gleichungssyst.)

$$inf(n) \;=\; \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcap \{\, [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \,\} & \text{sonst} \end{cases}$$

Veranschaulichung des *MaxFP*-Ansatzes (für $n \neq \mathbf{s}$):



Die *MaxFP*-Semantik

Lemma 8.7.1.2 (Fixpunktexistenz)

Ist $[\![\ ]\!]$ monoton, so hat Gleichungssystem 8.7.1.1 eine eindeutig bestimmte größte Lösung, die wir mit

$$\nu\text{-}inf_{c_s} : N \to \mathcal{C}$$

bezeichnen.

Definition 8.7.1.3 (*MaxFP*-Semantik)

Die von einer monotonen lokalen abstrakten Semantik $[\![\ ]\!]$ induzierte (deterministische) *MaxFP*-Semantik von G ist definiert durch:

$$[\![\ ]\!]_{MaxFP} : \mathcal{C} \to N \to \mathcal{C}$$

$$[\![\ ]\!]_{MaxFP} =_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \nu\text{-}inf_c(n)$$

Lemma 8.7.1.4 (*MaxFP*-Wohldefiniertheit)

Die *MaxFP*-Semantik monotoner lokaler abstrakter Semantiken für G ist wohldefiniert.

Kapitel 8.7.2

Minimale Fixpunktsemantik

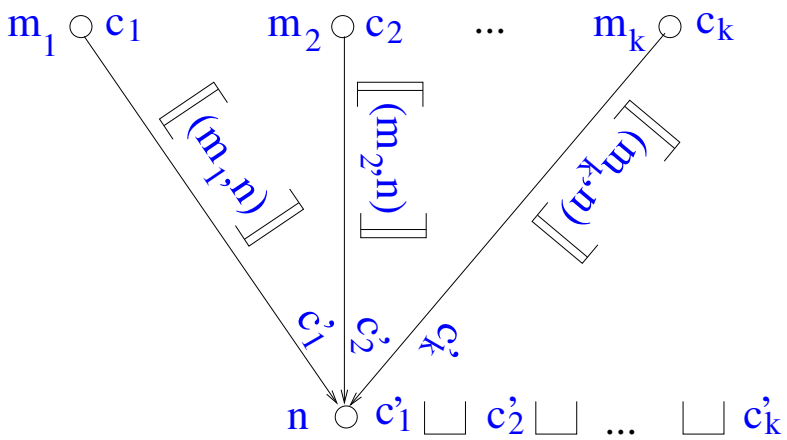
Die minimale Fixpunktsemantik (*MinFP*)

Sei $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ eine DFA-Spezifikation.

Gleichungssystem 8.7.2.1 (*MinFP*-Gleichungssyst.)

$$inf(n) \quad = \quad \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcup \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

Veranschaulichung des *MinFP*-Ansatzes (für $n \neq \mathbf{s}$):



Die *MinFP*-Semantik

Lemma 8.7.2.2 (Fixpunktexistenz)

Ist $[\![\ ]\!]$ monoton, so hat Gleichungssystem 8.7.2.1 eine eindeutig bestimmte kleinste Lösung, die wir mit

$$\mu\text{-}inf_{c_s}(n),\ \ n \in N$$

bezeichnen.

Definition 8.7.2.3 (*MinFP*-Semantik)

Die von einer monotonen lokalen abstrakten Semantik $[\![\ ]\!]$ induzierte (deterministische) *MinFP*-Semantik von G ist definiert durch:

$$[\![\ ]\!]_{MinFP} : \mathcal{C} \to N \to \mathcal{C}$$

$$[\![\ ]\!]_{MinFP} =_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \mu\text{-}inf_c(n)$$

Lemma 8.7.2.4 (*MinFP*-Wohldefiniertheit)

Die *MinFP*-Semantik monotoner lokaler abstrakter Semantiken für G ist wohldefiniert.

Kapitel 8.8

Entscheidbarkeit der *MaxFP*- und *MinFP*-Semantik

Die *MaxFP*- und *MinFP*-Semantik

...eines Flussgraphen sind aufgrund von *MaxFP*-Gleichungssystem 8.7.1.1 und *MinFP*-Gleichungssystem 8.7.2.1 praktisch relevant, da sie in generischer Weise ein

- iteratives Berechnungsverfahren (Algorithmus 8.8.1.1)

induzieren, das ihre größte und kleinste Lösung approximativ und unter gewissen zusätzlichen Voraussetzungen exakt zu berechnen erlaubt, mithin die *MaxFP*- und *MinFP*-Semantiken selbst.

Kapitel 8.8.1

Generischer Fixpunktalgorithmus

Der generische Fixpunktalgorithmus 8.8.1.1 (1)

Eingabe: Eine DFA-Spezifikation $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$.

Ausgabe: Nach Terminierung des Algorithmus (s. Terminationstheorem 8.8.2.1), enthält Variable $inf[n]$ die *MaxFP*-Lösung von $\mathcal{S}_G$ am Knoten n.

Zusätzlich gilt (s. Sicherheitstheorem 8.9.1 und Koinzidenztheorem 8.9.2): Ist $[\![\ ]\!]$

- distributiv: $inf[n]$ enthält die
- monoton: $inf[n]$ enthält eine untere Approximation der

SUP-Lösung von $\mathcal{S}_G$ am Knoten n.

Anmerkung: Die Variable *workset* dient zur Steuerung des iterativen Berechnungsvorgangs. Sie enthält diejenigen Knoten von G, deren Benennung kürzlich aktualisiert worden (d.h. kleiner geworden) ist, was sich entsprechend auf die ihrer direkten (und indirekten) Nachfolgerknoten auswirken kann.

Der generische Fixpunktalgorithmus 8.8.1.1 (2)

(Prolog: Initialisierung von inf und $workset$)
FORALL $n \in N \backslash \{\mathbf{s}\}$ DO $inf[n] := \top$ OD;
$inf[\mathbf{s}] := c_{\mathbf{s}}$;
$workset := N$;
(Hauptschleife: Die iterative Fixpunktberechnung)
WHILE $workset \neq \emptyset$ DO
    CHOOSE $m \in workset$;
        $workset := workset \backslash \{ m \}$;
        (Aktualisierung der Benennungen der Nachfolger von m)
        FORALL $n \in succ(m)$ DO
            $meet := [\![(m, n)]\!](inf[m]) \sqcap inf[n]$;
            IF $inf[n] \sqsupset meet$
                THEN
                    $inf[n] := meet$;
                    $workset := workset \cup \{ n \}$
            FI
        OD ESOOHC OD.

Kapitel 8.8.2

Effektivität, Terminierung

Effektivität, Terminierung

Theorem 8.8.2.1 (Effektivität, Terminierung)

Fixpunktalgorithmus 8.8.1.1 terminiert mit der

1. *MaxFP*-Semantik von $\mathcal{S}_G$, wenn gilt:
   - 1.1 $[\![\ ]\!]$ ist monoton.
   - 1.2 $\widehat{\mathcal{C}}$ erfüllt die absteigende Kettenbedingung.
2. *MinFP*-Semantik von $\mathcal{S}_G$, wenn gilt:
   - 2.1 $[\![\ ]\!]$ ist monoton.
   - 2.2 $\widehat{\mathcal{C}^{gst}}$ erfüllt die absteigende Kettenbedingung, wobei

$$\widehat{\mathcal{C}^{gst}} =_{df} (\mathcal{C}, \sqsupseteq, \sqcup, \sqcap, \top, \bot)$$

den auf den 'Kopf gestellten' (oder: 'gestürzten') Verband $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ bezeichnet.

Kapitel 8.9

MaxFP- und *MinFP*-Semantik als zueinander duale berechenbare DFA-Problemlösungen

Berechenbare DFA-Problemlösungen

...Fixpunktalgorithmus 8.8.1.1 und Terminierungstheorem 8.8.2.1 legen folgende Festlegung nahe:

Definition 8.9.1 (Berechenbare Lsg. eines DFA-P.)

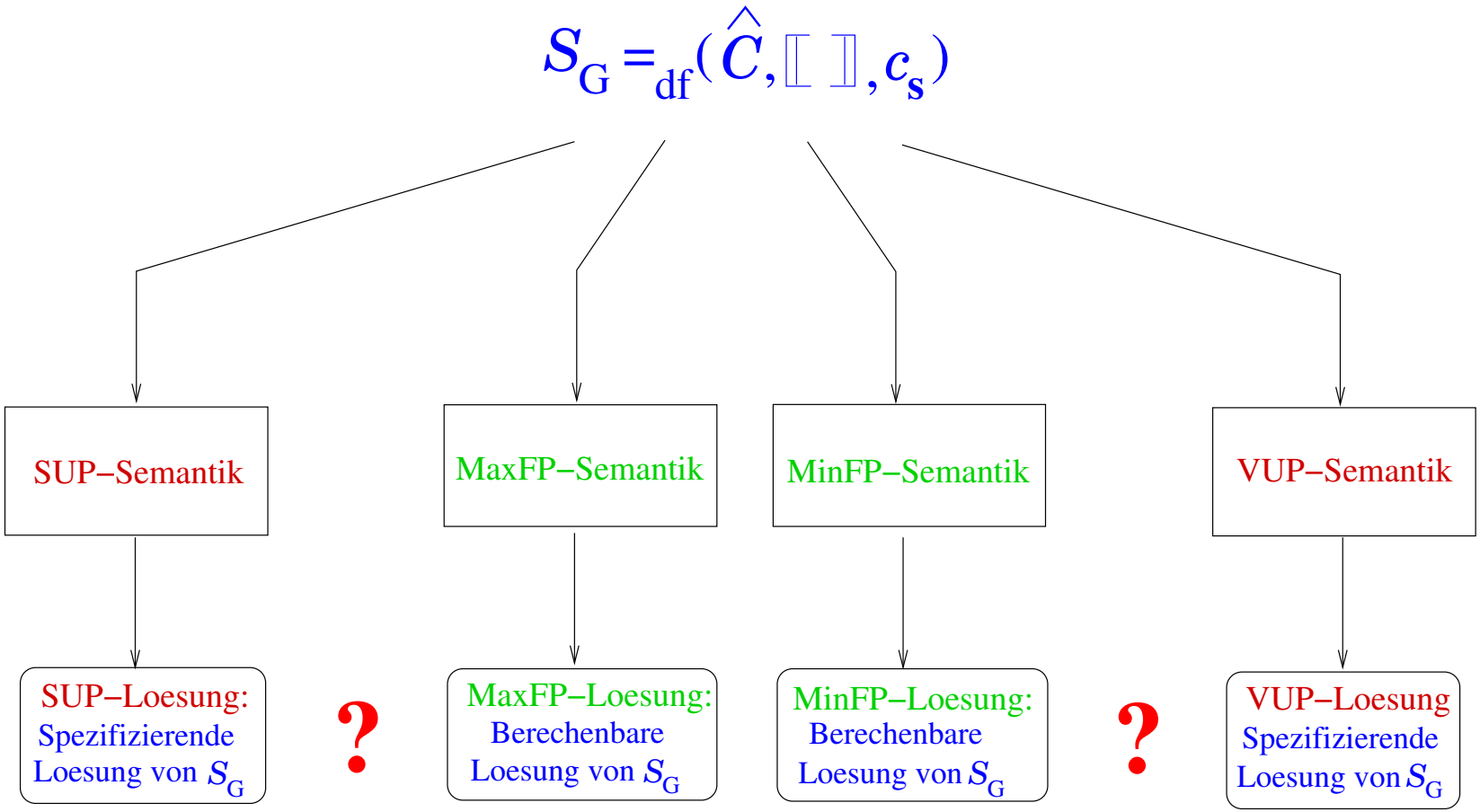
Die *MaxFP*- und *MinFP*-Semantik eines Flussgraphen definieren zwei zueinander duale berechenbare Lösungen eines DFA-Problems, seine sog.:

1. *MaxFP*-Lösung
2. *MinFP*-Lösung

Damit stellt sich die Frage nach dem Verhältnis, in dem die spezifierenden und berechenbaren Lösungen eines DFA-Problems zueinander stehen...

SUP/*MaxFP*- und *VUP*/*MinFP*-Semantik

...die Frage nach ihrer Beziehung:



Kapitel 8.10

Sicherheit, Koinzidenz

Sicherheit: Korrekth. v. *MaxFP*-, *MinFP*-Sem.

Sei $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ eine DFA-Spezifikation.

Theorem 8.10.1 (Sicherheit: Korrektheit)

1. Die *MaxFP*-Semantik von $\mathcal{S}_G$ ist eine sichere (d.h. untere) Approximation der *SUP*-Semantik von $\mathcal{S}_G$:

$$[\![\ ]\!]_{MaxFP} \sqsubseteq [\![\ ]\!]_{SUP}$$

2. Die *MinFP*-Semantik von $\mathcal{S}_G$ ist eine sichere (d.h. obere) Approximation der *VUP*-Semantik von $\mathcal{S}_G$:

$$[\![\ ]\!]_{MinFP} \sqsupseteq [\![\ ]\!]_{VUP}$$

wenn die lokale DFA-Semantik $[\![\ ]\!]$ monoton ist.

Koinzidenz: Akkurath. v. *MaxFP*-, *MinFP*-Sem.

Sei $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ eine DFA-Spezifikation.

Theorem 8.10.2 (Koinzidenz: Akkuratheit)

1. *MaxFP*- und *SUP*-Semantik von $\mathcal{S}_G$ stimmen überein (sind koinzident):

$$[\![\ ]\!]_{MaxFP} = [\![\ ]\!]_{SUP}$$

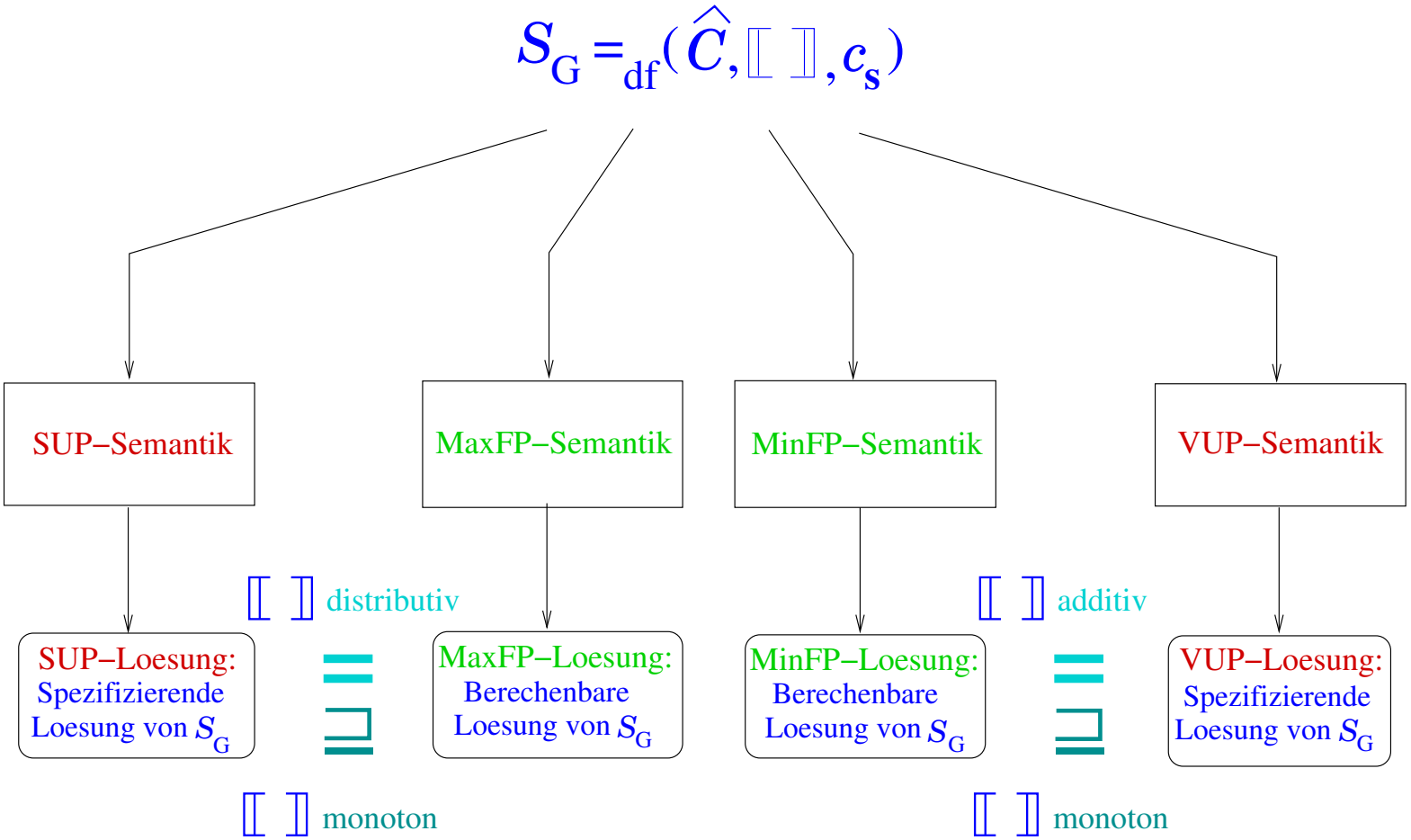
2. *MinFP*- und die *VUP*-Semantik von $\mathcal{S}_G$ stimmen überein (sind koinzident):

$$[\![\ ]\!]_{MinFP} = [\![\ ]\!]_{VUP}$$

wenn die lokale DFA-Semantik $[\![\ ]\!]$ distributiv bzw. additiv ist.

SUP/*MaxFP*- und *VUP*/*MinFP*-Semantik

...und die Antwort auf die Frage nach ihrer Beziehung:



Korrektheit von Fixpunktalgorithmus 8.8.1.1

Korollar 8.10.3 (*SUP*- (*VUP*-) Korrektheit)

Fixpunktalgorithmus 8.8.1.1 ist

- *SUP*- (*VUP*-) korrekt

für $\mathcal{S}_G$, d.h. er terminiert mit einer unteren (oberen) Approximation der *SUP*- (*VUP*-) Semantik von $\mathcal{S}_G$, wenn gilt:

1. $[\![\ ]\!]$ ist monoton.
2. $\widehat{\mathcal{C}}$ erfüllt die absteigende (aufsteigende) Kettenbedingung.

Akkuratheit von Fixpunktalgorithmus 8.8.1.1

Korollar 8.10.4 (*SUP*- (*VUP*-) Akkuratheit)

Fixpunktalgorithmus 8.8.1.1 ist

- *SUP*- (*VUP*-) akkurat

für $\mathcal{S}_G$, d.h. er terminiert mit der *SUP*- (*VUP*-) Semantik von $\mathcal{S}_G$, wenn gilt:

1. $[\![\ ]\!]$ ist distributiv (additiv).
2. $\widehat{\mathcal{C}}$ erfüllt die absteigende (aufsteigende) Kettenbedingung.

Kapitel 8.11

Analyseszenario, Gesamtbild: Korrektheit, Vollständigkeit für ϕ

Gesamtbild: Korrektheit, Vollständigkeit für ϕ

Analyseszenario:

- Sei ϕ eine interessierende Programmeigenschaft (z.B. die Verfügbarkeit eines Terms, die Lebendigkeit einer Variable, etc.).
- Sei $\mathcal{S}_G^\phi$ eine für ϕ entwickelte DFA-Spezifikation.

Definition 8.11.1 (Korrektheit von $\mathcal{S}_G^\phi$ für ϕ)

$\mathcal{S}_G^\phi$ ist *SUP*-korrekt (*VUP*-korrekt) für ϕ, wenn gilt: Zeigt die *SUP*-Semantik (*VUP*-Semantik) von $\mathcal{S}_G^\phi$ die Gültigkeit von ϕ an, so ist ϕ tatsächlich gültig.

Definition 8.11.2 (Vollständigkeit von $\mathcal{S}_G^\phi$ für ϕ)

$\mathcal{S}_G^\phi$ ist *SUP*-vollständig (*VUP*-vollständig) für ϕ, wenn gilt: Ist ϕ gültig, so zeigt die *SUP*-Semantik (*VUP*-Semantik) von $\mathcal{S}_G^\phi$ die Gültigkeit von ϕ auch an.

Informelle Interpretation

...von Korrektheit, Vollständigkeit einer DFA-Spezifikation $\mathcal{S}_G^\phi$ für eine Eigenschaft ϕ:

- *SUP*-Korrektheit von $\mathcal{S}_G^\phi$ bedeutet: $[\![\ ]\!]_{SUP}$ 'impliziert' ϕ.
- *SUP*-Vollständigkeit v. $\mathcal{S}_G^\phi$ bedeutet: ϕ 'impliziert' $[\![\ ]\!]_{SUP}$.

und

- *VUP*-Korrektheit von $\mathcal{S}_G^\phi$ bedeutet: ϕ 'impliziert' $[\![\ ]\!]_{VUP}$.
- *VUP*-Vollständigkeit v. $\mathcal{S}_G^\phi$ bedeutet: $[\![\ ]\!]_{VUP}$ 'impliziert' ϕ.

Korrektheit und Vollständigkeit für ϕ

...implizieren informell:

Ist $\mathcal{S}_G^\phi$ *SUP-* (*VUP-*) korrekt **und** vollständig für ϕ, bedeutet das:

Wir berechnen mit der *MaxFP-* (*MinFP-*) Semantik von $\mathcal{S}_G^\phi$

- die interessierende Eigenschaft,
- die ganze interessierende Eigenschaft,
- und nur die interessierende Eigenschaft.

Mit anderen Worten, wir berechnen

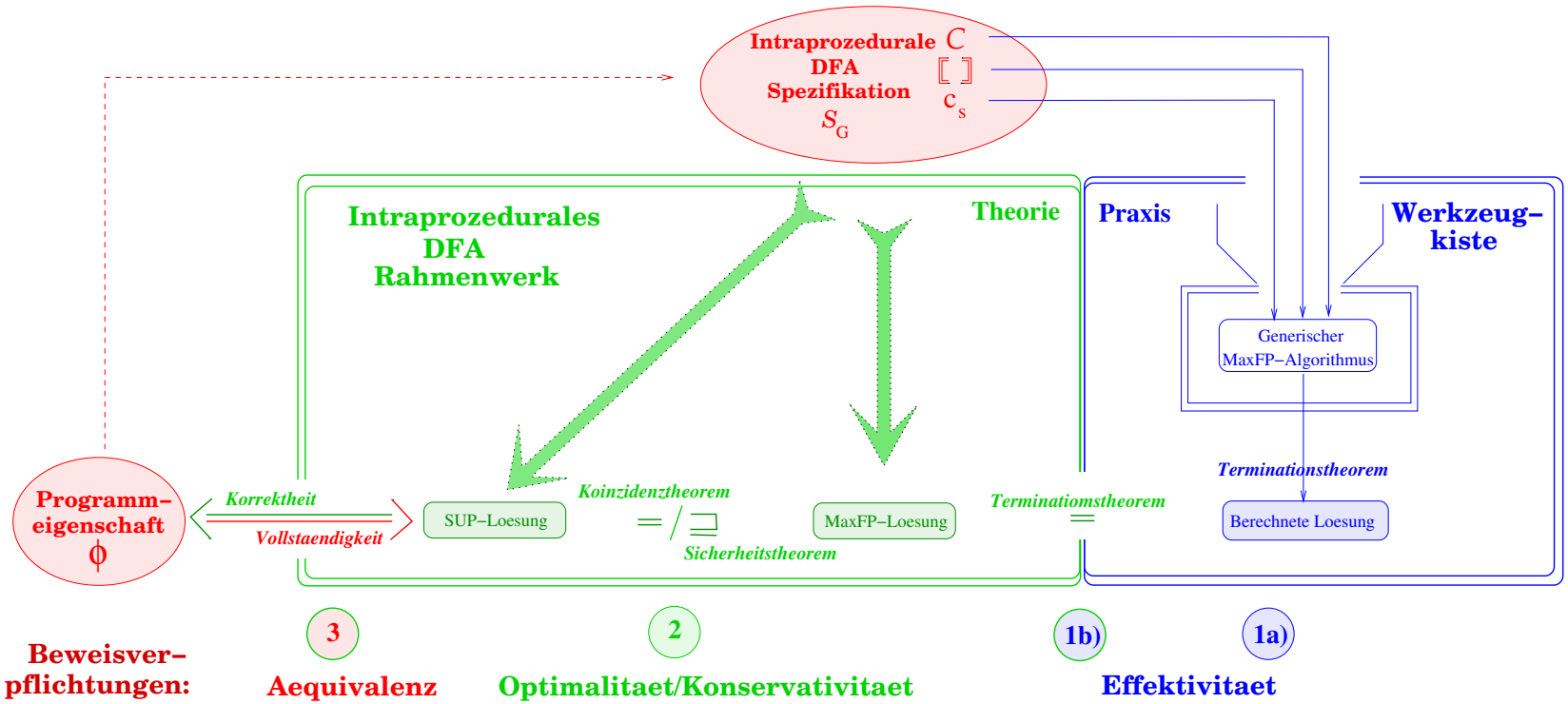
- die interessierende Eigenschaft ϕ präzise!

Kapitel 8.12

Datenflussanalyse in Rahmenwerk- und Werkzeugkistensicht

SUP / *MaxFP*-Datenflussanalyse

...in ganzheitlicher Rahmenwerk- und Werkzeugkistensicht:



(analog für *VUP* / *MinFP*-Datenflussanalyse)

In der Praxis

...ist die Arbeit mit Rahmenwerk und Werkzeugkiste ein dreistufiger Prozess:

1. Wahl der interessierenden Programmeigenschaft

   Wähle die interessierende Programmeigenschaft ϕ (z.B. die Verfügbarkeit eines Terms, die Lebendigkeit einer Variable, etc.) und definiere ϕ formal.

2. Entwicklung einer DFA-Spezifikation

   Entwickle eine DFA-Spezifikation $\mathcal{S}_G^\phi = (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ für ϕ.

3. Erbringung v. Beweisverpflichtungen, Erhalt v. Garantien

   Erbringe eine feststehende Reihe von Beweisverpflichtungen über die Komponenten von $\mathcal{S}_G^\phi$ und die Beziehung der *SUP*- (*VUP*-) Lösung und ϕ und erhalte Garantien, dass die *MaxFP*- (*MinFP*-) Lösung korrekt (d.h. approximierend) oder sogar korrekt und vollständig (d.h. akkurat) für ϕ ist.

Beweisverpflichtungen, implizierte Garantien (1)

Beweisverpflichtungen und Garantien im Detail:

- Beweisverpflichtungen 1a), 1b): Absteigende (aufsteigende) Kettenbedingung für $\widehat{\mathcal{C}}$, Monotonie für $[\![\ ]\!]$

  Garantien:

  - Effektivität: Terminierung von Algorithmus 8.8.1.1 mit der *MaxFP*- (*MinFP*-) Semantik von $\mathcal{S}_G^\phi$.
  - Korrektheit: Die *MaxFP*- (*MinFP*-) Lösung von $\mathcal{S}_G^\phi$ ist *SUP*- (*VUP*-) korrekt.

- Beweisverpflichtung 2): Distributivität (Additivität) für $[\![\ ]\!]$

  Garantie:

  - Akkuratheit: Die *MaxFP*- (*MinFP*-) Semantik von $\mathcal{S}_G^\phi$ ist *SUP*- (*VUP*-) akkurat.

Beweisverpflichtungen, implizierte Garantien (2)

- Beweisverpflichtung 3): Äquivalenz von $SUP_{\mathcal{S}_G^\phi}$ ($VUP_{\mathcal{S}_G^\phi}$) und ϕ

  Garantien:

  - Wann immer die *SUP*-Lösung von $\mathcal{S}_G^\phi$ die Gültigkeit von ϕ anzeigt, dann ist ϕ gültig: Korrektheit von $\mathcal{S}_G^\phi$ für ϕ.

    $\rightsquigarrow$ Wir berechnen die interessierende Programmeigenschaft und nur die interessierende Programmeigenschaft.
  - Wann immer ϕ gültig ist, dann zeigt die *SUP*-Lösung von $\mathcal{S}_G^\phi$ dies an: Vollständigkeit von $\mathcal{S}_G^\phi$ für ϕ.

    $\rightsquigarrow$ Wir berechnen die ganze interessierende Programmeigenschaft.
  - Analog und entsprechend für die *VUP*-Lösung von $\mathcal{S}_G^\phi$.

  Garantie von Korrektheit und Vollständigkeit für ϕ:

  - Wir berechnen ϕ präzise!

Kapitel 8.13

Anwendungen: Zwei kanonische Beispiele distributiver und monotoner DFA-Probleme

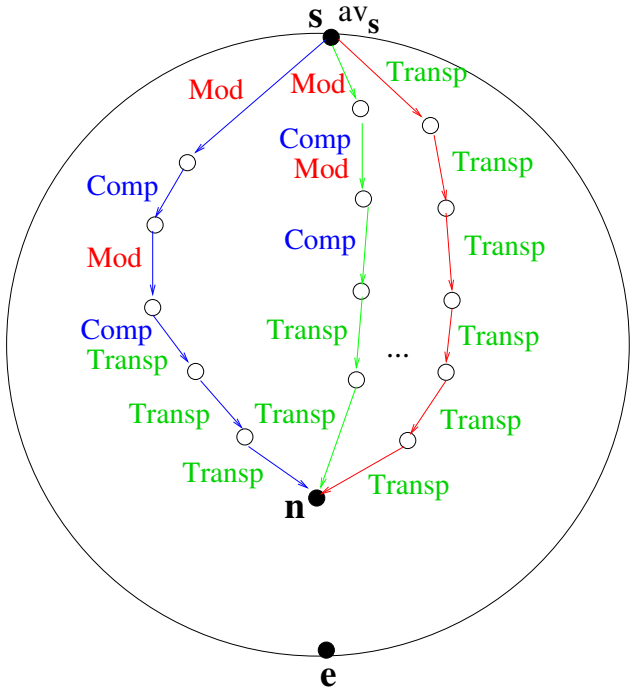
Kapitel 8.13.1

Verfügbare Ausdrücke, ein distributives DFA-Problem

Ein Term t

...heißt verfügbar am Knoten **n**, wenn t auf jedem Pfad vom Programmanfang **s** zu **n** berechnet wird und anschließend bis zum Erreichen von **n** keine seiner Operandenvariablen einen Wert zugewiesen erhält.

Veranschaulichung:



...kanonisches Beispiel eines distributiven DFA-Problems.

Arbeitsplan

...in der Folge spezifizieren wir fünf Varianten des Verfügbarkeitsproblems (engl. availability) von Termen für

- Einzelterme mittels
  1. Boolescher Verband (Kap. 8.13.1.1)
- Termmengen mittels
  2. Potenzmengenverband (Kap. 8.13.1.2)
  3. Kreuzproduktverband (Kap. 8.13.1.3)
  4. Bitvektorverband (Kap. 8.13.1.4)
  5. Gen/Kill-Formulierung (Kap. 8.13.1.5)

Im Vorbeigehen illustrieren wir so:

- die Klasse sog. Bitvektor- (oder: Gen/Kill-) DFA-Probleme mit Verfügbarkeit als typischem Vertreter.
- die Verwendung verschiedener DFA-Verbände.

Vorbereitungen

Sei $\iota_e \equiv x := exp$ (bzw. $\iota_e \equiv exp$) die Instruktion (bzw. Bedingung) an Kante e, t ein Term.

Lokale Prädikate (für Kanten)

- $Comp_e^t$

  ...ist **wahr**, wenn t an Kante ι_e berechnet wird (d.h. t ist ein Teilterm des rechtsseitigen Ausdrucks exp von ι_e), **falsch** sonst.

- Mod_e^t

  ...ist **wahr**, wenn t an Kante ι_e modifiziert wird (d.h. ι_e weist einem Operanden von t einen (neuen) Wert zu), **falsch** sonst.

- $Transp_e^t =_{df} \neg Mod_e^t$

  ...ist **wahr**, wenn e transparent für t ist (d.h. ι_e weist keinem Operanden von t einen Wert zu), **falsch** sonst.

Kapitel 8.13.1.1

Verfügbarkeit für Einzelterme: Boolesche Verbandsformulierung

V1: Verfügbarkeit für einen einzelnen Term t

Boolescher Verband

- $\widehat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, \leq, \mathbf{falsch}, \mathbf{wahr})$

  Boolescher Verband der Wahrheitswerte: Kleinstes Element **falsch**; größtes Element **wahr**; **falsch** $\leq$ **wahr** als Ordnungsrelation; logisches $\wedge$, logisches $\vee$ als Schnitt- bzw. Vereinigungsoperation.

Hilfsfunktionen

- $Cst_{\mathbf{wahr}}, Cst_{\mathbf{falsch}} : \mathbb{B} \to \mathbb{B}$ konstante Funktionen auf $\mathbb{B}$

  $Cst_{\mathbf{wahr}} =_{df} \lambda b.\ \mathbf{wahr}$
  $Cst_{\mathbf{falsch}} =_{df} \lambda b.\ \mathbf{falsch}$

- $Id_{\mathbb{B}} : \mathbb{B} \to \mathbb{B}$ Identität auf $\mathbb{B}$

  $Id_{\mathbb{B}} =_{df} \lambda b.\ b$

V1: Spezifizieren der DFA

DFA-Spezifikation

- DFA-Verband
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $(\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr}) = \widehat{\mathbb{B}}$
- DFA-Semantik
  $[\![\ ]\!]^t_{av} : E \to (\ \mathbb{B} \to \mathbb{B}\ )$
  $\forall\, e \in E\ [\![\ e\ ]\!]^t_{av} =_{df} \lambda\, b.\ (b \vee \mathit{Comp}^t_e) \wedge \mathit{Transp}^t_e$
- Anfangszusicherung
  $b_{\mathbf{s}} \in \mathbb{B}$

Verfügbarkeitsspezifikation für t

- Spezifikation: $\mathcal{S}^{av,t}_G = (\widehat{\mathbb{B}}, [\![\ ]\!]^t_{av}, b_{\mathbf{s}})$

V1: Erbringen der Beweisverpflichtungen

Lemma 8.13.1.1.1 (Charakterisierung)

$$\forall\, e \in E.\ [\![\, e \,]\!]^t_{av} = \begin{cases} Cst_{\mathbf{wahr}} & \text{falls } Comp^t_e \wedge Transp^t_e \\ Id_{\mathbb{IB}} & \text{falls } \neg Comp^t_e \wedge Transp^t_e \\ Cst_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

Lemma 8.13.1.1.2 (Absteigende Kettenbedingung)

$\widehat{\mathbb{IB}}$ erfüllt die absteigende Kettenbedingung.

Lemma 8.13.1.1.3 (Distributivität)

$[\![\ ]\!]^t_{av}$ ist distributiv.

Korollar 8.13.1.1.4 (Monotonie)

$[\![\ ]\!]^t_{av}$ ist monoton.

V1: Einsammeln der *SUP*-Semantikgarantien

...für Terminierung, Akkuratheit.

Theorem 8.13.1.1.5 (*MaxFP*-Terminierung)

Angewendet auf $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\ ]\!]_{av}^t, b_{\mathbf{s}})$ terminiert Algorithmus 8.8.1.1 mit der *MaxFP*-Semantik von $\mathcal{S}_G^{av,t}$.

Beweis. Folgt unmittelbar mit Lemma 8.13.1.1.2, Korollar 8.13.1.1.4 und Terminierungstheorem 8.8.2.1.

Theorem 8.13.1.1.6 (*SUP*-Akkuratheit)

Angewendet auf $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\ ]\!]_{av}^t, b_{\mathbf{s}})$ ist Algorithmus 8.8.1.1 *SUP*-akkurat für $\mathcal{S}_G^{av,t}$ (d.h. terminiert mit der *SUP*-Semantik von $\mathcal{S}_G^{av,t}$).

Beweis. Folgt unmittelbar mit Lemma 8.13.1.1.3, Koinzidenztheorem 8.10.2 und Terminierungstheorem 8.8.2.1.

V1: Analog für *VUP*-Semantikgarantien

Analog zu Lemma 8.13.1.1.2, 8.13.1.1.3 gilt offenbar auch:

1. $\widehat{\mathbb{IB}}$ erfüllt die aufsteigende Kettenbedingung.
2. $[\![\ ]\!]^t_{av}$ ist additiv.

Damit gilt analog zu Theorem 8.13.1.1.6 auch für die *VUP*-Semantik das entsprechende Terminierungs- und Akkuratheitsresultat:

Theorem 8.13.1.1.7 (*VUP*-Terminier., -Akkuratheit)

Angewendet auf das duale DFA-Problem von $\mathcal{S}^{av,t}_G$ mit gestürztem Verband $\widehat{\mathbb{IB}}$ ist Algorithmus 8.8.1.1 *VUP*-akkurat (d.h. terminiert mit der *VUP*-Semantik des dualen Problems).

Kapitel 8.13.1.2

Verfügbarkeit für endliche Termmengen: Potenzmengenverbandformulierung

V2: Verfügbarkeit für endliche Termmenge T

Potenzmengenverband

- $\widehat{\mathcal{P}(T)} =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T)$

  Potenzmengenverband von T: Kleinstes Element $\emptyset$; größtes Element T; Teilmengenrelation $\subseteq$ als Ordnungsrelation; Mengendurchschnitt $\cap$, Mengenvereinigung $\cup$ als Schnitt- bzw. Vereinigungsoperation.

V2: Spezifizieren der DFA

DFA-Spezifikation

- DFA-Verband
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T) = \widehat{\mathcal{P}(T)}$
- DFA-Semantik
  $[\![\ ]\!]^T_{av} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$
  $\forall e \in E\ [\![e]\!]^T_{av} =_{df}$
  $\lambda T'.\ \{t \in T \mid (t \in T' \vee Comp^t_e) \wedge Transp^t_e\}$
- Anfangszusicherung
  $T_{\mathbf{s}} \in \mathcal{P}(T)$

Verfügbarkeitsspezifikation für T

- Spezifikation: $\mathcal{S}^{av,T}_G = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]^T_{av}, T_{\mathbf{s}})$

V2: Erbringen der Beweisverpflichtungen

Lemma 8.13.1.2.1 (Absteigende Kettenbedingung)

$\widehat{\mathcal{P}(T)}$ erfüllt die absteigende Kettenbedingung.

Lemma 8.13.1.2.2 (Distributivität)

$[\![\ ]\!]^T_{av}$ ist distributiv.

Korollar 8.13.1.2.3 (Monotonie)

$[\![\ ]\!]^T_{av}$ ist monoton.

V2: Einsammeln der *SUP*-Semantikgarantien

...für Terminierung, Akkuratheit.

Theorem 8.13.1.2.4 (*MaxFP*-Terminierung)

Angewendet auf $\mathcal{S}_G^{av,T} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av}^T, T_{\mathbf{s}})$ terminiert Algorithmus 8.8.1.1 mit der *MaxFP*-Semantik von $\mathcal{S}_G^{av,T}$.

Beweis. Folgt unmittelbar mit Lemma 8.13.1.2.1, Korollar 8.13.1.2.3 und Terminierungstheorem 8.8.2.1.

Theorem 8.13.1.2.5 (*SUP*-Akkuratheit)

Angewendet auf $\mathcal{S}_G^{av,T} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av}^T, T_{\mathbf{s}})$ ist Algorithmus 8.8.1.1 *SUP*-akkurat für $\mathcal{S}_G^{av,T}$ (d.h. terminiert mit der *SUP*-Semantik von $\mathcal{S}_G^{av,T}$).

Beweis. Folgt unmittelbar mit Lemma 8.13.1.2.2, Koinzidenztheorem 8.10.2 und Terminierungstheorem 8.8.2.1.

V2: Analog für *VUP*-Semantikgarantien

Analog zu Lemma 8.13.1.2.1, 8.13.1.2.2 gilt auch:

1. $\widehat{\mathcal{P}(T)}$ erfüllt die aufbsteigende Kettenbedingung.
2. $[\![\ ]\!]^T_{av}$ ist additiv.

Damit gilt analog zu Theorem 8.13.1.2.5 auch für die *VUP*-Semantik das entsprechende Terminierungs- und Akkuratheitsresultat:

Theorem 8.13.1.2.6 (*VUP*-Terminier., -Akkuratheit)

Angewendet auf das duale DFA-Problem von $\mathcal{S}^{av,T}_G$ mit gestürztem Verband $\widehat{\mathcal{P}(T)}$ ist Algorithmus 8.8.1.1 *VUP*-akkurat (d.h. terminiert mit der *VUP*-Semantik des dualen Problems).

Kapitel 8.13.1.3

Verfügbarkeit für endliche Termmengen: Kreuzproduktverbandformulierung

V3: Verfügbarkeit für endliche Termmenge T

Boolescher Kreuzproduktverband

– $\widehat{\mathbb{B}^n} =_{df} (\mathbb{B}^n, \wedge_{pw}, \vee_{pw}, <_{pw}, \overline{\textbf{falsch}}, \overline{\textbf{wahr}})$

n-stelliger Kreuzproduktverband über $\mathbb{B}$: Kleinstes Element $\overline{\textbf{falsch}} =_{df} (\textbf{falsch}, \ldots, \textbf{falsch}) \in \mathbb{B}^n$; größtes Element $\overline{\textbf{wahr}} =_{df} (\textbf{wahr}, \ldots, \textbf{wahr}) \in \mathbb{B}^n$; $<_{pw}$ punktweise Ausdehnung von $<$ von $\widehat{\mathbb{B}}$ auf $\widehat{\mathbb{B}^n}$ als Ordnungsrelation; $\wedge_{pw}$, $\vee_{pw}$ punktweise Ausdehnung von logischem $\wedge$ bzw. logischem $\vee$ von $\widehat{\mathbb{B}}$ auf $\widehat{\mathbb{B}^n}$ als Schnitt- bzw. Vereinigungsoperation.

V3: Einführung von Hilfsfunktionen

Hilfsfunktionen

– $ix : T \to \{1, \ldots, n\},\ ix^{-1} : \{1, \ldots, n\} \to T$

Bijektive Indexabbildungen, die jedem Term $t \in T$ eindeutig eine Zahl in $\{1, \ldots, n\}$ zuordnen und umgekehrt.

Das $ix(t)$-te Element eines Tupels

$$\bar{b} = (b_1, \ldots, b_{ix(t)}, \ldots, b_n) \in \mathbb{B}^n$$

ist die für t in $\bar{b}$ abgelegte Verfügbarkeitsinformation.

– $\cdot\downarrow_\cdot : \mathbb{B}^n \to \{1, \ldots, n\} \to \mathbb{B}$

Projektionsfunktion, die das i-te Element eines Tupels $\bar{b} \in \mathbb{B}^n$ liefert:

$$\forall\, i \in \{1, \ldots, n\}.\ \bar{b}\downarrow_i =_{df} b_i$$

V3: Spezifizieren der DFA

DFA-Spezifikation (Kreuzproduktverband (kpv))

- DFA-Verband

$$\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$$
$$(\mathbb{B}^n, \wedge_{pw}, \vee_{pw}, <_{pw}, \overline{\textbf{falsch}}, \overline{\textbf{wahr}}) = \widehat{\mathbb{B}^n}$$

- DFA-Semantik

$$[\![\ ]\!]^T_{av,kpv} : E \to (\ \mathbb{B}^n \to \mathbb{B}^n\ )$$
$$\forall\, e \in E\ [\![\ e\ ]\!]^T_{av,kpv}(\bar{b}) =_{df} \lambda\, \bar{b}.\ \bar{b'}$$
$$\text{mit } \forall\, i \in \{1, \ldots, n\}.\ \bar{b'}{\downarrow_i} =_{df}$$
$$(\bar{b}{\downarrow_i} \vee Comp_e^{ix^{-1}(i)}) \wedge Transp_e^{ix^{-1}(i)}$$

- Anfangszusicherung

$$\bar{b}_{\mathbf{s}} \in \mathbb{B}^n$$

Verfügbarkeitsspezifikation für T

- Spezifikation: $\mathcal{S}_G^{av,T,kpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]^T_{av,kpv}, \bar{b}_{\mathbf{s}})$

V3: Erbringen der Beweisverpflichtungen

Lemma 8.13.1.3.1 (Absteigende Kettenbedingung)

$\widehat{\mathbb{B}^n}$ erfüllt die absteigende Kettenbedingung.

Lemma 8.13.1.3.2 (Distributivität)

$[\![\ ]\!]^T_{av,kpv}$ ist distributiv.

Korollar 8.13.1.3.3 (Monotonie)

$[\![\ ]\!]^T_{av,kpv}$ ist monoton.

V3: Einsammeln der *SUP*-Semantikgarantien

...für Terminierung, Akkuratheit.

Theorem 8.13.1.3.4 (*MaxFP*-Terminierung)

Angewendet auf $\mathcal{S}_G^{av,T,kpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]^T_{av,kpv}, \bar{b}_{\mathbf{s}})$ terminiert Algorithmus 8.8.1.1 mit der *MaxFP*-Semantik von $\mathcal{S}_G^{av,T,kpv}$.

Beweis. Folgt unmittelbar mit Lemma 8.13.1.3.1, Korollar 8.13.1.3.3 und Terminierungstheorem 8.8.2.1.

Theorem 8.13.1.3.5 (*SUP*-Akkuratheit)

Angewendet auf $\mathcal{S}_G^{av,T,kpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]^T_{av,kpv}, \bar{b}_{\mathbf{s}})$ ist Algorithmus 8.8.1.1 *SUP*-akkurat für $\mathcal{S}_G^{av,T,kpv}$ (d.h. terminiert mit der *SUP*-Semantik von $\mathcal{S}_G^{av,T,kpv}$).

Beweis. Folgt unmittelbar mit Lemma 8.13.1.3.2, Koinzidenztheorem 8.10.2 und Terminierungstheorem 8.8.2.1.

V3: Analog für *VUP*-Semantikgarantien

Analog zu Lemma 8.13.1.3.1, 8.13.1.3.2 gilt auch:

1. $\widehat{\mathbb{B}^n}$ erfüllt die aufsteigende Kettenbedingung.
2. $[\![\ ]\!]^T_{av,kpv}$ ist additiv.

Damit gilt analog zu Theorem 8.13.1.3.5 auch für die *VUP*-Semantik das entsprechende Terminierungs- und Akkuratheitsresultat:

Theorem 8.13.1.3.6 (*VUP*-Termin., -Akkuratheit)

Angewendet auf das duale DFA-Problem von $\mathcal{S}_G^{av,T,kpv}$ mit gestürztem Verband $\widehat{\mathbb{B}^n}$ ist Algorithmus 8.8.1.1 *VUP*-akkurat (d.h. terminiert mit der *VUP*-Semantik des dualen Problems).

Kapitel 8.13.1.4

Verfügbarkeit für endliche Termmengen: Bitvektorverbandformulierung

V4: Von Kreuzprodukt- zu Bitvektorsicht

Der Kreuzproduktverband $\widehat{\mathbb{B}^n}$ kann effizient als Bitvektorverband implementiert werden, indem Elemente von $\widehat{\mathbb{B}^n}$ als

- Bitvektoren der Länge n dargestellt werden:

$$\vec{bv} = [d_1, \ldots, d_n],\ d_i \in \{0,1\},\ 1 \leq i \leq n$$

Umsetzung:

- Bezeichne $\mathcal{BV}^n$ die Menge aller Bitvektoren der Länge n.
- Sei $\vec{bv}[i] = d_i$ für alle $\vec{bv} = [d_1, \ldots, d_n] \in \mathcal{BV}^n$, $1 \leq i \leq n$.
- Sei $\vec{0} =_{df} [0, \ldots, 0] \in \mathcal{BV}^n$ und $\vec{1} =_{df} [1, \ldots, 1] \in \mathcal{BV}^n$.
- Bezeichnen $min_{\mathcal{BV}}$ und $max_{\mathcal{BV}}$ die bitweisen Minimums- ('logisches $\wedge$') und Maximumsfunktionen ('logisches $\vee$') auf Bitvektoren, d.h.: $\forall\, \vec{bv_1}, \vec{bv_2} \in \mathcal{BV}^n\ \forall\, i \in \{1, \ldots, n\}$.
  - $(\vec{bv_1}\ min_{\mathcal{BV}}\ \vec{bv_2})[i] =_{df} min(\vec{bv_1}[i], \vec{bv_2}[i])$
  - $(\vec{bv_1}\ max_{\mathcal{BV}}\ \vec{bv_2})[i] =_{df} max(\vec{bv_1}[i], \vec{bv_2}[i])$

V4: Einführung von Hilfsfunktionen

Hilfsfunktionen:

– $ix : T \to \{1, \ldots, n\},\ ix^{-1} : \{1, \ldots, n\} \to T$

Bijektive Indexabbildungen, die jedem Term $t \in T$ eine eindeutig bestimmte Zahl in $\{1, \ldots, n\}$ zuordnen und umgekehrt.

Das $ix(t)$-te Element eines Bitvektors

$$\vec{bv} = [d_1, \ldots, d_{ix(t)}, \ldots, d_n)] \in \mathcal{BV}^n$$

ist die für t in $\vec{bv}$ abgelegte Verfügbarkeitsinformation.

V4: Ausdehnung und Anpassung

...der lokalen Prädikate auf Bitvektoren:

- $\overrightarrow{Comp}_e^T \in \mathcal{BV}^n$

  $$\forall\, i \in \{1, \ldots, n\}.\ \overrightarrow{Comp}_e^T\ [i] =_{df} \begin{cases} 1 & \text{falls } Comp_e^{ix^{-1}(i)} \\ 0 & \text{sonst} \end{cases}$$

- $\overrightarrow{Transp}_e^T \in \mathcal{BV}^n$

  $$\forall\, i \in \{1, \ldots, n\}.\ \overrightarrow{Transp}_e^T\ [i] =_{df} \begin{cases} 1 & \text{falls } Transp_e^{ix^{-1}(i)} \\ 0 & \text{sonst} \end{cases}$$

V4: Spezifizieren der DFA

DFA-Spezifikation (Bitvektorverband (bvv))

- DFA-Verband
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $$(\mathcal{BV}^n, min_{\mathcal{BV}}, max_{\mathcal{BV}}, <_{\mathcal{BV}}, \vec{0}, \vec{1}) = \widehat{\mathcal{BV}^n}$$
- DFA-Semantik
  $[\![\ ]\!]^T_{av,bvv} : E \to (\ \mathcal{BV}^n \to \mathcal{BV}^n\ )$
  $$\forall\, e \in E\ [\![\ e\ ]\!]^T_{av,bvs} =_{df}$$
  $$\lambda\, \vec{bv}.\ (\vec{bv}\ max_{\mathcal{BV}}\ \overrightarrow{Comp^T_e})\ min_{\mathcal{BV}}\ \overrightarrow{Transp^T_e}$$
- Anfangszusicherung
  $\vec{bv_s} \in \mathcal{BV}^n$

Verfügbarkeitsspezifikation für T

- Spezifikation: $\mathcal{S}_G^{av,T,bvv} = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]^T_{av,bvv}, \vec{bv_s})$

V4: Erbringen der Beweisverpflichtungen

Lemma 8.13.1.4.1 (Absteigende Kettenbedingung)

$\widehat{\mathcal{BV}^n}$ erfüllt die absteigende Kettenbedingung.

Lemma 8.13.1.4.2 (Distributivität)

$[\![\ ]\!]^T_{av,bvv}$ ist distributiv.

Korollar 8.13.1.4.3 (Monotonie)

$[\![\ ]\!]^T_{av,bvv}$ ist monoton.

V4: Einsammeln der *SUP*-Semantikgarantien

...für Terminierung, Akkuratheit.

Theorem 8.13.1.4.4 (*MaxFP*-Terminierung)

Angewendet auf $\mathcal{S}_G^{av,T,bvv} = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]_{av,bvv}^T, \vec{bv_s})$ terminiert Algorithmus 8.8.1.1 mit der *MaxFP*-Semantik von $\mathcal{S}_G^{av,T,bvv}$.

Beweis. Folgt unmittelbar mit Lemma 8.13.1.4.1, Korollar 8.13.1.4.3 und Terminierungstheorem 8.8.2.1.

Theorem 8.13.1.4.5 (*SUP*-Akkuratheit)

Angewendet auf $\mathcal{S}_G^{av,T,bvv} = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]_{av,bvv}^T, \vec{bv_s})$ ist Algorithmus 8.8.1.1 *SUP*-akkurat für $\mathcal{S}_G^{av,T,bvv}$ (d.h. terminiert mit der *SUP*-Semantik von $\mathcal{S}_G^{av,T,bvs}$).

Beweis. Folgt unmittelbar mit Lemma 8.13.1.4.2, Koinzidenztheorem 8.10.2 und Terminierungstheorem 8.8.2.1.

V4: Analog für *VUP*-Semantikgarantien

Analog zu Lemma 8.13.1.4.1, 8.13.1.4.2 gilt auch:

1. $\widehat{\mathcal{BV}^n}$ erfüllt die aufsteigende Kettenbedingung.
2. $[\![\ ]\!]^T_{av,bvv}$ ist additiv.

Damit gilt analog zu Theorem 8.13.1.4.5 auch für die *VUP*-Semantik das entsprechende Terminierungs- und Akkuratheitsresultat:

Theorem 8.13.1.4.6 (*VUP*-Terminier., -Akkuratheit)

Angewendet auf das duale DFA-Problem von $\mathcal{S}_G^{av,T,bvv}$ mit gestürztem Verband $\widehat{\mathcal{BV}^n}$ ist Algorithmus 8.8.1.1 *VUP*-akkurat (d.h. terminiert mit der *VUP*-Semantik des dualen Problems).

Performanzanmerkung: $\mathcal{S}_G^{kpv}$ vs. $\mathcal{S}_G^{bvv}$

Anders als für $\mathcal{S}_G^{av,T,kpv}$ (Kap. 8.13.1.3) kann der Fixpunktalgorithmus 8.8.1.1 für

- $\mathcal{S}_G^{av,T,bvv}$

die Vorteile der auf Prozessoren vorhandenen hochperformanten Operationen auf Bitvektoren ausnutzen und davon profitieren.

Verfügbarkeit wird deshalb auch als

- Bitvektorproblem

bezeichnet.

Kapitel 8.13.1.5

Verfügbarkeit für endliche Termmengen: Gen/Kill-Formulierung

V5: Verfügbarkeit für endliche Termmenge T

Einführung sog. Gen/Kill-Mengen für anw.-benannte Kanten

- $Gen_e^T =_{df} \{t \in T \mid Comp_e^t \wedge \neg Mod_e^t\}$
  $= \{t \in T \mid Comp_e^t \wedge Transp_e^t\}$

  ...'generating' the property of interest.

- $Kill_e^T =_{df} \{t \in T \mid Mod_e^t\}$
  $= \{t \in T \mid \neg Transp_e^t\}$

  ...'killing' the property of interest.

V5: Spezifizieren der DFA

DFA-Spezifikation (gen/kill (gk))

- DFA-Verband
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T) = \widehat{\mathcal{P}(T)}$
- DFA-Semantik
  $[\![\ ]\!]^T_{av,gk} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$
  $\forall e \in E.\ [\![e]\!]^T_{av,gk} =_{df} \lambda T'.\ (T' \setminus Kill^T_e) \cup Gen^T_e$
- Anfangszusicherung
  $T_{\mathbf{s}} \in \mathcal{P}(T)$

Verfügbarkeitsspezifikation für T

- Spezifikation: $\mathcal{S}^{av,T,gk}_G = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]^T_{av,gk}, T_{\mathbf{s}})$

V5: Erbringen der Beweisverpflichtungen

Aus dem Vergleich von:

- $[\![\ ]\!]^T_{av,gks} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$

  $\forall e \in E.\ [\![\ e\ ]\!]^T_{av,gk} =_{df} \lambda\, T'.\ (T' \setminus Kill^T_e) \cup Gen^T_e$

mit:

- $[\![\ ]\!]^T_{av} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$

  $\forall\, e \in E.\ [\![\ e\ ]\!]^T_{av} =_{df}$

  $\lambda\, T'.\ \{t \in T \mid (t \in T' \vee Comp^t_e) \wedge Transp^t_e\}$

erhalten wir:

Lemma 8.13.1.5.1 (Gleichheit)

$$[\![\ ]\!]^T_{av} = [\![\ ]\!]^T_{av,gk}$$

...woraus Distributivität (und Monotonie) der Gen/Kill-Semantikfunktionen folgt.

V5: Einsammeln der *SUP*-Semantikgarantien

...für Terminierung, Akkuratheit.

Theorem 8.13.1.5.4 (*MaxFP*-Terminierung)

Angewendet auf $\mathcal{S}_G^{av,T,gk} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av,gk}^T, T_{\mathbf{s}})$ terminiert Algorithmus 8.8.1.1 mit der *MaxFP*-Semantik von $\mathcal{S}_G^{av,T,gk}$.

Beweis. Folgt unmittelbar mit Lemma 8.13.1.2.1, Lemma 8.13.1.2.2, Korollar 8.13.1.2.3, Terminierungstheorem 8.8.2.1.

Theorem 8.13.1.5.5 (*SUP*-Akkuratheit)

Angewendet auf $\mathcal{S}_G^{av,T,gk} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av,gk}^T, T_{\mathbf{s}})$ ist Algorithmus 8.8.1.1 *SUP*-akkurat für $\mathcal{S}_G^{av,T,gk}$ (d.h. terminiert mit der *SUP*-Semantik von $\mathcal{S}_G^{av,T,gk}$).

Beweis. Folgt unmittelbar mit Lemma 8.13.1.2.1, Lemma 8.13.1.2.2, Koinzidenztheorem 8.10.2,Terminierungstheorem 8.8.2.1.

V5: Analog für *VUP*-Semantikgarantien

Analog zu absteigender Kettenbedingung und Distributivität gilt auch:

1. $\widehat{\mathcal{P}(T)}$ erfüllt die aufsteigende Kettenbedingung.
2. $[\![\ ]\!]^T_{av,gk}$ ist additiv.

Damit gilt analog zu Theorem 8.13.1.5.5 auch für die *VUP*-Semantik das entsprechende Terminierungs- und Akkuratheitsresultat:

Theorem 8.13.1.5.6 (*VUP*-Terminier., -Akkuratheit)

Angewendet auf das duale DFA-Problem von $\mathcal{S}_G^{av,T,gk}$ mit gestürztem Verband $\widehat{\mathcal{P}(T)}$ ist Algorithmus 8.8.1.1 *VUP*-akkurat (d.h. terminiert mit der *VUP*-Semantik des dualen Problems).

Verdeutlichung: Verfügb. als Gen/Kill-Prob. (1)

Die Spezialisierung des *MaxFP*-Gleichungssystems 8.7.1.1:

Gleichungssystem 8.7.1.1 (*MaxFP*-Gleichungssyst.)

$$inf(n) = \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcap \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

...für Verfügbarkeit liefert:

Gleichungssystem 8.13.1.5.7 (Verfügbarkeit)

$$Available(n) =$$

$$\begin{cases} T_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigcap \{ [\![(m,n)]\!]^{T}_{av,gk}(Available(m)) \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

Verdeutlichung: Verfügb. als Gen/Kill-Prob. (2)

...durch zusätzliches Expandieren von $[\![\ ]\!]^T_{av,gk}$ erhalten wir:

Gleichungssystem 8.13.1.5.7′ (Verfügbarkeit)

$$Available(n) =$$

$$\begin{cases} T_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigcap \{ (Available(m) \setminus Kill^T_{(m,n)}) \cup Gen^T_{(m,n)} \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

Aufgrund der Abstützung von Gleichungssystem 8.13.1.5.7′ und der Semantikfunktionen:

$$[\![\ ]\!]^T_{av,gk} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$$

$$\forall e \in E.\ [\![\ e\ ]\!]^T_{av,gks} =_{df} \lambda\, T'.\ (T' \setminus Kill^T_e) \cup Gen^T_e$$

auf Gen/Kill-Mengen, wird Verfügbarkeit auch als

– Gen/Kill-Problem

bezeichnet.

Gen/Kill- (oder Bitvektor-) Probleme

...umfassen neben Verfügbarkeit viele weitere Probleme wie

- beschäftigte Terme, lebendige Variablen, erreichende Definitionen, etc.

Trotz ihrer konzeptuellen Einfachheit bilden sie eine sehr wichtige Klasse von DFA-Problemen mit zahlreichen Anwendungen in der Programmoptimierung, darunter:

- Elimination partiell redundanter Ausdrücke
- Reduktion der Operatorstärke
- Elimination partiell toter Anweisungen
- Elimination partiell redundanter Anweisungen
- Anweisungsschieben
- ...

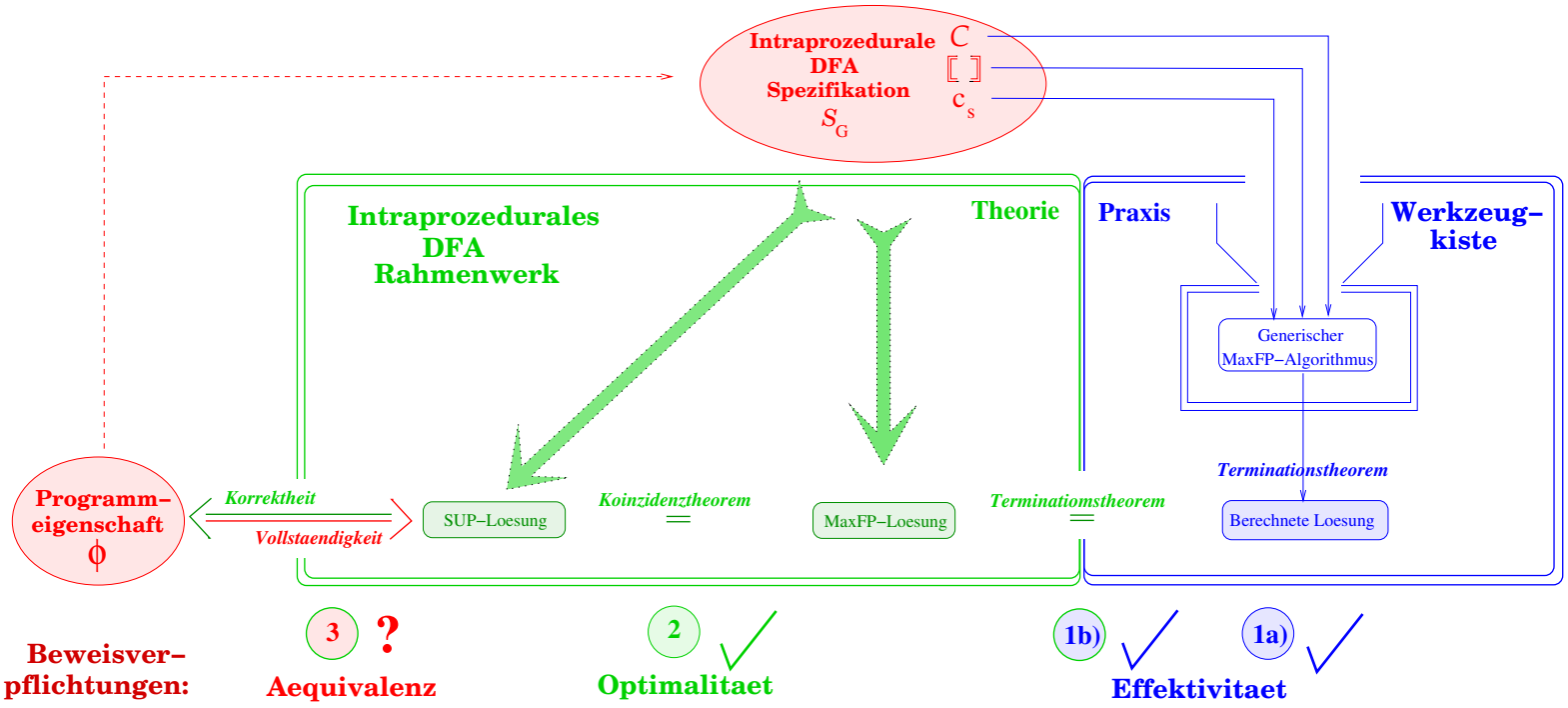
...siehe LVA 185.A04 Optimierende Übersetzer für mehr Details.

Kapitel 8.13.1.6

Schließen der äußeren Äquivalenzbeweislücke am Beispiel von Verfügbarkeitsvariante 1

V1: Schließen d. äußeren Äquivalenzbeweislücke

...am Beispiel des Korrektheits- und Vollständigkeitsbeweises für die *SUP*-Sicht des Verfügbarkeitsproblems $\mathcal{S}_G^{av,t}$:



Zur Erinnerung

...ein Term t heißt verfügbar an einem Knoten, wenn t auf jedem Pfad vom Programmanfang zu diesem Knoten berechnet wird und anschließend bis zum Erreichen dieses Knotens keine seiner Operandenvariablen einen Wert zugewiesen erhält.

Beachte

- Wird Programmanfang durch Prozeduranfang ersetzt, trifft die obige umgangssprachliche Definition von Verfügbarkeit keine Vorsorge für den Fall, dass ein Ausdruck am Prozeduranfang selbst verfügbar ist.
- Fälle, in denen die Verfügbarkeit am Prozeduranfang durch die Aufrufkontexte der Prozedur sichergestellt werden, sind deshalb nicht erfasst und können nicht behandelt werden.

Einführung nützlicher Schreibweise

...als Vorbereitung für eine formale Verfügbarkeitsdefinition.

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Flussgraph, *predicate* ein für Kanten $e \in E$ definiertes Prädikat und $p = \langle e_1, \ldots, e_q \rangle \in \mathbf{P}[m, n]$ ein Pfad, sowie:

- p_i, $1 \leq i \leq q$: i-te Kante e_i von p.
- $p_{[k,l]}$: Teilpfad $\langle e_k, \ldots, e_l \rangle$ von p.
- $\lambda_p = q$: Länge von p, d.h. Zahl der Kanten von p.

Für Prädikatquantifizierungen auf Pfaden definieren wir:

- $\mathit{predicate}_p^{\forall} \iff_{df} \forall\, 1 \leq i \leq \lambda_p.\ \mathit{predicate}_{p_i}$
- $\mathit{predicate}_p^{\exists} \iff_{df} \exists\, 1 \leq i \leq \lambda_p.\ \mathit{predicate}_{p_i}$

Verfügbarkeit

...formal definiert:

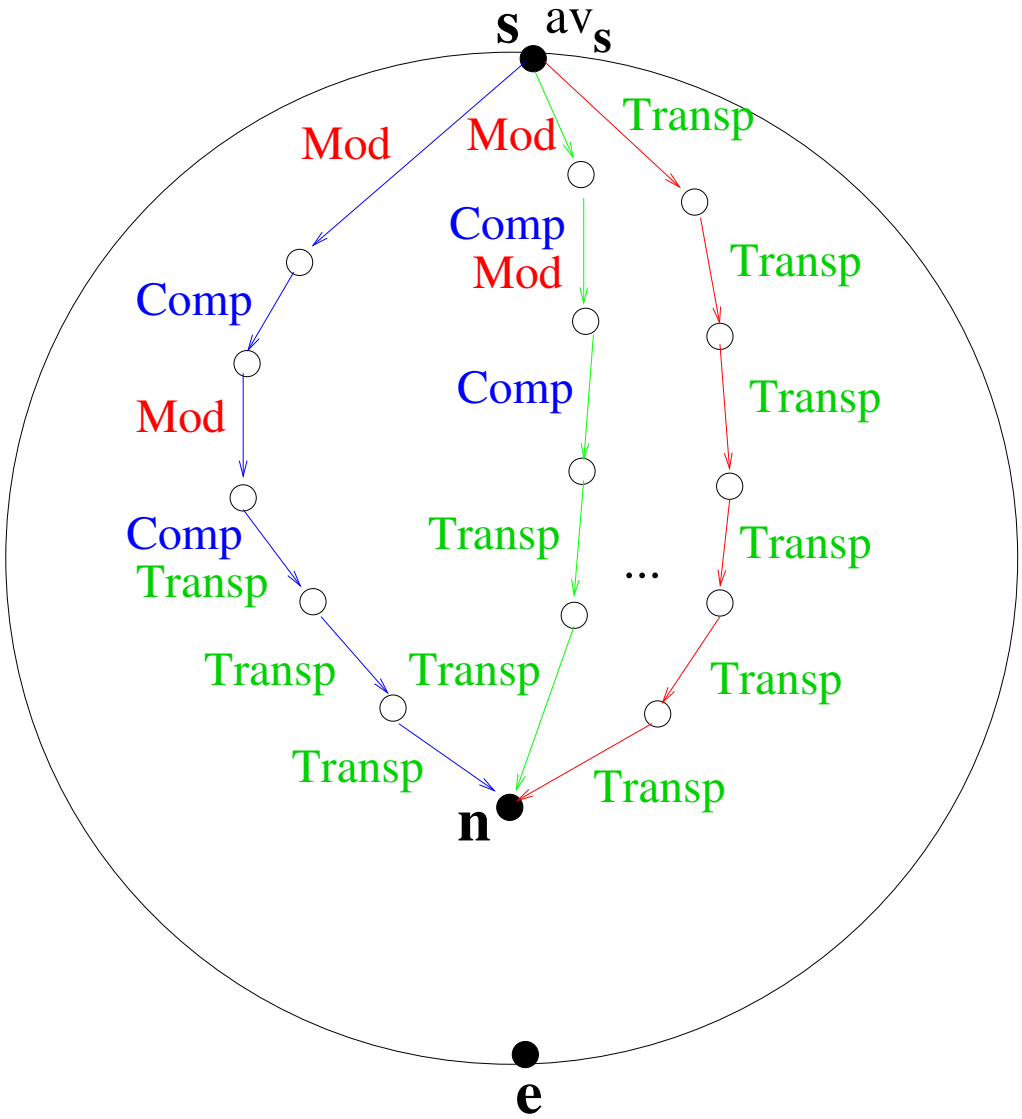
Definition 8.13.1.6.1 (Verfügbarkeit)

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Flussgraph, t ein Term und $av_{\mathbf{s}} \in \mathbb{B}$ die durch den Aufrufkontext von G für t an $\mathbf{s}$ zugesicherte Anfangsinformation. Dann definieren wir:

$$Available^t(n) \iff_{df}$$

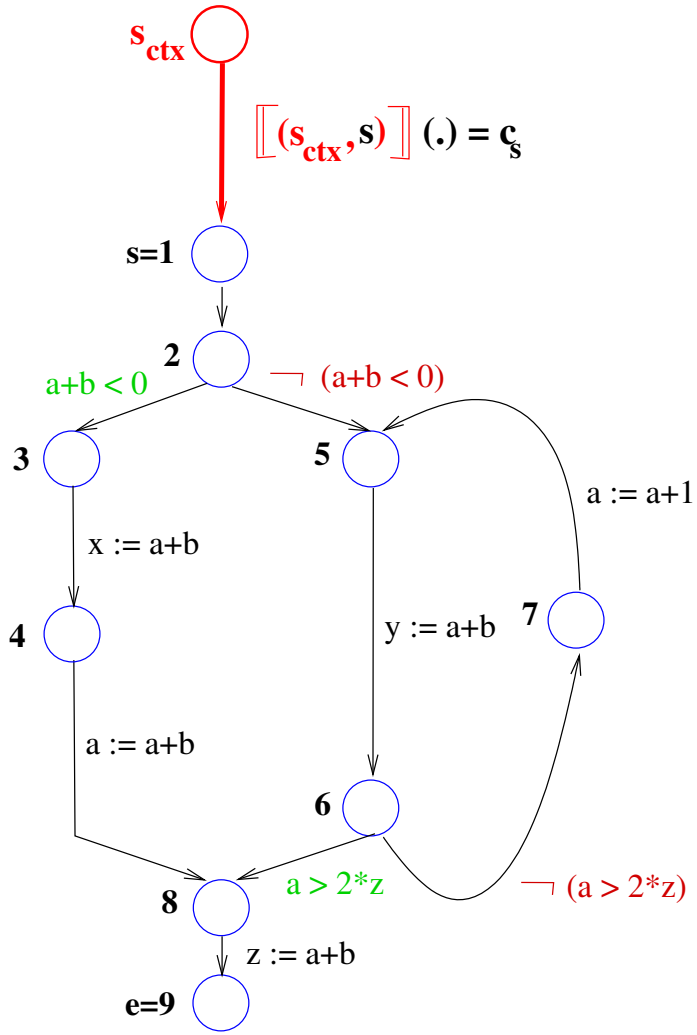
$$\begin{cases} av_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \forall\, p \in \mathbf{P}[\mathbf{s}, n].\ (av_{\mathbf{s}}^t \wedge Transp^t{}^{\forall}_{p}) \ \vee & \\ \qquad \exists\, i \leq \lambda_p.\ Comp^t_{p_i} \wedge Transp^t{}^{\forall}_{p[i,\lambda_p]} & \text{sonst} \end{cases}$$

Veranschaulichung von Definition 8.13.1.6.1



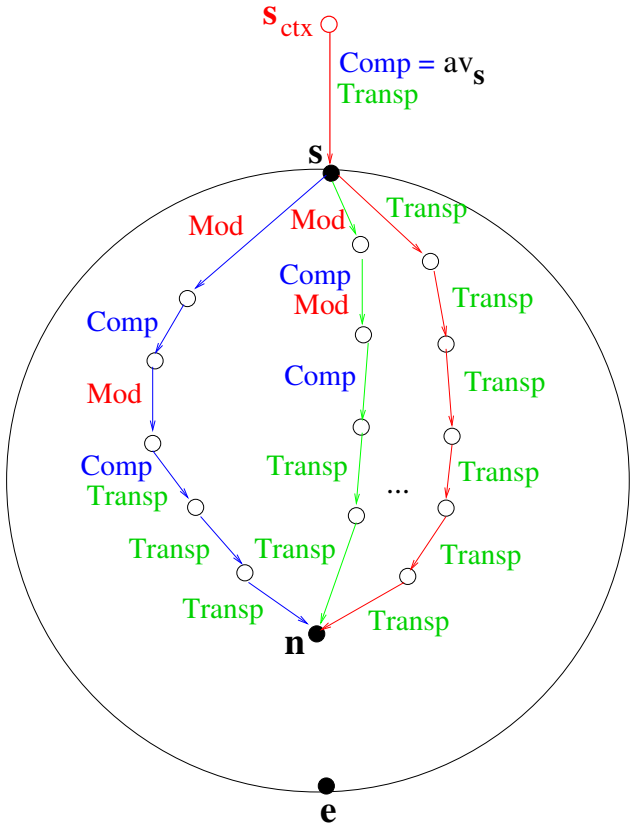
Kontextkanten

...erlauben eine einfachere, fallunterscheidungsfreie Definition von Verfügbarkeit:



Ausnutzung der Kontextkante

...um Verfügbarkeit fallunterscheidungsfrei zu definieren:



Definition 8.13.1.6.1′ (Verfügbarkeit)

$$\forall\, n \in N \backslash \{\mathbf{s}_{ctx}\}.\ \mathit{Available}^t(n) \iff_{df} \forall\, p \in \mathbf{P}[\mathbf{s}_{ctx}, n].\ \exists\, i \leq \lambda_p.\ \mathit{Comp}^t_{p_i} \wedge \mathit{Transp}^{t\,\forall}_{p[i,\lambda_p]}$$

Schließen der äußeren Äquivalenzbeweislücke

Theorem 8.13.1.6.2 (Korrektheit/Vollständigkeit)

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Flussgraph, t ein Ausdruck, $av_{\mathbf{s}} \in \mathbb{IB}$ die durch die Aufrufkontexte von G zugesicherte Verfügbarkeitsinformation für t am Startknoten $\mathbf{s}$ und sei $[\![\ ]\!]^{SUP}_{\mathcal{S}^{av,t}_G}$ die *SUP*-Semantik von G für die DFA-Spezifikation

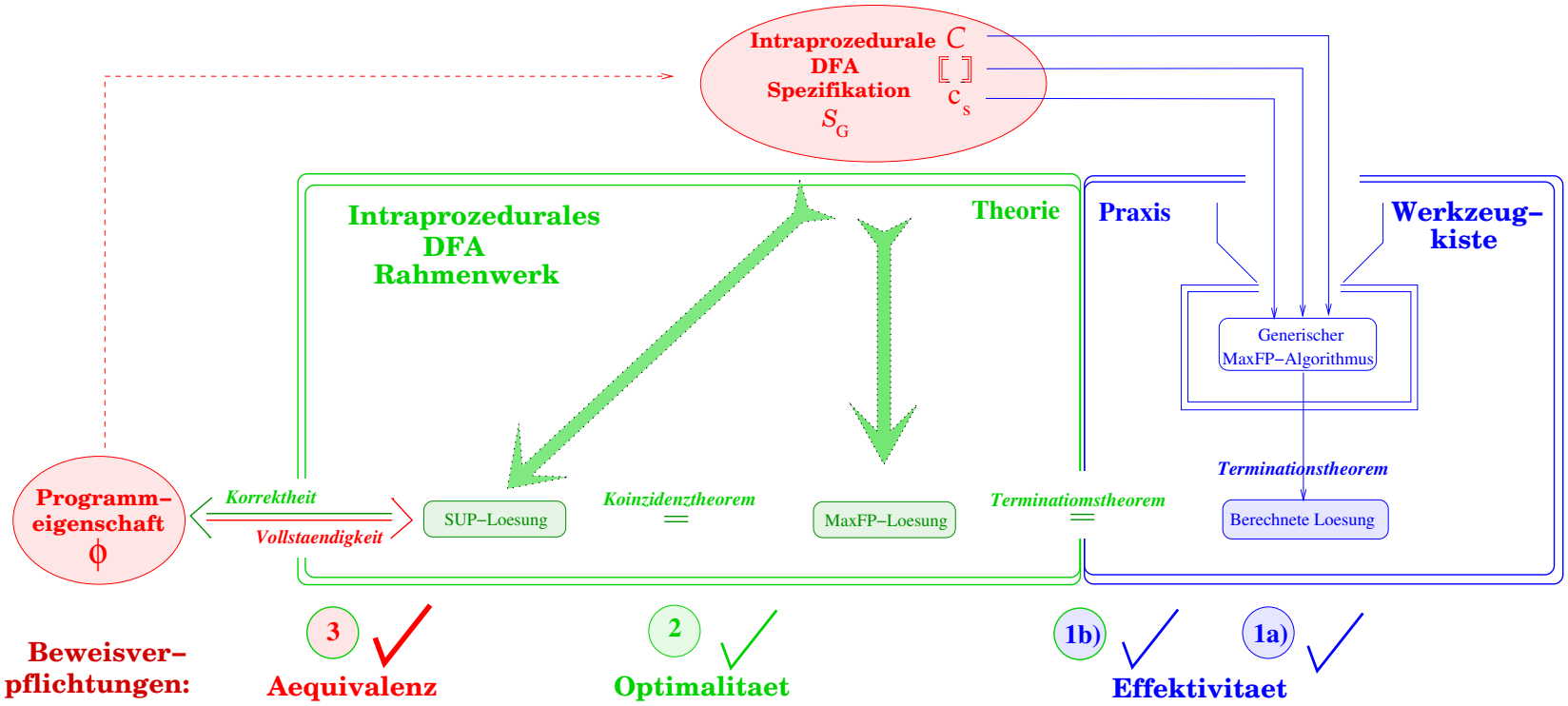
$$\mathcal{S}^{av,t}_G = (\widehat{\mathbb{IB}}, [\![\ ]\!]^t_{av}, av_{\mathbf{s}})$$

Dann gilt:

$$\forall\, n \in N.\ Available^t(n) \Longleftrightarrow [\![\, n \,]\!]^{SUP}_{\mathcal{S}^{av,t}_G}$$

Lücke geschlossen: Korrektheit/Vollständigkeit

...für die *SUP*-Sicht von $\mathcal{S}_G^{av,t}$ für Termverfügbarkeit bewiesen:



Übungsaufgabe 8.13.1.6.3

1. Was bedeuten Korrektheit, Vollständigkeit für die *VUP*-Sicht von $\mathcal{S}_G^{av,t}$-Termverfügbarkeit?
2. Wie lassen sie sich beweisen?

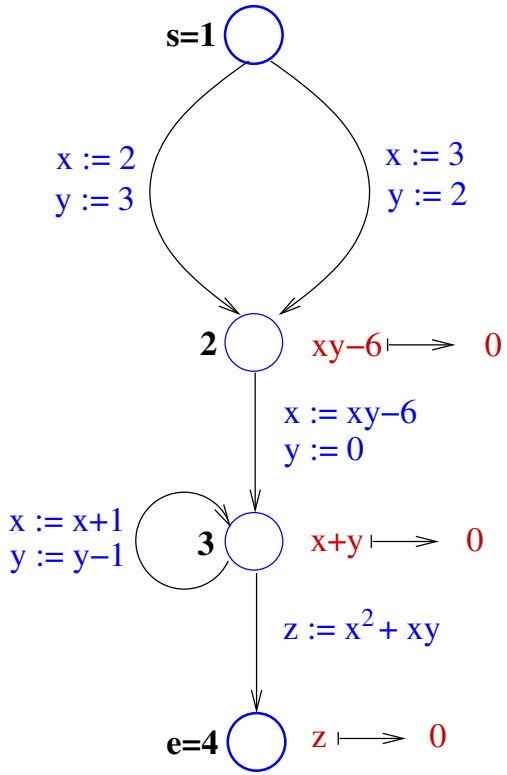
Kapitel 8.13.2

Einfache Konstanten, ein monotones DFA-Problem

Ein Term t

...ist die Konstante c am Knoten **n**, wenn die Auswertung von t an **n** stets den Wert c liefert, unabhängig vom Programmpfad, auf dem **n** von **s** aus erreicht worden ist.

Veranschaulichung:



Quelle Beispiel: Markus Müller-Olm, Helmut Seidl (SAS 2002)

Konstantenausbreitung und -faltung

...Terme von konstantem Wert können zur Übersetzungszeit durch diesen Wert ersetzt werden.

Dadurch wird Berechnungsauwand von der Laufzeit eines Programms in seine Übersetzungszeit verlagert und die Laufzeitperformanz verbessert, eine als

- Konstantenausbreitung und -faltung

bezeichnete Programmoptimierung.

Leider ist die Existenz eines stets erfolgreichen Algorithmus, ob ein Term an einer Programmstelle eine Konstante ist oder nicht, ausgeschlossen.

Kapitel 8.13.2.1

Unentscheidbarkeit des Konstantenerkennungsproblems

Unentscheidbarkeit von Konstantenerkennung

Theorem 8.13.2.1.1 (Unentscheidb., Reif&Lewis 1977)

Das Problem, alle arithmetischen Ausdrücke eines Programms zu bestimmen, die mit einer Konstanten wertgleich sind, ist unentscheidbar.

John H. Reif, Harry R. Lewis. Symbolic Evaluation and the Global Value Graph. In Conference Record of the 4th Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'77), 104-118, 1977.

Beweisskizze für Theorem 8.13.2.1.1 (1)

Reif und Lewis reduzieren den Beweis von Theorem 8.13.2.1.1 auf Hilberts 10-tes Problem, ob ein Polynom eine Wurzel in den natürlichen Zahlen hat:

Theorem 8.13.2.1.2 (Unentscheidbarkeit von Hilberts 10-tem Problem, Matijasevič 1970)

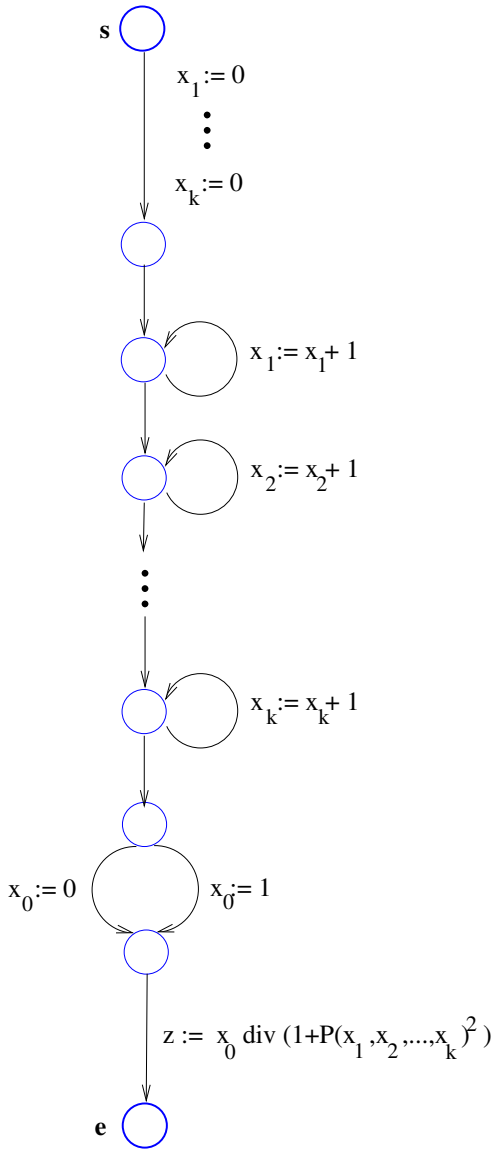
Sei $\{x_1, \ldots, x_k\}$, $k > 5$, eine Menge von Variablen und sei $P(x_1, \ldots, x_k)$ ein (multivariates) Polynom über $x_1, \ldots, x_k$.

Es ist nicht entscheidbar, ob $P(x_1, \ldots, x_k)$ eine Wurzel in den natürlichen Zahlen hat, d.h. es ist nicht entscheidbar, ob es natürlichzahlige Werte $n_1, \ldots, n_k$ gibt, so dass gilt:

$$P(x_1, \ldots, x_k)[n_1, \ldots, n_k / x_1, \ldots, x_k] = 0$$

Beweisskizze für 8.13.2.1.1 (2)

Betrachte folgendes als Flussgraph gegebene Programm G:



Beweisskizze für Theorem 8.13.2.1.1 (3)

Zeige:

P hat keine Wurzel in den natürlichen Zahlen gdw

z ist am Knoten **e** von G eine Konstante

Unter Ausnutzung von Theorem 8.13.2.1.2 erbringt der Beweis dieser Äquivalenz den Beweis von Theorem 8.13.2.1.1.

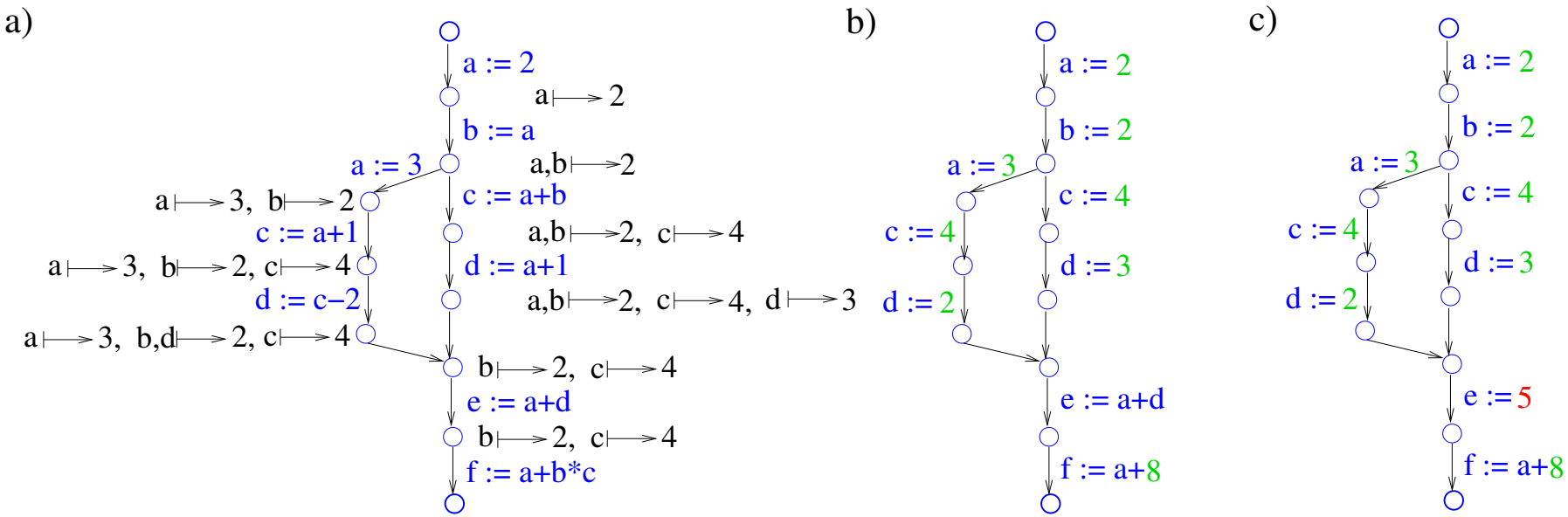
□

Einfache Konstanten: Entscheidbare K.-Klasse

...aufgrund dieses negativen Resultats werden in der Praxis einfachere Varianten (oder Klassen) des Konstantenausbreitungs- und faltungsproblems betrachtet, die entscheidbar sind; eine davon ist die Klasse der sog. einfachen Konstanten.

Informell: Ein Term ist eine einfache Konstante (engl. simple constant) an einem Programmpunkt, wenn jeder seiner Operanden an diesem Punkt einen eindeutigen konstanten Wert hat, unabhängig davon, auf welchem Pfad vom Programmanfang aus dieser Punkt erreicht wird.

Beispiel: Einfache Konstanten

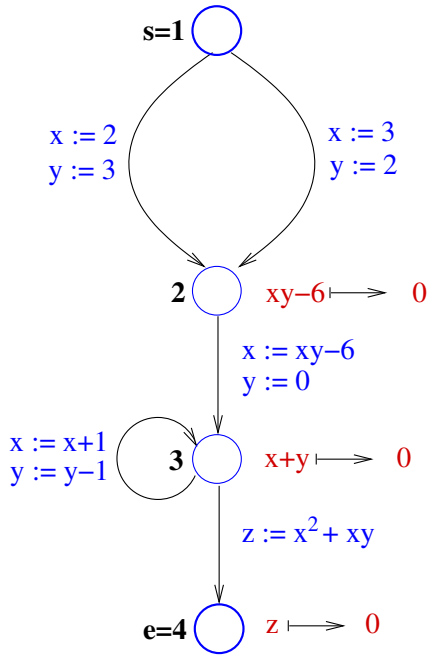


Mit Ausnahme von $a + d$ und $a + 8$ sind alle Terme einfache Konstanten (Abbildung b)).

Beachte:

- $a + 8$ ist keine Konstante.
- $a + d$ ist Konstante mit Wert 5, aber keine einfache Konstante (Abbildung c)).

Anti-Beispiel: Keine einfachen Konstanten



Kein (nichttrivialer) Term im Beispiel von Müller-Olm und Seidl ist eine einfache Konstante.

Beachte: $a + d$ im vorigen Beispiel und alle Terme im Beispiel von Müller-Olm/Seidl können von ausgefeilteren (und im zweiten Fall wesentlich berechnungsaufwändigeren) Verfahren als Konstanten erkannt werden (s. z.B. LVA 185.A04 Optimierende Übersetzer).

Einfache Konstanten: Monotones DFA-Problem

...die Erkennung einfacher Konstanten ist

- kanonisches Beispiel eines (nichtdistributiven) monotonen DFA-Problems.
- Beispiel einer korrekten, unvollständigen Analyse, die viele als Konstanten erkennbare Terme nicht als konstant erkennt, die aber effizient und mit für die Programmoptimierung noch immer nützlichen Ergebnissen ist:

  Aufgabe von Vollständigkeit zugunsten von Effizienz!
  (engl. trading completeness for efficiency!)

Arbeitsplan

...in der Folge spezifizieren wir das Einfache-Konstanten-Problem (engl. simple constants) für Terme.

Dabei gehen wir in folgenden Schritten vor:

- Kap. 8.13.2.2: Einführung von
  - DFA-Zuständen
  - Termsemantik
  - Instruktionssemantik
  - DFA-Zustandsverband für einfache Konstanten
- Kap. 8.13.2.3: DFA-Spezifikation, Beweisverpflichtungen, Garantien

Kapitel 8.13.2.2

Berechnung einfacher Konstanten: Vorbereitungen

Von Datenbereichen zu DFA-Verbänden

Sei $\mathbb{ID}$ ein

- interessierender Datenbereich (engl. data domain) (z.B. die Menge natürlicher Zahlen $\mathbb{IN}$, die Menge ganzer Zahlen $\mathbb{Z}$, die Menge der Wahrheitswerte $\mathbb{B}$, etc.) mit einem ausgezeichneten Element $\bot$, das den Wert *undefiniert* darstellt.

Wir erweitern $\mathbb{ID}$ um ein neues

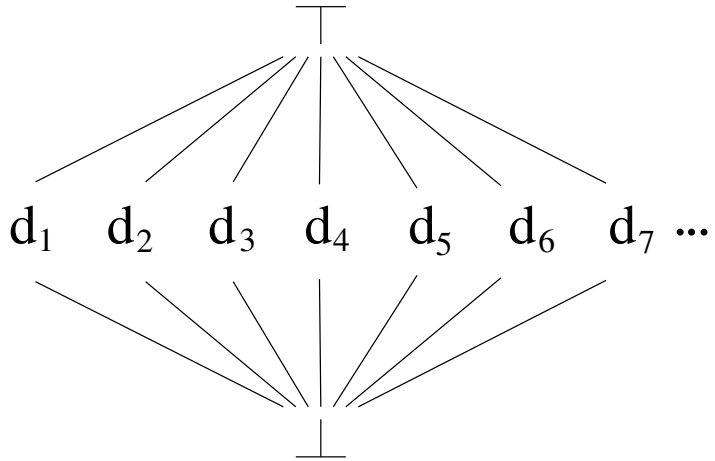
- neues Element $\top$ nicht in $\mathbb{ID}$, d.h.: $\top \notin \mathbb{ID}$.

Den erweiterten Bereich bezeichnen wir mit $\mathbb{ID}' =_{df} \mathbb{ID} \cup \{\top\}$.

Bem: $\bot$ als Element des zugrundeliegenden Datenbereichs anzunehmen, $\top$ jedoch nicht, erscheint willkürlich. Der Grund dafür ist, dass Datentypen oft so implementiert sind, dass sie einen speziellen Wert mit der Bedeutung 'undefiniert' enthalten.

Allgemein: Konstruktion von DFA-Verbänden

...ist $\mathbb{ID}'$ ein erweiterter Datenbereich, konstruieren wir den flachen Verband (engl. flat lattice) $\mathcal{FV}_{\mathbb{ID}'}$ (s. Anhang A.4):



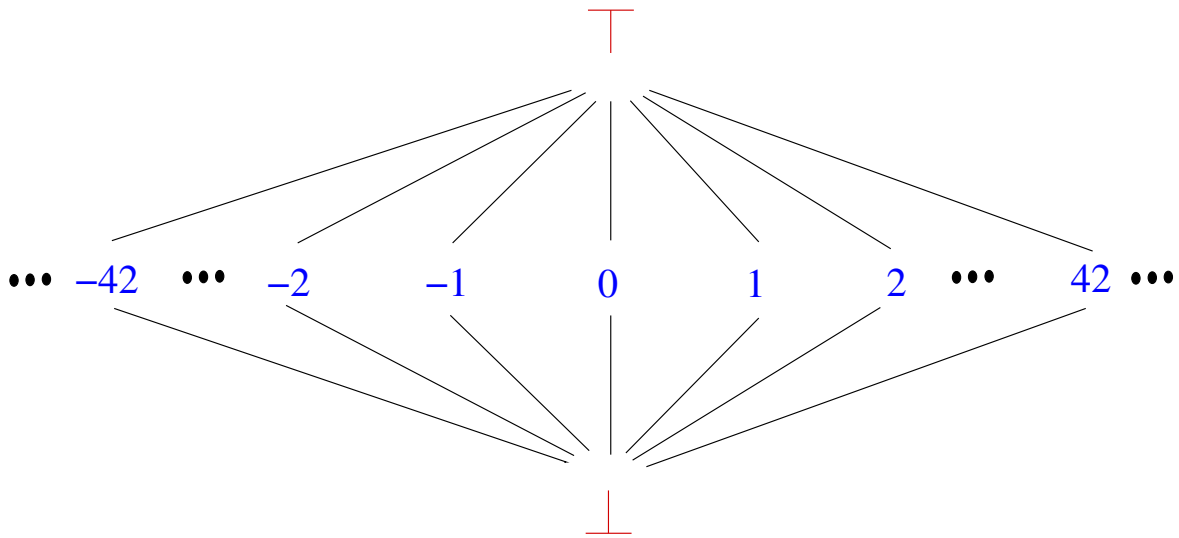
der der basale Verband der DFA-Analyse für einfache Konstanten ist.

Intuitiv

- $\top$ steht für vollständige, aber inkonsistente Information.
- d_i, $i \geq 1$, steht für akkurate Information.
- $\bot$ steht für keine Information, die 'leere' Information.

Konkret: Der basale DFA-Verband über $\mathbb{Z}$

...ist gegeben durch den flachen Verband $\mathcal{FV}_{\mathbf{Z}}$:



...der zur Berechnung der Klasse einfacher Konstanten über $\mathbb{Z}$ verwendet wird.

Abstrakte Programmzustände: DFA-Zustände

Definition 8.13.2.2.1 (DFA-Zustände)

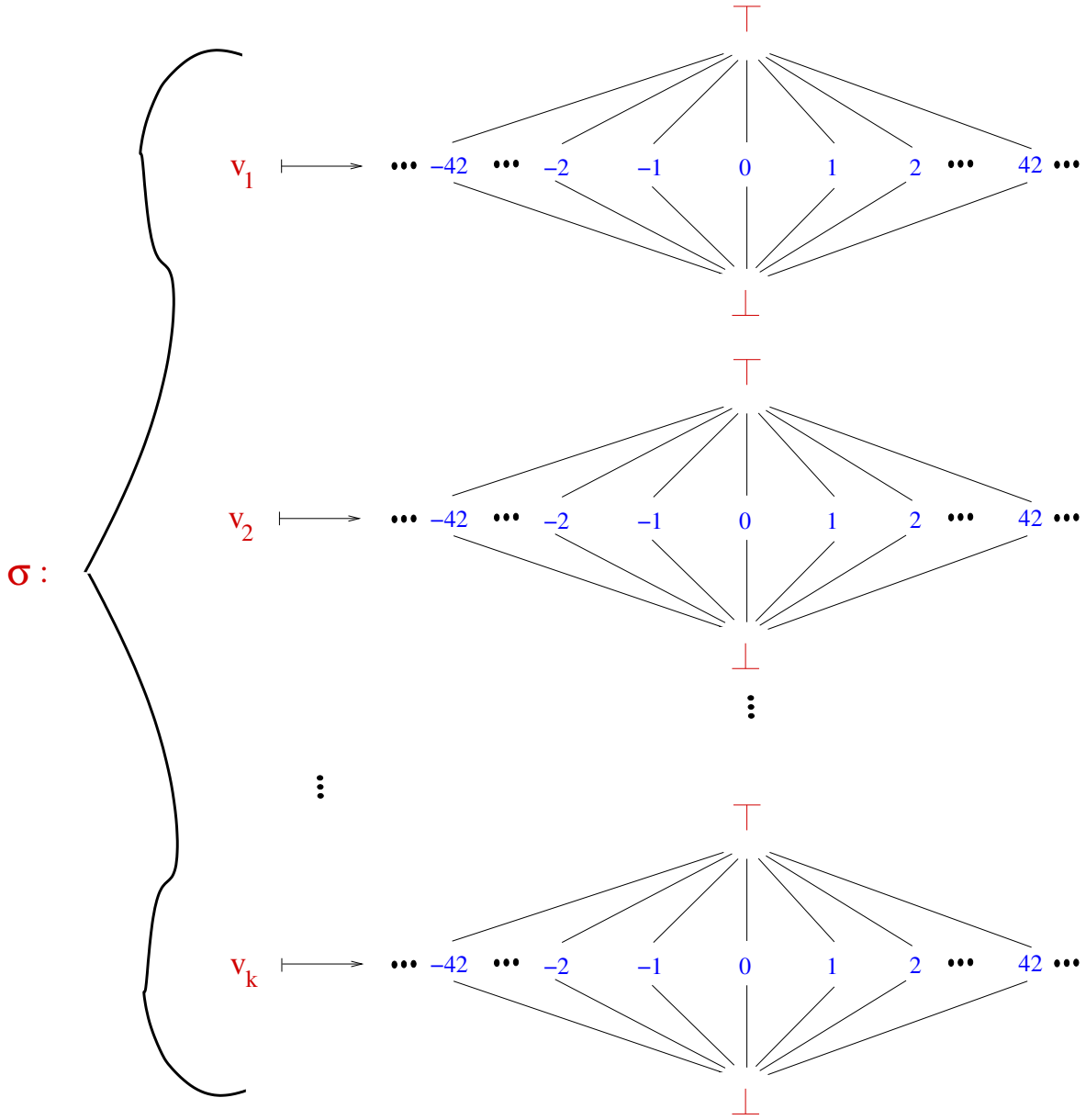
1. Ein DFA-Zustand ist eine totale Abbildung $\sigma : \mathbf{V} \to \mathbb{ID}'$, die jeder Variablen ein Datum $d \in \mathbb{ID}'$ zuweist.
2. Die Menge aller DFA-Zustände ist definiert durch:

$$\Sigma' =_{df} \{ \sigma \mid \sigma : \mathbf{V} \to \mathbb{ID}' \}.$$

3. $\sigma_\bot$ und $\sigma_\top$ bezeichnen zwei ausgezeichnete DFA-Zustände aus Σ', die folgendermaßen definiert sind:

$$\sigma_\bot = \lambda v. \bot, \ \sigma_\top = \lambda v. \top.$$

Veranschaulichung: DFA-Zustand σ über $\mathbb{Z}$



Initiale DFA-Zustände

...für initiale DFA-Zustände (d.h. Anfangszustände) verlangen wir, dass keiner Variablen der besondere Wert $\top$ zugewiesen ist, d.h. wir verlangen am Prozedur-/Programmanfang entweder akkurate Information über den Wert einer Variablen zu haben oder gar keine. Wir definieren:

Definition 8.13.2.2.2 (Initiale DFA-Zustände über ID′)

Die Menge initialer DFA-Zustände über ID′ ist definiert durch:

$$\Sigma'_{Init} =_{df} \{ \sigma \in \Sigma' \mid \forall v \in \mathbf{V}.\ \sigma(v) \neq \top \}$$

Ausdehnung der Interpretation

...für Konstanten- und Operatorsymbole von ID auf ID′.

Definition 8.13.2.2.3 (Ausdehnung der Interpretation)

Sei $I =_{df} (\mathrm{ID}, I_0)$ eine Interpretation der Konstanten- und Operatorsymbole über dem Datenbereich ID.

Dann ist $I' =_{df} (\mathrm{ID}', I_0')$ diejenige Interpretation über ID′, die I in folgender Weise ausdehnt:

- $I_0'(c) =_{df} I_0(c)$ für jedes Konstantenssymbol $c \in \mathbf{C}$
- $I_0'(op) : \mathrm{ID}'^k \to \mathrm{ID}'$ für jedes k-st. Operatorsymbol $op \in \mathbf{O}$:

$$\forall (d_1, \ldots, d_k) \in \mathrm{ID}'^k.\ I_0'(op)(d_1, \ldots, d_k) =_{df}$$

$$\begin{cases} I_0(op)(d_1, \ldots, d_k) & \text{falls } d_i = \bot \text{ für einige } 1 \leq i \leq k, \\ & \quad \text{oder } d_j \neq \top,\ 1 \leq j \leq k \\ \top & \text{falls } d_i \neq \bot,\ 1 \leq i \leq k, \text{ und} \\ & \quad d_j = \top \text{ for some } 1 \leq j \leq k \end{cases}$$

Die abstrakte Termsemantik über ID′

Definition 8.13.2.2.4 (AbstrakteTermsemantik)

Die abstrakte Semantik von Termen $t \in \mathbf{T}$ ist durch die Auswertungsfunktion

$$\mathcal{A} : \mathbf{T} \to (\Sigma' \to \mathrm{ID}')$$

gegeben, die folgendermaßen definiert ist:

$$\forall t \in \mathbf{T}.\ \mathcal{A}(t) =_{df}\ \lambda\sigma.\ \begin{cases} \sigma(x) & \text{falls } t \equiv x \in \mathbf{V} \\ I_0'(c) & \text{falls } t \equiv c \in \mathbf{C} \\ I_0'(op)(\mathcal{A}(t_1)(\sigma), \ldots, \mathcal{A}(t_k)(\sigma)) & \\ & \text{falls } t \equiv (op, t_1, \ldots, t_k) \end{cases}$$

Die abstrakte Instruktionssemantik

Definition 8.13.2.2.5 (Abstrakte Instruktionssemantik)

Die abstrakte Semantik

- einer Zuweisung $\iota \equiv x := t$ ist durch die Zustandstransformation(sfunktion) (engl. state transformer) $\theta_\iota \colon \Sigma' \to \Sigma'$ gegeben, die definiert ist durch:

$$\forall\, \sigma \in \Sigma'.\ \theta_\iota(\sigma) =_{df}\ \lambda\, y.\ \begin{cases} \mathcal{A}(t)(\sigma) & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

- der leeren Anweisung $\iota \equiv \mathit{skip}$ und einer (Verzweigungs-) Bedingung $\iota \equiv \mathit{cond}$ ist durch die identische Zustandstransformation $\mathit{Id}_{\Sigma'}$ gegeben, d.h.: $\theta_\iota =_{df} \mathit{Id}_{\Sigma'} =_{df} \lambda\sigma.\,\sigma$.

Bem: Die Ausführung von *skip* und die Auswertung von Bedingungen sind seiteneffektfrei.

Der DFA-Verband für einfache Konstanten

...die Menge der DFA-Zustände bildet mit der punktweisen Ordnung auf Zuständen, $\sqsubseteq_{\Sigma'}$, einen vollständigen Verband (s. Anhang A.4):

$$\forall\, \sigma, \sigma' \in \Sigma'.\ \sigma \sqsubseteq_{\Sigma'} \sigma' \iff_{df} \forall\, v \in \mathbf{V}.\ \sigma(v) \sqsubseteq_{\mathcal{FV}_{\mathbb{D}'}} \sigma'(v)$$

Lemma 8.13.2.2.6 (Verband der DFA-Zustände)

$\widehat{\Sigma'} =_{df} (\Sigma', \sqcap_{\Sigma'}, \sqcup_{\Sigma'}, \sqsubseteq_{\Sigma'}, \sigma_\bot, \sigma_\top)$ ist ein vollständiger Verband mit

- kleinstem Element $\sigma_\bot$,
- größtem Element $\sigma_\top$,
- punktweisem Schnitt $\sqcap_{\Sigma'}$ und Vereinigung $\sqcup_{\Sigma'}$ als Schnitt- bzw. Vereinigungsoperation.

Kapitel 8.13.2.3

Einfache Konstanten: Spezifikation, Beweisverpflichtungen, Garantien

Einfache Konstanten: Spezifizieren der DFA

DFA-Spezifikation

- DFA-Verband

  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\Sigma', \sqcap_{\Sigma'}, \sqcup_{\Sigma'}, \sqsubseteq_{\Sigma'}, \sigma_\bot, \sigma_\top) = \widehat{\Sigma}'$

  mit Σ' Menge der DFA-Zustände über $\mathbb{Z}$.

- DFA-Semantik

  $[\![\ ]\!]_{eK} : E \to (\ \Sigma' \to \Sigma'\ )$

  $[\![\ ]\!]_{eK} =_{df} \lambda\, e.\ \theta'_{\iota_e}$

- Anfangszusicherung

  $\sigma_{\mathbf{s}} \in \Sigma'_{Init}$

Einfache-Konstanten-Spezifikation

- Spezifikation: $\mathcal{S}_G^{eK} = (\widehat{\Sigma}', [\![\ ]\!]_{eK}, \sigma_{\mathbf{s}})$

Einf. Konstanten: Erbringen der Beweisverpfl.

Lemma 8.13.2.3.1 (Absteigende Kettenbedingung)

$\widehat{\Sigma}'$ erfüllt die absteigende Kettenbedingung.

Beachte: Folgt aus der Endlichkeit der Menge der Variablen in einem Programm.

Lemma 8.13.2.3.2 (Monotonie)

$[\![\ ]\!]_{eK}$ ist monoton.

Lemma 8.13.2.3.3 (Nichtdistributivität)

$[\![\ ]\!]_{eK}$ ist (i.a.) nicht distributiv.

Einf. Konst.: Einsammeln d. *SUP*-S.-Garantien

...für Terminierung, Korrektheit.

Theorem 8.13.2.3.4 (*MaxFP*-Terminierung)

Angewendet auf $\mathcal{S}_G^{eK} = (\widehat{\Sigma}', [\![\ ]\!]_{eK}, \sigma_{\mathbf{s}})$ terminiert Algorithmus 8.8.1.1 mit der *MaxFP*-Semantik von $\mathcal{S}_G^{eK}$.

Beweis. Umittelbar mit Lemma 8.13.2.3.1, Lemma 8.13.2.3.2 und Terminierungstheorem 8.8.2.1.

Theorem 8.13.2.3.5 (*SUP*-Terminier., -Korrektheit)

Angewendet auf $\mathcal{S}_G^{eK} = (\widehat{\Sigma}', [\![\ ]\!]_{eK}, \sigma_{\mathbf{s}})$ ist Algorithmus 8.8.1.1 *SUP*-korrekt für $\mathcal{S}_G^{eK}$ (d.h. terminiert mit einer unteren Approximation der *SUP*-Semantik von $\mathcal{S}_G^{eK}$).

Beweis. Unmittelbar mit Lemma 8.13.2.3.1, Lemma 8.13.2.3.2, Sicherheitstheorem 8.10.1 und Terminierungstheorem 8.8.2.1.

Einfache Konstanten: Negatives Ergebnis

...für Akkuratheit.

Theorem 8.13.2.3.6 (Nicht-Akkuratheit)

Angewendet auf $\mathcal{S}_G^{eK} = (\widehat{\Sigma}', [\![\ ]\!]_{eK}, \sigma_{\mathbf{s}})$ ist Algorithmus 8.8.1.1 i.a. nicht *SUP*-akkurat für $\mathcal{S}_G^{eK}$ (d.h. terminiert mit einer echten unteren Approximation der *SUP*-Lösung von $\mathcal{S}_G^{eK}$).

Beweis. Unmittelbar mit Lemma 8.13.2.3.1, Lemma 8.13.2.3.2, Lemma 8.13.2.3.3, Koinzidenztheorem 8.10.2 und Terminierungstheorem 8.8.2.1.

Abschließend: Die *MaxFP*-Lösung von $\mathcal{S}_G^{eK}$ ist stets sichere Approximation der *SUP*-Lösung von $\mathcal{S}_G^{eK}$. I.a. sind die operationelle *SUP*-Lösung von $\mathcal{S}_G^{eK}$ und ihr denotationelles *MaxFP*-Gegenstück verschieden.

Einf. Konst.: Analog für *VUP*-Sem.-Garantien

Analog zu Lemma 8.13.2.3.1, 8.13.2.3.3 gilt offenbar auch:

1. $\widehat{\Sigma}'$ erfüllt die aufsteigende Kettenbedingung.
2. $[\![\ ]\!]_{eK}$ ist i.a. nicht additiv.

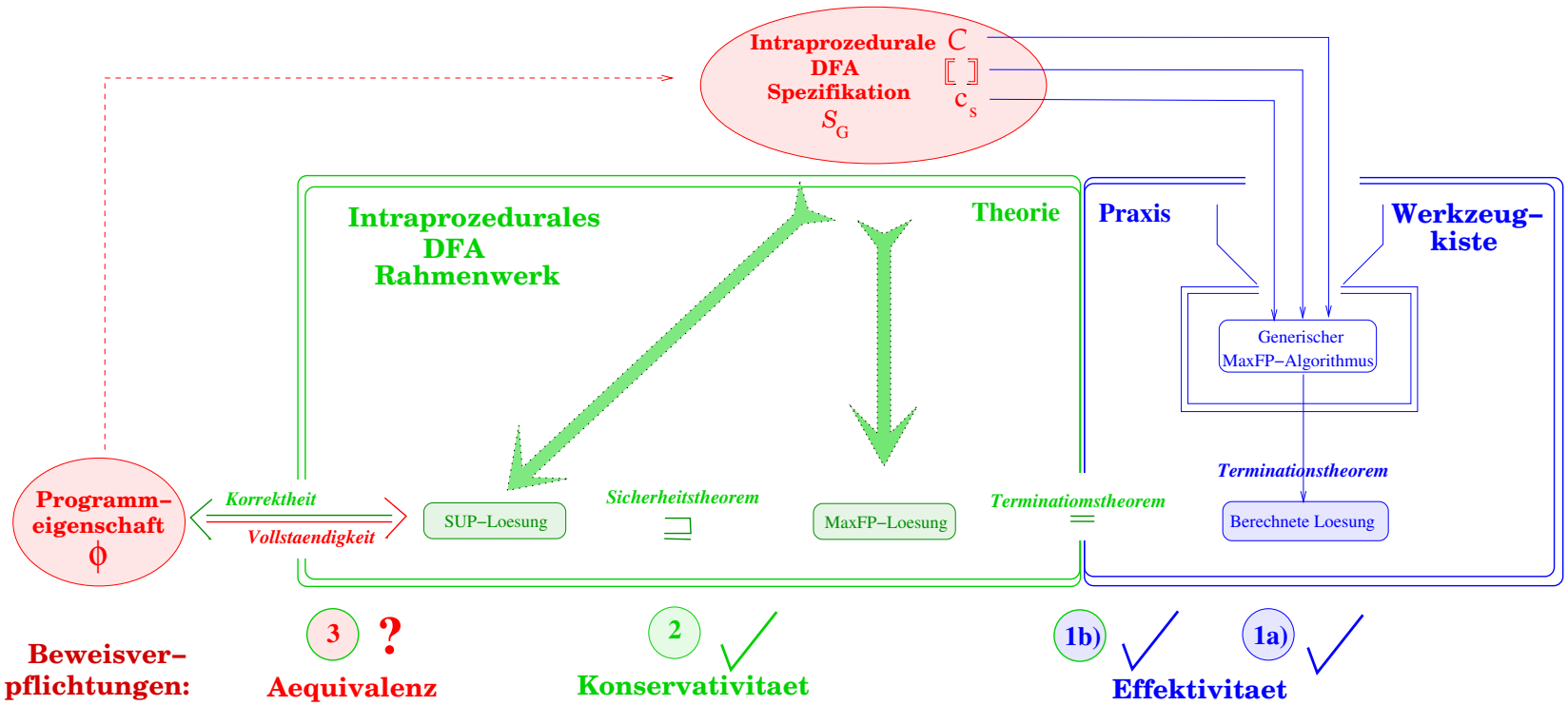
Damit gilt analog zu Theorem 8.13.2.3.5 auch für die *VUP*-Semantik das entsprechende Terminierungs- und Korrektheitsresultat:

Theorem 8.13.2.3.7 (*VUP*-Terminier., -Korrektheit)

Angewendet auf das duale DFA-Problem von $\mathcal{S}_G^{av,t}$ mit gestürztem Verband $\widehat{\Sigma}'$ ist Algorithmus 8.8.1.1 *VUP*-korrekt für $\mathcal{S}_G^{eK}$ (d.h. terminiert mit einer sicheren oberen Approximation der *VUP*-Semantik).

Einf. Konst.: Schließen äußerer Beweislücke

...Korrektheits und Vollständigkeitsbeweisverpflichtungen für die *SUP*-Sicht von $\mathcal{S}_G^{eK}$ für die einfache-Konstanten-Eigenschaft:



Einf. Konstanten: Korrektheit, Vollständigkeit

...für die *SUP*-Semantik.

Theorem 8.13.2.3.8 (Korrektheit, Vollständigkeit)

Die *SUP*-Semantik von $\mathcal{S}_G^{eK}$ ist

1. korrekt und vollständig für Variablen.
2. korrekt, aber nicht vollständig für (nichttriviale) Terme (d.h. für Terme mit mindestens einem (nichteinstelligen) Operatorsymbol).

...zu Theorem 8.13.2.3.8(2): Beachte, dass die *SUP*-Lösung an jedem Knoten als Zustand aufgefasst werden kann, d.h. als Abbildung von Variablen auf Werte, was die Auswertung von Termen gemäß Definition 8.13.2.2.4 erlaubt.

Einf. Konstanten: Korrektheit, Vollständigkeit

...für die *MaxFP*-Semantik.

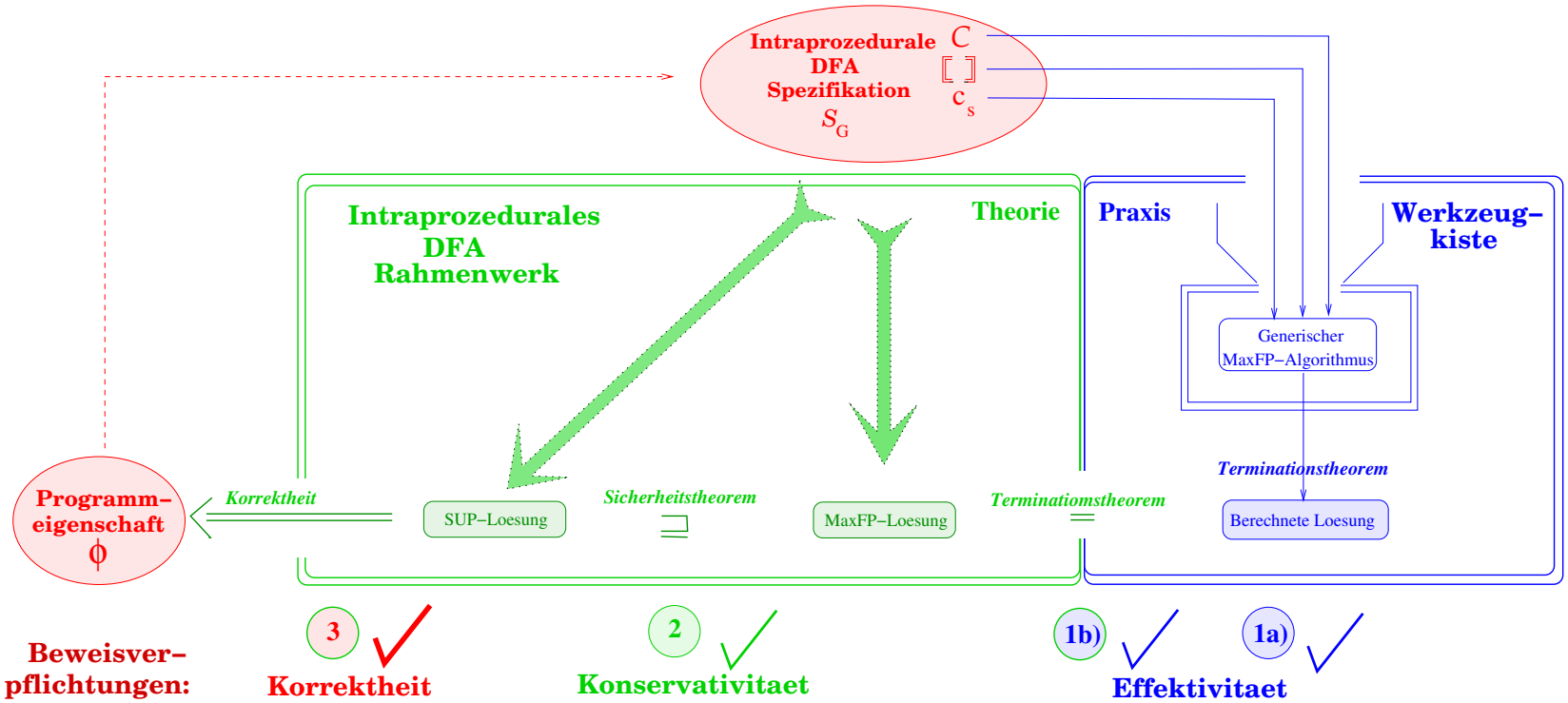
Theorem 8.13.2.3.9 (Korrektheit, Vollständigkeit)

Die *MaxFP*-Semantik von $\mathcal{S}_G^{eK}$ ist korrekt, aber nicht vollständig (sowohl für Terme als auch für Variablen).

...siehe Unterlagen zur LVA 185.A04 Optimierende Übersetzer für weitere Details.

Beweislücke partiell geschlossen: Korrektheit

...der *SUP*-Sicht von $\mathcal{S}_G^{eK}$ für die einfache-Konstanten-Eigenschaft bewiesen:



Übungsaufgabe 8.13.2.3.10

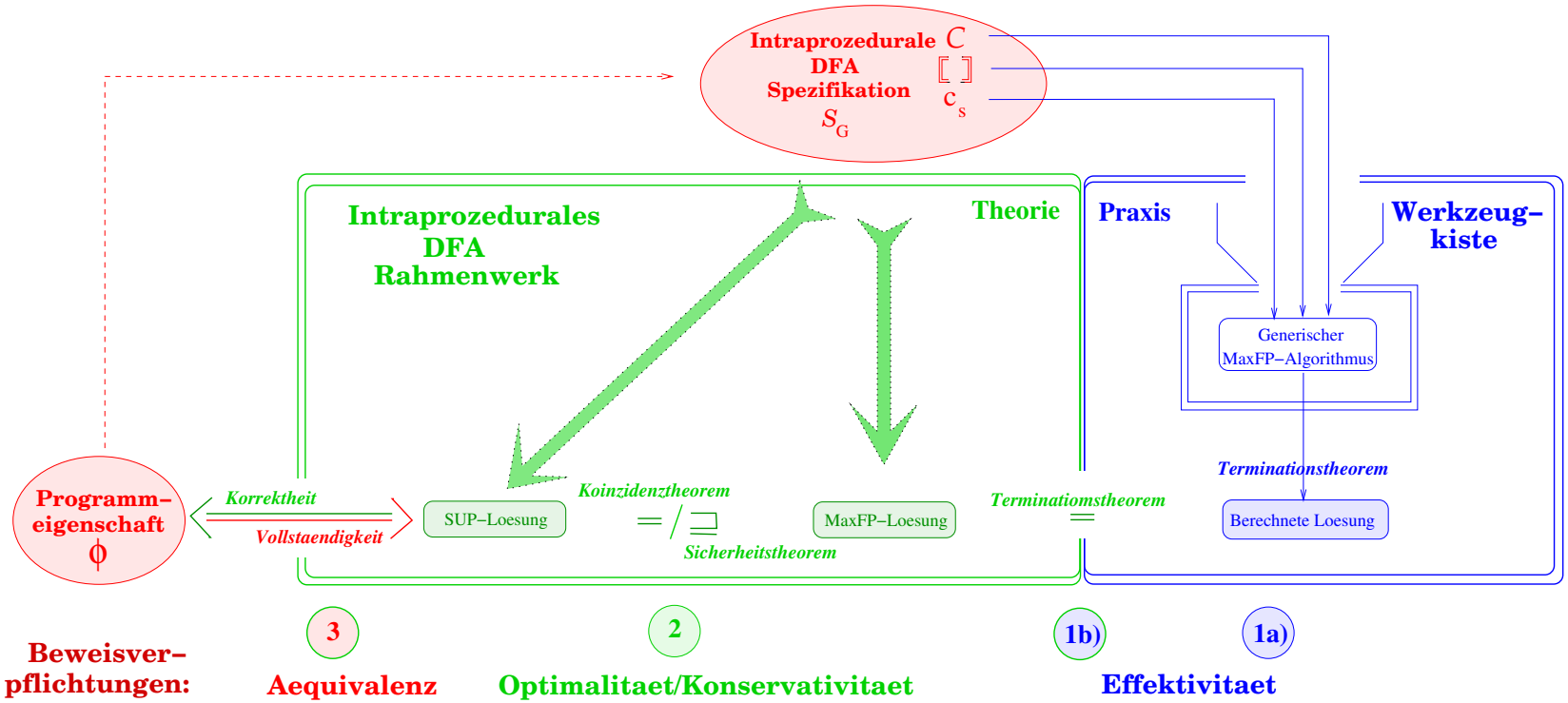
1. Was bedeuten Korrektheit, Vollständigkeit für die *VUP*-Sicht von $\mathcal{S}_G^{eK,G}$ für die einfache-Konstante-Eigenschaft?
2. Wie können sie bewiesen/widerlegt werden?

Kapitel 8.14

Zusammenfassung, Ausblick

Die Rahmenwerk- und Werkzeugkistensicht

...von (intraprozeduraler) *SUP/MaxFP*-Datenflussanalyse:



...unter der Perspektive von Korrektheit, Vollständigkeit und der muss/kann-Eigenschaft von ϕ.

Muss vs. kann-Eigenschaften

...grundsätzlich können wir für ϕ unterscheiden:

- Universell quantifizierte (oder: muss (engl. must)) Eigenschaften $\phi^{\forall}$: $\phi^{\forall}$ gilt an einem Knoten n, wenn ϕ entlang aller Pfade von **s** nach n an n gilt.
- Existentiell quantifizierte (oder: kann (engl. may)) Eigenschaften $\phi^{\exists}$: $\phi^{\exists}$ gilt an einem Knoten n, wenn ϕ entlang einiger Pfade von **s** nach n an n gilt.

Muss-Eigenschaften $\phi^{\forall}$ sind verbunden mit der

- operationellen *SUP*-Programmsemantik und ihrem berechenb. denotationellen Gegenstück, der *MaxFP*-Sem.

Kann-Eigenschaften $\phi^{\exists}$ sind verbunden mit der

- operationellen *VUP*-Programmsemantik und ihrem berechenb. denotationellen Gegenstück, der *MinFP*-Sem.

Korrektheit, Vollständigkeit

...es gibt zwei wesentliche Stellen, an denen Korrektheit und Vollständigkeit auf unterschiedlicher Ebene in der Rahmenwerk- und Werkzeugkisten-Sicht von DFA betrachtet werden:

Rahmenwerk/Werkzeugkisten-**intern**: Erfasst durch

- **Sicherheit** $\rightsquigarrow$ Korrektheit der Fixpunktlösungen
- **Koinzidenz** $\rightsquigarrow$ Akkuratheit der Fixpunktlösungen

...*MaxFP*/*MinFP*- und *SUP*/*VUP*-Lsg. in Beziehung setzend.

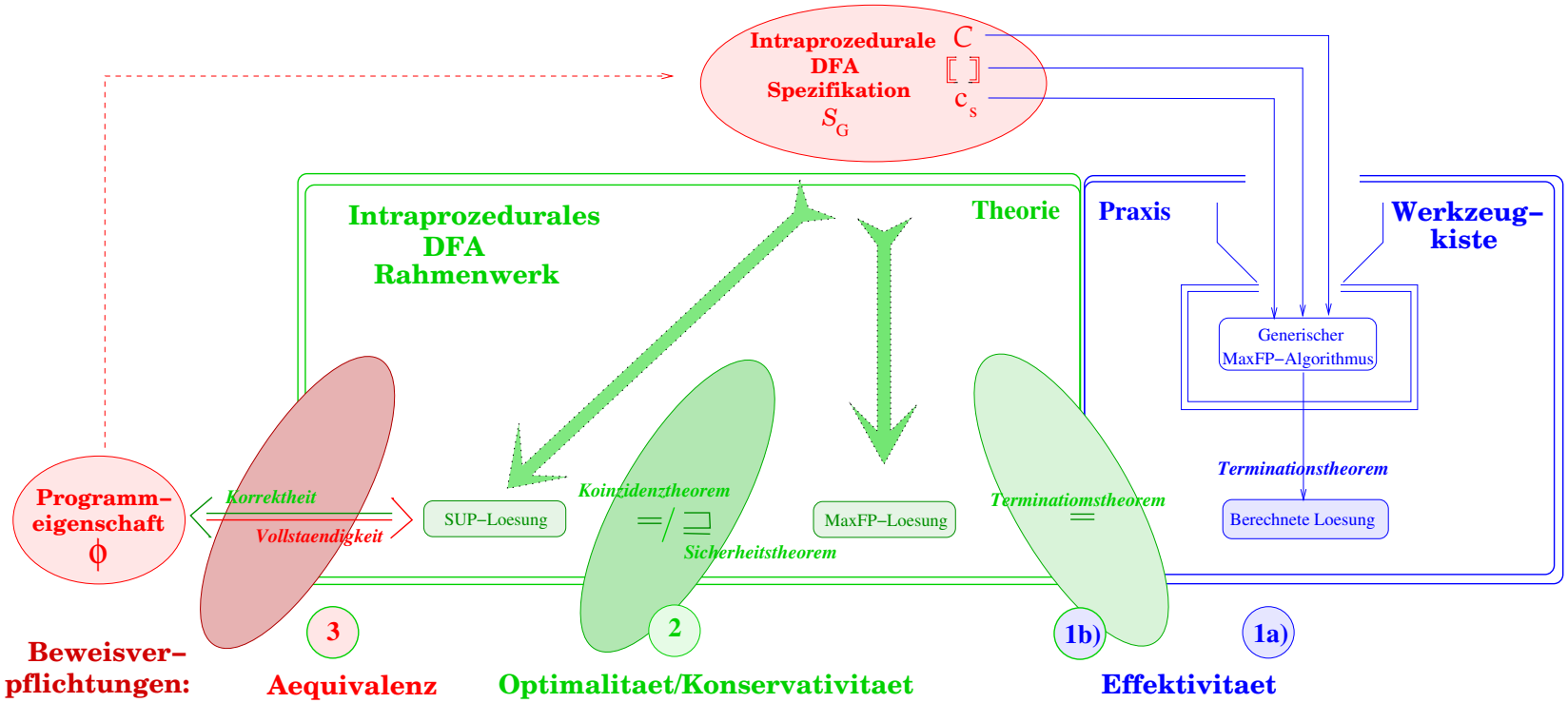
Rahmenwerk/Werkzeugkisten-**extern**: Erfasst durch

- **Korrektheit** $\rightsquigarrow$ Keine falschen Positive
- **Vollständigkeit** $\rightsquigarrow$ Keine falschen Negative

...*SUP*/*VUP*-Lsg. und $\phi^{\forall}/\phi^{\exists}$-Eigensch. in Beziehung setzend.

Veranschaulichung

...der Stellen Rahmenwerk/Werkzeugkisten-interner und -externer Behandlung von Korrektheit und Vollständigkeit:

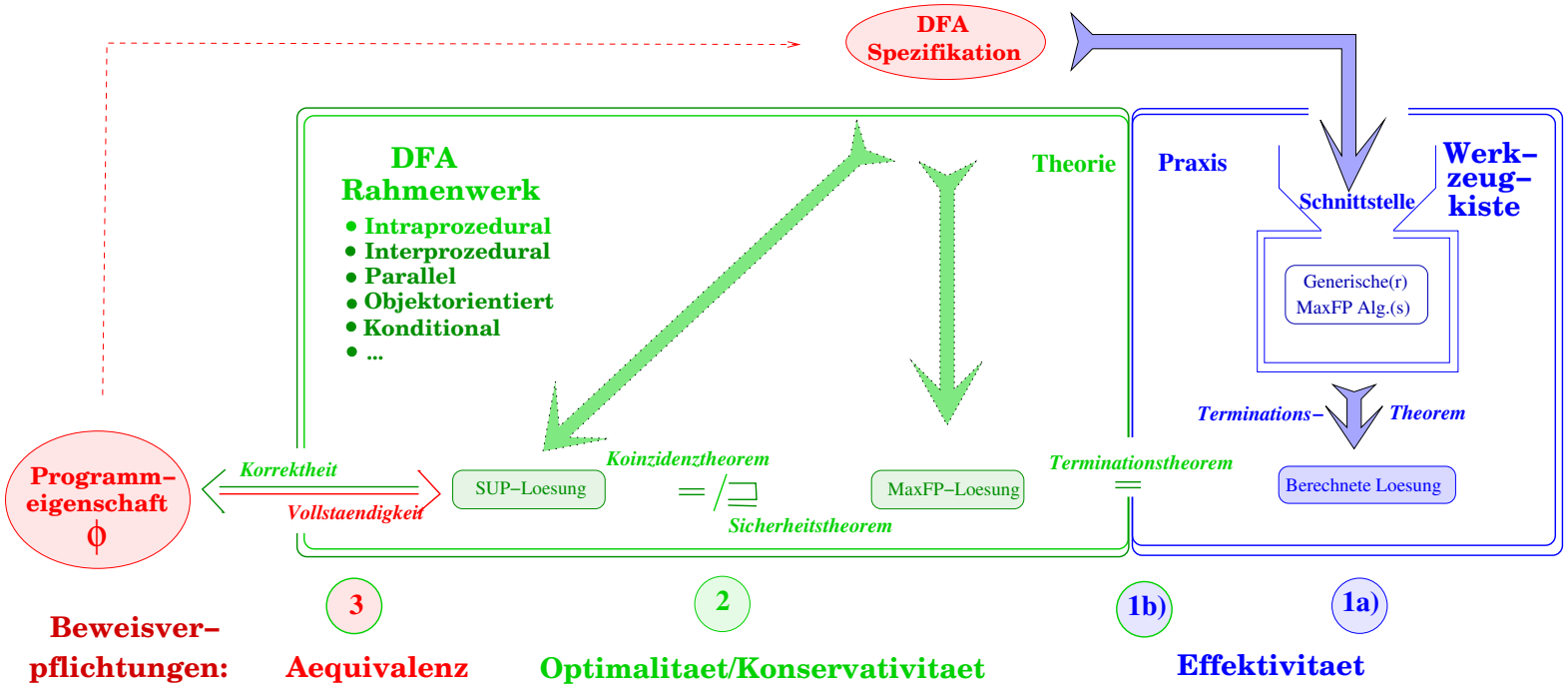


Ausblick: Die einheitliche Sicht von DFA

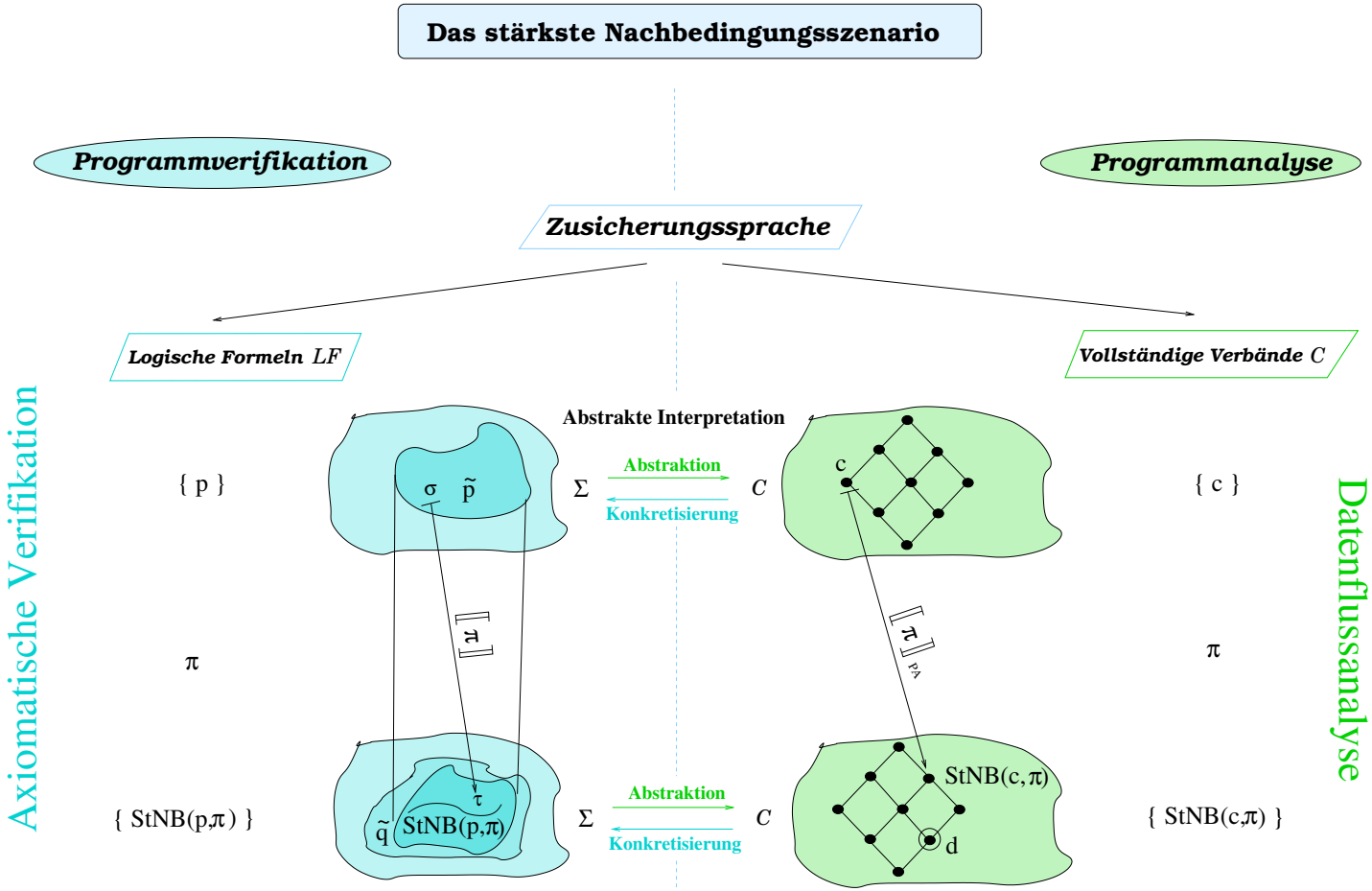
...werden wir im Lauf dieser (und auch der Vorlesung LVA 185.A04 Optimierende Übersetzer) sehen: Die

– Rahmenwerk- und Werkzeugkistensicht

ist über den Grundfall intraprozeduraler DFA hinaus erreichbar und erlaubt anwendungsszenariounabhängig eine einheitliche Sicht von DFA:



Verifikation u. Datenflussanalyse im Vergleich



StNB(p, π) ∈ *LF* *muss erfüllen:*

(1) $\models_{PV}$ { p } π { StNB(p,π) }

(2) ∀ q ∈ *LF*. $\models_{PV}$ { p } π { q } impliziert StNB(p,π) $\Longrightarrow$ q

StNB(c,π) ∈ *C* *muss erfüllen:*

(1) $\models_{PA}$ { c } π { StNB(c,π) }

(2) ∀ d ∈ *C*. $\models_{PA}$ { c } π { d } impliziert StNB(c,π) $\sqsupseteq$ d

Kapitel 8.15

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (1)

Lehrbuchdarstellungen

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley, 2nd edition, 2007. (Chapter 1.2, The Structure of a Compiler; Chapter 1.4, The Science of Building a Compiler; Chapter 1.4.2, The Science of Code Optimization; Chapter 9.1, The Principal Sources of Program Optimization)

Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004. (Appendix B.3.1, Graphical Intermediate Representations)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (2)


Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.


Uday P. Khedker, Amitabha Sanyal, Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press, 2009. (Chapter 3, Theoretical Abstractions in Data Flow Analysis; Chapter 4, General Data Flow Frameworks; Chapter 5, Complexity of Iterative Data Flow Analysis)


Robert Morgan. *Building an Optimizing Compiler*. Digital Press, 1998. (Chapter 2.3, Building the Flow Graph; Chapter 4.7, Structure of Program Flow Graph)


Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997. (Chapter 7, Control-Flow Analysis)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (3)


Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019. (Chapter 1, Program Graphs)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 5, Static Program Analysis)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 7, Program Analysis; Chapter 8, More on Program Analysis; Appendix B, Implementation of Program Analysis)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (4)

Grundlegende, wegweisende Arbeiten


Frances E. Allen, John A. Cocke. *A Program Data Flow Analysis Procedure*. Communications of the ACM 19(3):137-147, 1976.


Susan Horwitz, Alan J. Demers, Tim Teitelbaum. *An Efficient General Iterative Algorithm for Dataflow Analysis*. Acta Informatica 24(6):679-694, 1987.


Gary A. Kildall. *A Unified Approach to Global Program Optimization*. In Conference Record of the 1st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'73), 194-206, 1973.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (5)

John B. Kam, Jeffrey D. Ullman. *Global Data Flow Analysis and Iterative Algorithms*. Journal of the ACM 23:158-171, 1976.

John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.

Rahmenwerke, Werkzeugkisten

Marion Klein, Jens Knoop, Dirk Koschützki, Bernhard Steffen. *DFA&OPT-METAFrame: A Toolkit for Program Analysis and Optimization*. In Proceedings of the 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96), Springer-V., LNCS 1055, 422-426, 1996.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (6)

Jens Knoop. *From DFA-Frameworks to DFA-Generators: A Unifying Multiparadigm Approach*. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Springer-V., LNCS 1579, 360-374, 1999.

Thomas J. Marlowe, Barbara G. Ryder. *Properties of Data Flow Frameworks*. Acta Informatica 28(2):121-163, 1990.

Stephen P. Masticola, Thomas J. Marlowe, Barbara G. Ryder. *Lattice Frameworks for Multisource and Bidirectional Data Flow Problems*. ACM Transactions on Programming Languages and Systems (TOPLAS) 17(5):777-803, 1995.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (7)

Florian Martin. *PAG - An Efficient Program Analyzer Generator*. Journal of Software Tools for Technology Transfer 2(1):46-67, 1998.

Flemming Nielson. *Semantics-directed Program Analysis: A Tool-maker's Perspective*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 2-21, 1996.

Lösung von Gleichungssystemen, Fixpunktberechnung

Christian Fecht, Helmut Seidl. *An Even Faster Solver for General Systems of Equations*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 189-204, 1996.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (8)


Christian Fecht, Helmut Seidl. *Propagating Differences: An Efficient New Fixpoint Algorithm for Distributive Constraint Systems*. In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 90-104, 1998.


Christian Fecht, Helmut Seidl. *A Faster Solver for General Systems of Equations*. Science of Computer Programming 35(2):137-161, 1999.


Bernhard Steffen, Andreas Claßen, Marion Klein, Jens Knoop, Tiziana Margaria. *The Fixpoint Analysis Machine*. In Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95), Springer-V., LNCS 962, 72-87, 1995.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (9)

Flussgraphpragmatik

Larry Carter, Jeanne Ferrante, Clark Thomborson. *Folklore Confirmed: Reducible Flow Graphs are Exponentially Larger*. In Conference Record of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2003), 106-114, 2003.

Jens Knoop, Dirk Koschützki, Bernhard Steffen. *Basic-block Graphs: Living Dinosaurs?* In Proceedings of the 7th International Conference on Compiler Construction (CC'98), Springer-V., LNCS 1383, 65-79, 1998.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (10)

Verschiedenes

Stephen M. Blackburn, Amer Diwan, Matthias Hauswirth, Peter F. Sweeny, José Nelson Amaral, Tim Brecht, Lubomír Bulej, Cliff Click, Lieven Eeckhout, Sebastian Fischmeister, Daniel Frampton, Laurie J. Hendren, Michael Hind, Antony L. Hosking, Richard E. Jones, Tomas Kalibera, Nathan Keynes, Nathaniel Nystrom, Andreas Zeller. *The Truth, The Whole Truth, and Nothing But the Truth: A Pragmatic Guide to Assessing Empirical Evaluations*. ACM Transactions on Programming Languages and Systems 38(4), Article 15:1-20, 2016.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (11)

- Martin Davis. *Hilbert's Tenth Problem is Unsolvable*. American Mathematical Monthly 80:33-269, 1973.
- Martin Davis, Yuri Matijasevič, Julia Robinson. *Hilbert's Tenth Problem. Diophantine Equations: Positive Aspects of a Negative Solution*. In Proceedings of the Symposium on the Hilbert Problems (De Kalb, Illinois), May 1974, American Mathematical Society, Providence, R.I., 323-378, 1976.
- William Landi. *Undecidability of Static Analysis*. ACM Letters on Programming Languages and Systems 1(4):323-337, 1992.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (12)


Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009. (Chapter 7, What can one tell about a Program without its Execution: Static Analysis)


Yuri V. Matijasevič. *Enumerable Sets are Diophantine (auf Russisch)*. Doklady Akademii Nauk SSSR 191:279-282, 1970 (englische Übersetzung: Soviet Mathematics Doklady 11:354-357, 1970.


Yuri V. Matijasevič. *On Recursive Unsolvability of Hilbert's Tenth Problem*. In Proceedings of the 4th International Congress on Logic, Methodology and Philosophy of Science (Bucharest 1971), North-Holland, Amsterdam, 89-110, 1973.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (13)

- Yuri V. Matijasevič. *What Should We Do Having Proved a Decision Problem to be Unsolvable?* In Proceedings of Algorithms in Modern Mathematics and Computer Science, Springer-V., LNCS 122, 441-448, 1979.
- Yuri V. Matijasevič. *Hilbert's Tenth Problem*. MIT Press, 1993.
- Markus Müller-Olm, Helmut Seidl. *Polynomial Constants are Decidable*. In Proceedings of the 9th Static Analysis Symposium (SAS 2002), Springer-V., LNCS 2477, 4-19, 2002.

Kapitel 9

Reverse abstrakte Semantiken, reverse Analysesemantiken

Kapitel 9.1

Analyse vs. reverse Analyse

Analyse vs. reverse Analyse

...adressieren zwei unterschiedliche Fragestellungen.

Für gegebenen Analysefakt $c \in \mathcal{C}$ als

A: Anfangszusicherung am Startknoten **s** zielt Analyse für jeden Programmpunkt n auf die Berechnung des

- stärkst möglichen Analysefakts

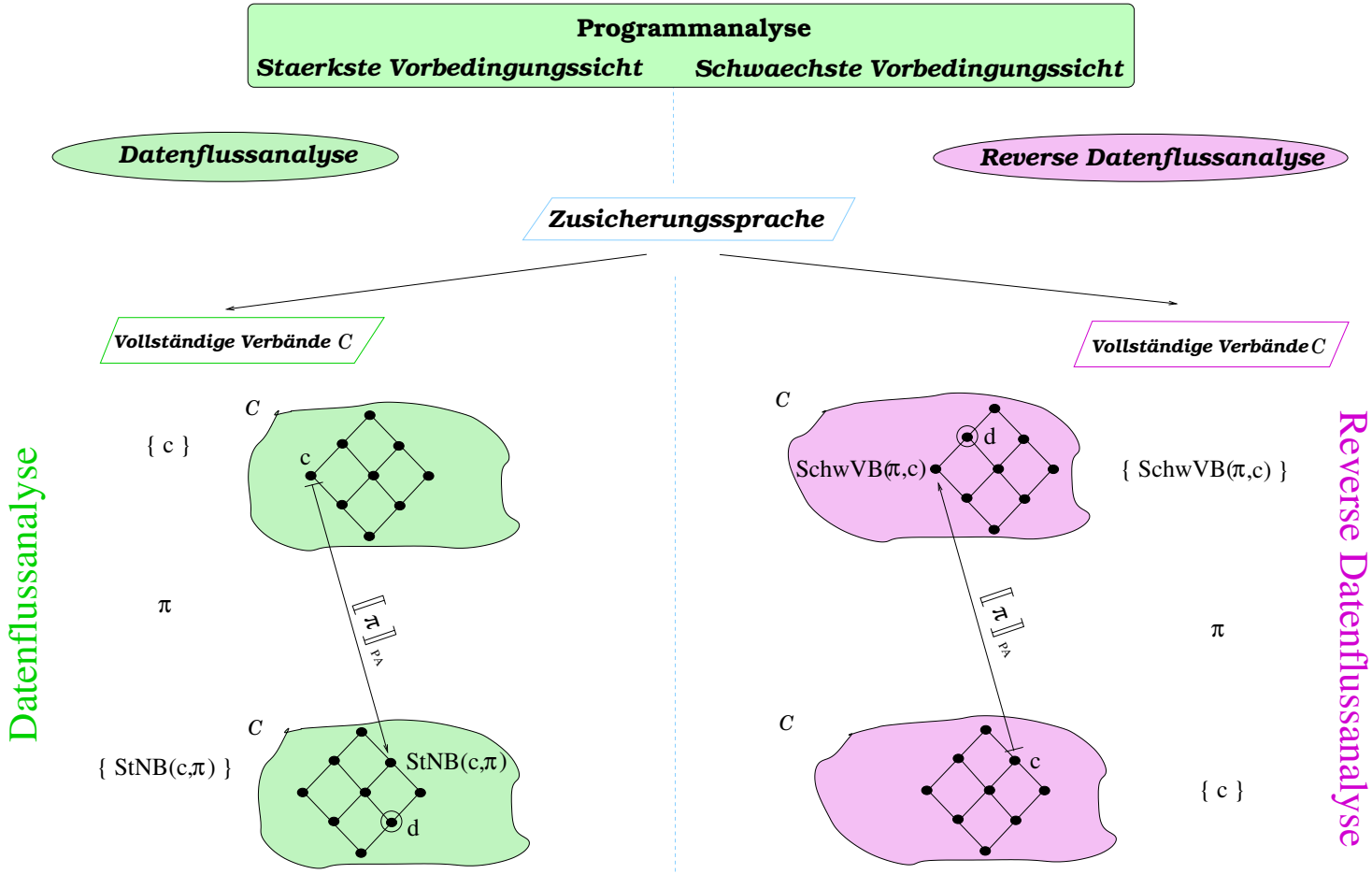
der an n durch c an **s** sichergestellt ist.

RA: Endzusicherung am Endknoten **e** zielt reverse Analyse für jeden Programmpunkt n auf die Berechnung des

- schwächst möglichen Analysefakts

der an n erfüllt sein muss, um die Gültigkeit von c an **e** sicherzustellen.

Veranschaulichung v. Analyse u. reverser Analyse



StNB(c,π) ∈ C *muss erfüllen:*

(1) $\models_{PA}$ { c } π { StNB(c,π) }

(2) ∀ d ∈ C. $\models_{PA}$ { c } π { d } impliziert StNB(c,π) ⊒ d

SchwVB(π,c) ∈ C *muss erfüllen:*

(1) $\models_{PA}$ { SchwVB(π,c) } π { c }

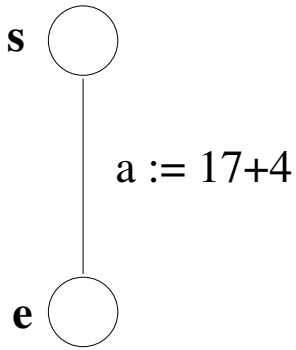
(2) ∀ d ∈ C. $\models_{PA}$ { d } π { c } impliziert d ⊒ SchwVB(π,c)

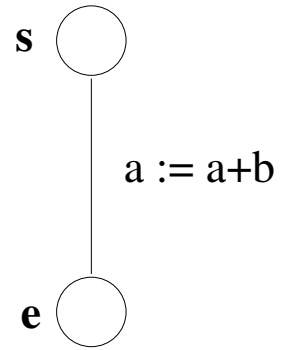
Kapitel 9.2

Fehlschlagserweiterung von Verbänden

Hintergrund

Bestimmte Analysefakten können für keine Anfangszusichrung an einem Programmpunkt erfüllt sein, etwa a=42 oder verfuegbar(a+b) jeweils am Knoten **e**:





Analyseverbände (wie die Verbände einfacher Konstanten oder der Wahrheitswerte) enthalten i.a. kein Element, die das anzuzeigen erlauben. Wir erweitern daher Analyseverbände um ein

- neues größtes Element Ψ.

Informell steht Ψ für eine nicht mögliche Zusicherung, für ein Unerfüllbarkeit anzeigendes Verbandselement, ein Fehlschlagselement.

Verbandserweiterung um Fehlschlagselement Ψ

...sei $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, , \sqcup, \bot, \top)$ ein Analyseverband, Ψ ein neues Element nicht in $\mathcal{C}$ und $\mathcal{C}_\Psi =_{df} \mathcal{C} \dot{\cup} \{\Psi\}$.

Definition 9.2.1 (Ψ-erweiterter Analyseverband)

Die Ψ-Erweiterung von $\widehat{\mathcal{C}}$ ist der Verband $\widehat{\mathcal{C}}_\Psi$:

$$\widehat{\mathcal{C}}_\Psi =_{df} (\mathcal{C}_\Psi, \sqsubseteq_\Psi, \sqcap_\Psi, \sqcup_\Psi, \bot, \Psi)$$

mit Ψ als größtem Element von $\widehat{\mathcal{C}}_\Psi$ bzgl. der Ordung $\sqsubseteq_\Psi$ mit:

1. $\forall\, c \in \mathcal{C}_\Psi.\ c \sqsubseteq_\Psi \Psi$
2. $\forall\, c, c' \in \mathcal{C}.\ c \sqsubseteq_\Psi c'$ gdw $c \sqsubseteq c'$

Es gilt:

- Mit $\sqsubseteq_\Psi$, sind auch $\sqcap_\Psi$, und $\sqcup_\Psi$ eindeutig festgelegt (s. Anhang A.4.6).
- Ist $\widehat{\mathcal{C}}$ vollständig, so ist auch $\widehat{\mathcal{C}}_\Psi$ vollständig.

Kapitel 9.3

Induzierte reverse lokale abstrakte Semantiken

Ψ-Ausdehnung lokaler abstrakter Semantiken

...lokale abstrakte Semantiken $(\widehat{\mathcal{C}}, [\![\ ]\!])$ werden in folgender Weise auf den Ψ-erweiterten Analyseverband $\widehat{\mathcal{C}}_\Psi$ ausgedehnt:

Definition 9.3.1 (Ψ-Ausdehnung lok. abstr. Sem. $[\![\ ]\!]$)

Die Ψ-Ausdehnung von $(\widehat{\mathcal{C}}, [\![\ ]\!])$ ist das Paar $(\widehat{\mathcal{C}}_\Psi, [\![\ ]\!]_\Psi)$ mit

- $\widehat{\mathcal{C}}_\Psi$: Ψ-Erweiterung von $\widehat{\mathcal{C}}$
- $[\![\ ]\!]_\Psi$: Ψ-Ausdehnung von $[\![\ ]\!]$ definiert durch:

$$[\![\ ]\!]_\Psi : E \to (\widehat{\mathcal{C}}_\Psi \to \widehat{\mathcal{C}}_\Psi)$$

$$\forall\, e \in E.\ [\![\, e \,]\!]_\Psi =_{df} \lambda\, c \in \widehat{\mathcal{C}}_\Psi.\ \begin{cases} [\![\, e \,]\!](c) & \text{falls } c \neq \Psi \\ \Psi & \text{sonst} \end{cases}$$

Eigenschaften der Ψ-Ausdehnung

Für die Ψ-Ausdehnung $(\widehat{\mathcal{C}}_\Psi, [\![\ ]\!]_\Psi)$ von $(\widehat{\mathcal{C}}, [\![\ ]\!])$ gilt:

Lemma 9.3.2

Der Verband $\widehat{\mathcal{C}}_\Psi$

1. ist vollständig, wenn $\widehat{\mathcal{C}}$ vollständig ist.
2. erfüllt die aufsteigende (absteigende) Kettenbedingung gdw $\widehat{\mathcal{C}}$ die aufsteigende (absteigende) Kettenbedingung erfüllt.

Lemma 9.3.3

1. $[\![\ ]\!]_\Psi$ monoton gdw $[\![\ ]\!]$ monoton.
2. $[\![\ ]\!]_\Psi$ distributiv gdw $[\![\ ]\!]$ distributiv.
3. $[\![\ ]\!]_\Psi$ additiv gdw $[\![\ ]\!]$ additiv.

Anm.: Die Umkehrung von Lemma 9.3.2(1) gilt nicht.

Induzierte reverse lokale abstrakte Semantiken

Sei $(\widehat{\mathcal{C}}_\Psi, [\![\ ]\!]_\Psi)$ die Ψ-Ausdehnung von $(\widehat{\mathcal{C}}, [\![\ ]\!])$ über vollständigem Verband $\widehat{\mathcal{C}}$.

Definition 9.3.4 (Reverse lokale abstrakte Semantik)

Die lokale abstrakte Semantik $[\![\ ]\!]_\Psi$ induziert eine reverse lokale abstrakte Semantik $[\![\ ]\!]_R$, die definiert ist durch:

$$[\![\ ]\!]_R : E \to (\mathcal{C}_\Psi \to \mathcal{C}_\Psi)$$

$$\forall\, e \in E\ .\ [\![\, e\,]\!]_R(c) =_{df}\ \lambda\, c \in \mathcal{C}_\Psi .\ \bigsqcap \{\, c' \mid [\![\, e\,]\!]_\Psi(c') \sqsupseteq c\,\}$$

Intuitiv: Die Gültigkeit der Inklusion $[\![\, e\,]\!]_\Psi(c') \sqsupseteq c$ bedeutet, dass c' am Eingang von e mindestens c (oder einen noch größeren Analysefakt als c) am Ausgang von e garantiert.

Eigenschaften von $[\![\ ]\!]$, $[\![\ ]\!]_\Psi$ und $[\![\ ]\!]_R$

Sei $[\![\ ]\!]_\Psi$ die Ψ-Ausdehnung von $[\![\ ]\!]$ und sei $[\![\ ]\!]_R$ die von $[\![\ ]\!]_\Psi$ induzierte reverse Semantik. Dann gilt:

Lemma 9.3.5

1. $\forall\, e \in E.\ [\![\ e\ ]\!]_R$ ist wohldefiniert und monoton.
2. $\forall\, e \in E.\ [\![\ e\ ]\!]_R$ ist additiv, falls $[\![\ e\ ]\!]_\Psi$ distributiv ist.

Lemma 9.3.6

1. $\forall\, e \in E.\ [\![\ e\ ]\!]_R \circ [\![\ e\ ]\!]_\Psi \sqsubseteq Id_{\mathcal{C}_\Psi}$, falls $[\![\ e\ ]\!]$ monoton ist.
2. $\forall\, e \in E.\ [\![\ e\ ]\!]_\Psi \circ [\![\ e\ ]\!]_R \sqsupseteq Id_{\mathcal{C}_\Psi}$, falls $[\![\ e\ ]\!]$ distributiv ist.

...in der Sprechweise der Theorie 'Abstrakter Interpretationen':

– $[\![\ e\ ]\!]_\Psi$ und $[\![\ e\ ]\!]_R$ bilden eine Galois-Verbindung (s. Kap. 16.2.1).

Kapitel 9.4

Reverse operationelle globale abstrakte Semantiken

Kapitel 9.4.1

Pfadausdehnung reverser lokaler Semantiken

Pfadausdehnung reverser lokaler Semantiken

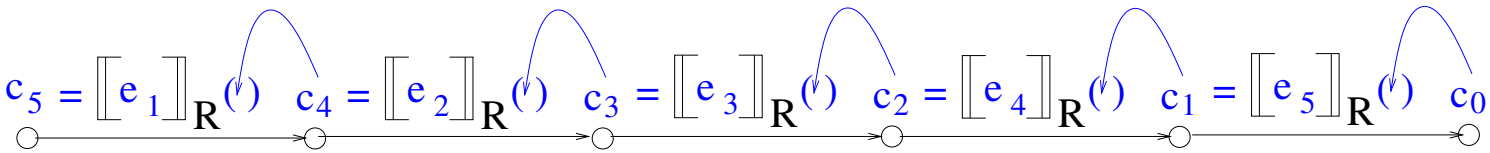
Definition 9.4.1.1 (Pfadausdehnung von $[\![\ ]\!]_R$)

Die Pfadausdehnung $[\![\ p\ ]\!]_R$ einer reversen lokalen abstrakten Semantik auf einen Pfad $p = \langle e_1, \ldots, e_{q-1}, e_q \rangle$ ist kompositionell definiert durch:

$$[\![\ p\ ]\!]_R =_{df} \begin{cases} Id_{\mathcal{C}_\Psi} & \text{falls } \lambda_p < 1 \\ [\![\ \langle e_1, \ldots, e_{q-1} \rangle\ ]\!]_R \circ [\![\ e_q\ ]\!]_R & \text{sonst} \end{cases}$$

wobei $Id_{\mathcal{C}_\Psi} = \lambda c \in \mathcal{C}_\Psi .\, c$ die Identität auf $\mathcal{C}_\Psi$ bezeichnet.

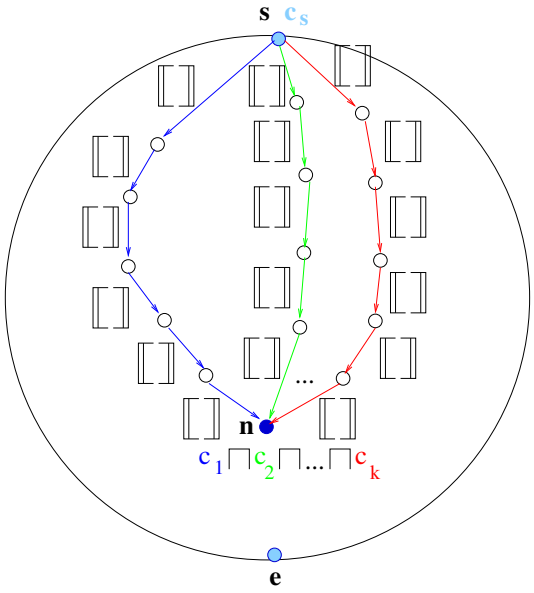
Veranschaulichung der Pfadausdehnung von $[\![\ ]\!]_R$:



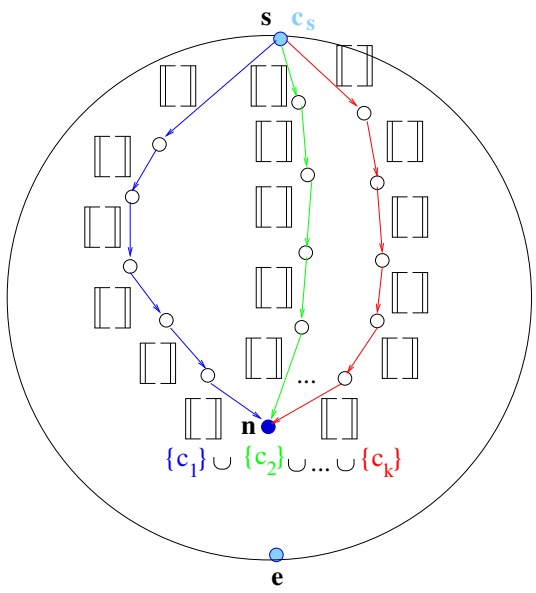
Beachte: Die Pfadausdehnung durchläuft p rückwärts.

!!! Ausgehend von der Pfadausdehnung !!!

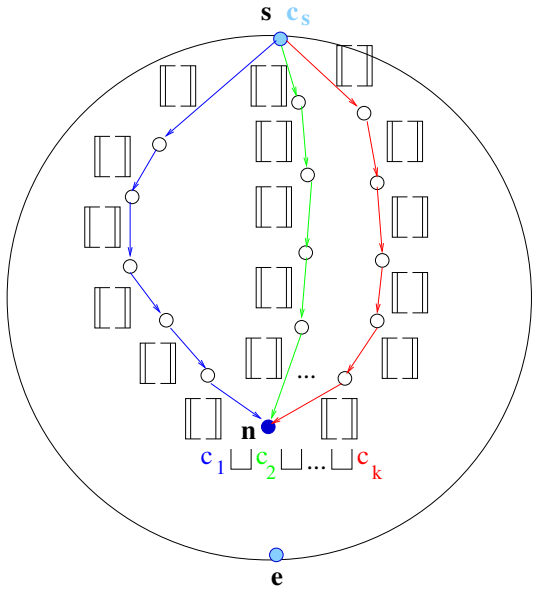
...einer reversen lokalen abstrakten Semantik $[\![\ ]\!]_R$, können wir jetzt analog zu Kapitel 7 drei Globalisierungstrategien zur Ausdehnung der reversen Semantik von Pfaden auf vollständige Flussgraphen angeben:



Schnitt–ueber–alle–Pfade–Semantik



Aufsammelsemantik



Vereinigung–ueber–alle–Pfade–Semanti

Sei in der Folge

...von Kapitel 9.4:

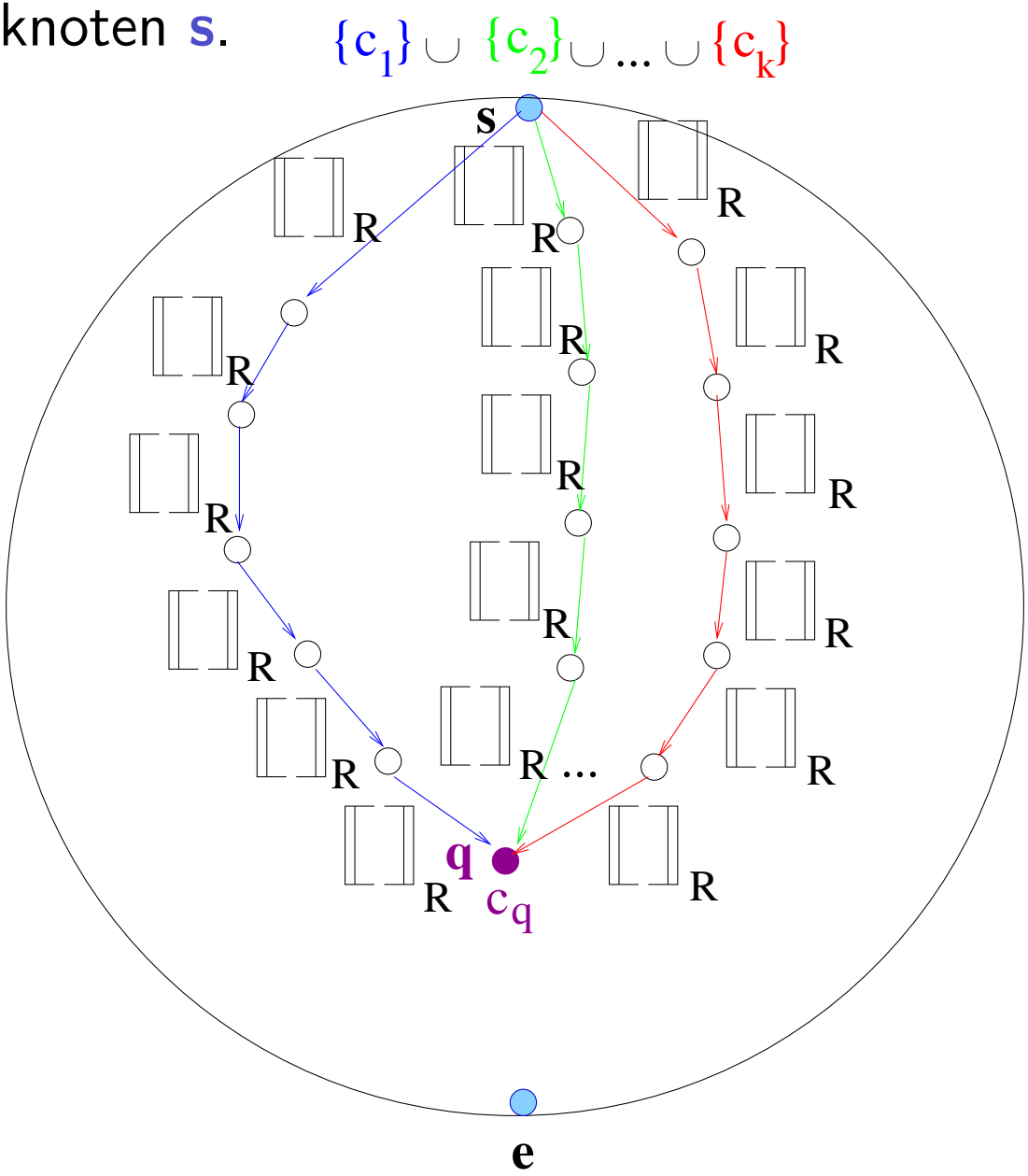
- $G = (N, E, \mathbf{s}, \mathbf{e})$ ein kantenbenannter Flussgraph
- $\widehat{\mathcal{C}_\Psi}$ ein Ψ-erweiterter vollständiger Verband
- $[\![\ ]\!]_R : E \to (\mathcal{C}_\Psi \to \mathcal{C}_\Psi)$ eine reverse lokale abstrakte Semantik für G

Kapitel 9.4.2

Reverse Aufsammelsemantik

Veranschaul. der reversen Aufsammelsemantik

...am Startknoten **s**.



Reverse Aufsammelsemantik

Definition 9.4.2.1 (Reverse Aufsammelsemantik)

Die von $[\![\ ]\!]_R$ induzierte reverse Aufsammelsemantik (oder: nichtdeterministische reverse globale abstrakte Semantik) (engl. reverse collecting semantics) von G ist definiert durch:

$$[\![\ ]\!]_{RAS} : \mathcal{C} \to N \to \mathcal{P}(\mathcal{C}_\Psi)$$

$$[\![\ ]\!]_{RAS} =_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \{\, [\![\, p \,]\!]_R(c) \mid p \in \mathbf{P}[n, \mathbf{e}] \,\}$$

wobei $\mathcal{P}$ den Potenzmengenoperator bezeichnet.

Ohne besondere Anforderungen an $\widehat{\mathcal{C}_\Psi}$ und $[\![\ ]\!]_R$ gilt:

Lemma 9.4.2.2 (RAS-Wohldefiniertheit)

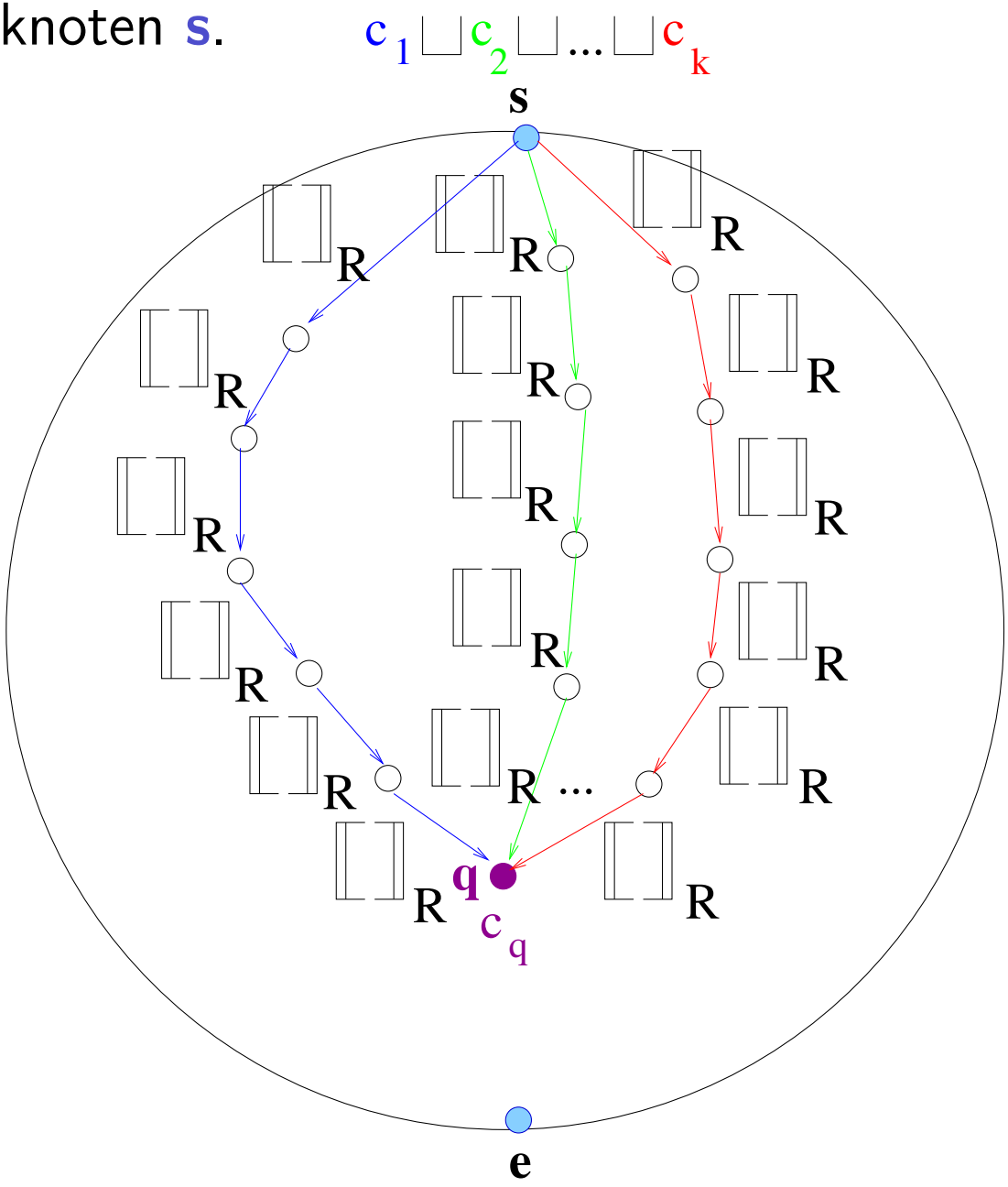
Die RA-Semantik $[\![\ ]\!]_{RAS}$ von G ist wohldefiniert.

Kapitel 9.4.3

Reverse Vereinigung-über-alle-Pfade-Semantik

Veranschaulichung der *RVUP*-Semantik

...am Startknoten **s**.



Die *RVUP*-Semantik

Definition 9.4.3.1 (*RVUP*-Semantik)

Die von $[\![\ ]\!]_R$ induzierte *RVUP*-Semantik (oder: deterministische reverse globale Vereinigung-über-alle-Pfade-Semantik) von G ist definiert durch:

$$[\![\ ]\!]_{RVUP} : \mathcal{C} \to N \to \mathcal{C}_\Psi$$

$$\begin{aligned} [\![\ ]\!]_{RVUP} &=_{df} \lambda c \in \mathcal{C}.\ \lambda n \in N.\ \bigsqcup [\![n]\!]_{RAS}(c) \\ &= \lambda c \in \mathcal{C}.\ \lambda n \in N.\ \bigsqcup \{ [\![p]\!]_R(c) \mid p \in \mathbf{P}[n, \mathbf{e}] \} \end{aligned}$$

Die Vollständigkeit von $\widehat{\mathcal{C}_\Psi}$ garantiert für alle $c \in \mathcal{C}$ und $n \in N$ die Existenz von $\bigsqcup [\![n]\!]_{RVUP}(c)$ und damit:

Lemma 9.4.3.2 (*RVUP*-Wohldefiniertheit)

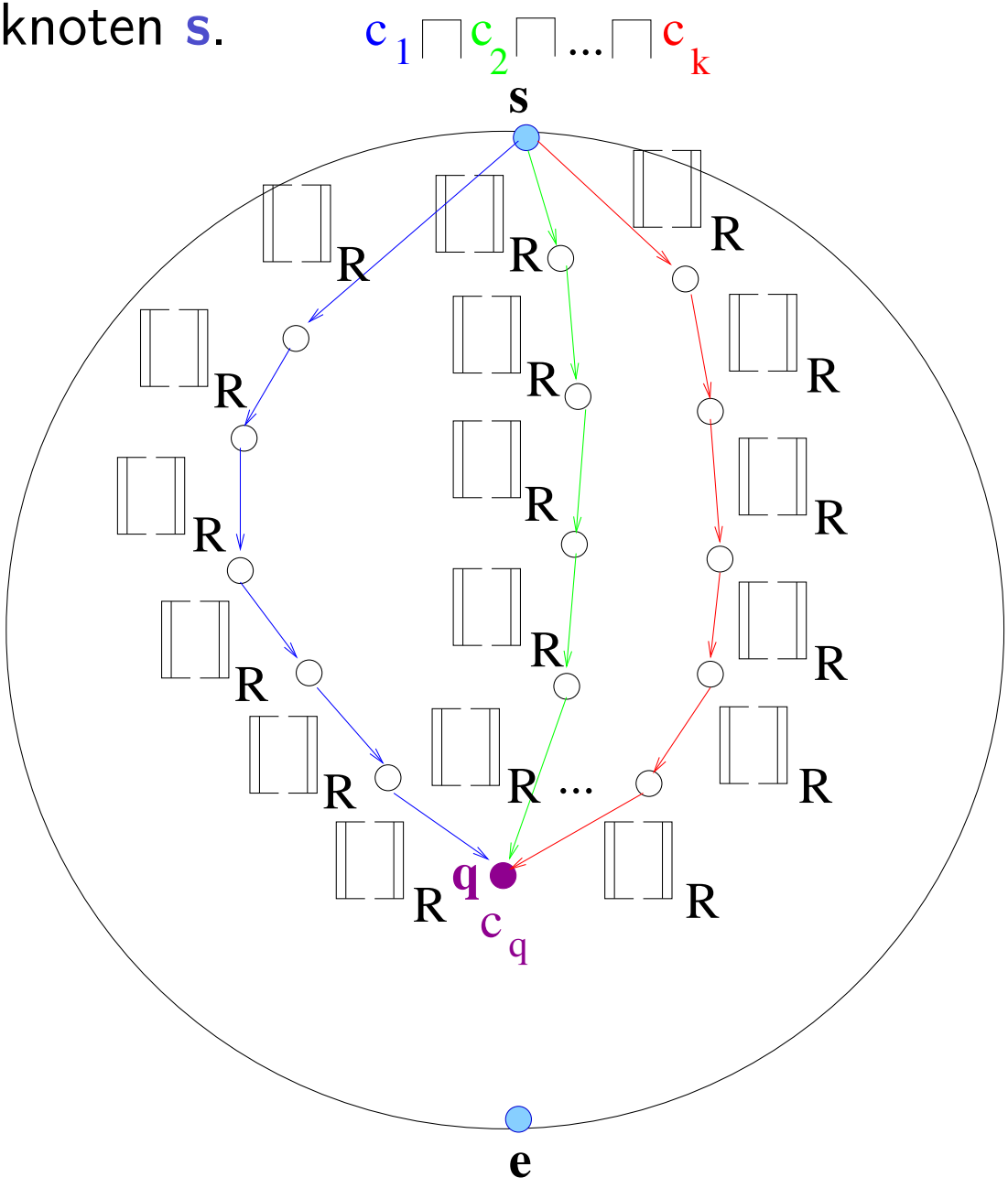
Die *RVUP*-Semantik $[\![\ ]\!]_{RVUP}$ von G ist wohldefiniert.

Kapitel 9.5.4

Reverse Schnitt-über-alle-Pfade-Semantik

Veranschaulichung der *RSUP*-Semantik

...am Startknoten **s**.



Die *RSUP*-Semantik

Definition 9.4.4.1 (*RSUP*-Semantik)

Die von $[\![\ ]\!]_R$ induzierte *RSUP*-Semantik (oder: deterministische reverse globale Schnitt-über-alle-Pfade-Semantik) von G ist definiert durch:

$$[\![\ ]\!]_{RSUP} : \mathcal{C} \to N \to \mathcal{C}_\Psi$$

$$\begin{aligned} [\![\ ]\!]_{RSUP} &=_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \bigsqcap [\![\ n\ ]\!]_{RAS}(c) \\ &= \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \bigsqcap \{\, [\![\ p\ ]\!]_R(c) \mid p \in \mathbf{P}[n, \mathbf{e}] \,\} \end{aligned}$$

Die Vollständigkeit von $\widehat{\mathcal{C}_\Psi}$ garantiert für alle $c \in \mathcal{C}$ und $n \in N$ die Existenz von $\bigsqcap [\![\ n\ ]\!]_{RSUP}(c)$ und damit:

Lemma 9.4.4.2 (*RSUP*-Wohldefiniertheit)

Die *RSUP*-Semantik $[\![\ ]\!]_{RSUP}$ von G ist wohldefiniert.

Kapitel 9.5

Zusammenfassung

Analyse - und reverse Analysesemantiken

Die reverse

1. Aufsammelsemantik $[\![\ ]\!]_{RAS}$
2. Vereinigung-über-alle-Pfade-Semantik $[\![\ ]\!]_{RVP}$
3. Schnitt-über-alle-Pfade-Semantik $[\![\ ]\!]_{RSUP}$

aus Kapitel 8 sind die reversen Gegenstücke der

1. Aufsammelsemantik $[\![\ ]\!]_{AS}$
2. Schnitt-über-alle-Pfade-Semantik $[\![\ ]\!]_{SUP}$
3. Vereinigung-über-alle-Pfade-Semantik $[\![\ ]\!]_{VUP}$

aus Kapitel 7.

Zur Definiertheit von $[\![\ ]\!]_{RAS}$, $[\![\ ]\!]_{RVUP}$, $[\![\ ]\!]_{RSUP}$

Wie ihr Gegenstück ist die reverse Aufsammelsemantik

- stets definiert.

Weder Verband noch reverse lokale abstrakte Semantik müssen dafür besonderen Anforderungen genügen (beachte aber, dass reverse lokale abstrakte Semantiken stets monoton sind, s. Lemma 9.3.5(1)).

Wie ihre Gegenstücke sind die reversen Vereinigung- und Schnitt-über-alle-Pfade-Semantiken

- definiert, wenn der Verband vollständig ist.

Die reverse lokale abstrakte Semantik muss dafür keinen besonderen Anforderungen genügen, auch wenn sie nach Lemma 9.3.5(1) stets monoton ist; auch die Vollständigkeit des Verbands ist stets gegeben, da sie bereits für die Wohldefiniertheit reverser lokaler abstrakter Semantiken vorausgesetzt ist.

Wie ihre Gegenstücke

...haben die reverse

- Aufsammelsemantik
- Vereinigung-über-alle-Pfade-Semantik
- Schnitt-über-alle-Pfade-Semantik

operationellen Charakter (orientiert an Programmpfaden).

...ist die reverse

- Aufsammelsemantik

nichtdeterministisch (i.S.v.: liefert die Menge möglicher Werte für jeden Programmpunkt).

...sind die reverse

- Vereinigung-über-alle-Pfade-Semantik
- Schnitt-über-alle-Pfade-Semantik

deterministisch (i.S.v.: liefern genau einen Wert für jeden Programmpunkt).

Zusammenfassend

Die

- Vereinigung/Schnitt-über-alle-Pfade (*RVUP*/*RSUP*) Semantiken zu einer reversen lokalen abstrakten DFA-Semantik $[\![\ ]\!]_R$

sind die zueinander dualen reversen Gegenstücke der

- Schnitt/Vereinigung-über-alle-Pfade (*SUP*/*VUP*) Semantiken zu einer lokalen abstrakten DFA-Semantik $[\![\ ]\!]$

wobei $[\![\ ]\!]_R$ von $[\![\ ]\!]$ induziert wird.

Ein DFA-Problem induziert also sein reverses Gegenstück, das RDFA-Problem, und liegt ihm in diesem Sinn zugrunde.

Kapitel 9.6

Literaturverzeichnis, Leseempfehlungen

Reading for Chapter 9

John Hughes, John Launchbury. *Reversing Abstract Interpretations*. In Proceedings of the 4th European Symposium on Programming (ESOP'92), Springer-V., LNCS 582, 269-286, 1992.

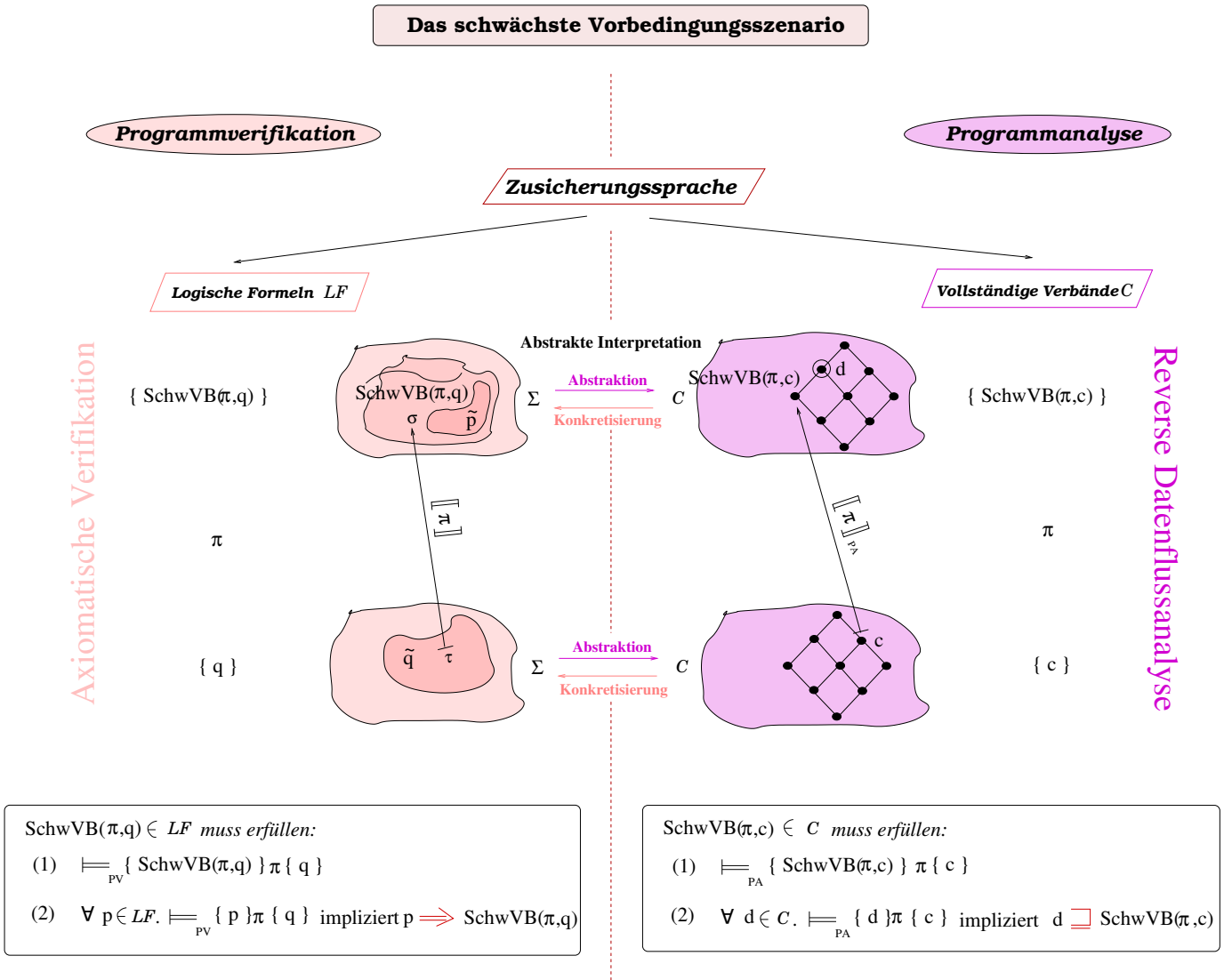
John Hughes, John Launchbury. *Reversing Abstract Interpretations*. Science of Computer Programming 22:307-326, 1994.

Kapitel 10

Reverse Datenflussanalyse

Reverse DFA: Berechnung schwächster

...Vorbedingungen: Analogon zu axiomatischer schwächster Vorbedingungsverfikation (s. Kapitel 4).



Reverse DFA, reverse DFA-Probleme

...vollständig beschrieben als Paar aus

- fehlschlagserweitertem vollständigen Verband $\mathcal{C}_\Psi$
- reverser lokaler abstrakter Semantik $[\![\ ]\!]_R$

die die (RDFA-) Semantik der reversen Datenflussanalyse festlegen, wobei $\mathcal{C}_\Psi$ und $[\![\ ]\!]_R$ von vollständigem Verband $\mathcal{C}$ und lokaler abstrakter Semantik $[\![\ ]\!]$ einer 'herkömmlichen' Datenflussanalyse induziert werden.

Anders als in Kapitel 9 möchten wir Zusicherungsanfragen nicht nur für den Endknoten eines Flussgraphen, sondern für jeden seiner Knoten stellen können.

Dafür ist die technische Vorbereitung aus Kapitel 10.1 erforderlich.

Kapitel 10.1

Reverse DFA-Spezifikationen

Sichtbarmachung von DFA-Anfrageknoten

...sei $G' = (N', E', \mathbf{s}', \mathbf{e}')$ ein Flussgraph und $q \in N'$ der Anfrageknoten (engl. query node), für den eine Datenflussanfrage (engl. query) gestellt werden soll.

Für die Formulierung reverser Datenflussanalyse ersetzen wir $G' = (N', E', \mathbf{s}', \mathbf{e}')$ durch einen Graphen $G = (N, E, \mathbf{s}, \mathbf{e})$, in dem für einen Anfrageknoten verschieden von $\mathbf{s}'$ und $\mathbf{e}'$ eine Kopie eingeführt wird.

Wir legen fest: Ist q

- $\mathbf{s}'$ oder $\mathbf{e}'$, so sind G und G' identisch.
- verschieden von $\mathbf{s}'$ und $\mathbf{e}'$, so entsteht G aus G' dadurch, dass N' um eine Kopie $\mathbf{q}$ von q erweitert wird, so dass $\mathbf{q}$ dieselben Vorgänger besitzt wie q, aber keine Nachfolger, d.h. $pred(\mathbf{q}) = pred(q)$ und $succ(\mathbf{q}) = \emptyset$.

Es gilt

...die Hinzunahme von $\mathbf{q}$ hat keinen Einfluss auf die

- *SUP*/*MaxFP*-Semantik
- *VUP*/*MinFP*-Semantik

irgendeines der ursprünglichen Knoten von G' (s. Korollar 10.1.2(1)).

Für $\mathbf{q}$ und q stimmen die

- *SUP*-Semantik
- *VUP*-Semantik

jeweils überein (s. Korollar 10.1.2(2)). Für die zugehörigen Fixpunktsemantiken (*MaxFP*, *MinFP*) gilt dies nicht.

Lemma 10.1.1 und Korollar 10.1.2 fassen das zusammen.

DFA-Zusammenhang von G' und G

Lemma 10.1.1

1. $\forall\, n \in N \backslash \{\mathbf{q}\}.\ \mathbf{P}_{G'}[\mathbf{s}, n] = \mathbf{P}_G[\mathbf{s}, n]$
2. $\forall\, q \in N' \backslash \{\mathbf{s}, \mathbf{e}\}.\ \mathbf{P}_{G'}[\mathbf{s}, q] = \mathbf{P}_G[\mathbf{s}, q] = \mathbf{P}_G[\mathbf{s}, \mathbf{q}]$

Korollar 10.1.2

1. $\forall\, n \in N \backslash \{\mathbf{q}\}.\ [\![\, n \,]\!]^X_{\mathcal{S}_{G'}} = [\![\, n \,]\!]^X_{\mathcal{S}_G}$
2. $\forall\, q \in N' \backslash \{\mathbf{s}, \mathbf{e}\}.\ [\![\, q \,]\!]^Y_{\mathcal{S}_{G'}} = [\![\, q \,]\!]^Y_{\mathcal{S}_G} = [\![\, \mathbf{q} \,]\!]^Y_{\mathcal{S}_G}$

mit $X \in \{SUP, MaxFP, VUP, MinFP\}$, $Y \in \{SUP, VUP\}$.

...wobei $\mathbf{q} \in N$ die zum Anfrageknoten $q \in N'$ gehörige Kopie bezeichnet mit $pred_G(\mathbf{q}) = pred_{G'}(q)$ und $succ_G(\mathbf{q}) = \emptyset$.

RDFA-Semantik, Reverse Datenflussanalyse

...mit den vorherigen Festlegungen und Beobachtungen können wir jetzt definieren:

Definition 10.1.3 (RDFA-Semantik, Reverse DFA)

Ein Paar $(\widehat{\mathcal{C}_\Psi}, [\![\ ]\!]_R)$ mit:

- $\widehat{\mathcal{C}_\Psi} =_{df} (\mathcal{C}_\Psi, \sqsubseteq_\Psi, \sqcap_\Psi, \sqcup_\Psi, \bot, \Psi)$ DFA-Verbandserweiterung zu $\widehat{\mathcal{C}}$ (gemäß Definition 9.2.1).
- $[\![\ ]\!]_R : E \to (\mathcal{C}_\Psi \to \mathcal{C}_\Psi)$ induzierte reverse lokale DFA-Semantik (gemäß Definition 9.3.1).

definiert die reverse (lokale) (DFA-) Semantik einer reversen Datenflussanalyse für G.

RDFA-Spezifikation, RDFA-Problem

Definition 10.1.4 (RDFA-Spez., RDFA-Problem)

Ein Tripel $(\widehat{\mathcal{C}}_\Psi, [\![\ ]\!]_R, c_q)$ mit:

- $(\widehat{\mathcal{C}}_\Psi, [\![\ ]\!]_R)$ reverse (lokale) (DFA-) Semantik für G
- $c_q \in \mathcal{C}$ Sollzusicherung (oder DFA-Anfrage) für Knoten q.

spezifiziert ein konkretes RDFA-Problem, die reverse Problem-instanz $\mathcal{S}_G^R =_{df} (\widehat{\mathcal{C}}_\Psi, [\![\ ]\!]_R, c_q)$.

Kapitel 10.2

RVUP- und *RSUP*-Semantik als zueinander duale spezifizierende RDFA-Problemlösungen

Spezifizierende RDFA-Problemlösungen

...nach dem Vorbild der *SUP*- und *VUP*-Semantik für DFA-Probleme definieren wir:

Definition 10.2.1 (Spezifizier. RDFA-Problemlsg.)

Die *RVUP*- und *RSUP*-Semantik eines Flussgraphen definieren zwei zueinander duale spezifizierende Lösungen eines reversen DFA-Problems, seine sog.:

1. *RVUP*-Lösung
2. *RSUP*-Lösung

Kapitel 10.3

Korrektheit, Vollständigkeit, Akkuratheit von RDFA-Algorithmen

Korrektheit, Vollständigkeit, Akkuratheit

...von RDFA-Algorithmen.

Definition 10.3.1 (Korrekt, vollständig, akkurat)

Ein RDFA-Algorithmus A heißt

1. *RVUP*-korrekt (*RSUP*-korrekt), wenn A für alle RDFA-Probleme $\mathcal{S}_G^R$ mit einer oberen (unteren) Approximation der *RVUP*- (*RSUP*-) Semantik von $\mathcal{S}_G^R$ terminiert.
2. *RVUP*-vollständig (*RSUP*-vollständig), wenn A für alle RDFA-Probleme $\mathcal{S}_G^R$ mit einer unteren (oberen) Approximation der *RVUP*- (*RSUP*-) Semantik von $\mathcal{S}_G^R$ terminiert.
3. *RVUP*-akkurat (*RSUP*-akkurat), wenn A für alle RDFA-Probleme $\mathcal{S}_G^R$ exakt mit der *RVUP*- (*RSUP*-) Semantik von $\mathcal{S}_G^R$ terminiert.

Es gilt

Lemma 10.3.2 (Akkuratheit)

Ein RDFA-Algorithmus A ist

1. *RVUP*-akkurat gdw A ist *RVUP*-korrekt und *RVUP*-vollständig.
2. *RSUP*-akkurat gdw A ist *RSUP*-korrekt und *RSUP*-vollständig.

Statt von akkurat sprechen wir auch von optimal.

Definition 10.3.3 (Optimalität)

Ein RDFA-Algorithmus A heißt *RVUP*-optimal (*RSUP*-optimal) gdw A ist *RVUP*-akkurat (*RSUP*-akkurat).

Zur Existenz

...korrekter, vollständiger, optimaler RDFA-Algorithmen.

Lemma 10.3.4 (Existenz)

Ein RDFA-Algorithmus A, der für den Anfrageknoten **s** stets die Anfangsfrage c_q und für alle anderen Programmpunkte stets die Information

1. $\Psi(\bot)$ liefert, ist *RVUP*-korrekt (*RVUP*-vollständig).
2. $\Psi(\bot)$ liefert, ist *RSUP*-korrekt (*RSUP*-vollständig).

Offenbar sind die RDFA-Algorithmen aus Lemma 10.3.4 wie ihre Gegenstücke aus Lemma 8.3.4 nutzlos.

Wie in Kapitel 8 ist es jetzt wieder Aufgabe, nützliche RDFA-Algorithmen zu finden.

Kapitel 10.4

Denotationelle reverse globale DFA-Semantiken

Kapitel 10.4.1

Reverse minimale Fixpunktsemantik

Die reverse min. Fixpunktsemantik (*RMinFP*)

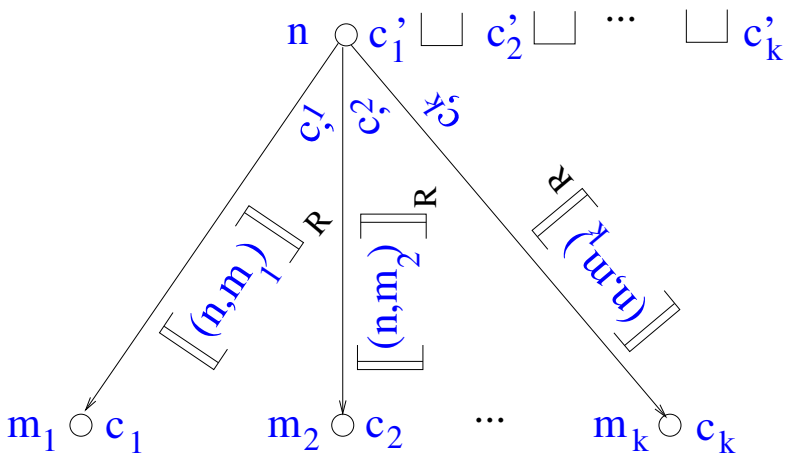
Sei $\mathcal{S}_G^R = (\widehat{\mathcal{C}_\Psi}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Gleichungssystem 10.4.1.1 (*RMinFP*-Ansatz)

$$reqInf(n) = \begin{cases} c_q & \text{falls } n = \mathbf{q} \\ \bigsqcup \{ [\![(n,m)]\!]_R(reqInf(m)) \mid m \in succ(n) \} & \text{sonst} \end{cases}$$

(*reqInf* steht für *'required information'*)

Veranschaulichung des *RMinFP*-Ansatzes ($n \neq \mathbf{e}$):



Die *RMinFP*-Semantik

Die Monotonie der reversen Semantikfunktionen garantiert die Existenz der kleinsten Lösung von Gleichungssystem 10.4.1.1, die wir mit

- $\mu\text{-}reqInf_{c_q} : N \to \mathcal{C}_\Psi$

bezeichnen.

Definition 10.4.1.2 (*RMinFP*-Semantik)

Die von einer reversen lokalen abstrakten Semantik $[\![\ ]\!]_R$ induzierte *RMinFP*-Semantik von G ist definiert durch:

$$[\![\ ]\!]_{RMinFP} : \mathcal{C} \to N \to \mathcal{C}_\Psi$$

$$[\![\ ]\!]_{RMinFP} =_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \mu\text{-}reqInf_c(n)$$

Lemma 10.4.1.3 (*RMinFP*-Wohldefiniertheit)

Die *RMinFP*-Semantik $[\![\ ]\!]_{RMinFP}$ von G ist wohldefiniert.

Kapitel 10.4.2

Reverse maximale Fixpunktsemantik

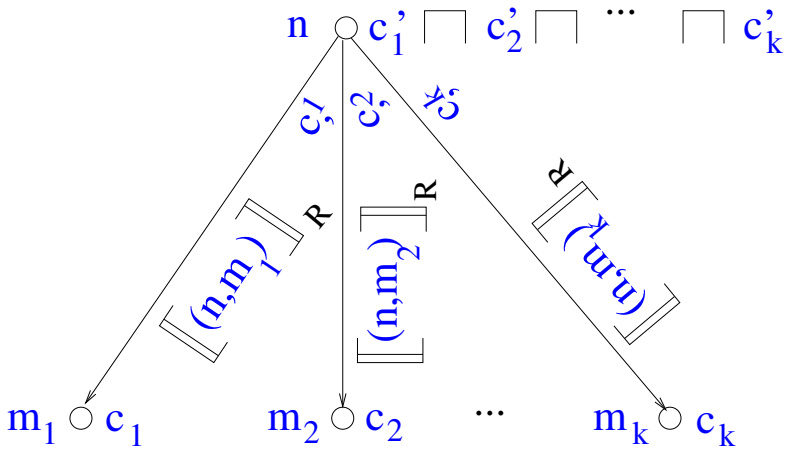
Die reverse max. Fixpunktsemantik (*RMaxFP*)

Sei $\mathcal{S}_G^R = (\widehat{\mathcal{C}_\Psi}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Gleichungssystem 10.4.2.1 (*RMaxFP*-Ansatz)

$$reqInf(n) = \begin{cases} c_q & \text{falls } n = \mathbf{q} \\ \bigsqcap \{ [\![(n,m)]\!]_R(reqInf(m)) \mid m \in succ(n) \} & \text{sonst} \end{cases}$$

Veranschaulichung des *RMaxFP*-Ansatzes ($n \neq \mathbf{e}$):



Die *RMaxFP*-Semantik

Die Monotonie der reversen Semantikfunktionen garantiert die Existenz der größten Lösung von Gleichungssystem 10.4.2.1, die wir mit

– $\nu\text{-}reqInf_{c_q} : N \to \mathcal{C}_\Psi$

bezeichnen.

Definition 10.4.2.2 (*RMaxFP*-Semantik)

Die von einer reversen lokalen abstrakten Semantik $[\![\ ]\!]_R$ induzierten *RMaxFP*-Semantik von G ist definiert durch:

$$[\![\ ]\!]_{RMaxFP} : \mathcal{C} \to N \to \mathcal{C}_\Psi$$

$$[\![\ ]\!]_{RMaxFP} =_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \nu\text{-}reqInf_c(n)$$

Lemma 10.4.2. 3 (*RMaxFP*-Wohldefiniertheit)

Die *RMaxFP*-Semantik $[\![\ ]\!]_{RMaxFP}$ von G ist wohldefiniert.

Kapitel 10.5

Entscheidbarkeit der *RMinFP*- und *RMaxFP*-Semantik

Die *RMinFP*- und *RMaxFP*-Semantik

...eines Flussgraphen sind aufgrund von *RMinFP*-Gleichungssystem 10.4.1.1 und *RMaxFP*-Gleichungssystem 10.4.2.1 praktisch relevant, da sie in generischer Weise ein

- iteratives Berechnungsverfahren (Algorithmus 10.5.1.1)

induzieren, das ihre kleinste und größte Lösung approximativ und unter gewissen zusätzlichen Voraussetzungen exakt zu berechnen erlaubt, mithin die *RMinFP*- und *RMaxFP*-Semantiken selbst.

Kapitel 10.5.1

Reverser generischer Fixpunktalgorithmus

Reverser gen. Fixpunktalgorithmus 10.5.1.1 (1)

Eingabe: Reverse DFA-Spezifikation $\mathcal{S}_G^R = (\widehat{\mathcal{C}_\Psi}, [\![\ ]\!]_R, c_q)$.

Ausgabe: Bei Terminierung des Algorithmus (s. Terminierungstheorem 10.5.2.1) enthält die Variable *reqInf*$[n]$ die *RMinFP*-Lösung von $\mathcal{S}_G^R$ am Knoten n.

Zusätzlich (s. Reverses Sicherheitstheorem 10.6.1 und Reverses Koinzidenztheorem 10.6.2) gilt: Wenn $[\![\ ]\!]_R$

- additiv: *reqInf*$[\mathbf{s}]$ enthält die
- monoton: *reqInf*$[\mathbf{s}]$ enthält eine obere Approximation der

der *RVUP*-Lösung von $\mathcal{S}_G^R$ am Knoten n.

Bemerkung: Die Variable *workset* steuert die iterative Berechnung. Ihre Elemente sind Flussgraphknoten, deren Annotation jüngst aktualisiert worden ist, was ihrerseits Aktualisierungen und damit verbandsmäßig größere Annotationen an ihren Vorgängerknoten zur Folge haben kann.

Reverser gen. Fixpunktalgorithmus 10.5.1.1 (2)

(Prolog: Initialisierung von *reqInf* und *workset*)
FORALL $n \in N \backslash \{\mathbf{q}\}$ DO $reqInf[n] := \bot$ OD;
$reqInf[\mathbf{q}] := c_q$;
$workset := \{N\}$;
(Hauptschleife: Iterative Fixpunktberechnung)
WHILE $workset \neq \emptyset$ DO
    CHOOSE $m \in workset$;
        $workset := workset \backslash \{m\}$;
        (Aktualisierung d. Vorgängerumgebung von Knoten m)
        FORALL $n \in pred(m)$ DO
            $join := [\![(n, m)]\!]_R(reqInf[m]) \sqcup_\Psi reqInf[n]$;
            IF $reqInf[n] \sqsubset_\Psi join$
                THEN
                    $reqInf[n] := join$;
                    $workset := workset \cup \{n\}$ FI
        OD ESOOHC OD.

Kapitel 10.5.2

Effektivität, Terminierung

Effektivität, Terminierung

...mit Lemma 9.3.3(1) ('die von einer lokalen DFA-Semantik induzierte reverse Semantik ist monoton') erhalten wir:

Theorem 10.5.2.1 (Effektivität, Terminierung)

Fixpunktalgorithmus 10.5.1.1 terminiert mit der

1. *RMinFP*-Semantik von $\mathcal{S}_G^R$, wenn $\widehat{\mathcal{C}_\Psi}$ die aufsteigende Kettenbedingung erfüllt.
2. *RMaxFP*-Semantik von $\mathcal{S}_G^R$, wenn $\widehat{\mathcal{C}_\Psi^{gst}}$ die aufsteigende Kettenbedingung erfüllt, wobei

$$\widehat{\mathcal{C}_\Psi^{gst}} =_{df} (\mathcal{C}_\Psi, \sqsupseteq_\Psi, \sqcup_\Psi, \sqcap_\Psi, \Psi, \bot)$$

den gestürzten auf den Kopf gestellten Verband $\widehat{\mathcal{C}_\Psi} = (\mathcal{C}_\Psi, \sqsubseteq_\Psi, \sqcap_\Psi, \sqcup_\Psi, \bot, \Psi)$ bezeichnet.

Effektivitäts-, Terminierungskorollar

...mit Lemma 9.3.2(2) ('der fehlschlagserweiterte DFA-Verband $\widehat{\mathcal{C}_\Psi}$ von $\mathcal{S}_G^R$ erfüllt die aufsteigende Kettenbedingung gdw der zugrundeliegende DFA-Verband $\widehat{\mathcal{C}}$ von $\mathcal{S}_G$ die aufsteigende Kettenbedingung erfüllt') können wir die verbleibende Voraussetzung von Theorem 10.5.2.1 auf die entsprechende Eigenschaft des Ursprungsproblems zurückführen:

Korollar 10.5.2.2 (Effektivität, Terminierung)

Fixpunktalgorithmus 10.5.1.1 terminiert mit der *RMinFP*-Semantik (*RMaxFP*-Semantik) von $\mathcal{S}_G^R$, wenn $\widehat{\mathcal{C}}$ ($\widehat{\mathcal{C}^{gst}}$) die aufsteigende Kettenbedingung erfüllt, wobei $\widehat{\mathcal{C}^{gst}}$ den gestürzten auf den Kopf gestellten Verband $\widehat{\mathcal{C}}$ bezeichnet.

Kapitel 10.6

RMinFP- und *RMaxFP*-Semantik als zueinander duale berechenbare RDFA-Problemlösungen

Berechenbare RDFA-Problemlösungen

...zusammen legen der reverse generische Fixpunktalgorithmus 10.5.1.1 und das Terminierungstheorem 10.5.2.1 folgende Festlegung nahe:

Definition 10.6.1 (Berechenbare Lsg. eines RDFA-P.)

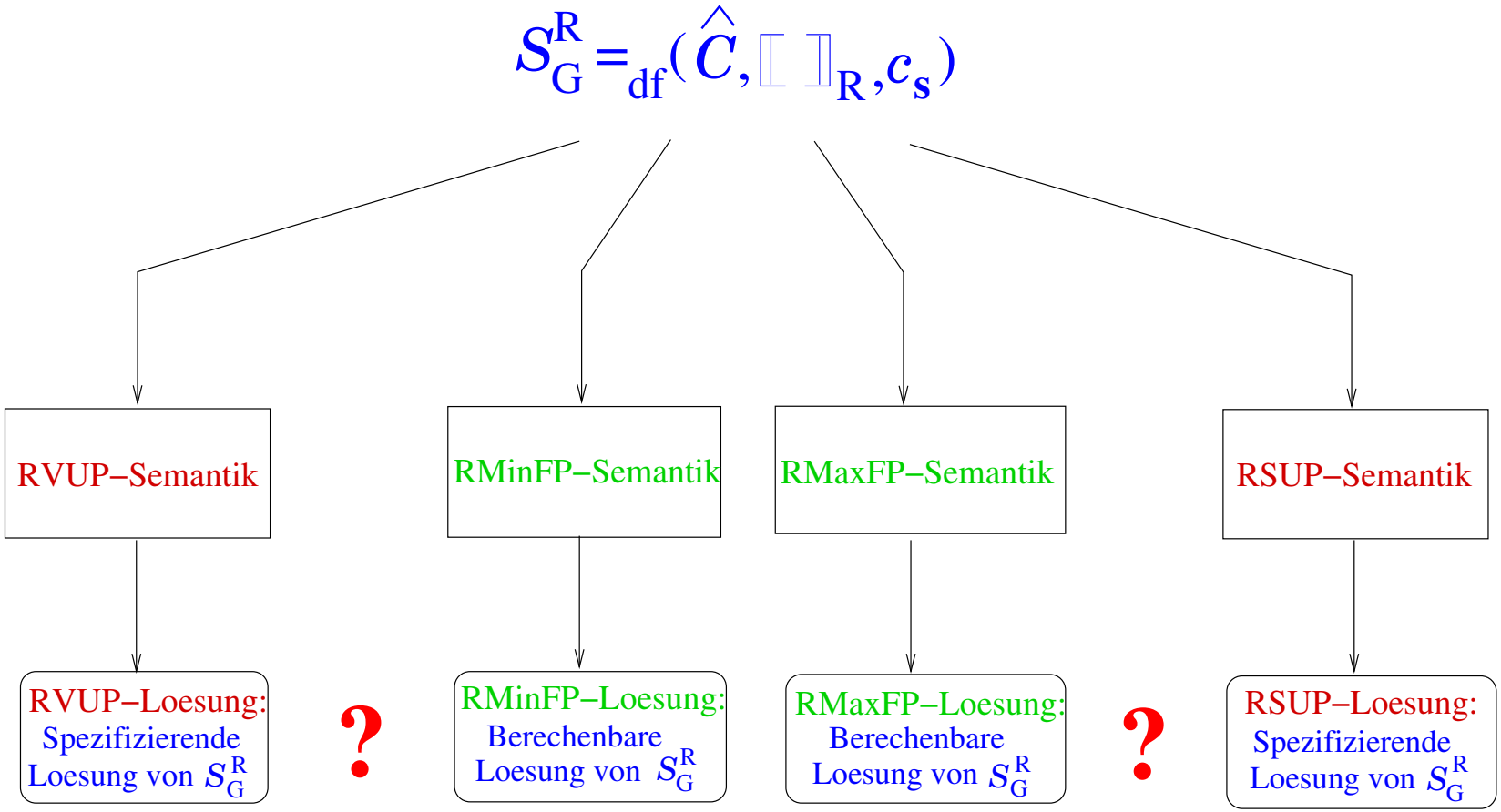
Die *RMinFP*- und *RMaxFP*-Semantik eines Flussgraphen definieren zwei zueinander duale berechenbare Lösungen eines RDFA-Problems, seine sog.:

1. *RMinFP*-Lösung
2. *RMaxFP*-Lösung

Damit stellt sich auch hier die Frage nach dem Verhältnis, in dem die spezifizierenden und berechenbaren Lösungen eines RDFA-Problems zueinander stehen...

RVUP/*RMinFP*- u. *RSUP*/*RMaxFP*-Semantik

...die Frage nach ihrer Beziehung:



Kapitel 10.7

Korrektheit, Vollständigkeit: Reverse Sicherheit, reverse Koinzidenz

Rev. Sicherh.: Korrekth. v. *RMinFP* / *RMaxFP*-S.

Sei $\mathcal{S}_G^R = (\widehat{\mathcal{C}_\Psi}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

...mit Lemma 9.3.3(1) ('die von einer lokalen DFA-Semantik induzierten reversen Semantikfkt. sind monoton') erhalten wir:

Theorem 10.7.1 (Reverse Sicherheit: Korrektheit)

1. Die *RMinFP*-Semantik von $\mathcal{S}_G^R$ ist eine sichere (d.h. obere) Approximation der *RVUP*-Semantik von $\mathcal{S}_G^R$:

$$[\![\ ]\!]_{RMinFP} \sqsupseteq [\![\ ]\!]_{RVUP}$$

2. Die *RMaxFP*-Semantik von $\mathcal{S}_G^R$ ist eine sichere (d.h. untere) Approximation der *RSUP*-Semantik von $\mathcal{S}_G^R$:

$$[\![\ ]\!]_{RMaxFP} \sqsubseteq [\![\ ]\!]_{RSUP}$$

Rev. Koinzidenz: Akkurat. *RMinFP* / *RMaxFP*-S.

Sei $\mathcal{S}_G^R = (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Theorem 10.7.2 (Reverse Koinzidenz: Akkuratheit)

Ist die reverse lokale DFA-Semantik $[\![\ ]\!]_R$ additiv bzw. distributiv, so stimmen

1. *RMinFP*- und *RVUP*-Semantik von $\mathcal{S}_G^R$ überein:

$$[\![\ ]\!]_{RMinFP} = [\![\ ]\!]_{RVUP}$$

2. *RMaxFP*- und *RSUP*-Semantik von $\mathcal{S}_G^R$ überein:

$$[\![\ ]\!]_{RMaxFP} = [\![\ ]\!]_{RSUP}$$

Reverses Koinzidenzkorollar

...mit Lemma 9.2.4 ('$[\![\ ]\!]_R$ ist distributiv/additiv, wenn $[\![\ ]\!]$ distributiv ist') können wir die verbleibenden Voraussetzungen von Theorem 10.7.2 auf zwei Eigenschaften des induzierenden Ursprungsproblems zurückführen:

Korollar 10.7.3 (Reverse Koinzidenz)

Ist die lokale DFA-Semantik $[\![\ ]\!]$ der induzierenden DFA-Spezifikation $\mathcal{S}_G$ distributiv, so stimmen

1. *RMinFP*- und *RVUP*-Semantik von $\mathcal{S}_G^R$ überein:

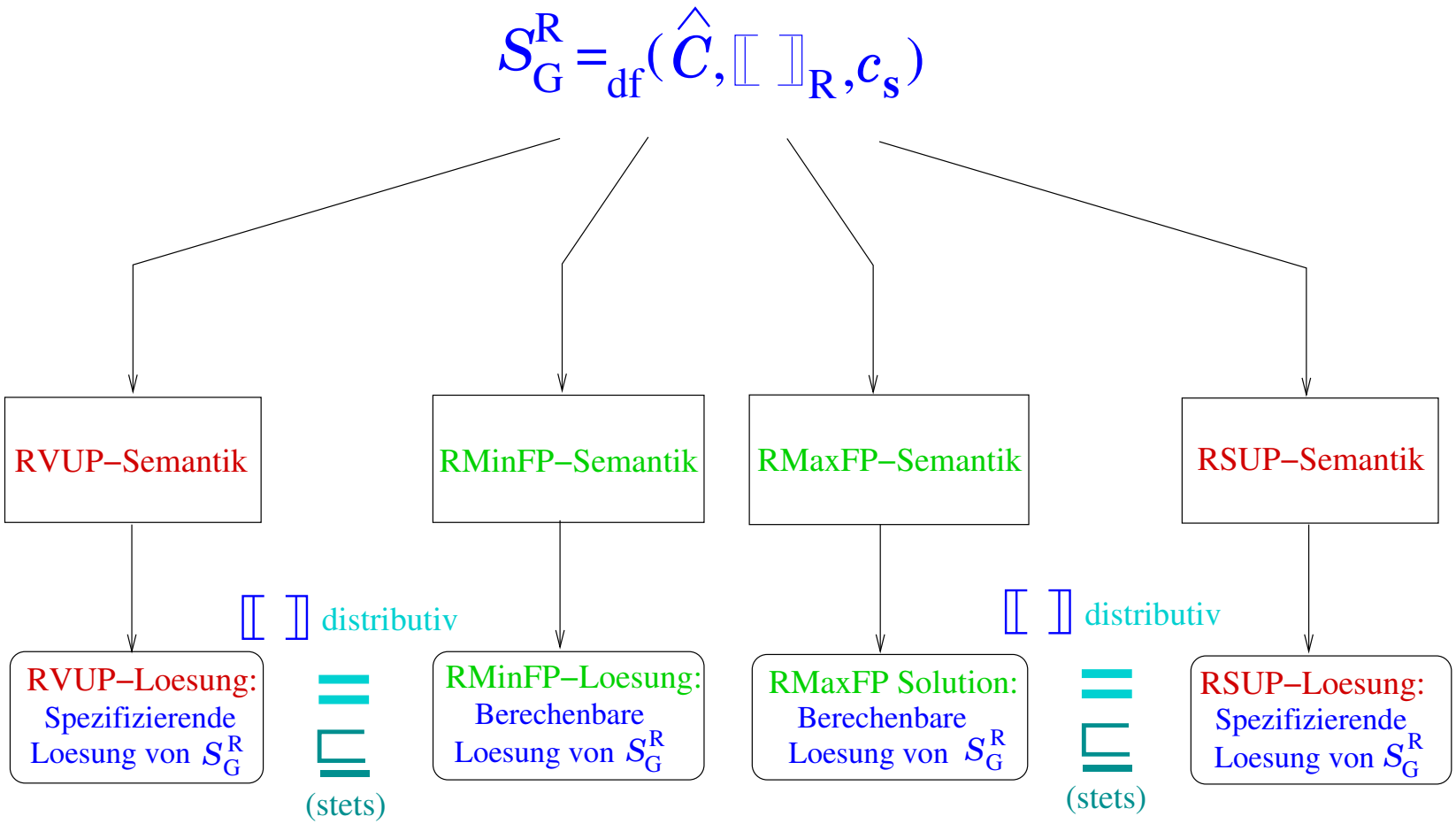
$$[\![\ ]\!]_{RMinFP} = [\![\ ]\!]_{RVUP}$$

2. *RMaxFP*- und *RSUP*-Semantik von $\mathcal{S}_G^R$ überein:

$$[\![\ n\ ]\!]_{RMaxFP} = [\![\ n\ ]\!]_{RSUP}$$

RVUP/*RMinFP*- u. *RSUP*/*RMaxFP*-Semantik

...und die Antwort auf die Frage nach ihrer Beziehung:



Korrektheit von Fixpunktalgorithmus 10.5.1.1

...mit Lemma 9.3.5 und Theorem 10.7.1 erhalten wir:

Korollar 10.7.4 (*RVUP*/*RSUP*-Korrektheit)

Fixpunktalgorithmus 10.5.1.1 ist

- *RVUP*- (*RSUP*-) korrekt

für $\mathcal{S}_G^R$, d.h. er terminiert mit einer oberen (unteren) Approximation der *RVUP*- (*RSUP*-) Semantik von $\mathcal{S}_G^R$, wenn der Verband $\widehat{\mathcal{C}}$ der induzierenden DFA-Spezifikation $\mathcal{S}_G$ (und damit auch der Verband $\widehat{\mathcal{C}_\Psi}$ von $\mathcal{S}_G^R$) die aufsteigende (absteigende) Kettenbedingung erfüllt.

Akkuratheit von Fixpunktalgorithmus 10.5.1.1

Korollar 10.7.5 (*RVUP*/*RSUP*-Straffheit)

Fixpunktalgorithmus 10.5.1.1 ist

- *RVUP*- (*RSUP*-) akkurat

für $\mathcal{S}_G^R$, d.h. er terminiert mit der *RVUP*- (*RSUP*-) Semantik von $\mathcal{S}_G^R$, wenn für die induzierende DFA-Spezifikation $\mathcal{S}_G$ gilt:

1. $[\![\ ]\!]$ ist distributiv.
2. $\widehat{\mathcal{C}}$ erfüllt die aufsteigende (absteigende) Kettenbedingung.

Beachte: Distributivität von $[\![\ ]\!]$ (und deshalb auch von $[\![\ ]\!]_\Psi$) impliziert Additivität von $[\![\ ]\!]_R$ (siehe Lemma 9.3.3(1) und 9.3.3(2)). Somit sind Distributivität und Additivität von $[\![\ ]\!]_R$ implizit in Korollar 10.7.5 gefordert, werden aber auf die Distributivitätseigenschaft des Ursprungsproblems zurückgeführt.

Kapitel 10.8

Anwendungen: Zwei kanonische Beispiele distributiver und additiver RDFA-Probleme

Kapitel 10.8.1

Total verfügbare Ausdrücke, ein additives RDFA-Problem

Totale Verfügbarkeit: Reverse DFA-Spezifik.

...für einen einzelnen Term t.

Reverse DFA-Spezifikation zu totaler Termverfügbarkeit:

1. RDFA-Verband:
   $(\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top) =_{df} (\mathbb{B}_\Psi, \leq_\Psi, \wedge_\Psi, \vee_\Psi, \textbf{falsch}, \Psi)$
   mit $\textbf{falsch} \leq_\Psi \textbf{wahr} \leq_\Psi \Psi$.
2. RDFA-Semantik:
   $[\![\ ]\!]_R^{av_t} : E \to (\ \mathbb{B}_\Psi \to \mathbb{B}_\Psi\ )$ definiert durch
   $\forall\, e \in E.\ [\![\, e\,]\!]_R^{av_t} =_{df}\ \lambda\, b.\ \bigsqcap\{b' \in \mathbb{B}_\Psi \mid [\![\, e\,]\!]_\Psi^{av_t}(b') \geq_\Psi b\}$
   wobei $[\![\, e\,]\!]_\Psi^{av_t} : E \to (\ \mathbb{B}_\Psi \to \mathbb{B}_\Psi\ )$ die Ausdehnung der lokalen DFA-Semantik $[\![\, e\,]\!]_{av}^{t} : E \to (\ \mathbb{B} \to \mathbb{B}\ )$ von Variante 1 aus Kapitel 8.13.1.1 von $\mathbb{B}$ auf $\mathbb{B}_\Psi$ gemäß Definition 9.3.1 ist (der Index av erinnert an *available*).
3. Gesucht: *RMinFP*-Semantik ($=$ *RVUP*-Semantik) als schwächste Zusicherung für totale Verfügbarkeit von t.

Charakterisierung der rev. DFA-Semantikfkt.

Mithilfe der Funktionen $Cst^R_{\mathbf{wahr}}$, $Cst^R_{\mathbf{falsch}}$ und $Id^R_{\mathbb{B}_\Psi}$ über

$$\mathbb{B}_\Psi =_{df} \{\mathbf{falsch}, \mathbf{wahr}, \Psi\}$$

die definiert sind durch:

$$Cst^R_{\mathbf{wahr}} =_{df} \lambda\, b.\ \begin{cases} \mathbf{falsch} & \text{falls } b \in \{\mathbf{falsch}, \mathbf{wahr}\} \\ \Psi & \text{falls } b = \Psi \end{cases}$$

$$Cst^R_{\mathbf{falsch}} =_{df} \lambda\, b.\ \begin{cases} \mathbf{falsch} & \text{falls } b = \mathbf{falsch} \\ \Psi & \text{falls } b \in \{\mathbf{wahr}, \Psi\} \end{cases}$$

$$Id^R_{\mathbb{B}_\Psi} =_{df} \lambda\, b.\ b$$

...können wir die induzierten reversen DFA-Semantikfunktionen $[\![\, e \,]\!]^{av_t}_R$, $e \in E$, direkt charakterisieren (s. Lemma 10.8.1.1).

Charakterisierung

...der induzierten reversen Semantikfunktionen.

Lemma 10.8.1.1 (Charakterisierung)

$$\forall\, e \in E.\ [\![\, e \,]\!]_R^{av_t} = \begin{cases} Cst^R_{\mathbf{wahr}} & \text{falls } [\![\, e \,]\!]^t_{av} = Cst_{\mathbf{wahr}} \\ Id^R_{\mathbb{B}_\psi} & \text{falls } [\![\, e \,]\!]^t_{av} = Id_{\mathbb{B}} \\ Cst^R_{\mathbf{falsch}} & \text{falls } [\![\, e \,]\!]^t_{av} = Cst_{\mathbf{falsch}} \end{cases}$$

wobei $[\![\, e \,]\!]^t_{av} : E \to (\, \mathbb{B} \to \mathbb{B}\,)$ die lokale DFA-Semantik von Variante 1 aus Kapitel 8.13.1.1 ist.

Kapitel 10.8.2

Partiell verfügbare Ausdrücke, ein distributives RDFA-Problem

Partielle Verfügbarkeit: Reverse DFA-Spezifik.

...für einen einzelnen Term t.

Reverse DFA-Spezifikation zu partieller Termverfügbarkeit:

1. RDFA-Verband:
   $(\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top) =_{df} (\mathbb{B}_\Psi, \leq_\Psi, \wedge_\Psi, \vee_\Psi, \mathbf{falsch}, \Psi)$
   mit $\mathbf{falsch} \leq_\Psi \mathbf{wahr} \leq_\Psi \Psi$.
2. RDFA-Semantik:
   $[\![\ ]\!]_R^{av_t} : E \to (\ \mathbb{B}_\Psi \to \mathbb{B}_\Psi\ )$ definiert durch
   $\forall\, e \in E.\ [\![\, e\,]\!]_R^{av_t} =_{df}\ \lambda\, b.\ \bigsqcap\{b' \in \mathbb{B}_\Psi \mid [\![\, e\,]\!]_\Psi^{av_t}(b') \geq_\Psi b\}$
   wobei $[\![\, e\,]\!]_\Psi^{av_t} : E \to (\ \mathbb{B}_\Psi \to \mathbb{B}_\Psi\ )$ die Ausdehnung der lokalen DFA-Semantik $[\![\, e\,]\!]_{av}^{t} : E \to (\ \mathbb{B} \to \mathbb{B}\ )$ von Variante 1 aus Kapitel 8.13.1.1 von $\mathbb{B}$ auf $\mathbb{B}_\Psi$ gemäß Definition 9.3.1 ist (der Index *av* erinnert an *available*).
3. Gesucht: *RMaxFP*-Semantik ($=$ *RSUP*-Semantik) als schwächste Zusicherung für partielle Verfügbarkeit von t

Beachte

- Die RDFA-Spezifikationen zu partieller und totaler Verfügbarkeit stimmen überein.
- Der Unterschied liegt einzig in der gesuchten Semantik:
  - *RMaxFP*-Semantik ($=$ *RVUP*-Semantik) im Fall partieller Verfügbarkeit.
  - *RMinFP*-Semantik ($=$ *RSUP*-Semantik) im Fall totaler Verfügbarkeit.
- Um mithilfe von Fixpunktalgorithmus 10.5.5.1 die *RMaxFP*-Semantik berechnen zu können, wird der Algorithmus mit dem gestürzten auf den Kopf gestellten Verband aufgerufen.
- Charakterisierungslemma 10.8.1.1 gilt für die identen RDFA-Spezifikationen aus Kapitel 10.8.1 und 10.8.2 in gleicher Weise.

Kapitel 10.9

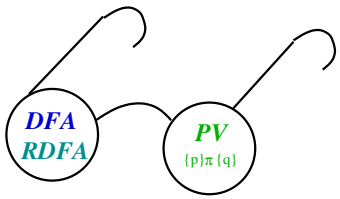
Zusammenhang von DFA und RDFA, darauf aufbauende Anwendungen

Kapitel 10.9.1

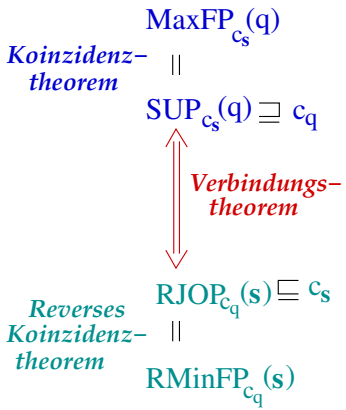
DFA/RDFA-Zusammenhang: Verbindungstheoreme

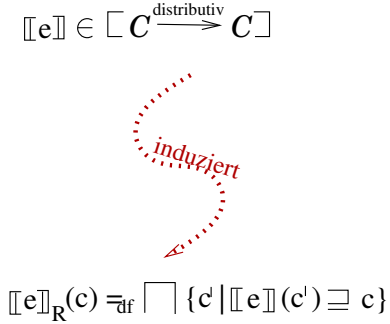
Veranschaulichung des DFA/RDFA-Zshgs.

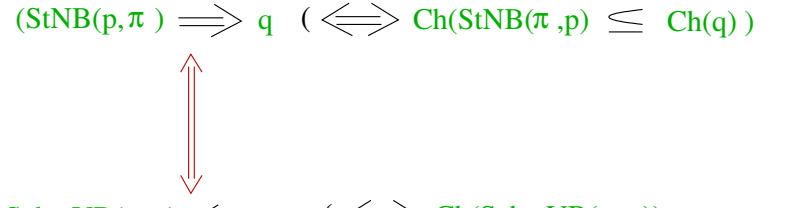
...im distributiven Fall und Analogie zu axiomatischer Programmverifikation:



Datenflussanalyse | **Programmverifikation**







Reverse Datenflussanalyse

"⊢ <c$_s$>P[s,q]<c$_q$>" ⟺ SUP$_{c_s}$(q) ⊒ c$_q$ ∧ RVUP$_{c_q}$(s) ⊑ c$_s$ ⊢ {p} π {q} ⟺ (StNB(p,π) ⟹ q ∧ SchwVB(π ,q) ⟸ p

DFA/RDFA-Zusammenhang distributiver Fall

Theorem 10.9.1.1 (Distributives Verbindungstheor.)

Ist $\mathcal{S}_G$ eine DFA-Spezifikation mit distributiver lokaler DFA-Semantik $[\![\ ]\!]$, so gilt für alle Anfangszusicherungen $c_{\mathbf{s}} \in \mathcal{C}$ an $\mathbf{s}$ und Sollzusicherungen $c_q \in \mathcal{C}$ an q:

$$([\![\, q \,]\!]_{MaxFP}(c_{\mathbf{s}}) \;=) \quad [\![\, q \,]\!]_{SUP}(c_{\mathbf{s}}) \;\sqsupseteq\; c_q$$

$$\Longleftrightarrow$$

$$([\![\, \mathbf{s} \,]\!]_{RMinFP}(c_q) \;=) \quad [\![\, \mathbf{s} \,]\!]_{RVUP}(c_q) \;\sqsubseteq\; c_{\mathbf{s}}$$

Informell: Gilt im *SUP*-Sinn bei Anfangszusicherung $c_{\mathbf{s}}$ an $\mathbf{s}$ mindestens c_q an q (obere Zeile), ist für Sollzusicherung c_q an q Anfangszusicherung $c_{\mathbf{s}}$ oder weniger an $\mathbf{s}$ ausreichend (untere Zeile) und umgekehrt (vgl. hierzu auch den Adjunktionsbegriff aus Def. 16.2.1.9).

DFA/RDFA-Zusammenhang additiver Fall

Theorem 10.9.1.2 (Additives Verbindungstheorem)

Ist $\mathcal{S}_G$ eine DFA-Spezifikation mit additiver lokaler DFA-Semantik $[\![\ ]\!]$, so gilt für alle Anfangszusicherungen $c_{\mathbf{s}} \in \mathcal{C}$ an $\mathbf{s}$ und Sollzusicherungen $c_q \in \mathcal{C}$ an q:

$$\begin{array}{lll} ([\![\, q \,]\!]_{MinFP}(c_{\mathbf{s}}) & =) & [\![\, q \,]\!]_{VUP}(c_{\mathbf{s}}) \quad \sqsupseteq \quad c_q \\ & & \Longleftrightarrow \\ ([\![\, \mathbf{s} \,]\!]_{RMaxFP}(c_q) & =) & [\![\, \mathbf{s} \,]\!]_{RSUP}(c_q) \quad \sqsubseteq \quad c_{\mathbf{s}} \end{array}$$

Informell: Gilt im *VUP*-Sinn bei Anfangszusicherung $c_{\mathbf{s}}$ an $\mathbf{s}$ mindestens c_q an q (obere Zeile), ist für Sollzusicherung c_q an q Anfangszusicherung $c_{\mathbf{s}}$ oder weniger an $\mathbf{s}$ ausreichend (untere Zeile) und umgekehrt (vgl. hierzu auch den Adjunktionsbegriff aus Def. 16.2.1.9).

Übungsaufgabe 10.9.1.3

Für monotone (nichtdistributive, nichtadditive) DFA-Spezifikationen fallen die Fixpunktsemantiken und reversen Fixpunktsemantiken nicht mit ihren operationellen Gegenstücken zusammen.

Gelten in Analogie zu Theorem 10.9.1.1 und 10.9.1.2 für monotone (nichtdistributive, nichtadditive) DFA-Spezifikationen $\mathcal{S}_G$ folgende abgeschwächte Beziehungen für alle Anfangszusicherungen $c_{\mathbf{s}} \in \mathcal{C}$ an $\mathbf{s}$ und Sollzusicherungen $c_q \in \mathcal{C}$ an q? Beweis oder Gegenbeispiel.

1. $[\![q]\!]_{MaxFP}(c_{\mathbf{s}}) \sqsupseteq c_q \iff [\![\mathbf{s}]\!]_{RMinFP}(c_q) \sqsubseteq c_{\mathbf{s}}$
2. $[\![q]\!]_{MinFP}(c_{\mathbf{s}}) \sqsupseteq c_q \iff [\![\mathbf{s}]\!]_{RMaxFP}(c_q) \sqsubseteq c_{\mathbf{s}}$

Kapitel 10.9.2

Korrektheit, Vollständigkeit reverser DFA bzgl. induzierender DFA

Wir betrachten

...ein Analyseszenario, in dem:

- ϕ die interessierende Programmeigenschaft (z.B., Verfügbarkeit eines Terms, Lebendigkeit einer Variablen, Wertkonstanz eines Ausdrucks, etc.)
- $\mathcal{S}_G^\phi$ eine DFA-Spezifikation für ϕ
- ${\mathcal{S}_G^\phi}^R$ die von $\mathcal{S}_G^\phi$ induzierte reverse DFA-Spezifikation
- $c_{\mathbf{s}}$ eine Anfangszusicherung an $\mathbf{s}$
- c_q eine Sollzusicherung an q

bezeichnen.

Korrektheit, Vollständigkeit von RDFA für DFA

Definition 10.9.2.1 (Korrektheit)

${\mathcal{S}_G^\phi}^R$ ist

1. *RVUP*-korrekt für $\mathcal{S}_G^\phi$, wenn für alle c_q gilt:
   $[\![\mathbf{s}]\!]_{RVUP}(c_q) \sqsubseteq c \Rightarrow [\![q]\!]_{SUP}(c) \sqsupseteq c_q$.
2. *RSUP*-korrekt für $\mathcal{S}_G^\phi$, wenn für alle c_q gilt:
   $[\![\mathbf{s}]\!]_{RSUP}(c_q) \sqsubseteq c \Rightarrow [\![q]\!]_{VUP}(c) \sqsupseteq c_q$.

Definition 10.9.2.2 (Vollständigkeit)

${\mathcal{S}_G^\phi}^R$ ist

1. *RVUP*-vollständig für $\mathcal{S}_G^\phi$, wenn für alle $c_\mathbf{s}$ gilt:
   $[\![q]\!]_{SUP}(c_\mathbf{s}) \sqsupseteq c \Rightarrow [\![\mathbf{s}]\!]_{RVUP}(c) \sqsubseteq c_\mathbf{s}$.
2. *RSUP*-vollständig für $\mathcal{S}_G^\phi$, wenn für alle $c_\mathbf{s}$ gilt:
   $[\![q]\!]_{VUP}(c_\mathbf{s}) \sqsupseteq c \Rightarrow [\![\mathbf{s}]\!]_{RSUP}(c) \sqsubseteq c_\mathbf{s}$.

Folgerungen

1. Für distributive und additive DFA-Probleme $\mathcal{S}_G^\phi$ liefern die Verbindungstheoreme 10.9.1.1 und 10.9.1.2, dass die induzierten reversen DFA-Probleme $\mathcal{S}_G^{\phi^R}$ korrekt und vollständig für $\mathcal{S}_G^\phi$ im Sinn der Definitionen 10.9.2.1 und 10.9.2.2 sind.
2. Ist $\mathcal{S}_G^\phi$ korrekt und vollständig für ϕ im Sinn der Definitionen 8.11.1 und 8.11.2 (und somit distributiv bzw. additiv), so kann eine *MaxFP*/*MinFP*-Analyse auf entsprechende (wiederholte) *RMinFP*/*RMaxFP*-Analysen zurückgeführt werden.
3. Für (unidirekt.) Bitvektorprobleme lassen sie sich sogar vollständig ersetzen, da deren Analyseresultate für Programmpunkte binär sind (Eigenschaft gilt/gilt nicht).

Auf diesen Folgerungen baut sog. anforderungsgetriebene Datenflussanalyse auf (s. Kapitel 10.9.3).

Übungsaufgabe 10.9.2.3

Was gilt für

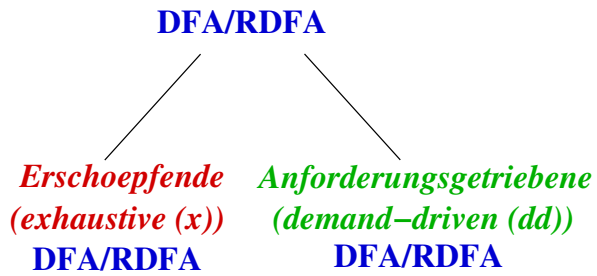
1. Korrektheit und Vollständigkeit im Sinn von Definition 10.9.2.1 und 10.9.2.2 für das induzierte reverse DFA-Problem $\mathcal{S}_G^{\phi^R}$ eines monotonen, aber nicht distributiven DFA-Problems $\mathcal{S}_G^{\phi}$?
2. Was gilt für die Rückführbarkeit der Datenflussanalyse für monotone, aber nicht distributive DFA-Probleme auf anforderungsgetriebene Datenflussanalyse auf Grundlage reverser Datenflussanalyse?

Begründen Sie die Richtigkeit ihrer Behauptungen.

Kapitel 10.9.3

Anforderungsgetriebene Datenflussanalyse

Erschöpfende vs. anforderungsgetrieb. Analyse



- Erschöpfende Analyse: Vollständige Berechnung der Fixpunktlösungen (*MaxFP*/*MinFP* bzw. *RMinFP*/*RMaxFP*) eines DFA/RDFA-Problems.
- Anforderungsgetriebene Analyse: Berechnung der Fixpunktlösungen eines DFA/RDFA-Problems für eine (ggf. leere) Teilmenge von Programmpunkten und approximative Berechnung für übrige Programmpunkte bei Terminierung der Fixpunktiteration sobald das Analyseergebnis für den 'interessanten' Teil feststeht.

  Im engeren Sinn: Lösung eines DFA-Problems durch anforderungsgetriebene Lösung des induz. RDFA-Problems.

Beobachtung

...in günstigen Fällen

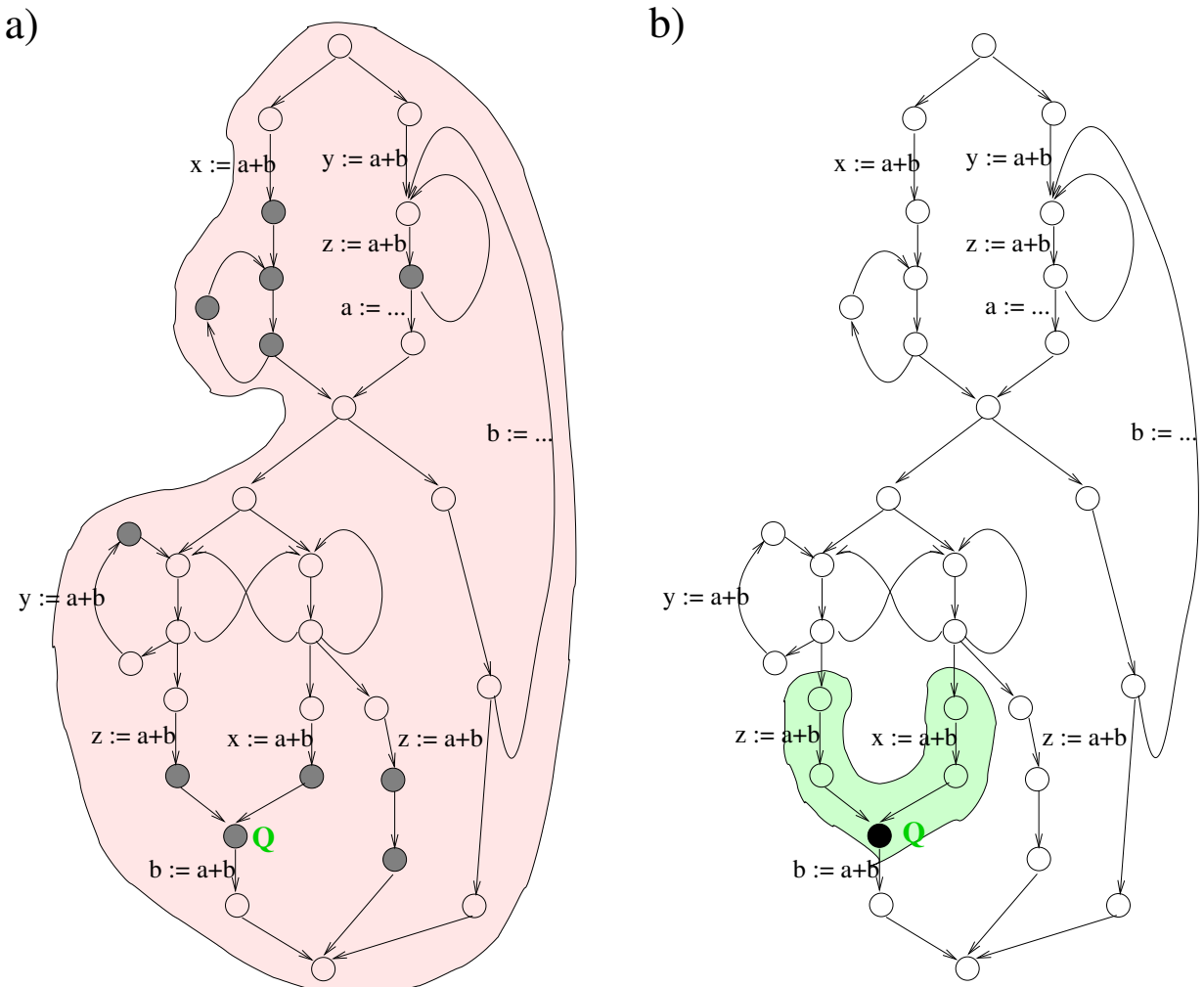
- kann der Aufwand anforderungsgetriebener Analyse erheblich niedriger sein als der der entsprechenden erschöpfenden Analyse.

...in ungünstigen Fällen

- ergibt sich kein Vorteil.

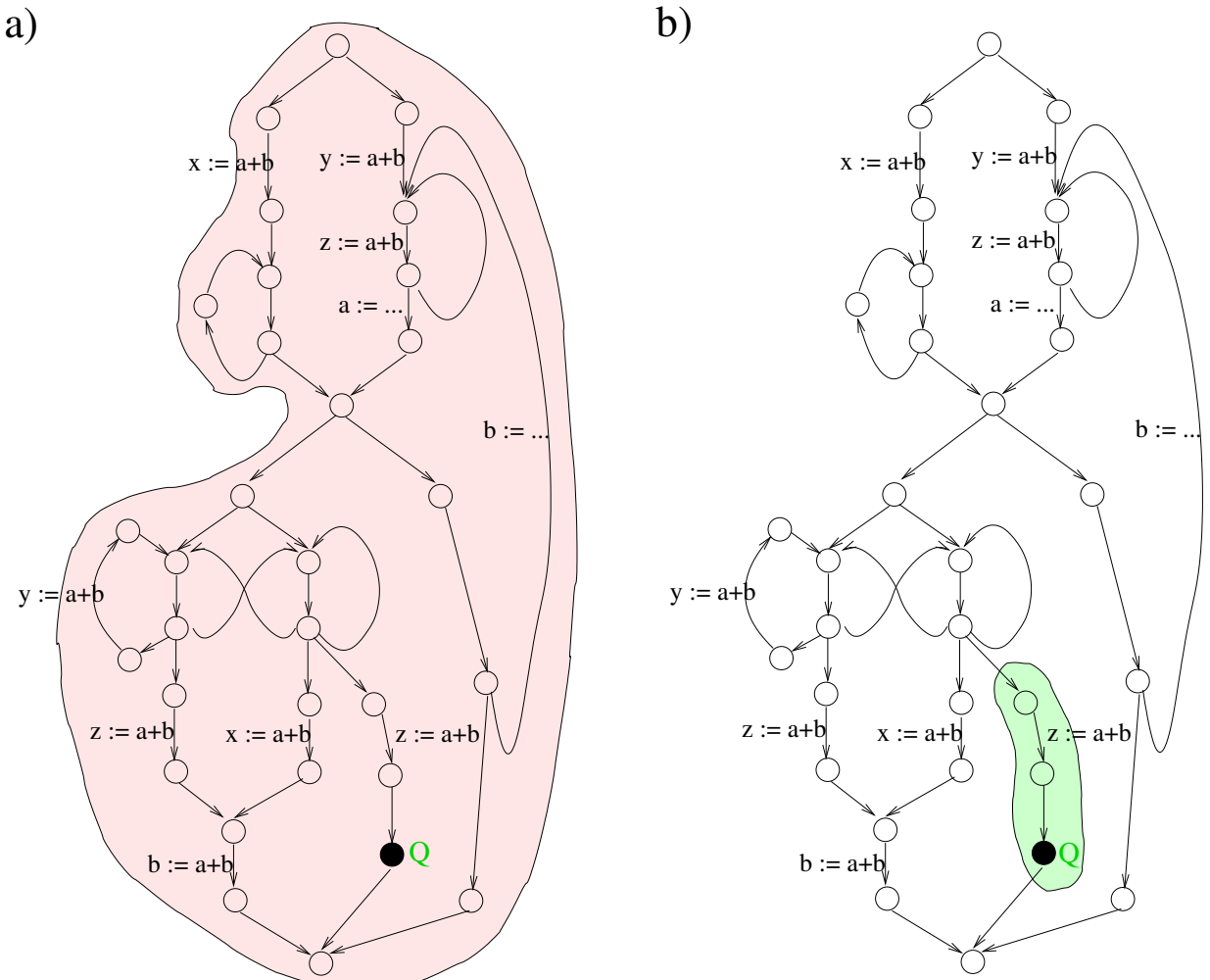
Beispiel: Aufwandsvergleich xDFA, ddDFA (1)

Verfügbarkeit an Programmpunkt Q: Informeller/anekdotischer Vergleich von Berechnungsaufwand erschöpfend (rosa), anforderungsgetrieben (grün):



Beispiel: Aufwandsvergleich xDFA, ddDFA (2)

Verfügbarkeit an Programmpunkt Q: Informeller/anekdotischer Vergleich von Berechnungsaufwand erschöpfend (rosa), anforderungsgetrieben (grün):

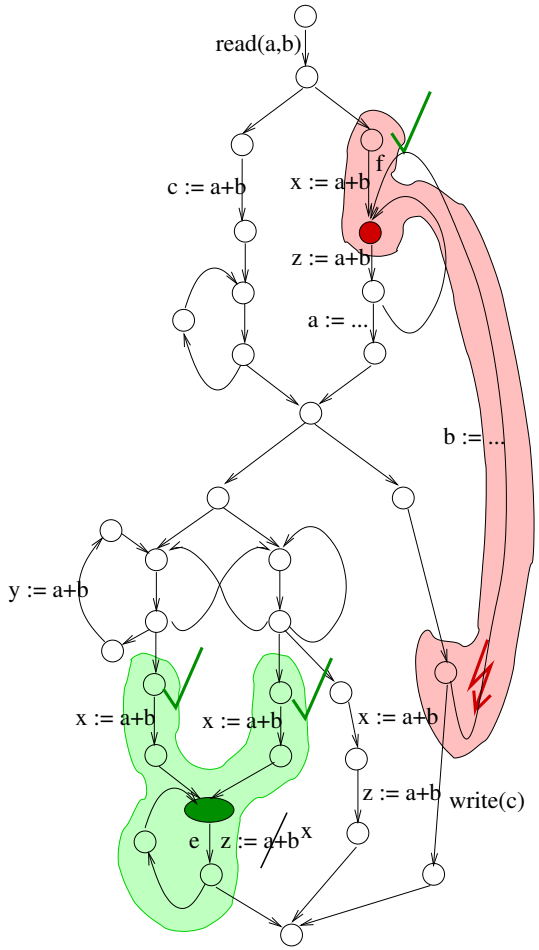


Kapitel 10.9.4

‘Hot Spot’-Analysatoren, -optimierer

‘Hot Spot’-Analysatoren und -optimierer

...fokussieren Analyse und Optimierung auf ‘interessante’ Programmstellen:



‘Hot Spot’–Optimierer

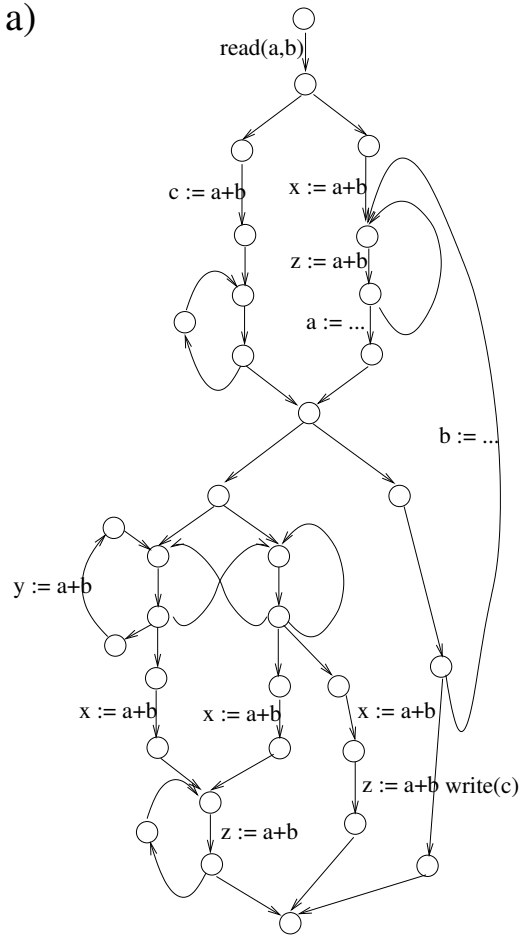
Programmpunkt ✓ erfuellt die Eigen–schaft **available**,

Punkt ⚡ jedoch nicht!

Zum Vergleich: Standardanalysatoren/-opt.

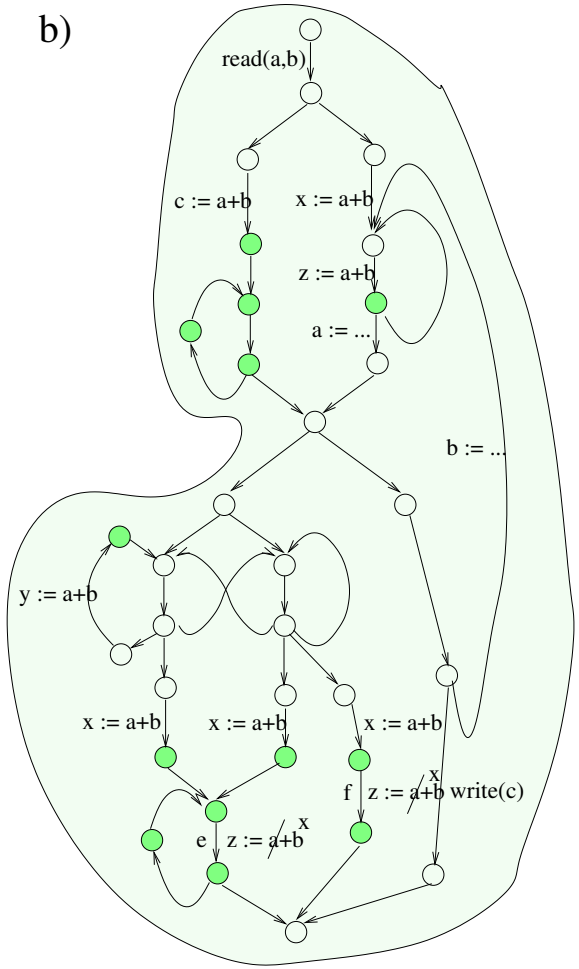
...betrachten stets das ganze Programm, auch 'nicht interessante' Teile:

a)



Flussgraph

b)



Datenflussinformation

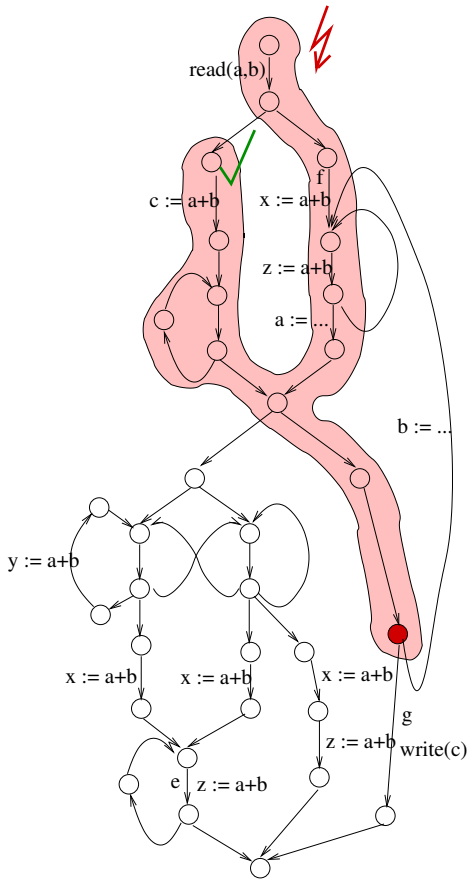
- : Programmpunkte, die die Eigenschaft **available** erfuellen.

Kapitel 10.9.5

Fehlersucher

Fehlersucher

...z.B. zur Erkennung uninitialisierter Variablen (praktisch relevant: Leerzeiger-Referenzen (engl. null pointer references)):



Fehlersucher (Debugger)

Variable c ist am Programm–punkt ● auf einigen Programm–pfaden nicht initialisiert worden.

Kapitel 10.10

Zusammenfassung, Ausblick

Reverse Datenflussanalyse

...und Datenflussanalyse bilden in gleicher Weise ein duales Paar wie stärkste Nachbedingungs- und schwächste Vorbedingungsverifikation axiomatischer Semantik, genauer untersucht in Kapitel 13.

...hat eine Vielzahl von Anwendungen, darunter

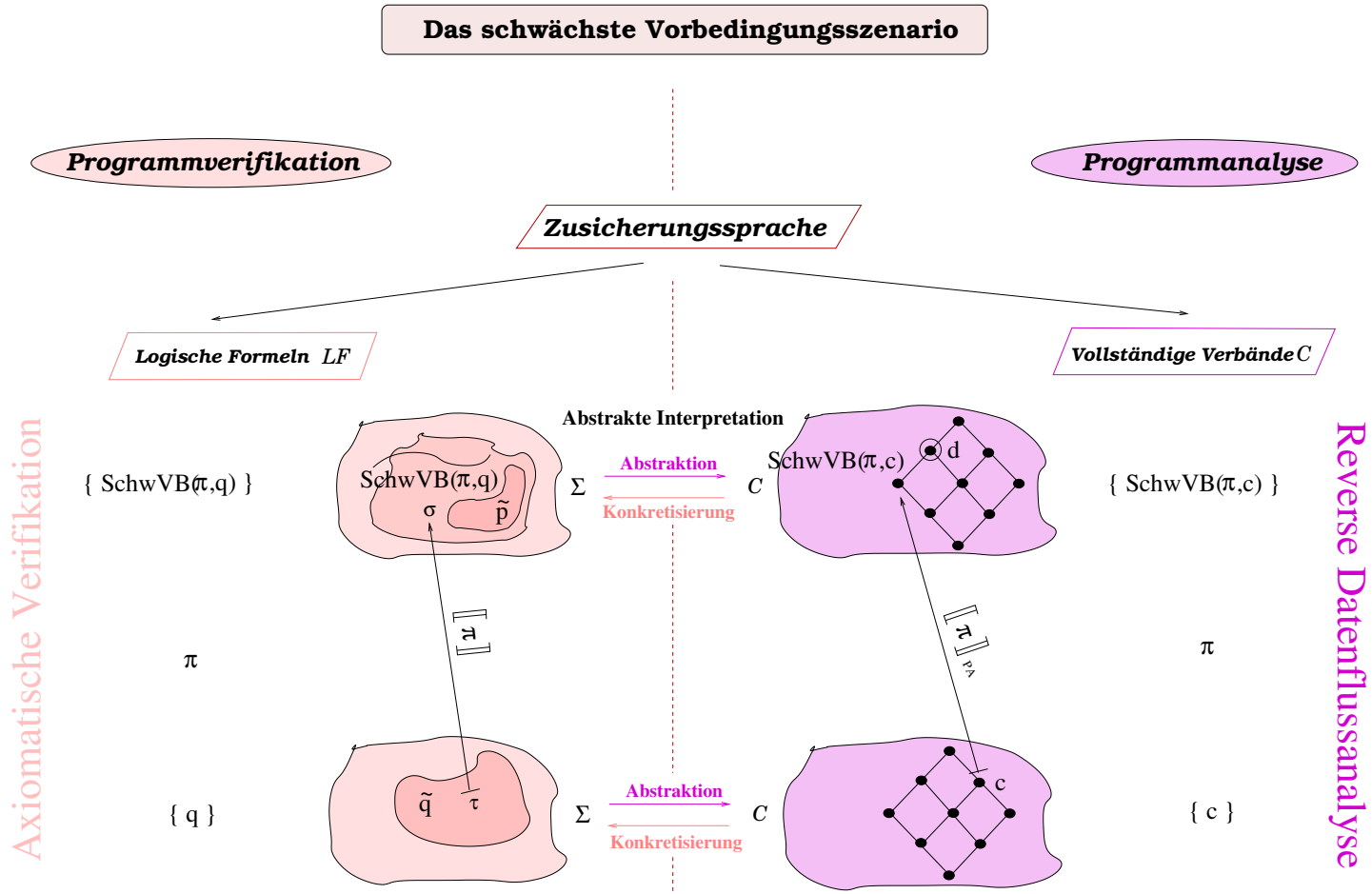
- anforderungsgetriebene Datenflussanalyse (engl. demand-driven data-flow analysis)

und darauf aufbauend und weitergehend den Bau von z.B.:

- 'Hot Spot'-Programmanalysatoren und -optimierern
- Fehlersuchern (engl. Debugger)

und vieler weiterer Programmanalysewerkzeuge.

Verifikation und reverse DFA im Vergleich



SchwVB(π,q) ∈ *LF muss erfüllen:*

(1) $\models_{PV} \{ \text{SchwVB}(\pi,q) \}\, \pi\, \{ q \}$

(2) $\forall\, p \in LF.\ \models_{PV} \{ p \}\pi \{ q \}$ impliziert $p \Longrightarrow \text{SchwVB}(\pi,q)$

SchwVB(π,c) ∈ *C muss erfüllen:*

(1) $\models_{PA} \{ \text{SchwVB}(\pi,c) \}\, \pi\, \{ c \}$

(2) $\forall\, d \in C.\ \models_{PA} \{ d \}\pi \{ c \}$ impliziert $d \sqsupseteq \text{SchwVB}(\pi,c)$

Kapitel 10.11

Literaturverzeichnis, Leseempfehlungen

Reading for Chapter 10 (1)


Gagan Agrawal. *Demand-driven Construction of Call Graphs*. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 125-140, 2000.


Wayne A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part I - Exhaustive Analysis*. Acta Informatica 10(3):245-264, 1978.


Wayne A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part II - Demand Analysis*. Acta Informatica 10(3):265-272, 1978.

Reading for Chapter 10 (2)

Ras Bodík, Rajiv Gupta, Vivek Sarkar. *ABCD: Eliminating Array Bounds Check on Demand*. In Proceedings of the ACM SIGPLAN Conference on Porgramming Language Design and Implementation (PLDI 2000), ACM SIGPLAN Notices 35(5):321-333, 2000.

Evelyn Duesterwald. *A Demand-driven Approach for Efficient Interprocedural Data-Flow Analysis*. PhD thesis, University of Pittsburgh, PA, USA, 1996.

Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *Demand-driven Computation of Interprocedural Data Flow*. In Conference Record of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95), 37-48, 1995.

Reading for Chapter 10 (3)


Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Demand-driven Analyzer for Data Flow Testing at the Integration Level*. In Proceedings of the IEEE Conference on Software Engineering (CoSE'96), 575-586, 1996.


Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Practical Framework for Demand-driven Interprocedural Data Flow Analysis*. ACM Transactions on Programming Languages and Systems (TOPLAS) 19(6):992-1030, 1997.


Leandro Faccinetti, Zachary Palmer, Scott Smith. *Higher-order Demand-driven Program Analysis*. ACM Transactions on Programming Languages and Systems (TOPLAS) 41(3):14:Computing Surveys 51(3):14:1-53, 2019.

Reading for Chapter 10 (4)

Susan Horwitz, Thomas Reps, Mooly Sagiv. *Demand Interprocedural Data Flow Analysis*. In Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-3), 104-115, 1995.

Jens Knoop. *Demand-driven Analysis of Explicitly Parallel Programs: An Approach based on Reverse Data-Flow Analysis*. In Proceedings of the 9th International Workshop on Compilers for Parallel Computers (CPC 2001), 151-162, 2001.

Jens Knoop. *Data-Flow Analysis for Hot-Spot Program Optimization*. In Proceedings of the 14th Biennial Workshop on 'Programmiersprachen und Grundlagen der Programmierung' (KPS 2007). Bericht A-07-07 der Institute für Mathematik und Informatik, Universität Lübeck, Deutschland, 124-131, 2007.

Reading for Chapter 10 (5)


Yuan Lin, David A. Padua. *Demand-driven Interprocedural Array Property Analysis*. In Proceedings of the 12th International Conference on Languages and Compilers for Parallel Computing (LCPC'99), Springer-V., LNCS 1863, 303-317, 1999.


Thomas Reps. *Solving Demand Versions of Interprocedural Analysis Problems*. In Proceedings of the 5th International Conference on Compiler Construction (CC'95), Springer-V., LNCS 786, 389-403, 1994.


Thomas Reps. *Demand Interprocedural Program Analysis using Logic Databases*. In Applications of Logic Databases, R. Ramakrishnan (Hrsg.), Kluwer Academic Publishers, 1994.

Reading for Chapter 10 (6)


Mary Lou Soffa. *Tutorial: Techniques to improve the Scalability and Precision of Data Flow Analysis*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 355-356, 1999.


Peng Tu, David A. Padua. *Gated SSA-based Demand-driven Symbolic Analysis for Parallelizing Computers*. In Proceedings of the International Conference on Supercomputing (ICS'95), 414-423, 1995.


Xin Yuan, Rajiv Gupta, Rami Melhem. *Demand-driven Data Flow Analysis for Communication Optimization*. Parallel Processing Letters 7(4):359-370, 1997.


Xin Zheng, Radu Rugina. *Demand-driven Alias Analysis for C*. In Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008), 197-208, 2008.

Kapitel 11

Abstrakte parallele Semantiken, parallele Analysesemantiken

Kapitel 11.1

Die Sprache ParWhile

Die Erweiterung von While zu ParWhile

Wir erweitern die Sprache While um eine

- Parallelanweisung, in Zeichen: $\|$

Die erweiterte Sprache nennen wir ParWhile und treffen folgende Annahmen:

- Die Komponenten von Parallelanweisungen werden parallel auf einem gemeinsamen Speicher (engl. shared memory) ausgeführt.
- Es gibt weder Sprünge von außen in Parallelanweisungen hinein noch Sprünge aus Parallelanweisungen heraus noch Sprünge von einer in andere Komponenten einer Parallelanweisung.
- Parallelanweisungen terminieren dann und nur dann, wenn alle ihre Komponenten terminiert sind.
- Zuweisungen (in Parallelanweisungen) sind atomar.

Verschränkungssemantik für ParWhile

...als Semantik von ParWhile-Programmen legen wir eine Verschränkungssemantik (engl. interleaving semantics) fest.

Wichtig: Trotz der sparsamen Erweiterung treten mit diesen Festlegungen die zentralen Phänomene und Charakteristika paralleler Programme und paralleler Programmausführungen auf: Die

1. Verschränkung der Ausführung von Anweisungen.
2. Synchronisation von parallelen Komponenten.

Erweiterungen (z.B. zur dynamischen Prozesserzeugung) sind möglich, werden in der Folge aber nicht betrachtet.

Kapitel 11.2

Abstrakte Programmmodellierung: Parallele Flussgraphen

Parallele Flussgraphen

Wir repräsentieren parallele Programme aus ParWhile in Form

- knotenbenannter paralleler Flussgraphen

$$G = (N, E, \mathbf{s}, \mathbf{e})$$

Dabei gilt: Teilgraphen von G, die

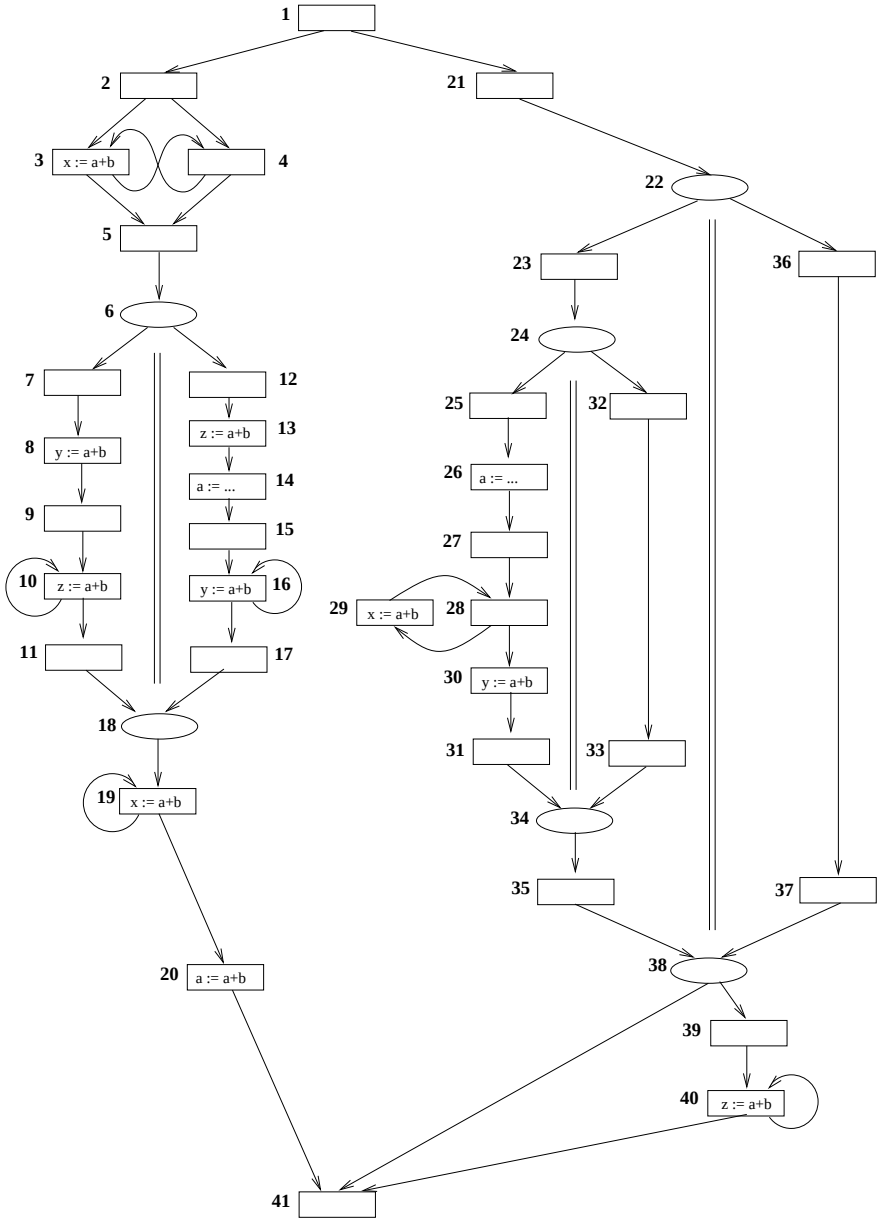
- parallele Anweisungen oder deren Komponenten

repräsentieren, sind Graphen i.S.v. Kapitel 7 mit je genau einem Eintritts- und Austrittspunkt (engl. single entry/single exit regions), die ausschließlich mit den Ein- bzw. Austrittspunkten ihrer Komponentengraphen verbunden sind.

Graphisch stellen wir Ein- und Austrittsknoten paralleler Anweisungen als Ellipsen dar und heben die Zusammengehörigkeit paralleler Komponenten durch zwei Parallelen hervor.

Beispiel: Ein paralleler Flussgraph

...der in Kapitel 11 und 12 als durchgehendes Beispiel dient.



Kapitel 11.2.1

Vereinbarungen, Bezeichnungen

Vereinbarungen, Bezeichnungen (1)

Sei G ein paralleler Flussgraph.

Vereinbarungen:

- Ein- und Austrittsknoten paralleler Anweisungen und ihrer Komponenten sind mit der leeren Anweisung `skip` benannt.

Bezeichnungen:

- $\mathcal{G}_{\mathcal{P}}(G)$: Die Menge aller parallelen Teilgraphen von G.
- $\mathcal{G}_{\mathcal{C}}(G')$: Die Menge der Komponentengraphen von G', $G' \in \mathcal{G}_{\mathcal{P}}(G)$.
- $\mathcal{G}_{\mathcal{C}}(G) =_{df} \bigcup\{\mathcal{G}_{\mathcal{C}}(G') \mid G' \in \mathcal{G}_{\mathcal{P}}(G)\}$
- $N^N_{\mathcal{P}} =_{df} \{\, start(G) \mid G \in \mathcal{G}_{\mathcal{P}}(G) \,\}$: Die Menge der Eintrittsknoten paralleler Teilgraphen von G.
- $N^X_{\mathcal{P}} =_{df} \{\, end(G) \mid G \in \mathcal{G}_{\mathcal{P}}(G) \,\}$: Die Menge der Austrittsknoten paralleler Teilgraphen von G.

Vereinbarungen, Bezeichnungen (2)

Weiters bezeichnen:

- $\mathcal{G}_{\mathcal{P}}^{max}(G) =_{df}$
  $\{ G \in \mathcal{G}_{\mathcal{P}}(G) \mid \forall\, G' \in \mathcal{G}_{\mathcal{P}}(G).\ G \subseteq G' \Rightarrow G = G' \}$
- $\mathcal{G}_{\mathcal{P}}^{min}(G) =_{df}$
  $\{ G \in \mathcal{G}_{\mathcal{P}}(G) \mid \forall\, G' \in \mathcal{G}_{\mathcal{P}}(G).\ G' \subseteq G \Rightarrow G' = G \}$

die Mengen maximaler (äußerster) und minimaler (innerster) paralleler Graphen von G.

Dabei gilt: $G_1 \subseteq G_2$ gdw $N_1 \subseteq N_2 \ \wedge\ E_1 \subseteq E_2$.

Vereinbarungen, Bezeichnungen (3)

Die Abbildungen *pfg* und *cfg* ordnen jedem Knoten aus einem Flussgraphen G' aus $\mathcal{G}_{\mathcal{P}}(G)$ bzw. $\mathcal{G}_{\mathcal{C}}(G)$ den kleinsten n enthaltenden Flussgraphen aus $\mathcal{G}_{\mathcal{P}}(G)$ bzw. $\mathcal{G}_{\mathcal{C}}(G)$ zu:

$$pfg(n) =_{df} \begin{cases} \bigcap\{\, G' \in \mathcal{G}_{\mathcal{P}}(G) \mid n \in Nodes(G') \,\} & \text{falls } n \in Nodes(\mathcal{G}_{\mathcal{P}}(G)) \\ G & \text{sonst} \end{cases}$$

$$cfg(n) =_{df} \begin{cases} \bigcap\{\, G' \in \mathcal{G}_{\mathcal{C}}(G) \mid n \in Nodes(G') \,\} & \text{falls } n \in Nodes(\mathcal{G}_{\mathcal{C}}(G)) \\ G & \text{sonst} \end{cases}$$

Wir dehnen die Abbildungen *pfg* und *cfg* von Knoten auf Graphen aus, indem wir das Bild von Graphen als Bild ihrer Knoten definieren.

Kapitel 11.2.2

Rang paralleler Graphen

Rang paralleler (Teil-) Graphen

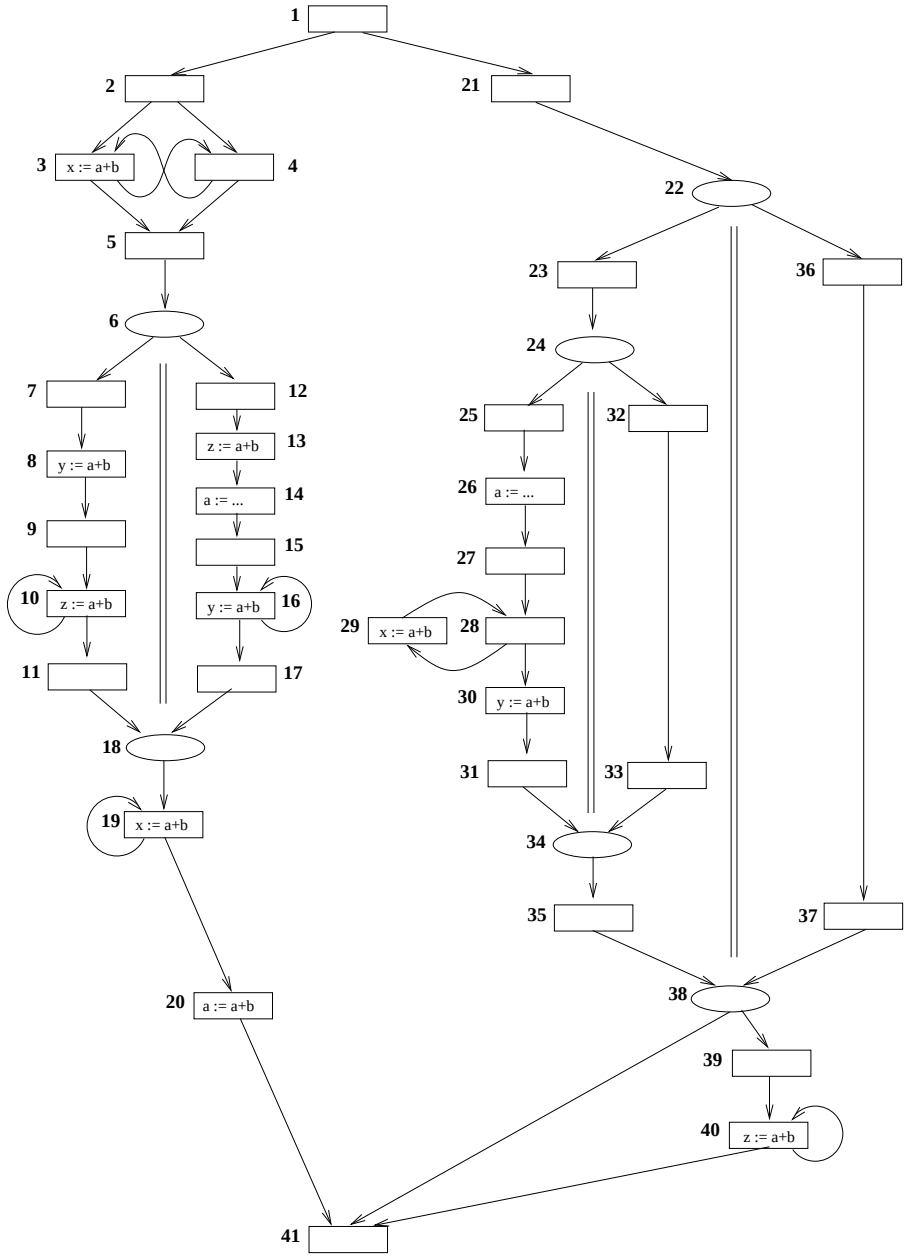
Sei G ein paralleler Flussgraph.

Definition 11.2.2.1 (Rang paralleler Graphen)

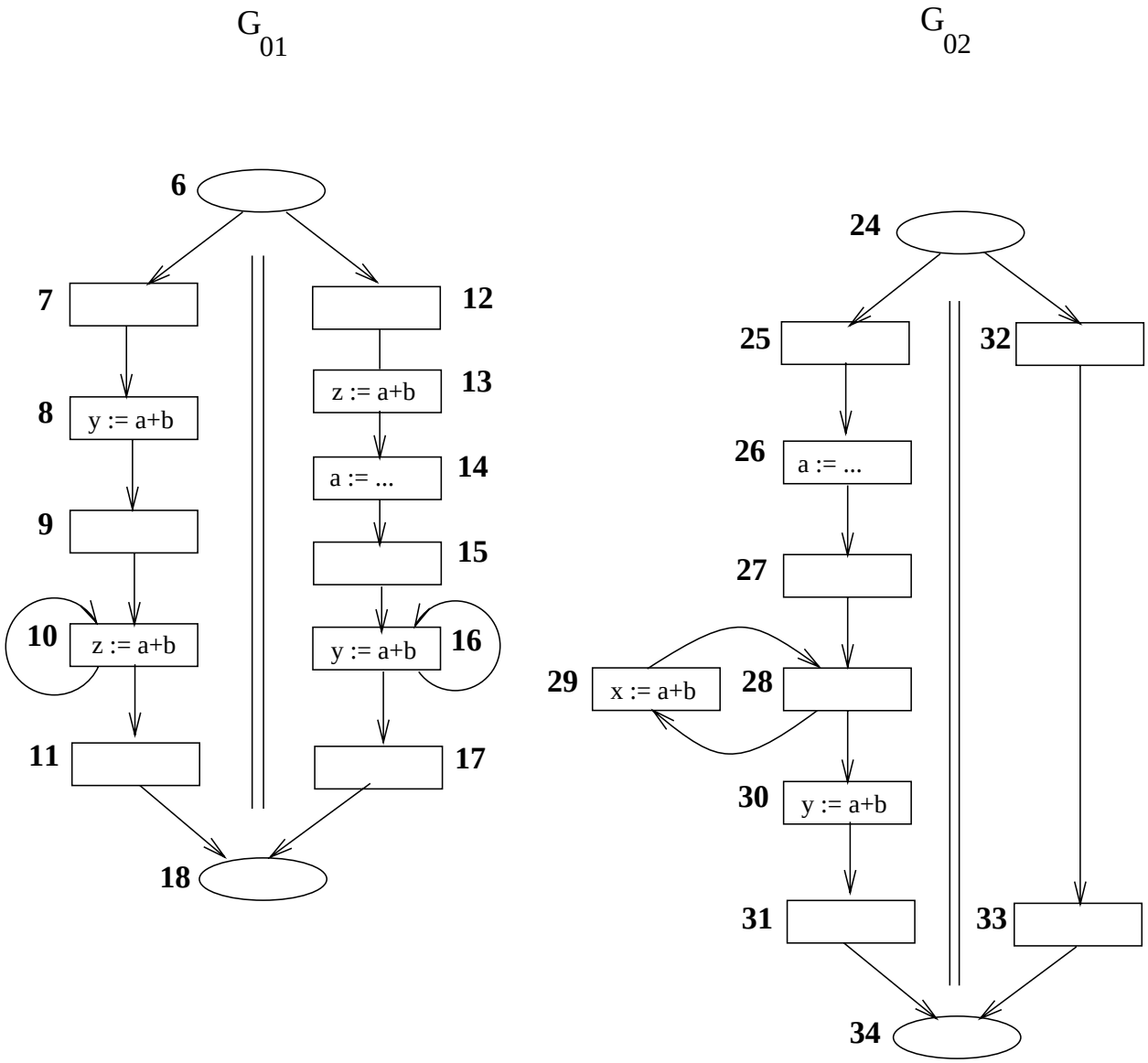
Der Rang (engl. rank) eines parallelen Graphen $G' \in \mathcal{G}_{\mathcal{P}}(G)$ ist definiert durch:

$$rank(G') =_{df} \begin{cases} 0 & \text{falls } G' \in \mathcal{G}_{\mathcal{P}}^{min}(G) \\ max\{\, rank(G'') \mid G'' \in \mathcal{G}_{\mathcal{P}}(G) \wedge G'' \subset G' \,\} + 1 & \text{sonst} \end{cases}$$

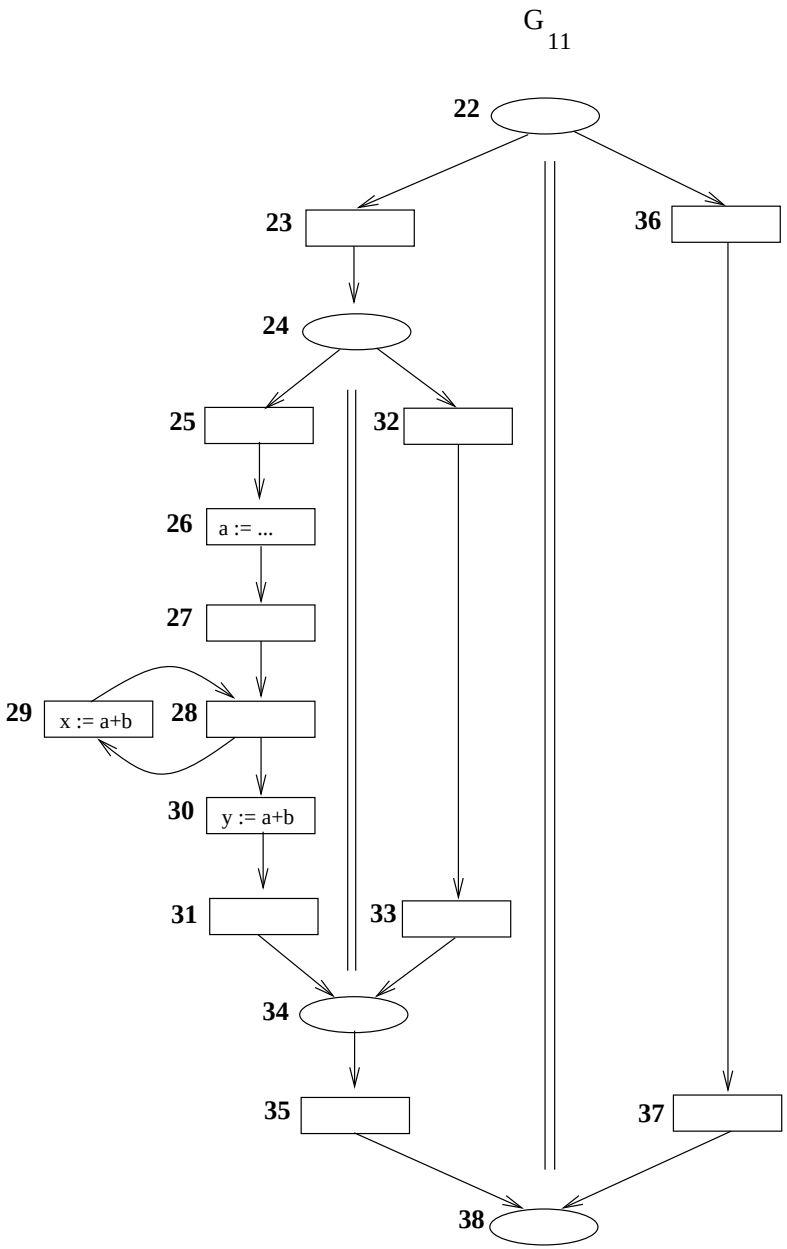
Veranschaul. anhand d. durchgeh. Beispiels G



G enthält zwei parallele Graphen von Rang 0



G enthält einen parallelen Graphen von Rang 1



Kapitel 11.2.3

Formal sequentialisierte Graphen

Formale Sequentialisierung

...paralleler Flussgraphen.

Definition 11.2.3.1 (Sequentialisierte Graphen)

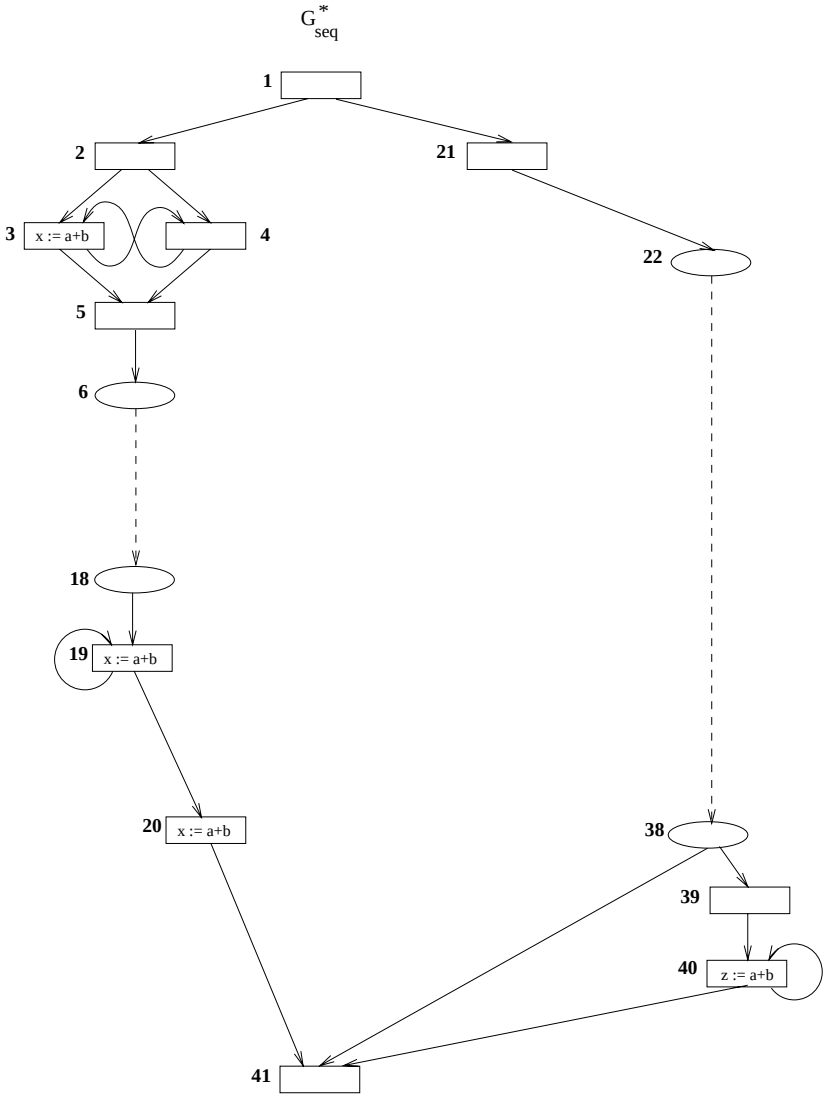
Ist $G' \in \mathcal{G}_{\mathcal{P}}(G)$, so ist G'_{seq} der G' zugeordnete (nicht bedeutungsgleiche!) (formal) sequentialisierte Flussgraph, der aus G' entsteht, in dem alle maximalen parallelen Teilgraphen $G'' \in \mathcal{G}^{max}_{\mathcal{P}}(G')$ durch eine Kante von $start(G'')$ nach $end(G'')$ ersetzt sind.

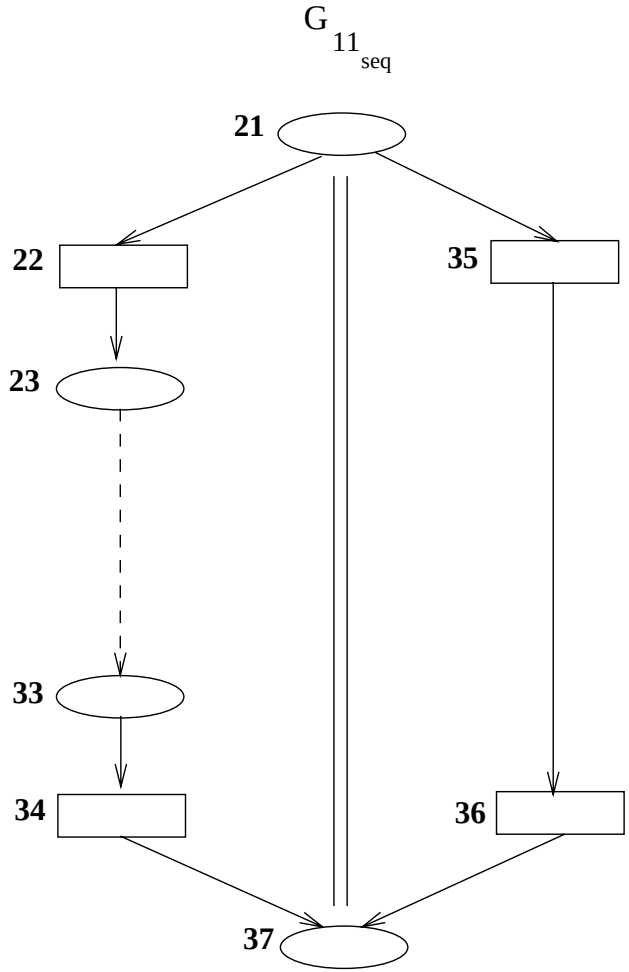
Wichtig:

- Sequentialisierte Graphen enthalten keine parallelen Teilgraphen.
- Im Rahmen der hierarchischen parallelen DFA (s. Kapitel 12.8, Verfahrensschritt 4) werden die beiden verbleibenden Knoten sequentialisierter Graphen mit ihrer Bedeutungsfunktion (Endknoten) und Interferenzwirksamkeit (Anfangsknoten) benannt.

Veranschaul. anhand des durchgeh. Beispiels

...die sequentialisierten Graphen zu Gesamtgraph G und maximalem parallelen Teilgraph G_{11}:





Kapitel 11.2.4

Parallele Geschwister

Parallele Geschwister

Sei G ein paralleler Flussgraph.

Definition 11.2.4.1 (Parallele Geschwistergraphen)

Die Menge der parallelen Geschwistergraphen (pGg) eines Komponentengraphen $G' \in \mathcal{G}_{\mathcal{C}}(G)$ einer Parallelanweisung ist gegeben durch:

$$PG_{pGg}(G') =_{df} \mathcal{G}_{\mathcal{C}}(pfg(G'))\backslash G' \cup \begin{cases} \emptyset & \text{falls } pfg(G') \in \mathcal{G}_{\mathcal{P}}^{max}(G) \\ PG_{pGg}(cfg(pfg(G'))) & \text{sonst} \end{cases}$$

Kapitel 11.2.5

Statische und verschränkte Vorgänger

Statische und verschränkte Vorgänger

...bilden die Menge der unmittelbar vor einem Knoten in einer Parallelanweisung dynamisch ausführbaren Knoten:

1. statische Vorgänger (d.h. gewöhnliche Komponentenknotenvorgänger i.S.v. Kapitel 7)
2. verschränkte Vorgänger (gemäß Definition 11.2.5.1)

Definition 11.2.5.1 (Verschränkte Vorgänger)

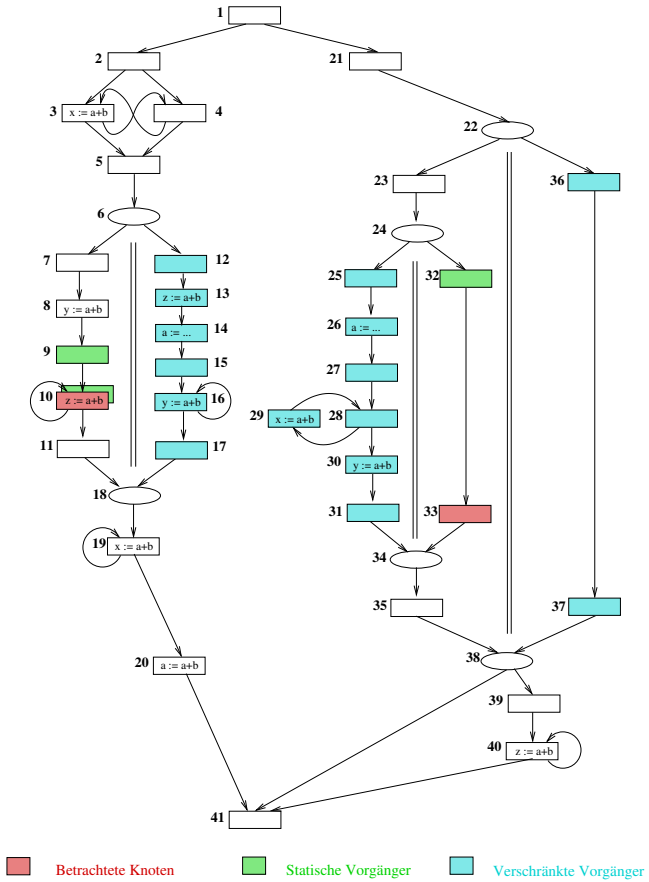
Die Menge der verschränkten Vorgänger (*verschrVorg*) (engl. interleaving predecessors) eines Knotens n eines parallelen Flussgraphens G ist gegeben durch seine Geschwisterknoten in parallelen Geschwistergraphen:

$$Pred_G^{verschrVorg}(n) =_{df} \begin{cases} Nodes(PG_{pGg}(cfg(n))) & \text{falls } n \in Nodes(\mathcal{G}_{\mathcal{C}}(G)) \\ \emptyset & \text{sonst} \end{cases}$$

Veranschaul. anhand des durchgeh. Beispiels

...statische und verschränkte Vorgänger von Knoten 10 u. 33:

- $pred_G(10) = \{9, 10\}$
  $Pred_G^{verschrVorg}(10) = \{12, ..., 17\}$.
- $pred_G(33) = \{32\}$
  $Pred_G^{verschrVorg}(33) = \{25, ..., 31, 36, 37\}$.



Kapitel 11.3

Parallele Pfade

G'-wohlgeformte Knotenfolgen

Sei G ein paralleler Flussgraph, $G' \in \mathcal{G}_{\mathcal{P}}(G)$. %smallskip

Definition 11.3.1 (G-wohlgeformte Knotenfolgen)

Eine Knotenfolge $p = (n_1, \ldots, n_q)$ aus G heißt G'-wohlgeformt gdw

1. die Projektion $p{\downarrow_{G'_{seq}}}$ von p auf G'_{seq} ist Element der Pfadmenge $\mathbf{P}_{G'_{seq}}[start(G'_{seq}), end(G'_{seq})]$.
2. für alle Knotenvorkommen $n_i \in N^N_{\mathcal{P}}$ von p gibt es einen Index $j \in \{i+1, \ldots, q\}$ mit der Eigenschaft:
   - 2.1 $n_j \in N^X_{\mathcal{P}}$.
   - 2.2 n_j ist der Nachfolger von n_i auf $p{\downarrow_{G'_{seq}}}$.
   - 2.3 $\forall\, G'' \in \mathcal{G}_{\mathcal{C}}(pfg(n_i))$. $(n_{i+1}, \ldots, n_{j-1})$ ist G''-wohlgeformt.

Wichtig: Bedingung 2.3 stellt Synchronisation sicher: Am Austrittsknoten einer Parallelanweisung müssen alle ihre parallelen Komponenten vollständig ausgeführt und terminiert sein.

Parallele Pfade

Sei G ein paralleler Flussgraph.

Definition 11.3.2 (Parallele Pfade)

Eine Knotenfolge

1. $p = (\mathbf{s} \equiv n_1, \ldots, n_k \equiv \mathbf{e})$ heißt paralleler Pfad von **s** nach **e**, wenn p G-wohlgeformt ist.
2. $p = (m \equiv n_1, \ldots, n_k \equiv m)$ heißt paralleler Pfad von m nach n, wenn p Teilpfad eines parallelen Pfades von **s** nach **e** ist.

Bezeichnungen

Wir bezeichnen die Menge der parallelen Pfade von m

- nach n mit:

$$\mathbf{PP}[m, n]$$

- zu einem (statischen oder verschränkten) Vorgänger von n mit:

$$\mathbf{PP}[m, n[$$

$$=_{df} \{(n_1, \ldots, n_k) \mid (n_1, \ldots, n_k, n_{k+1}) \in \mathbf{PP}[m, n]\}$$

Bemerkung: Ist G aus dem Kontext nicht eindeutig bestimmt, schreiben wir statt $\mathbf{PP}[m, n]$ und $\mathbf{PP}[m, n[$ genauer $\mathbf{PP}_G[m, n]$ und $\mathbf{PP}_G[m, n[$.

Sei in der Folge

... dieses Kapitels:

- $G = (N, E, \mathbf{s}, \mathbf{e})$ ein paralleler Flussgraph
- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq)$ bzw. $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ ein Verband bzw. vollständiger Verband

Kapitel 11.4

Parallele lokale abstrakte Semantiken

Parallele lokale abstrakte Semantiken

Gänzlich analog zu Kapitel 7.3 führen wir für G ein:

Definition 11.4.1 (Parallele lok. abstrakte Semantik)

Eine parallele lokale abstrakte (Knoten-) Semantik von G ist eine Funktion:

$$[\![\ ]\!] : N \to (\mathcal{C} \to \mathcal{C})$$

die jedem Knoten von G eine Funktion auf der Elementmenge $\mathcal{C}$ von $\widehat{\mathcal{C}}$ als Bedeutung zuordnet.

Kapitel 11.5

Parallele operationelle globale abstrakte Semantiken

Kapitel 11.5.1

Pfadausdehnung paralleler lokaler abstrakter Semantiken

Pfadausdehnung paralleler lokaler Semantiken

Vollkommen analog zu Kapitel 7.4.1 definieren wir:

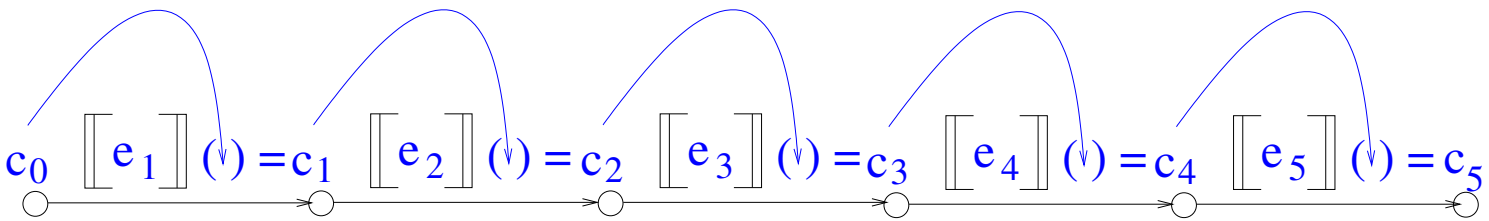
Definition 11.5.1.1 (Pfadausdehnung von $[\![\ ]\!]$)

Die Pfadausdehnung $[\![\ p\ ]\!]$ einer parallelen lokalen abstrakten Semantik $[\![\ ]\!]$ auf einen parallelen Pfad $p = \langle n_1, n_2, \ldots, n_k \rangle$ ist kompositionell definiert durch:

$$[\![\ p\ ]\!] =_{df} \begin{cases} Id_{\mathcal{C}} & \text{falls } \lambda_p < 1 \\ [\![\ \langle n_2, \ldots, n_k \rangle\ ]\!] \circ [\![\ n_1\ ]\!] & \text{sonst} \end{cases}$$

wobei $Id_{\mathcal{C}} = \lambda c \in \mathcal{C}.\, c$ die Identität auf $\mathcal{C}$ bezeichnet.

Veranschaulichung der Pfadausdehnung von $[\![\ ]\!]$:



Kapitel 11.5.2

Parallele Aufsammelsemantik

Parallele Aufsammelsemantik

Definition 11.5.2.1 (Parallele Aufsammelsemantik)

Die von $[\![\ ]\!]$ induzierte parallele Aufsammelsemantik (oder: nichtdeterministische parallele globale abstrakte Semantik) an den Eingängen der Knoten aus N ist definiert durch:

$$[\![\ ]\!]_{PAS} : \mathcal{C} \to N \to \mathcal{P}(\mathcal{C})$$

$$[\![\ ]\!]_{PAS} =_{df} \lambda c \in \mathcal{C}.\ \lambda n \in N.\ \{ [\![p]\!](c) \mid p \in \mathbf{PP}[\mathbf{s}, n[\}$$

wobei $\mathcal{P}$ den Potenzmengenoperator bezeichnet.

Ohne besondere Anforderungen an $\widehat{\mathcal{C}}$ und $[\![\ ]\!]$ gilt:

Lemma 11.5.2.2 (PAS-Wohldefiniertheit)

Die PA-Semantik $[\![\ ]\!]_{PAS}$ von G ist wohldefiniert.

Kapitel 11.5.3

Schnitt-über-alle-parallele-Pfade-Semantik

Die Schnitt-über-alle-par.-Pfade (*SUPP*) Sem.

Definition 11.5.3.1 (*SUPP*-Semantik)

Die deterministische *SUPP*-Semantik von G an Knoteneingängen ist definiert durch:

$$[\![\ ]\!]_{SUPP} : \mathcal{C} \to N \to \mathcal{C}$$

$$\begin{aligned} [\![\ ]\!]_{SUPP} &=_{df} \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \bigsqcap [\![\ n\ ]\!]_{PAS}(c) \\ &= \lambda\, c \in \mathcal{C}.\ \lambda\, n \in N.\ \bigsqcap \{ [\![\ p\ ]\!](c) \mid p \in \mathbf{PP}[\mathbf{s}, n[\ \} \end{aligned}$$

Es gilt:

Lemma 11.5.3.2 (*SUPP* -Wohldefiniertheit)

Die *SUPP* -Semantik $[\![\ ]\!]_{SUPP}$ von G ist wohldefiniert, wenn $\widehat{\mathcal{C}}$ $\sqcap$-vollständig ist.

Kapitel 11.5.4

Vereinigung-über-alle-parallele-Pfade-Semantik

Die Verein.-über-alle-par.-Pfade (*VUPP*) Sem.

Definition 11.5.4.1 (*VUPP*-Semantik)

Die deterministische *VUPP*-Semantik von G an Knoteneingängen ist definiert durch:

$$[\![\ ]\!]_{VUPP} : \mathcal{C} \to N \to \mathcal{C}$$

$$\begin{aligned} [\![\ ]\!]_{VUPP} &=_{df} \quad \lambda c \in \mathcal{C}.\ \lambda n \in N.\ \bigsqcup [\![\ n\ ]\!]_{PAS}(c) \\ &= \quad \lambda c \in \mathcal{C}.\ \lambda n \in N.\ \bigsqcup \{ [\![\ p\ ]\!](c) \mid p \in \mathbf{PP}[\mathbf{s}, n[\ \} \end{aligned}$$

Es gilt:

Lemma 11.5.4.2 (*VUPP*-Wohldefiniertheit)

Die *VUPP*-Semantik $[\![\ ]\!]_{VUPP}$ von G ist wohldefiniert, wenn $\widehat{\mathcal{C}}$ $\sqcup$-vollständig ist.

Kapitel 11.6

Zusammenfassung, Fazit

Zusammenfassung, Fazit

- Das Vorgehen zur Festlegung lokaler und globaler operationeller abstrakter Semantiken für parallele Programme ist vollkommen analog zum Vorgehen für sequentielle Programme in Kapitel 7: Einzig der Pfadbegriff muss aufgrund von
  - verschränkter Anweisungsausführung in parallelen Anweisungen
  - Synchronisation aller parallelen Komponenten als Abschluss paralleler Anweisungen

  dafür angepasst werden: Von durch reguläre Ausdrücke beschreibbaren Pfaden sequentieller Programme zu (technisch aufwändigeren) Beschreibungen von Pfaden paralleler Programme durch parallele Pfade.
- Der Wechsel von kanten- zu knotenbenannten Graphen zur Programmmodellierung ist konzeptuell bedeutungslos: Die Ausdehnung der globalen abstrakten Semantiken von Knoten- auch auf Knotenausgänge ist offensichtlich und trivial.

Kapitel 11.7

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 11 (1)

Jens Knoop, Bernhard Steffen, Jürgen Vollmer. *Parallelism for Free: Efficient and Optimal Bitvector Analyses for Parallel Programs*. ACM Transactions on Programming Languages and Systems 18(3):268-299, 1996.

Samuel P. Midkiff, David A. Padua. *Issues in the Optimization of Parallel Programs*. In Proceedings of the 18th International Conference on Parallel Processing (ICPP'90), Vol. II., 105-113, 1990.

Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019. (Chapter 8, Concurrency)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 11 (2)

Harini Srinivasan, Michael Wolfe. *Analyzing Programs with Explicit Parallelism*. In Proceedings of the 4th International Conference on Languages and Compilers for Parallel Computing (LCPC'91), Springer-V., LNCS 589, 405-419, 1991.

Michael Wolfe, Harini Srinivasan. *Data Structures for Optimizing Programs with Explicit Parallelism*. In Proceedings of the 1st International Conference of the Austrian Center for Parallel Computation, Springer-V., LNCS 591, 139-156, 1991.

Kapitel 12

Parallele Datenflussanalyse

Kapitel 12.1

Parallele DFA-Spezifikationen

Parallele DFA-Semantiken

Definition 12.1.1 (PDFA-Semantiken)

Ein Paar $(\widehat{\mathcal{C}}, [\![\ ]\!])$ mit:

- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ vollständiger Verband
- $[\![\ ]\!] : N \to (\mathcal{C} \to \mathcal{C})$ lokale abstrakte Semantik für G

definiert die (lokale) (DFA-) Semantik einer Datenflussanalyse für G.

PDFA-Spezifikationen, PDFA-Probleme

Definition 12.1.2 (PDFA-Spezifik., PDFA-Probleme)

Ein Tripel $(\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ mit:

- $(\widehat{\mathcal{C}}, [\![\ ]\!])$ (lokale) DFA-Semantik für G
- $c_s \in \mathcal{C}$ initiale Information (oder Anfangszusicherung)

spezifiziert ein konkretes paralleles DFA-Problem (PDFA-Problem), die Probleminstanz $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$.

Kapitel 12.2

SUPP- und *VUPP*-Semantik als zueinander duale spezifizierende Lösungen paralleler DFA-Probleme

Spezifizierende PDFA-Problemlösungen

Vollkommen analog zum sequentiellen Fall aus Kapitel 8.2 legen wir fest:

Definition 12.2.1 (Spezifiz. PDFA-Problemlsg.)

Die *SUPP*- und *VUPP*-Semantik eines parallelen Flussgraphen definieren zwei zueinander duale spezifizierende Lösungen eines parallelen DFA-Problems, seine sog.:

1. *SUPP*-Lösung
2. *VUPP*-Lösung

Kapitel 12.3

Korrektheit, Vollständigkeit, Akkuratheit von PDFA-Algorithmen

Korrektheit, Vollständigkeit, Akkuratheit

...von PDFA-Algorithmen. Auch hier definieren wir vollkommen analog zum sequentiellen Fall aus Kapitel 8.3:

Definition 12.3.1 (Korrekt, vollständig, akkurat)

Ein PDFA-Algorithmus A heißt

1. *SUPP*-korrekt (*VUPP*-korrekt), wenn A für alle PDFA-Probleme $\mathcal{S}_G$ mit einer unteren (oberen) Approximation der *SUPP*- (*VUPP*-) Semantik von G terminiert.
2. *SUPP*-vollständig (*VUPP*-vollständig), wenn A für alle PDFA-Probleme G mit einer oberen (unteren) Approximation der *SUPP*- (*VUPP*-) Semantik von G terminiert.
3. *SUPP*-akkurat (*VUPP*-akkurat), wenn A für alle PDFA-Probleme G exakt mit der *SUPP*- (*VUPP*-) Semantik von G terminiert.

Akkuratheit, Optimalität

Lemma 12.3.2 (Akkuratheit)

Ein PDFA-Algorithmus A ist

1. *SUPP*-akkurat gdw A ist *SUPP*-korrekt und *SUPP*-vollständig.
2. *VUPP*-akkurat gdw A ist *VUPP*-korrekt und *VUP*-vollständig.

Statt von akkurat sprechen wir auch (wieder) von optimal.

Definition 12.3.3 (Optimalität)

Ein PDFA-Algorithmus A heißt *SUPP*-optimal (*VUPP*-optimal) gdw A ist *SUPP*-akkurat (*VUPP*-akkurat).

Beachte: Auch bei *SUPP*-optimal kommt es ausgesprochen auf die harten p's an!

Kapitel 12.4

Unentscheidbarkeit der *SUPP*- und *VUPP*-Semantik

Unentscheidbarkeit der *SUPP*- und *VUPP*-Sem.

...als triviales Korollar der Unentscheidbarkeitsresultate aus Kapitel 8.4 des hier als Spezialfall enthaltenen sequentiellen Falls erhalten wir:

Korollar 12.4.1 (Unentscheidbarkeit)

Die *SUPP*- und *VUPP*-Semantik paralleler Programme bzw. Flussgraphen ist nicht entscheidbar.

Die Unentscheidbarkeit von *SUPP*- und *VUPP*-Semantik macht auch hier die Einführung berechenbarer Alternativen nötig.

Das für parallele Programme notorische Problem der sog. Zustandsexplosion macht dabei schärfere Einschränkungen als im sequentiellen Fall nötig, um Semantiken zu erhalten, die effizient und skalierend berechenbar sind.

Kapitel 12.5

Zustandsexplosion: Herausforderung effizienter und skalierbarer Analyse paralleler Programme

Zustandsexplosionsproblem

Die Zustandszahl paralleler Programme wächst

- exponentiell

in der Zahl paralleler Komponenten, das sog.

Zustandsexplosionsproblem!

Naive Antwort auf das Zustandsexplosionsprob.

...die Idee, die Analyse paralleler Programme auf die Analyse sequentieller Programme zurückzuführen durch Ersetzen paralleler Anweisungen durch ihre zugehörigen nichtdeterministischen sequentiellen Produktprogramme aus der

- Produktkonstruktion ihrer parallelen Komponenten

liegt nahe, ist aber

- nicht praktikabel

da die Größe der Produktprogramme

- exponentiell

in der Zahl paralleler Komponenten wächst.

Anm.: Parallele Programme können als kompakte Darstellung der nicht-deterministischen sequentiellen Produktprogramme angesehen werden.

Hauptergebnis von Kapitel 12

...für unidirektionale Bitvektoranalysen lässt sich das Zustandsexplosionsproblem vollständig vermeiden: sie lassen sich für

- parallele Programme aus ParWhile

ebenso einfach und effizient durchführen wie für

- sequentielle Programme aus While.

Die Vielzahl praktisch relevanter Optimierungen, die auf den Ergebnissen unidirektionaler Bitvektoranalysen beruhen, verleiht diesem Ergebnis Bedeutung: es öffnet den Weg

- darauf aufbauende Optimierungen (nach Anpassung!) ebenfalls von sequentiellen auf parallele Programme auszudehnen und zu übertragen.

Kapitel 12.6

Unidirektionale Bitvektoranalysen

Unidirektionale Bitvektoranalysen/-probleme

...zeichnen sich dadurch aus, dass ausschließlich die drei Funktionen der Menge $\mathcal{F}_{\mathbb{B}}$:

$$\mathcal{F}_{\mathbb{B}} =_{df} \{Cst_{\mathbf{wahr}}, Id_{\mathbb{B}}, Cst_{\mathbf{falsch}}\} \subseteq [\mathbb{B} \to \mathbb{B}]$$

mit

- $Cst_{\mathbf{wahr}} =_{df} \lambda b \in \mathbb{B}.\ \mathbf{wahr}$ ('gen': Eigensch.generierung)
- $Id_{\mathbb{B}} =_{df} \lambda b \in \mathbb{B}.\ b$ ('inv': Eigensch.invarianz)
- $Cst_{\mathbf{falsch}} =_{df} \lambda b \in \mathbb{B}.\ \mathbf{falsch}$ ('kill': Eigensch.zerstörung)

als lokale abstrakte Semantikfunktionen auftreten.

Bemerkung: Die Negation als vierte Funktion auf $\mathbb{B}$:

$$Neg_{\mathbb{B}} =_{df} \lambda b \in \mathbb{B}.\ \neg b$$

ist wegen fehlender Monotonie für DFA-Zwecke uninteressant.

Wichtige Resultate

...für unidirektionale Bitvektoranalysen/-probleme.

Lemma 12.6.1.1

1. Die drei Funktionen aus $\mathcal{F}_{\mathbb{IB}}$ sind distributiv und additiv.
2. $\mathcal{F}_{\mathbb{IB}}$ bildet mit der punktweisen Ordnung auf Funktionen einen unter Funktionskomposition abgeschlossenen vollständigen Verband mit kleinstem Element $Cst_{\mathbf{falsch}}$ und größtem Element $Cst_{\mathbf{wahr}}$.

Hauptlemma 12.6.1.2

Für Funktionen $f_i : \mathcal{F}_{\mathbb{IB}} \to \mathcal{F}_{\mathbb{IB}}$, $1 \leq i \leq k$, auf $\mathcal{F}_{\mathbb{IB}}$ gilt:

$$\exists\, n \in \{1, ..., k\}.\ f_k \circ ... \circ f_2 \circ f_1 = f_n \ \wedge\ \forall\, j \in \{n+1, ..., k\}.\ f_j = Id_{\mathbb{IB}}$$

...der Effekt der Funktionskomposition (in der Anwendung: eines Programmpfades!) ist durch eine einzige Funktion, f_n, bestimmt!

Kapitel 12.7

Parallele denotationelle globale DFA-Semantiken für unidirektionale Bitvektorprobleme: Fixpunktsemantiken

Kapitel 12.7.1

Vorbereitung: Der funktionale denotationelle Semantikansatz

Der funktionale denotat. Semantik-Ansatz

...für sequentielle Programme als punktweise Ausdehnung des *MaxFP*/*MinFP*-Ansatzes auf den Gesamtverband ist der Schlüssel für parallele unidirektionale Bitvektoranalysen.

Informell: Der funktionale denotationelle Semantik-Ansatz

- hebt das Niveau von *MaxFP*/*MinFP*-Ansatz von einzelnen Verbandselementen als Anfangszusicherung auf funktionales Niveau auf dem Gesamtverband, d.h. für jedes Verbandselement als Anfangszusicherung.

In der Folge entwickeln wir den funktionalen denotationellen Semantik-Ansatz für die *MaxFP*-Sicht; die Entsprechung für den *MinFP*-Sicht ergibt sich auf triviale Weise durch Vertauschen von Schnitt- und Vereinigungsoperation des Verbands.

Der funktionale denotat. Semantik-Ansatz

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein (sequentieller) Flussgraph und $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ eine DFA-Spezifikation für G.

Dann ist durch:

Gleichungssystem 12.7.1.1 (Fkt. *MaxFP*-Gleich.syst.)

$$fktinf(n) = \begin{cases} Id_{\mathcal{C}} & \text{falls } n = \mathbf{s} \\ \bigsqcap\{[\![(n,m)]\!] \circ (fktinf(m)) \mid m \in pred(n)\} & \text{sonst} \end{cases}$$

die punktweise funktionale Ausdehnung der denotationellen *MaxFP*-Semantik von G gegeben (vgl. Kapitel 8.7.1):

Gleichungssystem 12.7.1.2 (*MaxFP*-Gleichungssyst.)

$$inf(n) = \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcap\{[\![(m,n)]\!](inf(m)) \mid m \in pred(n)\} & \text{sonst} \end{cases}$$

Äquivalenzresultat

Bezeichnen

- $\nu\text{-}fktinf_{Id_C} : N \to (C \to C)$
- $\nu\text{-}inf_{c_s} : N \to C$

die größten Lösungen der Gleichungssysteme 12.7.1.1 und 12.7.1.2, so gilt das Äquivalenzresultat:

Theorem 12.7.1.3 (Äquivalenz)

$$\forall\, n \in N.\ \forall\, c \in C.\ \nu\text{-}fktinf_{Id_C}(n)(c) = \nu\text{-}inf_c(n)$$

Hauptergebnis: Korrekth., Sicherh., Koinzidenz

Theorem 12.7.1.3 besagt, dass *MaxFP*- und funktionale *MaxFP*-Semantik übereinstimmen; sie sind äquivalent.

Mit der zusätzlichen Bezeichnung $[\![\ ]\!] =_{df} \nu\text{-}fktinf_{Id_C}$ erhalten wir für die funktionale denotationelle Semantik als Korollare zu Theorem 12.7.1.3, Sicherheitstheorem 8.10.1(1) und Koinzidenztheorem 8.10.2(1):

Korollar 12.7.1.4 (Äquivalenz)

$$\forall\, n \in N.\ \forall\, c \in \mathcal{C}.\ [\![\, n \,]\!](c) \;=\; [\![\, n \,]\!]_{MaxFP}(c)$$

Korollar 12.7.1.5 (Sicherheit, Koinzidenz)

1. Sicherheit: $\forall\, n \in N.\ \forall\, c \in \mathcal{C}.\ [\![\, n \,]\!](c) \sqsubseteq [\![\, n \,]\!]_{SUP}(c)$, wenn $[\![\ ]\!]$ monoton ist.
2. Koinzidenz: $\forall\, n \in N.\ \forall\, c \in \mathcal{C}.\ [\![\, n \,]\!](c) = [\![\, n \,]\!]_{SUP}(c)$, wenn $[\![\ ]\!]$ distributiv ist.

Berechnung der globalen Semantikfunktion $[\![\ ]\!]$

...konzeptuell können die Funktionen $[\![\ n\ ]\!] : \mathcal{C} \to \mathcal{C}$, $n \in N$

- Argument für Argument mithilfe des Generischen Fixpunktalgorithmus 8.8.1.1 berechnet werden.

Mit Algorithmus 12.7.1.6 stellen wir ein weniger naives, direktes Verfahren zur Berechnung der Funktionen $[\![\ n\ ]\!]$, $n \in N$, auf dem Niveau von Funktionen auf dem Verband (nicht einzelnen Verbandselementen) vor, das zentraler Bestandteil paralleler unidirektionaler Bitvektoranalysen sein wird.

Generischer Fixpunktalg. 12.7.1.6 (1)

Eingabe: Eine DFA-Spezifikation $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!])$ (alle Startzusicherungen werden simultan betrachtet).

Ausgabe: Nach Terminierung von Algorithmus 12.7.1.6 (s. Terminierungtheorem 12.7.1.7) speichert Variable $fktinf[n]$ die funktionale *MaxFP*-Semantik bzgl. $\mathcal{S}_G$ am Knoten n.

Zusätzlich gilt (s. Korollar 12.7.1.5(1) u. 12.7.1.5(2)): Ist $[\![\ ]\!]$

- distributiv: $fktinf[n]$ speichert
- monoton: $fktinf[n]$ speichert eine untere Approximation

der (funktionalen) *SUP*-Semantik bzgl. $\mathcal{S}_G$ am Knoten n.

Bemerkung: Variable *workset* steuert den Ablauf des iterativen Prozesses. Sie speichert vorübergehend eine Menge von Knoten von G, deren Annotationen sich kürzlich geändert haben und damit die Annotationen ihrer Nachfolgerknoten beeinflussen können.

Generischer Fixpunktalg. 12.7.1.6 (2)

(Prolog: Initialisierung von *fktinf* und *workset*)
FORALL $n \in N \backslash \{\mathbf{s}\}$ DO $fktinf[n] := \lambda c.\ \top$ OD;
$fktinf[\mathbf{s}] := \lambda c.\ c$;
$workset := N$;
(Hauptprozess: Iterative Fixpunktberechnung)
WHILE $workset \neq \emptyset$ DO
    CHOOSE $m \in workset$;
        $workset := workset \backslash \{ m \}$;
        (Aktualisierung der Nachfolgerumgebung von Knoten m)
        FORALL $n \in succ(m)$ DO
            $meet := [\![(m, n)]\!] \circ fktinf[m] \sqcap fktinf[n]$;
            IF $fktinf[n] \sqsupset meet$
                THEN
                    $fktinf[n] := meet$;
                    $workset := workset \cup \{ n \}$
            FI
        OD ESOOHC OD.

Effektivität, Terminierung

Theorem 12.7.1.7 (Effektivität, Terminierung)

Fixpunktalgorithmus 12.7.1.6 terminiert mit der funktionalen *MaxFP*-Semantik bzgl. $\mathcal{S}_G$, wenn:

1. Das lokale DFA-Semantikfunktional $[\![\ ]\!]$ ist monoton.
2. Der Funktionenverband $[\mathcal{C} \to \mathcal{C}]$ erfüllt die absteigende Kettenbedingung.

Proposition 12.7.1.8

Erfüllt $[\mathcal{C} \to \mathcal{C}]$ die absteigende Kettenbedingung, so auch $\mathcal{C}$.

Kapitel 12.7.2

Interferenz und Synchronisation

Interferenzstabilität

...sei $\mathcal{S}_G = (\widehat{\mathcal{F}_{\mathbb{IB}}}, [\![\ ]\!])$ DFA-Spezifikation eines unidirektionalen Bitvektorproblems, n ein Knoten in einer Parallelanweisung und E die durch $\mathcal{S}_G$ beschriebene Eigenschaft.

Definition 12.7.2.1 (Interferenzstabil)

Knoten n heißt

1. *SUPP*-interferenzstabil für E (in Zeichen: $itf\text{-}stabil_n^{SUPP}$) gdw E kann an n nicht durch einen Verschränkungsvorgänger von n zerstört werden.
2. *VUPP*-interferenzstabil für E (in Zeichen: $itf\text{-}stabil_n^{VUPP}$) gdw E kann an n nicht durch einen Verschränkungsvorgänger von n erzeugt werden.

Interferenzlabilität

...analog (als Negation von Interferenzstabilität):

Definition 12.7.2.2 (Interferenzlabil)

Knoten n heißt

1. *SUPP*-interferenzlabil für E (in Zeichen: $itf\text{-}labil_n^{SUPP}$) gdw n ist nicht *SUPP*-interferenzstabil für E.
2. *VUPP*-interferenzlabil für E (in Zeichen: $itf\text{-}labil_n^{VUPP}$) gdw n ist nicht *VUPP*-interferenzstabil für E.

Interferenzwirksam

...dual zur Interferenzlabilität von Knoten nennen wir diejenigen Komponenten von Parallelanweisungen interferenzwirksam, die eine Anweisung enthalten, die die Knoten ihrer verschränkten Geschwistergraphen interferenzlabil machen.

Definition 12.7.2.3 (Interferenzwirksam)

Ein Komponentenflussgraph $G \in \mathcal{G}_{\mathcal{C}}(G^*)$ einer Parallelanweisung heißt

1. *SUPP*-interferenzwirksam für E (in Zeichen: $\mathit{itf\text{-}wirksam}_G^{SUPP}$) gdw G enthält einen Knoten n mit $[\![n]\!]^* = Cst_{\mathbf{falsch}}$.
2. *VUPP*-interferenzwirksam für E (in Zeichen: $\mathit{itf\text{-}wirksam}_G^{VUPP}$) gdw G enthält einen Knoten n mit $[\![n]\!]^* = Cst_{\mathbf{wahr}}$.

Interferenzlemma

Als Folgerung aus Hauptlemma 12.6.1.2, wonach für unidirektionale Bitvektoranalysen der Effekt eines ganzen Pfads durch den Effekt einer einzigen Anweisung bestimmt ist, erhalten wir:

Lemma 12.7.2.4 (Interferenzlemma)

Für alle $\forall\, n \in N$ gilt:

1. Interfererenzstabil
   - 1.1 $itf\text{-}stabil_n^{SUPP} \iff$
     $$\forall\, m \in Pred_G^{verschrVorg}(n).\ [\![\, m \,]\!] \in \{Cst_{\mathbf{wahr}}, Id_{\mathbb{B}}\}$$
   - 1.2 $itf\text{-}stabil_n^{VUPP} \iff$
     $$\forall\, m \in Pred_G^{verschrVorg}(n).\ [\![\, m \,]\!] \in \{Cst_{\mathbf{falsch}}, Id_{\mathbb{B}}\}$$
2. Interferenzlabil
   - 2.1 $itf\text{-}labil_n^{SUPP} \iff$
     $$\exists\, m \in Pred_G^{verschrVorg}(n).\ \ [\![\, m \,]\!] = Cst_{\mathbf{falsch}}$$
   - 2.2 $itf\text{-}labil_n^{VUPP} \iff$
     $$\exists\, m \in Pred_G^{verschrVorg}(n).\ [\![\, m \,]\!] = Cst_{\mathbf{wahr}}$$

Kapitel 12.7.3

Parallele maximale Fixpunktsemantik unidirektionaler Bitvektoranalysen

Das $PMaxFP_{UBV}$-Gleichungssystem

...sei $\mathcal{S}_G = (\widehat{\mathcal{F}_{\mathbb{IB}}}, [\![\ ]\!])$ die PDFA-Spezifikation eines unidirektionalen Bitvektorproblems.

Gleichungssystem 12.7.3.1 ($PMaxFP_{UBV}$-Gleich.sys.)

$$[\![n]\!]_{\sqcap} = \begin{cases} Id_{\mathbb{IB}} & \text{falls } n = \mathbf{s} \\ [\![pfg(n)]\!]_{\sqcap} \circ [\![start(pfg(n))]\!]_{\sqcap} \sqcap Cst_{itf\text{-}stabil_n^{SUPP}} & \text{falls } n \in N_X^* \\ \bigsqcap \{ [\![m]\!] \circ [\![m]\!]_{\sqcap} \mid m \in pred_G(n) \} \sqcap Cst_{itf\text{-}stabil_n^{SUPP}} & \text{sonst} \end{cases}$$

Beachte: Gleichungssystem 12.7.3.1 ist die natürliche Erweiterung von Gleichungssystems 12.7.1.1 auf parallele Programme.

Die $PMaxFP_{UBV}$-Semantik

Sei (überladen bezeichnet) $[\![\ ]\!]_{\sqcap} : N \to \mathcal{F}_{\mathbb{B}}$ die eindeutig bestimmte größte Lösung von Gleichungssystem 12.7.3.1. Dann legen wir als $PMaxFP_{UBV}$-Semantik fest:

Definition 12.7.3.2 ($PMaxFP_{UBV}$ -Semantik)

Die $PMaxFP_{UBV}$-Semantik für das von $\mathcal{S}_G$ spezifizierte unidirektionale Bitvektoranalyseproblem ist definiert durch:

$$[\![\ ]\!]_{PMaxFP_{UBV}} : N \to \mathcal{F}_{\mathbb{B}}$$

$$[\![\ ]\!]_{PMaxFP_{UBV}} =_{df} \lambda\, n \in N.\ [\![\, n\,]\!]_{\sqcap}$$

Kapitel 12.7.4

Parallele minimale Fixpunktsemantik

Das $PMinFP_{UBV}$-Gleichungssystem

...sei $\mathcal{S}_G = (\widehat{\mathcal{F}_{\mathbb{IB}}}, [\![\ ]\!])$ die PDFA-Spezifikation eines unidirektionalen Bitvektorproblems.

Gleichungssystem 12.7.4.1 ($PMinFP_{UBV}$-Gleich.sys.)

$$[\![n]\!]_{\sqcup} =$$

$$\begin{cases} Id_{\mathbb{IB}} & \text{falls } n = \mathbf{s} \\ [\![pfg(n)]\!]_{\sqcup} \circ [\![start(pfg(n))]\!]_{\sqcup} \sqcup Cst_{itf\text{-}stabil_n^{VUPP}} & \\ & \text{falls } n \in N_X^* \\ \bigsqcup \{ [\![m]\!] \circ [\![m]\!]_{\sqcup} \mid m \in pred_G(n) \} \sqcup Cst_{itf\text{-}stabil_n^{VUPP}} & \text{sonst} \end{cases}$$

Die $PMinFP_{UBV}$-Semantik

Sei (überladen bezeichnet) $[\![\ ]\!]_{\sqcup} : N \to \mathcal{F}_{\mathbb{B}}$ die eindeutig bestimmte kleinste Lösung von Gleichungssystem 12.7.4.1. Dann legen wir als $PMinFP_{UBV}$-Semantik fest:

Definition 12.7.4.2 ($PMinFP_{UBV}$ -Semantik)

Die $PMinFP_{UBV}$-Semantik für das von $\mathcal{S}_G$ spezifizierte unidirektionale Bitvektorproblem ist definiert durch:

$$[\![\ ]\!]_{PMinFP_{UBV}} : N \to \mathcal{F}_{\mathbb{B}}$$

$$[\![\ ]\!]_{PMinFP_{UBV}} = \lambda n \in N.\ [\![n]\!]_{\sqcup}$$

Kapitel 12.8

Entscheidbarkeit der $PMaxFP_{UBV}$- und $PMinFP_{UBV}$-Semantik

Es reicht

...ein Verfahren zur Berechnung der Funktionen $[\![n]\!]_{\sqcap}$ und $[\![n]\!]_{\sqcup}$, $n \in N$, anzugeben. Liegen diese Funktionen vor, kann die Fixpunkberechnung wie im sequentiellen Fall fortgeführt werden.

Kernstück dieser Berechnung ist die Synchronisationsbehandlung am Ende paralleler Anweisungen...

Synchronisation paralleler Komponenten (1)

...erfolgt in einem 4-schrittigen Verfahren:

1. Fortschritts- und Abbruchssteuerung: Enthält G keine parallelen Anweisungen, terminiere; anderenfalls fahre mit Schritt 2 fort.
2. Vorbereitung des Synchronisationsschritts: Berechne für alle maximalen Teilgraphen G' parallelanweisungsfreier (d.h. minimaler) Graphen aus $\mathcal{G}_{\mathcal{P}}(G)$ die *SUPP*- bzw. *VUPP*-Semantik $[\![G']\!]_{SUPP}$ bzw. $[\![G']\!]_{VUPP}$ dieser (rein sequentiellen) Graphen mittels Algorithmus 12.7.1.6.

Synchronisation paralleler Komponenten (2)

3. Synchronisationsschritt: Berechne für alle innersten parallelen Anweisungen $\bar{G}$ von G die *SUPP*- bzw. *VUPP*-Semantik gemäß:

$$[\![\bar{G}]\!]_{SUPP} = \begin{cases} Cst_{\mathbf{falsch}} & \text{falls } \exists\, G' \in \mathcal{G}_{\mathcal{C}}(\bar{G}).\ [\![end(G')]\!] = Cst_{\mathbf{falsch}} \\ Id_{\mathbb{IB}} & \text{falls } \forall\, G' \in \mathcal{G}_{\mathcal{C}}(\bar{G}).\ [\![end(G')]\!] = Id_{\mathbb{IB}} \\ Cst_{\mathbf{wahr}} & \text{sonst} \end{cases}$$

$$[\![\bar{G}]\!]_{VUPP} = \begin{cases} Cst_{\mathbf{wahr}} & \text{falls } \exists\, G' \in \mathcal{G}_{\mathcal{C}}(\bar{G}).\ [\![end(G')]\!] = Cst_{\mathbf{wahr}} \\ Id_{\mathbb{IB}} & \text{falls } \forall\, G' \in \mathcal{G}_{\mathcal{C}}(\bar{G}).\ [\![end(G')]\!] = Id_{\mathbb{IB}} \\ Cst_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

Synchronisation paralleler Komponenten (3)

4. Vorbereitung der nächsten Iterationsstufe: Ersetze alle innersten parallelen Anweisungen

$$\bar{G} = (\bar{N}, \bar{E}, \bar{\mathbf{s}}, \bar{\mathbf{e}})$$

in G durch die formal sequentialisierten Graphen

$$\bar{\bar{G}} = (\{\bar{\mathbf{s}}, \bar{\mathbf{e}}\}, \{(\bar{\mathbf{s}}, \bar{\mathbf{e}})\}, \bar{\mathbf{s}}, \bar{\mathbf{e}})$$

mit lokaler *SUPP*- bzw. *VUPP*-Semantik:

- $[\![\, \bar{\mathbf{s}} \,]\!]_{SUPP} = Id_{\mathbb{B}} \sqcap \bigsqcap \{ [\![\, n \,]\!] \mid n \in \bar{N} \}$, $[\![\, \bar{\mathbf{e}} \,]\!] = [\![\, \bar{G} \,]\!]_{SUPP}$
- $[\![\, \bar{\mathbf{s}} \,]\!]_{VUPP} = Id_{\mathbb{B}} \sqcup \bigsqcup \{ [\![\, n \,]\!] \mid n \in \bar{N} \}$, $[\![\, \bar{\mathbf{e}} \,]\!] = [\![\, \bar{G} \,]\!]_{VUPP}$

Fahre mit Schritt 1 fort.

Kapitel 12.9

$PMaxFP_{UBV}$- und $PMinFP_{UBV}$-Semantik als zueinander duale berechenbare Lösungen paralleler unidirektionaler Bitvektorprobleme

Berechenbare DFA-Problemlösungen

... die Entscheidbarkeit der Fixpunktsemantiken für parallele unidirektionale Bitvektorprobleme legt die Festlegung nahe:

Definition 12.9.1 (Berechenb. Lsg. eines PDFA-P.)

Die $PMaxFP_{UBV}$- und $PMinFP_{UBV}$-Semantik eines parallelen Flussgraphen definieren zwei zueinander duale berechenbare Lösungen eines parallelen unidirektionalen Bitvektorproblems, seine sog.:

1. $PMaxFP_{UBV}$-Lösung
2. $PMinFP_{UBV}$-Lösung

Wieder stellt sich die Frage nach dem Verhältnis, in dem spezifizierenden und berechenbaren Lösungen dieser Probleme zueinander stehen...

Kapitel 12.10

Koinzidenz für parallele unidirektionale Bitvektorprobleme

Hauptergebnis: Korrektheit und Vollständigkeit

...für unidirektionale Bitvektorprobleme stimmen die parallelen

- operationellen Schnitt- und Vereinigung-über-alle-Pfade-Semantiken

mit ihren

- denotationellen Fixpunktgegenstücken

überein: Paralleles Bitvektorkoinzidenztheorem 12.10.3.

Beweisgrundlage dafür: Synchronisationskorrektheit!

Synchronisationskorrektheit (1)

...Basis für die Korrektheit des Synchronisationsschritts ist Lemma 12.10.1 für innerste parallele Anweisungen, das wie die Korrektheit der Interferenzbehandlung eine Konsequenz aus Hauptlemma 12.6.1.2 ist, wonach für unidirektionale Bitvektoranalysen der Effekt eines ganzen Pfads durch den Effekt einer einzigen Anweisung bestimmt ist.

Das heißt (ausgeführt in Lemma 12.10.1):

- Der Effekt eines vollständigen Pfads durch den Graphen einer Parallelanweisung ist durch den Effekt der Projektion des Pfads auf die Knoten desjenigen Komponentengraphen gegeben, der diese effektbestimmende Anweisung enthält.
- Der Effekt einer Parallelanweisung ergibt sich daher aus der Effektkomposition der komponentenlokalen Pfade.

Synchronisationskorrektheit (2)

Lemma 12.10.1 (Synchronisationskorrekth.: Basis)

Ist $G \in \mathcal{G}_{\mathcal{P}}(G^*)$ eine Parallelanweisung mit ausschließlich rein sequentiellen Komponenten $G_1,\ldots,G_k$, so gilt:

1. Die *SUPP*-Semantik von G ist gegeben durch:

$$[\![end(G)]\!]_{SUPP} = [\![G]\!]_{SUPP}$$
$$\begin{cases} Cst_{\mathbf{falsch}} & \text{if } \exists\, 1 \leq i \leq k.\ [\![end(G_i)]\!]_{SUPP} = Cst_{\mathbf{falsch}} \\ Id_{\mathbb{B}} & \text{if } \forall\, 1 \leq i \leq k.\ [\![end(G_i)]\!]_{SUPP} = Id_{\mathbb{B}} \\ Cst_{\mathbf{wahr}} & \text{otherwise.} \end{cases}$$

2. Die *VUPP*-Semantik von G ist gegeben durch:

$$[\![end(G)]\!]_{VUPP} = [\![G]\!]_{VUPP}$$
$$\begin{cases} Cst_{\mathbf{wahr}} & \text{if } \exists\, 1 \leq i \leq k.\ [\![end(G_i)]\!]_{VUPP} = Cst_{\mathbf{wahr}} \\ Id_{\mathbb{B}} & \text{if } \forall\, 1 \leq i \leq k.\ [\![end(G_i)]\!]_{VUPP} = Id_{\mathbb{B}} \\ Cst_{\mathbf{falsch}} & \text{otherwise.} \end{cases}$$

Synchronisationskorrektheit (3)

...als induktive Erweiterung (der funktionalen Variante) des sequentiellen Koinzidenztheorems 8.10.2 für unidirektionale Bitvektorprobleme erhalten wir zusammen mit Lemma 12.10.1 die Korrektheit der hierarchisch erfolgenden Synchronisation(sschritte):

Hierarchisches Koinzidenztheorem 12.10.2

Sei $G' \in \mathcal{G}_{\mathcal{P}}(G)$ der Graph einer Parallelanweisung und sei $\mathcal{S}_G = (G, [\![\ ]\!])$ eine PDFA-Spezifikation mit $[\![\ ]\!] : N \to \mathcal{F}_{\mathbb{B}}$ lokales Semantikfunktional eines unidirektionalen Bitvektorproblems. Dann gilt:

1. $[\![\, end(G') \,]\!]_{SUPP} = [\![\, G' \,]\!]_{SUPP}$
2. $[\![\, end(G') \,]\!]_{VUPP} = [\![\, G' \,]\!]_{VUPP}$

Paralleles Bitvektorkoinzidenztheorem

...als Hauptresultat. Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein paralleler Flussgraph und $\mathcal{S}_G =_{df} (\widehat{\mathcal{F}_{\mathbb{B}}}, [\![\ ]\!])$ die DFA-Spezifikation eines unidirektionalen Bitvektorproblems für G:

Paralleles Bitvektorkoinzidenztheorem 12.10.3

Für unidirektionale Bitvektoprobleme stimmen

1. $SUPP$- und $PMaxFP_{UBV}$-Semantik überein:

$$\forall\, n \in N^*.\ [\![\, n \,]\!]_{SUPP} \ = \ [\![\, n \,]\!]_{PMaxFP_{UBV}}$$

2. $VUPP$- und $PMinFP_{UBV}$-Semantik überein:

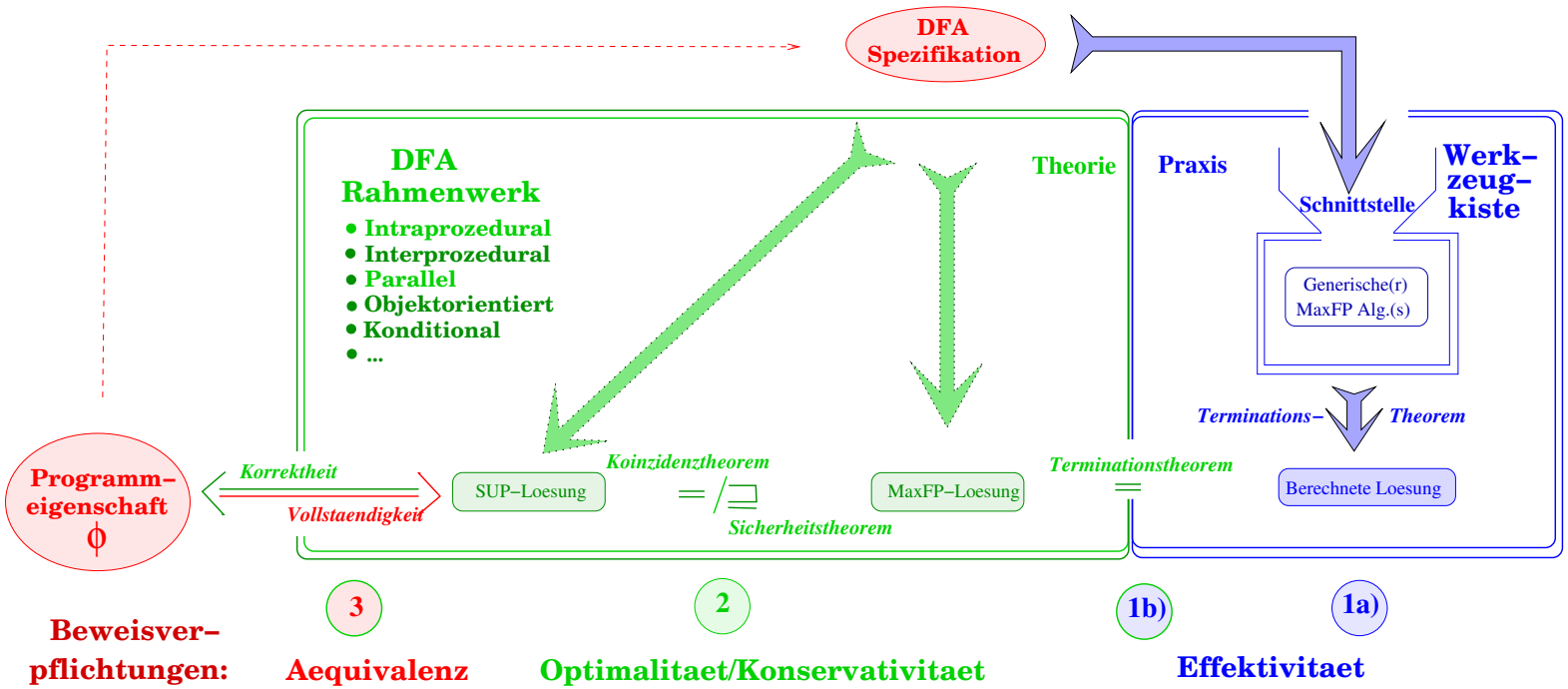
$$\forall\, n \in N^*.\ [\![\, n \,]\!]_{VUPP} \ = \ [\![\, n \,]\!]_{PMinFP_{UBV}}$$

Kapitel 12.11

Parallele Datenflussanalyse in Rahmenwerk- und Werkzeugkistensicht

Parallele Datenflussanalyse

...für unidirektionale Bitvektorprobleme in Rahmenwerk- und Werkzeugkistensicht:



Kapitel 12.12

Anwendungen

Verfügbarkeit v. Ausdrücken, tote Zuweisungen

Die lokalen Semantikfunktionen der unidirektionalen Bitvektorprobleme verfügbarer Ausdrücke (Vorwärtsproblem) und toter Zuweisungen (Rückwärtsproblem) auf dem Verband der Wahrheitswerte können wie folgt spezifiziert werden:

Verfügbarkeit von Ausdruck t (vgl. Kap. 8.13.1):

$$[\![\ ]\!]^t_{av} =_{df} \lambda n. \begin{cases} Const_{\mathbf{wahr}} & \text{falls } Transp^t_n \wedge Comp^t_n \\ Id_{\mathbb{B}} & \text{falls } Transp^t_n \wedge \neg Comp^t_n \\ Const_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

Tote Zuweisungen an Variable x (vgl. Kap. 14.3):

$$[\![\ ]\!]^x_{dead} =_{df} \lambda n. \begin{cases} Const_{\mathbf{wahr}} & \text{falls } \neg Used^x_n \wedge Mod^x_n \\ Id_{\mathbb{B}} & \text{falls } \neg(Used^x_n \vee Mod^x_n \\ Const_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

Globalisierung der lokalen Semantiken

...die lokalen Semantiken $[\![n]\!]^t_{av}$ und $[\![n]\!]^x_{dead}$ können durch den Übergang auf Bitvektoren auf die Mengen aller Ausdrücke T bzw. Variablen V eines Programms ausgedehnt werden:

$$[\![\cdot]\!]^T_{av} \text{ und } [\![\cdot]\!]^V_{dead}$$

und im Sinn der

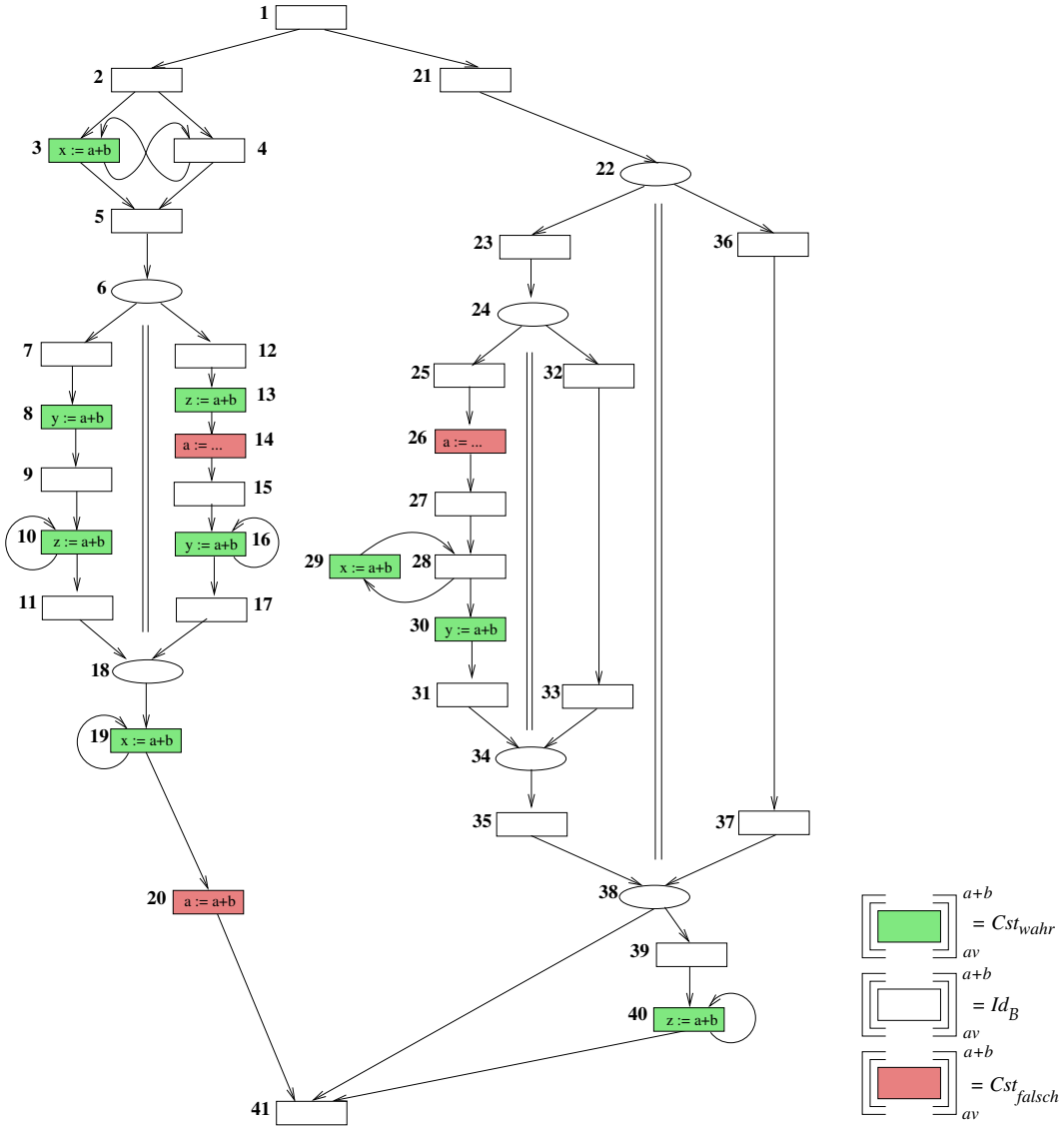
- *SUPP* - bzw. VUPP-Semantik

globalisiert werden, was zur Berechnung folgender Eigenschaften führt:

- *SUPP* -Globalisierung: Total verfügbare Ausdrücke, total tote Zuweisungen.
- *VUPP* -Globalisierung: Partiell verfügbare Ausdrücke, partiell tote Zuweisungen.

Lokale Semantik: Verfügbarkeitsanalyse

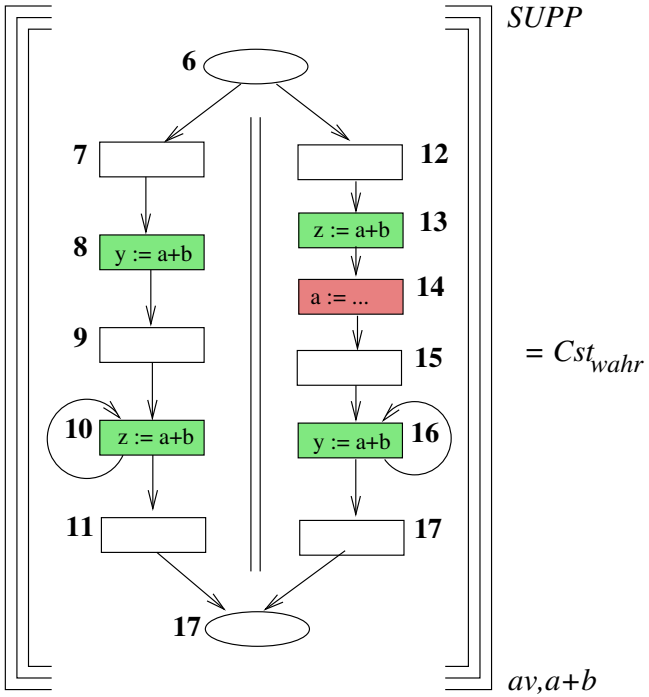
...für Ausdruck a+b:



SUPP -Semantikglobalisierung: 1. Iteration

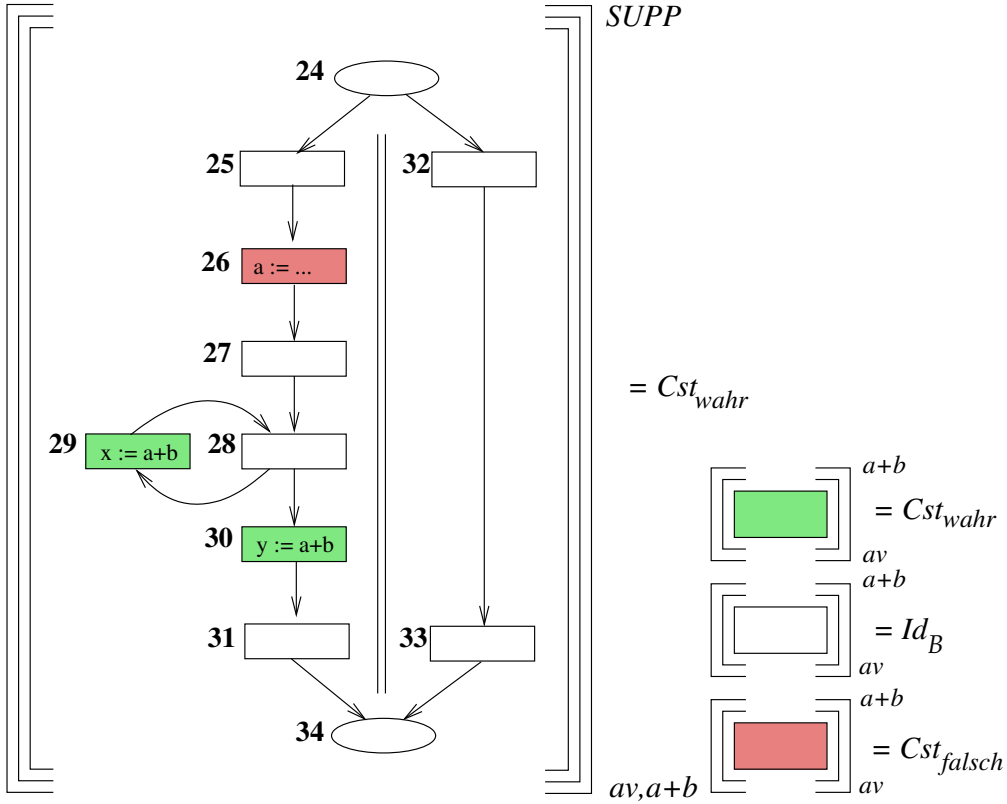
...entsprechend des 4-stufigen Verfahrens aus Kapitel 12.8:

$G_{01} = G_{01_{seq}}$



$\text{if–wirkmaechtig}^{SUPP}_{G_{01}} = \text{wahr}$

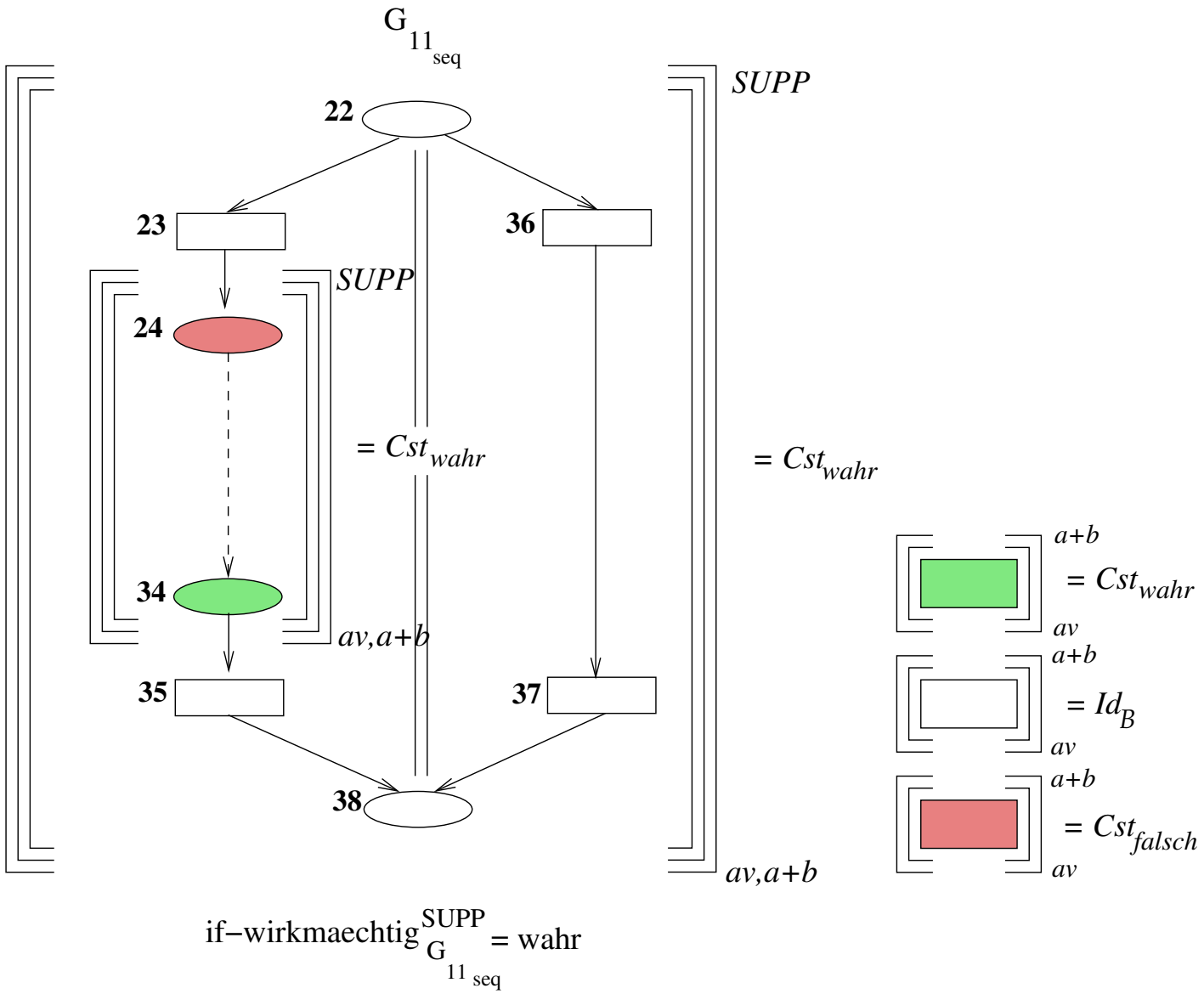
$G_{02} = G_{02_{seq}}$



$\text{if–wirkmaechtig}^{SUPP}_{G_{02}} = \text{wahr}$

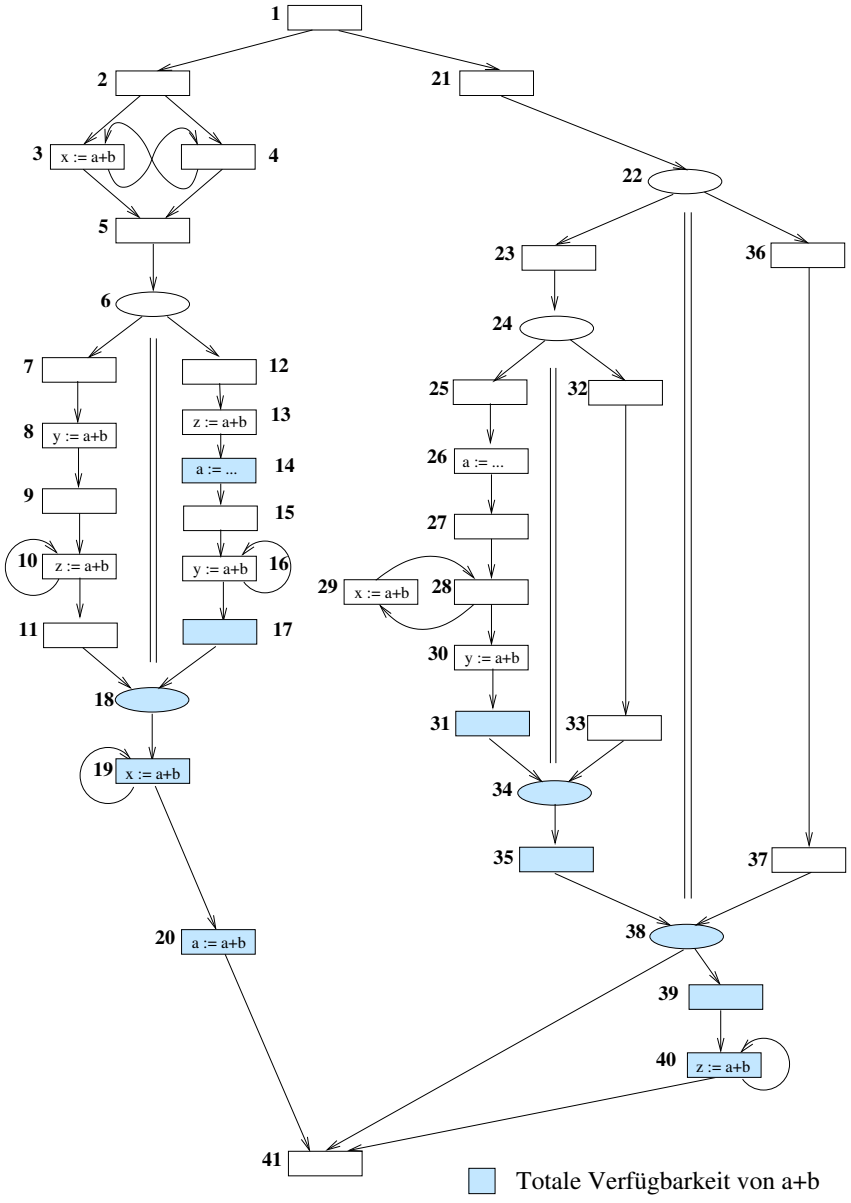
SUPP -Semantikglobalisierung: 2. Iteration

...entsprechend des 4-stufigen Verfahrens aus Kapitel 12.8:



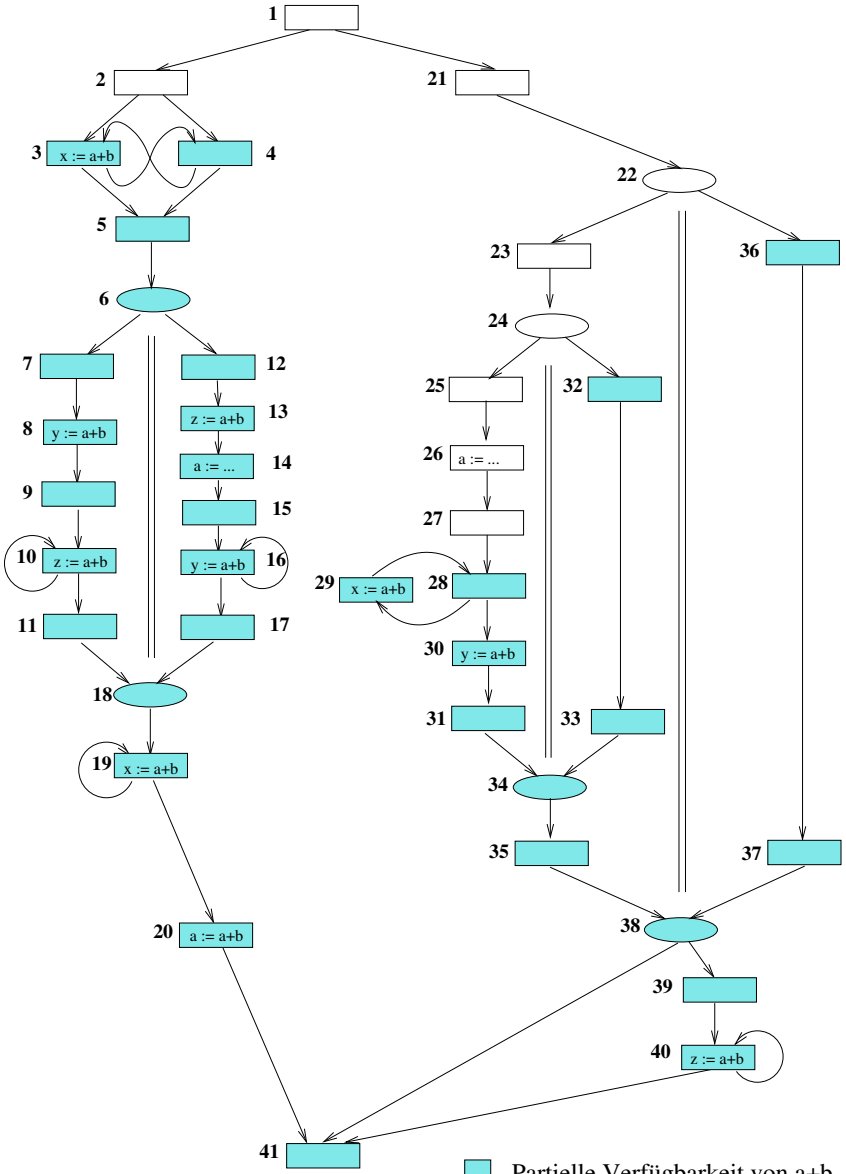
Ergebnis der totalen Verfügbarkeitsanalyse

...für a+b an Knoteneingängen:



Ergebnis der partiellen Verfügbarkeitsanalyse

...für a+b an Knoteneingängen:



Übungsaufgabe 12.12.1

Partielle Verfügbarkeitsanalyse für a+b:

1. Gib die *VUPP*-Semantikglobalisierung für das durchgehende Beispiel nach dem 1. und 2. Iterationsschritt des 4-stufigen Verfahrens aus Kapitel 12.8 an.

Tote-Zuweisungenanalyse für a:

2. Gib die *SUPP*- und *VUPP*-Semantikglobalisierungen für das durchgehende Beispiel nach dem 1. und 2. Iterationsschritt des 4-stufigen Verfahrens aus Kapitel 12.8 an.
3. An welchen Knotenausgängen ist Variable a total tot, an welchen partiell tot? Gib die entsprechend markierten Graphen nach dem Vorbild der totalen und partiellen Verfügbarkeitsanalyse für a+b an.

Beachte: Die Tote-Zuweisungenanalyse ist anders als die Verfügbarkeitsanalyse eine Rückwärtsanalyse.

Kapitel 12.13

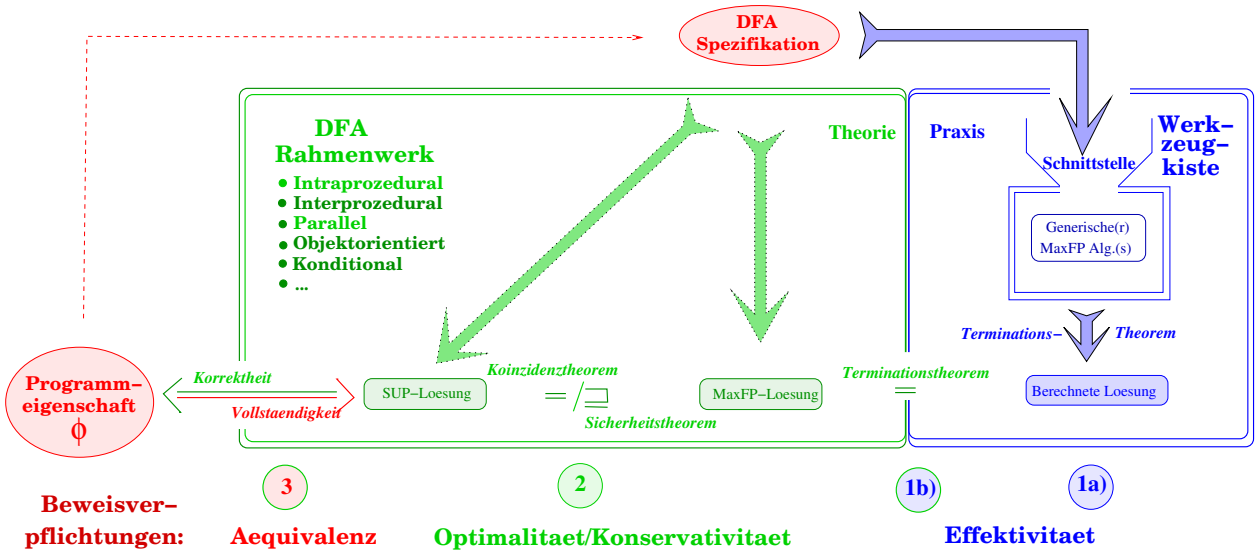
Zusammenfassung

Zusammenfassung

...mit dem Parallelen Koinzidenztheorem 12.10.3 für unidirektionale Bitvektorverfahren ist das Versprechen aus Kapitel 8.14 eingelöst, dass die

– einheitliche Rahmen- und Werkzeugkistensicht für DFA

über den Fall sequentieller intraprozeduraler DFA hinaus (s.a. LVA 185.A04 Optimierende Übersetzer) erreichbar ist:



Nur Mut!

...maßgeblich für die erfolgreiche Ausdehnung auf den Fall paralleler DFA ist die Beschränkung auf

- unidirektionale Bitvektorverfahren

für die sich die Probleme von Interferenz und Synchronisation paralleler Komponenten unter vollständiger Vermeidung des Zustandsexplosionsproblems ohne merklichen Mehraufwand im Vergleich zur DFA sequentieller Programme meistern lassen.

Verantwortlich dafür ist, dass $\mathcal{F}_{\mathsf{IB}}$ lediglich 3 Funktionen umfasst, deren äußere Semantik intuitiv oft wie folgt beschrieben werden kann:

- *Mod*: *M*odifikation
- *Use*: Benutzung (engl. *U*se)
- *Transp*: *T*ransparenz

Deshalb: Nur MUT! MUT zur DFA paralleler Programme!

Kapitel 12.14

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (1)

Jyh-Herng Chow, William L. Harrison. *Compile Time Analysis of Parallel Programs that share Memory*. In Conference Record of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'92), 130-141, 1992.

Jyh-Herng Chow, William L. Harrison. *State Space Reduction in Abstract Interpretation of Parallel Programs*. In Proceedings of the International Conference on Computer Languages (ICCL'94), 277-288, 1994.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (2)

Patrick Cousot, Radhia Cousot. *Invariance Proof Methods and Analysis Techniques for Parallel Programs*. In Automatic Program Construction Techniques, A. W. Biermann, G. Guiho, Y. Kodratoff (Hrsg.), Macmillan Publishing Company, Kapitel 12, 243-271, 1984.

Matthew B. Dwyer, Lori A. Clarke. *Data Flow Analysis for Verifying Properties of Concurrent Programs*. In Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering (SFSE'94), Software Engineering Notes 19(5):62-75, 1994.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (3)


Matthew B. Dwyer, Lori A. Clarke, Jamieson M. Cobleigh, Gleb Naumovich. *Flow Analysis for Verifying Properties of Concurrent Software Systems*. ACM Transactions on Software Engineering Methodology 13(4):359-430, 2004.


Jeanne Ferrante, Dirk Grunwald, Harini Srinivasan. *Compile-time Analysis and Optimization of Explicitly Parallel Programs*. Parallel Algorithms and Applications 12(1-3):21-56, 1997.


Dirk Grunwald, Harini Srinivasan. *Data Flow Equations for Explicitely Parallel Programs*. In Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'93), ACM SIGPLAN Notices 28(7):159-168, 1993.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (4)


Jens Knoop. *Parallel Constant Propagation*. In Proceedings of the 4th European Conference on Parallel Processing (Europar'98), Springer-V., LNCS 1470, 445-455, 1998.


Jens Knoop. *Demand-driven Analysis of Explicitly Parallel Programs: An Approach based on Reverse Data-Flow Analysis*. In Proceedings of the 9th International Workshop on Compilers for Parallel Computers (CPC 2001), 151-162, 2001.


Jens Knoop, Bernhard Steffen. *Code Motion for Explicitly Parallel Programs*. In Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99), ACM SIGPLAN Notices 34(8):13-24, 1999.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (5)

Jens Knoop, Bernhard Steffen, Jürgen Vollmer. *Parallelism for Free: Bitvector Analyses $\rightarrow$ No State Explosion!* In Proceedings of the 1st International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95), Springer-V., LNCS 1019, 264-289, 1995.

Jens Knoop, Bernhard Steffen, Jürgen Vollmer. *Parallelism for Free: Efficient and Optimal Bitvector Analyses for Parallel Programs*. ACM Transactions on Programming Languages and Systems 18(3):268-299, 1996.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (6)


Samuel P. Midkiff, José E. Moreira, Marc Snir. *A Constant Propagation Algorithm for Explicitly Parallel Programs*. International Journal of Computer Science 26(5):563-589, 1998.


Samuel P. Midkiff, David A. Padua. *Issues in the Optimization of Parallel Programs*. In Proceedings of the 18th International Conference on Parallel Processing (ICPP'90), Vol. II., 105-113, 1990.


Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019. (Chapter 8, Concurrency)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (7)

Harini Srinivasan, Michael Wolfe. *Analyzing Programs with Explicit Parallelism*. In Proceedings of the 4th International Conference on Languages and Compilers for Parallel Computing (LCPC'91), Springer-V., LNCS 589, 405-419, 1991.

Jürgen Vollmer. *Data Flow Analysis of Parallel Programs*. In Proceedings of the IFIP WG 10.3 Working Conference on Parallel Architectures and Compilation Techniques (PACT'95), 168-177, 1995.

Michael Wolfe, Harini Srinivasan. *Data Structures for Optimizing Programs with Explicit Parallelism*. In Proceedings of the 1st International Conference of the Austrian Center for Parallel Computation, Springer-V., LNCS 591, 139-156, 1991.

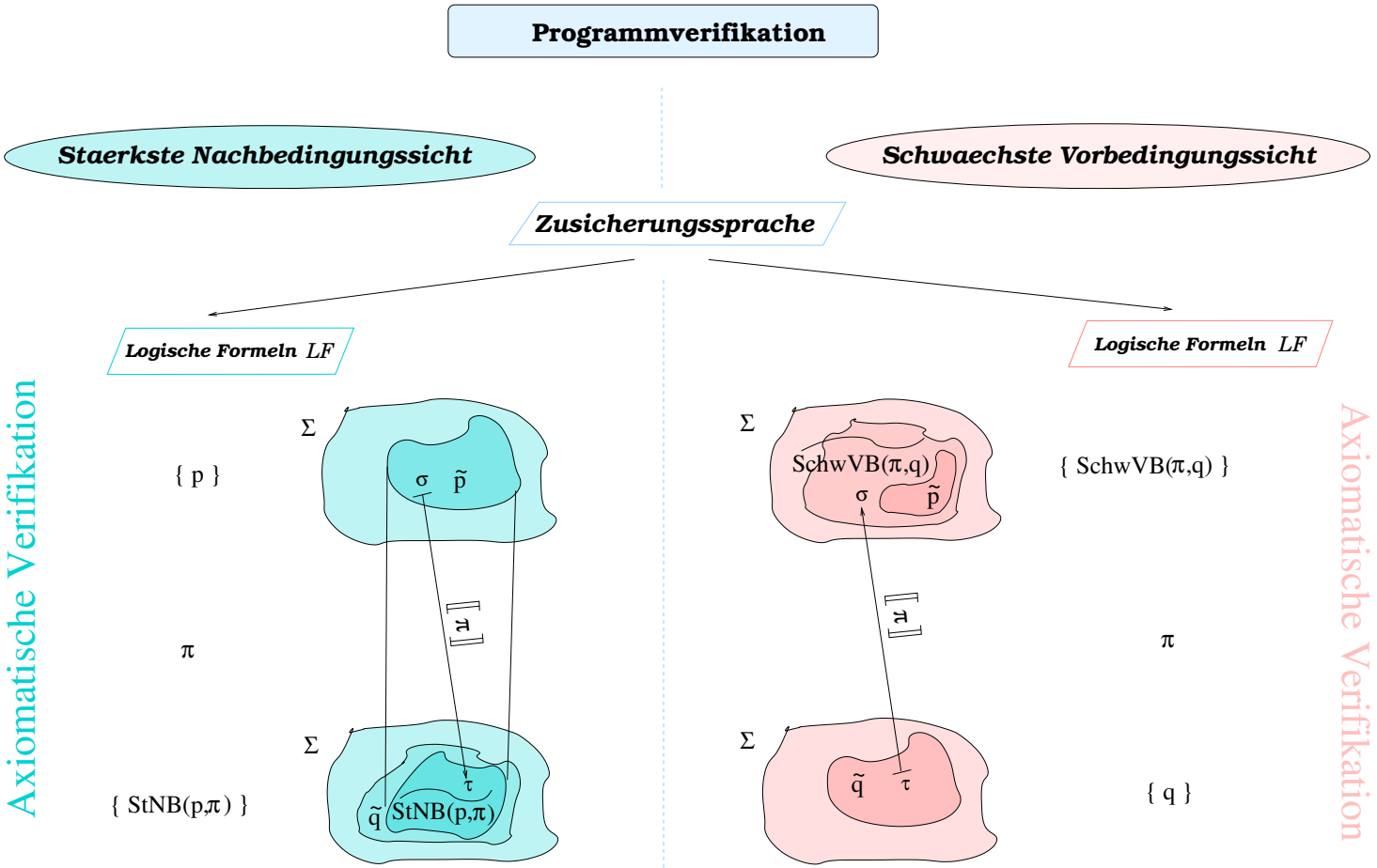
Kapitel 13

Datenflussanalyse und axiomatische Verifikation: Gegenüberstellung, Vergleich

Kapitel 13.1

Konzeptuell nach Formalismen und Problemsichten

Axiom. Verifikation in Nach-/Vorbedingungssicht



StNB(p, π) $\in$ *LF muss erfüllen:*

(1) $\models_{PV}$ { p } π { StNB(p,π) }

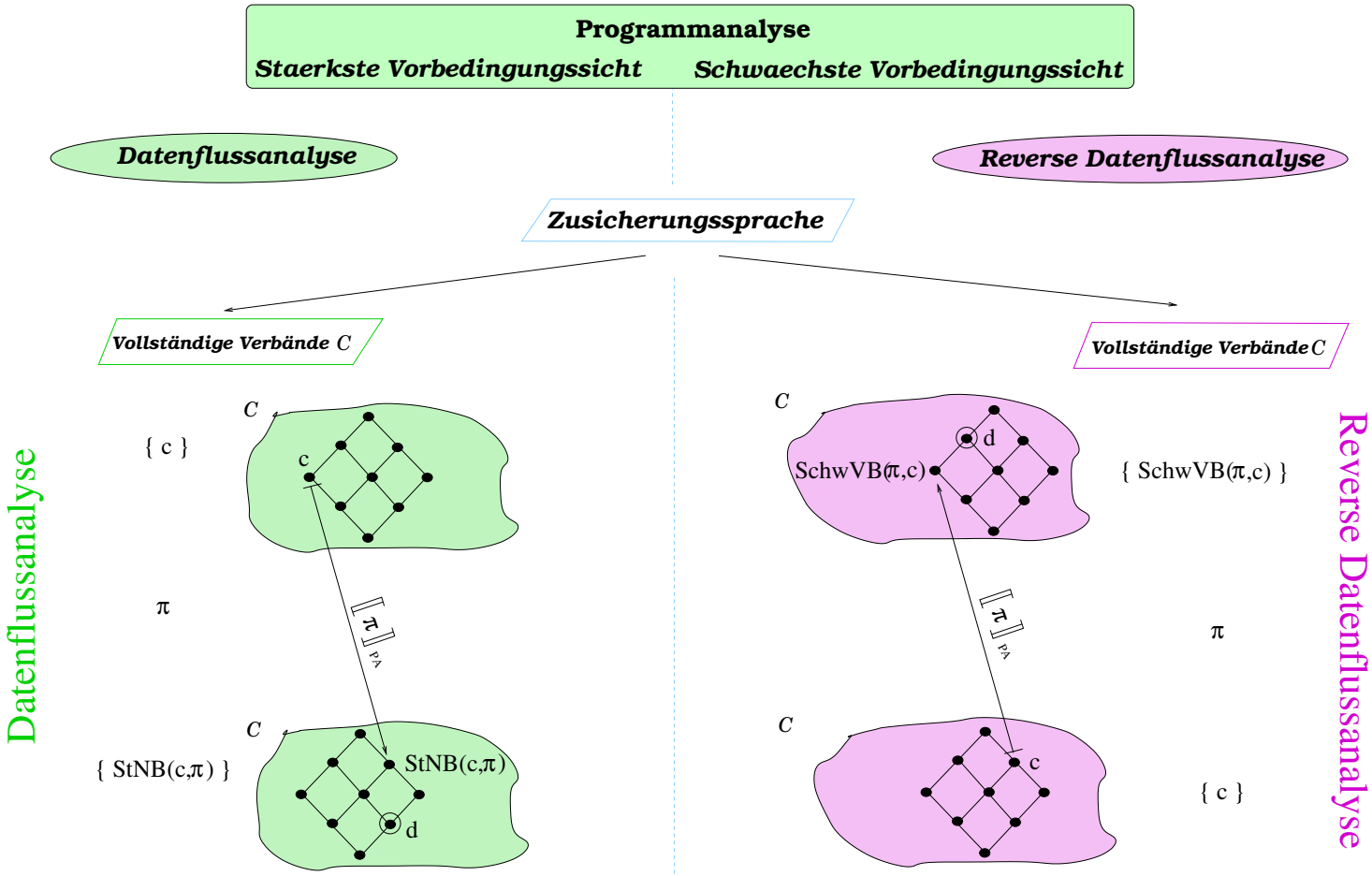
(2) $\forall$ q $\in$ *LF*. $\models_{PV}$ { p } π { q } impliziert StNB(p,π) $\Longrightarrow$ q

SchwVB(π,q) $\in$ *LF muss erfüllen:*

(1) $\models_{PV}$ { SchwVB(π,q) } π { q }

(2) $\forall$ p $\in$ *LF*. $\models_{PV}$ { p } π { q } impliziert p $\Longrightarrow$ SchwVB(π,q)

Datenflussanalyse in Nach-/Vorbedingungssicht



StNB(c,π) ∈ C *muss erfüllen:*

(1) $\models_{PA} \{ c \}\pi \{ StNB(c,\pi) \}$

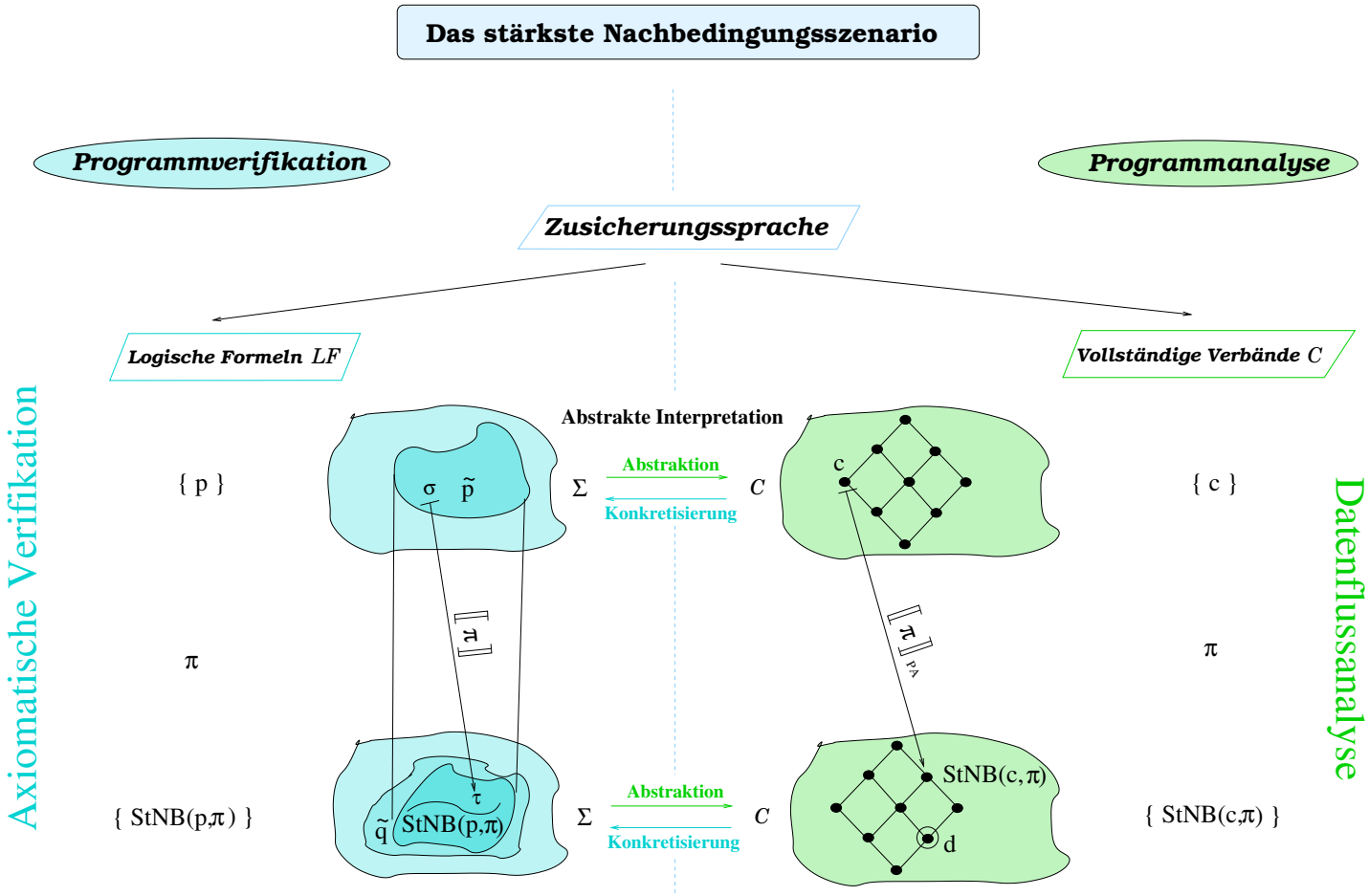
(2) $\forall d \in C. \models_{PA} \{ c \}\pi \{ d \}$ impliziert $StNB(c,\pi) \sqsupseteq d$

SchwVB(π,c) ∈ C *muss erfüllen:*

(1) $\models_{PA} \{ SchwVB(\pi,c) \}\ \pi \{ c \}$

(2) $\forall d \in C. \models_{PA} \{ d \}\pi \{ c \}$ impliziert $d \sqsupseteq SchwVB(\pi,c)$

DFA u. axiom. Verifik. in stärkster Nachbed.-S.



StNB(p, π) ∈ *LF* *muss erfüllen:*

(1) $\models_{PV} \{ p \}\pi\{ StNB(p,\pi) \}$

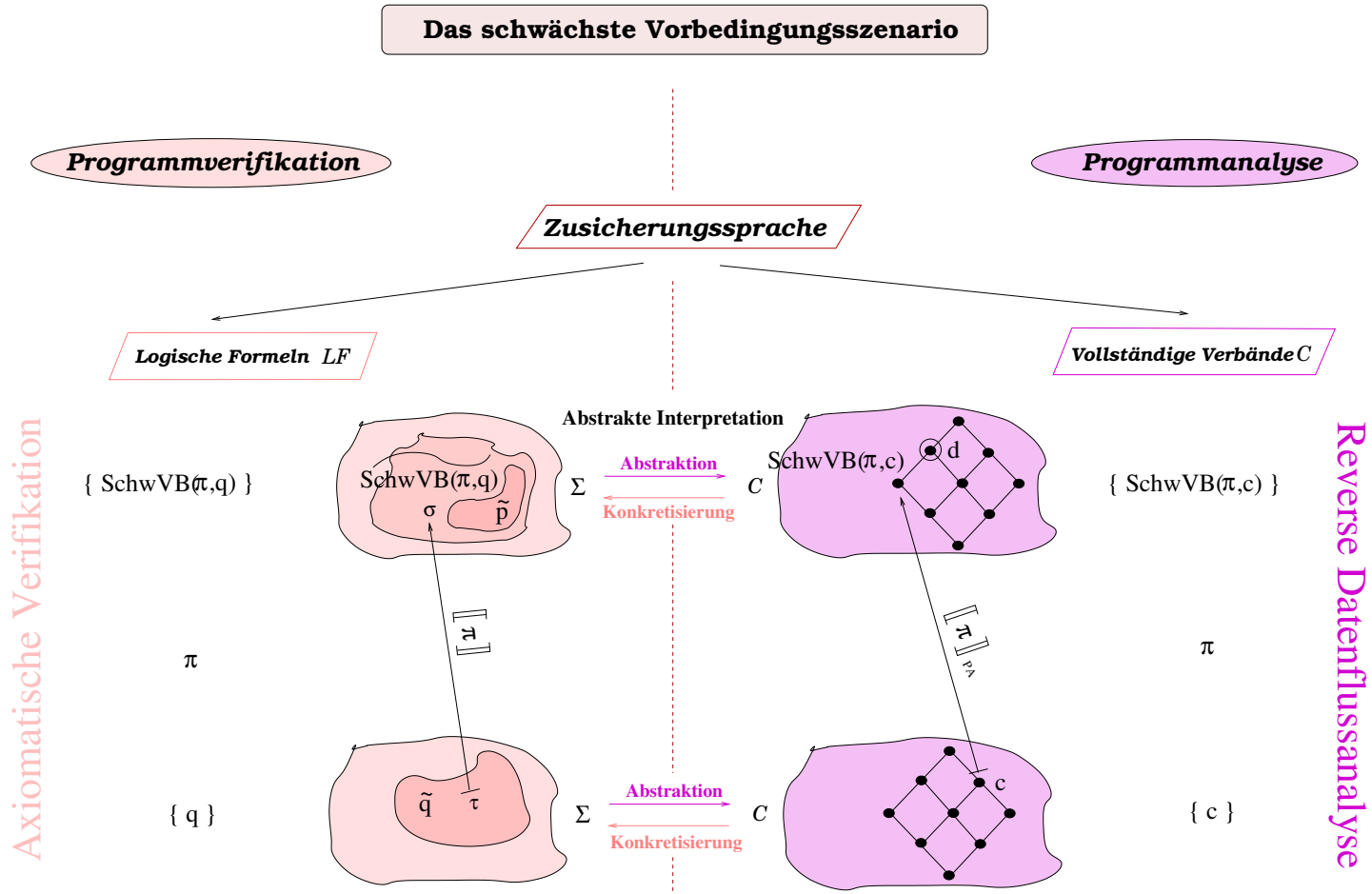
(2) $\forall\, q \in LF.\ \models_{PV} \{ p \}\pi\{ q \}$ impliziert $StNB(p,\pi) \Longrightarrow q$

StNB(c,π) ∈ *C* *muss erfüllen:*

(1) $\models_{PA} \{ c \}\pi\{ StNB(c,\pi) \}$

(2) $\forall\, d \in C.\ \models_{PA} \{ c \}\pi\{ d \}$ impliziert $StNB(c,\pi) \sqsupseteq d$

DFA u. axiom. Verifik. in schwächster Vorbed.-S.



SchwVB(π,q) ∈ *LF muss erfüllen:*

(1) $\models_{PV}$ { SchwVB(π,q) } π { q }

(2) ∀ p ∈ *LF*. $\models_{PV}$ { p } π { q } impliziert p $\Longrightarrow$ SchwVB(π,q)

SchwVB(π,c) ∈ *C muss erfüllen:*

(1) $\models_{PA}$ { SchwVB(π,c) } π { c }

(2) ∀ d ∈ *C*. $\models_{PA}$ { d } π { c } impliziert d $\sqsupseteq$ SchwVB(π,c)

Kapitel 13.2

Pragmatisch nach abgeleiteten und adressierten Problemstellungen

Abgeleitete Problemstellungen

Korrektheitstripel der Form $\{p\}\ \pi\ \{q\}$ lassen drei grundlegende Problemstellungen als Analyse-/Verifikationsaufgabe unterscheiden: Das

1. stärkste Nachbedingungsproblem (Implementierungsprob.)

$$\{p\}\ \pi/G_\pi\ \{?\}$$

2. schwächste Vorbedingungsproblem (Spezifikationsprob.)

$$\{?\}\ \pi/G_\pi\ \{q\}$$

3. Gültigkeitsproblem (Verifikationsprob.)

$$\{p\}\ \pi/G_\pi\ \{q\}\ ?$$

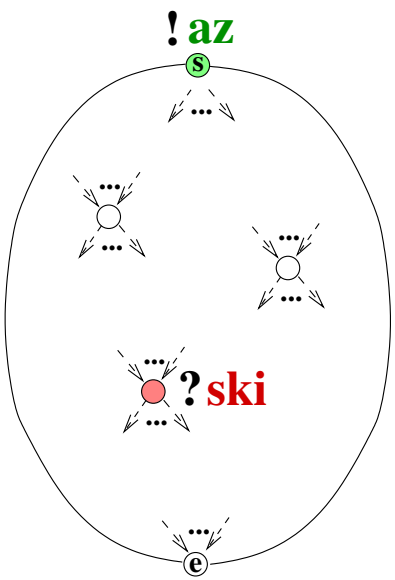
mit jeweils drei Ausprägungen für

a) Programme als Ganzes: "$\{\cdot\}\ \pi/G_\pi\ \{\cdot\}$"
b) Programmpunktauswahl: "$\{\cdot\}\ N'_{G_\pi}\ \{\cdot\}$" , $N'_{G_\pi} \subseteq N_{G_\pi}$
c) Programmpunktgesamtheit: "$\{\cdot\}\ N_{G_\pi}\ \{\cdot\}$"

Veranschaulichung der drei Grundaufgaben

...mit gegebener (in grün) und gesuchter (in rot) Information:

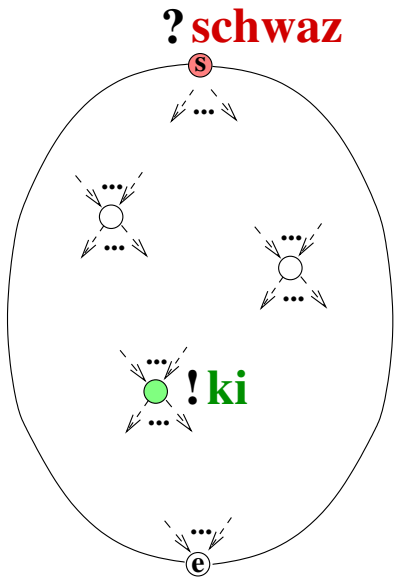
Implementierungsproblem



! **Gegeben:** Anfangszusicherung **az**

? **Gesucht:** Staerkste Knoteninformation **ski**

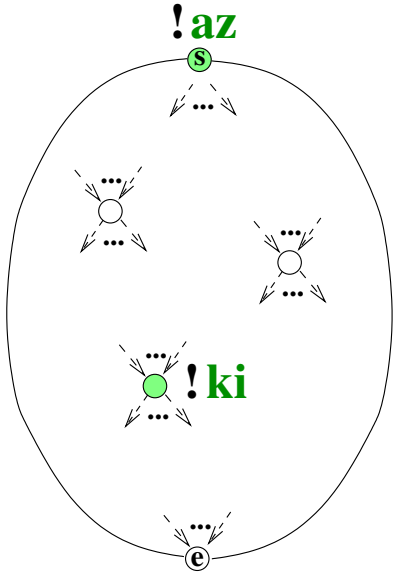
Spezifikationsproblem



! **Gegeben:** Knoteninformation **ki**

? **Gesucht:** Schwaechste Anfangszusicherung **schwaz**

Verifikationsproblem



! **Gegeben:** Anfangszusicherung **az**
Knoteninformation **ki**

? **Gesucht:** Gueltigkeit **ki** bzgl. **az**

Zusammengefasst als Problemmatrix

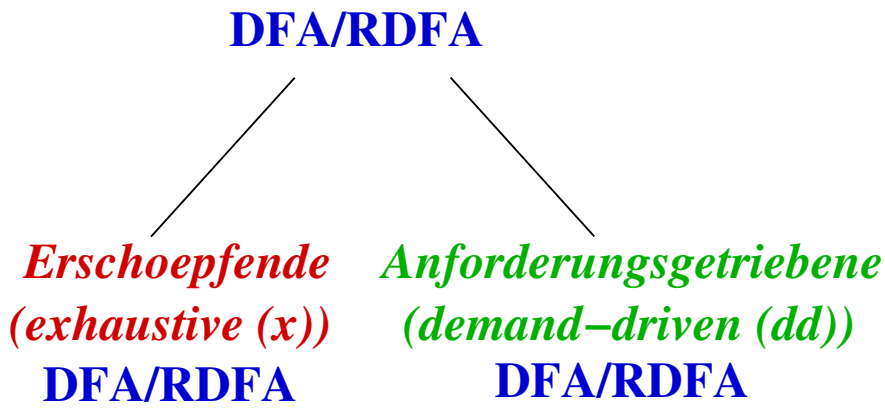
...ergeben sich daraus neun Analyse- und Verifikationsprobleme, von denen einige unmittelbar von axiomatischer Verifikation, Datenfluss- und reverser Datenflussanalyse addressiert werden:

	a) Ges.prog. π / G_π	b) Punktauswahl $N'_{G_\pi} \subseteq N_{G_\pi}$	c) Punktges. N_{G_π}
1. $\{p\}\ \pi\ \{?\}$			DFA
2. $\{?\}\ \pi\ \{q\}$			RDFA
3. $\{p\}\ \pi\ \{q\}$?	Axiom. Verif.		

Eine genauere Betrachtung und Auffächerung von DFA und RDFA ändert das Bild einer 'schwach besetzten' Matrix...

Die Unterscheidung

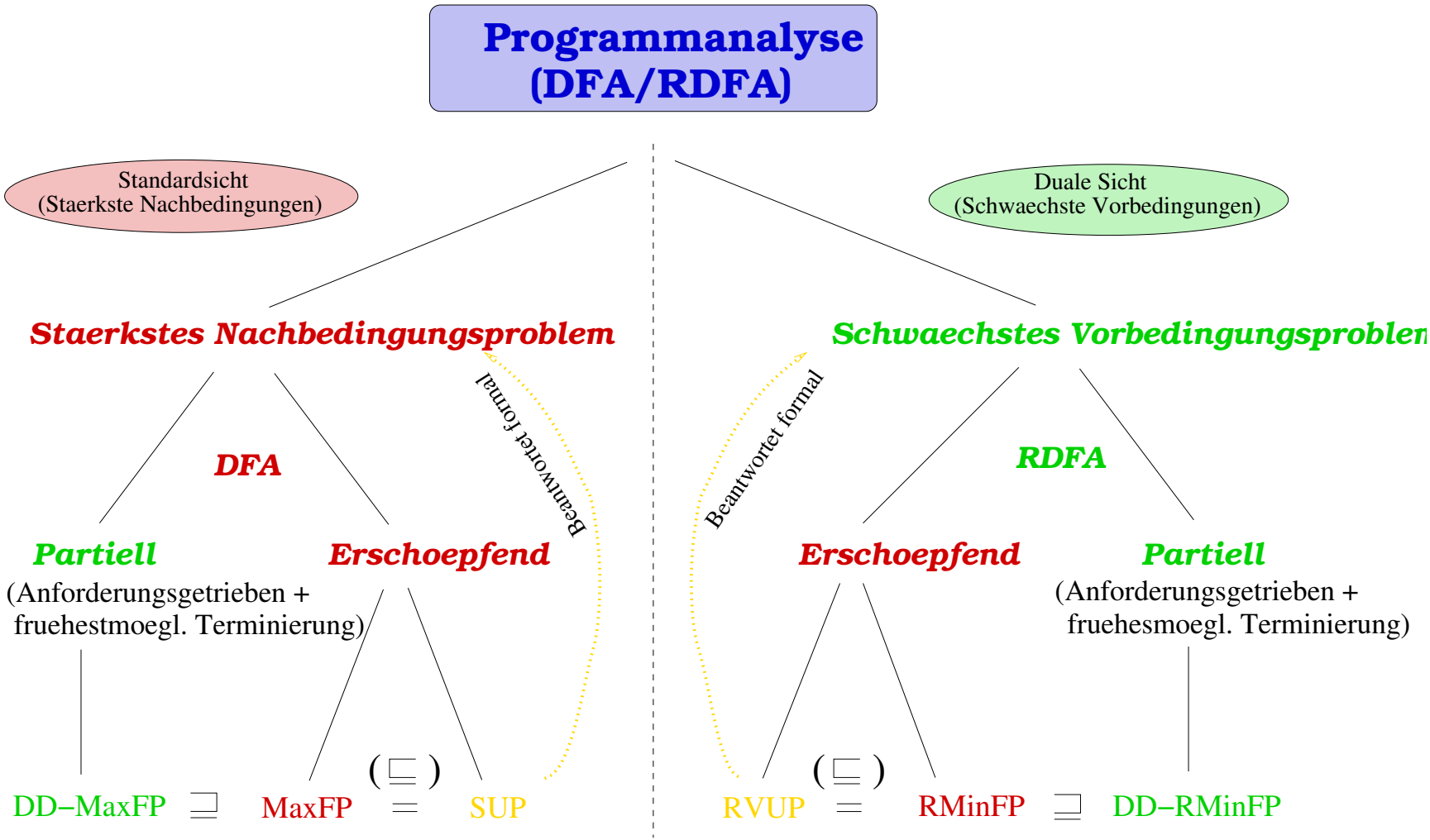
...von erschöpfender und anforderungsgetriebener Analyse liefert den Ausgangspunkt zur Auffächerung von DFA u. RDFA.



Dabei bedeuten:

- Erschöpfend: Die Analyse erfolgt für alle Programmpunkte bis zum Erreichen des Fixpunktes.
- Anforderungsgetrieben: Die Analyse konzentriert sich auf ausgewählte Programmbereiche/-punkte bei ggf. frühestmöglichem Analyseabbruch, d.h. bereits vor Erreichen des Fixpunktes mit noch echter Fixpunktapproximation, falls die Analysefrage bereits zweifellos beantwortet ist.

Die Auffächerung von DFA und RDFA



Die resultierende Problemmatrix

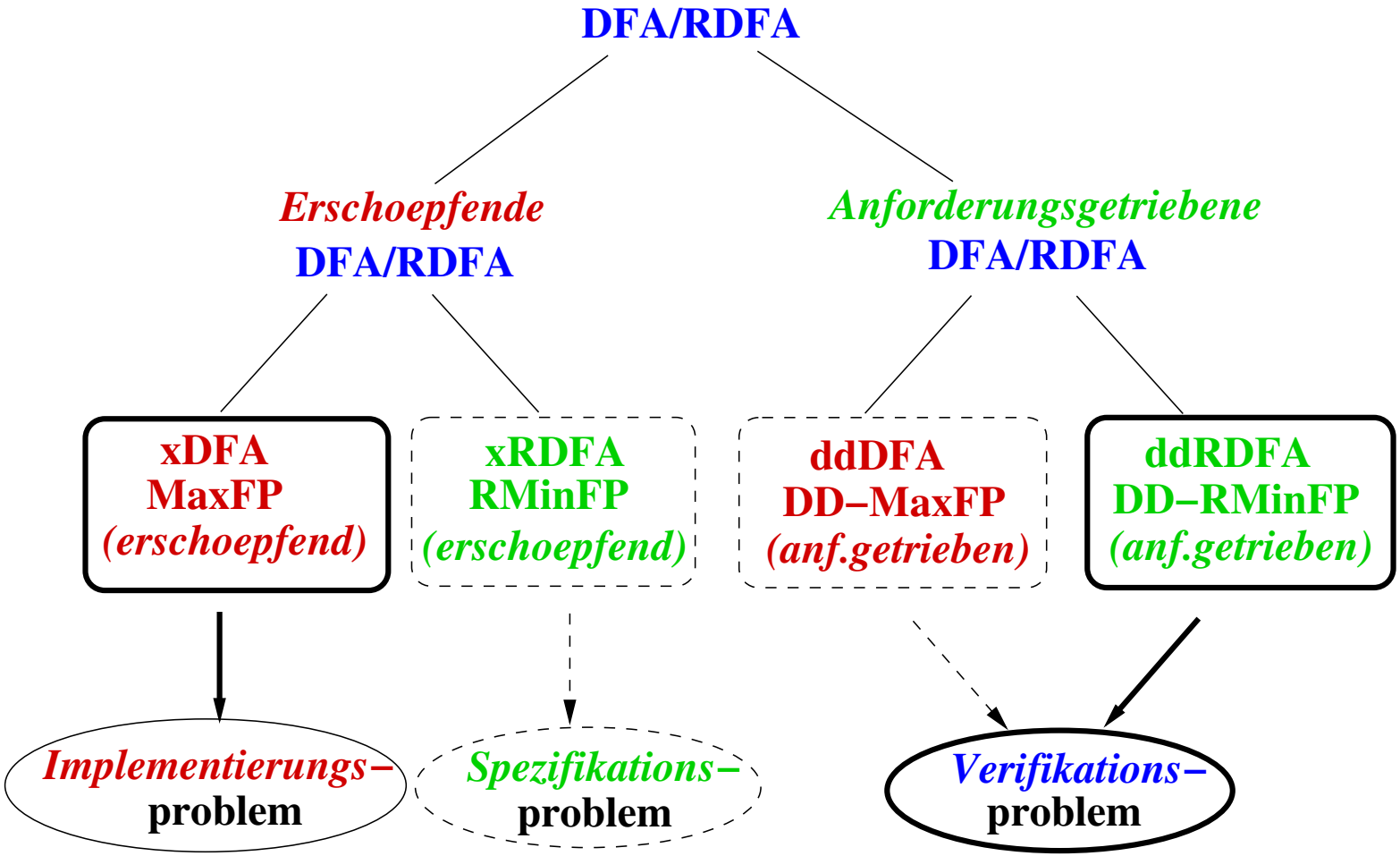
...mit neuer "adressiert-von"-Befüllung:

	a) Ges.prog. π / G_π	b) Punktauswahl $N'_{G_\pi} \subseteq N_{G_\pi}$	c) Punktges. N_{G_π}
1. $\{p\}\ \pi\ \{?\}$	DFA (subsumiert v. 1c)	DD-RDFA	DFA
2. $\{?\}\ \pi\ \{q\}$	RDFA (subsum. v. 2b für **e**)	DD-RDFA	RDFA
3. $\{p\}\ \pi\ \{q\}$?	Axiom. Verif. DD-DFA/DD-RDFA + frühestmögl. Term.	DD-DFA/DD-RDFA + frühestmögl. Term.	DD-DFA/DD-RDF + frühestmögl. Ter

...Einträge in grau weisen dabei auf geringe oder fehlende praktische Bedeutung und Relevanz hin.

Die Anwendungsschwerpunkte v. DFA u. RDFA

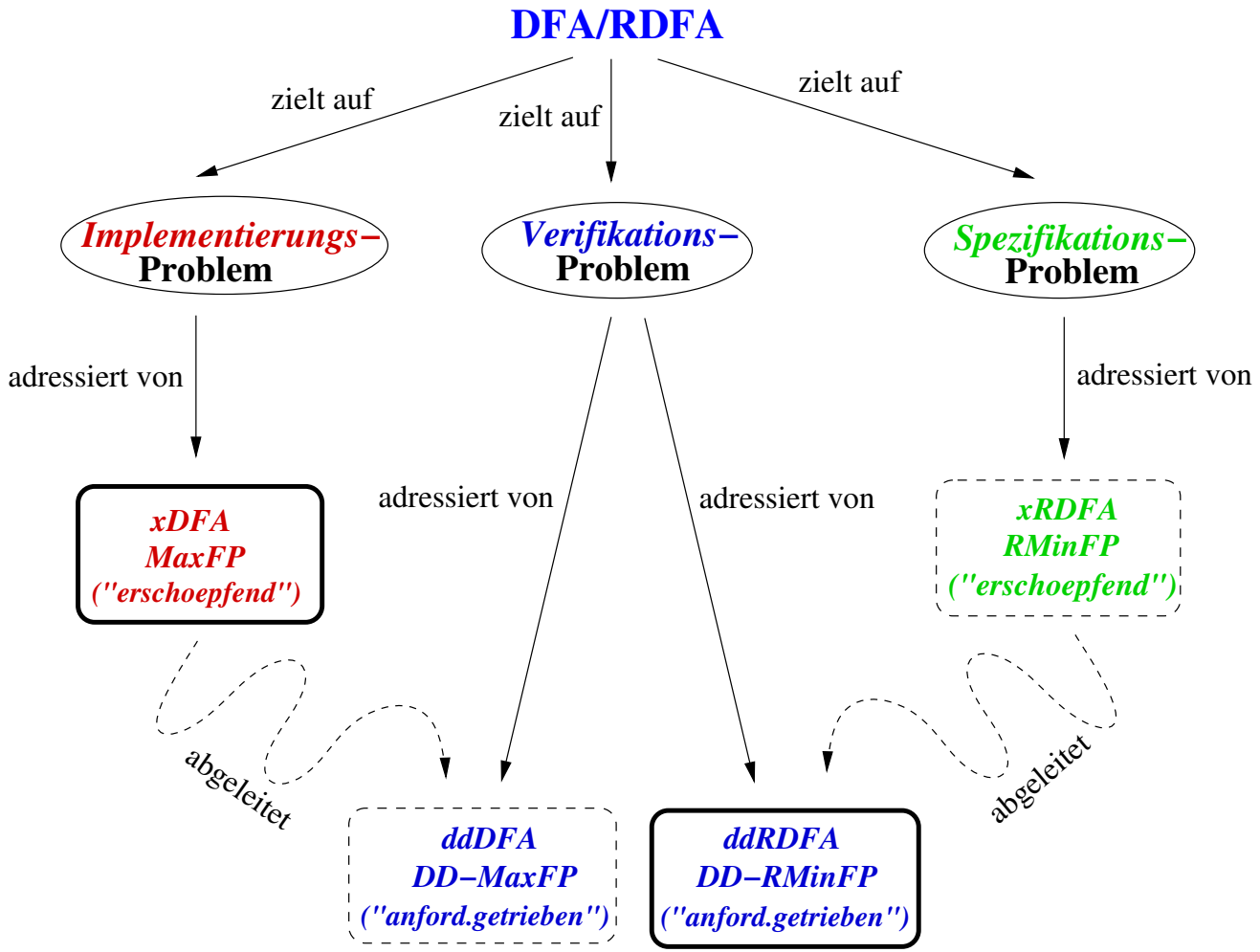
... in 'von Analyse zum Problem'-Sicht:



...mit Haupt- (durchgezogen) und Nebenwegen (gestrichelt).

Die Anwendungsschwerpunkte v. DFA u. RDFA

...umgekehrt in 'vom Problem zur Analyse'-Sicht:



...mit Haupt- (durchgezogen) und Nebenwegen (gestrichelt).

Kapitel 13.3

Pragmatisch nach adressierten Eigenschaften: Funktional vs. nichtfunktional

Axiomatische Verifikation

...fokussiert auf

- funktionale Eigenschaften (ein Programm sortiert, permutiert, zieht Wurzeln,...)

und damit auf funktionale Korrektheit.

Approximative Berechnungen der Eigenschaften sind wg. des Fokus auf funktionale Korrektheit i.a. nutzlos; die Aufgabe von Vollständigkeit zugunsten von Automatisierbarkeit, Performanz, Skalierbarkeit ist deshalb meist nicht möglich.

- Vollautomatisierung axiomatischer Verifikation ist aus theoretischen Gründen ausgeschlossen und unerreichbar!

Datenflussanalyse, reverse Datenflussanalyse

...fokussieren auf

- nichtfunktionale 'strukturelle' Eigenschaften (Berechnungen sind redundant, Variablen initialisiert, tot, Ausdrücke wertkonstant,...)

zur Klärung oder Verbesserung nichtfunktionaler Eigenschaften (Sicherheit, Performanz, Speichereffizienz,...)

Approximative Berechnungen der Eigenschaften sind wg. des Fokus auf nichtfunktionale Eigenschaften oft noch nützlich; die Aufgabe von Vollständigkeit zugunsten von Automatisierbarkeit, Performanz, Skalierbarkeit ist deshalb meist möglich (und nötig)!

- Vollautomatisierung von Datenfluss- und reverser Datenflussananalyse ist der Regelfall.

Kapitel 13.4

Zusammenfassung, Fazit

Axiomatische Verifikation

...unterstützt in erster Linie die Lösung des

- Gültigkeitsproblems (Verifikationsproblems)

$$\{p\}\ \pi\ \{q\}\ ?$$

Aufgabe: Verifikation (Erfolgsfall) bzw. Falsifikation (Misserfolgsfall), ob das Tripel $\{p\}\ \pi\ \{q\}$ gültig ist im Sinn partieller bzw. totaler Korrektheit.

Methode: Beweisführung im Hoare-Kalkül, semi-automatisch.

Mögliche Ergebnisse:

- Der Beweis gelingt: die Gültigkeit des Tripels ist bewiesen (verifiziert).
- Der Beweis scheitert: die Gültigkeit des Tripels ist widerlegt (falsifiziert).

Die Beweismethode axiomatischer Verifikation

...bietet zur systematischen Lösung des

- stärksten Nachbedingungsproblems (Implementier.prob.)

$$\{p\}\ \pi/G_\pi\ \{?\}$$

- schwächsten Vorbedingungsproblems (Spezifikationsprob.)

$$\{?\}\ \pi/G_\pi\ \{q\}$$

allenfalls indirekte oder keine Unterstützung:

- Im Misserfolgsfall liefert der fehlschlagende Beweis nur indirekt Hinweise zur Nachjustierung von Vor- oder/und Nachbedingung.
- Im Erfolgsfall liefert der gelungene Beweis nur indirekt Hinweise zur Stringenz von Vor- und Nachbedingung.
- Insgesamt unterstützt die Methode ausschließlich eine 'Gesamtprogrammverifikation' ('Anfang-zu-Ende'), keine für einzelne oder jeden Programmpunkt.

Datenflussanalye

...unterstützt in erster Linie die Lösung des

- stärksten Nachbedingungsproblems (Implementier.prob.)

$$\{c_s\}\ N_{G_\pi}\ \{?\}$$

Aufgabe: Für jeden Programmpunkt die Berechnung der größtmöglichen ($\widehat{=}$ stärksten) DFA-Information für gegebene Anfangszusicherung c_s am Programmanfang.

Methode: Berechnung der *MaxFP*-Semantik, vollautomatisch.

Speziell gilt:

- Die für den Endknoten $\mathbf{e}$ berechnete DFA-Information c_e ist die 'Gesamtprogrammanalyse' ('Anfang-zu-Ende'):

$$\{c_s\}\ G_\pi\ \{c_e\}$$

Datenflussanalye

...bietet (außer Raten und Ausprobieren) keine Unterstützung zur Lösung des

- schwächsten Vorbedingungsproblems (Spezifikationsprob.)

$$\{?\}\ \pi/G_\pi\ \{q\}$$

...bietet durch Lösung des stärksten Nachbedingungsproblems mit anschließendem Vergleich eine (ineffiziente) Unterstützung zur Lösung des

- Gültigkeitsproblems (Verifikationsproblems)

$$\{p\}\ \pi\ \{q\}\ ?$$

Reverse Datenflussanalye

...unterstützt in erster Linie die Lösung des

- schwächsten Vorbedingungsproblems (Spezifikationsprob.)

$$\{?\}\ N_{G_\pi}\ \{c_q\}$$

Aufgabe: Für jeden Programmpunkt die Berechnung der kleinstmöglichen ($\widehat{=}$ schwächsten) DFA-Information für gegebene Sollzusicherung c_q am Knoten q.

Methode: Berechnung d. *RMinFP*-Semantik, vollautomatisch.

Speziell gilt:

– Die für den Anfangsknoten **s** berechnete DFA-Information $c_{\mathbf{s}}$ zur Sollzusicherung $c_{\mathbf{e}}$ an Endknoten **e** ist die 'Gesamtprogrammanalyse' ('Ende-zu-Anfang'):

$$\{c_{\mathbf{s}}\}\ G_\pi\ \{c_{\mathbf{e}}\}$$

Reverse Datenflussanalye

...bietet (außer Raten und Ausprobieren) keine Unterstützung zur Lösung des

- stärksten Nachbedingungsproblems (Implementier.prob.)

$$\{c_s\}\ N_{G_\pi}\ \{?\}$$

...bietet durch Lösung des schwächsten Vorbedingungsproblems mit anschließendem Vergleich eine (u.U. effiziente) Unterstützung zur Lösung des

- Gültigkeitsproblems (Verifikationsproblems)

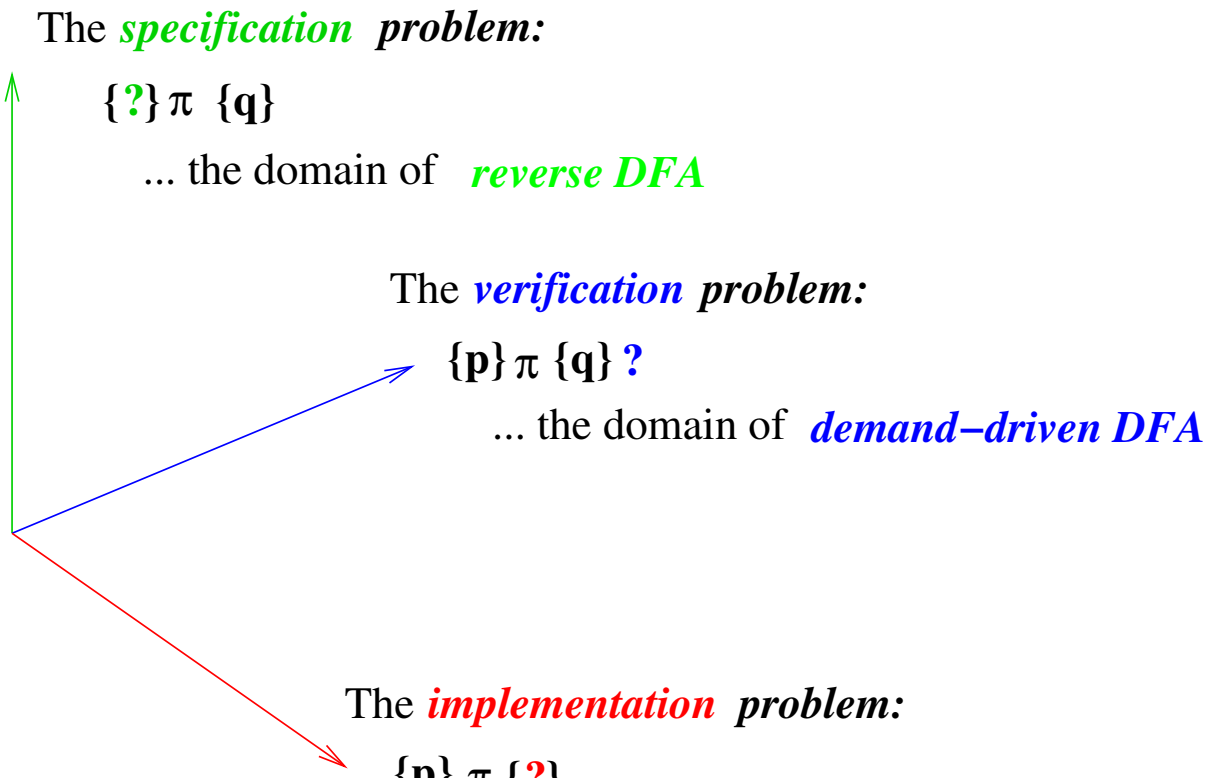
$$\{p\}\ \pi\ \{q\}\ ?$$

Zusammengefasst

Das Gültigkeits- bzw. Verifikationsproblem

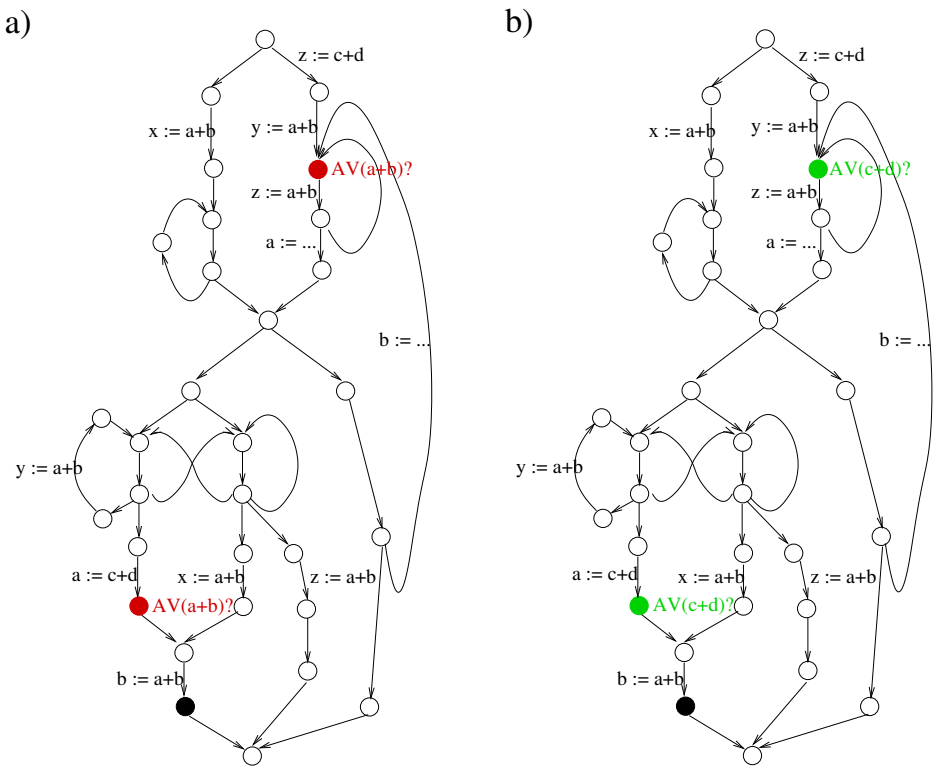
$$\{p\}\ \pi\ \{q\}\ ?$$

spielt weder für Datenfluss- noch reverse Datenflussanalyse in der Praxis eine Rolle, bietet sich aber für die davon abgeleiteten Formen anforderungsgetriebener Datenflussanalyse an.



Übungsaufgabe 13.4.1

Warum spielt der *DD-MaxFP*-Ansatz keine Rolle im Vergleich zum *DD-RMinFP*-Ansatz? Betrachte dazu, ob $a + b$ und $c + d$ an den farblich hervorgehobenen Programmpunkten verfügbar sind und vergleiche den Berechnungsaufwand von *DD-MaxFP*- und *DD-RMinFP*-Ansatz. Welcher generelle Unterschied zwischen den beiden Ansätzen wird deutlich?



Übungsaufgabe 13.4.2

Vergleiche Aufgaben und Vorgehen von:

- Typprüfung (ein Programmierervorschlag einer gültigen Typisierung wird auf Stichhaltigkeit überprüft):
  $\rightsquigarrow$ Verifikation(sproblem)
- Typinferenz (eine stichhaltigeTypisierung wird ohne vorgegebenen Typisierungsvorschlag generiert bzw. eine Fehlermeldung ausgegeben, wenn dies fehlschlägt):
  $\rightsquigarrow$ Analyse(problem)

Leisten Typprüfung und Typinferenz nicht ähnliches? Ist das Analyseproblem Inferenz nicht sogar schwerer als das Verifikationsproblem Prüfung?

Legt das Beispiel nicht nahe, dass Verifikation nicht notwendig tiefgehender oder konzeptuell oder berechnungsmäßig schwerer sein muss als Analyse, sondern dass Verifikation und Analyse Seiten derselben Münze oder gar Aspekte derselben Seite sind?

Teil V

Fixpunkte, Transformationen und Optimalität

Informell

Optimalität als

- Korrektheit und Vollständigkeit

von (Optimierungs-) Transformationen.

Kapitel 14

Chaotische Fixpunktiteration

Kapitel 14.1

Motivation

Motivation

...viele praktisch relevante Probleme in der Informatik lassen sich durch

- Systeme rekursiver Gleichungen

beschreiben, deren Lösung sich als

- größte/kleinste Lösung

dieser Systeme ergibt.

Bisherige Beispiele aus der Vorlesung:

- Denotationelle Semantik von WHILE (Kap. 3).
- Maximale/minimale DFA-Fixpunktsemantik (Kap. 8).
- Minimale/maximale RDFA-Fixpunktsemantik (Kap. 10).
- Maximale/minimale Fixpunktsemantik paralleler Bitvektorprobleme (Kap. 12).

Bisherige Beispiele aus der Vorlesung

- Gleichungssystem zur denotationellen WHILE -Semantik:

$$[\![\mathit{skip}]\!]_{ds} = \mathit{id}$$
$$[\![x := t]\!]_{ds}(\sigma) = \sigma[[\![t]\!]_A(\sigma)/x]$$
$$[\![\pi_1; \pi_2]\!]_{ds} = [\![\pi_2]\!]_{ds} \circ [\![\pi_1]\!]_{ds}$$
$$[\![\text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}]\!]_{ds} = \mathit{cond}\,([\![b]\!]_B, [\![\pi_1]\!]_{ds}, [\![\pi_2]\!]_{ds})$$
$$[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} = \mathit{FIX}\ F$$
$$\text{mit } F\ g = \mathit{cond}\,([\![b]\!]_B, g \circ [\![\pi]\!]_{ds}, \mathit{id})$$

- Gleichungssystem zur maximalen DFA-Fixpunktsemantik:

$$\mathit{inf}(n) = \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcap \{ [\![(m,n)]\!](\mathit{inf}(m)) \mid m \in \mathit{pred}(n) \} & \text{sonst} \end{cases}$$

- Gleichungssystem zur minimalen RDFA-Fixpunktsemantik:

$$\mathit{reqInf}(n) = \begin{cases} c_q & \text{falls } n = \mathbf{q} \\ \bigsqcup \{ [\![(n,m)]\!]_R(\mathit{reqInf}(m)) \mid m \in \mathit{succ}(n) \} & \text{sonst} \end{cases}$$

Allgemein(er)

...gesucht ist eine extreme (d.h., kleinste/größte) Lösung

$$\begin{aligned} x &= f_1(x) \\ x &= f_2(x) \\ &\vdots \\ x &= f_n(x) \end{aligned}$$

eines Systems rekursiver Gleichungen über einer Familie

$$\mathcal{F} =_{df} \{f_k : D \to D \mid 1 \leq k \leq n\}$$

monotoner Funktionen auf einer wohlfundierten partiellen Ordnung $(D, \sqsubseteq)$.

Ziel

...das Lösen von Gleichungssystemen und das Berechnen von Fixpunkten von (Familien von) Funktionen als Frage der Perspektive erkennen:

- Lösen eines Systems rekursiver Gleichungen

$$\begin{aligned} x &= f_1(x) \\ x &= f_2(x) \\ &\vdots \\ x &= f_n(x) \end{aligned}$$

- Berechnen eines gemeinsamen Fixpunktes einer Famile $\mathcal{F}$ von Funktionen, d.h. eines gemeinsamen Fixpunkts x mit

$$x = f_k(x)$$

für alle $1 \leq k \leq n$.

Lösungs- und Fixpunktberechnung

...mittels chaotischer Iterationsalgorithmen.

Iterative Algorithmen zur Berechnung kleinster Lösungen bzw. kleinster Fixpunke beginnen mit

- $\bot$, dem kleinsten Element von D, als initialer Approximation von x und aktualisieren den jeweiligen Approximationswert durch Anwendung der Funktionen f_i $\mathcal{F}$ in einer beliebigen Reihenfolge, um so den kleinsten gemeinsamen Fixpunkt der Funktionen aus $\mathcal{F}$ Schritt für Schritt besser und besser zu approximieren

und im Terminierungsfall exakt zu erreichen.

Diese Vorgehensweise wird als chaotische Iteration bezeichnet.

Wichtige Fixpunktresultate aus der Literatur

...das möglicherweise bekannteste und wichtigste Fixpunktresultat:

- Das Fixpunkttheorem von Tarski [1955]
  - Garantiert die Existenz kleinster Fixpunkte für monotone Funktionen auf vollständigen partiellen Ordnungen.
  - Iterationsschema: $\vec{x}_0 = \vec{\bot}, \vec{x}_1 = \vec{f}(\vec{x}_0), \vec{x}_2 = \vec{f}(\vec{x}_1), \ldots,$ wobei $\vec{x}_i$ den Wert von $\vec{x}$ nach der i-ten Iteration bezeichnet.
  - Vielfach anwendbar, oft aber noch zu speziell.

Verallgemeinerungen, Variationen des Tarskischen Iterationsschemas:

- Vektor-Iterationen: Robert [1976]
- Asynchrone Vektor-Iterationen: Baudet [1978], Cousot [1977], Üresin/Dubois [1989], Wei [1993]
- ...

Vektor-Iterationen, asynchrone Iterationen

Vektor-Iterationen (Robert [1976])

- Gegeben:
  Eine monotone Vektorfunktion $\vec{f} = (f^1, \ldots, f^m)$.
- Gesucht:
  Kleinster Fixpunkt $\vec{x} = (x^1, \ldots, x^m) \in D^m$ von $\vec{f}$.
- Iterationsschema:
  $\vec{x}_0 = \vec{\bot}, \vec{x}_1 = \vec{f}_{J_0}(\vec{x}_0), \vec{x}_2 = \vec{f}_{J_1}(\vec{x}_1), \ldots$, wobei
  $J_i \subseteq \{1, \ldots, m\}$ und die k-te Komponente $\vec{f}_{J_i}(\vec{x}_i)^k$ von $\vec{f}_{J_i}(\vec{x}_i)$ ist $f^k(\vec{x}_i)$, falls $k \in J_i$, und $\vec{x}_i^k$ sonst.

Asynchrone Vektor-Iterationen (Baudet [1978], Cousot [1977], Üresin/Dubois [1989], Wei [1993])

- $\vec{f}_{J_i}$ kann auf Komponenten früherer Vektoren der Iterationsfolge zurückgreifen $\vec{x}_j$, $j \leq i$.

Klassiker zum Nachschlagen und Nachlesen

DER Klassiker mit DEM Fixpunkttheorem schlechthin:

- Alfred Tarski. *A Lattice-theoretical fixpoint theorem and its applications*. Pacific Journal of Mathematics 5(2):285-309, 1955.

Zu chaotischer Iteration:

- F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, Frankreich, Dez. 1976.

Umfassender historischer Abriss zu Fixpunktresultaten:

- Jean-Louis Lassez, V.L. Nguyen, Elizabeth A. Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.

In der Folge

...Vorstellung eines weiteren Fixpunkttheorems, das ohne die übliche Monotonieforderung auskommt:

- Alfons Geser, Jens Knoop, Gerald Lüttgen, Oliver Rüthing, Bernhard Steffen. *Non-monotone Fixpoint Iterations to Resolve Second Order Effects*. In Proceedings of the 6th International Conference on Compiler Construction (CC'96), Springer-V., LNCS 1060, 106-120, 1996.

...mit Anwendungen in Kapitel 14.3, 15 und 16.

Weitere Fixpunktresultate:

- Siehe Anhang A5.

Kapitel 14.2

Chaotisches Fixpunktiterationstheorem

Vorbereitungen (1)

Definition 14.2.1 (Partielle Ordnung, wohlfund. PO)

Eine partielle Ordnung

- ist ein Paar $(D, \sqsubseteq)$ aus einer Menge D und einer reflexiven, antisymmetrischen und transitiven zweistelligen Relation $\sqsubseteq$ über D, d.h. $\sqsubseteq \subseteq D \times D$.
- heißt wohlfundiert, falls jede Kette stationär ist.

Definition 14.2.2 (Kette, stationäre Kette)

Eine aufsteigende Kette

- in einer partiellen Ordnung $(D, \sqsubseteq)$ ist eine Folge $(d_i)_{i \in \mathbb{N}}$ von Elementen aus D, $d_i \in D$, mit $\forall\, i \in \mathbb{N}.\ d_i \sqsubseteq d_{i+1}$.
- heißt stationär, falls $\{d_i \mid i \in \mathbb{N}\}$ endlich ist.

Vorbereitungen (2)

Definition 14.2.3 (Monotonie, Inflationärität)

Eine Funktion $f : D \to D$ auf einer partiellen Ordnung $(D, \sqsubseteq)$ heißt

- monoton, falls $\forall\, d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$.
- inflationär (oder vergrößernd), falls $\forall\, d \in D.\ d \sqsubseteq f(d)$.

Definition 14.2.4 (Funktionssequenzen)

Für eine Familie von Funktionen $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ und ein Wort $s =_{df} (s_1, \ldots, s_n) \in \mathbb{N}^*$ über $\mathbb{N}$, s also eine Folge natürlicher Zahlen, bezeichnet f_s die Funktionssequenz der Funktionen f_i, $1 \leq i \leq n$, gegeben durch ihre sequentielle Komposition:

- $f_s =_{df} f_{s_n} \circ \cdots \circ f_{s_1}$.

Strategien, Iterationsfolgen, faire Strategien

Sei $(D, \sqsubseteq)$ eine partielle Ordnung und $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine Familie inflationärer Funktionen $f_k : D \to D$.

Definition 14.2.5 (Strategie, chaotische Iterationsfolge, faire Strategie)

1. Eine Strategie ist eine beliebige Funktion $\gamma : \mathbb{N} \to \mathbb{N}$.
2. Eine Strategie γ und ein Element $d \in D$ induzieren eine induktiv definierte chaotische Iterationsfolge $f_\gamma(d) = (d_i)_{i \in \mathbb{N}}$, $d_i \in D$, mit $d_0 = d$ und $d_{i+1} = f_{\gamma(i)}(d_i)$.
3. Eine Strategie γ heißt fair gdw

$$\forall\, i, k \in \mathbb{N}.\, (f_k(d_i) \neq d_i \;\Rightarrow\; \exists\, j > i.\, d_j \neq d_i)$$

Familien-Monotonie

...ein abgeschwächter Monotoniebegriff.

Sei $(D, \sqsubseteq)$ eine partielle Ordnung.

Definition 14.2.6 (Familien-Monotonie)

Eine Familie $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ von Funktionen $f_k : D \to D$ heißt familien-monoton, falls für alle $k \in \mathbb{N}$ gilt:

$$d \sqsubseteq d' \Rightarrow \exists s \in \mathbb{N}^*. f_k(d) \sqsubseteq f_s(d')$$

Es gilt:

Lemma 14.2.7

Eine Familie $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ von Funktionen ist familien-monoton, wenn alle Funktionen f_k, $k \in \mathbb{N}$, (im üblichen Sinn) monoton sind.

Beispiel: Familien-Monotonie (1)

...betrachte:

- $(\mathbb{N}_1, \leq)$: die durch die Relation kleiner oder gleich partiell geordnete Menge der natürlichen Zahlen mit 1 als kleinstem Element:

$$1 \leq 2 \leq 3 \leq 4 \leq 5 \leq 6 \leq 7 \leq \ldots$$

- $f : \mathbb{N}_1 \to \mathbb{N}_1$ definiert durch:

$$\forall\, n \in \mathbb{N}_1.\ f(n) = \begin{cases} 4 & \text{falls } n = 1 \\ 3 & \text{falls } n = 2 \\ n & \text{sonst} \end{cases}$$

- $g : \mathbb{N}_1 \to \mathbb{N}_1$ definiert durch: $\forall\, n \in \mathbb{N}_1.\ g(n) = n + 1$
- $\mathcal{F} = \{f, g\}$ Familie von Funktionen.

Beispiel: Familien-Monotonie (2)

Proposition 14.2.8

1. Abbildungen f und g sind inflationär.
2. Abbildung g ist monoton.
3. Abbildung f ist nicht monoton.
4. Funktionenfamilie $\mathcal{F} = \{f, g\}$ ist familien-monoton.

Beweis. Zu 1) und 2): Offensichtlich erfüllt.

Zu 3) Es gilt:

- $1 \leq 2$, aber $f(1) = 4 \not\leq 3 = f(2)$ $(= f^i(2), i \in \mathbb{N}_1)$
- $1 \leq 3$, aber $f(1) = 4 \not\leq 3 = f(3)$ $(= f^i(3), i \in \mathbb{N}_1)$
- $\forall\,(m, n) \in \mathbb{N} \times \mathbb{N} \backslash \{(1, 2), (1, 3)\}.\ m \leq n \Rightarrow f(m) \leq f(n)$

Zu 4): Wegen 3) reicht:

- $1 \leq 2 \wedge f(1) \not\leq f(2)$, aber $f(1) = 4 \leq 4 = g(g(2)) = g(f(2))$
- $1 \leq 3 \wedge f(1) \not\leq f(3)$, aber $f(1) = 4 \leq 4 = g(3)$

Das 'monotoniefreie' chaot. Fixpunkttheorem

Theorem 14.2.9 (Chaotisches Fixpunktiterationsth.)

Sei $(D, \sqsubseteq)$ eine wohlfundierte partielle Ordnung mit kleinstem Element $\bot$, $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine familien-monotone Familie inflationärer Funktionen und $\gamma : \mathbb{N} \to \mathbb{N}$ eine faire Strategie.

Dann gilt:

1. Die Funktionsfamilie $\mathcal{F}$ hat einen kleinsten gemeinsamen Fixpunkt $\mu\mathcal{F}$ mit $\mu\mathcal{F} = \bigsqcup f_\gamma(\bot)$.
2. $\mu\mathcal{F}$ wird stets in einer endlichen Zahl von Iterationsschritten erreicht.

Generischer Fixpunktalgorithmus

...als nichtdeterministischer 'Rumpf'-Algorithmus.

Generischer Fixpunktalgorithmus 14.2.10

(Prolog: Initialisierung von d)
$d := \bot$;
(Hauptschleife: Iterative Fixpunktberechnung)
WHILE $\exists\, k \in \mathbb{N}.\ d \neq f_k(d)$ DO
  CHOOSE $k \in \mathbb{N}$ WITH $d \sqsubset f_k(d)$
    $d := f_k(d)$
  ESOOHC
OD.

Anmerkung: Wegen f_k, $k \in \mathbb{N}$, inflationär, folgt aus $d \neq f_k(d)$ unmittelbar $d \sqsubset f_k(d)$.

Kapitel 14.3

Anwendungen

Kapitel 14.3.1

Vektor-Iterationen

Vorbereitung

Sei

- $(C, \sqsubseteq_C)$ wohlfundierte partielle Ordnung
- $D =_{df} C^n$, $n \in \mathbb{N}$, partiell geordnet durch die punktweise Ausdehnung von $\sqsubseteq_C$ auf D
- $f : D \to D$ monotone Funktion auf D

Anstelle der Iterationsfolge

$$d_0 = \bot,\ d_1 = f(\bot),\ d_2 = f(d_1),\ \ldots$$

nach dem Schema aus Tarskis Fixpunkttheorem, können wir zu einer Zerlegung von f in seine Komponenten f^k übergehen mit

$$f(d) = (f^1(d), \ldots, f^n(d))$$

und zu selektiven Aktualisierungen durch ausgewählte Komponentenfunktionen, wobei wir mit oberen Indizes i die i-te Komponente eines Vektors der Länge n bezeichnen.

Vektor-Iterationen (1)

Definition 14.3.1.1 (Vektor-Iteration)

Eine Vektor-Iteration ist eine Iterationsfolge der Form

$$d_0 = \bot,\ d_1 = f_{J_0}(\bot),\ d_2 = f_{J_1}(d_1),\ \ldots$$

mit $J_i \subseteq \{1, \ldots, n\}$ und wo

$$f_J(d)^i =_{df} \begin{cases} f^i(d) & \text{falls } i \in J \\ d^i & \text{sonst} \end{cases}$$

selektiv die durch J spezifizierten Komponentenfunktionen anwendet und die entsprechenden Komponentenwerte aktualisiert.

Vektor-Iterationen (2)

Beachte:

- Die Menge der gemeinsamen Fixpunkte der Funktionenfamilie $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{1, \ldots, n\}\}$ ist gleich der Menge der Fixpunkte von $f : D \to D$.
- Jede Funktion f_J is monoton, da f monoton ist.

Anwendung von Fixpunkttheorem 14.2.9

...zur Modellierung der Vektor-Iteration.

Erforderlich: Verallgemeinerung des Strategiebegriffs auf einen Strategiebegriff für Mengen.

Definition 14.3.1.2 (Faire Mengenstrategie)

1. Eine Mengenstrategie ist eine (beliebige) Funktion

$$\gamma : \mathbb{N} \to \mathcal{P}(\{1, \ldots, n\}).$$

   Intuition: $\gamma(i)$ liefert eine Menge J_i von Indizes aus der Menge $\{1, \ldots, n\}$, deren zugehörige Komponenten im i-ten Schritt der Iteration aktualisiert werden sollen.

2. Eine Mengenstrategie heißt fair gdw

$$\forall\, i \in \mathbb{N},\, J \subseteq \mathbb{N}.\ (f_J(d_i) \neq d_i \ \Rightarrow\ \exists j > i.\, d_j \neq d_i)$$

Modellierungsresultate (1)

Sei

- $(C, \sqsubseteq_C)$ wohlfundierte partielle Ordnung mit kleinstem Element $\bot_C$.
- $D =_{df} C^n$, $n \in \mathbb{N}$, partiell geordnet durch die punktweise Ausdehnung von $\sqsubseteq_C$ auf D.

Lemma 14.3.1.3 (Ketten durch Vektor-Iteration)

Sei $f = (f^1, \ldots, f^n)$ eine monotone Funktion auf D, sei $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{1, \ldots, n\}\}$ eine Familie von Funktionen $f_J : D \to D$ im Sinn von Definition 14.3.1.1 und sei $\gamma : \mathbb{N} \to \mathcal{P}(\{1, \ldots, n\})$ eine Mengenstrategie.

Dann gilt: $f_\gamma(\bot)$ liefert eine Kette.

Das heißt: Jede chaotische Iterationsfolge liefert eine Kette.

Modellierungsresultate (2)

Theorem 14.3.1.4 (Chaotische Vektor-Iteration)

Sei $f = (f^1, \ldots, f^n)$ eine monotone Funktion auf D, sei $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{1, \ldots, n\}\}$ eine Familie von Funktionen $f_J : D \to D$ gemäß Definition 14.3.1.1 und sei γ eine faire Mengenstrategie.

Dann gilt:

1. $\bigsqcup f_\gamma(\bot)$ ist der kleinste Fixpunkt $\mu\mathcal{F}$ der Familie von Funktionen $\mathcal{F}$.
2. $\mu\mathcal{F}$ wird stets in einer endlichen Zahl von Iterationsschritten erreicht.
3. Die kleinsten Fixpunkte von f und $\mathcal{F}$ stimmen überein, d.h.:
$$\mu\mathcal{F} = \mu f$$

Anmerkungen

Die Aussage von Theorem 14.3.1.4

- ist ein Spezialfall des Chaotischen Fixpunktiterationstheorems 14.2.9 für Vektor-Iterationen und folgt zusammen mit Lemma 14.3.1.3.

Für $|\mathcal{F}| = 1$ reduziert sich

- die Aussage von Theorem 14.3.1.4 auf Tarskis Fixpunkttheorem für den Fall wohlfundierter partieller Ordnungen.

Kapitel 14.3.2

Datenflussanalyse

Anwendung von Fixpunkttheorem 14.2.9

...am Beispiel intraprozeduraler DFA und der Gleichungssysteme für die *MaxFP*- und *MinFP*-Semantik.

Das Gleichungssystem der *MaxFP*-Semantik:

$$inf(n) = \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcap \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

Die *MaxFP*-Semantik:

...definiert als größte Lösung des *MaxFP*-Gleichungssystems, bezeichnet mit:

$$\nu\text{-}inf_{c_{\mathbf{s}}} : N \to \mathcal{C}$$

Dual zur *MaxFP*-Semantik

...die *MinFP*-Semantik.

Das Gleichungssystem der *MinFP*-Semantik:

$$inf(n) \quad = \quad \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcup \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

Die *MinFP*-Semantik:

...definiert als kleinste Lösung des *MinFP*-Gleichungssystems, bezeichnet mit:

$$\mu\text{-}inf_{c_{\mathbf{s}}} : N \to \mathcal{C}$$

Generischer *MinFP*-Fixpunktalgorithmus

Generischer *MinFP*-Fixpunktalgorithmus 14.3.2.1

(Prolog: Initialisierung von *inf* und *workset*)

$inf[\mathbf{s}] := c_{\mathbf{s}}$;
FORALL $n \in N \backslash \{\mathbf{s}\}$ DO $inf[n] := \bot$ OD;
$workset := N$;

(Hauptschleife: Iterative Fixpunktberechnung)

WHILE $workset \neq \emptyset$ DO
  CHOOSE $k \in workset$
    $workset := workset \backslash \{ k \}$;
    $new := \bigsqcup \{ [\![(m, k)]\!](inf[m]) \mid m \in pred_G(k) \}$;
    IF $new \sqsupset inf[k]$ THEN
      $inf[k] := new$;
      $workset := workset \cup succ_G(k)$
    FI
  ESOOHC OD.

Zur Fixpunktcharakt. der *MinFP*-Lösung (1)

Vorbereitung:

Sei

- $G = (N, E, \mathbf{s}, \mathbf{e})$ ein beliebiger, fest gewählter Flussgraph.
- $\mathtt{n} =_{df} |N|$ die Zahl der Knoten in N.
- $\widehat{\mathcal{C}} =_{df} (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ ein vollständiger Verband.
- $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ eine monotone lokale DFA-Semantik für G.

Die Knoten der Menge N von G werde mit der Menge

- der natürlichen Zahlen $\{1, \ldots, \mathtt{n}\}$ als Ordnungsnummern identifiziert.

Zur Fixpunktcharakt. der *MinFP*-Lösung (2)

Sei $D =_{df} \mathcal{C}^{\mathtt{n}}$ versehen mit der punktweisen Ausdehnung der Ordnungsrelation $\sqsubseteq_{\widehat{\mathcal{C}}}$ von $\widehat{\mathcal{C}}$ auf D.

Mit dieser Festlegung gilt:

- $(D, \sqsubseteq)$ ist eine wohlfundierte partielle Ordnung.
- Ein Element $d = (d^1, \ldots, d^{\mathtt{n}}) \in D$ stellt eine Annotation des Flussgraphen dar, wobei der Knoten mit der Ordnungsnummer k mit dem Verbandselement $d^k \in \mathcal{C}$ als Wert benannt ist.

Zur Fixpunktcharakt. der *MinFP*-Lösung (3)

Für jeden Knoten des Flussgraphen definieren wir jetzt die knotenspezische Funktion $f^k : D \to C$ durch

$$f^k(d^1, \ldots, d^n) =_{df} d'^k$$

mit

$$d'^k = \bigsqcup \{ [\![(m,k)]\!](d^m) \mid m \in pred_G(k) \}$$

wobei k die Ordnungsnummer des Knotens ist.

Intuitiv: f^k beschreibt den Effekt der Aktualisierung der DFA-Information am Knoten mit der Ordnungsnummer k entsprechend des Vorgehens im Gen. Fixpunktalgorithmus 14.3.2.1:

$$new := \bigsqcup \{ [\![(m,k)]\!](inf[m]) \mid m \in pred_G(k) \}$$

entspricht:

$$d'^k = \bigsqcup \{ [\![(m,k)]\!](d^m) \mid m \in pred_G(k) \}$$

Charakterisierungsresultate

Lemma 14.3.2.2 (Lösung gdw. Fixpunkt)

Für alle Elemente $d \in D$ gilt:

d ist eine Lösung des *MinFP*-Gleichungssystems gdw d ist ein Fixpunkt der Funktion $f =_{df} (f^1, \ldots, f^n)$.

Theorem 14.3.2.3 (Korrektheit und Terminierung)

Jeder Lauf des generischen *MinFP*-Fixpunktalgorithmus 14.3.2.1 terminiert mit der *MinFP*-Semantik von G für die lokale DFA-Semantik $[\![\ ]\!]$ und die Startzusicherung c_s.

Anmerkungen

...zum Beweis von Theorem 14.3.2.3:

- Der *MinFP*-Fixpunktalgorithmus 14.3.2.1 folgt dem Muster von Rumpfalgorithmus 14.2.10 mit $\mathcal{F} = \{f_{\{k\}} \mid 1 \leq k \leq \mathtt{n}\}$.
- Die Verwendung von Variable *workset*, die die Invariante $\mathit{workset} \supseteq \{k \mid f_{\{k\}}(d) \neq d\}$ erfüllt, trägt zu höherer Effizienz bei.
- Offensichtlich gilt: f ist monoton.

Damit sind insgesamt die Voraussetzungen von Theorem 14.3.1.4 erfüllt, womit Theorem 14.3.2.4 folgt.

Kapitel 14.4

Zusammenfassung, Ausblick

Zusammenfassung

Die Suche nach

- Lösungen von Gleichungssystemen
- Fixpunkten von Funktionen

ergeben sich als unterschiedliche Sichten auf dasselbe Problem.

Die auf den ersten Blick unmotiviert erscheinende Sprechweise von Fixpunktlösungen der Gleichungssysteme der denotationellen DFA-Semantiken hat hierin ihren Ursprung und auch ihre Berechtigung.

Hieraus erklären sich auch die Bedeutung und Anwendungsvielfalt von Fixpunkttheoremen.

Ausblick

Speziell weitere Anwendungen des ‘monotoniefreien’ chaotischen Fixpunktiterationstheorems 14.2.9 werden in Kapitel 15 und 16 zum Beweis der Optimalität der Transformationen für die

- Elimination partiell toten Codes (Kap. 15)
- Elimination partiell redundanter Anweisungen (Kap. 16)

betrachtet; klassische Fixpunkttheoreme sind dafür nicht anwendbar.

Kapitel 14.5

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (1)


Nicolas Bourbaki. *Sur la théorème de Zorn*. Archiv der Mathematik 2:434-437, 1949/50.


Patrick Cousot, Radhia Cousot. *Constructive Versions of Tarski's Fixed Point Theorems*. Pacific Journal of Mathematics 82(1):43-57, 1979.


Alfons Geser, Jens Knoop, Gerald Lüttgen, Oliver Rüthing, Bernhard Steffen. *Non-monotone Fixpoint Iterations to Resolve Second Order Effects*. In Proceedings of the 6th International Conference on Compiler Construction (CC'96), Springer-V., LNCS 1060, 106-120, 1996.


Jean-Louis Lassez, V.L. Nguyen, Elizabeth A. Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (2)

- F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, Frankreich, Dez. 1976.
- Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5(2):285-309, 1955.
- Franklyn Turbak, David Gifford with Mark A. Sheldon. *Design Concepts in Programming Languages*. MIT Press, 2008. (Kapitel 5, Fixed Points)
- Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993. (Chapter 8, Introduction to domain theory; Chapter 9, Recursion equations; Chapter 10, Recursion techniques)

Kapitel 15

Unnötige Anweisungen

Kapitel 15.1

Motivation

Informell

...Anweisungen sind unnötig, wenn sich durch Streichen das 'beobachtbare' Programmverhalten, die Programmsemantik nicht ändern und erhalten bleiben.

In diesem Sinn unnötige Anweisungen treten in vielerlei Form auf, darunter als:

- Unerreichbare Anweisungen (Kapitel 15.2)
- Partiell tote/geisterhafte Anweisungen (Kapitel 15.3)
- Partiell redundante Anweisungen (Kapitel 15.4)
- ...

Transformationen

...zur Beseitigung (oder Elimination) unnötiger Anweisungen zielen darauf, die Performanz eines Programms zu verbessern, ohne dadurch das beobachtbare Verhalten oder die Semantik zu verändern.

Um dies handhabbar zu machen, ist es erforderlich, die Begriffe

- beobachtbares Verhalten
- zu erhaltende Semantik

zu präzisieren und exakt zu fassen.

Transformationen: Korrektheit, Vollständigkeit

...ohne diese Präzisierungen ist es weder möglich für Transformationen

- Korrektheit

zu definieren und nachzuweisen, noch für Optimierungstransformationen, ob, wann und in welchem Sinn sie

- vollständig (oder optimal) sind.

Die Wahl der Präzisierung beeinflusst Korrektheits- und Vollständigkeits- (oder Optimalitäts-) Begriff maßgeblich.

Am Beispiel von Transformationen

..zur Elimination unnötiger Anweisungen:

Die Wahl der Präzisierung beeinflusst maßgeblich, ob, wann und in welchem Sinn eine

- Anweisung als unnötig

angesehen werden kann.

Erst die Präzisierung auch dieses Begriffs erlaubt es, entsprechende (Eliminations-) Transformationen zu definieren und ihre Vollständigkeit (oder Optimalität) in einem wohldefinierten Sinn zu definieren und nachzuweisen.

Korrektheit und Vollständigkeit informell

Eine (Programm-) Transformation ist

- korrekt, wenn sie das beobachtbare Verhalten, die Semantik eines Programms erhält.

Eine Transformation für die Beseitigung unnötiger Anweisungen ist

- vollständig (oder optimal), wenn sie alle in einem wohldefinierten Sinn unnötigen Anweisungen in einem Programm beseitigt.

Relativität von Korrektheit und Vollständigkeit

...Korrektheit und Vollständigkeit (oder Optimalität) von Transformationen sind keine absoluten, sondern relative Eigenschaften:

- Korrekt relativ zum beobachtbaren Verhalten, der Programmsemantik.
- Vollständig (oder optimal) relativ zu einem (oder mehreren) Optimierungsziel(en).

Wichtig: Beide Definitionen erlauben triviale Lösungen: Die

- identische Programmtransformation `tuNix` ist korrekt.
- alles streichende Programmtransformation `streichAlles` ist vollständig.

Offenbar

...sind weder `tuNix` noch `streichAlles` sinnvoll oder gewollt:

- `tuNix`: Stets korrekt, selten vollständig.
- `streichAlles`: Stets vollständig, selten korrekt.

Mit der Suche nach Transformationen, die

- korrekt und vollständig

sind, aber auch

- wirksam (nicht nur in der Theorie, auch in der Praxis)
- effizient (in der Theorie)
- performant und skalierbar (in der Praxis)
- elegant und einfach (in Theorie und Praxis)
- ...

beginnt Informatik!

Für die Entwicklung von Transformationen

...zur Beseitigung unnötiger Anweisungen sind somit Antworten zentral auf:

- Beobachtbares Verhalten, zu erhaltende Semantik: Wie definiert?
- Anwendungsbereich: Wie ist Unnötigkeit definiert?
- Korrektheit: Werden höchstens in diesem Sinn unnötige Anweisungen gestrichen?
- Vollständigkeit (oder Optimalität): Werden alle in diesem diesem Sinn unnötige Anweisungen gestrichen?

...was wir für verschiedene Präzisierungen von:

- Beobachtbares Verhalten, zu erhaltende Semantik
- Unnötigkeit von Anweisungen

untersuchen und beantworten werden.

Vereinbarungen zu Flussgraphen

...in Kapitel 15 gehen wir davon aus, dass Flussgraphen

- knotenbenannt

sind und dass

- $\mathcal{G}$ die Menge aller Flussgraphen (oder: Programme) G
- $\mathcal{AM}$ die Menge aller Anweisungsmuster α, α' der Form $\alpha \equiv x := t$, $\alpha' \equiv x' := t'$

bezeichnen.

Kapitel 15.2

Unerreichbare Anweisungen

Zusatzvereinbarungen für Flussgraphen

...für Kapitel 15.2.

Für jeden Flussgraphen $G = (N, E, \mathbf{s}) \in \mathcal{G}$ gilt:

- $\mathbf{s} \in N$ hat keine Vorgänger: $pred(\mathbf{s}) = \emptyset$

und bezeichnet einen als Startknoten ausgezeichneten Knoten in G.

Wichtig: Anders als bisher verlangen wir nicht, dass es einen als Endknoten ausgezeichneten Knoten (ohne Nachfolger) in G gibt und jeder Knoten $n \in N$ auf einem Pfad von $\mathbf{s}$ nach $\mathbf{e}$ liegt.

Bezeichnung:

- $src(e)$, $dst(e)$: Anfangs- bzw. Endknoten der Kante e.

Kapitel 15.2.1

Statisch unerreichbare Anweisungen

Statisch unerreichbare Knoten und Kanten

...sei $G = (N, E, \mathbf{s})$ ein (knotenbenannter) Flussgraph.

Definition 15.2.1.1 (Stat. unerr. Knoten/Kanten)

1. Ein Knoten $n \in N$ ist statisch unerreichbar gdw n auf keinem Pfad vom Startknoten des Flussgraphen aus erreichbar ist, d.h., wenn:

$$\mathbf{P}_G[\mathbf{s}, n] = \emptyset$$

2. Eine Kante $e \in E$ ist statisch unerreichbar gdw der Anfangsknoten von e statisch unerreichbar ist, d.h., wenn:

$$\mathbf{P}_G[\mathbf{s}, src(e)] = \emptyset$$

3. Knoten oder Kanten, die nicht statisch unerreichbar sind, heißen statisch erreichbar.

Statisch unerreichbare Anweisungen

...eine 'syntaktische' Eigenschaft.

Definition 15.2.1.2 (Statisch unerreichbare Anw.)

Eine Anweisung α am Knoten $n \in N$ in G ist statisch unerreichbar gdw n (und damit α) statisch unerreichbar ist.

Ziel

...Elimination statisch unerreichbarer Knoten und Kanten als Optimierungstransformation OT zur

- Elimination statisch unerreichbarer Anweisungen als spezieller Klasse $\mathcal{K}$ unnötiger Anweisungen

zusammen mit dem Nachweis, dass OT

- Programmsemantik und beobachtbares Verhalten

in einem bestimmten Sinn erhält und für die Elimination unnötiger Anweisungen aus $\mathcal{K}$

- korrekt, vollständig und optimal ist.

Beobachtb. Verhalten, zu erhaltende Semantik

Beobachtbares Verhalten:

- Mengen von Programmzuständen an Programmpunkten.

Definition 15.2.1.3 (Semantische Äquivalenz)

Zwei Programme $G = (N, E, \mathbf{s})$ und $G' = (N', E', \mathbf{s}')$ heißen semantisch äquivalent bzgl. der nichtdeterministischen Aufsammelsemantik (oder: streichäquivalent) (in Zeichen: $G \approx_{[\![\ ]\!]} G'$) gdw:

1. $N' \subseteq N$, $E' \subseteq E$, $\mathbf{s} = \mathbf{s}'$
2. $\forall\, n \in N'.\ [\![\, n \,]\!]^{CS}_{G'} = [\![\, n \,]\!]^{CS}_{G} \quad (\subseteq \Sigma)$
3. $\forall\, n \in N \backslash N'.\ [\![\, n \,]\!]^{CS}_{G} = \emptyset$

Beobachtungsäquivalenz

Definition 15.2.1.4 (Beobachtungsäquivalenz)

Zwei Programme G und G' heißen beobachtungsäquivalent (in Zeichen: $G \approx_{\mathcal{B}} G'$) gdw G und G' sind streichäquivalent: $G \approx_{[\![\,]\!]} G'$.

Lemma 15.2.1.5 (Beobachtungsäquivalenz)

Seien $G = (N, E, \mathbf{s})$ und $G' = (N', E', \mathbf{s}')$ zwei Programme. Dann sind äquivalent:

1. G und G' sind beobachtungsäquivalent ($G \approx_{\mathcal{B}} G'$).
2. G und G' sind streichäquivalent ($G \approx_{[\![\,]\!]} G'$).
3. $(\forall\, n \in N \cap N'.\ \mathbf{P}_{G'}[\mathbf{s}', n] = \mathbf{P}_G[\mathbf{s}, n]) \wedge$
   $(\forall\, n \in N \backslash (N \cap N').\ \mathbf{P}_G[\mathbf{s}, n] = \emptyset) \wedge$
   $(\forall\, n \in N' \backslash (N \cap N').\ \mathbf{P}_{G'}[\mathbf{s}, n] = \emptyset)$

Unnötige Anweisungen, Knoten und Kanten

Definition 15.2.1.6 (Unnötig)

Seien $G = (N, E, \mathbf{s})$ und $G' = (N', E', \mathbf{s}')$ zwei beobachtungsäquivalente Programme mit $N' \subseteq N$, $E' \subseteq E$ und $\mathbf{s} = \mathbf{s}'$.

Dann heißt

1. eine Anweisung in G unnötig, wenn sie einen Knoten $n \in N \backslash N'$ benennt.
2. ein Knoten $n \in N$ unnötig, wenn er mit einer unnötigen Anweisung benannt ist.
3. eine Kante $e \in E$ unnötig, wenn ihr Anfangsknoten unnötig ist.

Klasse $\mathcal{K}$ unnötiger Anweisungen

Definition 15.2.1.7 (Klasse $\mathcal{K}$ unnötiger Anw.)

1. $\mathcal{K}$ ist die Klasse von Anweisungen, Knoten und Kanten von Programmen, die unnötig sind im Sinn von Definition 15.2.1.6.
2. Eine Anweisung, Knoten oder Kante aus $\mathcal{K}$ heißt $\mathcal{K}$-unnötige Anweisung, $\mathcal{K}$-unnötiger Knoten oder $\mathcal{K}$-unnötige Kante (oder $\mathcal{K}$-unnötig).

Bezeichne für ein Programm $G \in \mathcal{G}$:

- $\mathcal{U}_G^A$: Die Menge $\mathcal{K}$-unnötiger Anweisungen in G.
- $\mathcal{U}_G^N$: Die Menge $\mathcal{K}$-unnötiger Knoten in G.
- $\mathcal{U}_G^E$: Die Menge $\mathcal{K}$-unnötiger Kanten in G.

$\mathcal{K}$-Korrektheitstheorem

Theorem 15.2.1.8 ($\mathcal{K}$-Korrektheit)

Seien $G = (N, E, \mathbf{s})$ und $G' = (N', E\,\mathbf{s}')$ zwei Programme mit:

- $N' \subseteq N$, $E' \subseteq E$, $\mathbf{s} = \mathbf{s}'$
- $\forall\, n \in N \backslash N'.\ n \in \mathcal{U}_G^N$
- $\forall\, e \in E \backslash E'.\ n \in \mathcal{U}_G^E$

Dann gilt:

1. G und G' sind streich- und beobachtungsäquivalent:

$$G \approx_{[\![\ ]\!]} G' \ \wedge\ G \approx_{\mathcal{B}} G$$

2. G' enthält höchstens so viele $\mathcal{K}$-unnötige Anweisungen wie G:

$$\mathcal{U}_{G'}^A \subseteq \mathcal{U}_G^A$$

Eliminationstransformationen

Definition 15.2.1.9 (Eliminationstransformation)

1. Eine Programmtransformation, die angewendet auf ein Programm $G = (N, E, \mathbf{s})$ ein Programm $G' = (N', E', \mathbf{s}')$ liefert mit $N' \subseteq N$, $E' \subseteq E$ und $\mathbf{s} = \mathbf{s}'$ heißt Eliminationstransformation.
2. Ist ET eine Eliminationstransformation und G ein Programm, so bezeichnet G_{ET} dasjenige Programm, das aus der Anwendung von ET auf G entsteht.
3. Die Menge aller Eliminationstransformationen wird mit $\mathcal{ET}$ bezeichnet.

Korrekte Eliminationstransformationen

...korrekte Transformationen erzeugen beobachtungsäquivalente Programme.

Definition 15.2.1.10 (Korrekte Eliminationstransf.)

1. Eine Eliminationstransformation $ET \in \mathcal{ET}$ heißt korrekt gdw für alle Programme G gilt, dass G und G_{ET} beobachtungsäquivalent sind:

$$G \approx_{\mathcal{B}} G_{ET} \ (\Leftrightarrow G \approx_{[\![\ ]\!]} G_{ET})$$

2. Die Menge aller korrekten Eliminationstransformationen wird mit $\mathcal{ET}_{korr}$ bezeichnet.

$\mathcal{K}$-vollständige Eliminationstransformationen

...$\mathcal{K}$-vollständige Transformationen eliminieren alle $\mathcal{K}$-unnötigen Anweisungen.

Definition 15.2.1.11 ($\mathcal{K}$-vollst. Eliminationstransf.)

1. Eine Eliminationstransformation $ET \in \mathcal{ET}$ heißt $\mathcal{K}$-vollständig gdw für alle Programme G gilt, dass G_{ET} frei von $\mathcal{K}$-unnötigen Anweisungen ist:

$$\mathcal{U}^A_{G_{ET}} = \emptyset$$

2. Die Menge aller $\mathcal{K}$-vollständigen Eliminationstransformationen wird mit $\mathcal{ET}_{\mathcal{K}\text{-vollst}}$ bezeichnet.

$\mathcal{K}$-optimale Eliminationstransformationen

...$\mathcal{K}$-optimale Transformationen eliminieren alle und ausschließlich $\mathcal{K}$-unnötige Anweisungen.

Definition 15.2.1.12 ($\mathcal{K}$-optimale Eliminationstr.)

1. Eine Eliminationstransformation $ET \in \mathcal{ET}$ heißt $\mathcal{K}$-optimal gdw ET ist korrekt und $\mathcal{K}$-vollständig:

$$\forall\, G \in \mathcal{G}.\ G \approx_{\mathcal{B}} G_{ET} \ \wedge\ \mathcal{U}^{A}_{G_{ET}} = \emptyset$$

2. Die Menge aller $\mathcal{K}$-optimalen Eliminationstransformationen wird mit $\mathcal{ET}_{\mathcal{K}\text{-}opt}$ bezeichnet.

Besser als, best

Definition 15.2.1.13 (Bessere Eliminationstransf.)

Seien $ET, ET' \in \mathcal{ET}_{korr}$ zwei korrekte Eliminationstransformationen. Dann heißt ET mindestens so gut wie (oder besser als) ET' gdw für alle Programme G gilt: $N_{G_{ET}} \subseteq N_{G_{ET'}}$ und $E_{G_{ET}} \subseteq E_{G_{ET'}}$

Definition 15.2.1.14 (Beste Eliminationstransf.)

1. Eine korrekte Eliminationstransformation $ET \in \mathcal{ET}_{korr}$ heißt best gdw ET ist besser als jede andere zulässige Eliminationstransformation $ET' \in \mathcal{ET}_{korr}$.
2. Die Menge aller besten Eliminationstransformationen wird mit $\mathcal{ET}_{best}$ bezeichnet.

Best impliziert $\mathcal{K}$-optimal und umgekehrt

Lemma 15.2.1.15 ($\mathcal{K}$-Unnötigkeitsfreiheit)

Sei $ET \in \mathcal{ET}_{best}$ eine beste Eliminationstransformation. Dann gilt für alle Programme G, dass G_{ET} frei von $\mathcal{K}$-unnötigen Anweisungen ist:

$$\mathcal{U}^A_{G_{ET}} = \emptyset$$

Korollar 15.2.1.16 ($\mathcal{K}$-Optimalität)

Beste Eliminationstransformationen sind $\mathcal{K}$-optimal und umgekehrt:

$$\forall\, ET \in \mathcal{ET}.\ ET \in \mathcal{ET}_{best} \ \Leftrightarrow\ ET \in \mathcal{ET}_{\mathcal{K}\text{-}opt}$$

Noch zu tun

...konkrete Angabe einer (Eliminations-) Transformation

$$OT : \mathcal{G} \to \mathcal{G}$$

und der Nachweis:

$$OT \in \mathcal{ET}_{\mathcal{K}\text{-}opt}$$

Arbeitsplan:

1. Statische Erreichbarkeitsanalysen induzieren Eliminationstransformationen.
2. Korrekte und vollständige Erreichbarkeitsanalysen induzieren $\mathcal{K}$-optimale Eliminationstransformationen.

Charakterisierung stat. unerreichbarer Knoten

Proposition 15.2.1.17 (Äquivalenz)

Sei $n \in N \backslash \{\mathbf{s}, \mathbf{e}\}$ ein Knoten. Dann sind äquivalent:

1. n ist statisch unerreichbar.
2. Alle in n eingehenden Kanten sind statisch unerreichbar.
3. Alle von n ausgehenden Kanten sind statisch unerreichbar.

Proposition 15.2.1.18 (Implikation)

Sei $n \in N$ ein Knoten. Dann gilt:

1. Ist n statisch unerreichbar, dann gilt $n \neq \mathbf{s}$.
2. Die Umkehrung von Teil 1) gilt nicht.

Proposition 15.2.1.19 (Speziell)

s ist statisch erreichbar.

Charakterisierung stat. unerreichbarer Kanten

Proposition 15.2.1.20 (Äquivalenz)

Sei $e \in E$ eine Kante. Dann sind äquivalent:

1. e ist statisch unerreichbar.
2. Der Anfangsknoten von e ist statisch unerreichbar.

Proposition 15.2.1.21 (Implikation)

Sei $e \in E$ eine Kante. Dann gilt:

1. Ist der Endknoten von e statisch unerreichbar, dann ist e statisch unerreichbar.
2. Die Umkehrung von Teil 1) gilt nicht.

Proposition 15.2.1.22 (Speziell)

Gilt $src(e) = \mathbf{s}$, so ist e statisch erreichbar.

Charakterisierung von Unnötigkeit

...durch statische Unerreichbarkeit.

Lemma 15.2.1.23 (Äquivalenz)

Seien $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Programm. Dann gilt:

1. Eine Anweisung α in G ist unnötig gdw α ist statisch unerreichbar.
2. Ein Knoten $n \in N$ ist unnötig gdw n ist statisch unerreichbar.
3. Eine Kante $e \in E$ ist unnötig gdw e ist statisch unerreichbar.

Korollar 15.2.1.24 (Unnötige Knoten, Kanten)

1. Ein statisch unerreichbarer Knoten ist unnötig.
2. Eine statisch unerreichbare Kante ist unnötig.

Korrekte, vollständige Erreichbarkeitsanalysen

Definition 15.2.1.25 (Erreichbarkeitsanalyse)

Eine Analyse, die angewendet auf ein Programm $G = (N, E, \mathbf{s})$ einige Knoten und Kanten als erreichbar markiert, heißt Erreichbarkeitsanalyse.

Definition 15.2.1.26 (Korrekte Erreichb.-Analyse)

Eine Erreichbarkeitsanalyse *EA* heißt korrekt gdw ein Knoten oder eine Kante von *EA* als erreichbar gekennzeichnet worden ist, dann ist dieser Knoten oder Kante statisch erreichbar.

Definition 15.2.1.27 (Vollständige Erreichb.-Analyse)

Eine Erreichbarkeitsanalyse *EA* heißt vollständig gdw ein Knoten oder eine Kante statisch erreichbar ist, dann ist dieser Knoten oder Kante von *EA* als erreichbar gekennzeichnet worden.

Optimale Erreichbarkeitsanalysen

Definition 15.2.1.28 (Optimale Erreichb.-Analyse)

Eine Erreichbarkeitsanalyse *EA* heißt optimal gdw *EA* korrekt und vollständig ist.

Korollar 15.2.1.29 (Statische Unerreichbarkeit)

Sei *EA* eine optimale Erreichbarkeitsanalyse. Dann gilt: Ein Knoten oder eine Kante ist statisch unerreichbar gdw dieser Knoten bzw. Kante von *EA* nicht als erreichbar markiert worden ist.

Induzierte Eliminationstransformation

...einer Erreichbarkeitsanalyse.

Definition 15.2.1.30 (Induzierte Eliminationstransf.)

Eine Erreichbarkeitsanalyse EA induziert eine Eliminationstransformation ET_{EA}, die angewendet auf ein Programm G alle Knoten und Kanten in G streicht, die von EA nicht als erreichbar markiert worden sind.

Optimale Erreichbarkeitsanalyse O_{EA}

...ohne die Analyse im Detail auszuführen, ist offensichtlich, dass wir eine Erreichbarkeitsanalyse O_{EA} so realisieren können, dass für O_{EA} gilt:

Lemma 15.2.1.31 (Optimalität von O_{EA})

O_{EA} ist optimal, d.h. O_{EA} ist korrekt und vollständig.

Optimierungstransformation OT

...zur Elimination unnötiger Anweisungen.

Definition 15.2.1.32 (Optimierung)

Das Optimierungsverfahren zur Elimination $\mathcal{K}$-unnötiger Anweisungen besteht aus zwei Stufen:

1. Analysestufe: Erreichbarkeitsanalyse in Graphen mittels einer Erreichbarkeitsanalyse O_{EA}, O_{EA} optimal.
2. Transformationsstufe: Die von O_{EA} induzierte Eliminationstransformation $ET_{O_{EA}}$, die alle von O_{EA} nicht als erreichbar erkannten Knoten und Kanten streicht.

Die Optimierung aus Analyse- und Transformationsstufe werde bezeichnet mit:

$$OT =_{df} ET_{O_{EA}}$$

Korrektheit, Vollständigkeit, Optimalität

...von OT zur Elimination $\mathcal{K}$-unnötiger Anweisungen.

Lemma 15.2.1.33 (Korrektheit)

OT ist

1. korrekt ($OT \in \mathcal{ET}_{korr}$).
2. vollständig ($OT \in \mathcal{ET}_{\mathcal{K}\text{-}vollst}$).
3. best ($OT \in \mathcal{ET}_{best}\ (= \mathcal{ET}_{korr} \cap \mathcal{ET}_{\mathcal{K}\text{-}vollst})$).

Korollar 15.2.1.34 (Optimalität)

OT ist $\mathcal{K}$-optimal ($OT \in \mathcal{ET}_{\mathcal{K}\text{-}opt}\ (= \mathcal{ET}_{best})$).

Übungsaufgabe 15.2.1.35

...Beobachtungs- und äquivalent dazu Streichäquivalenz erhalten die Semantik nicht nur im Aufsammelsinn, sondern pfadweise (oder: pfadgenau).

Zeige: Zwei Programme G und G' sind streichäquivalent gdw G und G' beschreiben pfadweise dieselbe Zustandstransformation:

$$G \approx_{[\![\,]\!]} G' \text{ gdw}$$

1. $\forall \in N_G \cap N_{G'}.\ \forall\, p \in \mathbf{P}_G[\mathbf{s}, n) \cup \mathbf{P}_G[\mathbf{s}, n]\, \forall \sigma \in \Sigma.$
$$[\![\, p \,]\!]_G(\sigma) = [\![\, p' \,]\!]_{G'}(\sigma)$$
2. $\forall \in N_G \setminus N_{G'}.\ \mathbf{P}_G[\mathbf{s}, n) = \emptyset = \mathbf{P}_G[\mathbf{s}, n]$
3. $\forall \in N_{G'} \setminus N_G.\ \mathbf{P}_{G'}[\mathbf{s}, n) = \emptyset = \mathbf{P}_{G'}[\mathbf{s}, n]$

wobei p und p' sich entsprechende Pfade in G und G' sind.

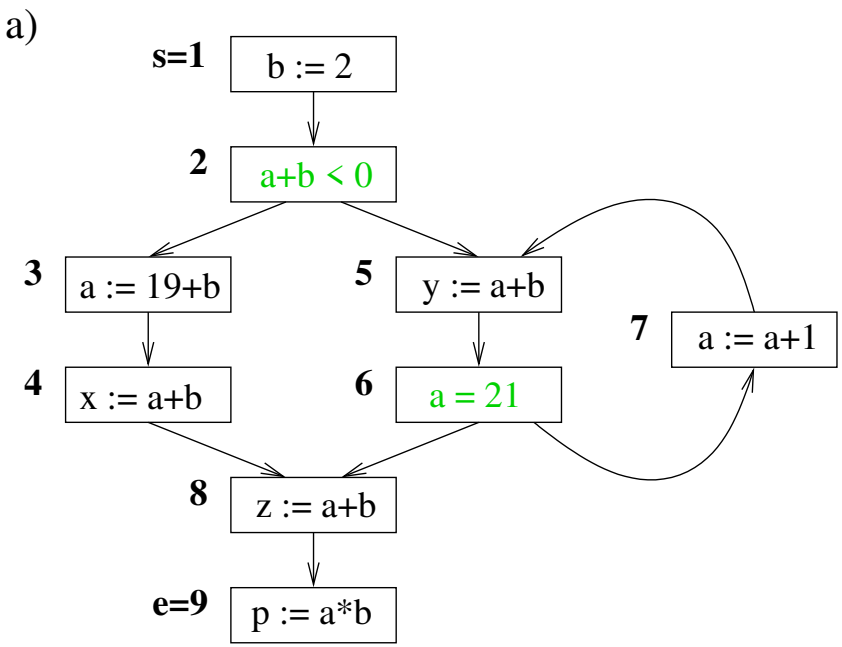
Kapitel 15.2.2

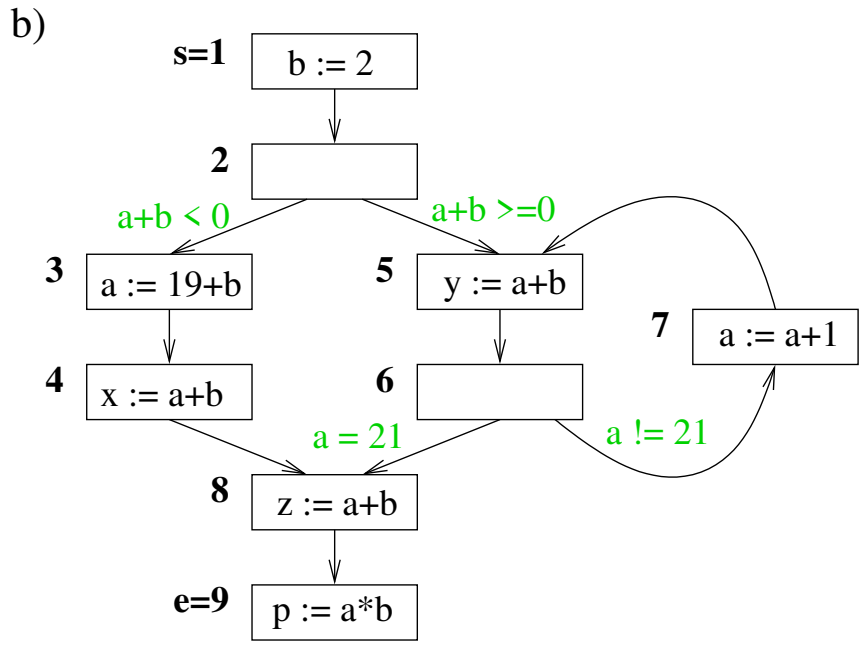
Dynamisch unerreichbare Anweisungen

Informell

...ein Knoten $n \in N$ ist dynamisch unerreichbar gdw n von einer mit einer Bedingung $b \in \mathbf{Bexpr}$ benannten Kante $e \in E$ dominiert wird, die nie erfüllt ist, d.h. für 'keinen an e möglichen Programmzustand wahr' ist, d.h.:

$$\forall \sigma \in \Sigma.\ [\![b]\!]_B([\![src(e)]\!]_{\mathrm{WHILE}}(\sigma)) = \mathbf{falsch}$$





Dynamisch unerreichbare Anweisungen

...eine 'semantische' Eigenschaft.

Definition 15.2.2.1 (Dynamisch unerreichbare Anw.)

Eine Anweisung α am Knoten $n \in N$ in G ist dynamisch unerreichbar gdw n (und damit α) dynamisch unerreichbar ist.

Aus der Unentscheidbarkeit des Konstantenproblems (s. Kapitel 7.10.2, `if x=0 then ... else ... fi`) folgt unmittelbar:

Lemma 15.2.2.2 (Unentscheidbarkeit)

Dynamische Unerreichbarkeit von Anweisungen ist unentscheidbar.

Entscheidbare Teilklassen dyn. Unerreichbark.

...Lemma 15.2.2.2 erfordert es eingeschränkte entscheidbare Klassen $\mathcal{K}$ dynamisch unerreichbarer Anweisungen α zu identifizieren, für die gilt:

> Ist α $\mathcal{K}$-unerreichbar, dann ist α dynamisch unerreichbar.

Die Identifikation möglicher Kandidaten für entscheidbare Klassen $\mathcal{K}$ kann an entscheidbaren Teilklassen des Konstantenproblems ansetzen:

Eine Anweisung α am Knoten n mit einer mit b benannten dominierenden Bedingungskante e ist

- $\mathcal{K}_{einfK}$-unerreichbar, wenn b eine einfache Konstante
- $\mathcal{K}_{endlK}$-unerreichbar, wenn b eine endliche Konstante
- $\mathcal{K}_{polyK}$-unerreichbar, wenn b eine polynomiale Konstante
- ...

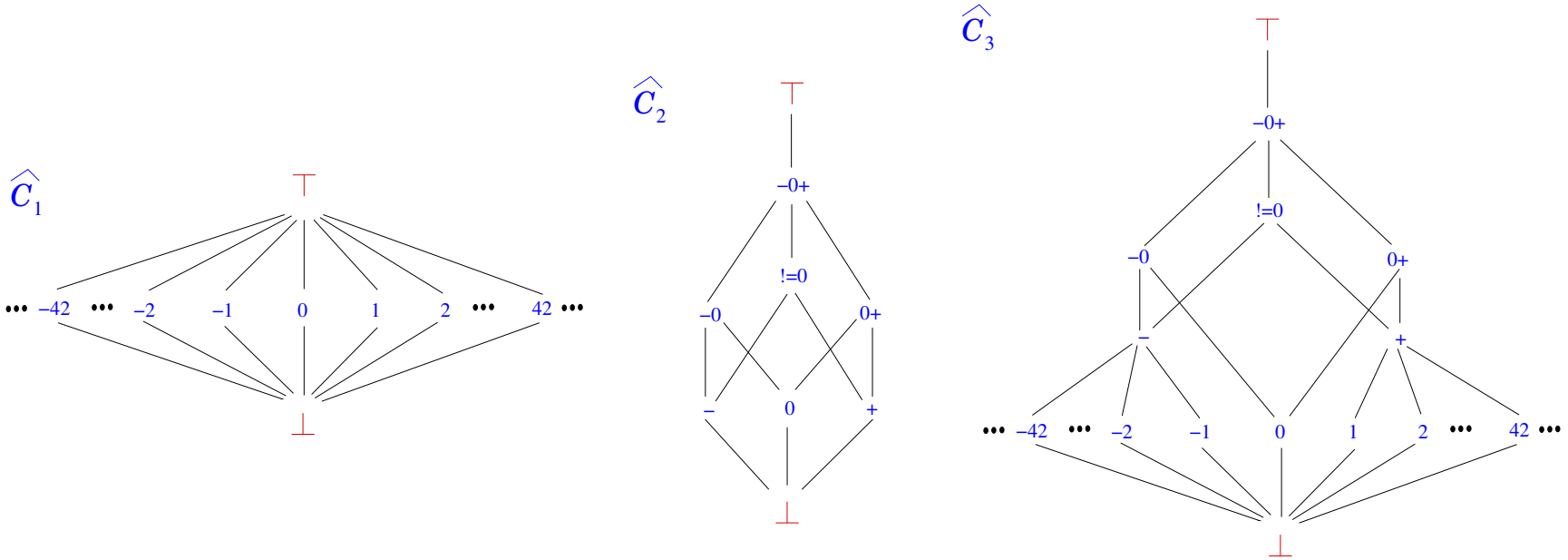
vom Wert **wahr** ist.

Übungsaufgabe 15.2.2.3

Führe die Idee der $\mathcal{K}$-Unerreichbarkeit von Anweisungen nach dem Vorbild statisch unerreichbarer Anweisungen aus Kapitel 15.2.1 im Detail für Konstanten- oder/und Vorzeichenanalysen über folgenden (Grund-) Verbänden und Zustandsmengen

$$\Sigma =_{df} \{\sigma \mid \sigma : \mathbf{V} \to \mathcal{C}_i\},\ i \in \{1,2,3\}$$

aus:



Kapitel 15.2.3

Senken, Sackgassen und schwarze Löcher

Senken, Sackgassen, schwarze Löcher (1)

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Programm.

Definition 15.2.3.1 (Statisch liegen in)

Ein Knoten $n \in N$ liegt statisch in

1. einer Senke gdw $\mathbf{e}$ ist auf keinem Pfad von n aus erreichbar, d.h. wenn: $\mathbf{P}_G[n, \mathbf{e}] = \emptyset$.
2. einem schwarzen Loch gdw n liegt statisch in einer Senke und es gilt: $\bigcup_{m \in N} \{\mathbf{P}_G[n, m]\}$ ist unendlich.
3. einer Sackgasse gdw n liegt statisch in einer Senke und es gilt: $\bigcup_{m \in N} \{\mathbf{P}_G[n, m]\}$ ist endlich.

Senken, Sackgassen, schwarze Löcher (2)

Definition 15.2.3.2 (Statische Senken, etc.)

1. Eine statische Senke ist eine Menge von Knoten, die statisch in einer Senke liegen.
2. Ein statisches schwarzes Loch ist eine Menge von Knoten, die statisch in einem schwarzen Loch liegen.
3. Eine statische Sackgasse ist eine Menge von Knoten, die statisch in einer Sackgasse liegen.

Wir bezeichen mit N_G^{st-S}, N_G^{st-sL} und N_G^{st-Sg} die Mengen aller Knoten eines Programms G, die in einer statischen Senke, einem statischen schwarzen Loch bzw. einer statischen Sackgasse von G liegen.

Eigensch. v. Senken, Sackg., schw. Löchern (1)

Es ist leicht einzusehen:

Proposition 15.2.3.3

Liegt $n \in N$ in

1. einer statischen Senke, so kann n statisch erreichbar sein oder nicht.
2. einem statischen schwarzen Loch, so
   - 2.1 ist von n aus ein Knoten m erreichbar, der in einer Schleife liegt, d.h.: $\mathbf{P}_G[n, m] \neq \emptyset$ und $\mathbf{P}_G[m, m]$ unendlich.
   - 2.2 sind fast alle (d.h. bis auf endlich viele) von n ausgehenden Pfade unendlich lang.
3. einer statischen Sackgasse, so ist jeder von n ausgehende Pfad endlich lang.

Eigensch. v. Senken, Sackg., schw. Löchern (2)

Proposition 15.2.3.4

Die Knotenmengen statischer schwarzer Löcher und Sackgassen $N_G^{st\text{-}sL}$ und $N_G^{st\text{-}Sg}$ partitionieren die Knotenmenge $N_G^{st\text{-}S}$ statischer Senken eines Programms G, d.h.:

$$N_G^{st\text{-}S} = N_G^{stsL} \dot{\cup} N_G^{st\text{-}Sg}$$

Rückbetrachtung

...unserer Generalvereinbarung (oder Generalvoraussetzung) für Programme $G = (N, E, \mathbf{s}, \mathbf{e})$ aus Kapitel 7:

- Jeder Knoten $n \in N$ liegt auf einem Pfad von **s** nach **e**

unter den Aspekten

- Erhaltung von Semantik
- Gewährleistung von Beobachtungsäquivalenz

bezüglich des möglicherweise nötigen Streichens von Knoten und Kanten zur Etablierung der Generalvoraussetzung für ein Programm.

Zerlegung der Generalvorauss. für Programme

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Programm. Dann gilt:

Proposition 15.2.3.5

Folgende Aussagen sind äquivalent:

1. G erfüllt die Generalvoraussetzung.
2. $\forall\, n \in N.\ \mathbf{P}_G[\mathbf{s}, n] \neq \emptyset \wedge \mathbf{P}_G[n, \mathbf{e}] \neq \emptyset$

D.h., für Programme gemäß der Generalvoraussetzung gilt:

Korollar 15.2.3.6

G erfüllt die Generalvoraussetzung gdw:

1. Alle Knoten in G sind statisch erreichbar.
2. Kein Knoten von G liegt in einer statischen Senke.

Zusicherbarkeit der Generalvoraussetzung

...offenbar ist es leicht möglich, jedes Programm (oder Flussgraphen) $G = (N, E, \mathbf{s}, \mathbf{e})$ so zu transformieren, dass die Generalvoraussetzung erfüllt ist:

Elimination

- statisch unerreichbarer Knoten: Siehe Kapitel 15.2.1.
- der Knoten statischer Senken: Anwendung des Konzepts statischer Unerreichbarkeit aus Kapitel 15.2.1 auf den reversen Flussgraphen $G_{rev} = (N, E_{rev}, \mathbf{e}, \mathbf{s})$ von G mit

$$E_{rev} =_{df} \{(n, m) \mid (m, n) \in E\}$$

OBdA kann deshalb angenommen werden, dass die Generalvoraussetzung von allen Programmen erfüllt ist.

Allerdings

...die Konzepte statisch unerreichbarer Knoten und von Knoten statischer Senken addressieren unterschiedliche Konzepte von Unnötigkeit von Anweisungen:

1. Unnötig, weil statisch nicht erreichbar (und damit definitiv unausführbar zur Laufzeit des Programms).
2. Unnötig, weil in statischer Sackgasse oder schwarzem Loch gefangen (und damit definitiv unausführbar zur Laufzeit des Programms im Zuge einer regulär am Endknoten terminierenden Ausführung).

Daraus folgt

...die Elimination von Anweisungen

- statisch unerreichbarer Knoten erhält jede (vernünftige) Form von Programmsemantik und gewährleistet jede (vernünftige) Form von Beobachtungsäquivalenz bei Streichen solcher Knoten und inzidierender Kanten.

Für die Elimination von Anweisungen

- in statischen Senken gilt dies nur für Laufzeitausführungen des Programms entlang von Pfaden in $\mathbf{P}_G[\mathbf{s}, \mathbf{e}]$.

Sackgassen und schwarze Löcher

...können 'Licht' emittieren.

Enthalten Knoten in dynamisch erreichbaren statischen Senken Ausgabeanweisungen, so terminieren entsprechende Laufzeitausführungen zwar nie regulär am Endknoten des Programms, aber erzeugen (möglicherweise sogar unendlich viel)

- beobachtbares Verhalten.

Das Streichen von Knoten (und damit Anweisungen) in statischen Senken ist damit anders als das Streichen statisch unerreichbarer Knoten (und damit Anweisungen)

- willkürlich

und eine (hoffentlich)

- bewusste Designentscheidung.

Ob die Designentscheidung

...für das Streichen von Senken und damit die Außerachtlassung nicht regulär terminierender Ausführungen zur Sicherstellung des zweiten Teils der Generalvoraussetzung für Programme aus Kapitel 7 unter den Aspekten Erhaltung von Semantik und Beobachtungsäquivalenz gerechtfertigt ist, kann einzig im Anwendungskontext begründet sein. Vergleiche z.B.:

- Semi-Entscheidungsverfahren (möglicherweise gerechtfertigt).

und

- Steuerungsprogramme reaktiver Systeme (vermutlich nicht oder nie gerechtfertigt).

Zu guter Letzt: In Programmen ohne goto-Anweisung ist die Existenz statischer Senken ein Indikator fehlerhafter Flussgraphkonstruktion und sollte Anlass einer Fehlermeldung sein.

Übungsaufgabe 15.2.3.7

Analog zu dynamisch unerreichbaren Anweisungen gibt es dynamisch unerreichbare Sackgassen und schwarze Löcher.

1. Übertrage die Konzepte und Überlegungen für statisch unerreichbare Sackgassen und schwarze Löcher auf ihre dynamischen Gegenstücke und arbeite sie aus. Was gilt weiterhin? Was stellt sich möglicherweise anders dar?
2. Was gilt für die Entscheidbarkeit dynamischer Sackgassen und schwarzer Löcher?
3. Welches Vorgehen oder welche Methoden bieten sich zur korrekten approximativen Berechnung dynamischer Sackgassen und schwarzer Löcher an?
4. Arbeite eine dieser Methoden in größerem Detail aus.

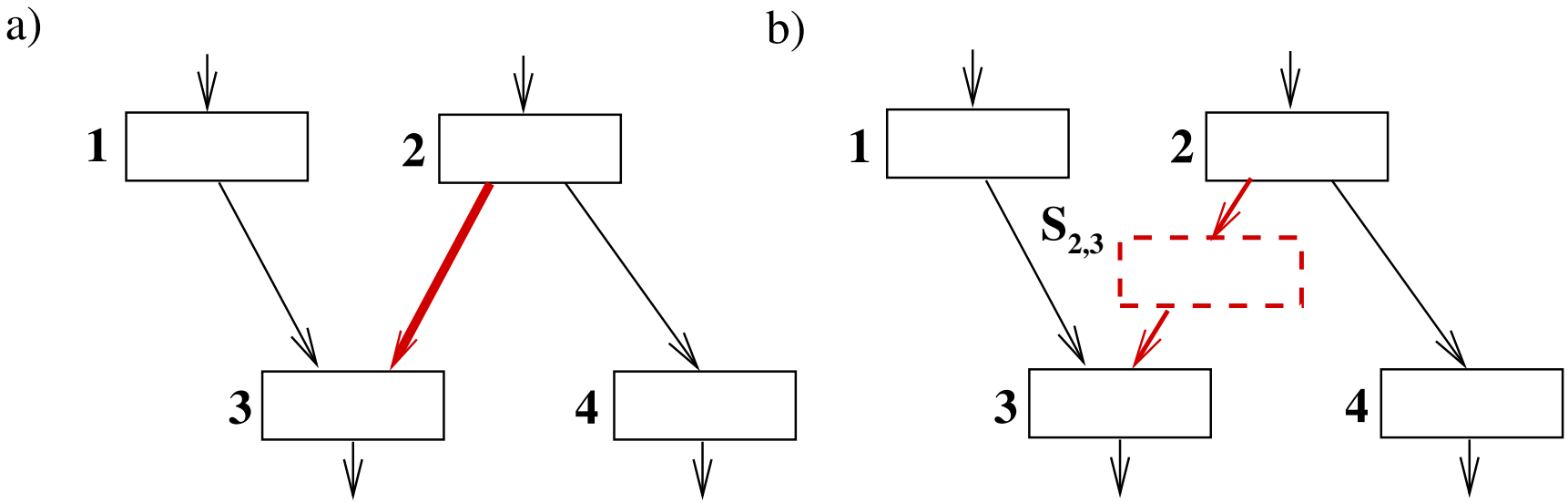
Kapitel 15.3

Partiell tote und geisterhafte Anweisungen

Kritische Kanten

Definition 15.3.1 (Kritische Kanten)

Eine Kante heißt kritisch gdw sie von einem Knoten mit mehr als einem Nachfolger zu einem Knoten mit mehr als einem Vorgänger führt (s. Abb. a)).



...kritische Kanten können durch Spalten und Einfügen eines sog. synthetischen Knotens beseitigt werden (s. Abb. b)).

Illustration: Kritische Kanten (1)

...können die Elimination unnötiger Anweisungen verhindern:

- Abb. a): Der Wert von x aus der Zuweisung in Knoten 2 wird nur für Programmfortsetzungen über Knoten 3 benötigt, nicht über Knoten 4.
- Abb. b) und c): Beide Transformationen verändern das beobachtbare Verhalten und sind daher nicht korrekt.

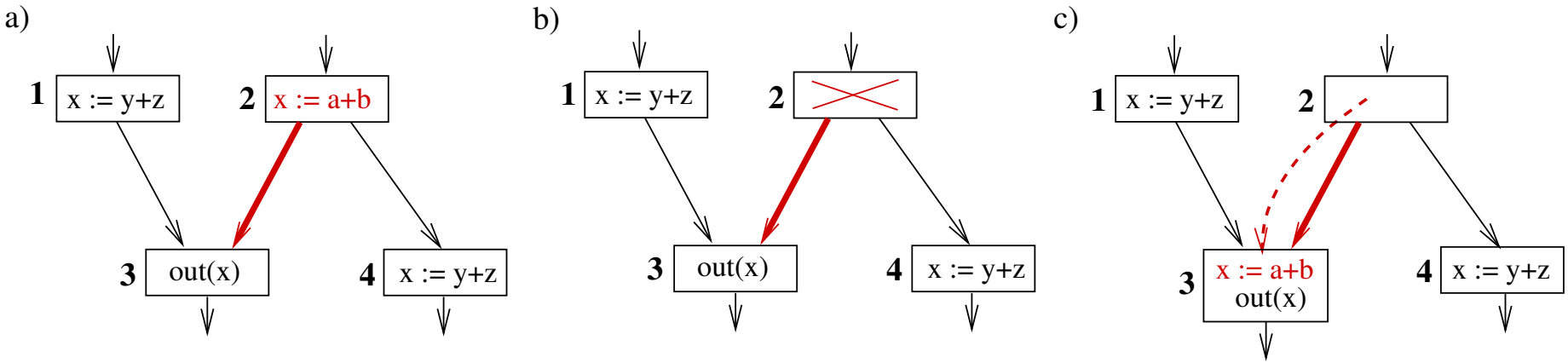
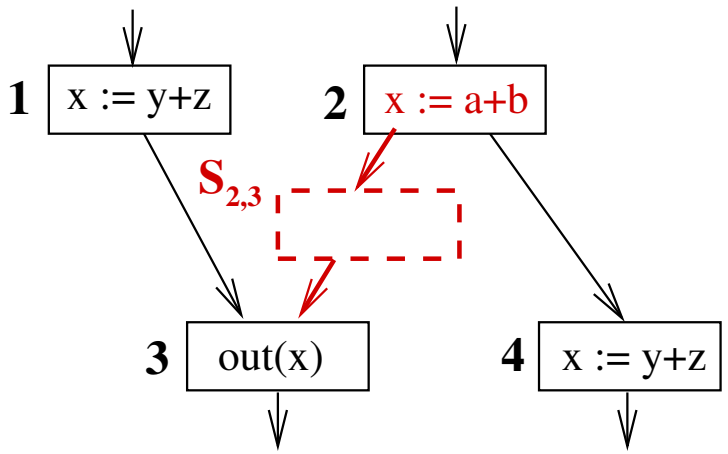


Illustration: Kritische Kanten (2)

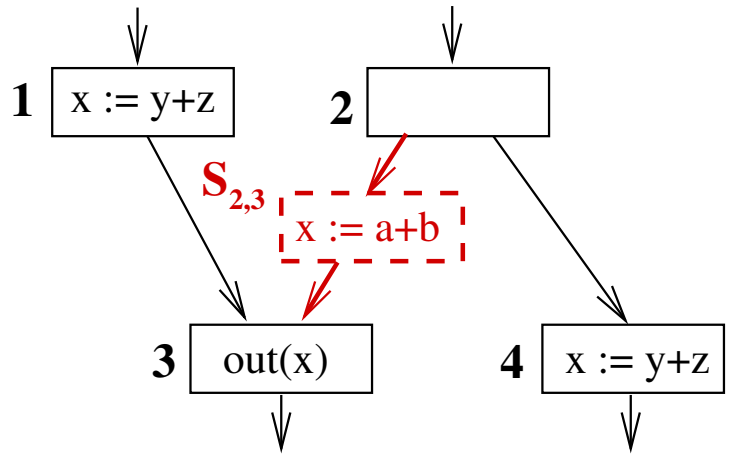
Spalten kritischer Kanten:

- Abb. a): Die kritische Kante wird durch Einfügen des synthetischen Knotens $S_{2,3}$ gespalten und beseitigt.
- Abb. b): Die Transformation verbessert die Performanz, ohne das beobachtbare Verhalten zu verändern.

a)

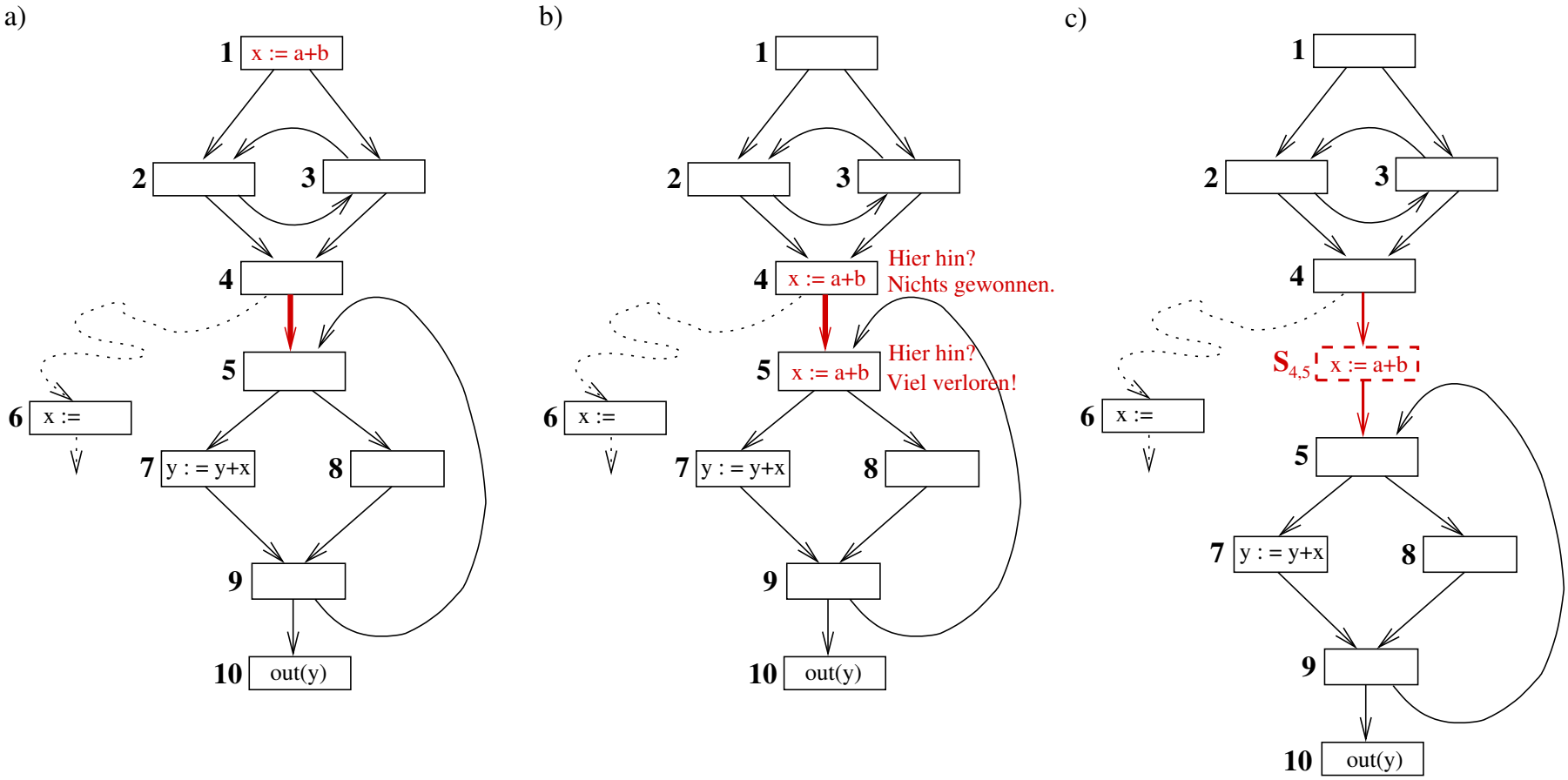


b)



Beachte: Die performanzverbessernde Transformation wird erst durch das Spalten der kritischen Kante möglich.

Kritische Kanten in einem größeren Beispiel



...auch hier ist performanzverbessernde Transformation erst durch das Spalten der kritischen Kante möglich.

Vereinbarung zu kritischen Kanten

...um bestmögliche Transformationsresultate zu ermöglichen, insbesondere garantieren zu können, niemals Anweisungen in Schleifen zu verschieben, nehmen wir an, dass jede

- kritische Kante

in einem Flussgraphen durch Einfügen eines (synthetischen) Knotens gespalten und dadurch beseitigt ist.

Zusatzvereinbarungen für Flussgraphen

...für Kapitel 15.3 (und 15.4).

Für jeden Flussgraphen $G = (N, E, \mathbf{s}, \mathbf{e}) \in \mathcal{G}$ gilt:

- $\mathbf{s} \in N$ hat keine Vorgänger: $pred(\mathbf{s}) = \emptyset$
- $\mathbf{e} \in N$ hat keine Nachfolger: $succ(\mathbf{e}) = \emptyset$
- Jeder Knoten $n \in N$
  - liegt auf einem Pfad von **s** nach **e**.
  - ist mit einer Instruktion (Zuweisung, Schreibanweisung, Bedingung) benannt (keine Basisblöcke).
- Keine Kante ist kritisch.

s und **e** bezeichnen als Start- und Endknoten ausgezeichnete Knoten in G.

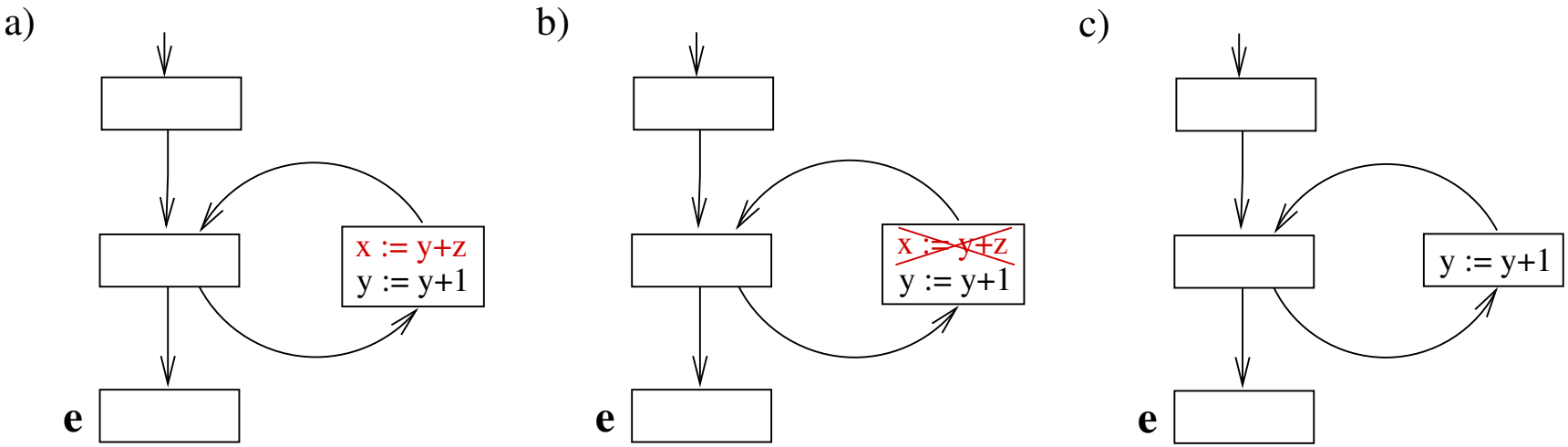
Kapitel 15.3.1

Motivation

Tote Anweisungen

...und ihre Elimination.

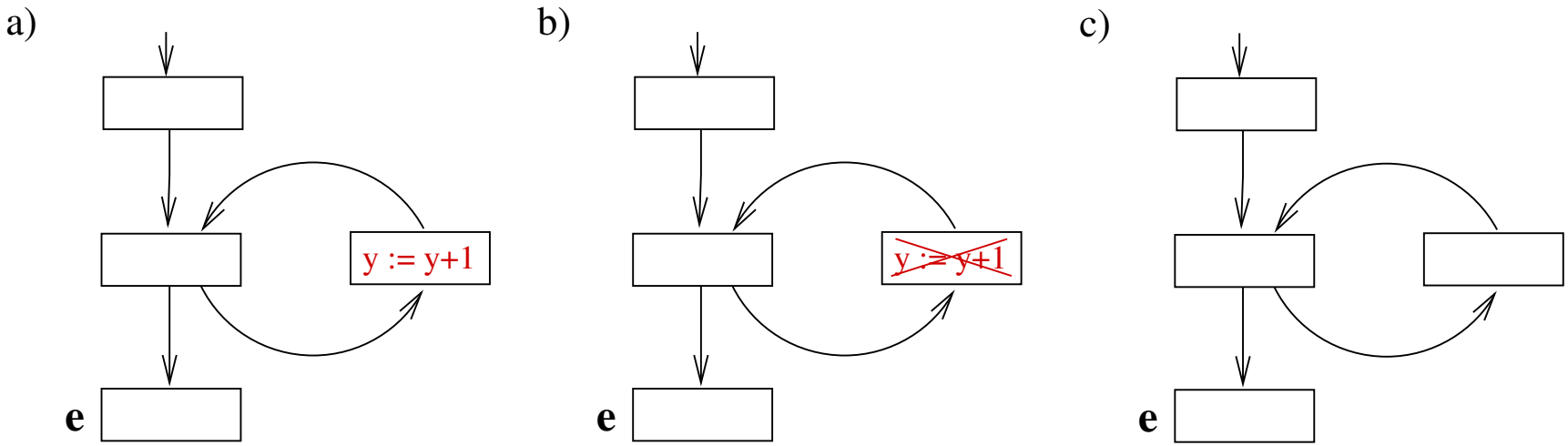
Das Grundmuster: Die Anweisung `x := y+z` ist tot (oder total tot) (engl. (totally) dead): `x` zugewiesene Werte werden an keiner Stelle im Programm gelesen.



Geisterhafte Anweisungen

.....und ihre Elimination.

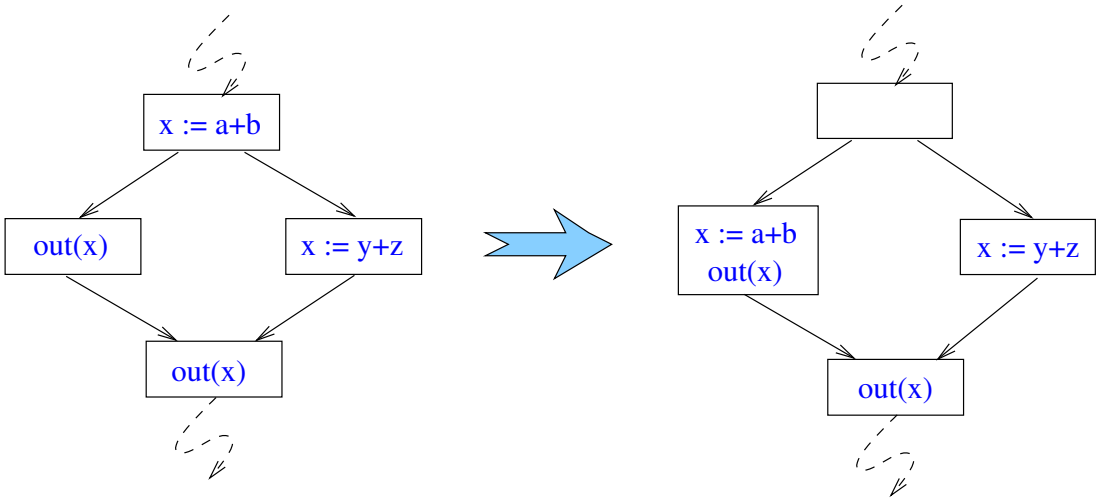
Das Grundmuster: Die Anweisung y := y+1 ist nicht tot, sondern lebendig (engl. live), aber geisterhaft (engl. faint): y zugewiesene Werte beeinflussen weder direkt noch indirekt Ausgabe- oder Bedingungswerte (und damit das beobachtbare Programmverhalten).



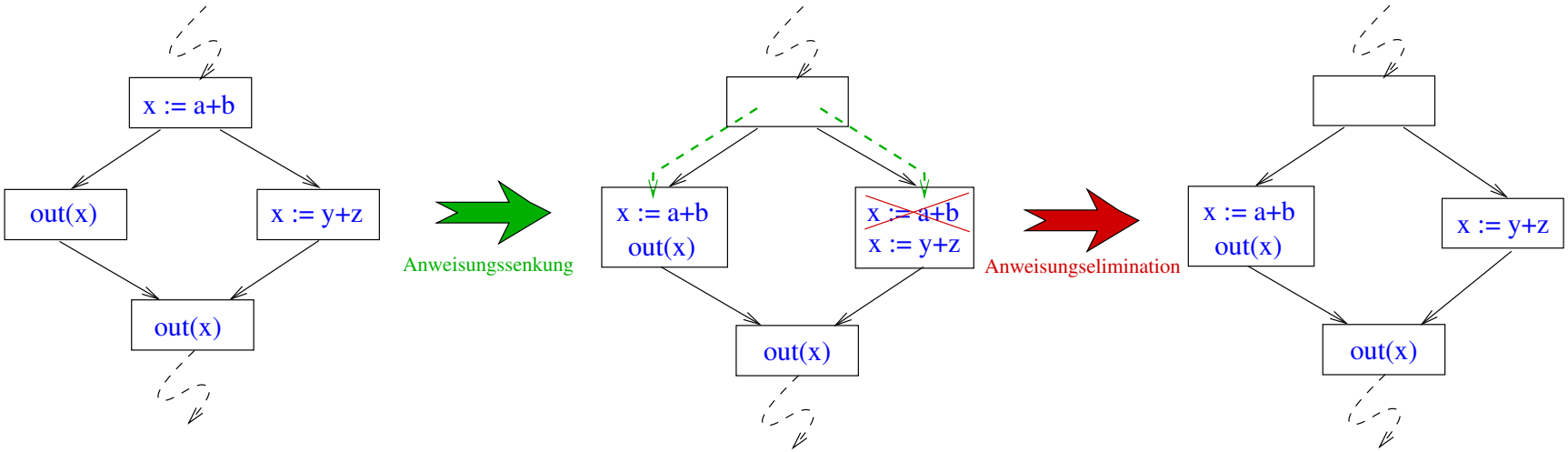
Partiell tote Anweisungen

...und ihre Elimination.

Das Grundmuster:



Die konzeptuelle Verfahrensidee:

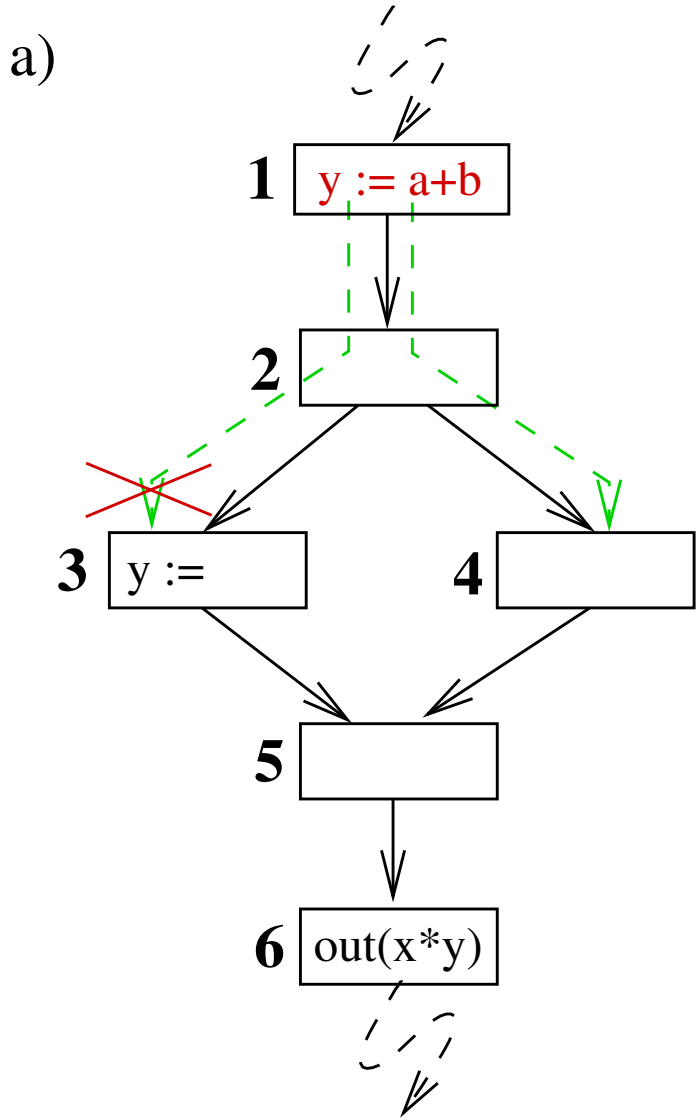


Kapitel 15.3.2

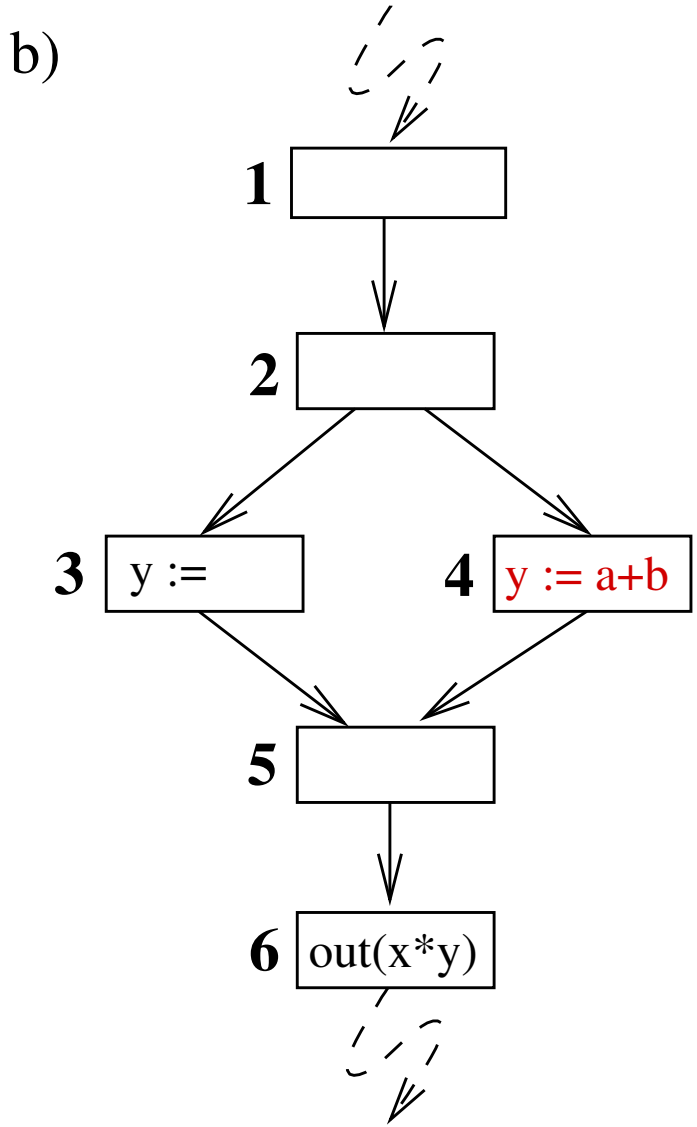
Beispiele

Bsp. 1: Elimination partiell toter Anweisungen

Ausgangsprogramm:

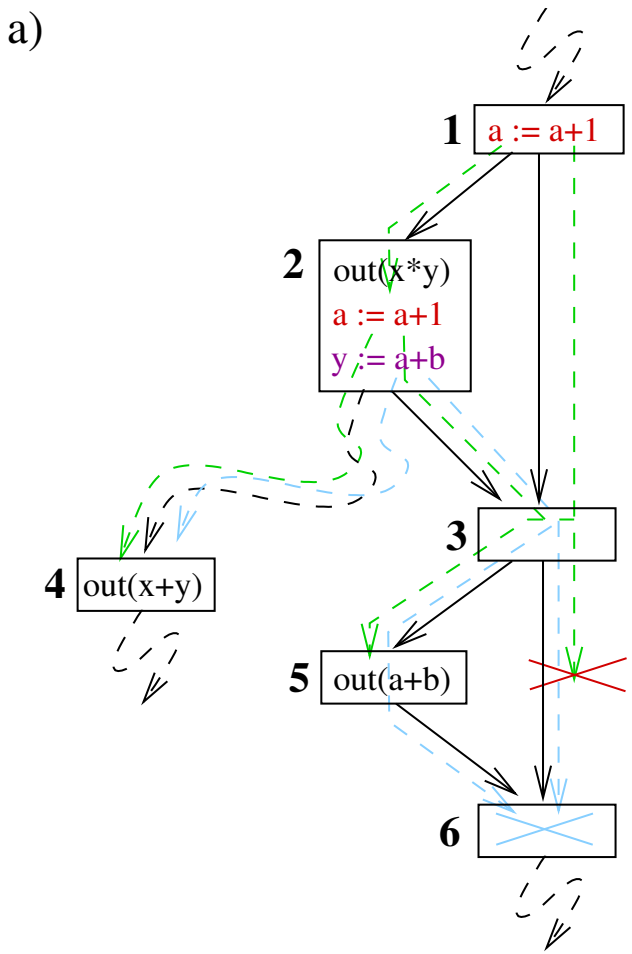


Optimiertes Programm:

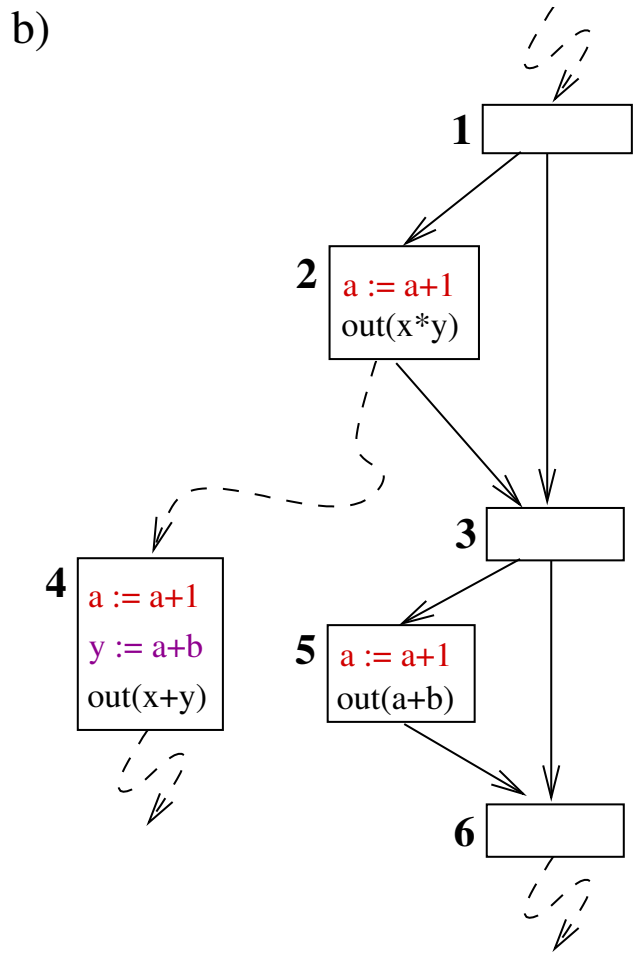


Bsp. 2: Elimination partiell toter Anweisungen

Ausgangsprogramm:



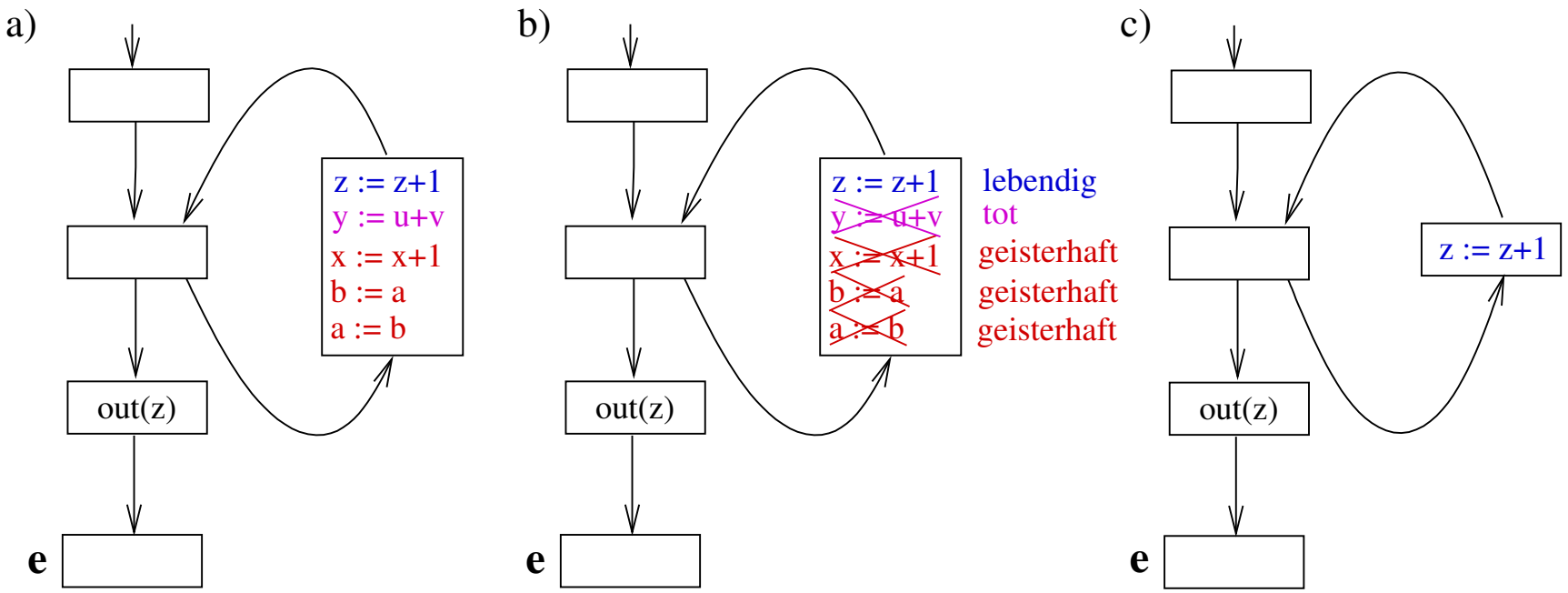
Optimiertes Programm:



...ist i.a. eine 'm2n'-Transformation (hier für $\alpha \equiv a := a + 1$).

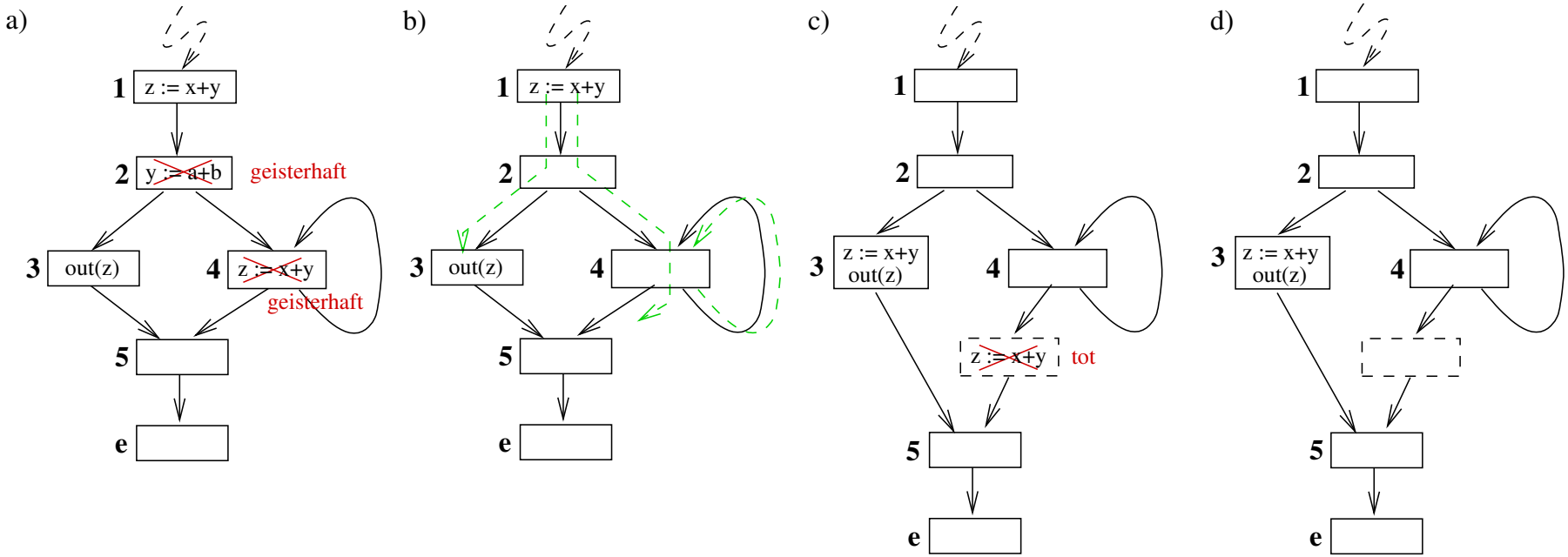
Bsp. 3: Elimination geisterhafter Anweisungen

Ausgangsprogramm:



Bsp. 4: Elimination geisterhafter Anweisungen

Ausgangs-Prg.: Geisteranw.-Elim.: Anw.-Senkung: Elim. toter Anw.:



Anmerkung

...'echt' partiell geisterhafte Anweisungen gibt es nicht:

Anweisungen, die auf

- jeder Programmfortsetzung geisterhaft oder tot sind, sind geisterhaft.
- mindestens einer Programmfortsetzung weder geisterhaft noch tot sind, sind lebendig (nicht 'partiell geisterhaft').

Die Elimination geisterhafter Anweisungen

- kann aber durch die Beseitigung von Senkungsblockaden die Elimination weiterer partiell toter Anweisungen ermöglichen.

In diesem Sinne ist hier

- Elimination partiell geisterhafter Anweisungen

zu verstehen (s. Bsp. 3)).

Elimination partiell toter/geisterhafter Anw.

...zwei verschiedene (Optimierungs-) Transformationen:

- EPTA: Elimination partiell toter Anweisungen
  $\rightsquigarrow$ Partial Dead-Code Elimination (PDCE)
- EPGA: Elimination partiell geisterhafter Anweisungen
  $\rightsquigarrow$ Partial Faint-Code Elimination (PFCE)

als Wiederholung von 3 Elementartransformationen:

- ETA: Elimination toter Anweisungen
  $\rightsquigarrow$ Dead-Code Elimination (DCE)
- EGA: Elimination geisterhafter Anweisungen
  $\rightsquigarrow$ Faint-Code Elimination (FCE)
- AS: Anweisungssenkungen
  $\rightsquigarrow$ Assignment Sinking (AS)

wobei AS-Schritte (immer wieder) Potential für E-Schritte schaffen (sog. Effekte 2. Ordnung).

Kapitel 15.3.3

Elementartransformationen

Tote und geisterhafte Anweisungen

Definition 15.3.3.1 (Tote Anweisung)

Eine Anweisung $x := t$ am Knoten n ist tot gdw auf jeder Programmfortsetzung bis zum Endknoten gilt:

- x wird nicht gelesen oder
- dem ersten Lesen von x geht ein Schreiben von x voraus.

Definition 15.3.3.2 (Geisterhafte Anweisung)

Eine Anweisung $x := t$ am Knoten n ist geisterhaft gdw auf jeder Programmfortsetzung bis zum Endknoten gilt:

- x wird nicht gelesen oder
- dem ersten Lesen von x geht ein Schreiben von x voraus oder
- x wird rechtsseitig in einer selbst geisterhaften Anweisung gelesen.

Tot impliziert geisterhaft

Proposition 15.3.3.3

Tote Anweisungen sind geisterhaft.

Beachte: Die Umkehrung von Proposition 15.3.3.3 gilt i.a. nicht.

Anweisungseliminationen

Definition 15.3.3.4 (Anweisungselimination)

Eine Programmtransformation, die einige Anweisungen aus dem Programm streicht, heißt Anweisungselimination (engl. code elimination).

Definition 15.3.3.5 (t/g-Anweisungselimination)

Eine Anweisungselimination, die einige

1. tote Anweisungen aus dem Programm streicht, heißt t-Anweisungselimination (engl. dead-code elimination).
2. geisterhafte Anweisungen aus dem Programm streicht, heißt g-Anweisungselimination (engl. faint-code elimination).

Definition 15.3.3.6 (Korrekte Anweisungselim.)

Eine Anweisungselimination ist korrekt, wenn sie eine t- oder g-Anweisungselimination ist.

Anweisungssenkungen

Definition 15.3.3.7 (Anweisungssenkung)

Eine Anweisungssenkung für ein Anweisungsmuster $\alpha \equiv x := t$ (oder α-Anweisungssenkung) ist das simultane stetige Verschieben eines oder mehrerer Vorkommen von α in Richtung des Kontrollflusses zu einem oder mehreren anderen Knoten.

Definition 15.3.3.8 (Korrekte Anweisungssenkung)

Eine α-Anweisungssenkung, $\alpha \equiv x := t$, ist korrekt, wenn zu jedem Zeitpunkt während des Schiebens gilt:

1. Kein α-Vorkommen wird über eine Anweisung hinweggeschoben, die x liest oder modifiziert oder einen Operanden von t modifiziert (und α dadurch blockiert).
2. Kein α-Vorkommen wird in einen Zusammenflussknoten geschoben, wenn dies nicht von jedem Vorgänger des Zusammenflussknotens aus geschieht.

Sprechweisen und Bezeichnungen

Definition 15.3.3.9 (Blockiert)

Ein Anweisung α der Form $x := t$ ist von einer Anweisung α' senkungsblockiert (oder: blockiert), wenn α'

- Variable x
  - liest ($\alpha' \equiv \ldots := \ldots x \ldots$) oder
  - modifiziert ($\alpha' \equiv x := \ldots$) oder einen
- Operanden von t modifiziert ($\alpha' \equiv y := \ldots$, $t \equiv \ldots y \ldots$).

Kapitel 15.3.4

Effekte zweiter Ordnung

Effekte zweiter Ordnung

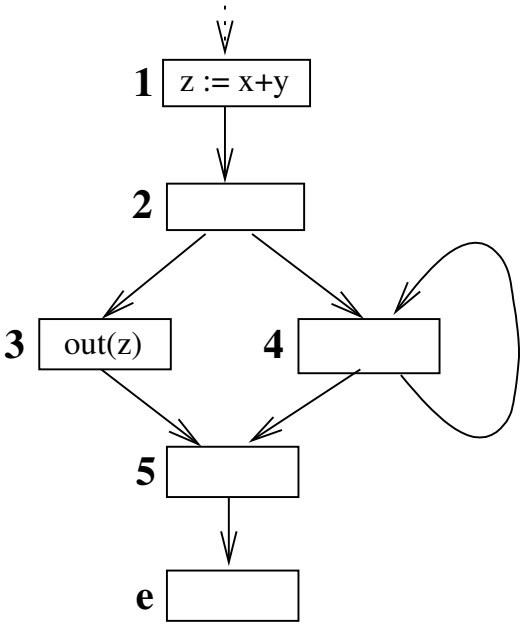
...(engl. second-order effects) treten in 4 Formen auf und sind essentiell für die kombinierte Wirkung der Elementartransformationen von EPTA und EPGA:

1. Senkungs-Eliminations-Effekte (Zieleffekt)
   $\rightsquigarrow$ Sinking-elimination effects
2. Senkungs-Senkungs-Effekte (Potentialeffekt)
   $\rightsquigarrow$ Sinking-sinking effects
3. Eliminations-Senkungs-Effekte (Potentialeffekt)
   $\rightsquigarrow$ Elimination-sinking effects
4. Eliminations-Eliminations-Effekte (Zieleffekt)
   $\rightsquigarrow$ Elimination-elimination effects

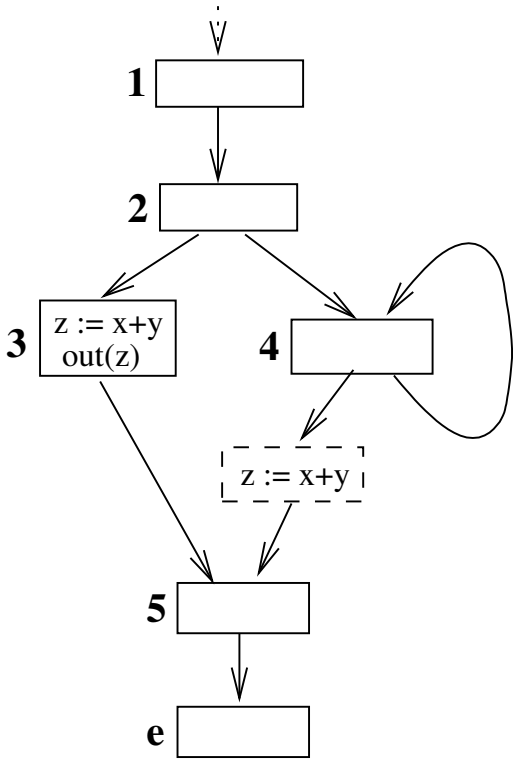
1) Senkungs-Eliminations-Effekt

...Zieleffekt: Eine Elementartransformation (hier: Senkung) ermöglicht anschließend eine Elimination (die erneut Senkungs- oder Eliminationspotential schaffen kann).

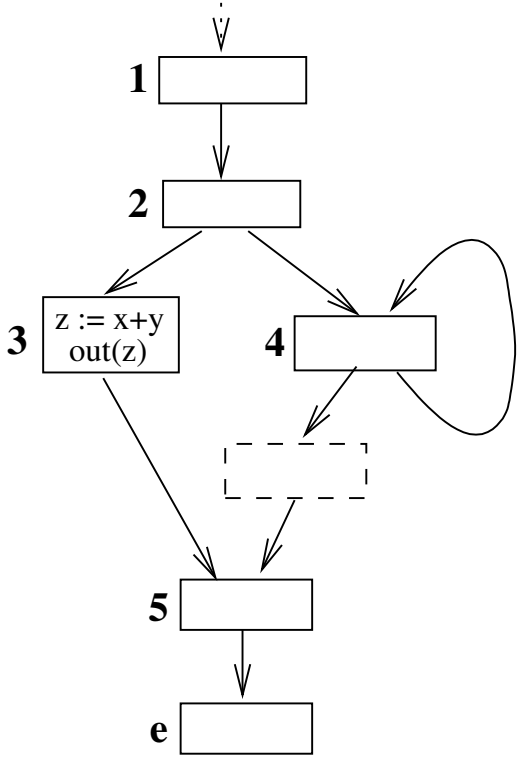
Ausgangsprogramm



Senkung 1. Ord.



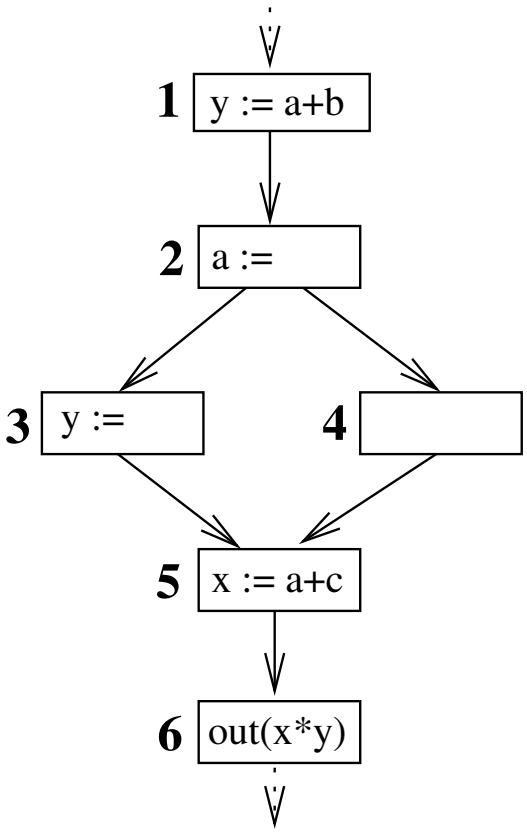
Elimination 2. Ord.



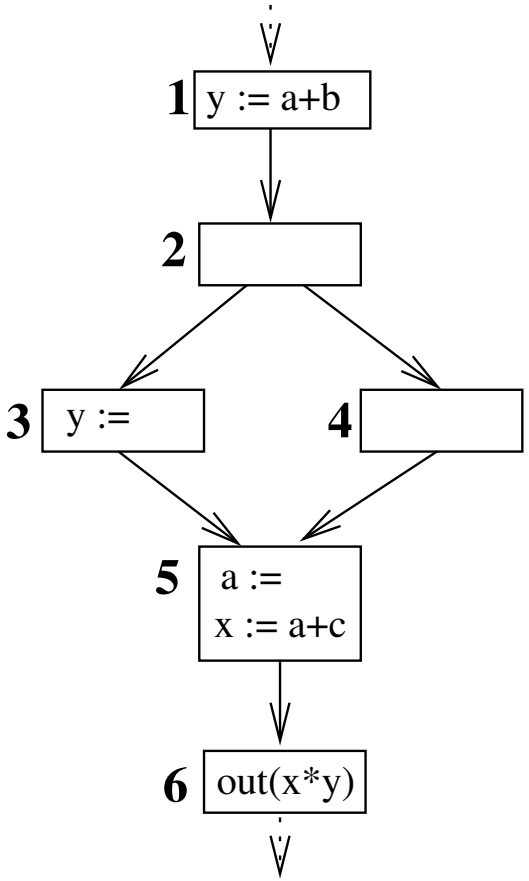
2) Senkungs-Senkungs-Effekt

...Potentialeffekt: Eine Elementartransformation (hier: Senkung) ermöglicht anschließend eine (weitere) Senkung, die erneut Senkungs- oder Eliminationspotential schaffen kann.

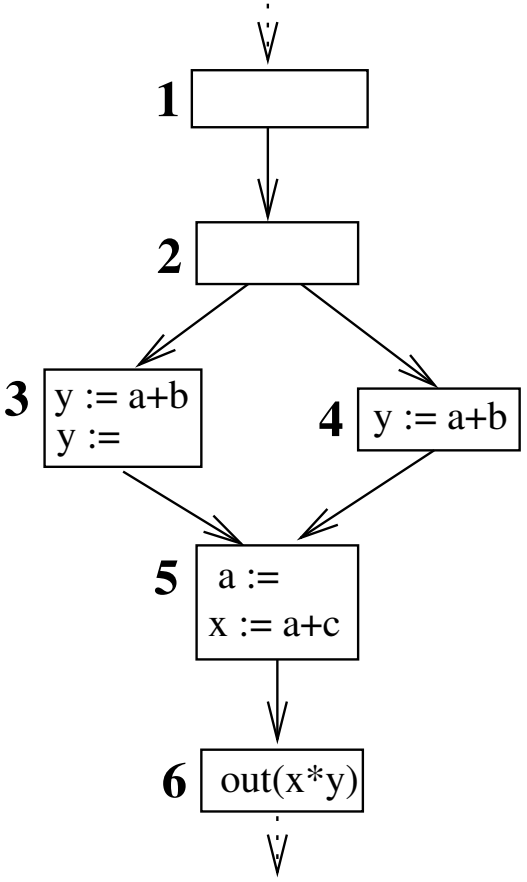
Ausgangsprogramm



Senkung 1. Ord.



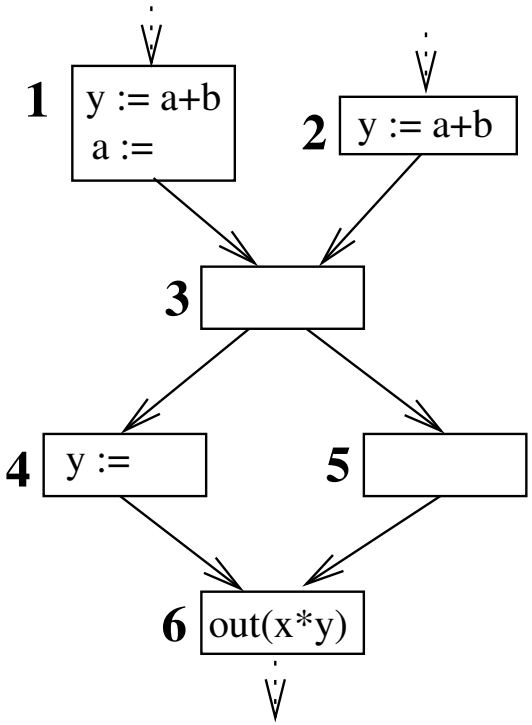
Senkung 2. Ord.



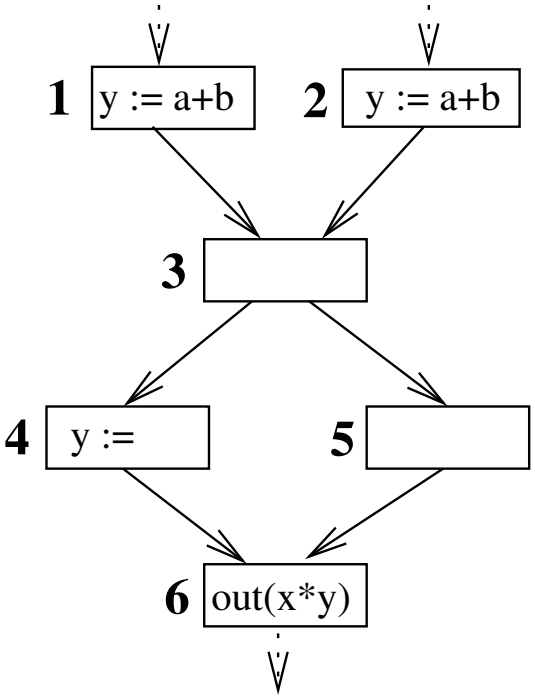
3) Eliminations-Senkungs-Effekt

...Potentialeffekt: Eine Elementartransformation (hier: Elimination) ermöglicht anschließend eine Senkung, die erneut Senkungs- oder Eliminationspotential schaffen kann.

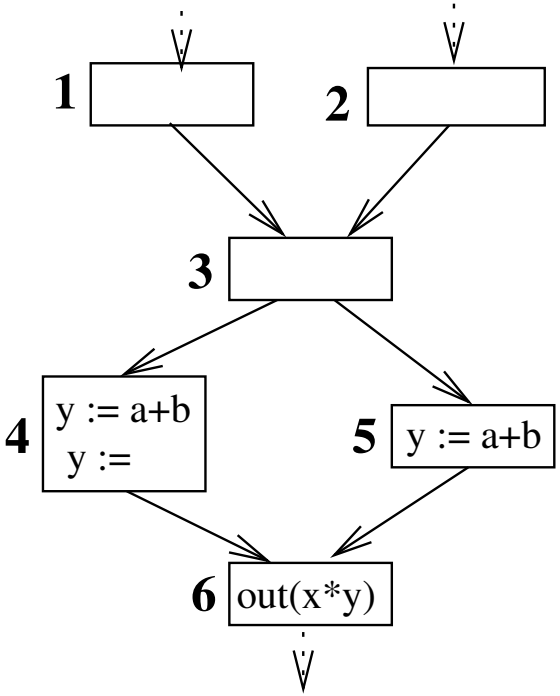
Ausgangsprogramm



Elimination 1. Ord.



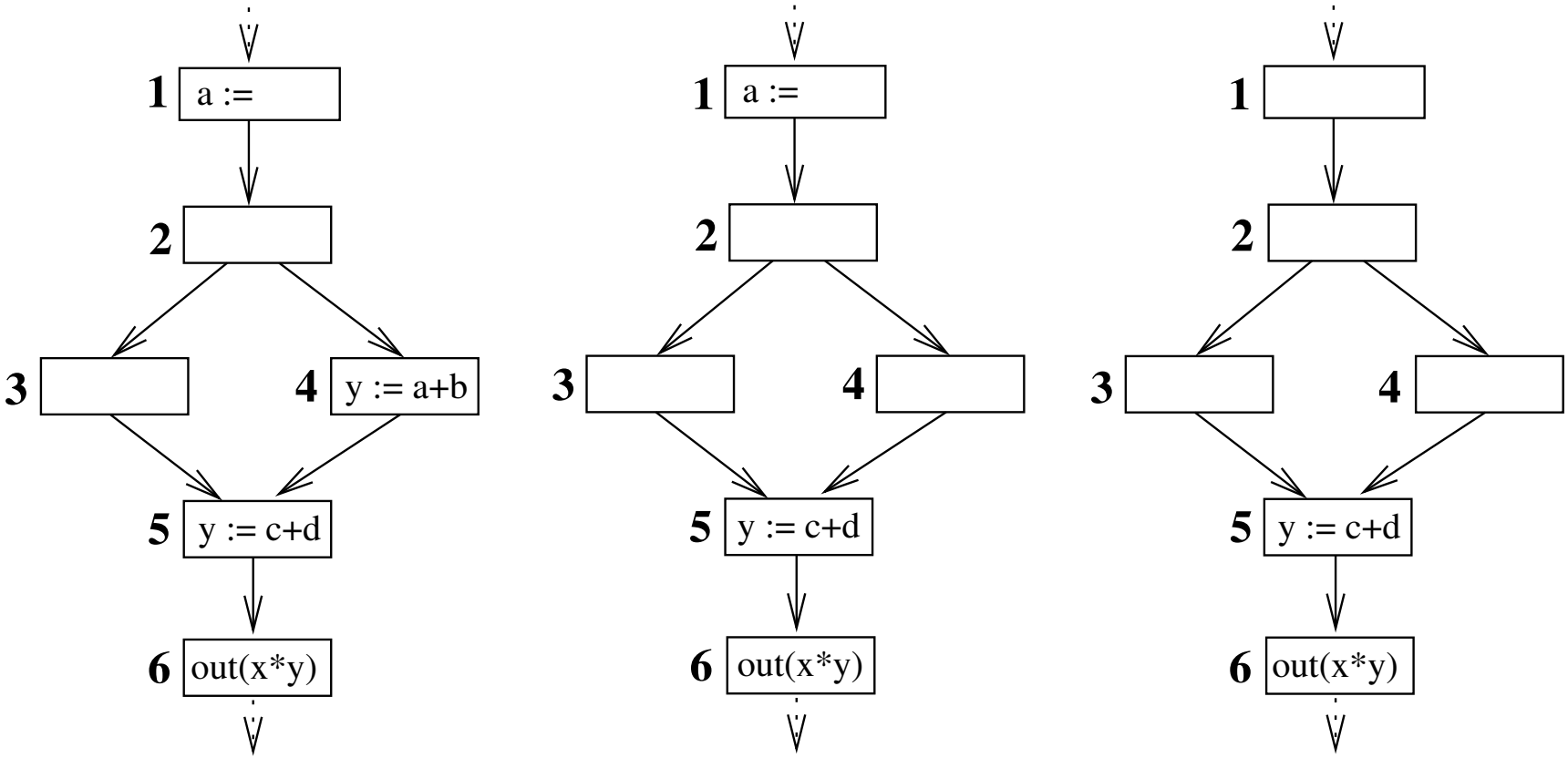
Senkung 2. Ord.



4) Eliminations-Eliminations-Effekt

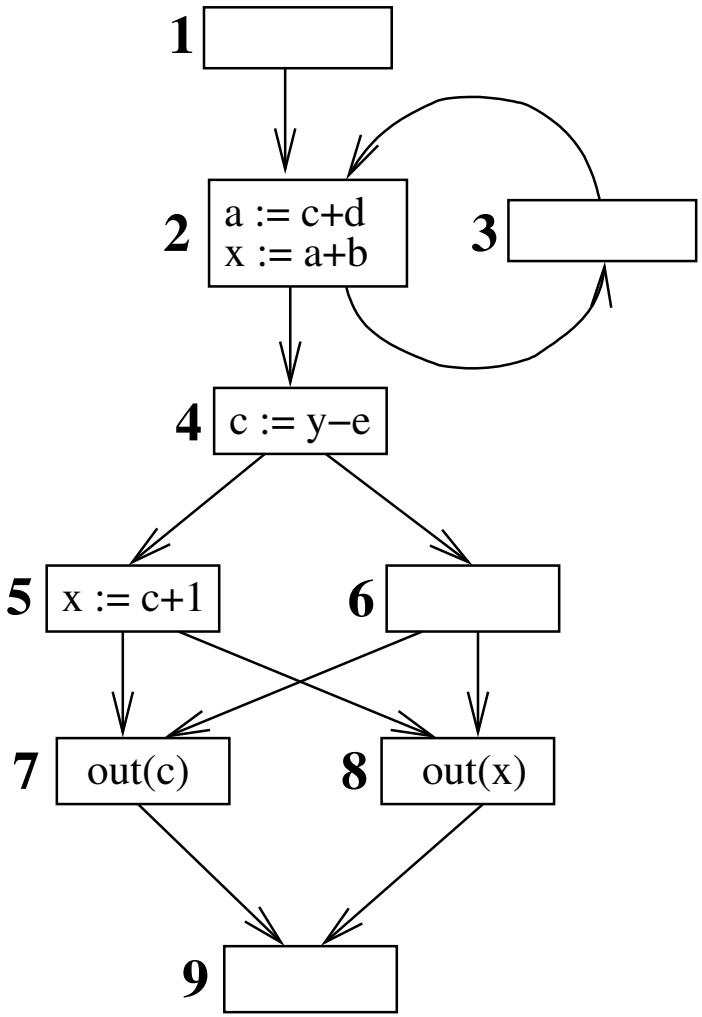
...Zieleffekt: Eine Elementartransformation (hier: Elimination) ermöglicht anschließend eine (weitere) Elimination (die erneut Senkungs- oder Eliminationspotential schaffen kann).

Ausgangsprogramm Elimination 1. Ord. Elimination 2. Ord.

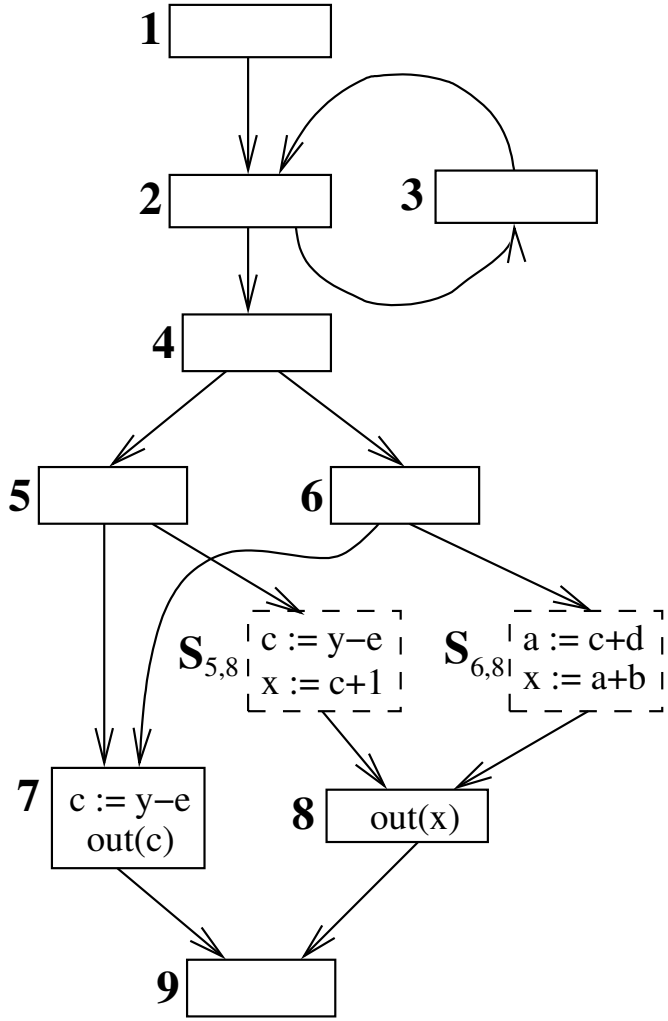


Sequenzwirkung von Effekten 2. Ordnung

Ausgangsprogramm



Optimiertes Programm



Kapitel 15.3.5

EPTA/EPGA: Transformationen

Unnötige Anweisungen

Sei G ein Programm.

Definition 15.3.5.1 (t-unnötige Anweisung)

Eine Anweisung α am Knoten n in G heißt t-unnötig gdw α ist tot am Knoten n.

Definition 15.3.5.2 (g-unnötige Anweisung)

Eine Anweisung α am Knoten n in G heißt g-unnötig gdw α ist geisterhaft am Knoten n.

Sprechweisen und Bezeichnungen (1)

Wir bezeichnen informell mit

- $AS \quad =_{df} \bigcup \{AS_\alpha^G \mid \alpha \in \mathcal{AM}, G \in \mathcal{G}\}$
- $ETA =_{df} \bigcup \{ETA_\alpha^G \mid \alpha \in \mathcal{AM}, G \in \mathcal{G}\}$
- $EGA =_{df} \bigcup \{EGA_\alpha^G \mid \alpha \in \mathcal{AM}, G \in \mathcal{G}\}$

die Mengen aller zulässigen Anweisungssenkungen und -eliminationen (für beliebige Programme und Anweisungsmuster).

Wir bezeichnen ebenso informell mit Wörtern der von den regulär-artigen Ausdrücken

$$(AS + ETA)^*, \ (AS + EGA)^*, \ (AS + ETA + EGA)^*$$

erzeugten Sprachen

$$\mathcal{L}((AS + ETA)^*), \ \mathcal{L}((AS + EGA)^*), \ \mathcal{L}((AS + ETA + EGA)^*)$$

Folgen zulässiger AS-, ETA- und EGA-Transformationen (für beliebige Programme und Anweisungsmuster).

Sprechweisen und Bezeichnungen (2)

Mit diesen Schreibweisen bezeichne:

- $T^G_{AS,ETA} = \{\tau \,|\, \tau = (\tau_i)_{i \in \mathbb{N}} \in \mathcal{L}((AS + ETA)^*)\}$
- $T^G_{AS,EGA} = \{\tau \,|\, \tau = (\tau_i)_{i \in \mathbb{N}} \in \mathcal{L}((AS + EGA)^*)\}$

die Menge aller Transformationsfolgen aus zulässigen AS- und ETA- bzw. EGA-Transformationen für ein Programm G.

Geht G aus dem Kontext hervor, schreiben wir statt $T^G_{AS,ETA}$ und $T^G_{AS,EGA}$ einfacher $T_{AS,ETA}$ und $T_{AS,EGA}$.

Ist $\tau = (\tau_i)_{i \in \mathbb{N}}$ eine Transformationsfolge, so bezeichne:

- $(\tau_i)_{i \leq k}$ das Anfangsstück von τ bis zum Index k einschließlich.
- τ_j die Elementartransformation mit Index j von τ .
- G_τ, $G_{(\tau_i)_{i \leq k}}$ und $G_{(\tau_j)}$ diejenigen Programme, die aus G durch Anwendung von τ, $(\tau_i)_{i \leq k}$, auf G entstehen.

EPTA- und EPGA-Transformationen

Sei G ein Programm.

Definition 15.3.5.3 (EPTA/EPGA-Transf.)

1. Eine EPTA-Transformation (EPGA-Transformation) ist eine beliebige Abfolge zulässiger Anweisungssenkungen und Eliminationen toter (geisterhafter) Anweisungen.
2. T^G_{EPTA} und T^G_{EPGA} bezeichnen die Mengen aller EPTA- und EPGA-Transformationen für G, in Zeichen:
   - $T^G_{EPTA} =_{df} T^G_{AS,ETA}$
   - $T^G_{EPGA} =_{df} T^G_{AS,EGA}$
3. Ist $\tau \in T^G_{EPTA} \cup T^G_{EPGA}$, so bezeichnet G_τ das Programm, das τ angewendet auf G liefert.

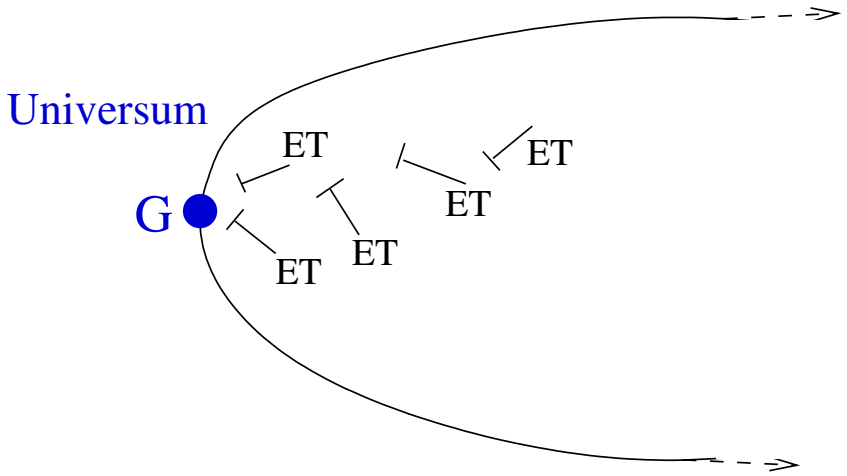
Transformationsrelation, Programmuniversum

- Transformationsrelation:

  $G \vdash_\tau G'$, $\tau \in AS \cup ETA \cup EGA$: G' resultiert aus G durch Anwendung von τ mit τ zulässige Senkungs- oder Eliminationstransformation toter/geisterhafter Anweisungen.

- Induziertes Programmuniversum:

  $\mathcal{U}_T^G =_{df} \{G' \mid G \vdash_{(\tau_i)_{i \le k}} G',\ \tau = (\tau_i)_{i \in \mathbb{N}} \in T, k \in \mathbb{N}\}$,
  $T = T^G_{EPTA}$ oder $T = T^G_{EPGA}$: Das von G durch die Präfixe der Transformationsfolgen $\tau \in T$ aufgespannte Universum.



Kapitel 15.3.6

EPTA/EPGA: Besser, best, optimal

Vergleichsrelation 'besser' für Programme

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Programm und seien $G', G'' \in \mathcal{U}_T^G$ mit $T = T_{EPTA}^G$ oder $T = T_{EPGA}^G$.

Definition 15.3.6.1 (Besser)

G' heißt besser als G'' (in Zeichen: $G'' \sqsubseteq\!\!\!\sim G'$) gdw:

$$\forall\, p \in \mathbf{P}_G[\mathbf{s}, \mathbf{e}]\ \forall\, \alpha \in \mathcal{AM}.\ \#_\alpha(p_{G'}) \leq \#_\alpha(p_{G''})$$

wobei $\#_\alpha(p_{G'})$ und $\#_\alpha(p_{G''})$ die Anzahl von Anweisungen des Anweisungsmusters α auf p in G' bzw. G'' bezeichnen.

Beachte: Anweisungssenkungen und -eliminationen erhalten die Verzweigungs- und Knotenstruktur eines Programms G. Die einem Pfad in G eineindeutig in G' und G'' entsprechenden Pfade können deshalb einfach identifiziert werden.

Eigenschaften der Relation ‘besser’

Lemma 15.3.6.2 (Quasiordnung)

Die Programmvergleichsrelation besser $\sqsubseteq$ ist eine Quasiordnung (d.h. reflexiv und transitiv, aber nicht antisymmetrisch).

Lemma 15.3.6.3 (Verbesserung, Verb.-Neutralität)

Seien G und G' zwei Programme mit $G \vdash_\tau G'$ und $\tau \in AS \cup ETA \cup EGA$. Dann gilt:

1. $G \sim G'$, falls τ eine zulässige Anweisungssenkung ist.
2. $G \stackrel{\sqsubset}{\not\sim} G'$, falls τ eine nichttriviale ($\neq Id$) zulässige Elimination toter oder geisterhafter Anweisungen ist.

...das heißt: Eliminationen bewirken umittelbar echte Verbesserungen, während Senkungen verbesserungsneutral sind, aber durch spätere Eliminationen als Effekte zweiter Ordnung mittelbar Verbesserungen ermöglichen können.

Global beste (oder optimale) Programme

Sei G ein Programm und $T = T^G_{EPTA}$ oder $T = T^G_{EPGA}$.

Definition 15.3.6.4 (Global beste (optimale) Prg.)

1. Ein Programm $G^* \in \mathcal{U}^G_T$ heißt global T-best (oder global T-optimal) gdw G^* ist besser als jedes andere Programm aus $\mathcal{U}^G_T$:
$$\forall\, G' \in \mathcal{U}^G_T.\ G' \sqsubseteq_{\approx} G^*$$
2. Bezeichne $\mathcal{G}^{opt}_T(G)$ die Menge der global T-besten (oder global T-optimalen) Programme in $\mathcal{U}^G_T$.

Lokal beste (oder optimale) Programme

Sei G ein Programm und $T = T^G_{EPTA}$ oder $T = T^G_{EPGA}$.

Definition 15.3.6.5 (Lokal beste (optimale) Prg.)

1. Ein Programm $G^* \in \mathcal{U}^G_T$ heißt lokal T-best gdw:

$$\forall\, G' \in \mathcal{U}^{G^*}_T .\ G^* \ \approx\ G'$$

2. Bezeichne $\mathcal{G}^{lokopt}_T(G)$ die Menge der lokal T-besten (oder lokal T-optimalen) Programme in $\mathcal{U}^G_T$.

Intuitiv: Ein Programm ist lokal optimal, wenn beliebige weitere (Folgen von) Elementartransformationen nicht mehr zu einer echten Verbesserung führen, sondern höchstens noch Anweisungen durch Senkungen an anderen Programmstellen platzieren.

Vergleichsrelation 'besser' für Transf.-Folgen

Sei G ein Programm und $T = T^G_{EPTA}$ oder $T = T^G_{EPGA}$.

Definition 15.3.6.6 (EPTA/EPGA-bessere Transf.)

Eine Transformationsfolge (oder Transformation)

1. $\tau \in \mathsf{T}^G_{EPTA}$ heißt EPTA-besser für G als $\tau' \in T^G_{EPTA}$ gdw: $G_{\tau'} \sqsubseteq\!\!\!\!\sim\; G_\tau$.
2. $\tau \in \mathsf{T}^G_{EPGA}$ heißt EPGA-besser für G als $\tau' \in T^G_{EPGA}$ gdw: $G_{\tau'} \sqsubseteq\!\!\!\!\sim\; G_\tau$.

Global, lokal beste Transformationsfolgen

Definition 15.3.6.7 (Global EPTA/EPGA-beste T.)

Eine Transformationsfolge (oder Transformation)

1. $\tau \in T^G_{EPTA}$ heißt global EPTA-best für G gdw τ ist EPTA-besser für G als jede andere Transformation in T^G_{EPTA}.
2. $\tau \in T^G_{EPGA}$ heißt global EPGA-best für G gdw τ ist EPGA-besser für G als jede andere Transformation in T^G_{EPGA}.

Definition 15.3.6.8 (Lokal EPTA/EPGA-beste T.)

Eine Transformationsfolge (oder Transformation)

1. $\tau \in T^G_{EPTA}$ heißt lokal EPTA-best für G gdw keine EPTA-Verlängerung von τ ist echt EPTA-besser als τ
2. $\tau \in T^G_{EPGA}$ heißt lokal EPGA-best für G gdw keine EPGA-Verlängerung von τ ist echt EPGA-besser als τ

d.h. τ ist genauso gut wie jede Verlängerung von τ.

Mengen bester (oder optimaler) Transf.-Folgen

Definition 15.3.6.9 (Beste (oder optimale) Transf.)

Wir bezeichnen mit:

1. $T^{opt}_{EPTA}(G)/T^{lokopt}_{EPTA}(G)$ die Menge der global/lokal EPTA-besten (oder EPTA-optimalen) Transformationen für G.
2. $T^{opt}_{EPGA}(G)/T^{lokopt}_{EPGA}(G)$ die Menge der global/lokal EPGA-besten (oder EPGA-optimalen) Transformationen für G.

Globale Optimalität

...von Programmen und Transformationen impliziert ihre lokale Optimalität.

Proposition 15.3.6.10

1. Global EPTA/EPGA-optimale Programme sind lokal EPTA/EPGA-optimal.
2. Global EPTA/EPGA-optimale Transformationen sind lokal EPTA/EPGA-optimal.

Universumskorrekt, universumsoptimal

Sei G ein Programm.

Lemma 15.3.6.11 (Universumskorrekt)

1. Ist $\tau \in T^G_{EPTA}$, so ist $G_\tau \in \mathcal{U}^G_{T_{EPTA}}$.
2. Ist $\tau \in T^G_{EPGA}$, so ist $G_\tau \in \mathcal{U}^G_{T_{EPGA}}$.

Lemma 15.3.6.12 (Universumsoptimal)

1. Ist $\tau \in T =_{df} T^G_{EPTA}$ global EPTA-best für G, so ist $G_\tau \in \mathcal{G}^{opt}_T(G)$ global T-best.
2. Ist $\tau \in \ =_{df} T^G_{EPGA}$ global EPGA-best für G, so ist $G_\tau \in \mathcal{G}^{opt}_T(G)$ global T-best.

Kapitel 15.3.7

EPTA/EPGA: Optimalität

Maximale Transformationsfolgen

Sei G ein Programm und $T = T^G_{EPGA}$ oder $T = T^G_{EPGA}$.

Definition 15.3.7.1 (Maximale Transf.-Folge)

Eine unendliche oder endliche Transformationsfolge τ für G mit $\tau \in T$ und

- $\tau = (\tau_i)_{i \in \mathbb{N}}$ oder
- $\tau = (\tau_i)_{i \leq k}$, $k \in \mathbb{N}$

heißt maximal, wenn τ lokal optimal ist (d.h. weitere Eliminationstransformationen lassen das Programm G_τ unverändert, weitere Senkungstransformationen platzieren lediglich Anweisungen an anderen Programmstellen ohne dadurch neue Eliminationsmöglichkeiten zu eröffnen).

Faire Transformationsfolgen

Sei G ein Programm und $T = T^G_{EPGA}$ oder $T = T^G_{EPGA}$.

Definition 15.3.7.2 (Faire Transf.-Folge)

Eine Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T$, für G heißt fair, wenn

$$\forall\, k \in \mathbb{N}.\ (\tau_i)_{i \leq k} \text{ nicht maximal } \Rightarrow\ \exists k' > k.\ G_{(\tau_i)_{i \leq k}} \stackrel{\sqsubset}{\not\sim} G_{(\tau_i)_{i \leq k'}}$$

Lemma 15.3.7.3 (Maximale Transf.-Folge fair)

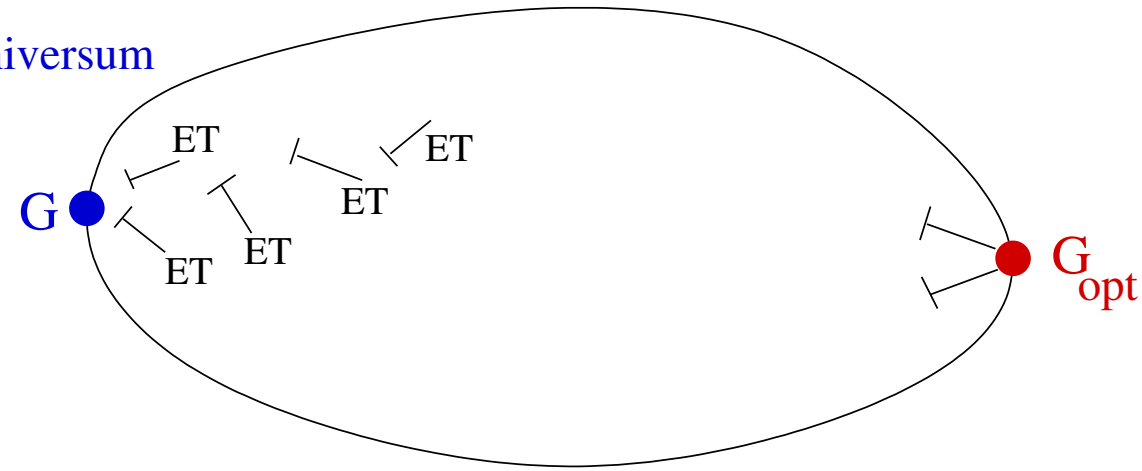
Maximale Transformationsfolgen für G sind fair.

Globale EPTA/EPGA-Optimalität

Sei G ein Programm und $T = T^G_{EPTA}$ oder $T = T^G_{EPGA}$.

Theorem 15.3.7.4 (Glob. EPTA/EPGA-Optimalität)

1. Jede faire Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T$, für G ist global optimal, d.h. endet in einem bis auf irrelevante ($\widehat{=}$ bedeutungsgleiche) Umsortierungen von Anweisungen in Basisblöcken eindeutig bestimmten global optimalen Programm $G_{opt} \in \mathcal{G}^{opt}_T(G)$.
2. Jede faire Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T$, hat ein endliches Anfangsstück $\tau' = (\tau_i)_{i \leq k}$ mit $G_{opt} \sim G_\tau \sim G_{\tau'}$.



Beweisskizze von Theorem 15.3.7.4

...für zwei Beweisvarianten: Über

1. Monotonie, Dominanz und Fixpunkttheorem 14.2.9 (Variante 1).
2. Konfluenz und Termination der Transformationsrelation, s. Theorem 15.3.7.5 (Variante 2).

Variante 1: Monotonie, Dominanz, FP-Theor.

Zeige: Die Menge der EPTA- und EPGA-Elementartransformationen bildet für $\vec{\sqsubseteq}_\tau =_{df} (\sqsubseteq \cap \vdash_\tau)^*$ eine Familie $\mathcal{F}_\tau$ von Funktionen mit folgenden Eigenschaften:

- $\mathcal{F}_\tau \subseteq \{f \mid f : \mathcal{G} \to \mathcal{G}\}$ ist endliche Familie von Funktionen mit
  1. Monotonie:
     $\forall\, G', G'' \in \mathcal{G}\ \forall\, f \in \mathcal{F}_\tau.\ \ G' \vec{\sqsubseteq}_\tau G'' \Rightarrow f(G') \vec{\sqsubseteq}_\tau f(G'')$
  2. Dominanz:
     $\forall\, G', G'' \in \mathcal{G}.\ G' \vdash_\tau G'' \Rightarrow \exists\, f \in \mathcal{F}_\tau.\ G'' \vec{\sqsubseteq}_\tau f(G')$

Zusammen mit dem Fixpunkttheorem 14.2.9 folgt daraus die

- Korrektheit und Optimalität

fairer EPTA- und EPGA-Transformationsfolgen.

Variante 2: Konfluenz, Terminierung

Zeige: Die ETPA- und EPGA-Transformationsrelationen

1. $\cdot \vdash_\tau \cdot, \tau \in AS \cup ETA$ (EPTA)
2. $\cdot \vdash_\tau \cdot, \tau \in AS \cup EGA$ (EPGA)

sind (bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken) konfluent und terminierend.

Beide Eigenschaften zusammen liefern die

- Korrektheit und globale Optimalität

fairer EPTA- und EPGA-Transformationsfolgen.

EPTA, EPGA: Konfluenz, Terminierung

Theorem 15.3.7.5 (Konfluenz, Terminierung)

Die EPTA- und EPGA-Transformationsrelationen

1. $\cdot \vdash_\tau \cdot, \tau \in AS \cup ETA$ (EPTA)
2. $\cdot \vdash_\tau \cdot, \tau \in AS \cup EGA$ (EPGA)

sind (bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken) konfluent und terminierend.

Determiniertheit maximaler Transformationen

Sei G ein Programm und $T = T^G_{EPTA}$ oder $T = T^G_{EPGA}$.

Korollar 15.3.7.6 (Determiniertheit max. Transf.-F.)

Sind $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau' = (\tau_j)_{j \in \mathbb{N}}$, $\tau, \tau' \in T$, maximal (und damit fair) für G, so stimmen G_τ und $G_{\tau'}$ bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken überein, d.h. das finale Programm maximaler Transformationsfolgen ist determiniert

Korollar 15.3.7.7 (Maximale endl. Transf.-Folge)

Ist $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T$, fair für G, so hat τ ein endliches Anfangsstück, das maximal für G ist, d.h. es gibt ein $k \in \mathbb{N}$ mit $\tau' = (\tau_i)_{i \leq k}$ maximal für G.

Endliche faire Transformationsfolgen

Theorem 15.3.7.8 (Endliche faire Transf.-Folge)

1. Eine Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T$, für G, die für jedes Anweisungsmuster Senkungs- und Eliminationstransformation nach höchstens endlich vielen Eliminations- und Senkungstransformationen für andere Anweisungsmuster wieder anwendet, ist fair.
2. Eine endliche Transformationsfolge für G ist maximal (und damit fair), wenn ein voller Zyklus von Senkungs- und Eliminationstransformationen für alle Anweisungsmuster keine echte Verbesserung mehr erbracht hat.

Korollar 15.3.7.9 (Existenz global opt. Transf.)

1. $\forall\, G \in \mathcal{G}.\ T^{opt}_{EPTA}(G) \neq \emptyset$.
2. $\forall\, G \in \mathcal{G}.\ T^{opt}_{EPGA}(G) \neq \emptyset$.

Existenz und Konstruktion

...global optimaler Programme und Transformationen.

Korollar 15.3.7.10 (Existenz global opt. Programme)

1. $\forall\, G \in \mathcal{G}.\ \mathcal{G}^{opt}_{T_{EPTA}}(G) \neq \emptyset$.
2. $\forall\, G \in \mathcal{G}.\ \mathcal{G}^{opt}_{T_{EPGA}}(G) \neq \emptyset$.

Korollar 15.3.7.11 (Determiniertheit)

Bis auf irrelevante Umsortierungen von Anweisungen gilt:

1. $|\mathcal{G}^{opt}_{T_{EPTA}}(G)| = 1$.
2. $|\mathcal{G}^{opt}_{T_{EPGA}}(G)| = 1$.

Korollar 15.3.7.12

Theorem 15.3.7.8 beschreibt konstruktiv und effektiv die Bildung endlicher maximaler (und damit fairer und optimaler) EPTA/EPGA-Transformationsfolgen.

EPGA wirkmächtiger als EPTA

...jede tote Anweisung ist eine Geisteranweisung.

Für die bis auf irrelevante Umsortierungen in Basisblöcken eindeutig bestimmten global optimalen Programme

$$G^{opt}_{EPTA} \in \mathcal{G}^{opt}_{T_{EPTA}}(G) \text{ und } G^{opt}_{EPGA} \in \mathcal{G}^{opt}_{T_{EPGA}}(G)$$

gilt deshalb: G^{EPGA}_{opt} ist besser (i.S.v. Definition 15.3.6.1) als G^{opt}_{EPTA}:

Lemma 15.3.7.13 (EPGA besser als EPTA)

$$\forall\, G \in \mathcal{G}.\ G^{opt}_{EPTA} \sqsubseteq_{\sim} G^{opt}_{EPGA}$$

Übungsaufgabe 15.3.7.14

Gemäß Lemma 15.3.7.13 ist EPGA wirkmächtiger als EPTA. Wie oft allerdings müssen durch Eliminationstransformationen Geisteranweisungen in einer maximalen Transformationsfolge zu G^{opt}_{EPGA} eliminiert werden? Mehrfach? Stets? Reicht einmal?

Betrachte folgende Transformationsmengen:

1. $T_1 =_{df} T^G_{AS,EGA} = \{\tau \,|\, \tau = (\tau_i)_{i\in\mathbb{N}} \in \mathcal{L}((AS + EGA)^*)\}$
2. $T_2 =_{df} T^G_{EGA\circ(AS,ETA)} = \{\tau \,|\, \tau = (\tau_i)_{i\in\mathbb{N}} \in \mathcal{L}(EGA_{\mathcal{AM}} * (AS + ETA)^*)\}$
3. $T_3 =_{df} T^G_{(AS,ETA)\circ EGA\circ(AS,ETA)} = \{\tau \,|\, \tau = (\tau_i)_{i\in\mathbb{N}} \in \mathcal{L}((AS+ETA)^* * EGA_{\mathcal{AM}} * (AS+ETA)^*)\}$

wobei T_2 und T_3 genau eine EGA-Transformation simultan für alle Anweisungsmuster zu Anfang oder an beliebiger Stelle in einer Transformationsfolge vorsehen.

Übungsaufgabe 15.3.7.14 (fgs.)

Untersuche die Gültigkeit folgender Behauptung:

Die bis auf irrelevante Umsortierungen eindeutig bestimmten global optimalen Programme $G_1^{opt} \in \mathcal{G}_{T_1}^{opt}(G) = \mathcal{G}_{T_{EPGA}}^{opt}(G)$, $G_2^{opt} \in \mathcal{G}_{T_2}^{opt}(G)$ und $G_3^{opt} \in \mathcal{G}_{T_3}^{opt}(G)$ sind gleich gut bzgl. der Relation besser (aus Definition 12.3.6.1):

$$G_1^{opt} \sim G_2^{opt} \sim G_3^{opt}$$

Beweis oder Gegenbeispiel.

Kapitel 15.3.8

EPTA/EPGA: Implementierung

Beobachtung

...um die Elementartransformationen von EPTA und EPGA und diese selbst implementieren zu können, ist die Angabe von DFAs für folgende Aufgaben nötig (und ausreichend):

Datenflussanalyseverfahren zur Berechnung

- toter Anweisungen
- schattenhafter Anweisungen
- der Endpunkte von Anweisungssenkungen

Tote Variablenanalyse (1)

...für knotenbenannte Instruktionsgraphen.

Lokale Prädikate (assoziiert mit Instruktionsknoten):

- Mod_n^v: Die Anweisung von Knoten n modifiziert Variable v.
- $\mathrm{Use}_\varepsilon^v$: Variable v wird von der Anweisung von Knoten n gelesen. (z.B. rechtsseitig in einer Zuweisung, in einer Ausgabeanweisung, Verzweigungsbedingung oder Schleifenabbruchbedingung).
- $LhsVar_n$: Bezeichnet die linksseitige Variable der Zuweisung von Knoten n.

Tote Variablenanalyse (2)

Das TVA-Gleichungssystem für knotenbenannte Instruktionsgraphen (simultan für alle Variablen v):

$$\text{N-DEAD}_n^v = \overline{\text{Use}_n^v} * (\text{X-DEAD}_n^v + \text{Mod}_n^v)$$

$$\text{X-DEAD}_n^v = \prod_{m \in succ(n)} \text{N-DEAD}_m^v$$

Größte Lösung: $\nu\text{-}N\text{-}DEAD_n^v$, $\nu\text{-}X\text{-}DEAD_n^v$.

Lemma 15.3.8.1 (Tote Anweisungen)

Eine Anweisung α am Knoten n ist tot gdw die linksseitige Variable von α ist geisterhaft am Ausgang von Knoten n, d.h. $\nu\text{-}X\text{-}DEAD_n^{LhsVar_n} = \textbf{wahr}$.

Intuitiv

...eine Variable v ist tot am Eingang des Knotens n, wenn v

- von der Anweisung am Knoten n nicht durch lesen 'zu leben gezwungen' wird (1-tes Konjunktionsglied).
- am Ausgang des Knotens n bereits tot ist oder durch die Anweisung am Knoten n modifiziert und dadurch das Leben verliert und zu Tode kommt, tot wird (2-tes Konjunktionsglied).

...eine Variable v ist tot am Ausgang des Knotens n, wenn v

- am Eingang aller Nachfolgeknoten tot ist.

Geistervariablenanalyse (1)

...für knotenbenannte Instruktionsgraphen.

Lokale Prädikate (assoziiert mit Instruktionsknoten):

- Mod_n^v: Die Anweisung von Knoten n modifiziert Variable v.
- AssUse_n^v: Die Anweisung von Knoten n ist eine Zuweisung und liest Variable v.
- $\mathrm{LifeEnforcingUse}_n^v$: Variable v wird von der Anweisung am Knoten n gelesen und dadurch 'zu leben gezwungen' (z.B. wenn die Anweisung eine Ausgabeanweisung, Verzweigungsbedingung oder Schleifenabbruchbedingung ist).
- LhsVar_n: Bezeichnet die linksseitige Variable der Zuweisung von Knoten n.

Geistervariablenanalyse (2)

Das GVA-Gleichungssystem für knotenbenannte Instruktionsgraphen (simultan für alle Variablen v):

$$\begin{aligned}
\text{N-FAINT}_n^v &= \overline{\text{LifeEnforcingUse}_n^v} * \\
&\quad (\text{X-FAINT}_n^v + \text{Mod}_n^v) * \\
&\quad (\text{X-FAINT}_n^{LhsVar_n} + \overline{\text{AssUse}_{n)}^v}) \\
\text{X-FAINT}_n^v &= \prod_{m \in succ(n)} \text{N-FAINT}_m^v
\end{aligned}$$

Größte Lösung: $\nu\text{-}N\text{-}FAINT_n^v$, $\nu\text{-}X\text{-}FAINT_n^v$.

Lemma 15.3.8.2 (Geisterhafte Anweisungen)

Eine Anweisung α am Knoten n ist geisterhaft gdw die linksseitige Variable von α ist geisterhaft am Ausgang von Knoten n, d.h. $\nu\text{-}X\text{-}FAINT_n^{LhsVar_n} = \textbf{wahr}$.

Intuitiv

...eine Variable v ist geisterhaft am Eingang des Knotens n, wenn v

- von der Anweisung am Knoten n nicht 'zu leben gezwungen' wird (1-tes Konjunktionsglied).
- am Ausgang des Knotens n bereits geisterhaft ist oder durch die Anweisung am Knoten n modifiziert und dadurch geisterhaft wird (2-tes Konjunktionsglied).
- von der Anweisung am Knoten n nicht benutzt wird oder höchstens der Wertzuweisung an eine andere geisterhafte Variable dient (3-tes Konjunktionsglied).

...eine Variable v ist geisterhaft am Ausgang des Knotens n, wenn v

- am Eingang aller Nachfolgeknoten geisterhaft ist.

Anweisungssenkungsanalyse

...für knotenbenannte Instruktionsgraphen.

Lokale Prädikate (assoziiert mit Instruktionsknoten):

- Sinkable_n^α: Die Anweisung von Knoten n ist vom Anweisungsmuster α (und steht deshalb bereits am Ende von n).
- Blocked_n^α: Die Senkung (oder Verschiebung) von α über die Anweisung am Knoten n hinweg wird von dieser n blockiert.

Die Anweisungssenkungsanalyse

Das ASA-Gleichungssystem für knotenbenannte Instruktionsgraphen (simultan für alle Anweisungsmuster α):

$$\text{N-SINK}_n^\alpha = \begin{cases} \textbf{falsch} & \text{falls } n = \textbf{s} \\ \prod_{m \in pred(n)} \text{X-SINK}_m^\alpha & \text{sonst} \end{cases}$$

$$\text{X-SINK}_n^\alpha = \text{Sinkable}_n^\alpha + \text{N-SINK}_n^\alpha * \overline{\text{Blocked}_n^\alpha}$$

Größte Lösung: $\nu\text{-}N\text{-}SINK_n^\alpha$, $\nu\text{-}X\text{-}SINK_n^\alpha$.

Endpunkte der Anweisungssenkung

Die sich aus der AS-Analyse ergebenden Einsetzungspunkte:

$$\text{N-Insert}_n^\alpha =_{df} \nu\text{-N-SINK}_n^\alpha * \text{Blocked}_n^\alpha$$

$$\text{X-Insert}_n^\alpha =_{df} \nu\text{-X-SINK}_n^\alpha * \sum_{m \in succ(n)} \overline{\nu\text{-N-SINK}_m^\alpha}$$

Wichtig: Die Berechnung der Einsetzungspunkte (d.h. der Endpunkte der Schiebung) erfordert keine iterative globale DFA, sondern kann lokal an jedem Knoten berechnet werden mithilfe des lokalen Prädikats Blocked und der größten Lösung des AS-Gleichungssystems.

Übungsaufgabe 15.3.8.1

Wie lauten die Spezifikationen der Analysen zur

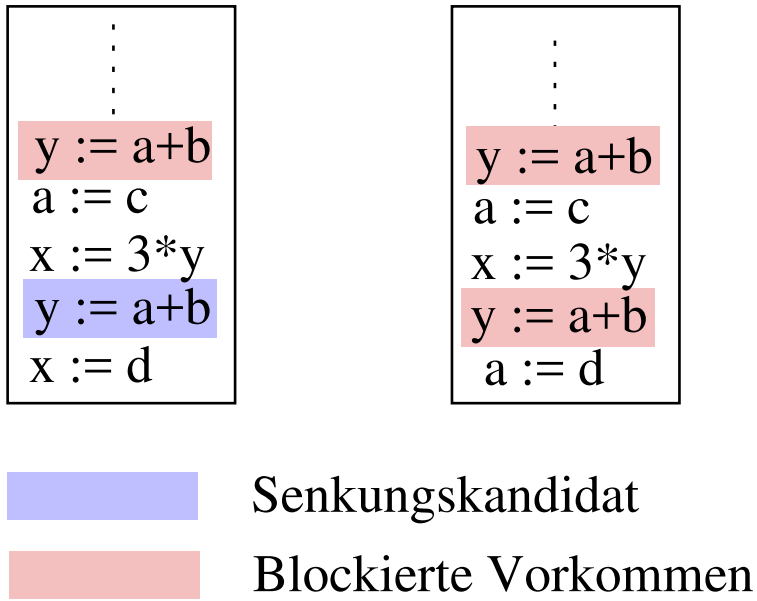
1. Erkennung toter Anweisungen
2. Erkennung geisterhafter Anweisungen
3. Senkung von Anweisungen

im Stil von Kapitel 8?

Gib die entsprechenden Spezifikationstupel an.

Anmerkung zu Basisblock-Graphen (1)

...wenn sie existieren, sind Senkungskandidaten in Basisblöcken eindeutig bestimmt:



Nur das blau markierte Vorkommen von `y := a+b` ist ein Senkungskandidat; die pink markierten Vorkommen von `y := a+b` sind lokal in den Basisblöcken blockiert.

Anmerkung zu Basisblock-Graphen (2)

Senkungsanalyse

- Die Eindeutigkeit von Senkungskandidaten erlaubt die Senkungsanalyse unmittelbar (ohne Änderungen oder Anpassungen) von Instruktions- auf Basisblockgraphen zu übertragen.

Tote Variablenanalyse

- Anpassungen zur Übertragung der Analyse von Instruktions- auf Basisblockgraphen sind erforderlich; siehe Anhang B für Details.

Geistervariablenanalyse

- Als sog. nicht-separates Analyseproblem keine Übertragung auf Basisblockgraphen möglich; siehe Anhang B für Details.

Kapitel 15.4

Partiell redundante Anweisungen

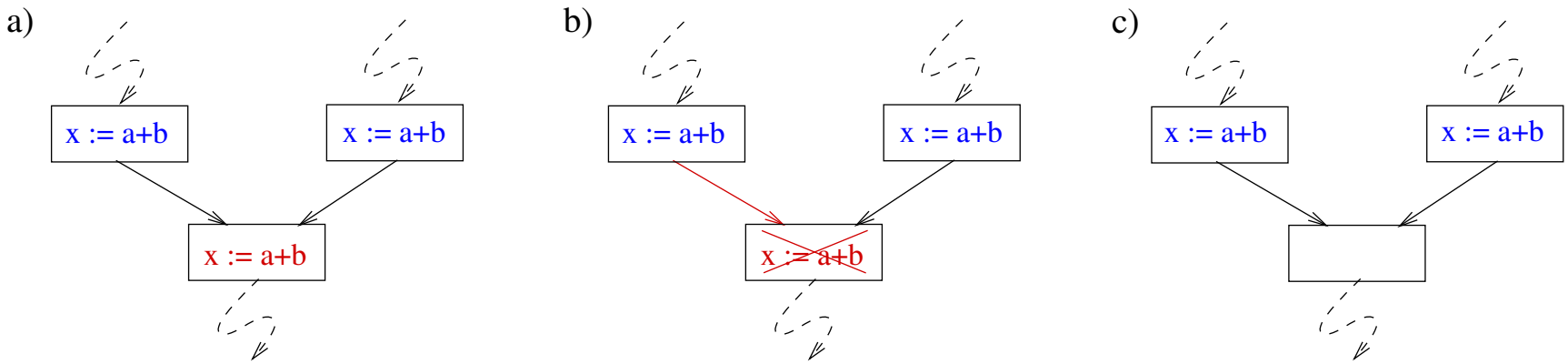
Kapitel 15.4.1

Motivation

Redundante Anweisungen

...und ihre Elimination.

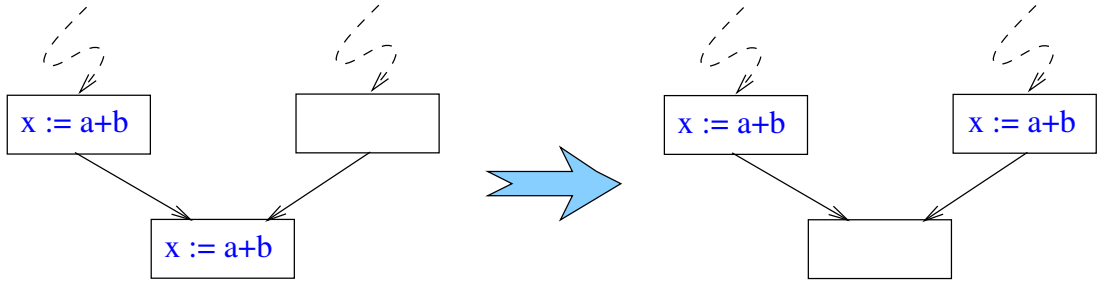
Das Grundmuster: Das rote Vorkommen der Anweisung `x := a+b` ist redundant (oder total redundant) (engl. (totally) redundant) gegenüber den beiden blauen, da weder `x` noch `a` oder `b` zwischen den Vorkommen geschrieben werden.



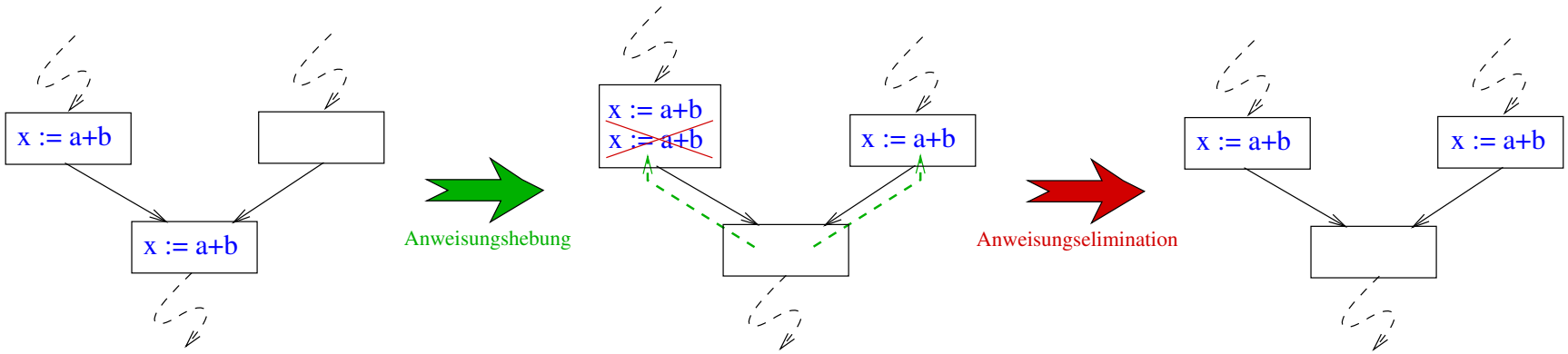
Partiell redundante Anweisungen

...und ihre Elimination.

Das Grundmuster:



Die konzeptuelle Verfahrensidee:



Kapitel 15.4.2

Elementartransformationen

Redundante Anweisungen

Definition 15.4.2.1 (Redundante Anweisung)

Eine Anweisung vom Muster $\alpha \equiv x := t$ am Knoten n ist redundant gdw jeder Programmpfad vom Startknoten zu einem Vorgänger m von n führt über einen Knoten k, wobei gilt:

- m enthält ein α-Vorkommen als Anweisung.
- auf der Pfadfortsetzung von k zu m werden `x`, `a` und `b` nicht geschrieben.

Anweisungseliminationen

Definition 15.4.2.2 (Elimination redundanter Anw.)

Eine Anweisungselimination, die einige redundante Anweisungen aus dem Programm streicht, heißt Elimination redundanter Anweisungen (engl. redundant assignment elimination).

Definition 15.4.2.3 (Korrekte Anweisungselim.)

Eine Anweisungselimination ist korrekt, wenn sie eine Elimination redundanter Anweisungen ist.

Anweisungshebungen

Definition 15.4.2.4 (Anweisungshebung)

Eine Anweisungshebung für ein Anweisungsmuster $\alpha \equiv x := t$ (oder α-Anweisungshebung) ist das simultane stetige Verschieben eines oder mehrerer Vorkommen von α entgegen der Richtung des Kontrollflusses zu einem oder mehreren anderen Knoten.

Definition 15.4.2.5 (Korrekte Anweisungshebung)

Eine α-Anweisungshebung, $\alpha \equiv x := t$, ist korrekt, wenn zu jedem Zeitpunkt während des Schiebens gilt:

1. Kein α-Vorkommen wird über eine Anweisung hinweggeschoben, die x liest oder modifiziert oder einen Operanden von t modifiziert (und α dadurch blockiert).
2. Kein α-Vorkommen wird (rückwärts) in einen Verzweigungsknoten geschoben, wenn dies nicht von jedem Nachfolger des Verzweigungsknotens aus geschieht.

Sprechweisen und Bezeichnungen

Definition 15.4.2.6 (Blockiert)

Ein Anweisung α der Form $x := t$ ist von einer Anweisung α' hebungsblockiert (oder: blockiert), wenn α'

- Variable x
  - liest ($\alpha' \equiv ... := ...x...$) oder
  - modifiziert ($\alpha' \equiv x := ...$) oder
- einen Operanden von t modifiziert.

Proposition 15.4.2.7

Eine Anweisung blockiert die Hebung einer Anweisung gdw sie deren Senkung blockiert.

Kapitel 15.4.3

Effekte zweiter Ordnung

Effekte zweiter Ordnung

...treten bei EPRA ähnlich wie bei EPTA und EPGA in 4 Formen auf und sind auch hier maßgeblich für die kombinierte Wirkung der Elementartransformationen von EPRA:

1. Hebungs-Eliminations-Effekte (Zieleffekt)
   $\rightsquigarrow$ Hoisting-elimination effects
2. Hebungs-Hebungs-Effekte (Potentialeffekt)
   $\rightsquigarrow$ Hoisting-hoisting effects
3. Eliminations-Hebungs-Effekte (Potentialeffekt)
   $\rightsquigarrow$ Elimination-hoisting effects
4. Eliminations-Eliminations-Effekte (Zieleffekt)
   $\rightsquigarrow$ Elimination-elimination effects

Übungsaufgabe 15.4.3.1

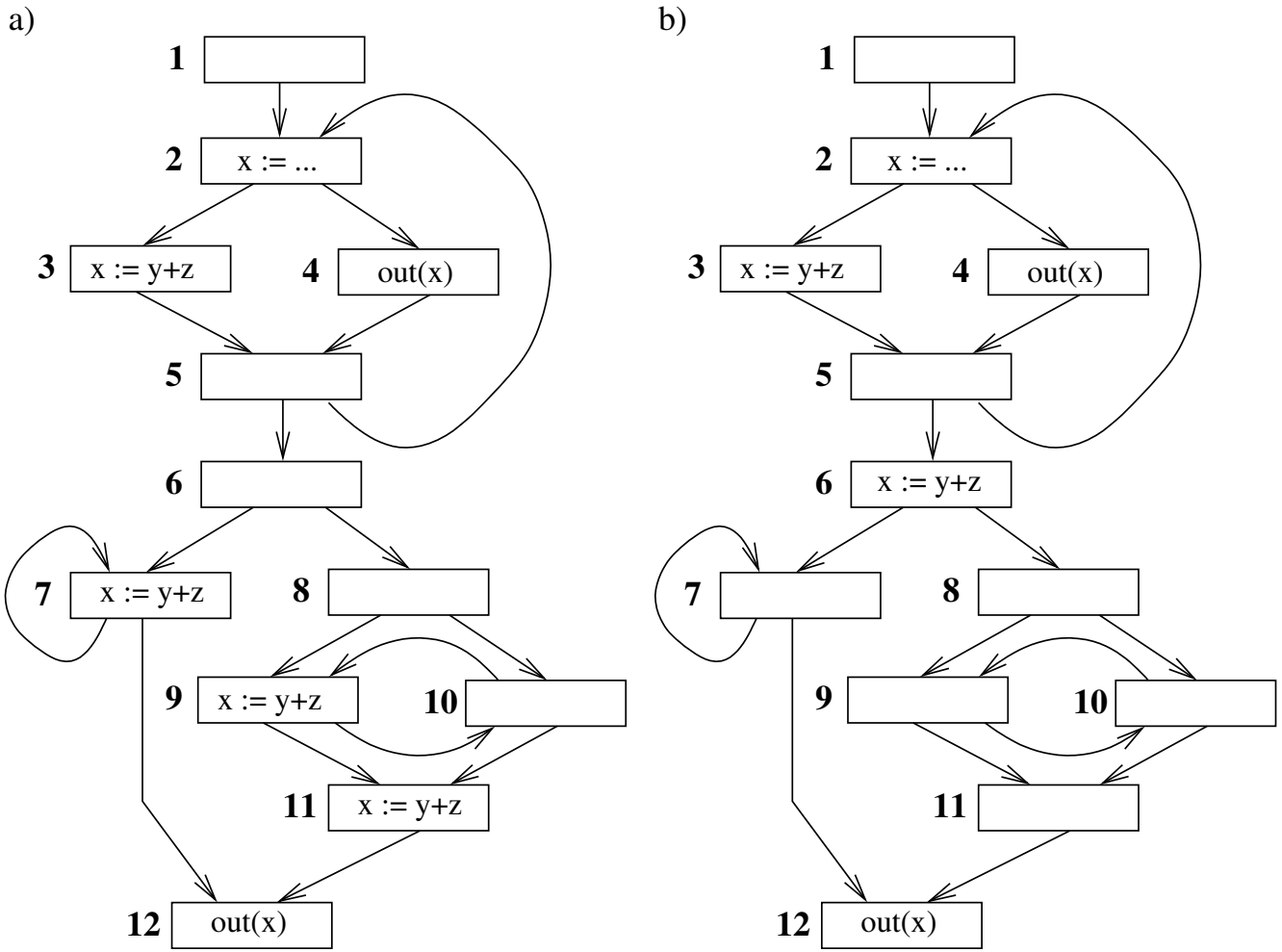
Gib für jeden der vier Effekte zweiter Ordnung von EPRA ein Beispiel an:

1. Hebungs-Eliminations-Effekt
2. Hebungs-Hebungs-Effekt
3. Eliminations-Hebungs-Effekt
4. Eliminations-Eliminations-Effekt

Sequenzwirkung von Effekten 2. Ordnung

Ausgangsprogramm

Optimiertes Programm



Beachte: Kein Schieben von Anweisungen in Schleifen!

Kapitel 15.4.4

EPRA: Transformation

Unnötige Anweisungen

Sei G ein Programm.

Definition 15.4.4.1 (Unnötige Anweisung)

Eine Anweisung α am Knoten n in G heißt unnötig gdw α ist redundant am Knoten n.

Sprechweisen und Bezeichnungen (1)

Wir bezeichnen informell mit

- $AH \quad =_{df} \bigcup \{AH_\alpha^G \mid \alpha \in \mathcal{AM}, G \in \mathcal{G}\}$
- $ERA =_{df} \bigcup \{ERA_\alpha^G \mid \alpha \in \mathcal{AM}, G \in \mathcal{G}\}$

die Mengen aller zulässigen Anweisungshebungen und -eliminationen (für beliebige Programme).

Wir bezeichnen ebenso informell mit Wörtern der von dem regulär-artigen Ausdruck

$$(AH + ERA)^*$$

erzeugten Sprache

$$\mathcal{L}((AH + ERA)^*)$$

Folgen zulässiger AH- und ERA-Transformationen (für beliebige Programme und Anweisungsmuster).

Sprechweisen und Bezeichnungen (2)

Mit diesen Schreibweisen bezeichne:

- $T^G_{AH,ERA} = \{\tau \,|\, \tau = (\tau_i)_{i \in \mathbb{N}} \in \mathcal{L}((AH + ERA)^*)\}$

die Menge aller Transformationsfolgen aus zulässigen AH- und ERA-Transformationen für ein Programm G.

Geht G aus dem Kontext hervor, schreiben wir statt $T^G_{AH,ERA}$ einfacher $T_{AH,ERA}$.

Ist $\tau = (\tau_i)_{i \in \mathbb{N}}$ eine Transformationsfolge, so bezeichne:

- $(\tau_i)_{i \leq k}$ das Anfangsstück von τ bis zum Index k einschließlich.
- τ_j die Elementartransformation mit Index j von τ .
- G_τ, $G_{(\tau_i)_{i \leq k}}$ und $G_{(\tau_j)}$ diejenigen Programme, die aus G durch Anwendung von τ, $(\tau_i)_{i \leq k}$, auf G entstehen.

EPRA-Transformation

Sei G ein Programm.

Definition 15.4.4.2 (EPRA-Transformation)

1. Eine EPRA-Transformation ist eine beliebige Abfolge zulässiger Anweisungshebungen und Eliminationen redundanter Anweisungen.
2. T^G_{EPRA} bezeichnet die Menge aller EPRA-Transformationen für G, in Zeichen:

$$T^G_{EPRA} =_{df} T^G_{AH,ERA}$$

3. Ist $\tau \in T^G_{EPRA}$, so bezeichnet G_τ das Programm, das τ angewendet auf G liefert.

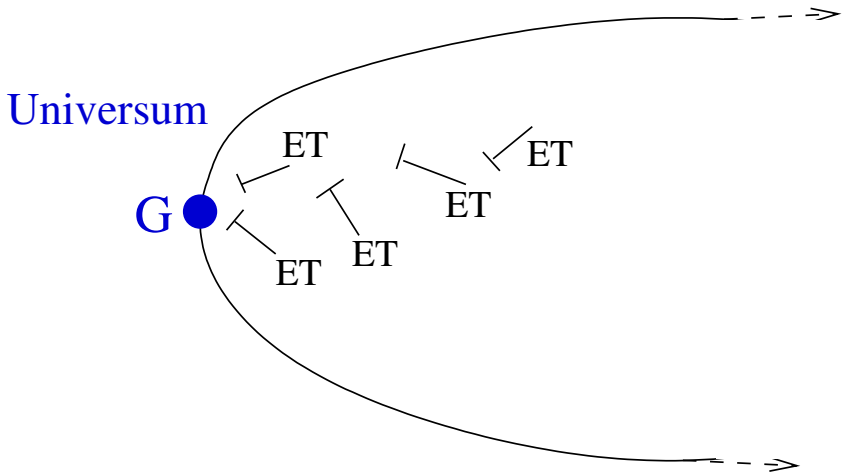
Transformationsrelation, Programmuniversum

- Transformationsrelation:

  $G \vdash_\tau G'$, $\tau \in AH \cup ERA$: G' resultiert aus G durch Anwendung von τ mit τ zulässige Hebungs- oder Eliminationstransformation redundanter Anweisungen.

- Induziertes Programmuniversum:

  $\mathcal{U}^G_{T_{EPRA}} =_{df} \{G' \mid G \vdash_{(\tau_i)_{i \le k}} G', \tau = (\tau_i)_{i \in \mathbb{N}} \in T^G_{EPRA}, k \in \mathbb{N}\}$: Das von G durch die Präfixe der Transformationsfolgen $\tau \in T^G_{EPRA}$ aufgespannte Universum.



Kapitel 15.4.5

EPRA: Besser, best, optimal

Vergleichsrelation 'besser' für Programme

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Programm und seien $G', G'' \in \mathcal{U}^G_{T_{EPRA}}$.

Definition 15.4.5.1 (Besser)

G' heißt besser als G'' (in Zeichen: $G'' \sqsubseteq G'$) gdw. G' ist besser als G'' i.S.v. Definition 15.3.6.1, d.h.:

$$\forall\, p \in \mathbf{P}_G[\mathbf{s}, \mathbf{e}]\ \forall\, \alpha \in \mathcal{AM}.\ \#_\alpha(p_{G'}) \leq \#_\alpha(p_{G''})$$

wobei $\#_\alpha(p_{G'})$ und $\#_\alpha(p_{G''})$ die Anzahl von Anweisungen des Anweisungsmusters α auf p in G' bzw. G'' bezeichnen.

Beachte: Anweisungshebungen und -eliminationen erhalten die Verzweigungs- und Knotenstruktur eines Programms G. Die einem Pfad in G eineindeutig in G' und G'' entsprechenden Pfade können deshalb einfach identifiziert werden.

Eigenschaften der Relation 'besser'

Lemma 15.4.5.2 (Quasiordnung)

Die Programmvergleichsrelation besser $\sqsubseteq_{\sim}$ ist eine Quasiordnung (d.h. reflexiv und transitiv, aber nicht antisymmetrisch).

Lemma 15.4.5.3 (Verbesserung, Verb.-Neutralität)

Seien G und G' zwei Programme mit $G \vdash_\tau G'$ und $\tau \in AH \cup ERA$. Dann gilt:

1. $G \sim G'$, falls τ eine zulässige Anweisungshebung ist.
2. $G \sqsubset_{\not\sim} G'$, falls τ eine nichttriviale ($\neq Id$) zulässige Elimination redundanter Anweisungen ist.

...das heißt: Eliminationen bewirken umittelbar echte Verbesserungen, während Hebungen verbesserungsneutral sind, aber durch spätere Eliminationen als Effekte zweiter Ordnung mittelbar Verbesserungen ermöglichen können.

Beste (oder optimale) Programme

Sei G ein Programm.

Definition 15.4.5.4 (Gobal beste (optimale) Prg.)

1. Ein Programm $G^* \in \mathcal{U}^G_{T_{EPRA}}$ heißt global EPRA-best (oder global EPRA-optimal) gdw G^* ist besser als jedes andere Programm aus $\mathcal{U}^G_{T_{EPRA}}$:

$$\forall\, G' \in \mathcal{U}^G_{T_{EPRA}}.\ G' \sqsubseteq_{\approx} G^*$$

2. Bezeichne $\mathcal{G}^{opt}_{T_{EPRA}}(G)$ die Menge der global EPRA-besten (oder global EPRA-optimalen) Programme in $\mathcal{U}^G_{T_{EPRA}}$.

Lokal beste (oder optimale) Programme

Sei G ein Programm.

Definition 15.4.5.5 (Lokal beste (optimale) Prg.)

1. Ein Programm $G^* \in \mathcal{U}_{T_{EPRA}}^{G}$ heißt lokal EPRA-best gdw:

$$\forall\, G' \in \mathcal{U}_{T_{EPRA}}^{G^*}.\ G^* \sim G'$$

2. Bezeichne $\mathcal{G}_{T_{EPRA}}^{lokopt}(G)$ die Menge der lokal EPRA-besten (oder lokal EPRA-optimalen) Programme in $\mathcal{U}_{T_{EPRA}}^{G}$.

Intuitiv: Ein Programm ist lokal optimal, wenn beliebige weitere (Folgen von) Elementartransformationen nicht mehr zu einer echten Verbesserung führen, sondern höchstens noch Anweisungen durch Hebungen an anderen Programmstellen platzieren.

Vergleichsrelation ‘besser’ für Transf.-Folgen

Sei G ein Programm.

Definition 15.4.5.6 (EPRA-bessere Transf.)

Eine Transformationsfolge (oder Transformation) $\tau \in \mathsf{T}^G_{EPRA}$ heißt EPRA-besser für G als $\tau' \in T^G_{EPRA}$ gdw: $G_{\tau'} \sqsubseteq\!\!\!\sim G_\tau$.

Global, lokal beste Transformationsfolgen

Definition 15.4.5.7 (Global EPRA-beste Transf.)

Eine Transformationsfolge (oder Transformation) $\tau \in T^G_{EPRA}$ heißt global EPRA-best für G gdw τ ist EPRA- besser für G als jede andere Transformation in T^G_{EPRA}.

Definition 15.4.5.8 (Lokal EPRA-beste Transf.)

Eine Transformationsfolge (oder Transformation) $\tau \in T^G_{EPRA}$ heißt lokal EPRA-best für G gdw keine EPRA- Verlängerung von τ ist echt EPRA-besser als τ, d.h. τ ist genauso gut wie jede Verlängerung von τ.

Mengen bester (oder optimaler) Transf.-Folgen

Definition 15.4.5.9 (Beste (oder optimale) Transf.)

Wir bezeichnen mit $T^{opt}_{EPRA}(G)/T^{lokopt}_{EPRA}(G)$ die Menge der global/lokal EPRA-besten (oder EPRA-optimalen) Transformationen für G.

Globale Optimalität

...von Programmen und Transformationen impliziert ihre lokale Optimalität.

Proposition 15.4.5.10

1. Global EPRA-optimale Programme sind lokal EPRA-optimal.
2. Global EPRA-optimale Transformationen sind lokal EPRA-optimal.

Universumskorrekt, universumsoptimal

Sei G ein Programm.

Lemma 15.4.5.11 (Universumskorrekt)

Ist $\tau \in T^G_{EPRA}$, so ist $G_\tau \in \mathcal{U}^G_{T_{EPRA}}$.

Lemma 15.4.5.12 (Universumsoptimal)

Ist $\tau \in T^G_{EPRA}$ global EPRA-best für G, so ist $G_\tau \in \mathcal{G}^{opt}_T(G)$ global EPRA-best.

Kapitel 15.4.6

EPRA: Optimalität

Maximale Transformationsfolgen

Sei G ein Programm.

Definition 15.4.6.1 (Maximale Transf.-Folge)

Eine unendliche oder endliche Transformationsfolge τ für G mit $\tau \in T^G_{EPRA}$ und

- $\tau = (\tau_i)_{i \in \mathbb{N}}$ oder
- $\tau = (\tau_i)_{i \leq k}$, $k \in \mathbb{N}$

heißt maximal, wenn τ lokal optimal ist (d.h. weitere Eliminationstransformationen lassen das Programm G_τ unverändert, weitere Senkungstransformationen platzieren lediglich Anweisungen an anderen Programmstellen ohne dadurch neue Eliminationsmöglichkeiten zu eröffnen).

Faire Transformationsfolgen

Sei G ein Programm.

Definition 15.4.6.2 (Faire Transf.-Folge)

Eine Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T^G_{EPRA}$, für G heißt fair, wenn

$$\forall\, k \in \mathbb{N}.\ (\tau_i)_{i \leq k} \text{ nicht maximal } \Rightarrow\ \exists k' > k.\ G_{(\tau_i)_{i \leq k}} \overset{\sqsubset}{\not\sim} G_{(\tau_i)_{i \leq k'}}$$

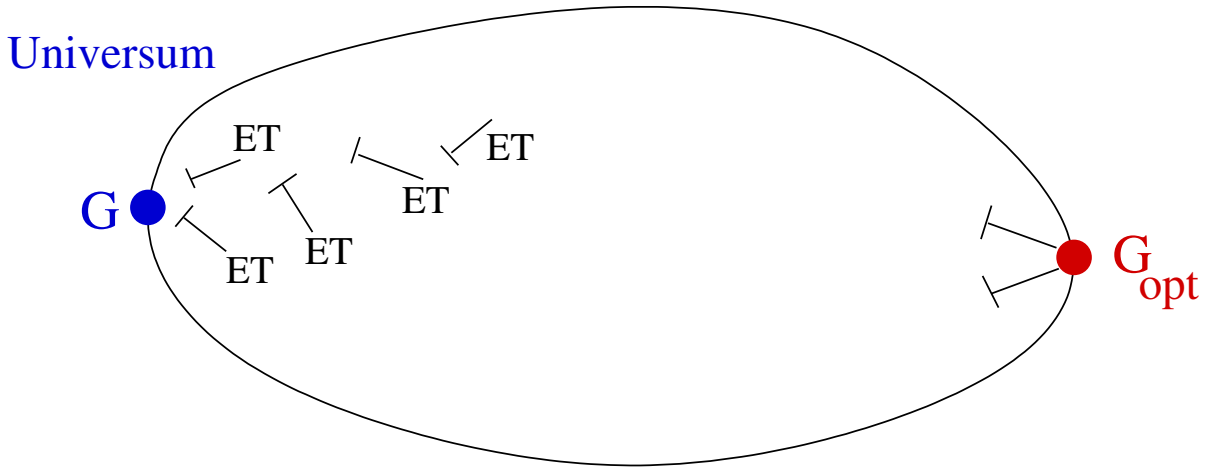
Lemma 15.4.6.3 (Maximale Transf.-Folge fair)

Maximale Transformationsfolgen für G sind fair.

Globale EPRA-Optimalität

Theorem 15.4.6.4 (Globale EPRA-Optimalität)

1. Jede faire EPRA-Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$ für ein Programm G ist global optimal, d.h. endet in einem bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken eindeutig bestimmten global optimalen Programm $G_{opt} \in \mathcal{G}^{opt}_{T_{EPRA}}(G)$.
2. Jede faire EPRA-Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$ hat ein endliches Anfangsstück $\tau' = (\tau_i)_{i \leq k}$ mit $G_{opt} \sim G_\tau \sim G_{\tau'}$



Beweis von Theorem 15.4.6.4

...analog zum Beweis von Theorem 15.3.7.4 in zwei Varianten.
Über:

1. Monotonie, Dominanz und Fixpunkttheorem 14.2.9 (Variante 1).
2. Konfluenz und Termination der Transformationsrelation, s. Theorem 15.4.6.5 (Variante 2).

EPTA/EPGA-T.-Relation: Konfluenz, Term.

Theorem 15.4.6.5 (Konfluenz, Terminierung)

Die EPRA-Transformationsrelation

$$\cdot \vdash_\tau \cdot, \tau \in AH \cup ERA$$

ist (bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken) konfluent und terminierend.

Determiniertheit maximaler Transformationen

Korollar 15.4.6.6 (Determiniertheit max. Transf.-F.)

Sind $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau' = (\tau_j)_{j \in \mathbb{N}}$, $\tau, \tau' \in T^G_{EPRA}$, maximal (und damit fair) für G, so stimmen G_τ und $G_{\tau'}$ bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken überein, d.h. das finale Programm maximaler Transformationsfolgen ist determiniert

Korollar 15.4.6.7 (Endliche max. Transf.-Folge)

Ist $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T^G_{EPRA}$, fair für G, so hat τ ein endliches Anfangsstück, das maximal für G ist, d.h. es gibt ein $k \in \mathbb{N}$ mit $\tau' = (\tau_i)_{i \leq k}$ maximal für G.

Endliche faire Transformationsfolgen

Theorem 15.4.6.8 (Endliche faire Transf.-Folge)

1. Eine Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T^G_{EPRA}$, für G, die für jedes Anweisungsmuster Hebungs- und Eliminationstransformation nach höchstens endlich vielen Eliminations- und Hebungstransformationen für andere Anweisungsmuster wieder anwendet, ist fair.
2. Eine endliche Transformationsfolge für G ist maximal (und damit fair), wenn ein voller Zyklus von Hebungs- und Eliminationstransformationen für alle Anweisungsmuster keine echte Verbesserung mehr erbracht hat.

Korollar 15.4.6.9 (Existenz global opt. Transf.)

$$\forall\, G \in \mathcal{G}.\ T^{opt}_{EPRA}(G) \neq \emptyset$$

Existenz und Konstruktion

...global optimaler Programme und Transformationen.

Korollar 15.4.6.10 (Existenz global opt. Programme)

$$\forall\, G \in \mathcal{G}.\ \mathcal{G}^{opt}_{T_{EPRA}}(G) \neq \emptyset$$

Korollar 15.4.6.11 (Determiniertheit)

Bis auf irrelevante Umsortierungen von Anweisungen gilt:

$$|\mathcal{G}^{opt}_{T_{EPRA}}(G)| = 1$$

Korollar 15.4.6.12

Theorem 15.4.6.7 beschreibt konstruktiv und effektiv die Bildung endlicher maximaler (und damit fairer und optimaler) EPRA-Transformationsfolgen.

Kapitel 15.4.7

EPRA: Implementierung

Beobachtung

...um die Elementartransformationen von EPRA und EPRA selbst implementieren zu können, ist die Angabe von DFAs für folgende Aufgaben nötig (und ausreichend):

Datenflussanalysen zur Berechnung

- redundanter Anweisungen
- der Endpunkte von Anweisungshebungen

Redundante Anweisungsanalyse

...für knotenbenannte Instruktionsgraphen.

Lokale Prädikate (assoziiert mit Instruktionsknoten):

- Transp_n^α: Die Anweisung von Knoten n modifiziert weder die linksseitige Variable noch einen Operanden des rechtsseitigen Ausdrucks von α.
- Comp_n^α: Die Anweisung von Knoten n ist vom Muster α.

Übungsaufgabe 15.4.7.1: Red. Anw.-Analyse

Spezifiziere das RAA-Gleichungssystem für die Erkennung redundanter Anweisungen:

$$\begin{aligned} \text{N-RED}_n^\alpha &= \ldots \\ \text{X-RED}_n^\alpha &= \ldots \end{aligned}$$

Anweisungshebungsanalyse

...für knotenbenannte Instruktionsgraphen.

Lokale Prädikate (assoziiert mit Instruktionsknoten):

- $\text{Hoistable}_n^\alpha$: Die Anweisung von Knoten n ist vom Anweisungsmuster α (und steht deshalb bereits am Anfang von n.
- Blocked_n^α: Die Hebung (oder Verschiebung) von α über die Anweisung am Knoten n hinweg wird von dieser blokkiert.

Übungsaufgabe 15.4.7.2: Anw.-Heb.-Analyse

Spezifiziere das AHA-Gleichungssystem für Anweisungshebung:

$$\begin{aligned} \text{N-HOIST}_n^\alpha &= \ldots \\ \text{X-HOIST}_n^\alpha &= \ldots \end{aligned}$$

sowie die Prädikate für die Endpunkte (oder Einsetzungspunkte) von Anweisungshebungen:

$$\begin{aligned} \text{N-Insert}_n^\alpha &=_{df} \ldots \\ \text{X-Insert}_n^\alpha &=_{df} \ldots \end{aligned}$$

Übungsaufgabe 15.4.7.3

Wie lauten die Spezifikationen der Analysen zur

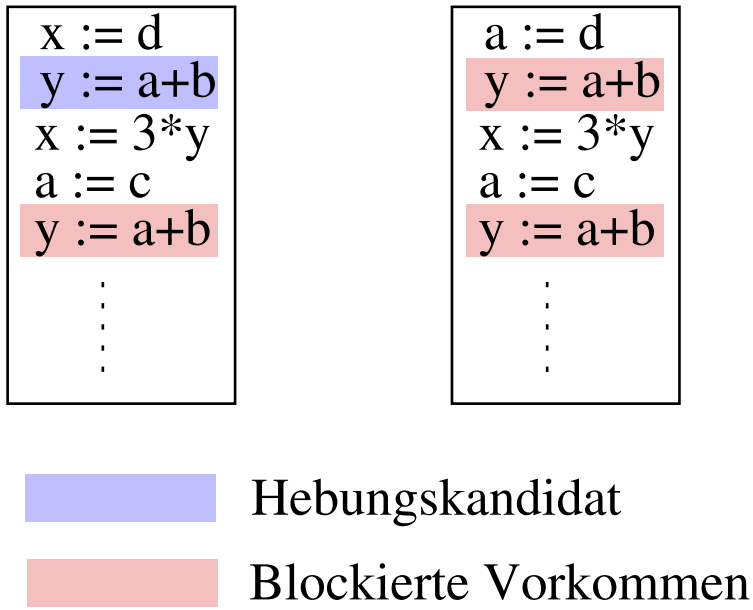
1. Erkennung redundanter Anweisungen
2. Hebung von Anweisungen

im Stil von Kapitel 8?

Gib die entsprechenden Spezifikationstupel an.

Anmerkung zu Basisblock-Graphen (1)

...wenn sie existieren, sind Hebungskandidaten in Basisblöcken eindeutig bestimmt:



Nur das blau markierte Vorkommen von `y := a+b` ist ein Hebungskandidat; die pink markierten Vorkommen von `y := a+b` sind lokal in den Basisblöcken blockiert.

Anmerkung zu Basisblock-Graphen (2)

Hebungsanalyse

- Die Eindeutigkeit von Hebungskandidaten erlaubt die Hebungsanalyse unmittelbar (ohne Änderungen oder Anpassungen) von Instruktions- auf Basisblockgraphen zu übertragen.

Redundanzanalyse

- Anpassungen zur Übertragung der Analyse von Instruktions- auf Basisblockgraphen sind erforderlich; siehe Anhang B für Details.

Kapitel 15.5

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (1)

Ras Bodik, Rajiv Gupta. *Partial Dead Code Elimination using Slicing Transformations*. In Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation (PLDI'97), ACM SIGPLAN Notices 32(6):159-170, 1997.

L. Feigen, D. Klappholz, R. Casazza, X. Xue. *The Revival Transformation*. In Conference Record of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'94), 1994.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (2)


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Partial Dead Code Elimination*. In Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI'94), ACM SIGPLAN Notices 29(6):147-158, 1994.


Ronald J. Mintz, Gerald A. Fisher, Micha Sharir. *The Design of a Global Optimizer*. In Proceedings of the ACM SIGPLAN'79 Symposium on Compiler Construction (SoCC'79), ACM SIGPLAN Notices 14(8):226-234, 1979.


Munehiro Takimoto, Kenichi Harada. *Partial Dead Code Elimination Using Extended Value Graph*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 179-193, 1999.

Kapitel 16

Transformationskombinationen

Kapitel 16.1

EPTRA: EPTA/EPRA-Kombination

Kapitel 16.1.1

EPTA, EPRA: Grundtransformationen

Motivation

...konzeptuell können wir EPTA und EPRA als 'Summe' von je zwei fair wiederholt angewendeter Elementartransformationen verstehen:

- EPTA $= (\mathrm{AS} + \mathrm{ETA})^*$
- EPRA $= (\mathrm{AH} + \mathrm{ERA})^*$

Das legt nahe, auch die 'Summe' aller vier fair wiederholt angewendeter Elementarfunktionen als Verfahrenskombination einzuführen für die Elimination partiell toter und redundanter Anweisungen (EPTRA) :

- EPTRA $= (\mathrm{AS} + \mathrm{ETA} + \mathrm{AH} + \mathrm{ERA})^*$

Erwartung:

- EPTRA ist mächtiger als EPTA und EPRA für sich!

Zur Wiederholung

Bezeichne:

- $T^G_{AS,ETA,AH,ERA} = \{\tau \mid \tau = (\tau_i)_{i \in \mathbb{N}} \in \mathcal{L}((AS + ETA + AH + ERA)^*)\}$

die Menge aller Transformationsfolgen aus zulässigen Anweisungssenkungen, -hebungen und Eliminationen toter oder redundanter Anweisungen für ein Programm G.

Geht G aus dem Kontext hervor, schreiben wir statt $T^G_{AS,ETA,AH,ERA}$ einfacher $T_{AS,ETA,AH,ERA}$.

Ist $\tau = (\tau_i)_{i \in \mathbb{N}}$ eine Transformationsfolge, so bezeichne:

- $(\tau_i)_{i \leq k}$ das Anfangsstück von τ bis zum Index k einschließlich.
- τ_j die Elementartransformation mit Index j von τ .
- G_τ, $G_{(\tau_i)_{i \leq k}}$ und $G_{(\tau_j)}$ diejenigen Programme, die aus G durch Anwendung von τ, $(\tau_i)_{i \leq k}$, auf G entstehen.

EPTA/EPRA: Konfluenz, Terminierung

...für die EPTA/EPRA-Transformationsrelationen gilt (s. Kapitel 15.3 und 15.4):

Theorem 16.1.1.1 (Konfluenz, Terminierung)

Die EPTA- und EPRA-Transformationsrelationen

1. $\cdot \vdash_\tau \cdot, \tau \in AS \cup ETA$ (EPTA)
2. $\cdot \vdash_\tau \cdot, \tau \in AH \cup ERA$ (EPRA)

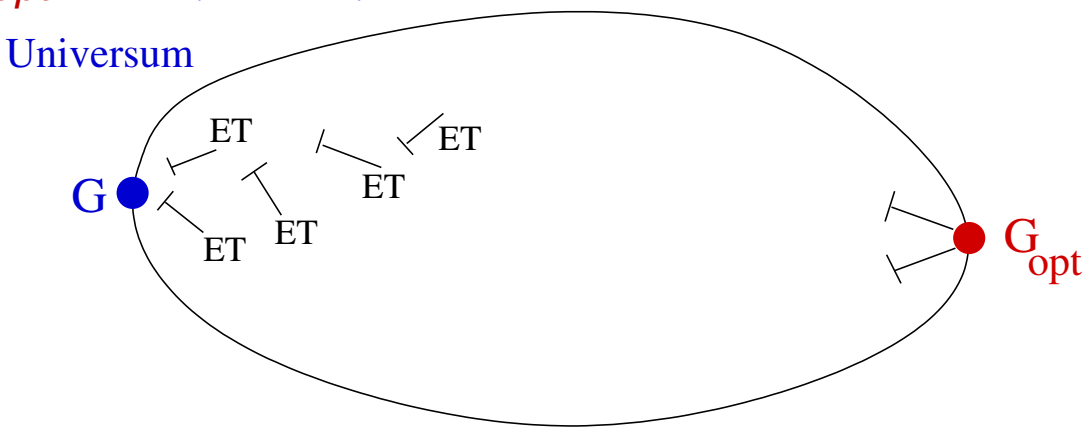
sind (bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken) konfluent und terminierend.

EPTA/EPRA: Globale Optimalität

...für faire EPTA/EPRA-Transformationsfolgen gilt (s. Kapitel 15.3 und 15.4):

Theorem 16.1.1.2 (Globale EPTA-/EPRA-Opt.)

1. Jede faire EPTA- und EPRA-Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$ ist global optimal, d.h. endet in einem bis auf irrelevante Umsortierungen in Basisblöcken eindeutig bestimmten global optimalen Programm $G_{opt} \in \mathcal{G}_T^{opt}(G)$.
2. Jede faire EPTA- und EPRA-Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$ hat ein endliches Anfangsstück $\tau' = (\tau_i)_{i \leq k}$ mit $G_{opt} \sim G_\tau \sim G_{\tau'}$.



Kapitel 16.1.2

EPTRA: Transformation

EPTRA-Transformationsfolgen

Sei G ein Programm.

Definition 16.1.2.1 (EPTRA-Transformationsfolge)

1. Eine EPTRA-Transformationsfolge (oder EPTRA-Transformation) ist eine beliebige Abfolge zulässiger Anweisungssenkungen, -hebungen und Eliminationen toter oder redundanter Anweisungen.
2. T^G_{EPTRA} bezeichne die Menge aller EPTRA-Transformation(sfolg)en für G, in Zeichen:

$$T^G_{EPRTA} = T^G_{AS,ETA,AH,ERA}$$

3. Ist $\tau \in T^G_{EPTRA}$, so bezeichnet G_τ das Programm, das τ angewendet auf G liefert.

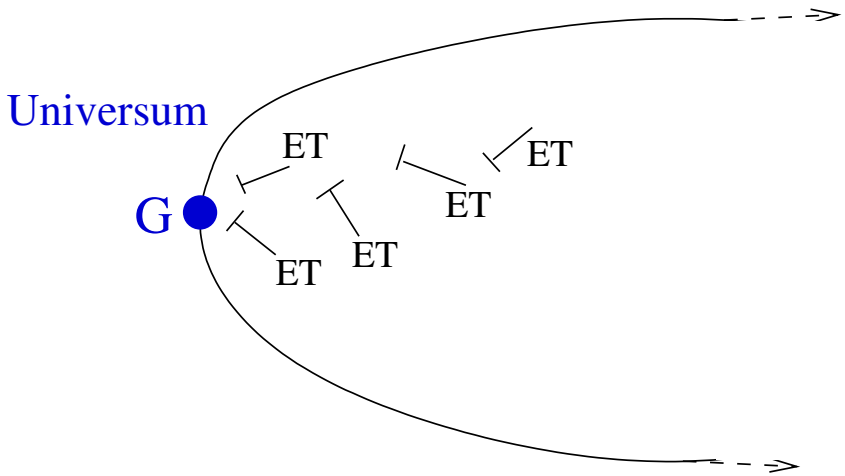
Transformationsrelation, Programmuniversum

- Transformationsrelation:

  $G \vdash_\tau G'$, $\tau \in AS \cup ETA \cup AH \cup ERA$: G' resultiert aus G durch Anwendung einer zulässigen Senkungs-, Hebungs- oder Eliminationstransformation toter oder redundanter Anweisungen.

- Induziertes Programmuniversum:

  $\mathcal{U}^G_{T_{EPRTA}} =_{df} \{G' \mid G \vdash_{(\tau_i)_{i \leq k}} G', \tau = (\tau_i)_{i \in \mathbb{N}} \in T^G_{EPTRA}, k \in \mathbb{N}\}$: Das von G durch die Präfixe der Transformationsfolgen $\tau \in T^G_{EPTRA}$ aufgespannte Universum.



Kapitel 16.1.3

EPTRA: Besser, best, optimal

Vergleichsrelation 'besser' für Programme

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Programm und seien $G', G'' \in \mathcal{U}^G_{T_{EPTRA}}$.

Definition 16.1.3.1 (Besser)

G' heißt besser als G'' (in Zeichen: $G'' \sqsubseteq G'$) gdw G' ist besser als G'' i.S.v. Definition 15.3.6.1, d.h.:

$$\forall\, p \in \mathbf{P}_G[\mathbf{s}, \mathbf{e}]\ \forall\, \alpha \in \mathcal{AM}.\ \#_\alpha(p_{G'}) \leq \#_\alpha(p_{G''})$$

wobei $\#_\alpha(p_{G'})$ und $\#_\alpha(p_{G''})$ die Anzahl von Anweisungen des Anweisungsmusters α auf p in G' bzw. G'' bezeichnen.

Eigenschaften der Relation 'besser'

Lemma 16.1.3.2 (Quasiordnung)

Die Programmvergleichsrelation besser $\sqsubseteq_{\sim}$ ist eine Quasiordnung (d.h. reflexiv und transitiv, aber nicht antisymmetrisch).

Lemma 16.1.3.3 (Verbesserung, Verb.-Neutralität)

Seien G und G' zwei Programme mit $G \vdash_\tau G'$ und $\tau \in AS \cup ETA \cup AS \cup ERA$. Dann gilt:

1. $G \sim_T G'$, falls τ eine zulässige Anweisungssenkung oder -hebung ist.
2. $G \sqsubset_{\not\sim T} G'$, falls τ eine nichttriviale ($\neq Id$) zulässige Elimination toter oder redundanter Anweisungen ist.

...d.h.: Eliminationen bewirken echte Verbesserungen, während Senkungen und Hebungen verbesserungsneutral sind, aber durch spätere Eliminationen als Effekte zweiter Ordnung mittelbar Verbesserungen ermöglichen können.

Global beste (oder optimale) Programme

Sei G ein Programm.

Definition 16.1.3.4 (Global beste (optimale) Prg.)

1. Ein Programm $G^* \in \mathcal{U}^G_{T_{EPTRA}}$ heißt global EPTRA-best (oder global EPTRA-optimal) gdw G^* ist besser als jedes andere Programm aus $\mathcal{U}^G_{T_{EPTRA}}$:

$$\forall\, G' \in \mathcal{U}^G_{T_{EPTRA}}.\ G' \sqsubseteq_{\approx} G^*$$

2. Bezeichne $\mathcal{G}^{opt}_{T_{EPTRA}}(G)$ die Menge der global EPTRA-besten (oder global EPTRA-optimalen) Programme in $\mathcal{U}^G_{T_{EPTRA}}$.

Lokal beste (optimale) Programme

Sei G ein Programm.

Definition 16.1.3.5 (Lokal beste (optimale) Prg.)

1. Ein Programm $G^* \in \mathcal{U}^G_{T_{EPTRA}}$ heißt lokal EPTRA-best (oder lokal EPTRA-optimal) gdw:

$$\forall\, G' \in \mathcal{U}^G_{T_{EPTRA}}.\ G' \approx G^*$$

2. Bezeichne $\mathcal{G}^{lokopt}_{T_{EPTRA}}(G)$ die Menge der lokal EPTRA-besten (oder lokal EPTRA-optimalen) Programme in $\mathcal{U}^G_{T_{EPTRA}}$.

Intuitiv: Ein Programm ist lokal optimal, wenn beliebige weitere Anwendungsfolgen von Elementartransformationen nicht mehr zu einer echten Verbesserung führen, sondern nur Anweisungen durch Senkungen oder Hebungen an anderen Programmstellen platzieren.

Vergleichsrelation 'besser' für Transf.-Folgen

Sei G ein Programm.

Definition 16.1.3.6 (EPRTA-bessere Transf.)

Eine Transformationsfolge (oder Transformation) $\tau \in \mathsf{T}^G_{EPRTA}$ heißt EPTRA-besser für G als $\tau' \in T_{EPTRA}$ gdw: $G_{\tau'} \sqsubseteq\!\!\!\!\sim\; G_\tau$

Definition 16.1.3.7 (Global, lokal EPTRA-beste T.)

Eine Transformationsfolge (oder Transformation) $\tau \in T^G_{EPTRA}$ heißt

1. global EPTRA-best für G gdw τ ist EPTRA-besser für G als jede andere Transformation in T^G_{EPTRA}.
2. lokal EPTRA-best für G gdw keine EPTRA-Verlängerung von τ ist echt EPTRA-besser als τ, d.h. τ ist genauso gut wie jede Verlängerung von τ.

Kapitel 16.1.4

EPTRA: Optimalität

EPTRA: Nicht-Konfluenz, Termination

...für die EPTRA-Transformationsrelation gilt:

Theorem 16.1.4.1 (Nicht-Konfluenz, Terminierung)

Die EPTRA-Transformationsrelation

$$\cdot \vdash_\tau \cdot, \tau \in AS \cup ETA \cup AH \cup ERA \quad \text{(EPTRA)}$$

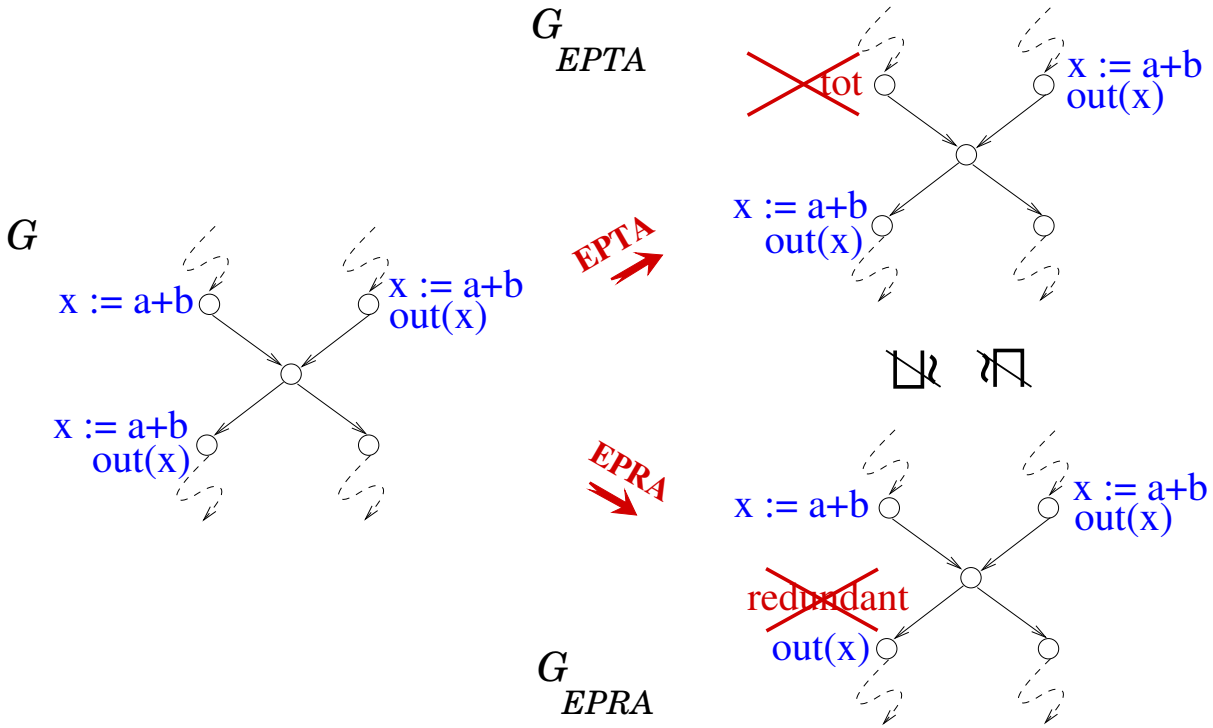
ist (bis auf irrelevante Umsortierungen von Anweisungen im Programm) terminierend, aber (i.a.) nicht konfluent.

Beachte: Irrelevante Umsortierungen sind nicht länger auf Basisblöcke beschränkt, sondern können das gesamte Programm betreffen, da sowohl irrelevante Hebungen als auch Senkungen über Basisblockgrenzen hinaus möglich sind.)

Beweisskizze zu Theorem 16.1.4.1

...durch Angabe eines Beispiels:

Betrachte die Effekte von EPTA und EPRA auf Programm G:



Offenbar gilt: G_{EPTA} und G_{EPRA} sind unvergleichbar bezüglich der Relation besser $\sqsubseteq$; es gilt:

$$G_{EPTA} \not\sqsubseteq G_{EPRA} \ \wedge \ G_{EPRA} \not\sqsubseteq G_{EPTA}$$

EPTRA: Lokale, keine globale Optimalität

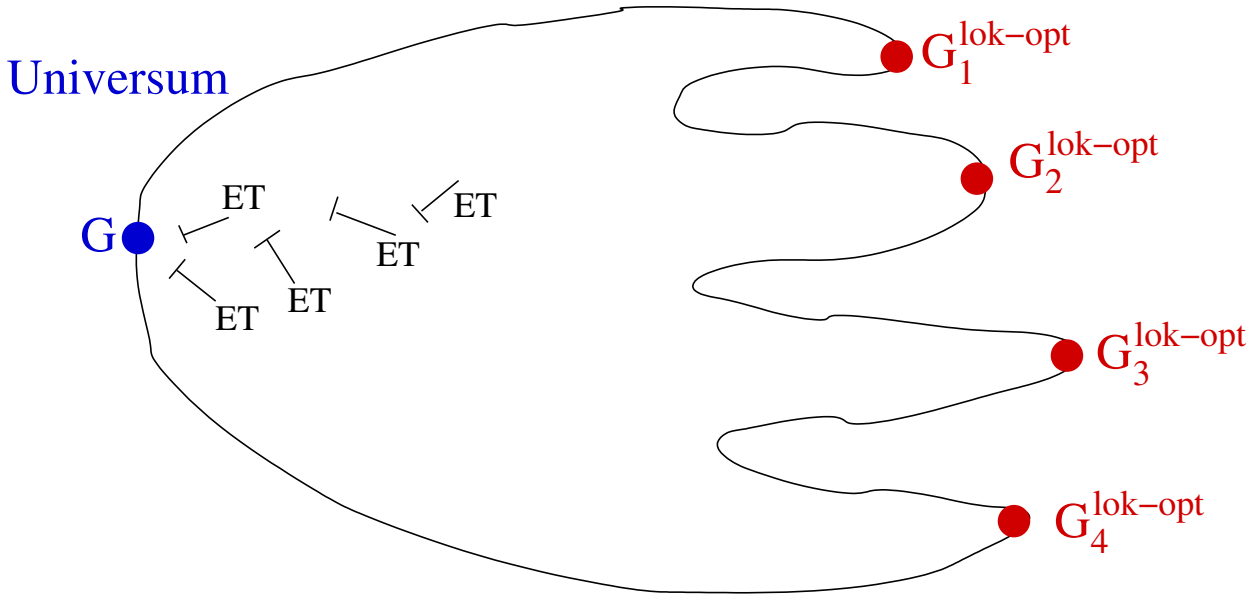
...für faire EPTRA-Transformationsfolgen gilt:

Theorem 16.1.4.2 (Lokale EPTRA-Optimalität)

1. Jede faire EPTRA-Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$ ist lokal optimal, d.h. endet in einem bis auf irrelevante Umsortierungen von Anweisungen eindeutig bestimmten lokal optimalen Programm $G_{lokopt} \in \mathcal{G}^{lokopt}_{EPTRA}(G)$.
2. Jede faire bis auf irrelevante Umsortierungen in einem Programm $G_{lokopt} \in \mathcal{G}^{lokopt}_{EPTRA}(G)$ endende EPTRA-Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$ hat ein endliches Anfangsstück $\tau' = (\tau_i)_{i \leq k}$ mit $G_{lokopt} \sim G_\tau \sim G_{\tau'}$.
3. Global optimale EPTRA-Transformationsfolgen und -Programme existieren i.a. nicht.

EPTRA: Verlust von Konfluenz, globaler Opt.

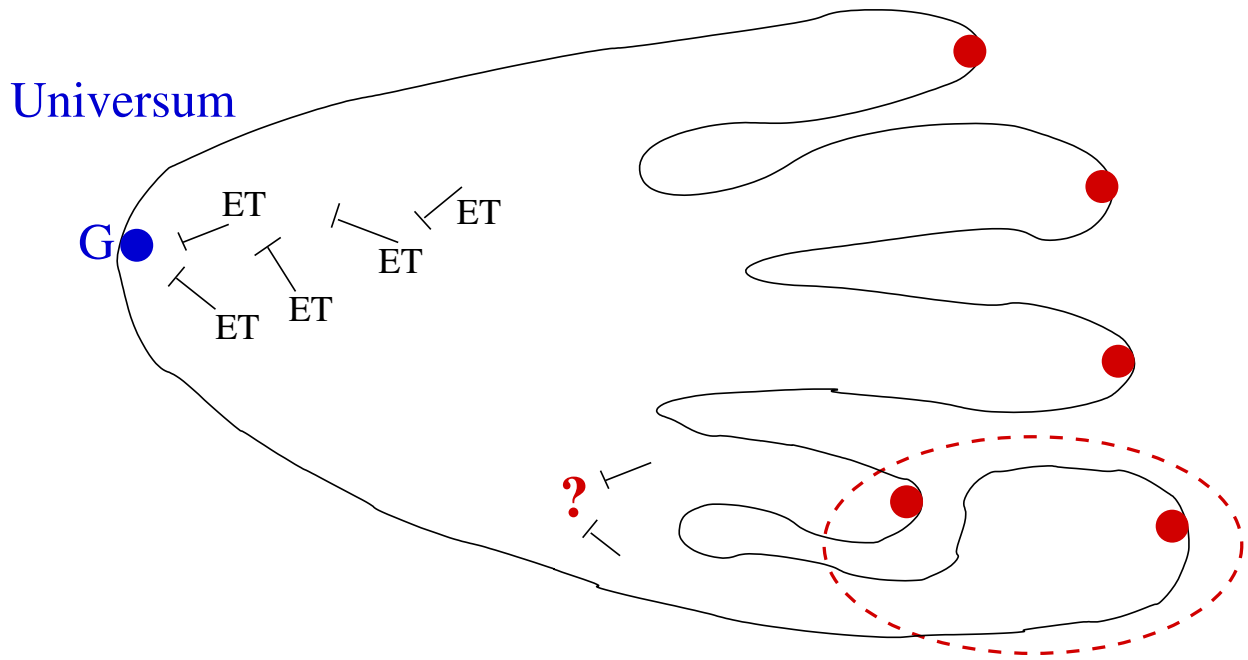
...Konfluenz und globale Optimalität sind für EPTRA verloren!



Lokale Optimalität bleibt für EPTRA zumindest erhalten!

EPTRA: Verlust auch lokaler Optimalität

...allerdings möglich in speziellen Szenarien aufgrund spezieller Instruktionen (konkret: High-Performance Fortran):



Für Details siehe:

- Jens Knoop, Eduard Mehofer. Distribution Assignment Placement: Effective Optimization of Redistribution Costs. IEEE Transactions on Parallel and Distributed Systems 13(6):628-647, 2002.

Kapitel 16.1.5

EPTRA: Purismus vs. Pragmatismus

Purismus vs. Pragmatismus

...aus theoretischer Sicht ist das lokale Optimalitätsergebnis für

- EPTRA = (AS + ETA +AH + ERA)*

weniger elegant und befriedigend als die globalen Optimalitätsergebnisse für

- EPRA = (AH + ERA)*
- EPTA = (AS + ETA)*

Aus pragmatischer Sicht ist allerdings

- EPTRA

EPRA ebenso wie EPTA überlegen, weil wirkmächtiger; im Fall von High-Performance Fortran durch die Einsparung besonders rechenaufwändiger unnötiger Distributionsanweisungen sogar in der Größenordnung mehrerer hundert Prozent!

EPTRA wirkmächtiger als EPRA und EPTA

Für die bis auf irrelevante Umsortierungen in Programmen bzw. Basisblöcken eindeutig bestimmten lokal bzw. global optimalen Programme

$G^{lokopt}_{EPTRA} \in \mathcal{G}^{lokopt}_{T_{EPTRA}}(G)$, $G^{opt}_{EPRA} \in \mathcal{G}^{opt}_{T_{EPRA}}(G)$ und $G^{opt}_{EPTA} \in \mathcal{G}^{opt}_{T_{EPTA}}(G)$

gilt: G^{lokopt}_{EPTRA} ist besser (i.S.v. Definition 15.3.6.1) als G^{opt}_{EPRA} und G^{opt}_{EPTA}:

Lemma 16.1.5.1 (EPTRA besser als EPRA, EPTA)

$$\forall\, G \in \mathcal{G}.\ G^{opt}_{EPRA} \sqsubseteq_{\approx} G^{lokopt}_{EPTRA} \ \wedge\ G^{opt}_{EPTA} \sqsubseteq_{\approx} G^{lokopt}_{EPTRA}$$

Übungsaufgabe 16.1.5.2

Zeige, dass die Erwartung, dass EPTRA mächtiger ist als EPTA und EPRA für sich, begründet ist:

Finde dazu ein Programm G, so dass EPTRA angewendet auf G zu einem performanteren Programm führt als EPTA und EPRA jeweils für sich alleine.

Kapitel 16.2

EPRAA: EPRA/EPRAd-Kombination

Kapitel 16.2.1

EPRA, EPRAd: Grundtransformationen

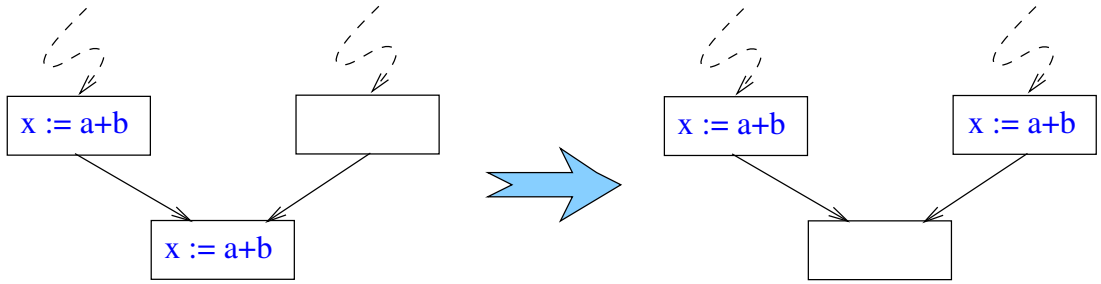
EPRA und EPRAd

...zwei Grundtransformationen zur Elimination redundanten Berechnungsaufwands:

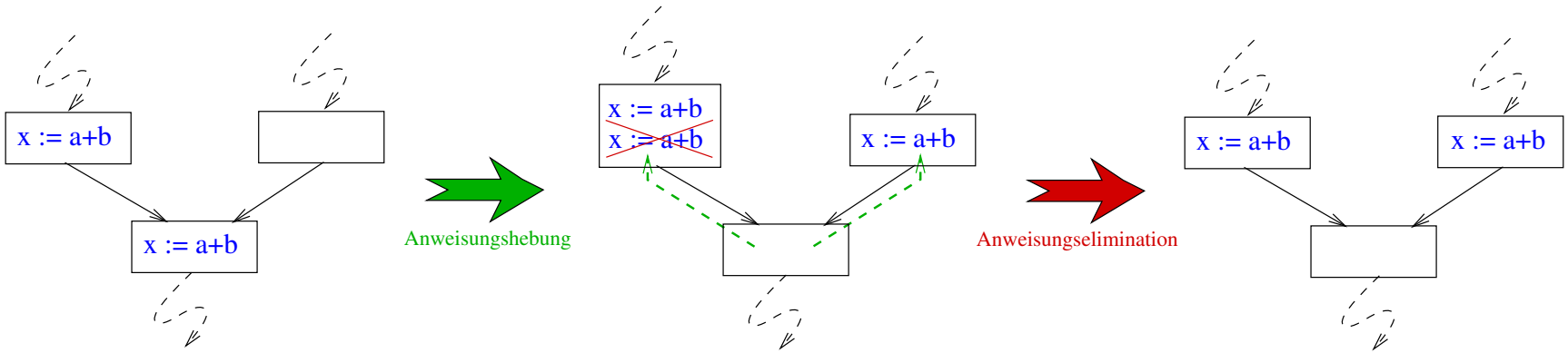
- Elimination partiell redundanter Ausdrücke (EPRAd)
  - $\rightsquigarrow$ Partially Redundant Expression Elimination (PREE)
  - $\rightsquigarrow$ Expression Motion (EM)
- Elimination partiell redundanter Anweisungen (EPRA)
  - $\rightsquigarrow$ Partially Redundant Assignment Elimination (PRAE)
  - $\rightsquigarrow$ Assignment Motion (AM)

Elimination partiell redundanter Anweisungen

Das EPRA-Grundmuster:

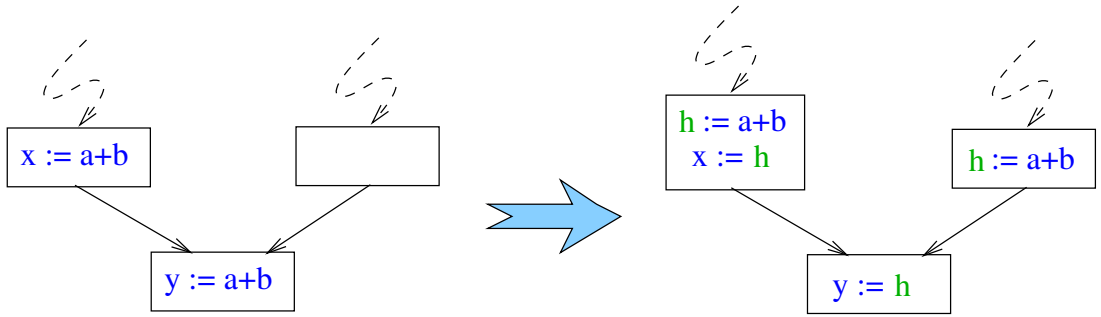


Die konzeptuelle EPRA-Verfahrensidee:

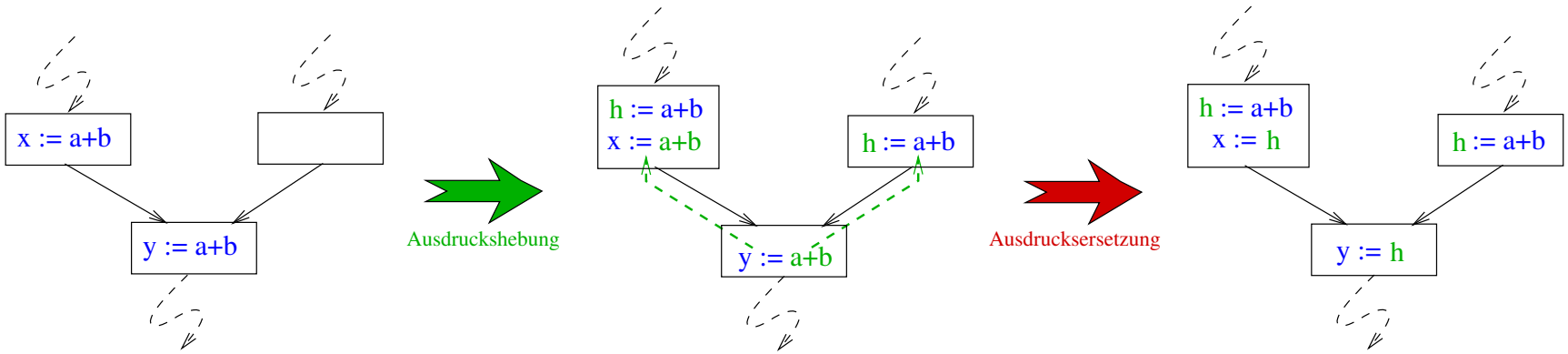


Elimination partiell redundanter Ausdrücke

Das EPRAd-Grundmuster:

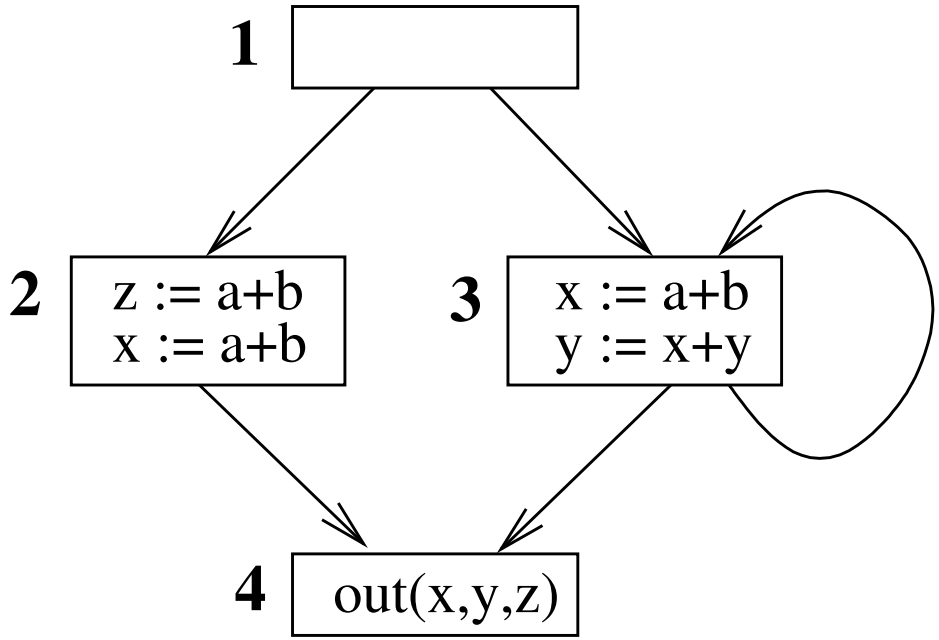


Die konzeptuelle EPRAd-Verfahrensidee:



Ein gemeinsames Beispiel

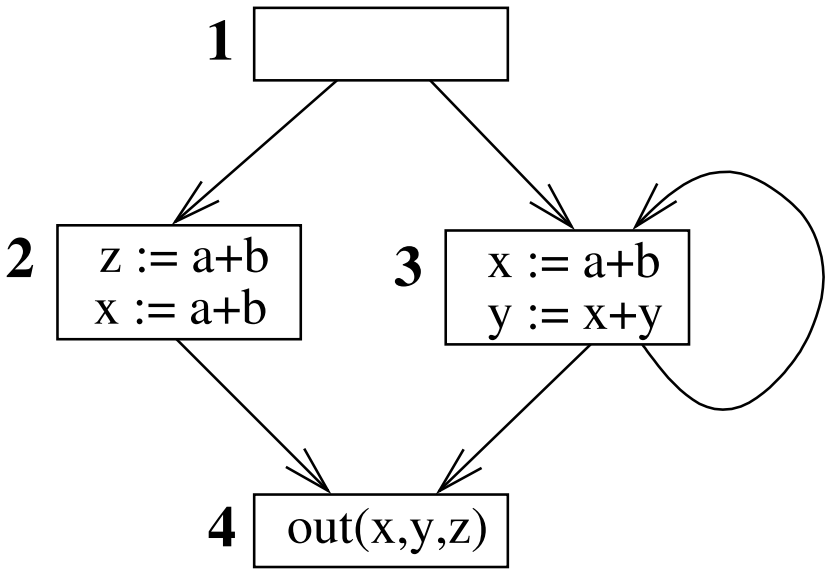
...für die Illustration der verschiedenen Transformationseffekte:



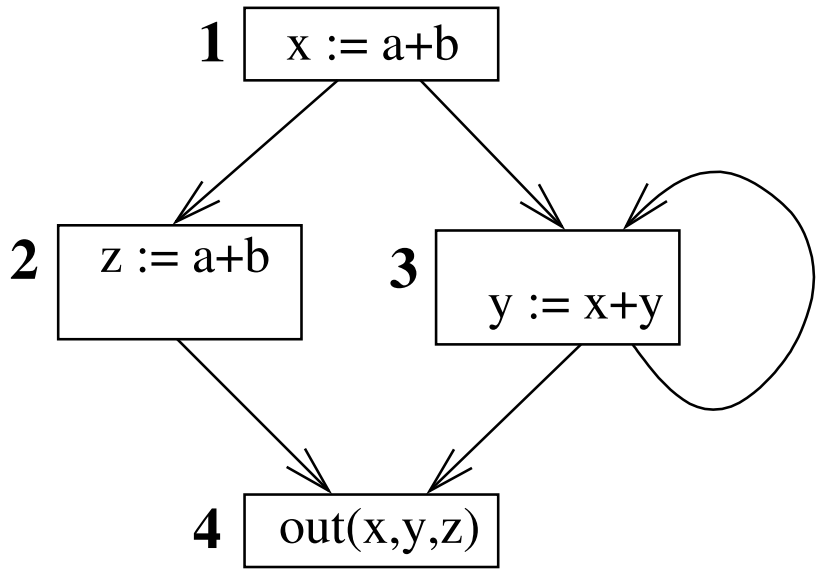
Elimination partiell redundanter Anweisungen

...der EPRA-Effekt auf das gemeinsame Beispiel:

Ausgangsprogramm



Optimiertes Programm

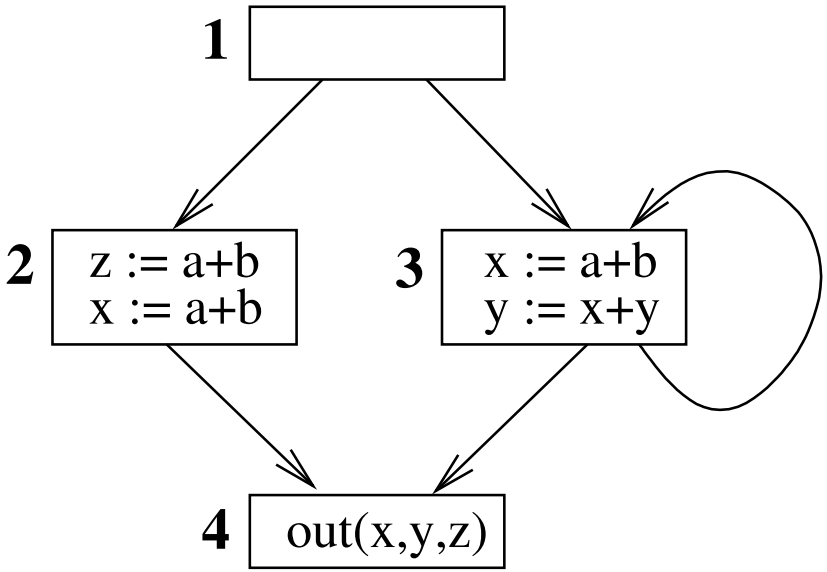


Beachte: Wären die kritischen Kanten $(1,3)$ und $(3,3)$ gespalten, würden maximale EPRA-Transformationen die Anweisung `y := x+y` vom Knoten 3 in die synthetischen Knoten $S_{1,3}$ und $S_{3,3}$ schieben, was bezüglich der Relation 'besser' keinen Unterschied zum obigen optimierten Programm machte.

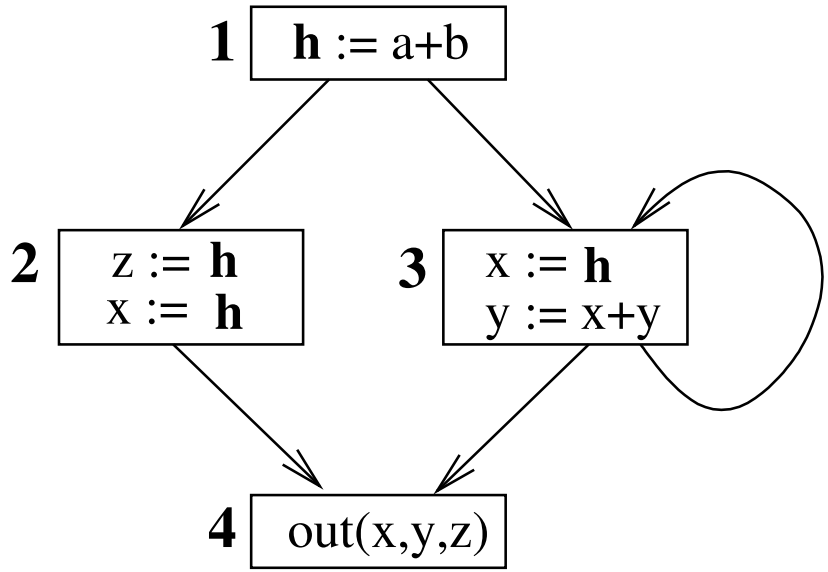
Elimination partiell redundanter Ausdrücke

...der EPRAd-Effekt auf das gemeinsame Beispiel:

Ausgangsprogramm



Optimiertes Programm



Beachte: Der Term `x+y` ist nicht schleifeninvariant. Merken seines Wertes in einer Hilfsvariablen und Wiederbenutzung des gemerkten Werts führt deshalb nicht zu einer Verbesserung.

Globale Optimalität d. Grundtransformationen

...konzeptuell können die

- Elimination partiell redundanter Anweisungen (EPRA)
- Elimination partiell redundanter Ausdrücke (EPRAd)

im Sinne folgender regulär-ähnlicher Ausdrücke verstanden werden:

- EPRA $= (\mathrm{AH} + \mathrm{ERA})^*$
- EPRAd $= \mathrm{AdH}_{max} * \mathrm{ETRAd}_{max} * (\mathrm{AdS}_{max})^k,\ k \in \{0, 1\}.$

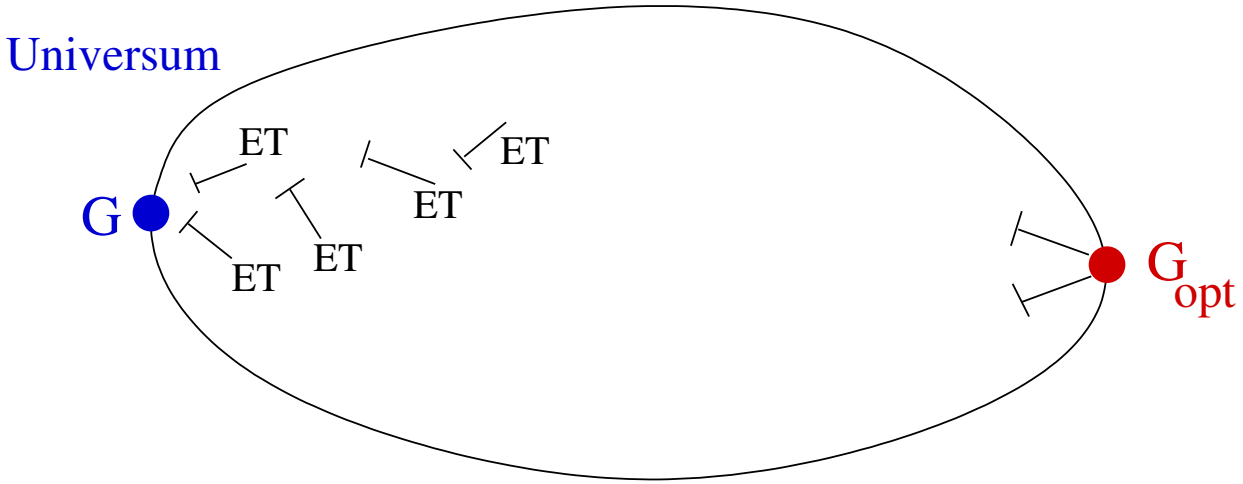
Beachte: EPRAd ist anders als EPRA frei von Effekten zweiter Ordnung: Eine maximale Ausdruckshebung (oder Vorziehen) gefolgt von einer anschließenden einmaligen Elimination total redundanter Ausdrücke und einer optionalen berechnungsneutralen maximalen Zurückschiebung liefert deshalb bereits die bestmögliche Verbesserung und optional geringste Zahl an Hilfsvariableninitialisierungen und -lebenszeiten.

Globale Optimalität von EPRA

...für EPRA gilt (s. Kapitel 15.4):

Theorem 16.2.1.1 (Globale EPRA-Optimalität)

1. Jede faire EPRA-Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$ für ein Programm G ist global optimal, d.h. endet in einem bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken eindeutig bestimmten global optimalen Programm $G_{opt} \in \mathcal{G}^{opt}_{T_{EPRA}}(G)$.
2. Jede faire EPRA-Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$ hat ein endliches Anfangsstück $\tau' = (\tau_i)_{i \leq k}$ mit $G_{opt} \sim G_\tau \sim G_{\tau'}$



Globale Optimalität von EPRAd

...für EPRAd gilt (s. Vorlesung 185.A04 Optimierende Übersetzer):

Theorem 16.2.1.2 (Globale Berechnungsoptimalität)

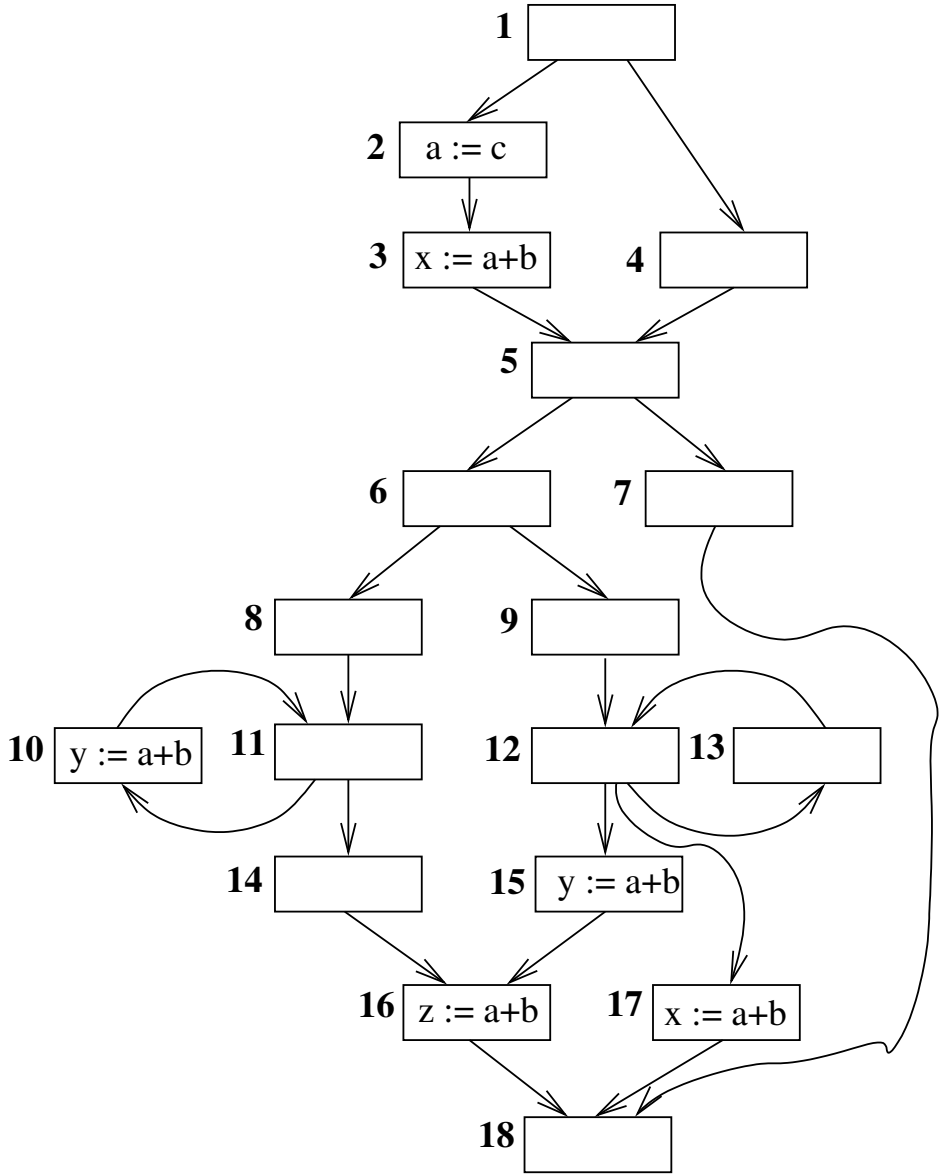
Initialisieren der Hilfsvariablen an den frühest möglichen sicheren Programmpunkten führt zu global besten berechnungsoptimalen Programmen.

Theorem 16.2.1.3 (Globale Lebenszeitoptimalität)

Initialisieren der Hilfsvariablen an den spätest möglichen berechnungsoptimalen Programmpunkten führt zu global besten berechnungs- und hilfsvariablenlebenszeitoptimalen Programmen.

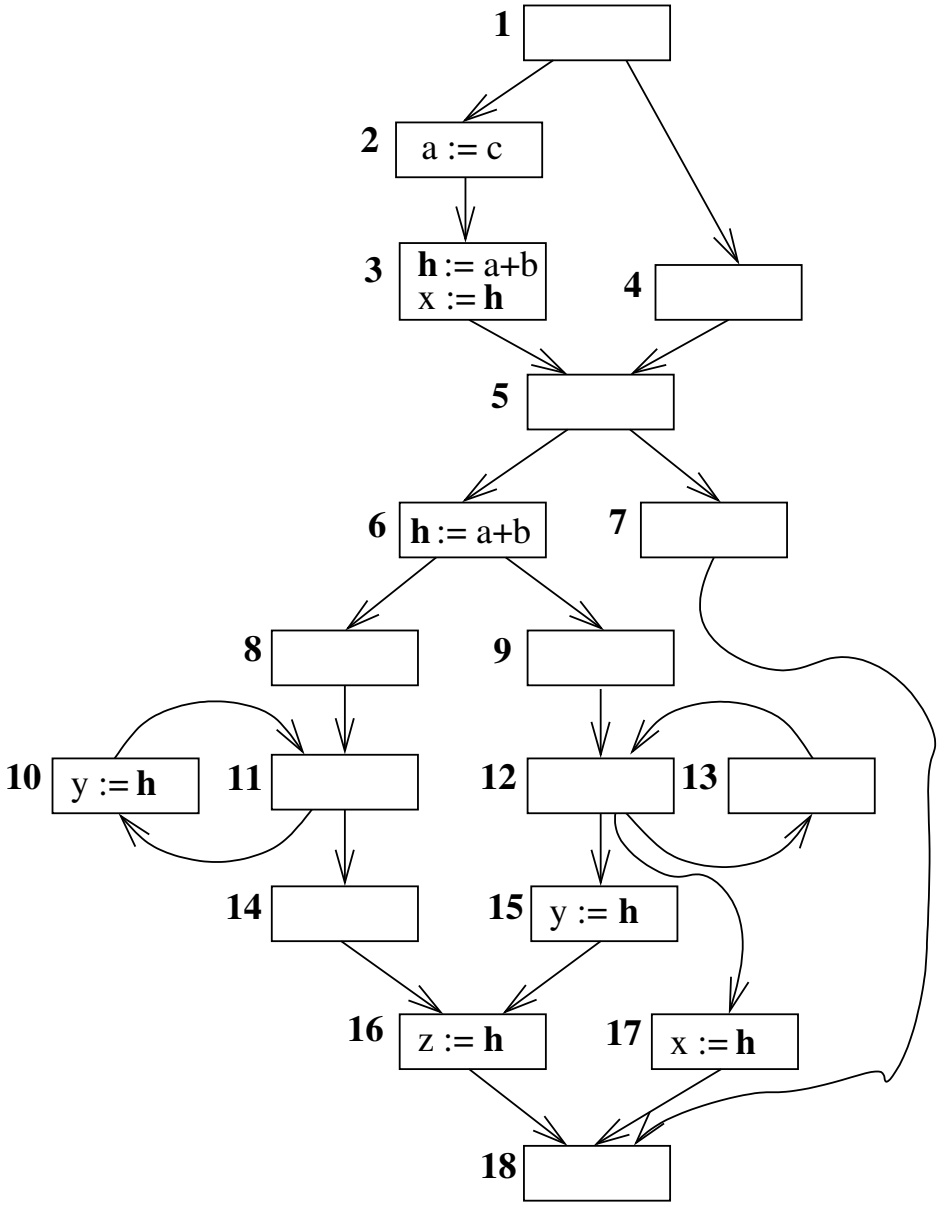
Veranschaul. v. Theorem 16.2.1.2 u. 16.2.1.3

...anhand eines Beispiels:



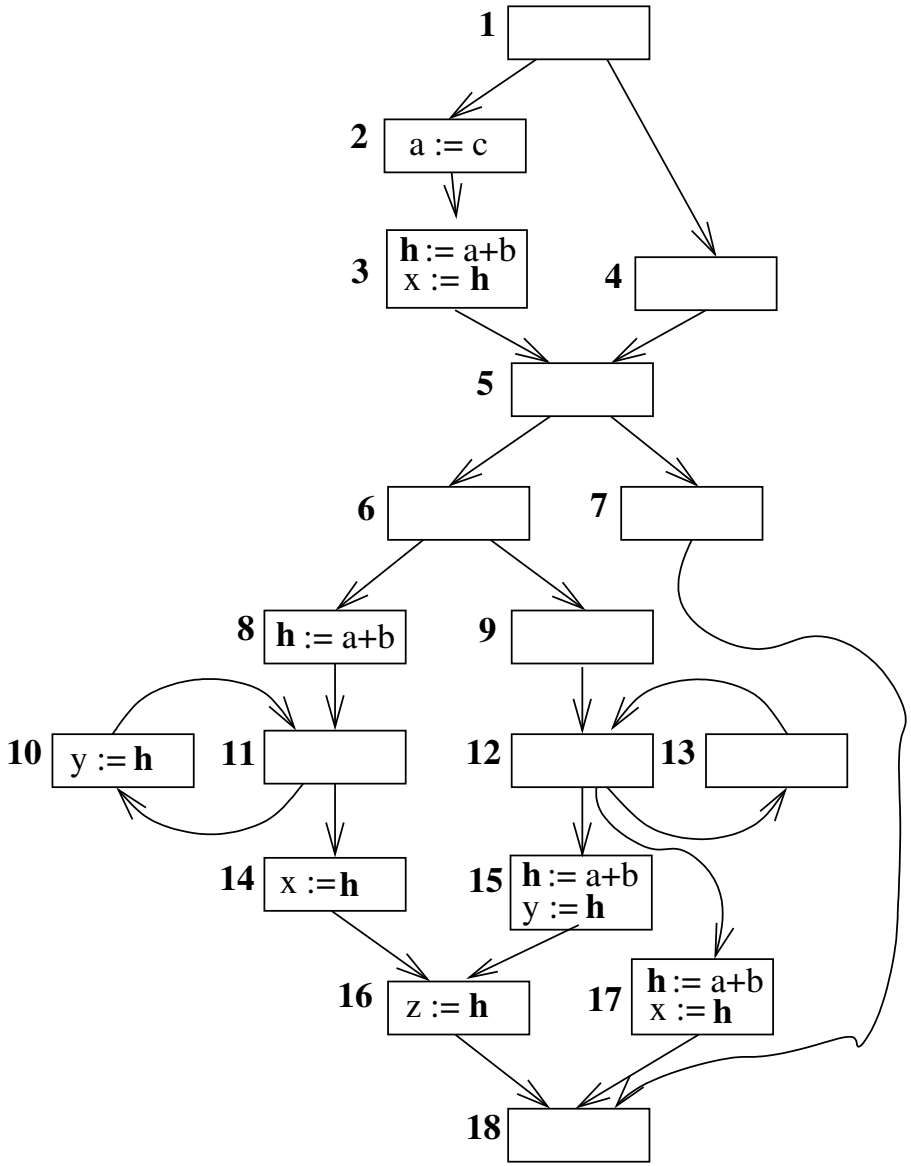
Th. 16.2.1.2: Frühestmögl. sichere Platzierung

...ist global berechnungsoptimal:



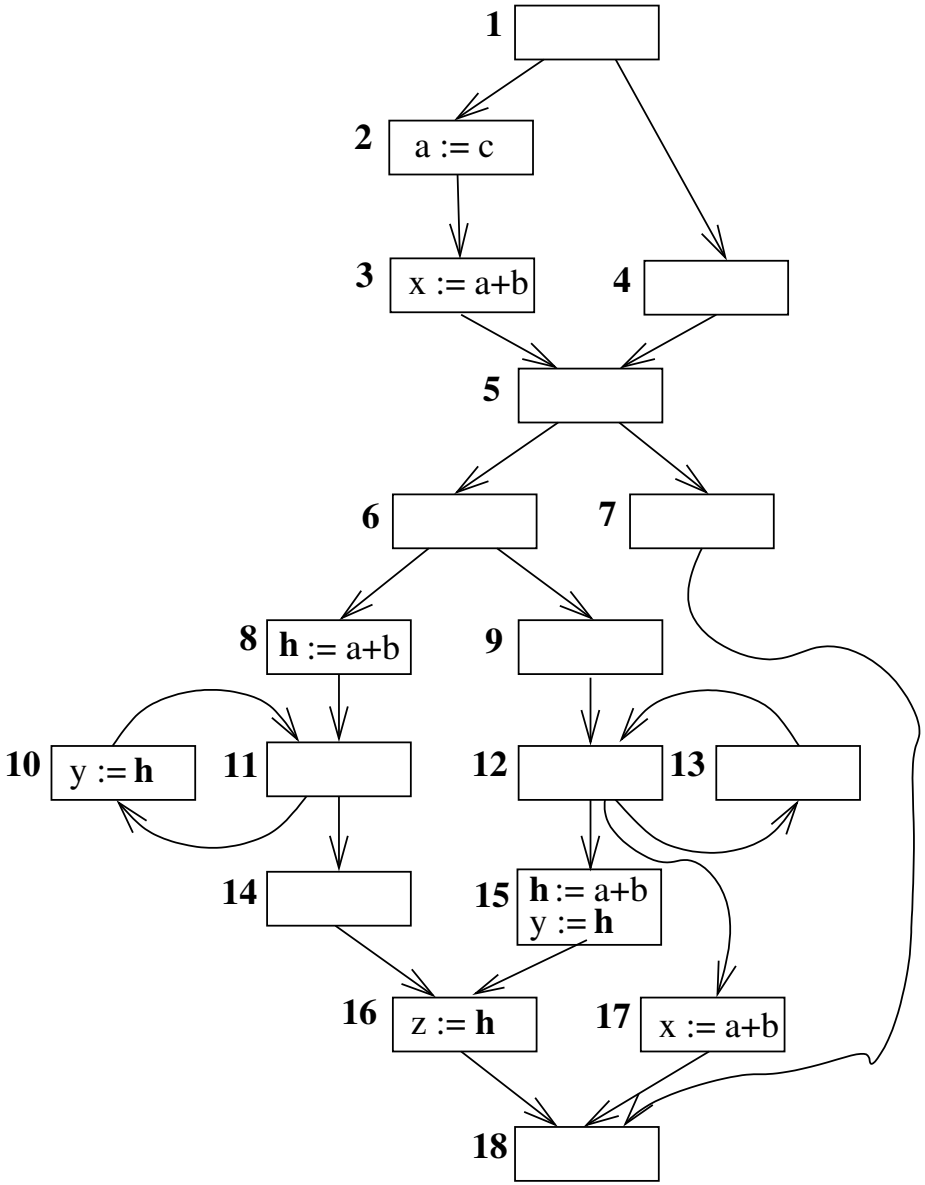
Th. 16.2.1.3: Spätestm. berechn.-opt. Platz.

...ist global berechnungs- und hilfsvariablenlebenszeitoptimal:



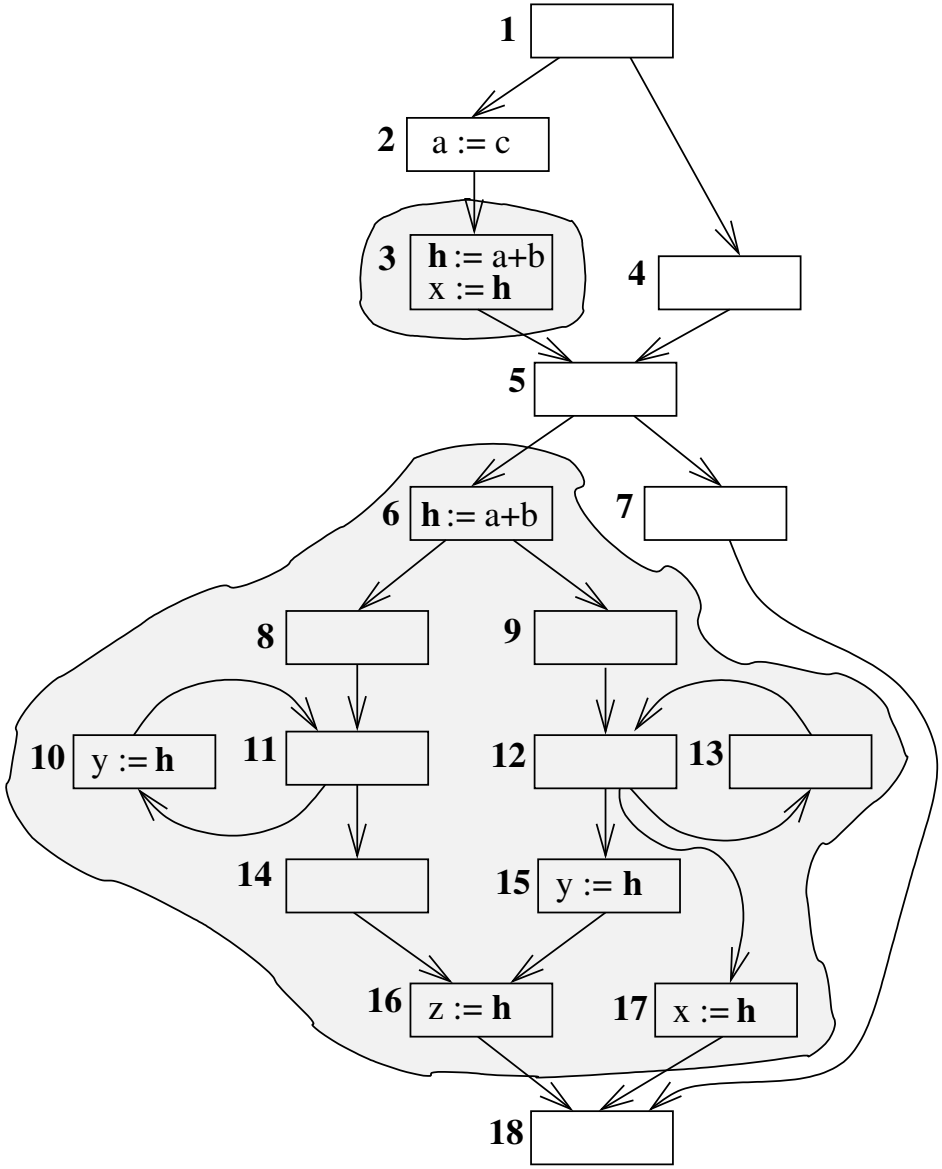
Th. 16.2.1.3: Spätestm. berechn.-opt. Platz.

...nach abschließendem 'Aufräumen':



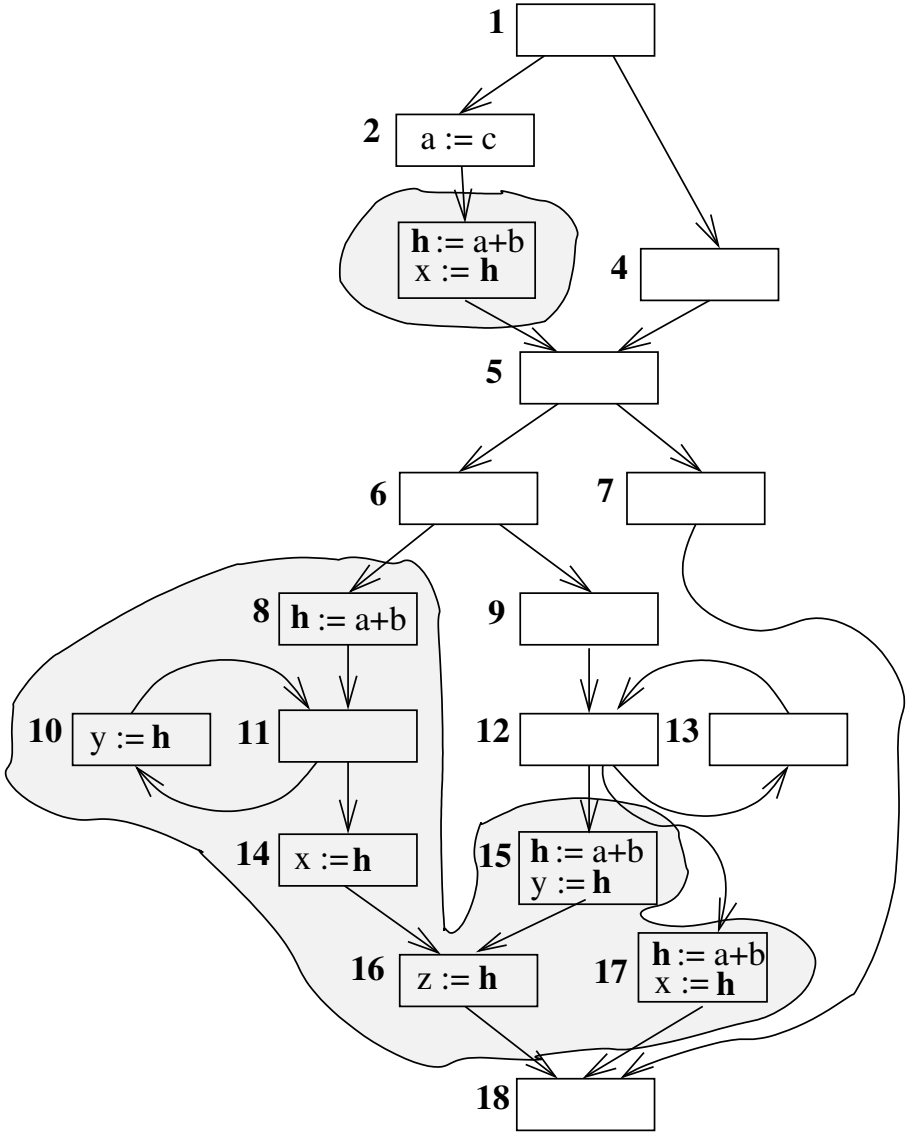
Frühestmögliche sichere Platzierung

...berechnungsoptimal, aber max. Hilfsvariablenlebenszeiten.



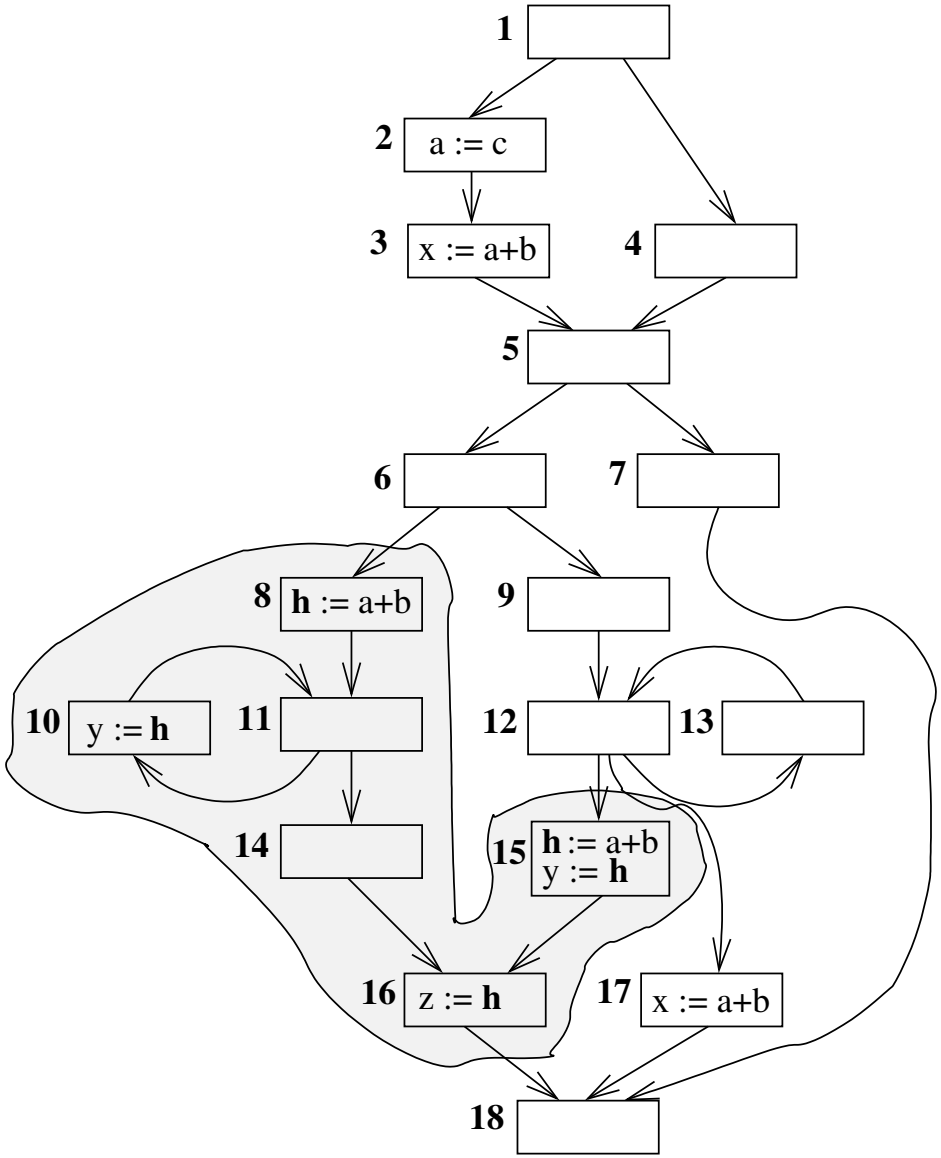
Spätestmögliche berechn.-optimale Platzierung

...berechnungsoptimal, aber min. Hilfsvariablenlebenszeiten.



Spätestmögliche berechn.-optimale Platzierung

...nach ‘Aufräumen’: min. Hilfsvariableninitialisierungen und -lebenszeiten.



Kapitel 16.2.2

EPRAA: Transformation

Die EPRAA-Transformation

...EPRAA, einheitliche, kombinierte Transformation zur:

- Elimination partiell redundanter Ausdrücke und Anweisungen (EPRAA)
  - $\rightsquigarrow$ Partially Redundant Expression and Assignment Elimination (PREAE)
  - $\rightsquigarrow$ Expression and Assignment Motion (EAM)

Das EPRAA-Verfahren

...ein dreistufiger Algorithmus:

1. Präprozess
   Ersetze jedes Vorkommen einer Anweisung `x := t` durch die Anweisungssequenz $\texttt{h}_t$ `:= t;` `x :=` $\texttt{h}_t$.
2. Hauptprozess
   Wende die Transformationen
   2.1 Heben von Anweisungen (AH)
   $\rightsquigarrow$ Assignment Hoisting
   2.2 Eliminieren (total) redundanter Anweisungen (ERA)
   $\rightsquigarrow$ (Totally) Redundant Assignment Elimination
   wiederholt so lange an bis Stabilität eintritt.
3. Postprozess
   Aufräumen isolierter Initialisierungen.

Der Präprozess ist Schlüssel

...zur einheitlichen Behandlung von Ausdrücken und Anweisungen.

Die Einführung spezifischer Hilfsvariablen $\mathtt{h}_t$ für jedes rechtsseitig in einer Anweisung auftretende Termmuster $\mathtt{t}$ im Präprozess bewirkt, dass

- EPRA als Hauptprozess des EPRAA-Verfahrens

die Effekte von EPRA und EPRAd einheitlich erfasst und abdeckt!

Dabei gilt:

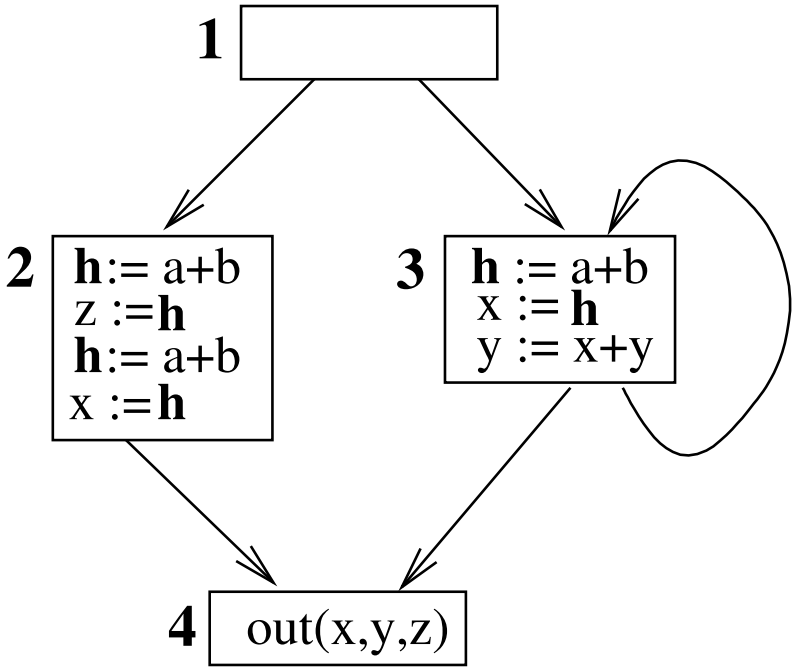
- 'Das Ganze ist mehr als die Summe seiner Teile':

$$\text{EPRAA} > \text{EPRA} + \text{EPRAd}$$

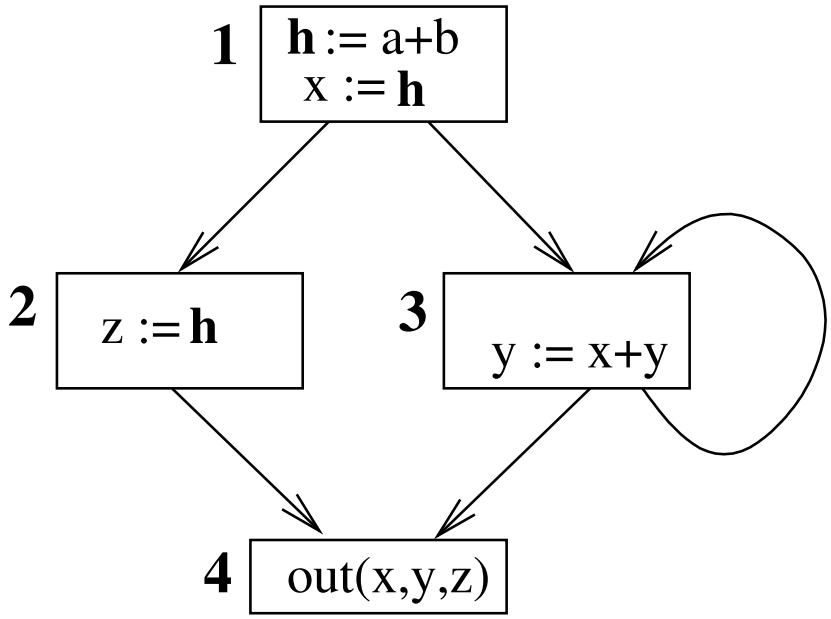
Einheitl. Elimination part. red. Ausd. u. Anw.

...der EPRAA-Effekt auf das gemeinsame Beispiel:

Ausgangsprogramm nach Präprozess



Optimiertes Programm



Effekte zweiter Ordnung für EPRAA

...(engl. second order effects) im EPRAA-Fall:

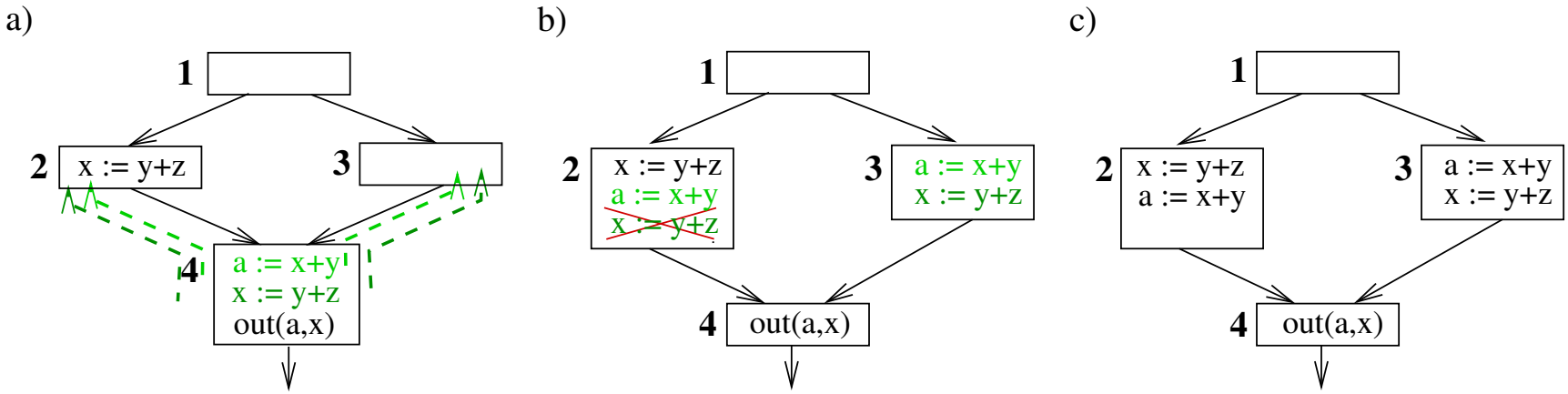
- Hebungs-Eliminations-Effekte (Zieleffekt)
  $\rightsquigarrow$ Hoisting-Elimination effects (HE)
- Hebungs-Hebungs-Effekte (Potentialeffekt)
  $\rightsquigarrow$ Hoisting-Hoisting effects (HH)
- Eliminations-Hebungs-Effekte (Potentialeffekt)
  $\rightsquigarrow$ Elimination-Hoisting effects (EH)
- Eliminations-Eliminations-Effekte (Zieleffekt)
  $\rightsquigarrow$ Elimination-Elimination effects (EE)

Beispiel für einen Hebungs-Eliminations-Effekt

Ausgangsprogramm

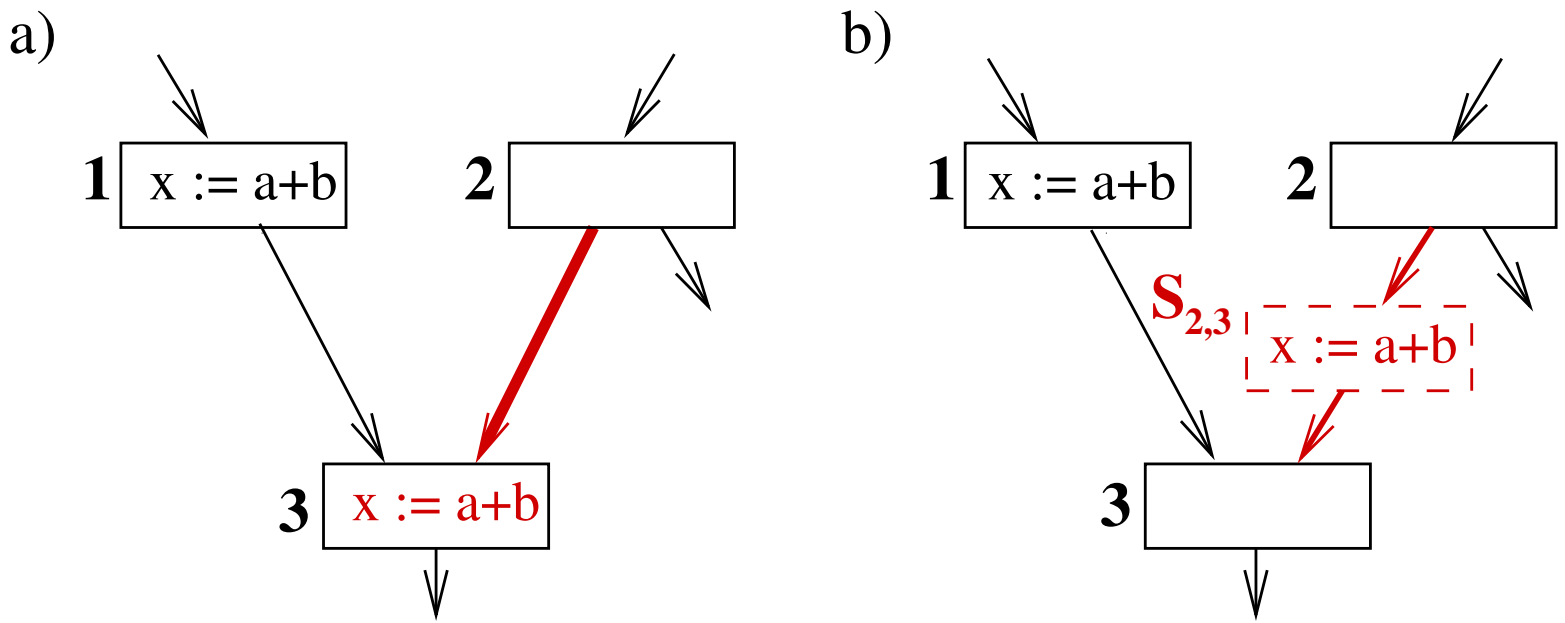
Anweisungshebung (Effekt 1. Ord.)

Elim. red. Anw. (Effekt 2. Ord.)



Anmerkung zu kritischen Kanten

...wie für EPTA, EPGA und EPRA müssen kritische Kanten gespalten werden, um bestmögliche Ergebnisse erzielen zu können:



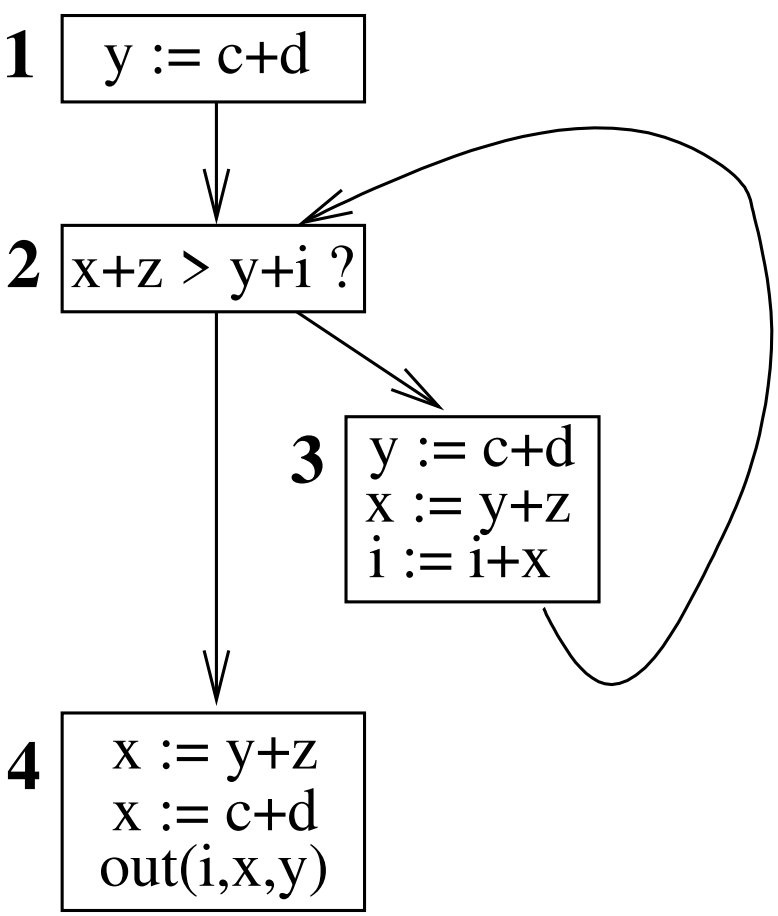
Kapitel 16.2.3

EPRAA: Beispiel

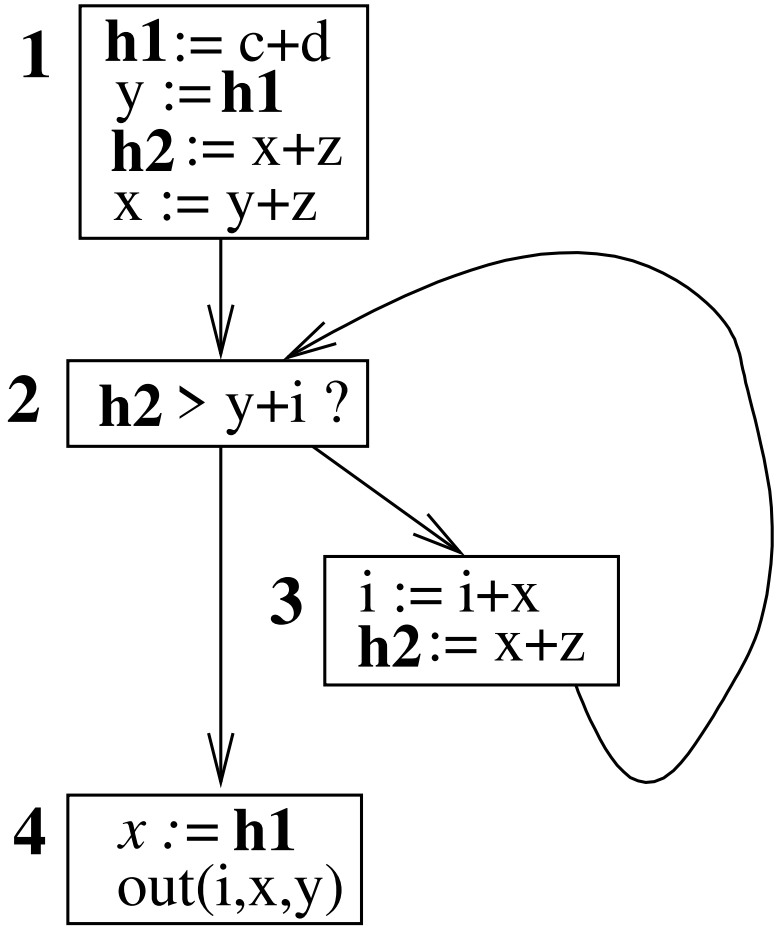
Die EPRAA-Transformation

...illustriert anhand eines größeren Beispiels.

Ausgangsprogramm

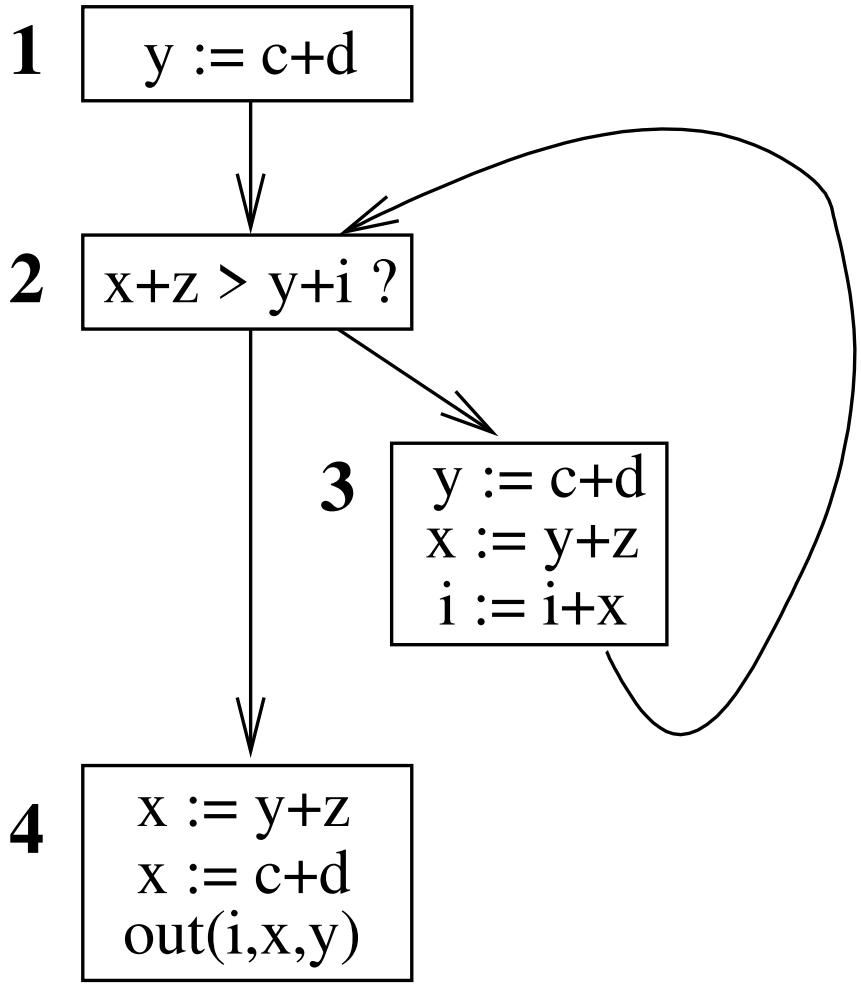


Optimiertes Programm



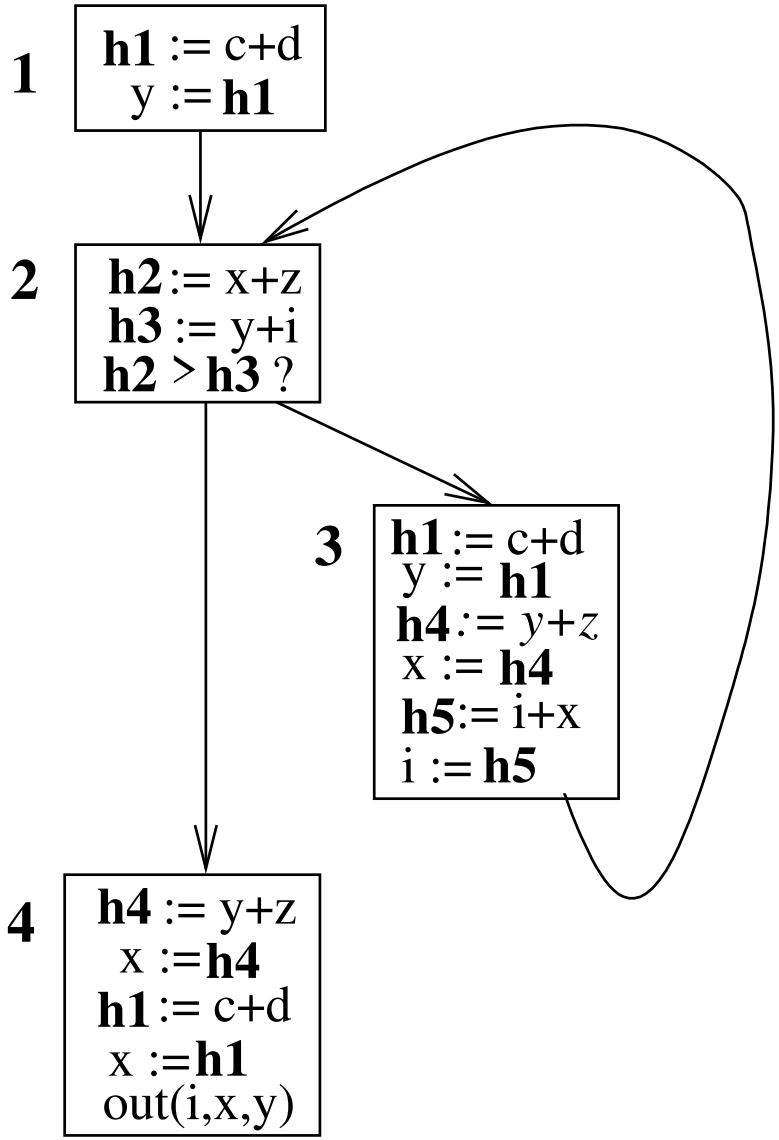
Die EPRAA-Transformation im Detail (1)

Ausgangsprogramm



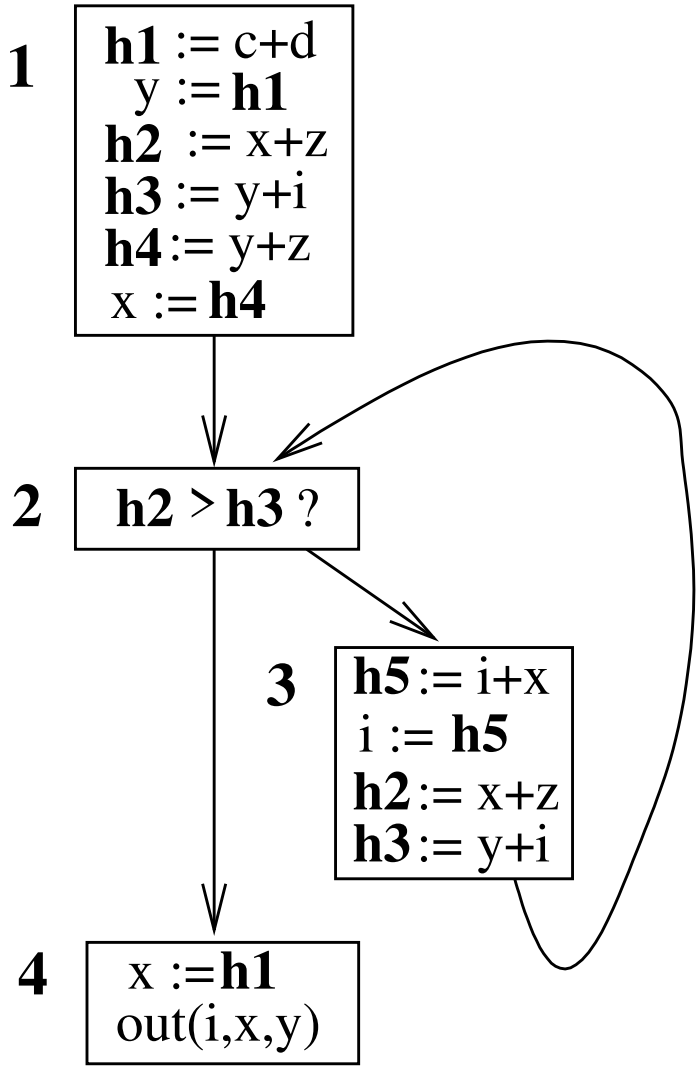
Die EPRAA-Transformation im Detail (2)

Programm nach Präprozess



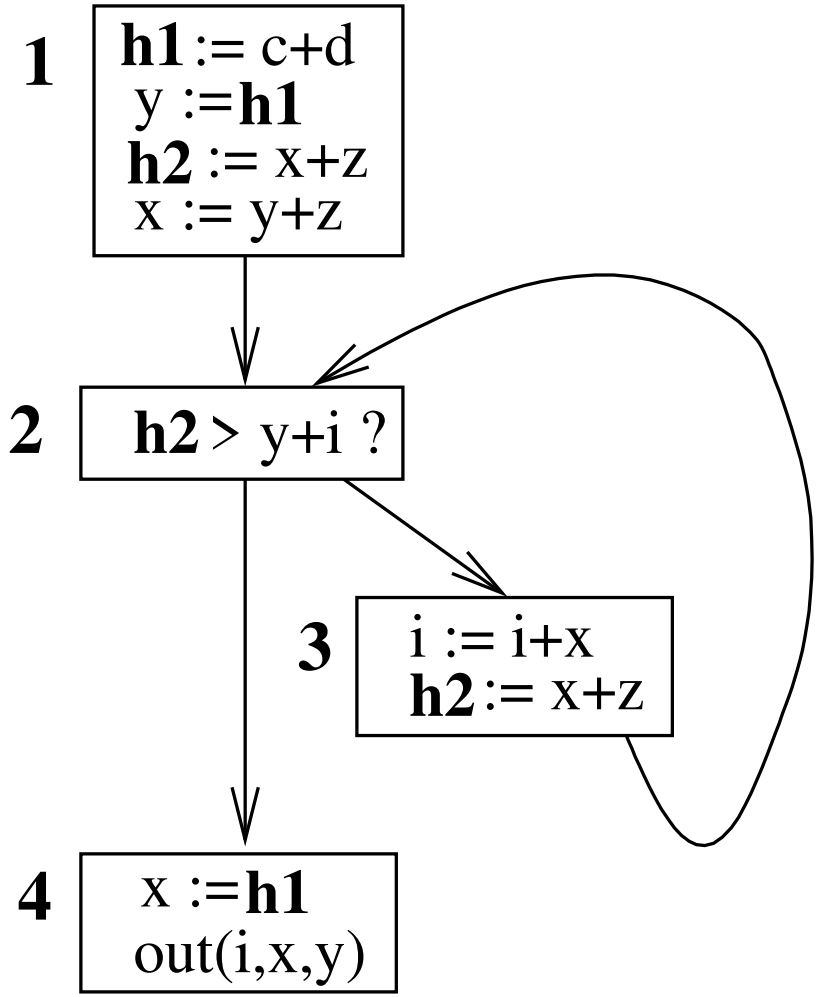
Die EPRAA-Transformation im Detail (3)

Programm nach Hauptprozess



Die EPRAA-Transformation im Detail (4)

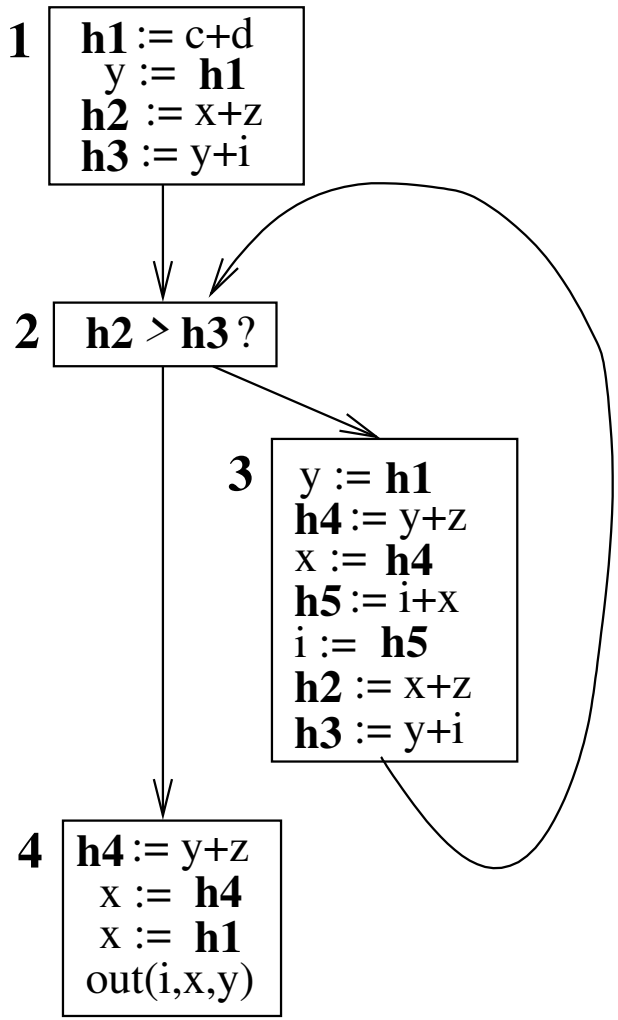
Programm nach Postprozess - das EPRAA-optimierte Prg.



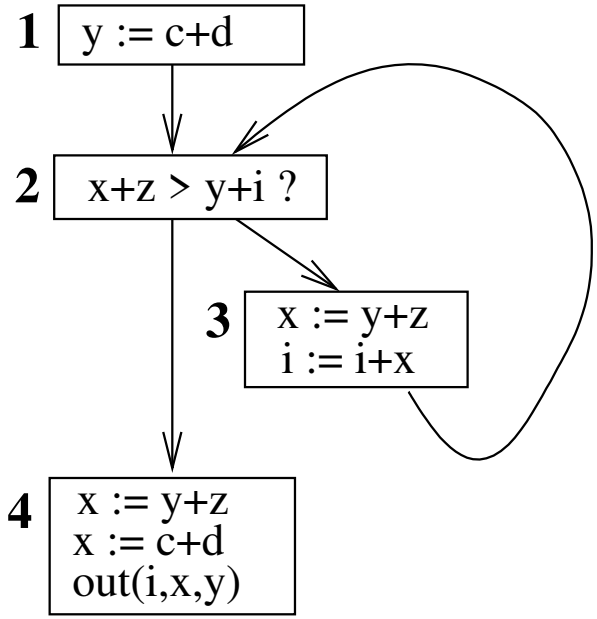
Zum Vergleich: Die schwächeren Effekte

...der EPRAd- und EPRA-Transformationen:

EPRAd-Effekt



EPRA-Effekt



Kapitel 16.2.4

EPRAA: Optimalität

EPRAA-Optimalitätsresultate

...anders als für EPTA, EPGA und EPRA mit je einer globalen Optimalitätsaussage zerfällt Optimalität für EPRAA in drei Aussagen über:

- Ausdrücke (globale Optimalität)
- Anweisungen (lokale Optimalität)
- Hilfsvariablen (lokale Optimalität)

Im folgenden bezeichnen für ein Programm G:

- $\mathcal{U}^G_{EPRAA} =_{df} \{G' \mid G \vdash_{(\tau_i)_{i \leq k}} G', \tau = (\tau_i)_{i \in \mathbb{N}} \in T^G_{EPRAA}, k \in \mathbb{N}\}$: Das von G durch die Präfixe der Transformationsfolgen $\tau \in T^G_{EPRAA}$ aufgespannte Universum.
- G_τ: Das durch Anwendung von $\tau \in \mathrm{T}^G_{EPRAA}$ auf G entstehende Programm.

EPRAA: Globale, lokale Optimalität

Sei $\tau_{max} \in \mathsf{T}^{G}_{EPRAA}$ eine maximale EPRAA-Transformation.

Theorem 16.2.4.2 (Globale Ausdrucksoptimalität)

$G_{\tau_{max}}$ is ausdrucksoptimal in $\mathcal{U}^{G}_{EPRAA}$, d.h., während seiner Ausführung werden höchstens so viele Ausdrücke ausgewertet wie in jedem anderen Programm in $\mathcal{U}^{G}_{EPRAA}$.

Theorem 16.2.4.3 (Lokale Anweisungsoptimalität)

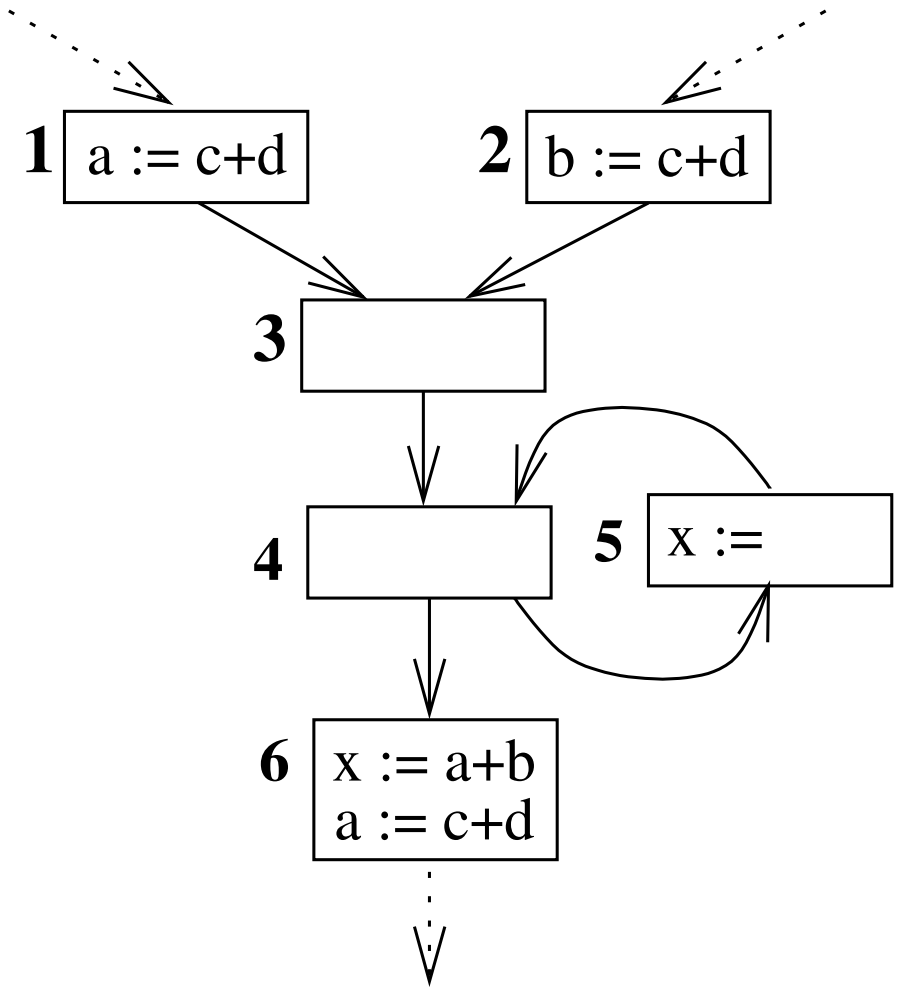
$G_{\tau_{max}}$ ist lokal anweisungsoptimal in $\mathcal{U}^{G}_{EPRAA}$, d.h. es ist nicht möglich, die Zahl der von $G_{\tau_{max}}$ zur Laufzeit ausgeführten Anweisungen durch weitere EPRAA-Elementartransformationen zu verringern.

Theorem 16.2.4.4 (Lokale Hilfsvariablenoptimalität)

$G_{\tau_{max}}$ ist lokal hilfsvariablenoptimal in $\mathcal{U}^{G}_{EPRAA}$, d.h. es ist nicht möglich, die Zahl der Zuweisungen an Hilfsvariablen oder die Länge der Lebenszeiten von Hilfsvariablen in $G_{\tau_{max}}$ durch weitere EPRAA-Elementartransformationen zu verringern.

Warum nur lokale Anw./Hilfsv.-Optimalität?

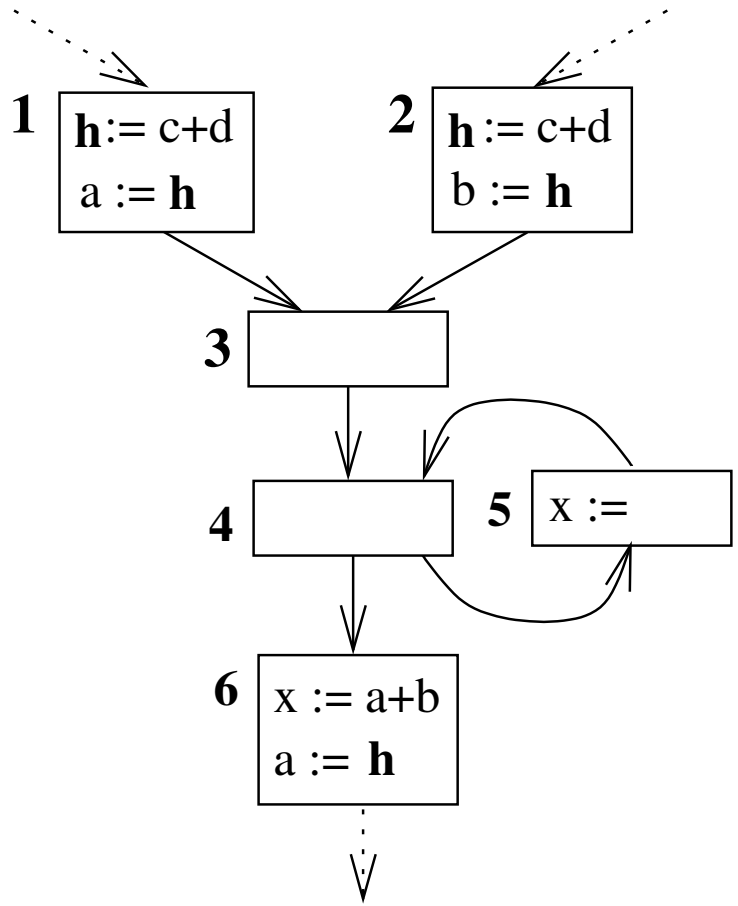
...betrachte folgendes Programm:



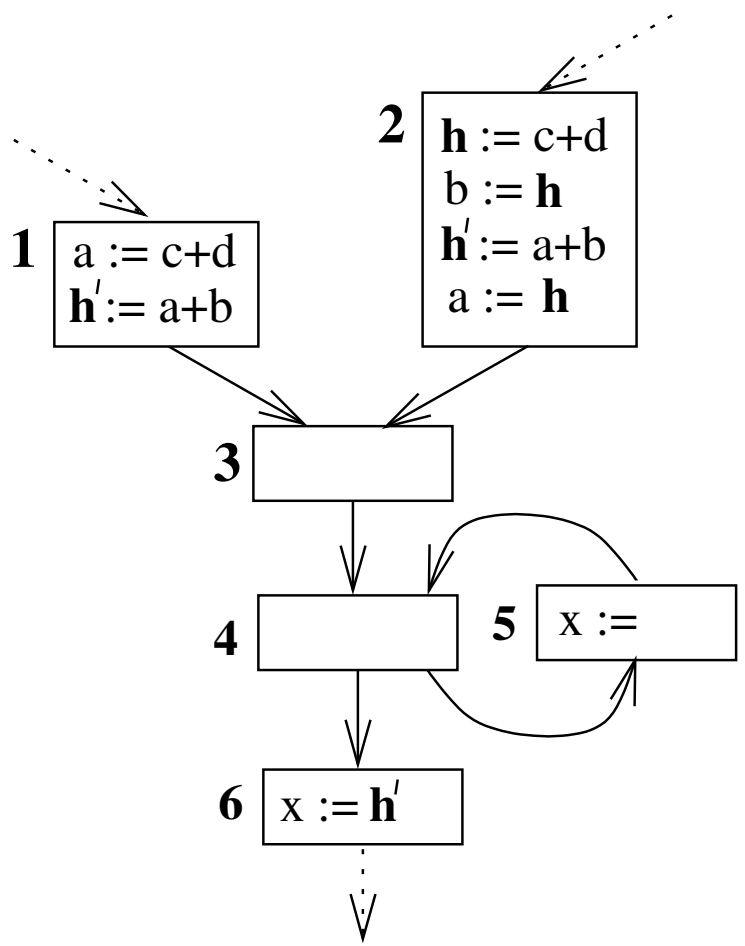
Darum nur lokale Anw./Hilfsv.-Optimalität!

...und folgende zwei unvergleichbare Transformationsresultate:

a)



b)



⇒ Lokale Anw./Hilfsv.-Optimalität ist das bestmögliche!

Kapitel 16.3

Ohne Beschränkung der Allgemeinheit

Kapitel 16.3.1

Motivation

Ohne Beschränkung der Allgemeinheit

...eine häufig getroffene Annahme, z.B.:

- Jeder Knoten eines Flussgraphen liegt auf einem Pfad vom Start- zum Endknoten (s. Kap. 15.2.3).
- Gibt es mehrere Endknoten, kann durch Einfügen eines künstlichen Endknotens und entsprechender Kanten dies erreicht werden (s. Kap. 15.2.3, indirekt behandelt).
- Anweisungen liegen im Format von Drei-Adress-Code vor (s. Kap. 16.3.2).
- Flussgraphen sind wahlfrei knoten- oder kantenbenannt, mit einzelnen Instruktionen oder mit Sequenzen von Instruktionen (sog. Basisblöcken) (s. Kap. 16.3.3 und Anhang B).
- ...

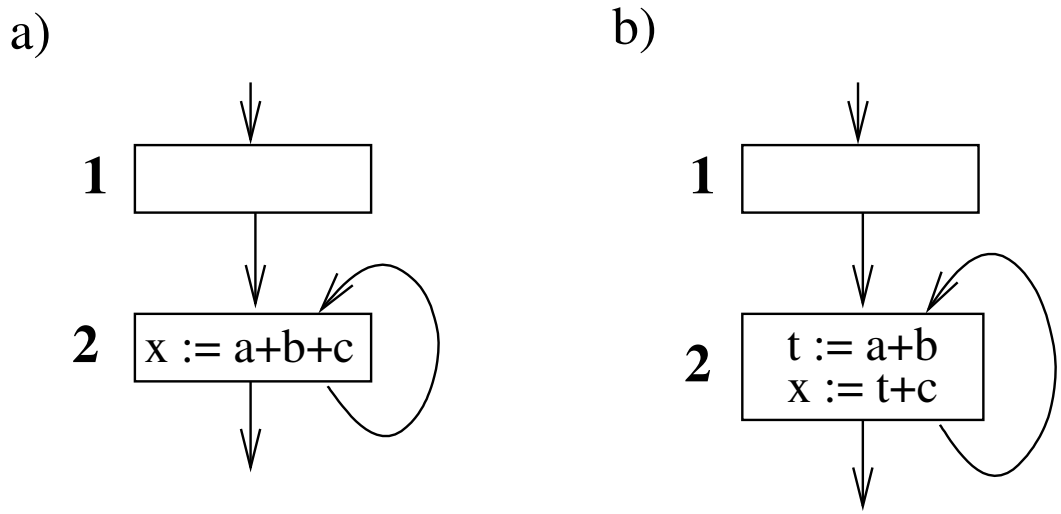
Nicht immer sind solche Annahmen uneingeschränkt unbeschränkend und frei von Nebenwirkungen...

Kapitel 16.3.2

Drei-Adress-Code vs. allgemeiner Code

Drei-Adress-Code

...entsteht durch Aufspalten von Anweisungen mit rechten Seiten mit mehr als einem Operator in Anweisungssequenzen, in der die rechten Seiten jeder Anweisung (höchstens) einen Operator besitzen (allgemein: Zahl der Operatoren der Ausgangsanweisung ist gleich Zahl der Anweisungen der Anweisungssequenz):

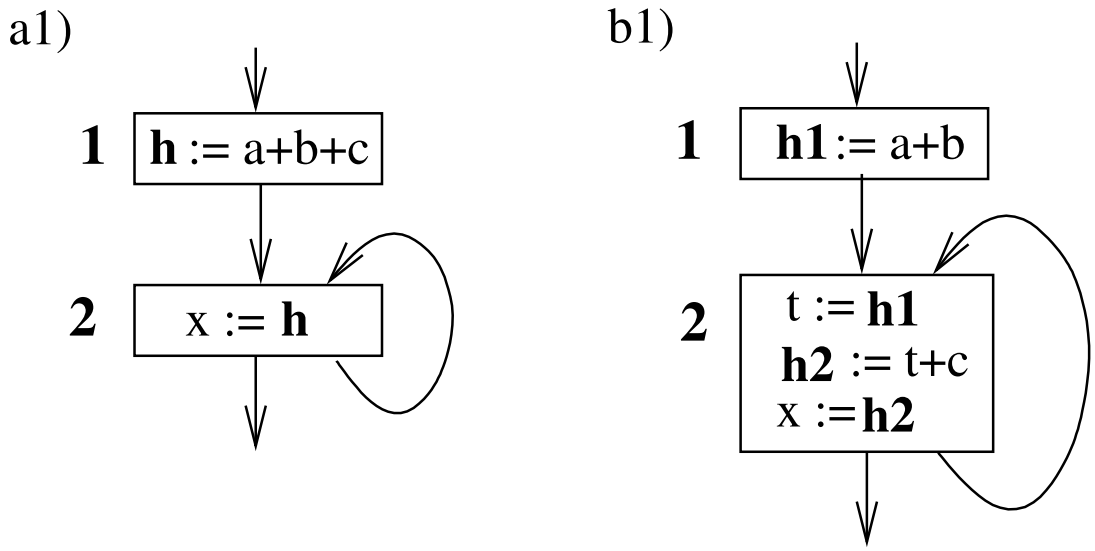


...offenbar ist eine solche Aufspaltung immer möglich;

- OBdA darf deshalb angenommen werden, dass Anweisungen in Drei-Adress-Form vorliegen.

Frei von Nebenwirkungen oBdA?

...vergleiche die Wirkung von EPRA auf die Programme aus Abb. a) und b) in Abb. a1) und b1):



...offenbar ist das transformierte Programm in Abb. a1) performanter als das in Abb. b1). Ist 'OBdA' hier gerechtfertigt? Nebenwirkungsfrei möglicherweise dank Variablensubsumption?

Variablensubsumption

...die mit dem Übergang zu Drei-Adress-Code verbundene Einführung zunächst vieler neuer Variablen ist nicht wesentlich, da nach Transformationsabschluss das Programm abschließend mittels

- Variablensubsumption (engl. variable subsumption)

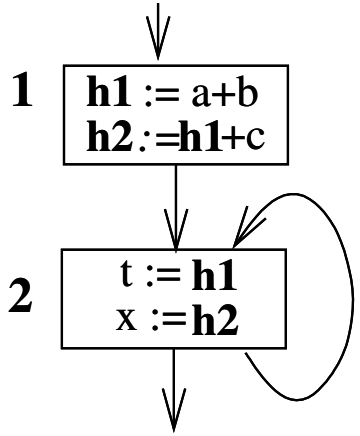
'aufgeräumt' werden kann, woduch Variablen zusammengefasst und in ihrer Zahl reduziert werden.

Frei von Nebenwirkungen oBdA?

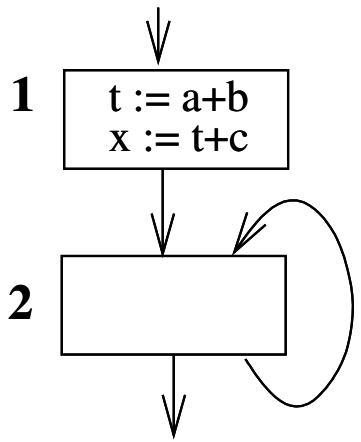
...vergleiche die Wirkungen von

- EPRA gefolgt von Variablensubsumption auf das Programm aus Abb. b) in Abb. b2).
- EPRAA ohne/mit Variablensubsumption auf das Programm aus Abb. b) in Abb. b3) und b4).

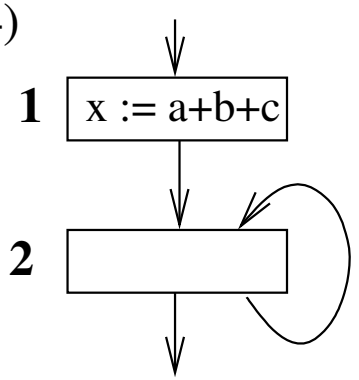
b2)



b3)



b4)



...offenbar ist das transformierte Programm in Abb. b3) bereits performanter als das in Abb. b2). 'OBdA' durch Variablensubsumption gerechtfertigt? Allenfalls bei richtiger Transformationswahl (im Beispiel EPRAA statt EPRA).

Kapitel 16.3.3

Basisblock- vs. Instruktionsgraphen, knoten- vs. kantenbenannte Graphen

Ohne Beschränkung der Allgemeinheit

...wird die Wahl der Programmdarstellung als

- knoten- oder kantenbenannte
- Instruktions- oder Basisblockflussgraphen

i.a. als ohne Nebenwirkungen und deshalb oBdA freie Wahlentscheidung gesehen.

Einschätzung: Fast immer zutreffend gibt es Ausnahmen, die eine genauere Betrachtung und sorgfältigere Wahl nahelegen oder gar verlangen...

Basisblock- vs. Instruktionsgraphen

...eine Abwägung, die historisch zugunsten von

- Basisblockgraphen

ausgefallen ist, da Basisblockgraphen weniger Knoten als Instruktionsgraphen enthalten und deshalb

- zu schnellerer Konvergenz von Fixpunktanalysen führen.

Der konzeptuelle und implementierungstechnische Mehraufwand durch die erforderliche Dreistufigkeit der Analyse aus

- Präprozess (Berechnung der Basisblocksemantik)
- Hauptprozess (Fixpunktanalyse auf Basisblockgraph)
- Postprozess (Basisblockanalyse)

würde dadurch aufgewogen.

Einschätzung: Während der Mehraufwand real ist, realisiert sich der Performanzvorteil in der Praxis nicht (oder heute nicht mehr); siehe Anhang B für eine genauere Betrachtung.

Knoten- vs. kantenbenannte Graphen

...eine Abwägung, die historisch zugunsten von

- knotenbenannten Graphen

ausgefallen ist, ohne aus der Literatur ersichtliche tiefergehende Überlegungen.

Einschätzung:

- Pragmatische Unterschiede zwischen knoten- und kantenbenannten Graphen beschränken sich für DFA-Probleme i.w. auf die konzeptuelle Ebene mit leichten Vorteilen für kantenbenannte Graphen, die zu knapperen Analysespezifikationen führen; siehe Anhang B für Details.
- Interessant ist, dass im Bereich von Modellprüfung mit Kripke-Strukturen (knotenbenannte Graphen) und Transitionssystemen (kantenbenannte Graphen) beide Varianten gezielt gewählt werden; siehe Kapitel 19 für Details.

Von Instruktions- zu Basisblockgraphen

...die Analysen der Elementartransformationen für die Berechnung

- toter und redundanter Anweisungen
- der Endpunkte von Anweisungssenkungen und -hebungen

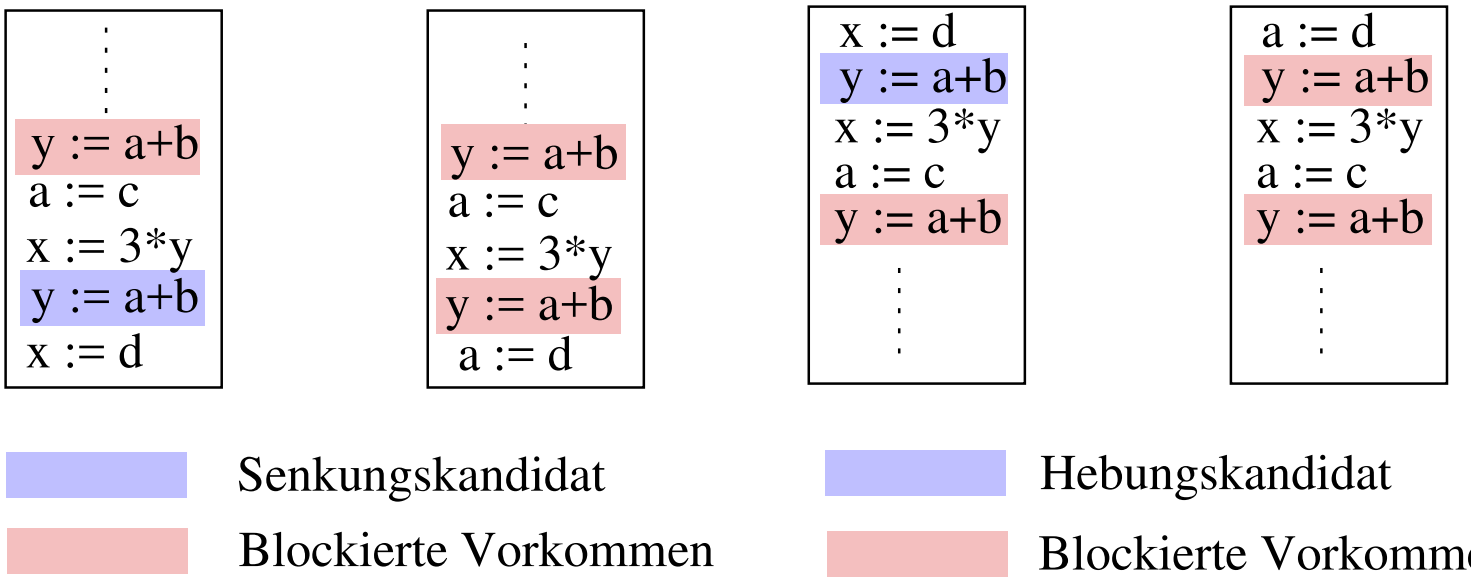
lassen sich in natürlicher Weise von Instruktions- auf Basisblockgraphen ausdehnen; für die Analyse zur Berechnung

- geisterhafter Anweisungen

als sog. nicht-separables Problem gilt das nicht, eine Ausdehnung auf Basisblockgraphen ist nicht möglich (s. Kapitel 16.3 und Anhang B für Details).

Senkungs-/Hebungskandidaten in Basisblöcken

...sind für jedes Anweisungsmuster eindeutig bestimmt: Höchstens das letzte bzw. erste Vorkommen eines Musters ist Senkungs-/Hebungskandidat, wenn es nicht lokal blockiert ist.



Beispiel: Blau markiert sind die eindeutig bestimmten Senkungs- bzw. Hebungskandidaten des Anweisungsmusters $\alpha \equiv$ y := a+b; rot markiert sind lokal blockierte α-Vorkommen, die keine Senkungs- oder Hebungskandidaten sind.

Anpassung der Senkungs-/Hebungsanalyse

...auf Basisblockgraphen.

Es reicht, die Interpretation (oder Bedeutung) der lokalen Prädikate wie folgt zu ändern:

Lokale Prädikate (assoziiert mit Basisblockknoten):

- Sinkable_n^α/$\text{Hoistable}_n^\alpha$: Es gibt einen α-Senkungs-/Hebungskandidaten im Basisblockknoten n (d.h. es gibt ein letztes/erstes α-Vorkommen in n), das von keiner nachfolgenden/vorausgehenden Anweisung in n blockiert wird.
- Blocked_n^α: Basisblockknoten n enthält eine Anweisung, die das Schieben eines weiteren α-Vorkommens an den Basisblockausgang/-eingang blockiert.

...mit diesen Änderungen können das AS/HS-Gleichungssystem für Instruktionsgraphen aus Kapitel 15.3.8 bzw. 15.4.7 unverändert für Basisblockgraphen übernommen werden.

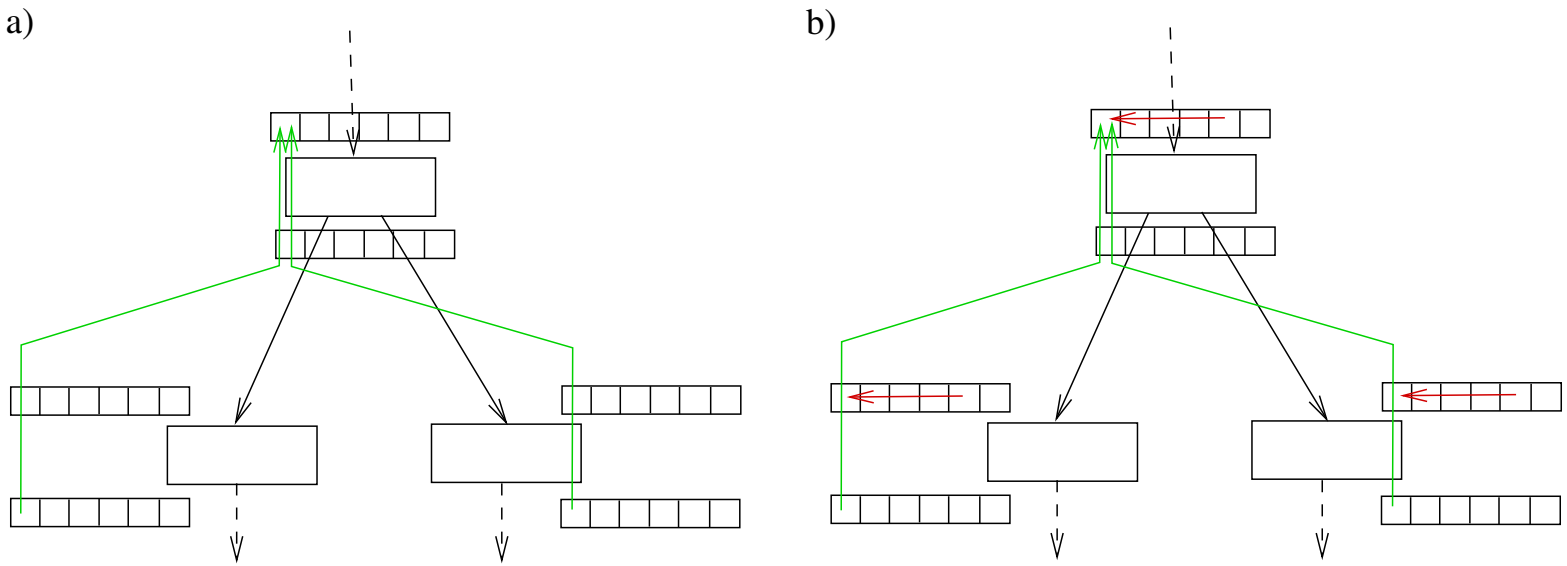
Separable vs. nicht-separable DFA-Probleme

...nicht für alle DFA-Probleme ist die Ausdehnung auf Basisblockgraphen so natürlich und einfach möglich; für sog. nicht-separable DFA-Probleme sogar gar nicht. Der Grund liegt im zusätzlich 'quer' verlaufenden Informationsfluss bei nicht-separablen Problemen wie der Geistervariablenanalyse; s.a. Anh. B.

Informationsfluss in Bitvektoren

Tote-Variablen-Analyse (separabel)

Geistervariablenanalyse (nicht-separabel)



Kapitel 16.4

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (1)

- Dhananjay M. Dhamdhere. *Register Assignment using Code Placement Techniques*. Journal of Computer Languages 13(2):75-93, 1988.
- Dhananjay M. Dhamdhere. *A usually linear Algorithm for Register Assignment using Edge Placement of Load and Store Instructions*. Journal of Computer Languages 15(2):83-94, 1990.
- Dhananjay M. Dhamdhere. *Practical Adaptation of the Global Optimization Algorithm of Morel and Renvoise*. ACM Transactions on Programming Languages and Systems 13(2):291-294, 1991. Technical Correspondence.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (2)


Jens Knoop, Eduard Mehofer. *Distribution Assignment Placement: Effective Optimization of Redistribution Costs*. IEEE Transactions on Parallel and Distributed Systems 13(6):628-647, 2002.


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Lazy Code Motion*. In Proceedings of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation (PLDI'92), ACM SIGPLAN Notices 27(7):224-234, 1992.


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Optimal Code Motion: Theory and Practice*. ACM Transactions on Programming Languages and Systems 16(4):1117-1155, 1994.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (3)

Jens Knoop, Oliver Rüthing, Bernhard Steffen. *The Power of Assignment Motion*. In Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI'95), ACM SIGPLAN Notices 30(6):233-245, 1995.

Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Code Motion and Code Placement: Just Synonyms?* In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 154-169, 1998.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (4)

Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Retrospective: Lazy Code Motion*. In '20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation (1979 - 1999): A Selection,' ACM SIGPLAN Notices 39(4):460-461&462-472, 2004.

Munehiro Takimoto, Kenichi Harada. *Effective Partial Redundancy Elimination based on Extended Value Graph*. Information Processing Society of Japan 38(11):2237-2250, 1990.

Kapitel 17

Konstantenanalyse auf nichtklassischen Programm- und Datenstrukturen

Konstantenanalyse

...auf dem Wertegraphen (engl. value graph) eines Programms bzw. Flussgraphen.

- Hintergrund, Motivation
- Die VG-Konstantenanalyse
  - Basisalgorithmus
  - Voller Algorithmus
- Die PVG-Konstantenanalyse auf prädikatiertem Code

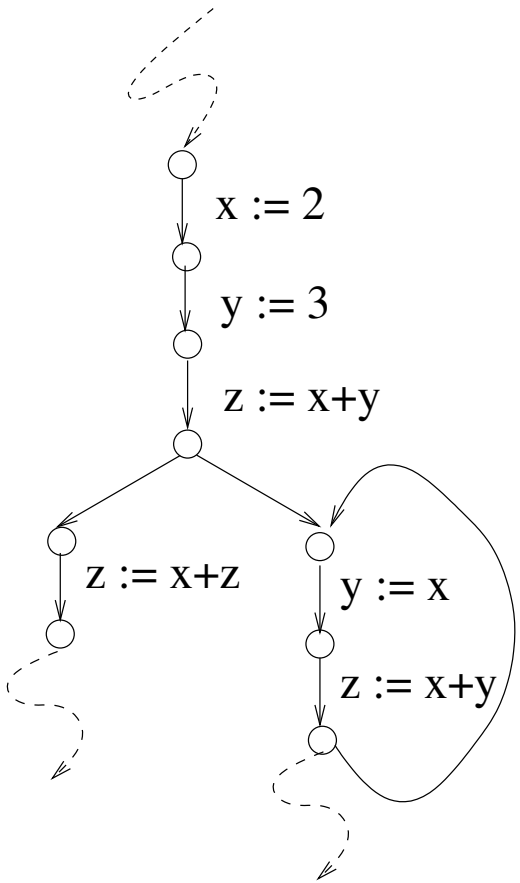
Kapitel 17.1

Motivation

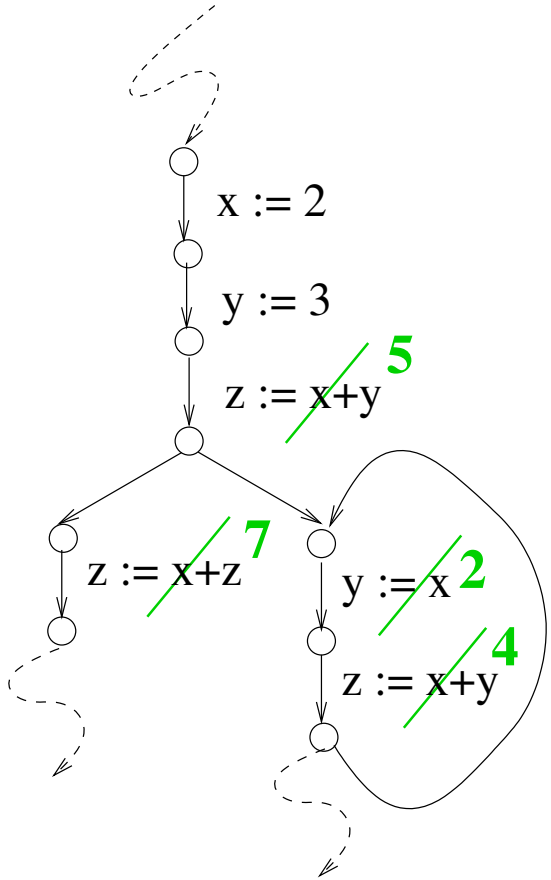
Konstantenanalyse

...anhand eines Beispiels für einfache Konstanten:

a)



b)



Ausgangsprogramm

Nach "einfacher Konstanten"–Analyse

Ausgangspunkt und Treiber

...von Algorithmen zur Konstantenanalyse:

- Gary A. Kildalls Algorithmus zur Berechnung einfacher Konstanten (engl. simple constants (SC) (POPL'73).

  Beachte: Der Algorithmus zur Berechnung einfacher Konstanten aus Kapitel 8.13.2 stimmt im Ergebnis, nicht jedoch in den verwendeten Datenstrukturen mit Kildalls Algorithmus überein.

Erweiterungen von Kildalls SC-Algorithmus (1)

...zielen auf Verstärkung oder Verbesserung der:

- Ausdruckskraft
  - 'SC+': Kam, Ullman (Acta Informatica, 1977)
  - Konditionale Konstanten (engl. conditional constants): Wegman, Zadeck (POPL'85)
  - Endliche Konstanten (engl. finite constants): Steffen, Knoop (MFCS'89)
  - Polynomiale Konstanten (engl. polynomial constants): Müller-Olm, Seidl (SAS 2002)
  - ...

  zulasten der Performanz.
- Performanz
  - SSA-Form: Wegman, Zadeck (POPL'85)
  - ...

  ohne Erhöhung der Ausdruckskraft.

Erweiterungen von Kildalls SC-Algorithmus (2)

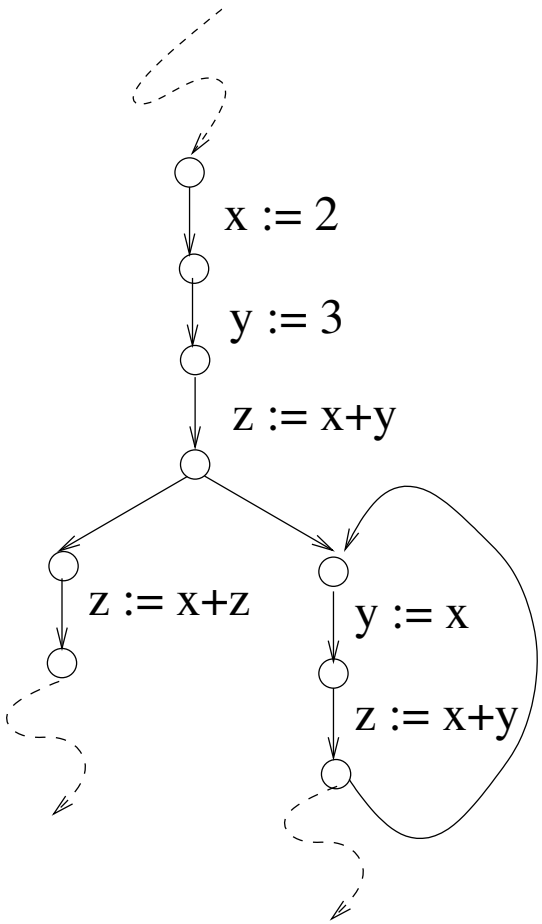
- Anwendungsreichweite
  - Interprozedural
    - Callahan, Cooper, Kennedy, Torczon (SCC'86)
    - Grove, Torczon (PLDI'93)
    - Metzer, Stroud (LOPLAS, 1993)
    - Sagiv, Reps, Horwitz (TAPSOFT'95)
    - Duesterwald, Gupta, Soffa (TOPLAS, 1997)
    - ...
  - Explizit parallel
    - Lee, Midkiff, Padua (J. of Parallel Prog., 1998)
    - Knoop (Euro-Par'98)
    - ...

  zulasten der Ausdruckskraft, z.B. Kopier- und lineare Konstanten (engl. copy constants, linear constants) anstelle einfacher Konstanten (engl. simple constants).

...siehe auch LVA 185.A04, Kapitel 5.

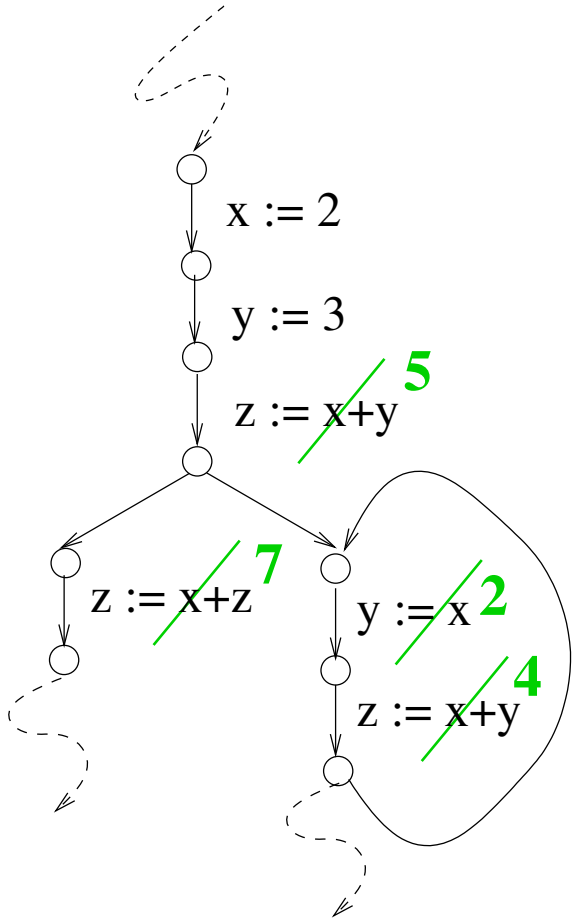
Warum streben nach größerer Ausdruckskraft?

a)



Ausgangsprogramm

b)



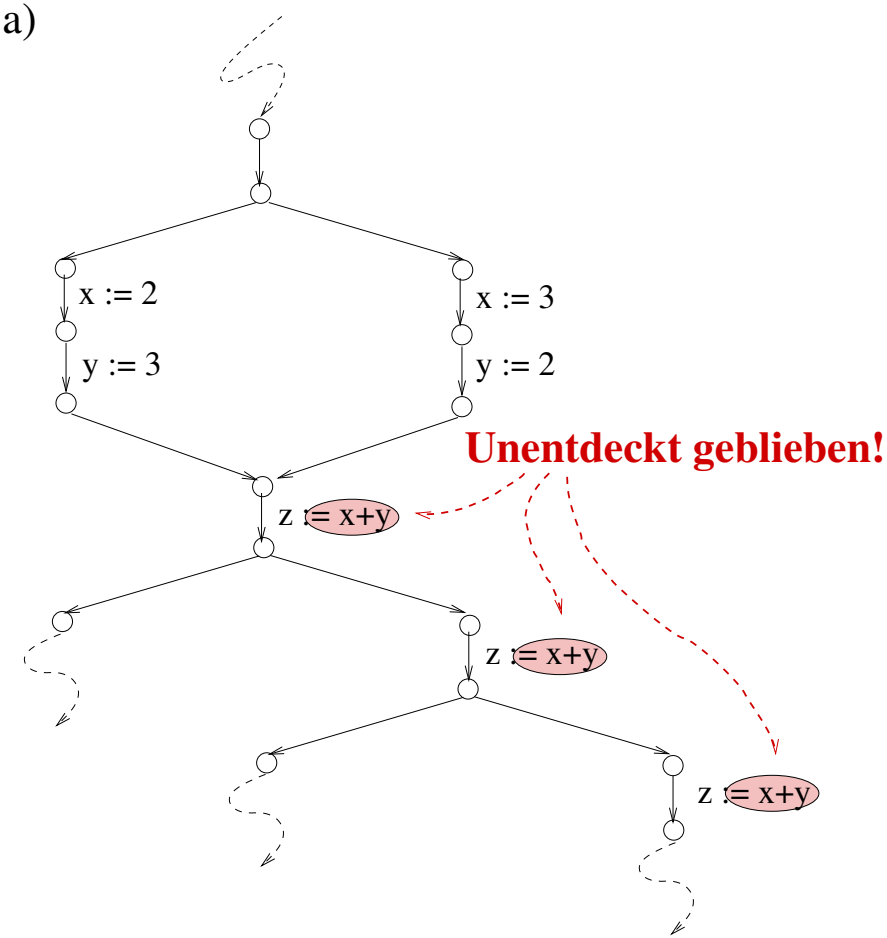
Nach "einfacher Konstanten"–Analyse

...das Ergebnis der SC-Analyse scheint hier überzeugend.

Der Schein trügt

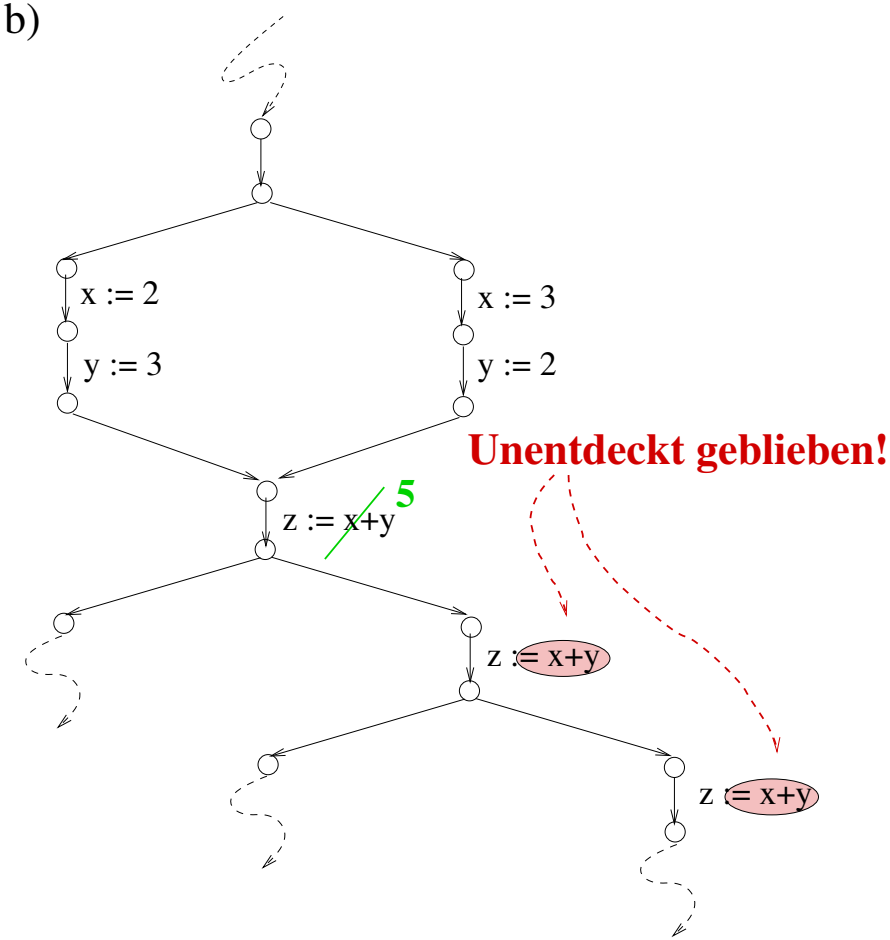
...das Konzept einfacher Konstanten ist tatsächlich schwach:

a)



Nach "einfacher Konstanten"–Analyse
(Beachte: Ueberhaupt kein Effekt!)

b)



Nach "einfacher Konstanten"–Analyse
verstaerkt um die "1–Block–Vorschau"
von Kam und Ullman

Entscheidbarkeitsfragen für Konstantenanalyse

Einerseits: Konstantenanalyse ist

- unentscheidbar: Reif, Lewis (POPL 1977, vgl. Kapitel 8.13.2)

für allgemeine Programme.

Andererseits: Konstantenanalyse ist definitiv

- entscheidbar

für schleifenfreie, azyklische Programme (engl. directed acyclic graphs (DAGs)):

- Die Zahl der Programmpfade ist endlich!

Das Konzept endlicher Konstanten

...ist optimal (oder vollständig) für schleifenfreie Programme, d.h. jede Konstante in einem schleifenfreien Programm ist eine endliche Konstante (engl. finite constant)!

Der Schlüssel endlicher Konstantenanalyse ist

- in einem Präprozess für jeden Programmpunkt n eine endliche Menge interessanter Terme $\mathbf{T}_n$ zu berechnen.
- im Hauptprozess DFA-Zustandsmenge und lokale Semantikfunktionen einfacher Konstantenanalyse:

$$\Sigma = \{\sigma \mid \sigma : \mathbf{V} \to \mathbb{Z}_\perp^\top\},\ [\![e]\!]_{sc} : \Sigma \to \Sigma$$

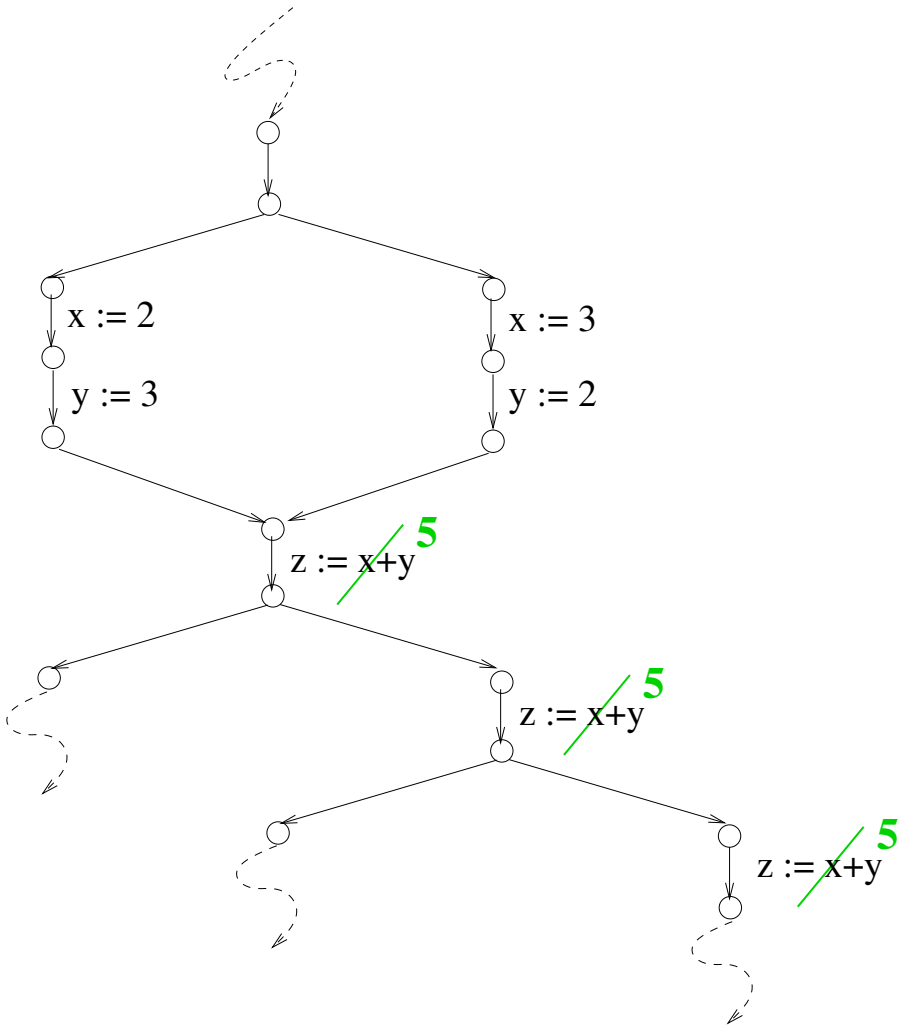
durch jene endlicher Konstantenanalyse:

$$\Sigma_N = \{\sigma_n \mid \sigma_n : \mathbf{T}_n \to \mathbb{Z}_\perp^\top,\ n \in N\},\ [\![e]\!]_{fc} : \Sigma_{\mathbf{T}_{src(e)}} \to \Sigma_{\mathbf{T}dst(e)}$$

zu ersetzen.

Endliche Konstanten

...überkommen die konzeptuelle Schwäche einfacher Konstanten:



Die Wirkung des neuen Verfahrens

Hauptergebnisse für endliche Konstanten

Positiv: Endliche Konstanten sind

- optimal (oder vollständig) für schleifenfreie Programme.
- echte Obermenge einfacher Konstanten für allgemeine Programme (d.h. mit beliebigem Kontrollfluss).

Aber: Der Algorithmus für endliche Konstantenanalyse ist

- exponentiell (bereits für schleifenfreie Programme).

Allerdings, bevor man den Stab vorschnell bricht:

Theorem 17.1.1

Konstantenanalyse für schleifenfreie Programme ist co-NP-vollständig.

Knoop, Rüthing (CC'00)
Müller-Olm, Rüthing (ESOP'01)

Die Herausforderung von Konstantenanalyse

...eine angemessene und gute Balance zu wahren zwischen Genauigkeit und Effizienz!

Einfache Konstanten

- effizient, aber ungenau.

Endliche Konstanten

- genau, aber ineffizient.

Gesucht: Eine entscheidbare Konstantenklasse

- deutlich umfassender als einfache Konstanten.
- wesentlich effizienter als endliche Konstanten.

Konstantenanalyse auf dem Wertegraphen

...oder VG-Konstantenanalyse bietet eine ausgewogene Balance zwischen

- Ausdruckskraft und Effizienz.

Die VG-Konstantenanalyse stützt sich auf den

- Wertegraphen (engl. value graph)

von Alpern, Wegman, and Zadeck (POPL'88).

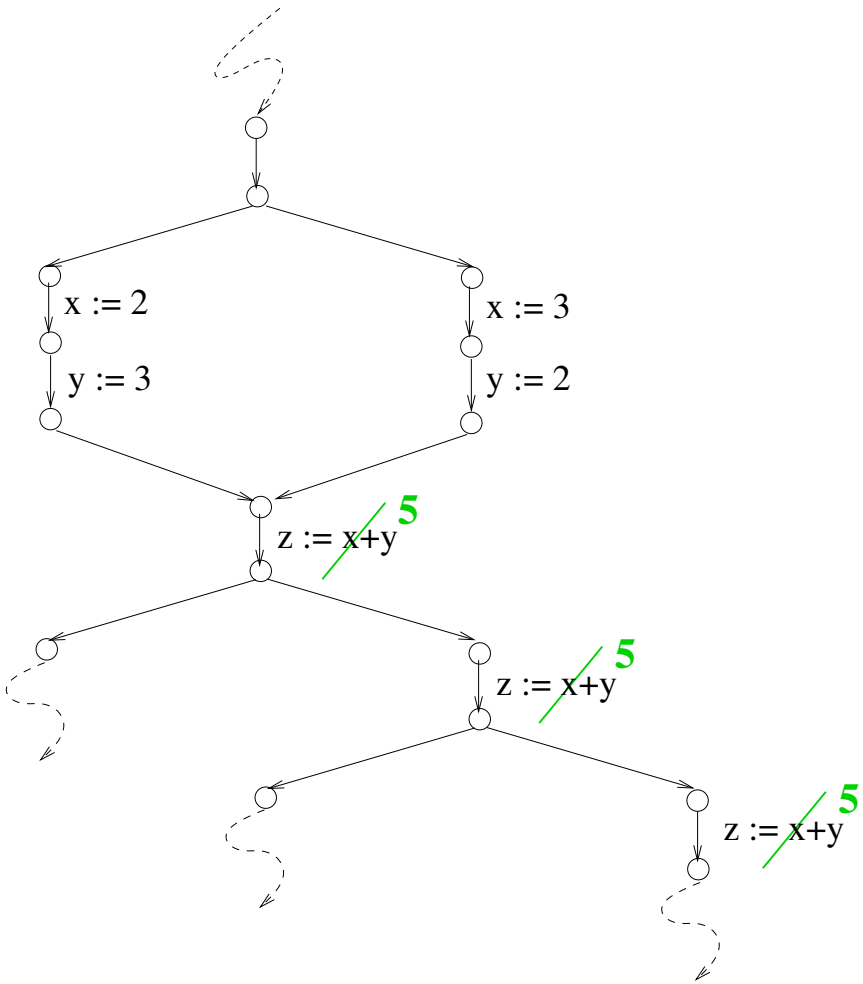
Der Wertegraph stützt sich seinerseits auf die

- Einmal-Zuweisungs-Repräsentation (engl. static single assignment (SSA) form) von Programmen

von Cytron et al. (POPL'89)).

Zurück zum laufenden Beispiel

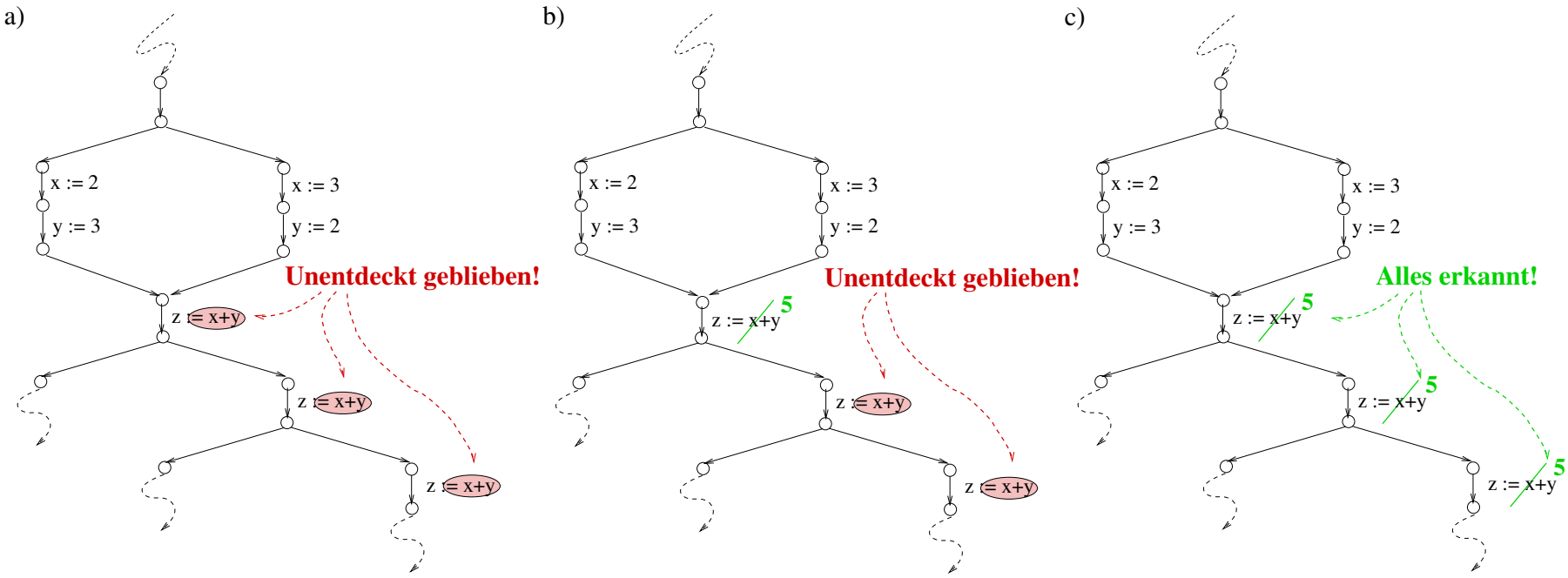
...wie endliche Konstantenanalyse erkennt auch VG-Konstantenanalyse alle Terme als Konstanten:



Die Wirkung des neuen Verfahrens

Insbesondere

...geht VG-Konstantenanalyse weit über eine frühere effiziente *ad hoc*-Verbesserung von Kam und Ullman, 1977, von Kildalls SC-Algorithmus hinaus und eine hier SC+ genannte Klasse von '1-Vorschau'-Konstanten erkennt:



Nach "einfacher–Konstanten"–Analyse
(Beachte: Ueberhaupt kein Effekt!)

Nach "einfacher–Konstanten"–Analyse
verstaerkt um die "1–Block–Vorschau"
von Kam und Ullman

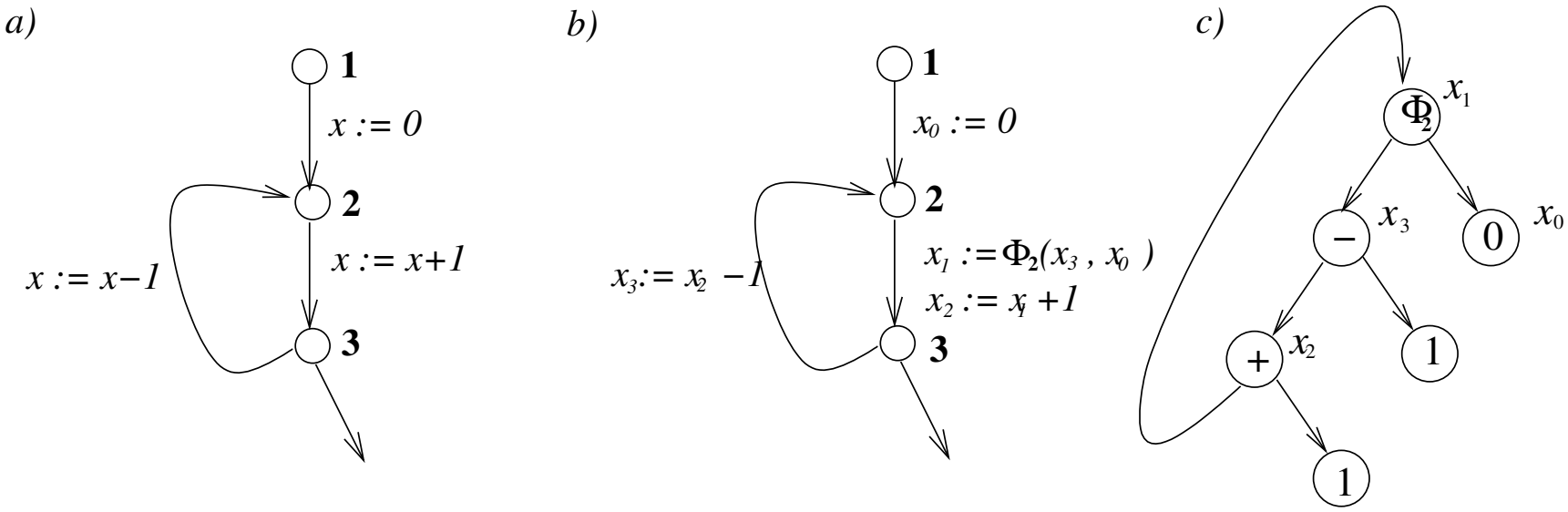
Die Wirkung des neuen Verfahrens

Kapitel 17.2

Konstantenanalyse auf dem Wertegraph

Der Wertegraph

...eines Programms nach Alpern, Wegman, Zadec (POPL'88):



Flussgraph SSA-Graph Wertegraph

Konstantenanalyse auf dem Wertegraphen

...in zwei unterschiedlich mächtigen Varianten:

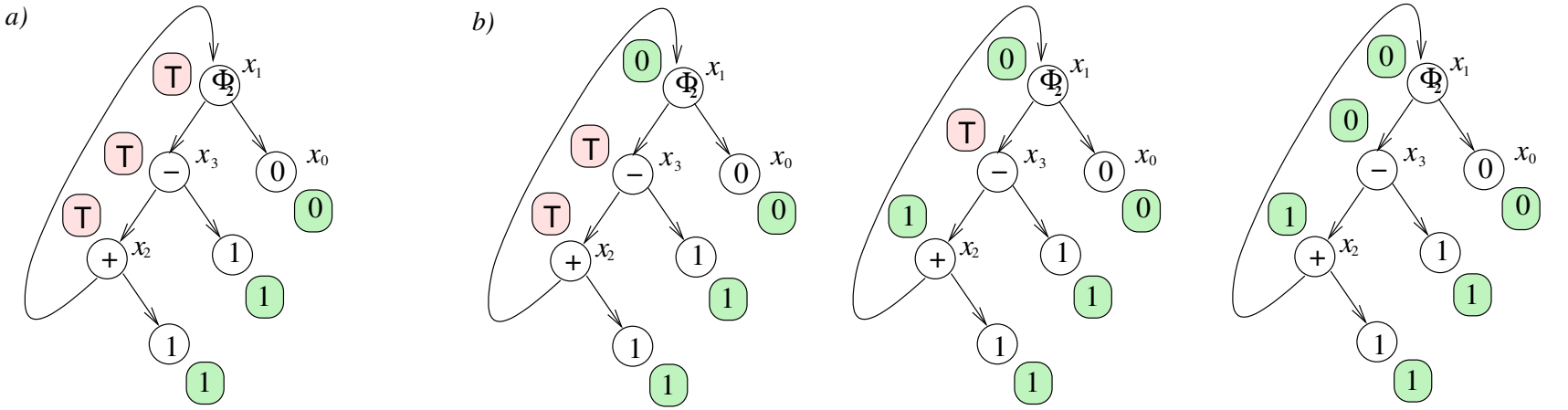
- VG-Basiskonstantenanalyse
  ...erkennt die Klasse einfache Konstanten.
- Volle VG-Konstantenanalyse
  ...erkennt eine Klasse von Konstanten, die weit über die Klassen der einfachen und SC+-Konstanten hinausgeht.

Kapitel 17.2.1

VG-Basiskonstantenanalyse

Konstantenanalyse auf dem Wertegraphen

...illustriert anhand eines Beispiels:



Nach der Initialisierung | Nach der 1. Iteration | Nach der 2. Iteration | Nach der 3. Iteration: Stabil!

Analyseresultat: x_2 und x_3 sind (einfache) Konstanten!

Hauptresultat

...für die VG-Basiskonstantenanalyse.

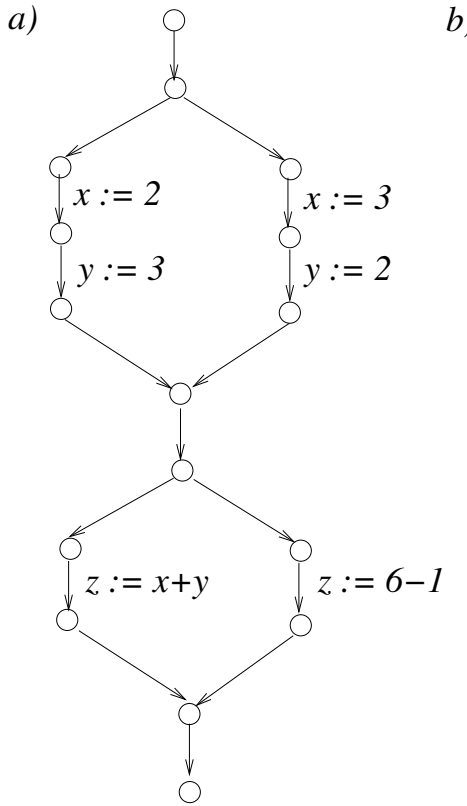
Lemma 17.2.1.1

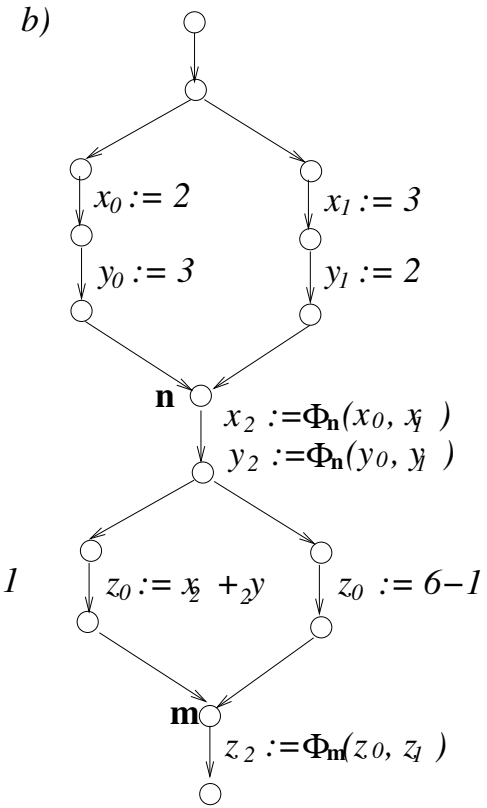
Die VG-Basiskonstantenanalyse ist korrekt und erkennt die Klasse der einfachen Konstanten.

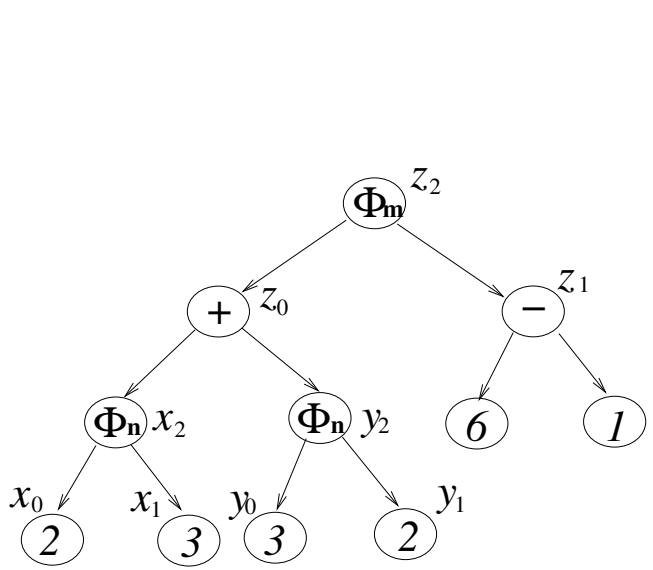
Kapitel 17.2.2

Volle VG-Konstantenanalyse

Illustrierendes Beispiel



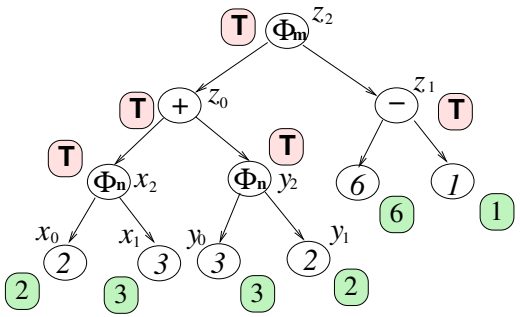




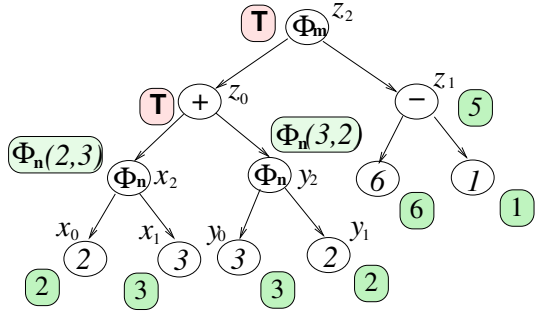
Flussgraph SSA-Graph Wertegraph

Volle VG-Konstantenanalyse

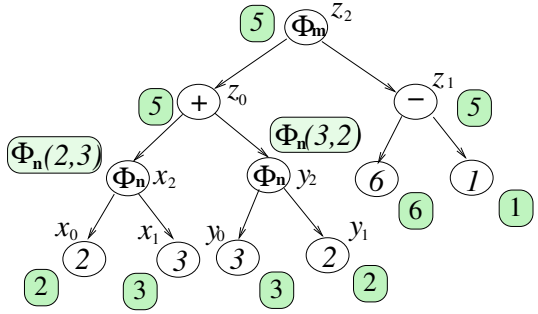
...illustriert anhand des laufenden Beispiels:



Nach der Initialisierung



Nach der 1. Iteration



Nach der 2. Iteration: Stabil!

Der technische Clou: Die

- Einführung von Φ-Konstanten
- Anpassung der Evaluationsfunktion auf Wertegraphen!

Hauptresultate

Lemma 17.2.2.1

Die volle VG-Konstantenanalyse ist korrekt und erkennt

- eine Obermenge der Klasse einfacher Konstanten.
- in schleifenfreien Programmen jede injektive Konstante, d.h. jeden Term, dessen relevante Operanden ausschließlich durch injektive Operatoren verknüpft sind.

Insgesamt erreicht die volle VG-Konstantenanalyse damit eine

- ausgewogene Balance zwischen Ausdruckskraft und Effizienz.

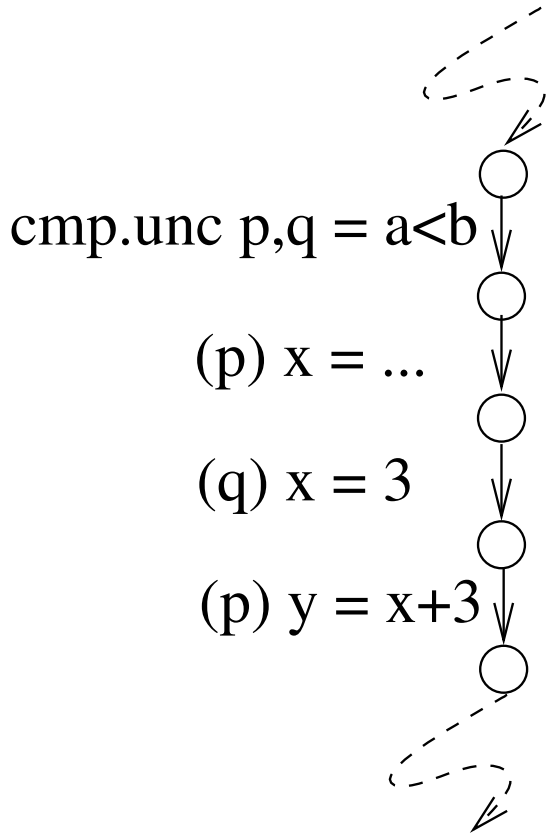
Essentiell dafür ist die Abstützung auf den Wertegraph (und damit indirekt auf den SSA-Graphen eines Programms).

Kapitel 17.3

Konstantenanalyse auf dem prädikatierten Wertegraph

Prädikatierter Code

...als Resultat sog. if-Konversion:

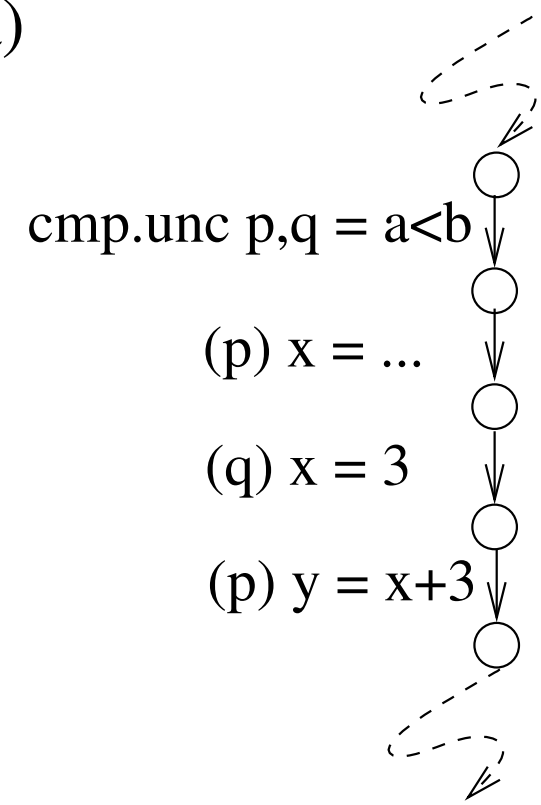


...mit dem Ziel besserer Parallelisierbarkeit durch erhöhte Parallelität auf Instruktionsebene (engl. instruction-level parallelism).

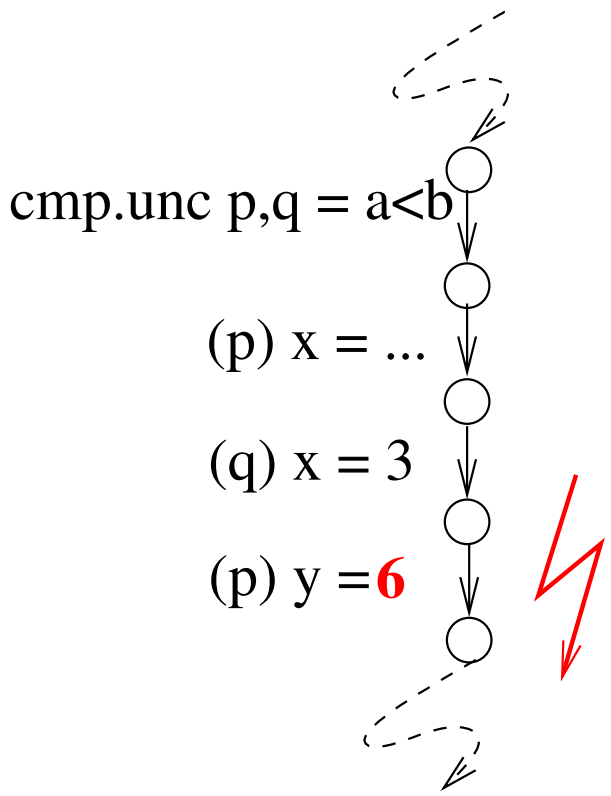
Naive Übertragung

...von Konstantenanalysetechniken von unprädikatiertem auf prädikatierten Code schlägt fehl:

a)



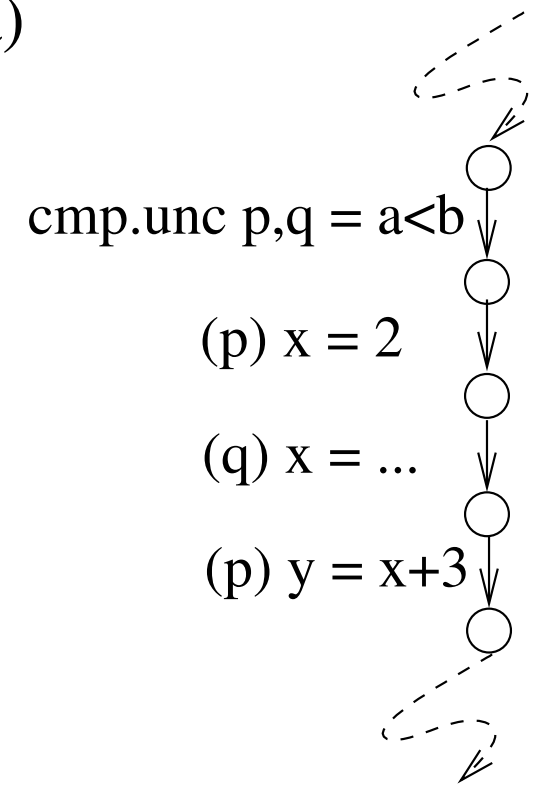
b)



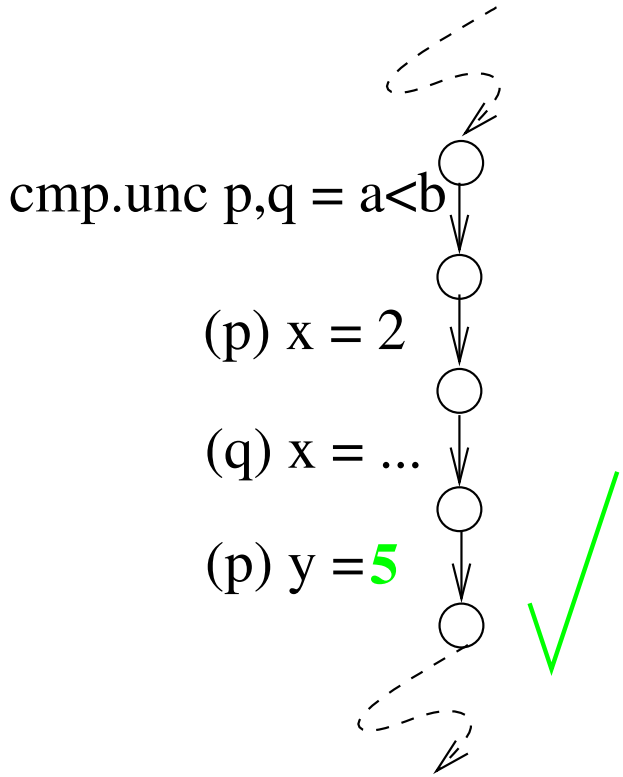
Naive korrekte Übertragung

...von Konstantenanalysetechniken von unprädikatiertem auf prädikatierten Code ist andererseits zu konservativ und erkennt zu viele Konstanten nicht wie etwa im folgenden Beispiel:

a)



b)



PVG-Konstantenanalyse

...für einen angemesseneren Umgang mit prädikatiertem Code zur Konstantenanalyse mit ‘zwei+’ Varianten:

- PVG-Basiskonstantenanalyse
- Volle PVG-Konstantenanalyse

zuzüglich

- performanz-verbesserter Variationen.

Alle Varianten und Variationen sind zweistufig und bestehen aus einer

- lokalen
- globalen

Analysestufe.

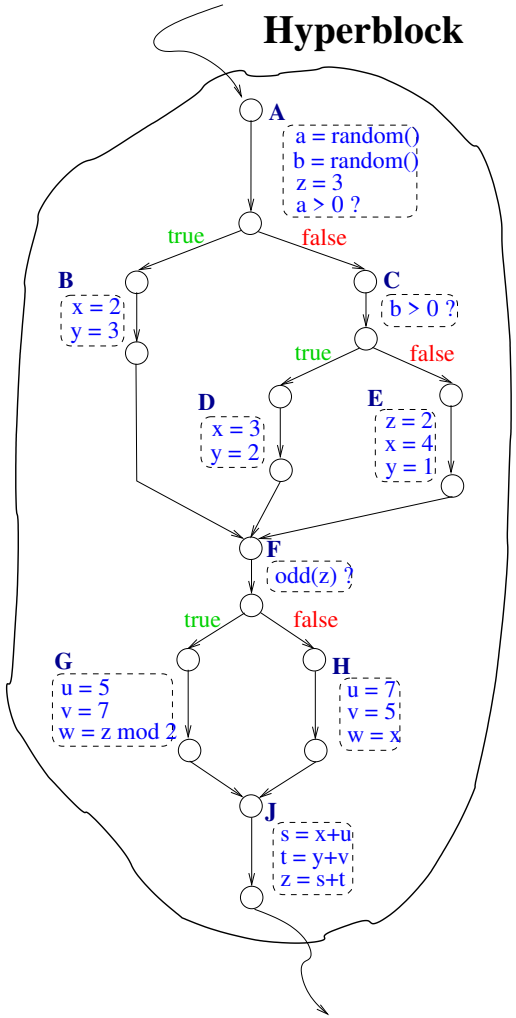
Kapitel 17.3.1

Hyperblöcke, Hyperblockgraphen

Hyperblöcke

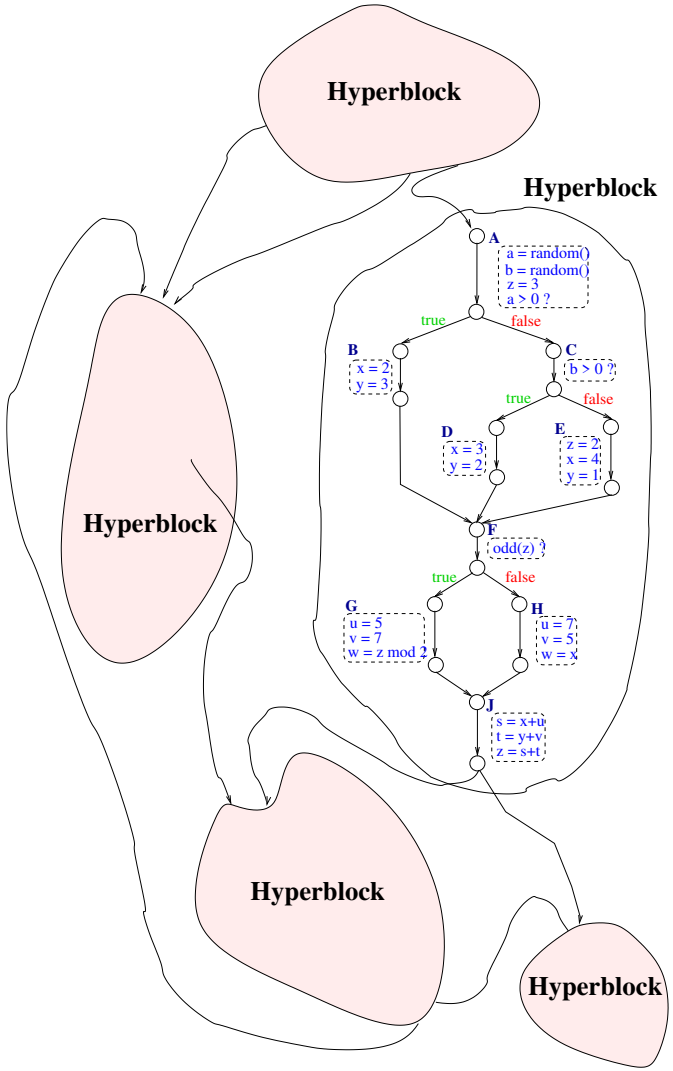
...sind Schlüsselbausteine prädikatierten Codes gekennzeichnet durch:

- ein Eintrittspunkt, mehrere Austrittspunkte.



Hyperblockgraphen

...sind die Hyperblockzerlegung eines Flussgraphen, hier anhand des durchgehenden Beispiels illustriert:



Aufbau der PVG-Konstantenanalyse

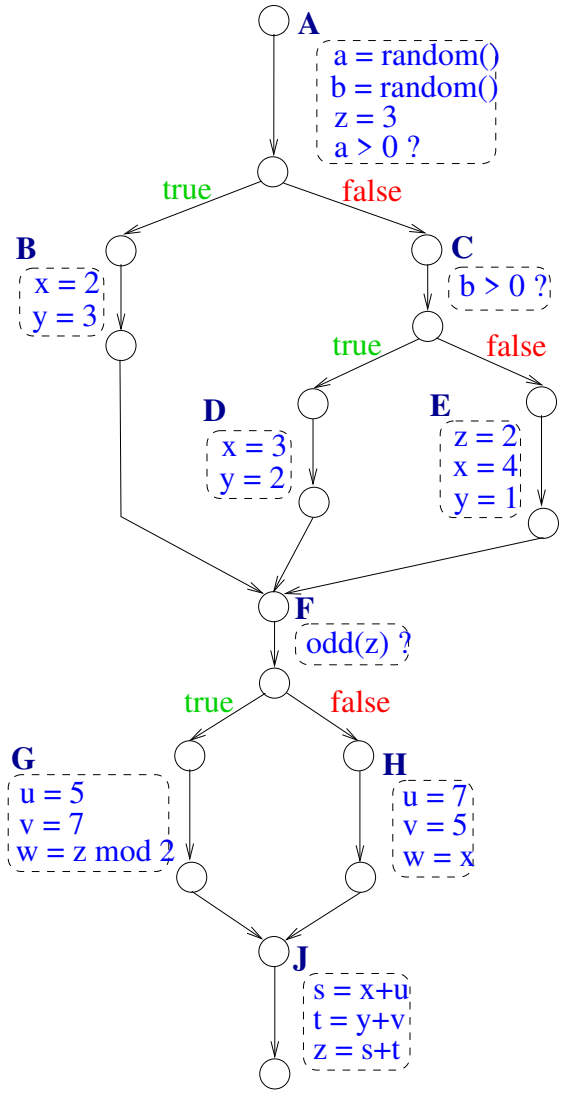
1. Stufe: Lokale Analyse
   - Separate und unabhängige Analyse aller Hyperblöcke eines Programms.
2. Stufe: Globale Analyse
   - Globalisierung der Ergebnisse der lokalen Analysestufe.

Kapitel 17.3.2

Lokale Hyperblock-Konstantenanalyse

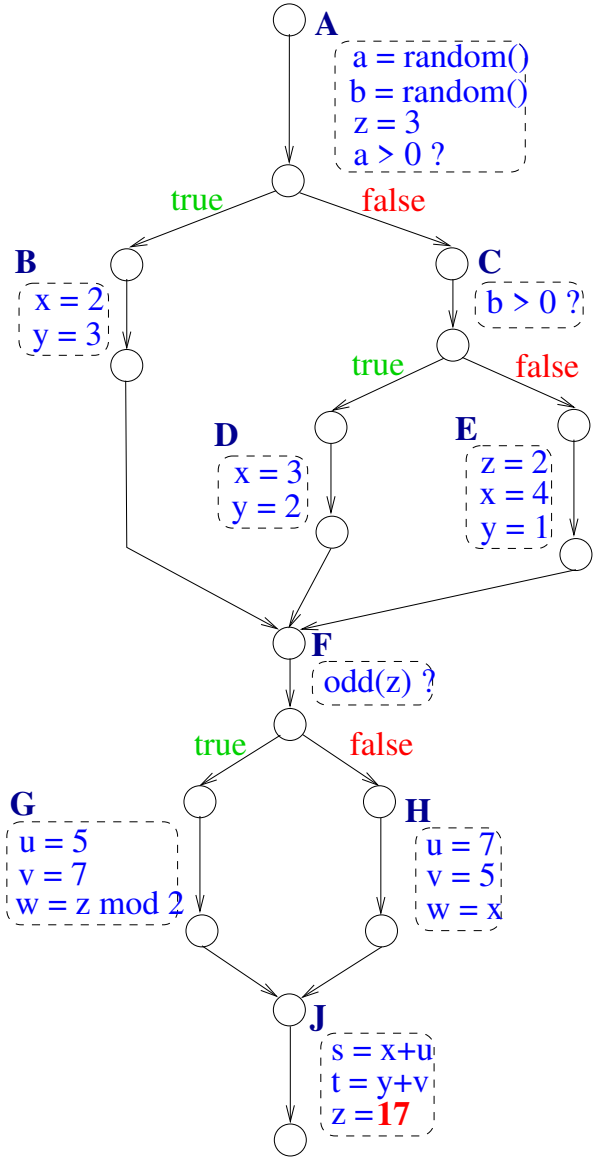
Diskussion der lokalen Analysestufe

....anhand des Hyperblocks unseres durchgehenden Beispiels:



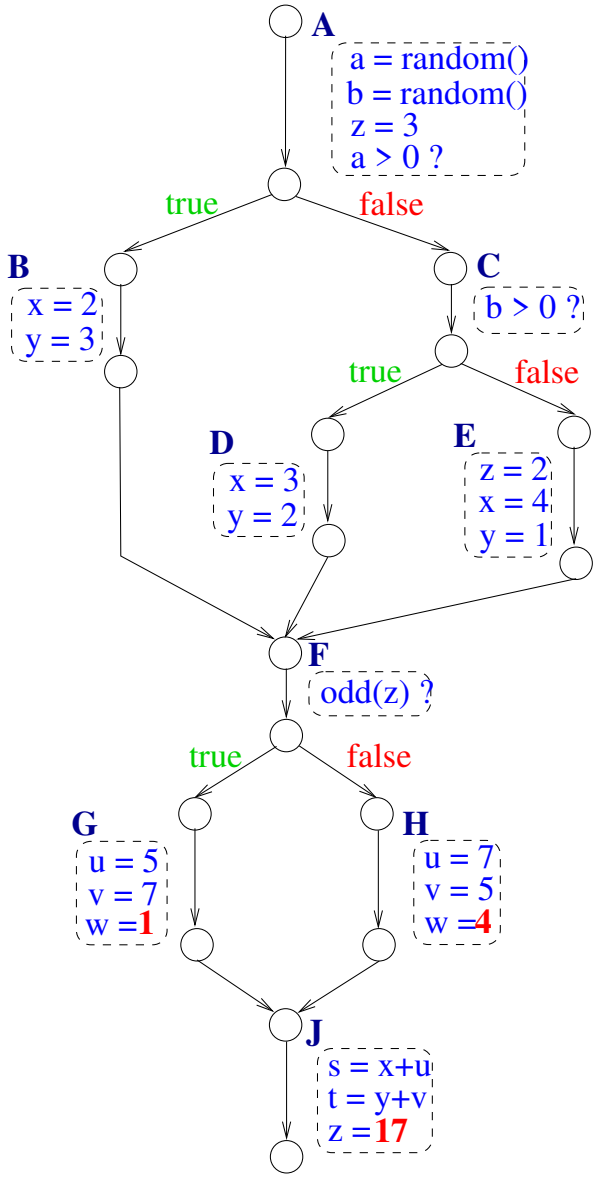
Der Ausgangs–Hyperblock

Die PVG-Grundalgorithmustransformation



Nach nichtdeterministischer
pfadpraeziser Grundoptimierung

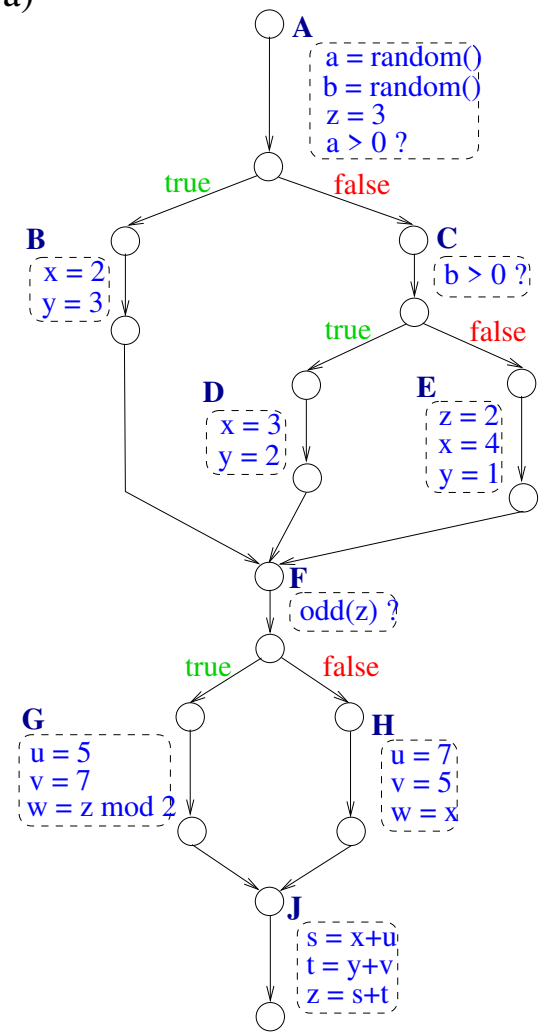
Die volle PVG-Algorithmustransformation



Nach deterministischer
pfadpraeziser voller Optimierung

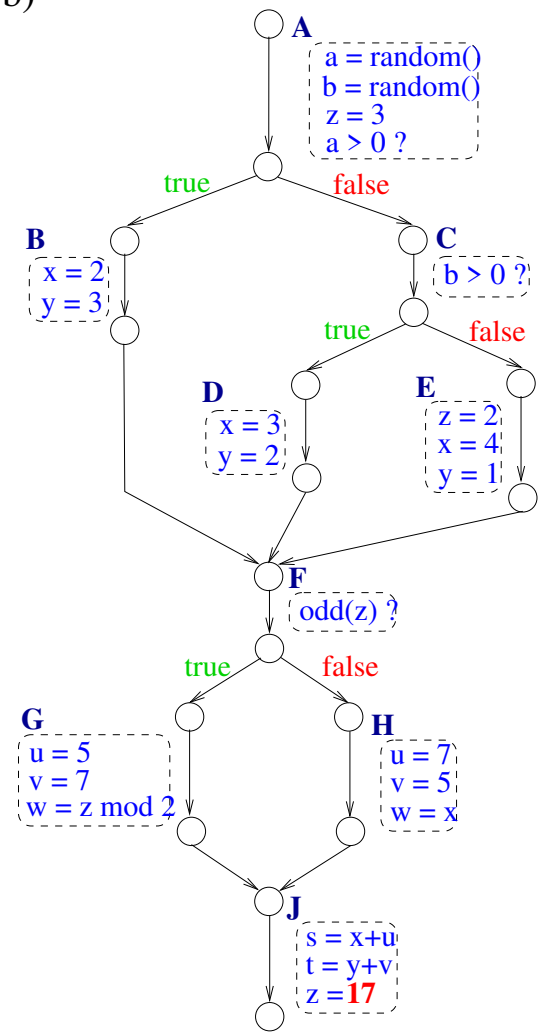
Ausgangsprogramm & beide Transformationen

a)



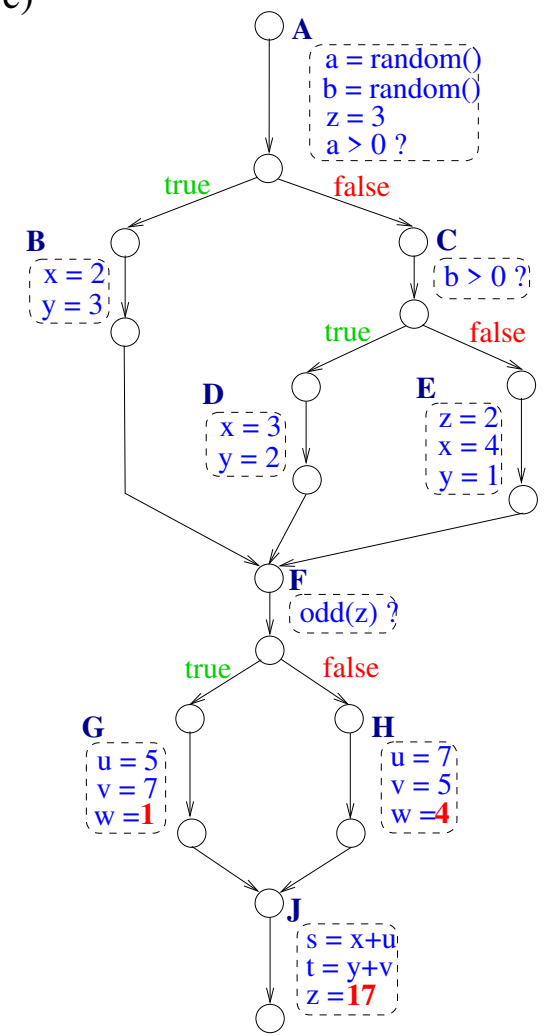
Der Ausgangs–Hyperblock

b)



Nach nichtdeterministischer pfadpraeziser Grundoptimierung

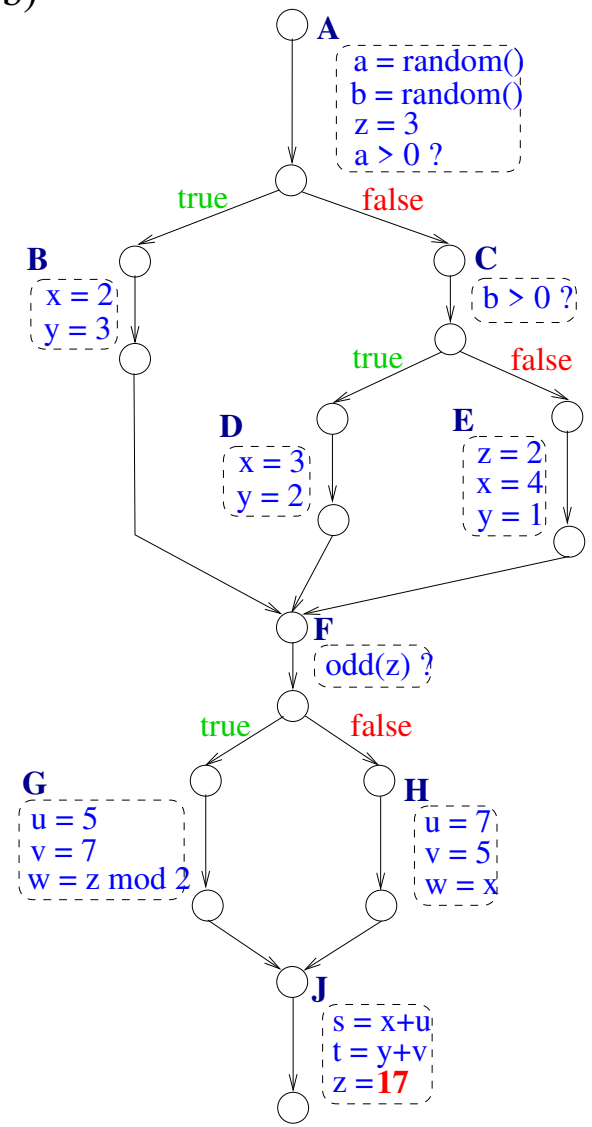
c)



Nach deterministischer pfadpraeziser voller Optimierung

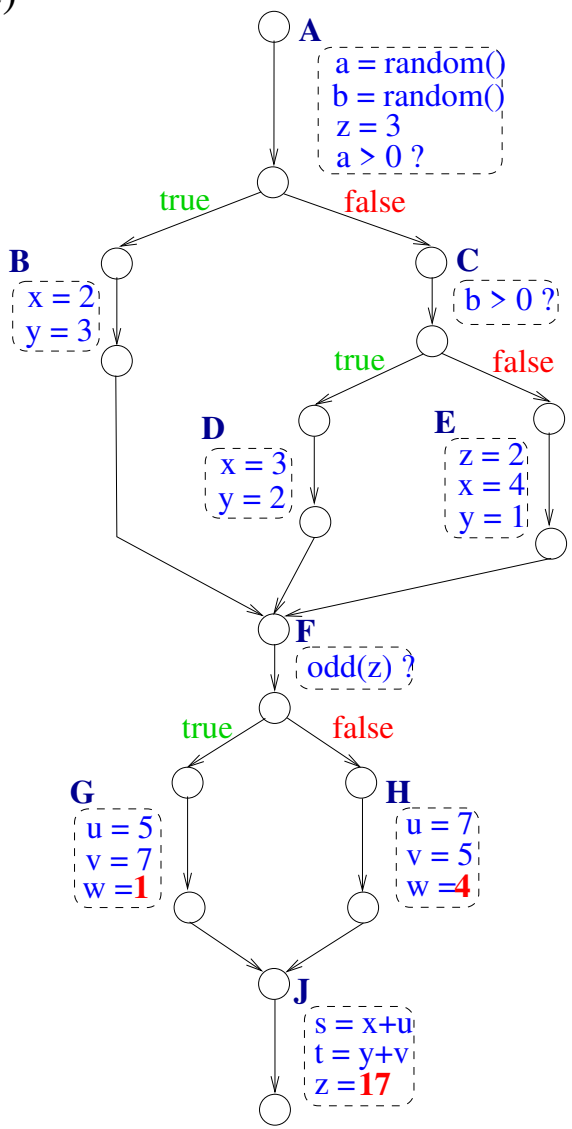
Beide Transformationen auf einen Blick

b)



Nach nichtdeterministischer pfadpraeziser Grundoptimierung

c)



Nach deterministischer pfadpraeziser voller Optimierung

Ursprünglicher und prädikatierter Code

```
Urspruenglicher Hyperblock | Nach if-Konversion
=========================== | ==================
```

```
begin \\ Original Hyperblock   | begin \\ After if-Conversion
 (a,b) = (random(),random());  |   (p0) (a,b) = (random(),random());
 z = 3;                        |   (p0) z = 3;
 if a>0 then                   |   (p0) cmp.unc B,C (a>0);
   x = 2;                      |   (B)  x = 2;
   y = 3                       |   (B)  y = 3;
 elsif b>0 then                |   (C)  cmp.unc D,E (b>0);
   x = 3;                      |   (D)  x = 3;
   y = 2                       |   (D)  y = 2;
 else                          |
   z = 2;                      |   (E)  z = 2;
   x = 4;                      |   (E)  x = 4;
   y = 1 fi;                   |   (E)  y = 1;
 if odd(z) then                |   (p0) cmp.unc G,H (odd(z));
   u = 5;                      |   (G)  u = 5;
   v = 7;                      |   (G)  v = 7;
   w = z mod 2                 |   (G)  w = z mod 2;
 else                          |
   u = 7;                      |   (H)  u = 7;
   v = 5;                      |   (H)  v = 5;
   w = x fi;                   |   (H)  w = x;
 s = x+u;                      |   (p0) s = x+u;
 t = y+v;                      |   (p0) t = y+v;
 z = s+t end.                  |   (p0) z = s+t end.
```

Prädikatierte SSA-Form (PSSA-Form)

...von Carter, Simon, Calder, Ferrante (PACT'99):

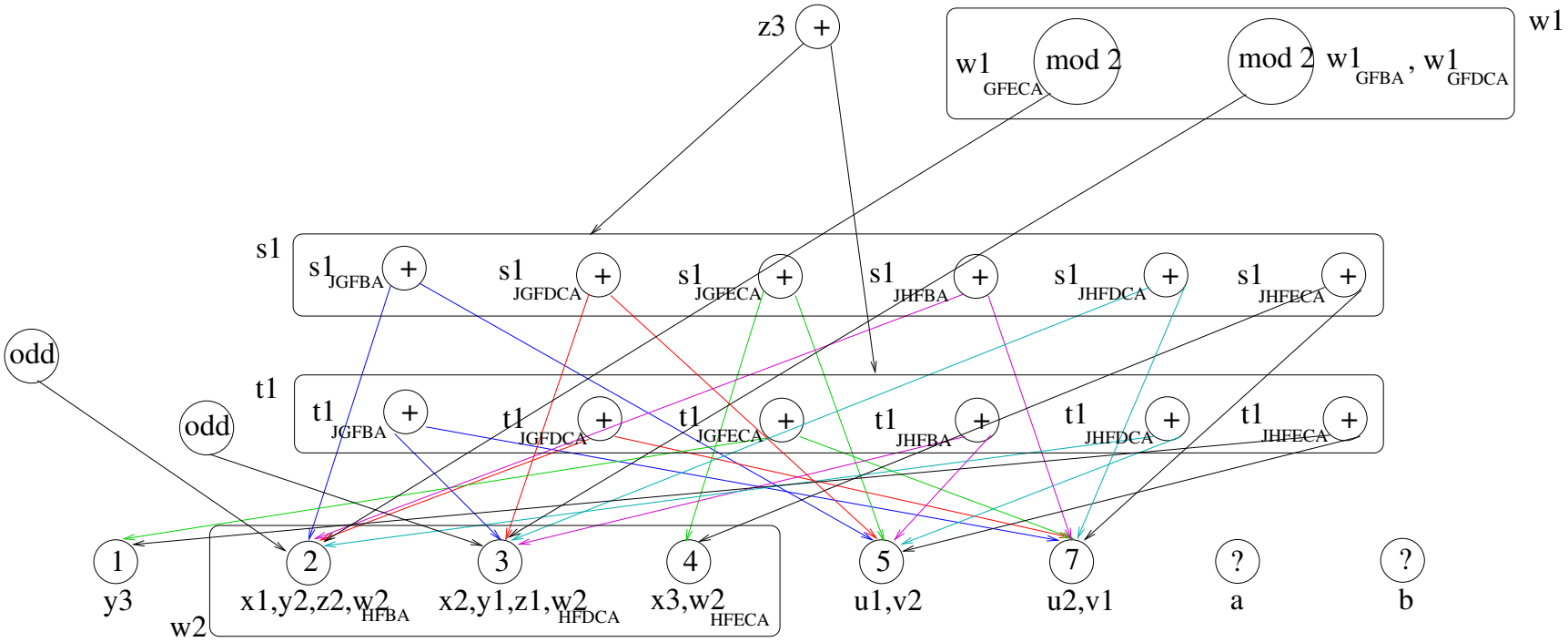
```
begin (p0)     A = OR(TRUE);                    | [*] (HFBA)   w2 = x1;
      (A)      (a1,b1) = (random(),random());   | [*] (HFDCA)  w2 = x2;                 |
      (A)      z1 = 3;                          |     (HFECA)  w2 = x3;
      (A)      cmp.unc BA,CA (a1>0);            |     (H)      u2 = 7;
      (p0)     B = OR(BA);                      |     (H)      v2 = 5;
      (p0)     C = OR(CA);                      |     (GFBA)   JGFBA = OR(TRUE);
      (B)      x1 = 2;                          |     (GFDCA)  JGFDCA = OR(TRUE);
      (B)      y1 = 3;                          | [*] (GFECA)  JGFECA = OR(TRUE);
      (C)      cmp.unc DCA,ECA (b1>0);          | [*] (HFBA)   JHFBA = OR(TRUE);
      (p0)     D = OR(DCA);                     | [*] (HFDCA)  JHFDCA = OR(TRUE);
      (p0)     E = OR(ECA);                     |     (HFECA)  JHFECA = OR(TRUE);
      (D)      x2 = 3;                          | [-] (p0)     J = OR(JGFBA,JGFDCA,
      (D)      y2 = 2;                          |                     JGFECA,JHFBA,
      (E)      z2 = 2;                          |                     JHFDCA,JHFECA);
      (E)      x3 = 4;                          |     (JGFBA)  s1 = x1+u1;
      (E)      y3 = 1;                          |     (JGFBA)  t1 = y1+v1;
      (BA)     FBA = OR(TRUE);                  | [*] (JGFDCA) s1 = x2+u1;
      (DCA)    FDCA = OR(TRUE);                 | [*] (JGFDCA) t1 = y2+v1;
      (ECA)    FECA = OR(TRUE);                 |     (JGFECA) s1 = x3+u1;
      (p0)     F = OR(FBA,FDCA,FECA);           |     (JGFECA) t1 = y3+v1;
      (FBA)    cmp.unc GFBA,HFBA (odd(z1));     | [*] (JHFBA)  s1 = x1+u2;
      (FDCA)   cmp.unc GFDCA,HFDCA (odd(z1));   | [*] (JHFBA)  t1 = y1+v2;
      (FECA)   cmp.unc GFECA,HFECA (odd(z2));   | [*] (JHFDCA) s1 = x2+u2;
  [-] (p0)     G = OR(GFBA,GFDCA,GFECA);        | [*] (JHFDCA) t1 = y2+v2;
  [-] (p0)     H = OR(HFBA,HFDCA,HFECA);        |     (JHFECA) s1 = x3+u2;
      (GFBA)   w1 = z1 mod 2;                   |     (JHFECA) t1 = y3+v2;
      (GFDCA)  w1 = z1 mod 2;                   |     (J)      z3 = s1+t1;
  [*] (GFECA)  w1 = z2 mod 2;                   |    end.
      (G)      u1 = 5;                          |
      (G)      v1 = 7;                          |
```

Kapitel 17.3.3

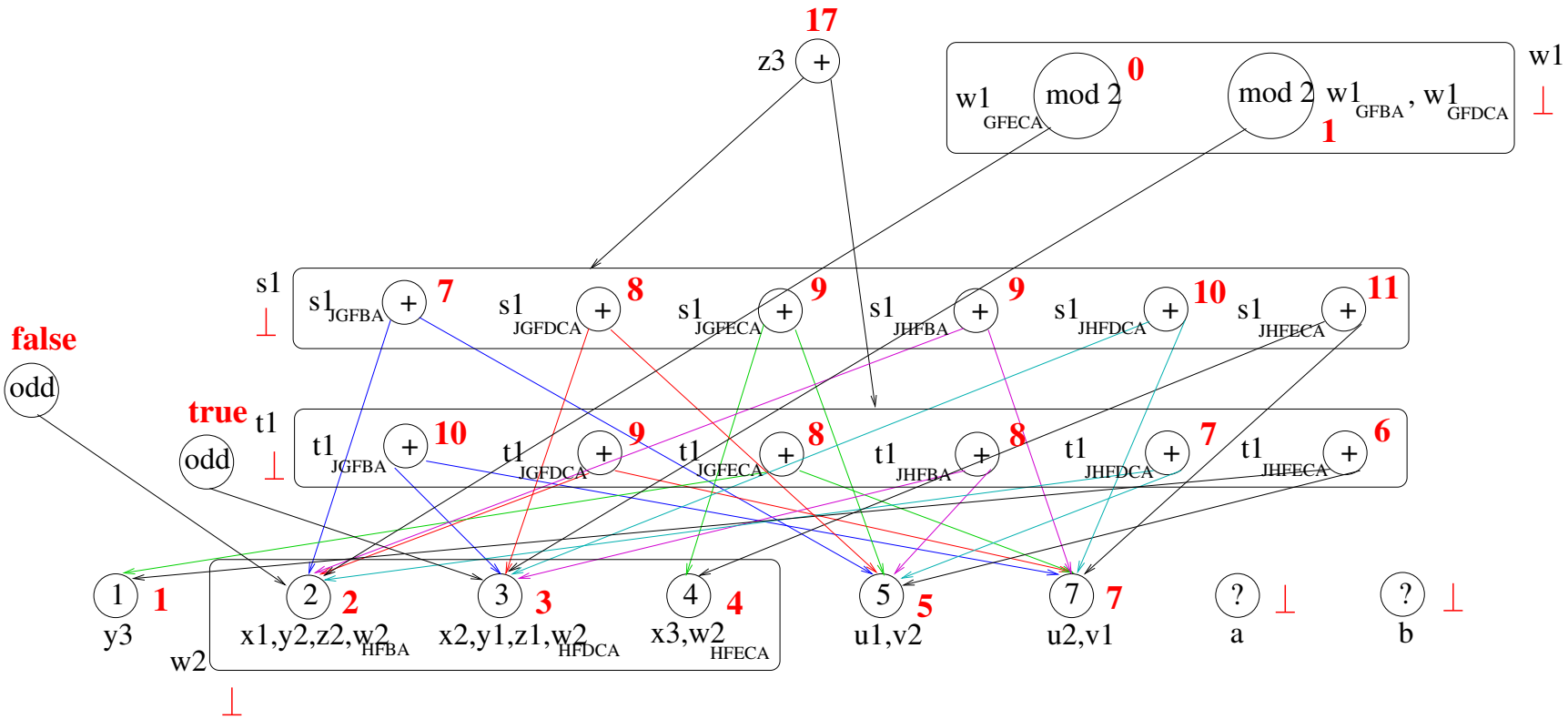
PVG-Basiskonstantenanalyse

Die PVG-Basiskonstantenanalyse

... auf PSSA-basiertem PVG ohne Wächterprädikatausnutzung (engl. guarding predicates):

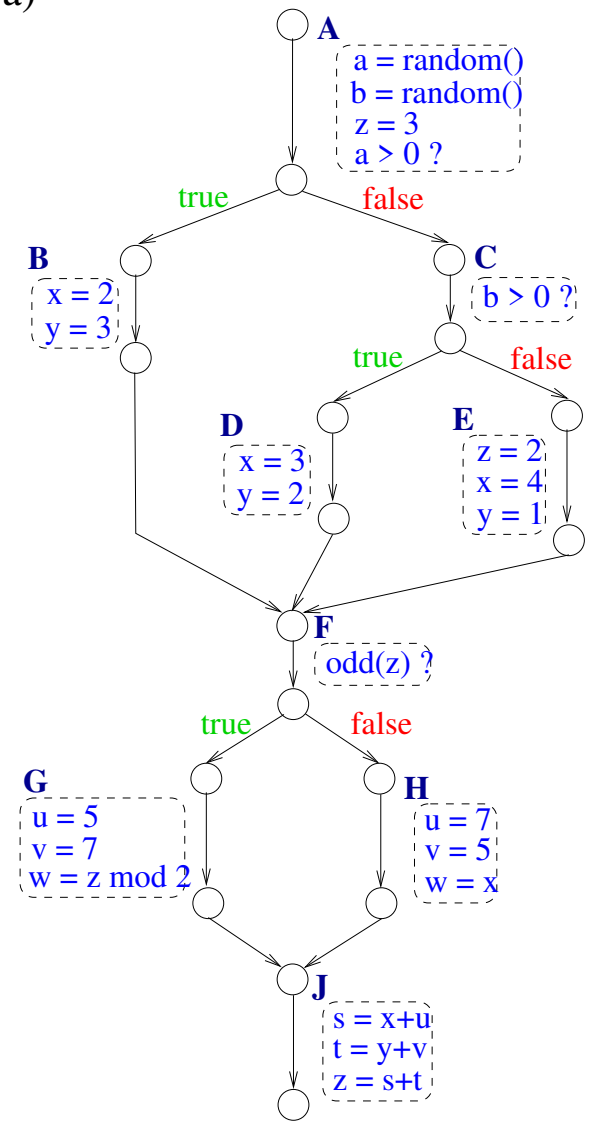


Resultat der PVG-Basiskonstantenanalyse



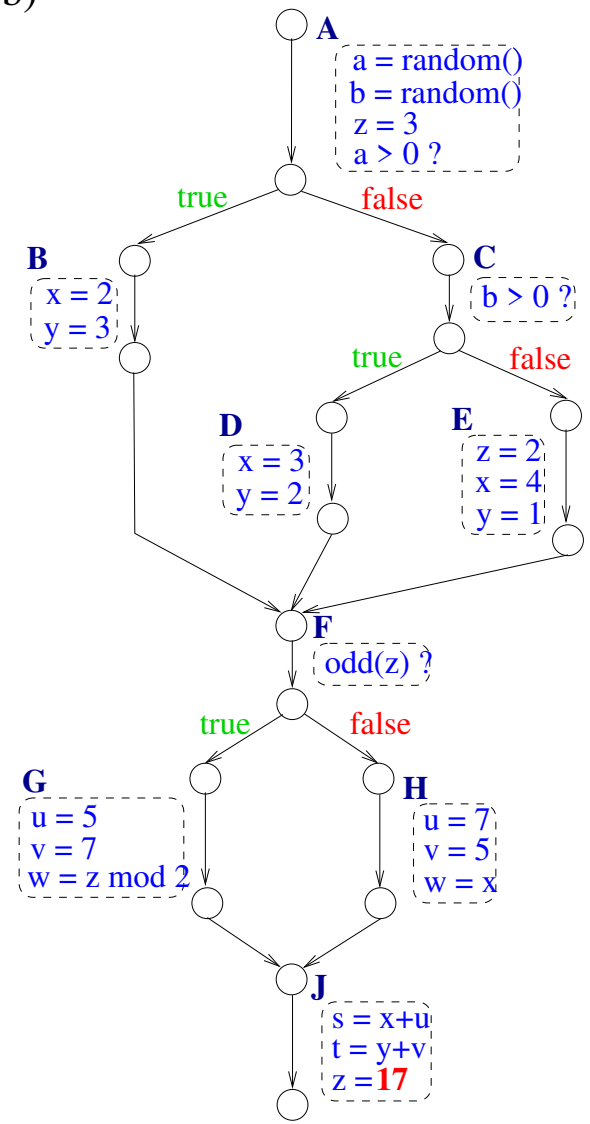
PVG-Basiskonstantenanalysetransformation

a)



Der Ausgangs–Hyperblock

b)



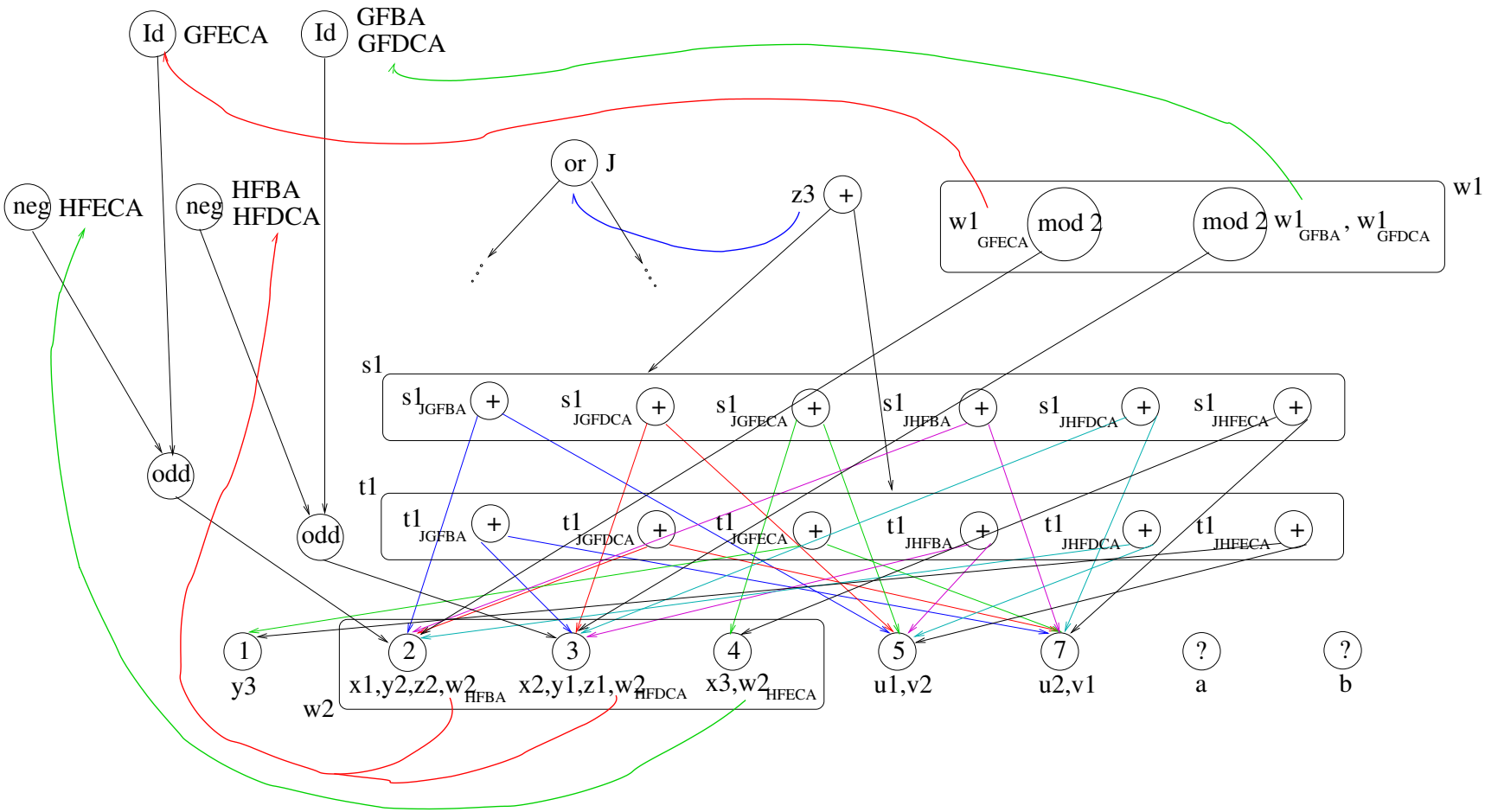
Nach nichtdeterministischer pfadpraeziser Grundoptimierung

Kapitel 17.3.4

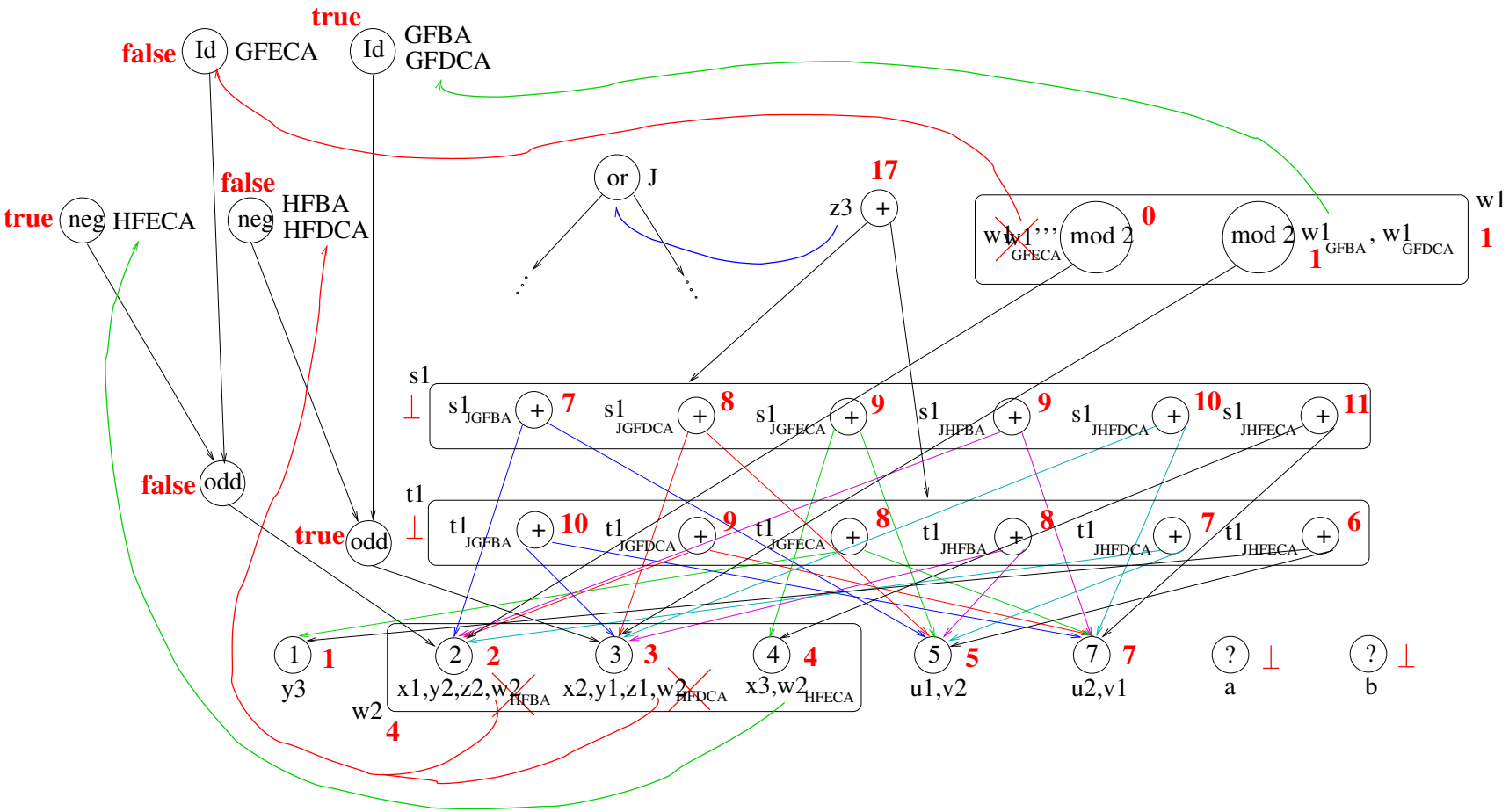
Volle PVG-Konstantenanalyse

Der prädikatierte Wertegraph

...unter Ausnutzung der Wächterprädikate (engl. guarding predicates):

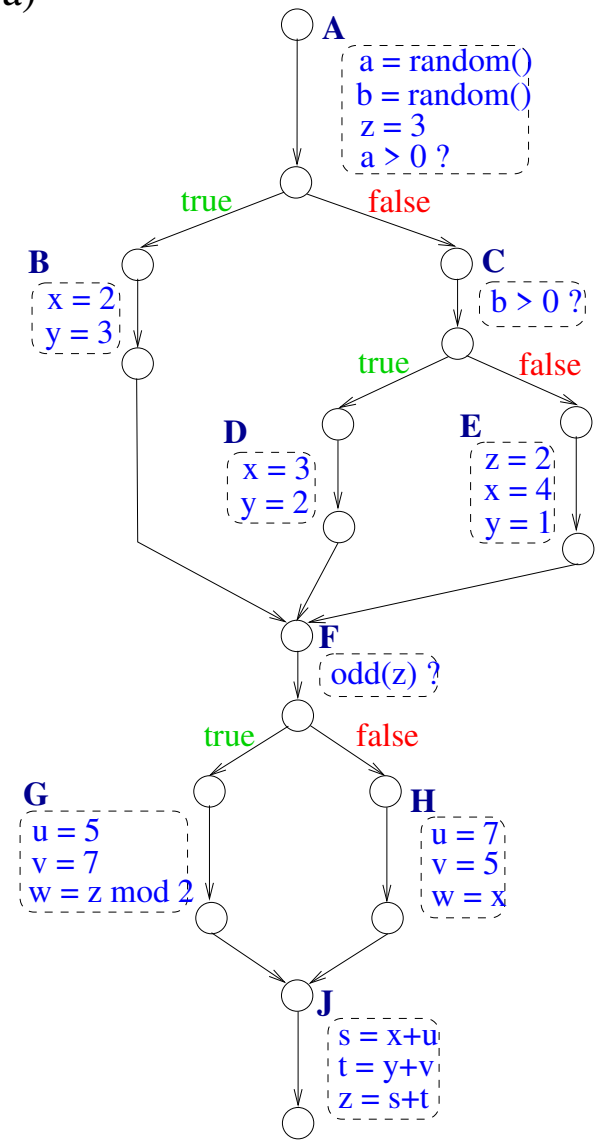


Resultat der vollen PVG-Konstantenanalyse



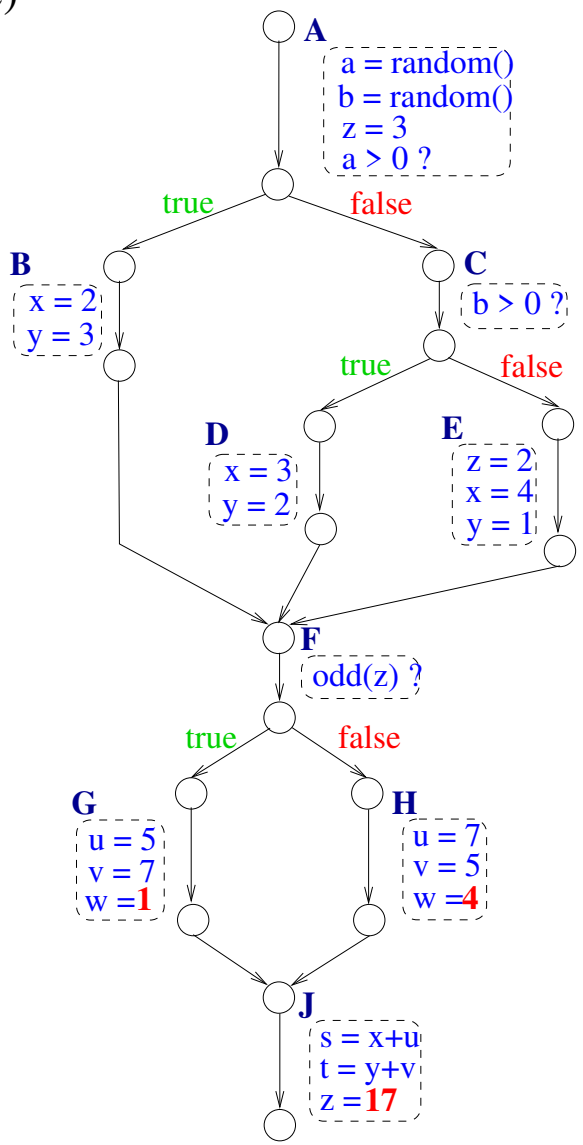
Volle PVG-Konstantenanalysetransformation

a)



Der Ausgangs–Hyperblock

c)



Nach deterministischer
pfadpraeziser voller Optimierung

Der transformierte Hyperblock in PSSA-Form

```
begin (p0)     A = OR(TRUE);                |
      (A)      a1 = random();               | [-] (p0)     G = OR(GFBA,GFDCA);
      (A)      b1 = random();               | [-] (p0)     H = OR(HFECA);
      (A)      z1 = 3;                      |     (G)      w1 = 1;
      (A)      cmp.unc BA,CA (a1>0);        |     (G)      u1 = 5;
      (p0)     B = OR(BA);                  |     (G)      v1 = 7;
      (p0)     C = OR(CA);                  |     (HFECA)  w2 = 4;
      (B)      x1 = 2;                      |     (H)      u2 = 7;
      (B)      y1 = 3;                      |     (H)      v2 = 5;
      (C)      cmp.unc DCA,ECA (b1>0);      |     (GFBA)   JGFBA = OR(TRUE);
      (p0)     D = OR(DCA);                 |     (GFDCA)  JGFDCA = OR(TRUE);
      (p0)     E = OR(ECA);                 |     (HFECA)  JHFECA = OR(TRUE);
      (D)      x2 = 3;                      | [-] (p0)     J = OR(JGFBA,JGFECA,
      (D)      y2 = 2;                      |                     JHFECA);
      (E)      z2 = 2;                      |     (JGFBA)  s1 = 7;
      (E)      x3 = 4;                      |     (JGFBA)  t1 = 10;
      (E)      y3 = 1;                      |     (JGFECA) s1 = 9;
      (BA)     FBA = OR(TRUE);              |     (JGFECA) t1 = 8;
      (DCA)    FDCA = OR(TRUE);             |     (JHFECA) s1 = 11;
      (ECA)    FECA = OR(TRUE);             |     (JHFECA) t1 = 6;
      (p0)     F = OR(FBA,FDCA,FECA);       |     (J)      z3 = 17;
      (FBA)    cmp.unc GFBA,HFBA (TRUE));   |   end.
      (FDCA)   cmp.unc GFDCA,HFDCA (TRUE);  |
      (FECA)   cmp.unc GFECA,HFECA (FALSE); |
```

Hauptresultate

Lemma 17.3.4.1 (Korrektheit)

Die PVG-Basis- und die volle PVG-Konstantenanalyse sind korrekt.

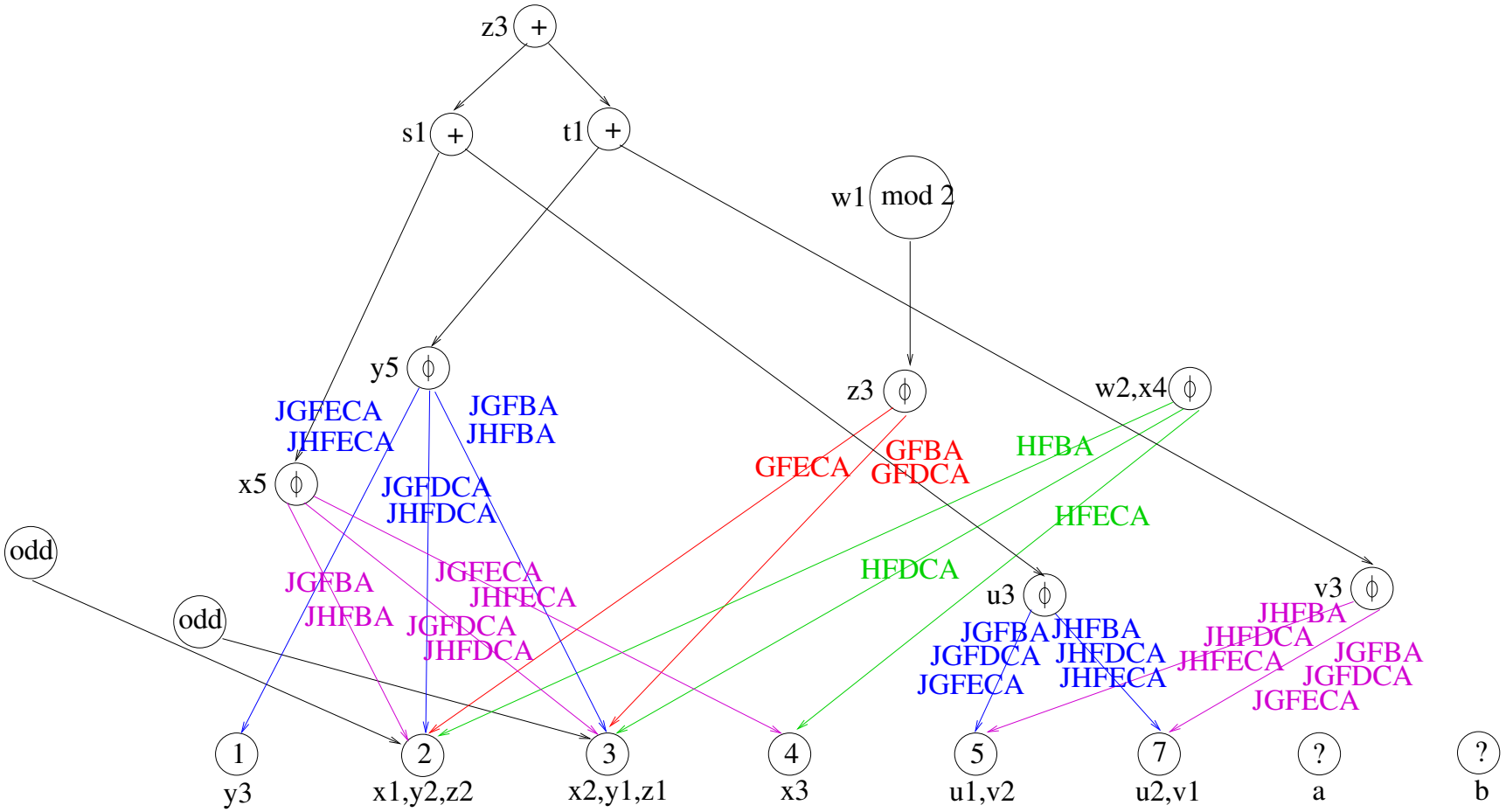
Lemma 17.3.4.2 (Vollständigkeit/Optimalität)

- Die PVG-Basiskonstantenanalyse ist pfad-präzise für nicht-deterministische Interpretation von Verzweigungsbedingungen.
- Die volle PVG-Konstantenanalyse ist prädikatsensitiv pfad-präzise.

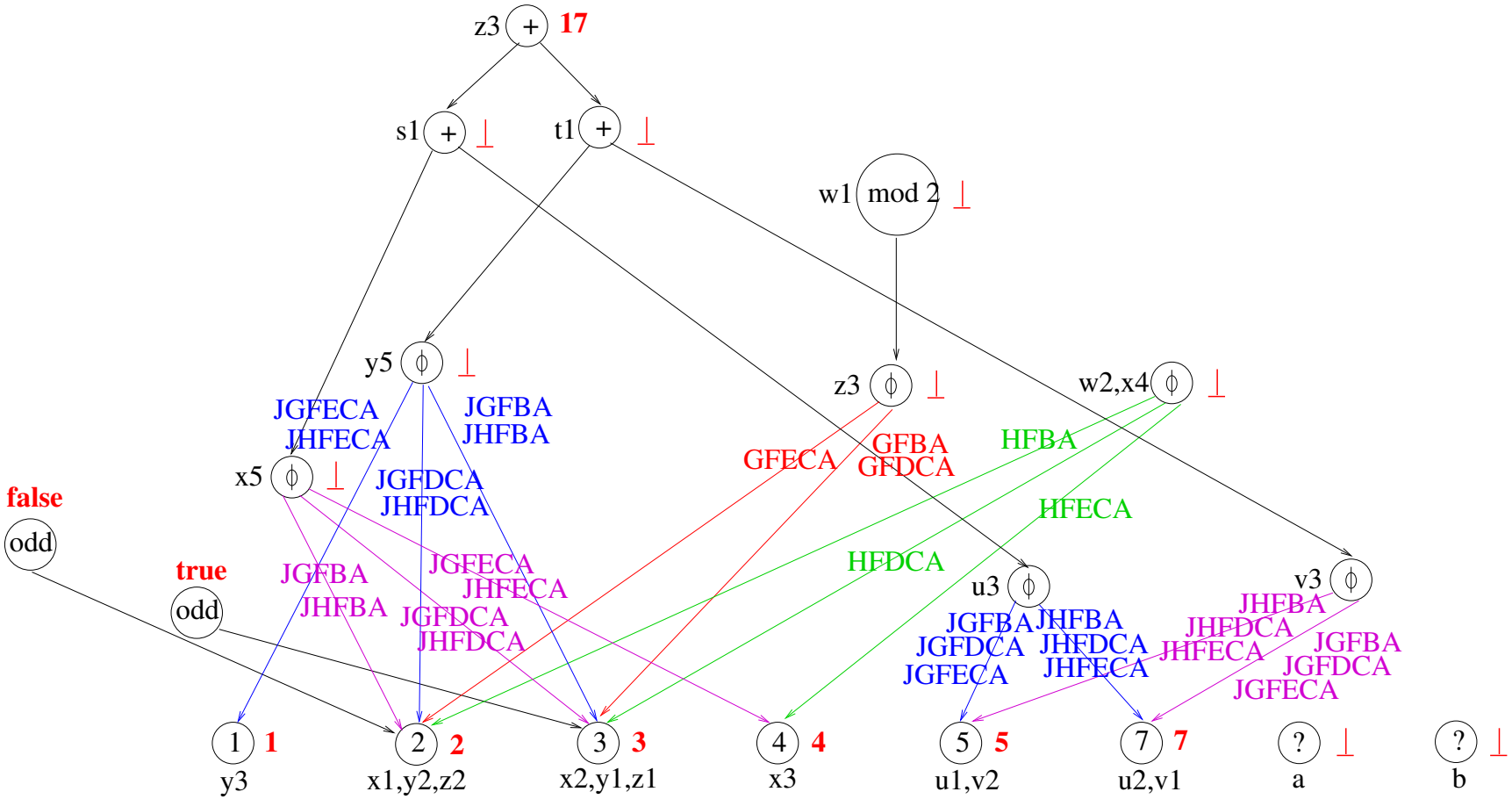
Kapitel 17.3.5

Variationen zur Performanzverbesserung

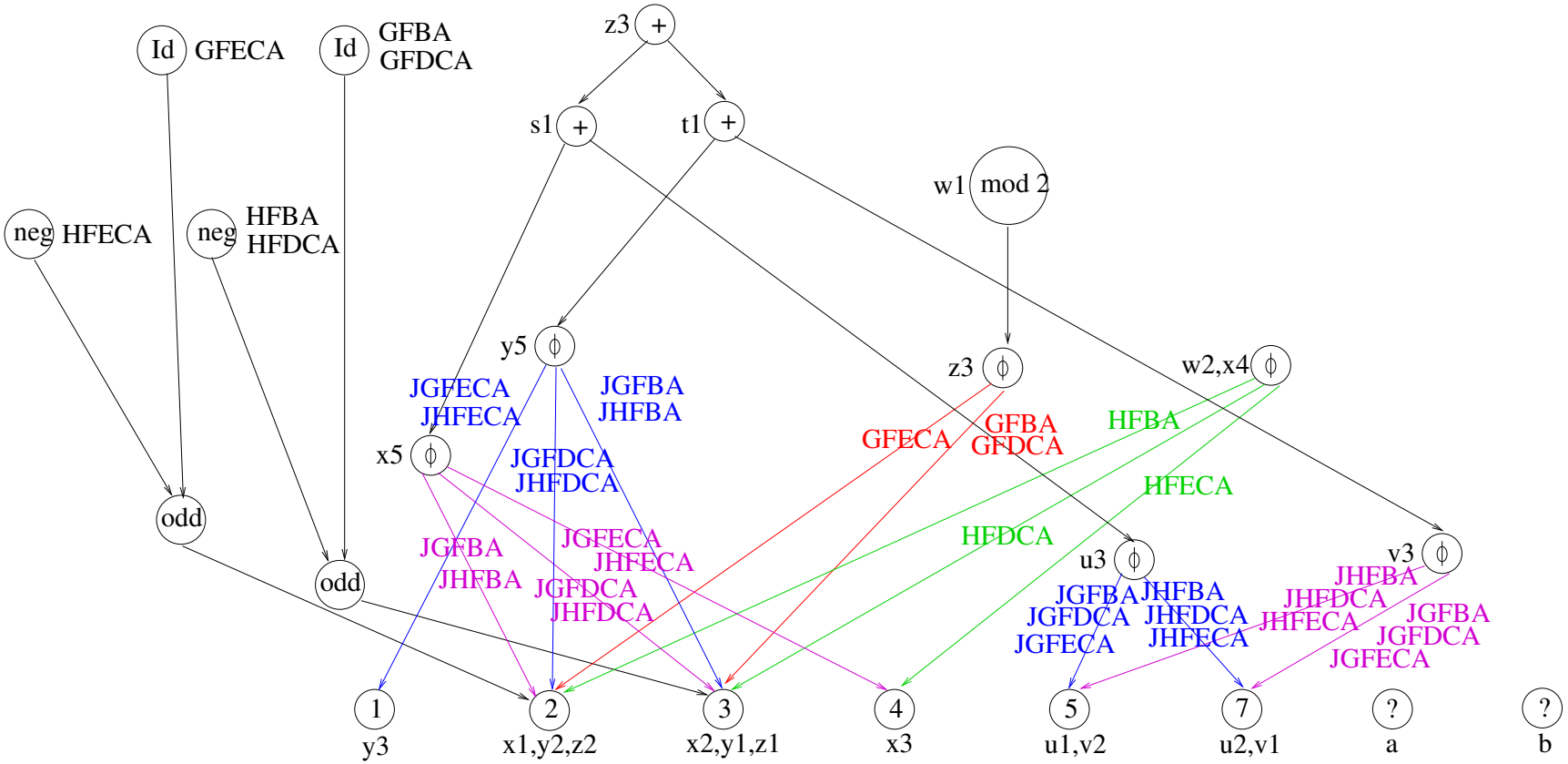
Performanzverbesserung der PVG-Basisanalyse



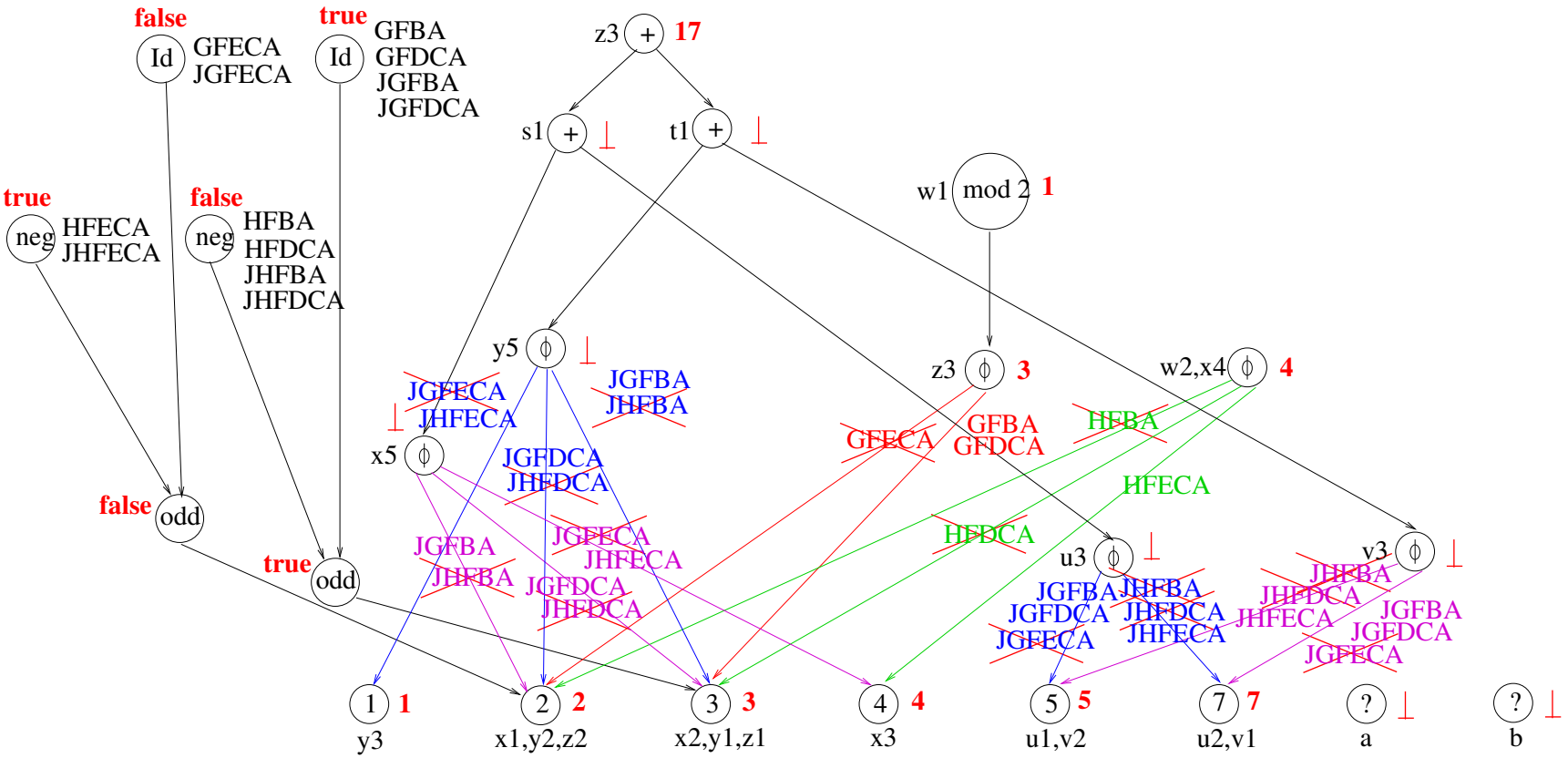
Wirkung d. Performanzverb. f. d. Basisanalyse



Performanzverbesserung d. vollen PVG-Analyse



Wirkung d. Performanzverb. f. d. volle Analyse



Kapitel 17.4

Zusammenfassung

Zusammenfassung

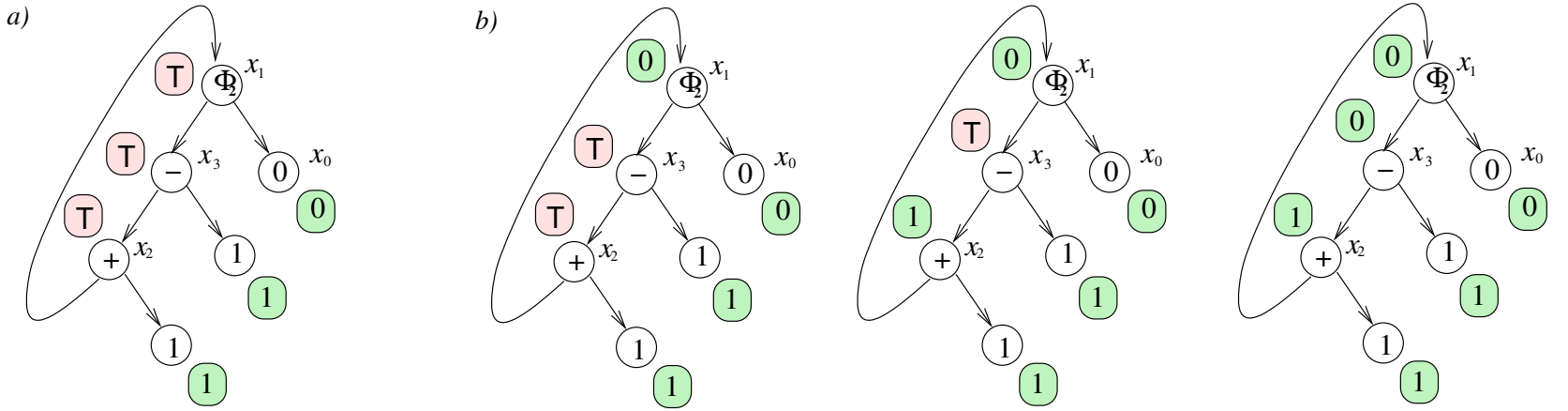
Konstantenanalyse und SSA/PSSA-basierter (prädikatierter) Wertegraph sind

- perfekt aufeinander abgestimmt: SSA/PSSA-Programmdarstellungen zeigen sich als
  - wirklich von Hilfe für Konstantenanalyse.
  - Grundlage und Schlüssel für Wertegraph und prädikatierten Wertegraph.
- offen für Erweiterungen, z.B.:
  - Bedingte Konstanten (engl. conditional constants)

  auf dem (prädikatierten) Wertegraph.

Konstantenanalyse auf dem Wertegraph

...erzielt **Triple E** Rating: **E**xpressive, **E**fficient, **E**asy!



Nach der Initialisierung | Nach der 1. Iteration | Nach der 2. Iteration | Nach der 3. Iteration: Stabil!

und ist von daher Vorzeigeanwendung, Vorteile von

- (P)SSA als Programmdarstellung für Analyse und Transformation, insbesondere Optimierung

aufzuzeigen.

Literaturhinweise

...zur VG-Konstantenanalyse:

- Jens Knoop, Oliver Rüthing. Constant Propagation on the Value Graph: Simple Constants and Beyond. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 94-109, 2000.

...zur PVG-Konstantenanalyse:

- Jens Knoop, Oliver Rüthing. Constant Propagation on Predicated Code. Journal of Universal Computer Science 9(8):829-850, 2003. (Sonderausgabe zur SBLP'03).

Kapitel 17.5

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (1)

Bowen Alpern, Mark N. Wegman, F. Kenneth Zadeck. *Detecting Equality of Variables in Programs*. In Conference Record of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'88), 1-11, 1988.

Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck. *An Efficient Method of Computing Static Single Assignment Form*. In Conference Record of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'89), 25-35, 1989.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (2)


Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck. *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph*. ACM Transactions on Programming Languages and Systems (TOPLAS) 13(4):451-490, 1991.


John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.


Gary A. Kildall. *A Unified Approach to Global Program Optimization*. In Conference Record of the 1st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'73), 194-206, 1973.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (3)


Jens Knoop, Oliver Rüthing. *Constant Propagation on the Value Graph: Simple Constants and Beyond*. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 94-109, 2000.


Jens Knoop, Oliver Rüthing. *Constant Propagation on Predicated Code*. Journal of Universal Computer Science 9(8):829-850, 2003. (Sonderausgabe zur SBLP'03).


Markus Müller-Olm, Helmut Seidl. *Polynomial Constants are Decidable*. In Proceedings of the 9th Static Analysis Symposium (SAS 2002), Springer-V., LNCS 2477, 4-19, 2002.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (4)


John H. Reif, Harry R. Lewis. *Symbolic Evaluation and the Global Value Graph*. In Conference Record of the 4th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77), 104-118, 1977.


Oliver Rüthing, Jens Knoop, Bernhard Steffen. *Detecting Equalities of Variables: Combining Efficiency with Precision*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 232-247, 1999.


Oliver Rüthing, Markus Müller-Olm. *On the Complexity of Constant Propagation*. In Proceedings of the 10th European Symposium on Programming (ESOP 2001), Springer-V., LNCS 2028, 190-205, 2001.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (5)

- Bernhard Steffen, Jens Knoop. *Finite Constants: Characterizations of a New Decidable Set of Constants*. Theoretical Computer Science 80(2):303-318, 1991.
- Munehiro Takimoto, Kenichi Harada. *Effective Partial Redundancy Elimination based on Extended Value Graph*. Information Processing Society of Japan 38(11):2237-2250, 1990.
- Munehiro Takimoto, Kenichi Harada. *Partial Dead Code Elimination Using Extended Value Graph*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 179-193, 1999.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (6)

Mark N. Wegman, F. Kenneth Zadeck. *Constant Propagation with Conditional Branches*. In Conference Record of the 12th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'85), 291-299, 1985.

Mark N. Wegman, F. Kenneth Zadeck. *Constant Propagation with Conditional Branches*. ACM Transactions on Programming Languages and Systems 13(2):181-201, 1991.

Teil VI

Abstrakte Interpretation und Modellprüfung

Kapitel 18

Abstrakte Interpretation und Datenfluss-analyse

Kapitel 18.1

Motivation

Motivation

...abstrakte Interpretation – ein 'mondäner' Programmanalyse-ansatz (cf. Nielson, Nielson, Hankin, 2005).

Intuitiv, der

- DFA-Ansatz beinhaltet die Spezifikation einer Programmanalyse, deren Korrektheit separat und unabhängig *a posteriori*
- abstrakte Interpretationsansatz beinhaltet den Korrektheitsnachweis von Anfang an als integralen Bestandteil der Spezifikation einer Programmanalyse, wodurch Korrektheit *a priori*

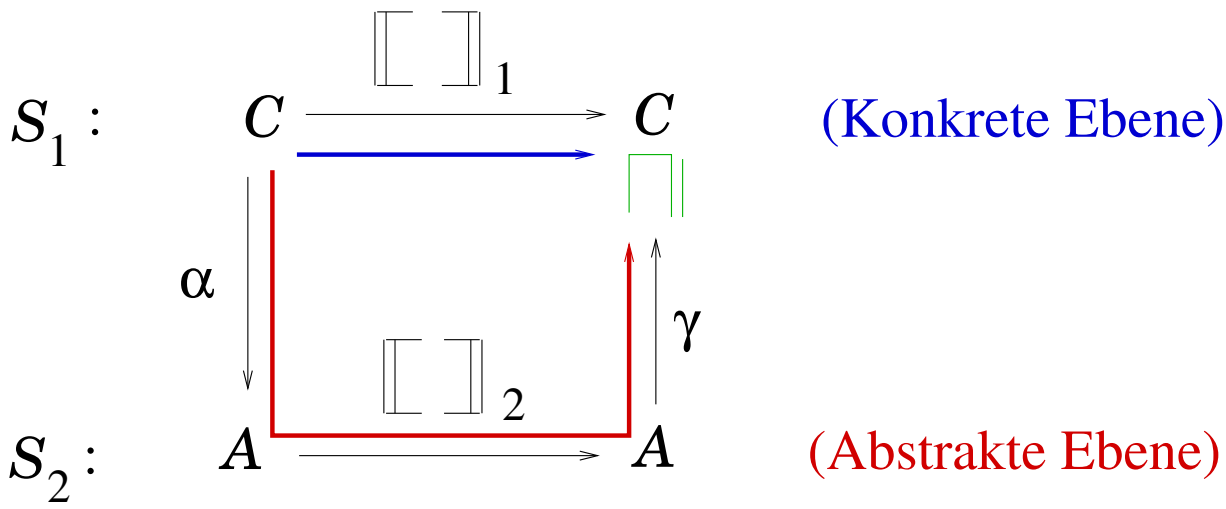
bewiesen wird.

Kapitel 18.2

Theorie abstrakter Interpretation

Theorie abstrakter Interpretation

...ein Ansatz mit zwei (oder mehr) Beobachtungsniveaus:



zusammengehalten durch Wohlzusammenhangsforderungen:

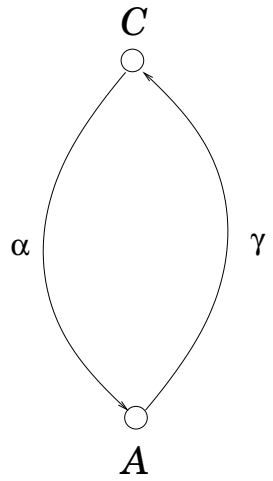
1. $\alpha : \mathcal{C} \to \mathcal{A}$ und $\gamma : \mathcal{A} \to \mathcal{C}$ als sog. Abstraktions- und Konkretisierungsfunktionen bilden eine Galois-Verbindung.
2. das Diagramm (schwach) kommutativ ist.

Im folgenden

...bezeichnen:

- $\mathcal{C} = (C, \sqcup_{\mathcal{C}}, \sqsubseteq_{\mathcal{C}}, \bot_{\mathcal{C}}, \top_{\mathcal{C}})$, $\mathcal{A} = (A, \sqcup_{\mathcal{A}}, \sqsubseteq_{\mathcal{A}}, \bot_{\mathcal{A}}, \top_{\mathcal{A}})$: Vollständige (Vereinigungs-) Halbverbände.
- $\alpha : C \to A$: sog. Abstraktionsfunktion.
- $\gamma : A \to C$: sog. Konkretisierungsfunktion.

...womit das Tupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ folgende Situation beschreibt:



Generalvereinbarung:

- Verbandmäßig kleiner heißt bessere, genauere Information!
- $Id_C : C \to C$, $Id_A : A \to A$: Identitäten auf C und A, d.h.: $Id_C = \lambda\, c.\, c$ und $Id_A(a) = \lambda\, a.\, a$.

Beachte

...in der Theorie abstrakter Interpretationen ist die Generalvereinbarung verbandmäßig kleiner bedeutet

- bessere, genauere Information

genauer andersherum getroffen als in der Theorie der Datenflussanalyse, wo verbandmäßig kleiner

- schlechtere, ungenauere Information

bedeutet (vgl. Kapitel 8.1).

Kapitel 18.2.1

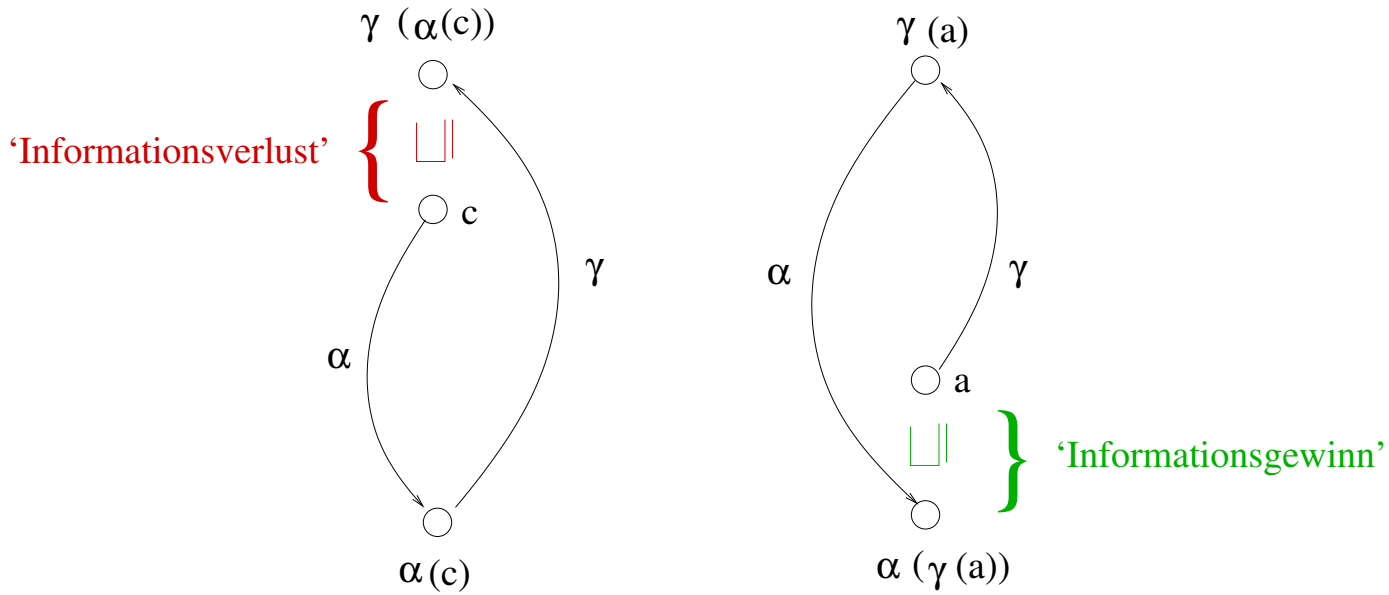
Galois-Verbindungen

Galois-Verbindungen

Definition 18.2.1.1 (Galois-Verbindung)

Ein Quadrupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ heißt Galois-Verbindung (engl. Galois connection) gdw:

1. α und γ sind monoton
2. $\gamma \circ \alpha \sqsupseteq_{\mathcal{C}} Id_{\mathcal{C}}$ ($\sqsupseteq_{\mathcal{C}}$: linksseitig 'Informationsverlust')
3. $\alpha \circ \gamma \sqsubseteq_{\mathcal{A}} Id_{\mathcal{A}}$ ($\sqsubseteq_{\mathcal{A}}$: linksseitig 'Informationsgewinn')



Informell

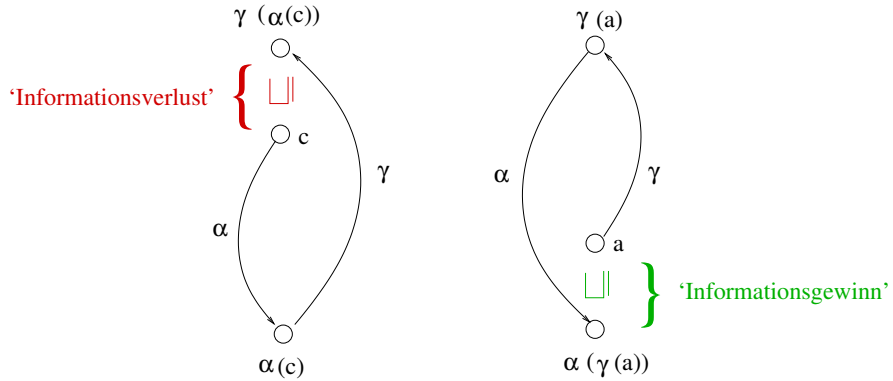
Abstraktion

- kann schaden: Ist die Inklusion $\gamma \circ \alpha \sqsupseteq_C Id_C$ in Definition 18.2.1.1(1) echt, so bedeutet das eine Schlechterstellung, einen Informationsverlust auf der konkreten Ebene.

Konkretisierung

- darf nicht schaden: Ist die Inklusion $\alpha \circ \gamma \sqsubseteq_A Id_A$ in Definition 18.2.1.1(2) echt, so bedeutet das (sogar) eine Besserstellung, einen Informationsgewinn auf der abstrakten Ebene.

...entsprechend der Generalvereinbarung: 'kleiner' ist 'besser'.



Maximaler Informationsverlust, -gewinn

Proposition 18.2.1.2 (Triviale, nutzlose Galois-Verb.)

$Q_{triv} = (\mathcal{C}, \alpha, \gamma, \mathcal{A})$ mit $\alpha : \mathcal{C} \to \mathcal{A}$, $\gamma : \mathcal{A} \to \mathcal{C}$ definiert durch:

- $\alpha = \lambda\, c.\, \bot_{\mathcal{A}}$
- $\gamma = \lambda\, a.\, \top_{\mathcal{C}}$

ist eine Galois-Verbindung.

Beachte: Q_{triv} erfüllt die Anforderungen an eine Galois-Verb.

- trivialerweise.
- maximiert die Informations-
  - Schlechterstellung durch Abstraktions-/Konkretisierungs-abfolge: $\gamma \circ \alpha = \lambda c.\ \top_{\mathcal{C}} \ \sqsupseteq_{\mathcal{C}} \ Id_C$
  - Besserstellung durch Konkretisierungs-/Abstraktionsab-folge: $\alpha \circ \gamma = \lambda a.\ \bot_{\mathcal{A}} \ \sqsubseteq_{\mathcal{A}} \ Id_A$
- ist nutzlos für praktische Anwendungen.

Beispiel: Galois-Verbindung (1)

Bezeichne

- $IV =_{df} \{[m, n] \mid m, n \in \mathbb{Z}^{\infty}_{-\infty}, m \leq n\} \dot{\cup} \{[\,]\}$ die Menge der Intervalle über der Menge ganzer Zahlen

$$\mathbb{Z}^{\infty}_{-\infty} =_{df} \mathbb{Z} \dot{\cup} \{-\infty, \infty\}$$

wobei Intervalle Teilmengen von $\mathbb{Z}^{\infty}_{-\infty}$ bezeichnen:

$$\begin{array}{lll} [\,] & =_{df} & \emptyset \\ \forall\, m \leq n \in \mathbb{Z}^{\infty}_{-\infty}.\ [m, n] & =_{df} & \{z \mid -\infty \leq m \leq z \leq n \leq \infty\} \end{array}$$

- $\widehat{\mathbb{Z}^{\infty}_{-\infty}} =_{df} (\mathbb{Z}^{\infty}_{-\infty}, \leq, \sqcap_{\widehat{\mathbb{Z}^{\infty}_{-\infty}}}, \sqcup_{\widehat{\mathbb{Z}^{\infty}_{-\infty}}}, -\infty, \infty)$ den durch die (in natürlicher Weise auf $\mathbb{Z}^{\infty}_{-\infty}$ erweiterte) kleiner/gleich-Relation geordneten vollständigen Verband $\widehat{\mathbb{Z}^{\infty}_{-\infty}}$.

Beispiel: Galois-Verbindung (2)

Seien $\mathcal{C}$ und $\mathcal{A}$ die vollständigen (Vereinigungs-) Verbände:

- $\mathcal{C} =_{df} (\mathcal{P}(\mathbb{Z}), \cup, \subseteq, \emptyset, \mathbb{Z})$ der Potenzmengenverband ganzer Zahlen.
- $\mathcal{A} =_{df} (IV, \sqcup, \sqsubseteq, [\,], [-\infty, \infty])$ der Intervallverband ganzer Zahlen mit:

$$\forall\, [m,n], [p,q] \in IV.$$

$$[\,] \sqsubseteq [\,]$$

$$[\,] \sqsubseteq [p,q]$$

$$[m,n] \sqsubseteq [-\infty, \infty]$$

$$[m,n] \sqsubseteq [p,q] \iff_{df} [m,n] \subseteq [p,q]$$

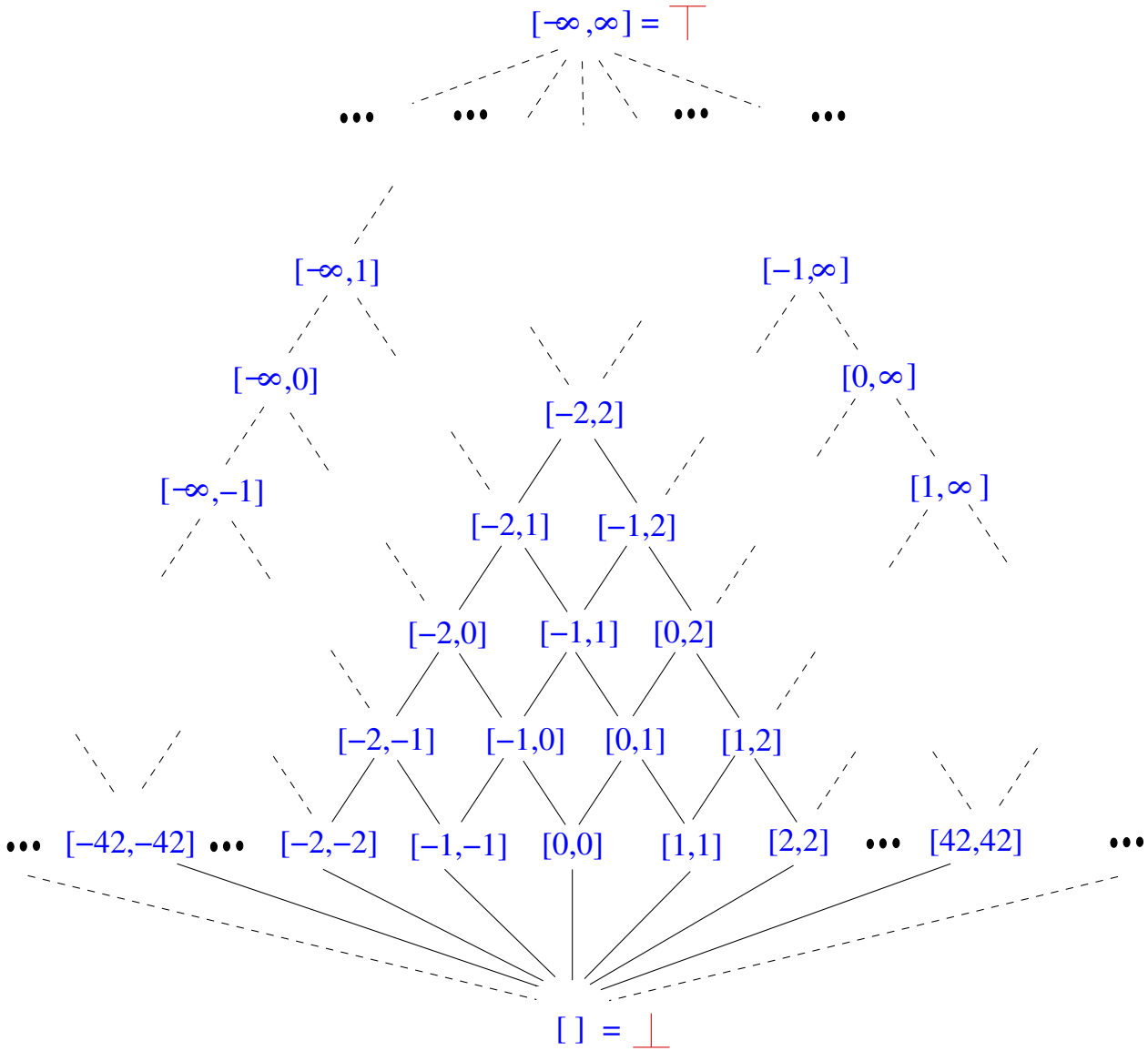
$$\iff p \leq m \leq n \leq q$$

$$[m,n] \sqcup [p,q] =_{df} [\bigsqcap_{\widehat{\mathbb{Z}^{\infty}_{-\infty}}}([m,n] \cup [p,q]),$$

$$\bigsqcup_{\widehat{\mathbb{Z}^{\infty}_{-\infty}}}([m,n] \cup [p,q])\,]$$

Beispiel: Galois-Verbindung (3)

Der Intervallverband $\mathcal{A}$:



Beispiel: Galois-Verbindung (4)

Proposition 18.2.1.3

Das Quadrupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ mit $\alpha : \mathcal{C} \to \mathcal{A}$ und $\gamma : \mathcal{A} \to \mathcal{C}$ definiert durch:

$$\forall\, Z \in \mathcal{P}(\mathbb{Z}).\ \alpha(Z) =_{df} \begin{cases} [\,] & \text{falls } Z = \emptyset \\ [\bigsqcap_{\widehat{\mathbf{Z}^{\infty}_{-\infty}}} Z, \bigsqcup_{\widehat{\mathbf{Z}^{\infty}_{-\infty}}} Z\,] & \text{sonst} \end{cases}$$

$$\forall\, iv \in IV.\ \gamma(iv) =_{df} \begin{cases} \emptyset & \text{falls } iv = [\,] \\ \{z \in \mathbb{Z}^{\infty}_{-\infty} \mid m \leq z \leq n\} & \text{falls } iv = [m, n] \end{cases}$$

ist eine Galois-Verbindung.

Eigenschaften von Galois-Verbindungen (1)

...keine weiteren Informationsverluste oder -gewinne durch fortgesetzte Abstraktionen und Konkretisierungen, Neutralität.

Lemma 18.2.1.4 (Neutralität)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung. Dann gilt:

1. $\alpha \circ \gamma \circ \alpha = \alpha$
2. $\gamma \circ \alpha \circ \gamma = \gamma$

Eigenschaften von Galois-Verbindungen (2)

...Abstraktions- und Konkretisierungsfunktionen bestimmen einander eindeutig in Galois-Verbindungen.

Lemma 18.2.1.5 (Eindeutigkeit von α und γ)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung. Dann gilt:

1. γ ist durch α eindeutig bestimmt und es gilt:
   $\forall\, a \in A.\ \gamma(a) = \bigsqcup_C \{c \in C \mid \alpha(c) \sqsubseteq_A a\}$.
2. α ist durch γ eindeutig bestimmt und es gilt:
   $\forall\, c \in C.\ \alpha(c) = \bigsqcap_A \{a \in A \mid c \sqsubseteq_C \gamma(a)\}$.
3. α ist additiv und γ distributiv.

   Insbesondere gilt: $\alpha(\bot_C) = \bot_A$ und $\gamma(\top_A) = \top_C$.

...vergleiche die Charakterisierungen von γ und α in Lemma 18.2.1.5(1)/(2) mit der Festlegung reverser lokaler Semantikfunktionen in Definition 9.3.4.

Eigenschaften von Galois-Verbindungen (3)

...Additivität bzw. Distributivität von Abstraktions- und Konkretisierungsfunktion garantieren einander wechselweise Existenz und eindeutige Bestimmtheit.

Lemma 18.2.1.6 (Existenz, eindeutige Vervollst.)

1. Ist $\alpha : C \to A$ additiv, dann gibt es ein $\gamma : A \to C$, so dass $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung ist.
2. Ist $\gamma : A \to C$ distributiv, dann gibt es ein $\alpha : C \to A$, so dass $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung ist.

Beweis

- Für 1): Setze $\forall\, a \in A.\ \gamma(a) = \bigsqcup_{\mathcal{C}} \{c \in C \mid \alpha(c) \sqsubseteq_A a\}$.
- Für 2): Setze $\forall\, c \in C.\ \alpha(c) = \bigsqcap_{\mathcal{A}} \{a \in A \mid c \sqsubseteq_C \gamma(a)\}$.

Eigenschaften von Galois-Verbindungen (4)

Proposition 18.2.1.7

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung und

$$A_\alpha =_{df} \{\alpha(c) \mid c \in \mathcal{C}\} \subseteq A$$

Dann ist

$$\mathcal{A}_\alpha =_{df} (A_\alpha, \sqcup_{\mathcal{A}_{|\mathcal{A}_\alpha}}, \sqsubseteq_{\mathcal{A}_{|\mathcal{A}_\alpha}}, \bot_A, \top_A)$$

ein vollständiger (Vereinigungs-) Halbverband, wobei $\sqcup_{\mathcal{A}_{|\mathcal{A}_\alpha}}$ und $\sqsubseteq_{\mathcal{A}_{|\mathcal{A}_\alpha}}$ die Einschränkungen von $\sqcup_{\mathcal{A}}$ und $\sqsubseteq_{\mathcal{A}}$ von A auf A_α bezeichnen.

Eigenschaften von Galois-Verbindungen (5)

Proposition 18.2.1.8

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung und $\rho : A \to A$ der wie folgt definierte Reduktionsoperator:

$$\forall\, a \in A.\ \rho(a) =_{df} \bigsqcap\nolimits_{\mathcal{A}} \{a' \in A \mid \gamma(a) = \gamma(a')\}$$

und $R_\rho =_{df} \{\rho(a) \mid a \in A\}$ $R_\rho =_{df} \{\rho(a) \mid a \in A\}$.

Dann gilt:

1. $\mathcal{R}_\rho =_{df} (R_\rho, \sqcup_{\mathcal{A}|\mathcal{R}_\rho}, \sqsubseteq_{\mathcal{A}|\mathcal{R}_\rho}, \bot_{\mathcal{A}}, \top_{\mathcal{A}})$ ist ein vollständiger Verband.
2. $\forall\, a \in A.\ \rho(a) = \alpha(\gamma(a))$
3. $\mathcal{A}_\alpha = \mathcal{R}_\rho$

...siehe auch Lemma 18.2.2.4.

Adjunktionscharakterisierung v. Galois-Verbind.

...mit der Adjunktionscharaktisierung von Galois-Verbindungen (Lemma 18.2.1.10) ist oft einfacher zu arbeiten als mit ihrer unmittelbaren Definition (Definition 18.2.1.1); siehe dazu auch Übungsaufgabe 18.3.2.7.

Definition 18.2.1.9 (Adjunktion)

Ein Quadrupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ heißt Adjunktion (engl. adjunction) gdw:

1. α und γ sind total
2. $\forall\, c \in C\ \forall\, a \in A.\ \alpha(c) \sqsubseteq_{\mathcal{A}} a \iff c \sqsubseteq_{\mathcal{C}} \gamma(a)$

Lemma 18.2.1.10 (Charakterisierung v. Galois-Verb.)

$(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ ist eine Adjunktion gdw $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ ist eine Galois-Verbindung.

Kapitel 18.2.2

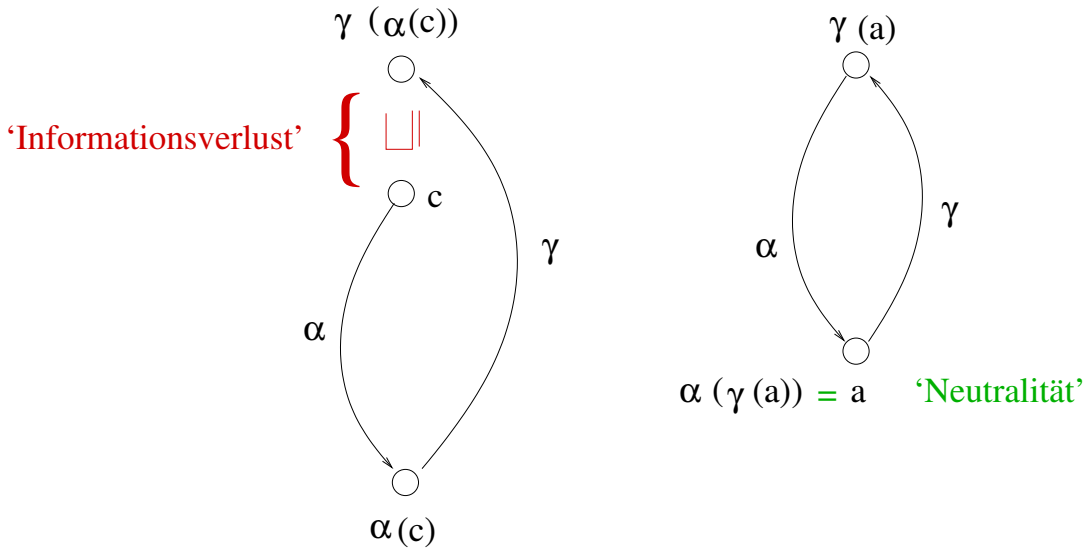
Galois-Passungen

Galois-Passungen

Definition 18.2.2.1 (Galois-Passung)

Ein Quadrupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ heißt Galois-Passung (engl. Galois insertion) gdw:

1. α und γ sind monoton
2. $\gamma \circ \alpha \sqsupseteq_C Id_C$ ($\sqsupseteq_{\mathcal{C}}$: linksseitig 'Informationsverlust')
3. $\alpha \circ \gamma =_{\mathcal{A}} Id_A$ ($=_{\mathcal{A}}$: linksseitig 'Neutralität')



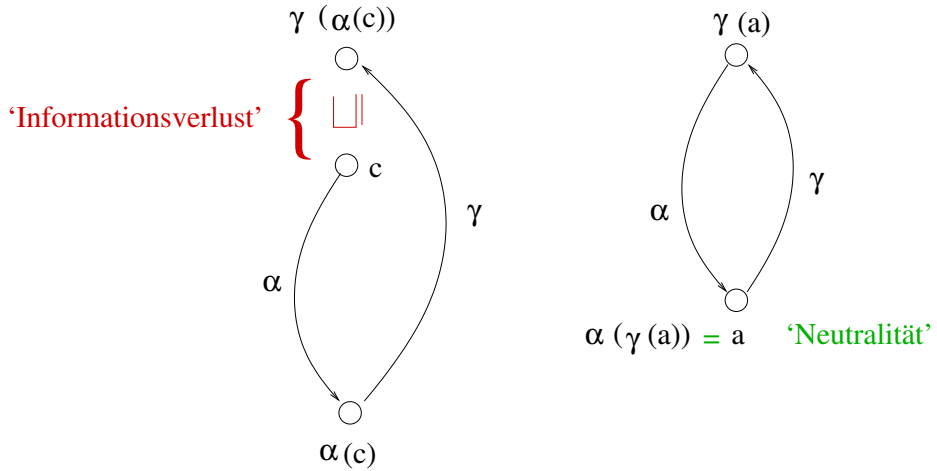
Informell

...wie für Galois-Verbindungen:

- Abstraktion kann schaden: Ist die Inklusion $\gamma \circ \alpha \sqsupseteq_{\mathcal{C}} Id_C$ in Definition 18.2.2.1(1) echt, so bedeutet das eine Schlechterstellung, einen Informationsverlust auf der konkreten Ebene.

Anders als für Galois-Verbindungen:

- Konkretisierung darf nicht schaden, aber auch nicht zu einer Besserstellung, einem Informationsgewinn auf der abstrakten Ebene führen: $\alpha \circ \gamma =_{\mathcal{A}} Id_A$.

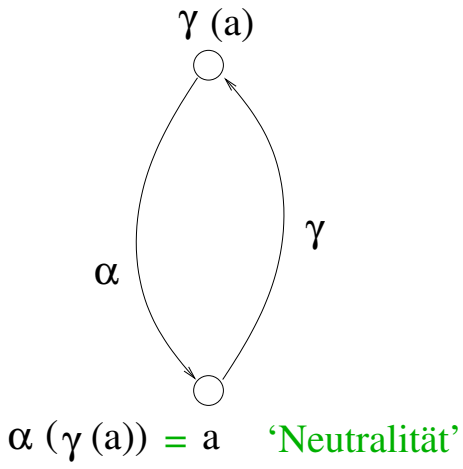


Galois-Passungen: Spezielle Galois-Verbind.

Proposition 18.2.2.2

Eine Galois-Verbindung $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ ist eine Galois-Passung gdw:

$$\alpha \circ \gamma =_{\mathcal{A}} Id_{\mathcal{A}}$$



Bem.: Die triviale, nutzlose Galois-Verbindung aus Proposition 18.2.1.2 ist keine Galois-Passung.

Charakterisierung von Galois-Passungen

Lemma 18.2.2.3 (Äquivalenzaussagen)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung. Dann sind folgende Aussagen äquivalent:

1. $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ ist eine Galois-Passung.
2. α ist surjektiv, d.h.: $\forall\, a \in A\ \exists\, c \in C.\ \alpha(c) = a$.
3. γ ist injektiv, d.h.:
   $\forall\, a_1, a_2 \in A.\ \gamma(a_1) = \gamma(a_2) \ \Rightarrow\ a_1 = a_2$.
4. γ ist ordnungsähnlich, d.h.:
   $\forall\, a_1, a_2 \in A.\ \gamma(a_1) \sqsubseteq_C \gamma(a_2) \ \Leftrightarrow\ a_1 \sqsubseteq_A a_2$.

Beachte: Die Rückrichtungsimplikation in 4)

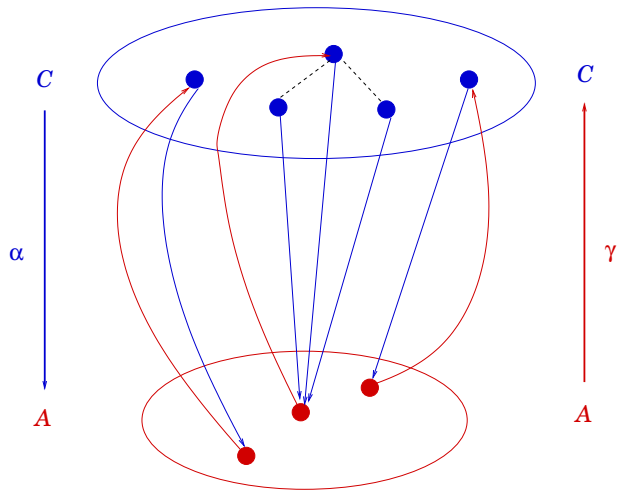
$$\forall\, a_1, a_2 \in A.\ \gamma(a_1) \sqsubseteq_C \gamma(a_2) \ \Leftarrow\ a_1 \sqsubseteq_A a_2$$

folgt bereits aus der Galois-Verbindungseigenschaft v. $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$.

Informell

- In einer Galois-Verbindung können mehrere Elemente aus $\mathcal{A}$ dasselbe Element aus $\mathcal{C}$ beschreiben; in einer Galois-Passung nicht.
- Aus diesem Grund ist in einer Galois-Verbindung die Konkretisierungsfunktion γ i.a. nicht injektiv, die Abstraktionsfunktion α i.a. nicht surjektiv; in einer Galois-Passung schon: γ ist stets injektiv, α stets surjektiv.
- In einer Galois-Passung kann $\mathcal{A}$ deshalb keine Elemente enthalten, die keine Elemente aus $\mathcal{C}$ beschreiben, d.h. $\mathcal{A}$ enthält in diesem Sinn keine überflüssigen Elemente.

Galois-Passung: α surjektiv, γ injektiv.



Konstruktion von Galois-Passungen

Lemma 18.2.2.4 (Galois-Passungskonstruktion)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung, $\rho : A \to A$ der wie folgt definierte Reduktionsoperator:

$$\forall\, a \in A.\ \rho(a) =_{df} \bigsqcap\nolimits_A \{a' \in A \mid \gamma(a) = \gamma(a')\}$$

und $R_\rho =_{df} \{\rho(a) \mid a \in A\}$.

Dann gilt:

1. $\mathcal{R}_\rho =_{df} (R_\rho, \sqcup_{\mathcal{A}|\mathcal{R}_\rho}, \sqsubseteq_{\mathcal{A}|\mathcal{R}_\rho}, \bot_{\mathcal{A}}, \top_{\mathcal{A}})$ ist ein vollständiger Verband.
2. Das Quadrupel $(\mathcal{C}, \alpha, \gamma, \mathcal{R}_\rho)$ ist eine Galois-Passung.

Beweis mit Proposition 18.2.1.6 und 18.2.1.7 und Lemma 18.2.1.9 und 18.2.2.3.

Übungsaufgabe 18.2.2.5

Ist die Galois-Verbindung $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ aus Proposition 18.2.1.2 zur Intervallanalyse eine Galois-Passung? Beweis oder Gegenbeispiel.

Kapitel 18.3

Systematische Konstruktion von Galois-Verbindungen

Übersicht

...wir betrachten acht Methoden, aufgeteilt in zwei Gruppen:

- Aus dem 'Nichts' erschaffende Methoden

  ...zur Konstruktion neuer Galois-Verbindungen ohne bereits gegebene Galois-Verbindungen.

- Kombinationsmethoden

  ...zur Konstruktion neuer Galois-Verbindungen aus gegebenen Galois-Verbindungen.

Im einzelnen

Erschaffende Methoden

1. Extraktionsmethode
2. Spezialfall der Extraktionsmethode

Kombinierende Methoden

1. Unabhängige Attributemethode
2. Relationale Methode
3. Totale Funktionenraummethode
4. Monotone Funktionenraummethode
5. Direkte Produktmethode
6. Direkte Tensorproduktmethode

Kapitel 18.3.1

Erschaffende Methoden

1) Extraktionsmethode

Lemma 18.3.1.1 (Extraktionsmethode)

Sei $\beta : \mathbf{V} \to \mathcal{C}$ eine Abbildung von der Menge der Variablen $\mathbf{V}$ in einen Verband $\mathcal{C} = (C, \sqcup_{\mathcal{C}}, \sqsubseteq_{\mathcal{C}}, \bot_{\mathcal{C}}, \top_{\mathcal{C}})$.

Dann ist das Quadrupel $(\mathcal{P}(\mathbf{V}), \alpha, \gamma, \mathcal{C})$ mit

$$\begin{aligned} \forall\, V \in \mathcal{P}(\mathbf{V}).\ \alpha(V) &=_{df} \bigsqcup\nolimits_{\mathcal{C}} \{\beta(v) \mid v \in V\} \\ \forall\, c \in C.\ \gamma(c) &=_{df} \{v \in \mathbf{V} \mid \beta(v) \sqsubseteq_{\mathcal{C}} c\} \end{aligned}$$

eine Galois-Verbindung.

2) Spezialfall der Extraktionsmethode

Lemma 18.3.1.2 (Spezialfall d. Extraktionsmethode)

Sei D eine Menge, $\mathcal{C} = (\mathcal{P}(D), \cup, \subseteq, \emptyset, D)$ der Potenzmengenverband von D, $\eta : \mathbf{V} \to D$ eine Extraktionsfunktion von der Menge der Variablen $\mathbf{V}$ in D und $\beta_\eta : \mathbf{V} \to \mathcal{C}$ eine Abbildung mit

$$\forall\, v \in \mathbf{V}.\ \beta_\eta(v) =_{df} \{\eta(v)\}$$

Dann ist das Quadrupel $(\mathcal{P}(\mathbf{V}), \alpha_\eta, \gamma_\eta, \mathcal{C})$ mit

$$\begin{array}{rcl} \forall\, V \in \mathcal{P}(\mathbf{V}).\ \alpha_\eta(V) & =_{df} & \bigcup\{\beta_\eta(v) \mid v \in V\} \\ & = & \{\eta(c) \mid v \in V\} \\ \forall\, D' \in \mathcal{P}(D).\ \gamma_\eta(D') & =_{df} & \{v \in \mathbf{V} \mid \beta_\eta(v) \in D'\} \\ & = & \{v \mid \eta(v) \in D'\}) \end{array}$$

eine Galois-Verbindung.

Übungsaufgabe 18.3.1.3

Beweise:

1. Lemma 18.3.1.1
2. Lemma 18.3.1.2

Kapitel 18.3.2

Kombinierende Methoden

1) Unabhängige Attributemethode

Lemma 18.3.2.1 (Unabhängige Attributemethode)

Seien $(\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1)$ und $(\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2)$ zwei Galois-Verbindungen.

Dann ist auch $(\mathcal{C}_1 \times \mathcal{C}_2, \alpha, \gamma, \mathcal{A}_1 \times \mathcal{A}_2)$ mit

$$\begin{aligned} \forall (c_1, c_2) \in C_1 \times C_2.\ \alpha(c_1, c_2) &=_{df} (\alpha_1(c_1), \alpha_2(c_2)) \\ \forall (a_1, a_2) \in A_1 \times A_2.\ \gamma(a_1, a_2) &=_{df} (\gamma_1(a_1), \gamma_2(a_2)) \end{aligned}$$

eine Galois-Verbindung.

2) Relationale Methode

...bezeichne $\mathcal{P}$ den Potenzmengenoperator.

Lemma 18.3.2.2 (Relationale Methode)

Seien $(\mathcal{P}(C_1), \alpha_1, \gamma_1, \mathcal{P}(A_1))$ und $(\mathcal{P}(C_2), \alpha_2, \gamma_2, \mathcal{P}(A_2))$ zwei Galois-Verbindungen.

Dann ist auch $(\mathcal{P}(C_1 \times C_2), \alpha, \gamma, \mathcal{P}(A_1 \times A_2))$ mit

$$\begin{aligned} \alpha(CC) &=_{df} \bigcup \{\alpha_1(\{c_1\}) \times \alpha_2(\{c_2\}) \,|\, (c_1, c_2) \in CC\} \\ \gamma(AA) &=_{df} \{(c_1, c_2) \,|\, \alpha_1(\{c_1\}) \times \alpha_2(\{c_2\}) \subseteq AA\} \end{aligned}$$

für $CC \subseteq C_1 \times C_2$ und $AA \subseteq A_1 \times A_2$ eine Galois-Verbindung.

3) Totale Funktionenraummethode

...bezeichne $[P \overset{tot}{\to} Q]$, P, Q Mengen, die Menge der total definierten Funktionen von P nach Q.

Lemma 18.3.2.3 (Totale Funktionenraummethode)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung und M eine Menge.

Dann ist auch $([M \overset{tot}{\to} C], \alpha', \gamma', [M \overset{tot}{\to} A])$ mit

$$\forall f \in [M \overset{tot}{\to} C].\ \alpha'(f) =_{df} \alpha \circ f$$
$$\forall g \in [M \overset{tot}{\to} A].\ \gamma'(g) =_{df} \gamma \circ g$$

eine Galois-Verbindung.

4) Monotone Funktionraummethode

...bezeichne $[P \overset{mon}{\to} Q]$, P, Q Mengen, die Menge der monotonen Funktionen von P nach Q.

Lemma 18.3.2.4 (Monotone Funktionenraummeth.)

Seien $(\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1)$ und $(\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2)$ zwei Galois-Verbindungen.

Dann ist auch $([C_1 \overset{mon}{\to} C_2], \alpha, \gamma, [A_1 \overset{mon}{\to} A_2])$ mit

$$\forall f \in [C_1 \overset{mon}{\to} C_2].\ \alpha(f) \quad =_{df} \quad \alpha_2 \circ f \circ \gamma_1$$
$$\forall g \in [A_1 \overset{mon}{\to} A_2].\ \gamma(g) \quad =_{df} \quad \gamma_2 \circ g \circ \alpha_1$$

eine Galois-Verbindung.

5) Direkte Produktmethode

Lemma 18.3.2.5 (Direkte Produktmethode)

Seien $(\mathcal{C}, \alpha_1, \gamma_1, \mathcal{A}_1)$ und $(\mathcal{C}, \alpha_2, \gamma_2, \mathcal{A}_2)$ zwei Galois-Verbindungen.

Dann ist auch $(\mathcal{C}, \alpha, \gamma, \mathcal{A}_1 \times \mathcal{A}_2)$ mit

$$\begin{aligned} \forall\, c \in C.\ \alpha(c) &=_{df} (\alpha_1(c), \alpha_2(c)) \\ \forall\, (a_1, a_2) \in A_1 \times A_2.\ \gamma(a_1, a_2) &=_{df} \gamma_1(a_1) \sqcap \gamma_2(a_2) \end{aligned}$$

eine Galois-Verbindung.

(Beachte die Abstützung von γ auf $\sqcap$, nicht $\sqcup$; s.a. Übungsaufgabe 18.3.2.7).

6) Direkte Tensorproduktmethode

...bezeichne $\mathcal{P}$ den Potenzmengenoperator.

Lemma 18.3.2.6 (Direkte Tensorproduktmethode)

Seien $(\mathcal{P}(C), \alpha_1, \gamma_1, \mathcal{P}(A_1))$ und $(\mathcal{P}(C), \alpha_2, \gamma_2, \mathcal{P}(A_2))$ zwei Galois-Verbindungen.

Dann ist auch $(\mathcal{P}(C), \alpha, \gamma, \mathcal{P}(A_1 \times \mathcal{A}_2))$ mit

$$\forall\, C' \in \mathcal{P}(C).\ \alpha(C') =_{df} \bigcup \{\alpha_1(\{c\}) \times \alpha_2(\{c\}) \mid c \in C'\}$$
$$\forall\, AA \in \mathcal{P}(A_1 \times A_2).\ \gamma(AA) =_{df} \{c \mid \alpha_1(\{c\}) \times \alpha_2(\{c\}) \subseteq AA\}$$

eine Galois-Verbindung.

Übungsaufgabe 18.3.2.7

Beweise Lemma 18.3.2.5, d.h. sind $(\mathcal{C}, \alpha_1, \gamma_1, \mathcal{A}_1)$ und $(\mathcal{C}, \alpha_2, \gamma_2, \mathcal{A}_2)$ Galois-Verbindungen, dann ist auch $(\mathcal{C}, \alpha, \gamma, \mathcal{A}_1 \times \mathcal{A}_2)$ mit α und γ definiert durch:

$$\begin{aligned} \forall\, c \in C.\ \alpha(c) &=_{df} (\alpha_1(c), \alpha_2(c)) \\ \forall\, (a_1, a_2) \in A_1 \times A_2.\ \gamma(a_1, a_2) &=_{df} \gamma_1(a_1) \sqcap \gamma_2(a_2) \end{aligned}$$

eine Galois-Verbindung.

Zeige dazu:

$$\alpha(c) \sqsubseteq (a_1, a_2) \Longleftrightarrow c \sqsubseteq \gamma(a_1, a_2)$$

woraus mithilfe von Lemma 18.2.1.10 (Adjunktionscharakterisierung von Galois-Verbindungen) die Behauptung folgt.

Übungsaufgabe 18.3.2.8

Beweise die übrigen Lemmata aus Kapitel 18.3.2, d.h. beweise:

1. Lemma 18.3.2.1
2. Lemma 18.3.2.2
3. Lemma 18.3.2.3
4. Lemma 18.3.2.4
5. Lemma 18.3.2.6

Kapitel 18.4

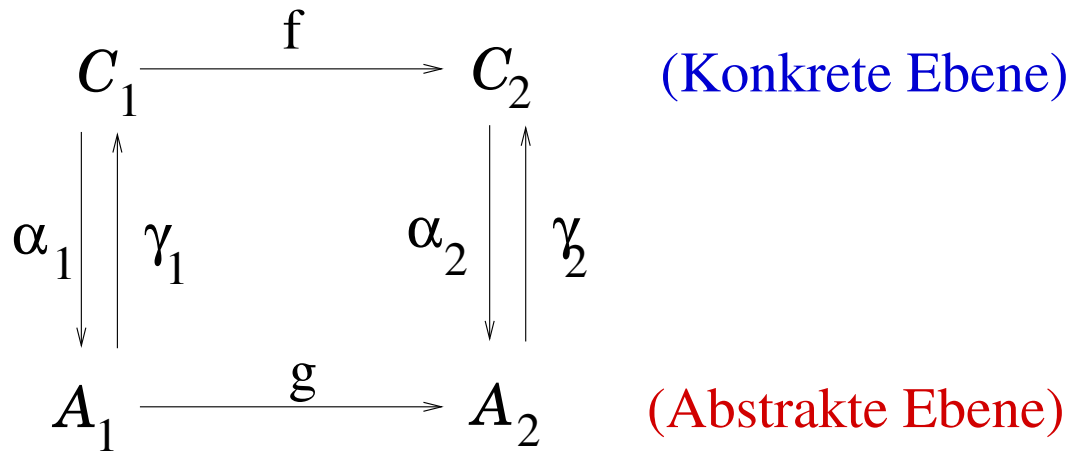
Galois-Systeme

Galois-Systeme

Definition 18.4.1 (Galois-Systeme)

Seien $(\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1)$ und $(\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2)$ zwei Galois-Verbindungen und seien $f : C_1 \to C_2$ und $g : A_1 \to A_2$ zwei monotone Abbildungen.

Dann heißt das Tupel $(f, g, (\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1), (\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2))$ ein Galois-System, das folgende Situation beschreibt:



Charakterisierung von Galois-Systemen

Lemma 18.4.2 (Charakterisierung)

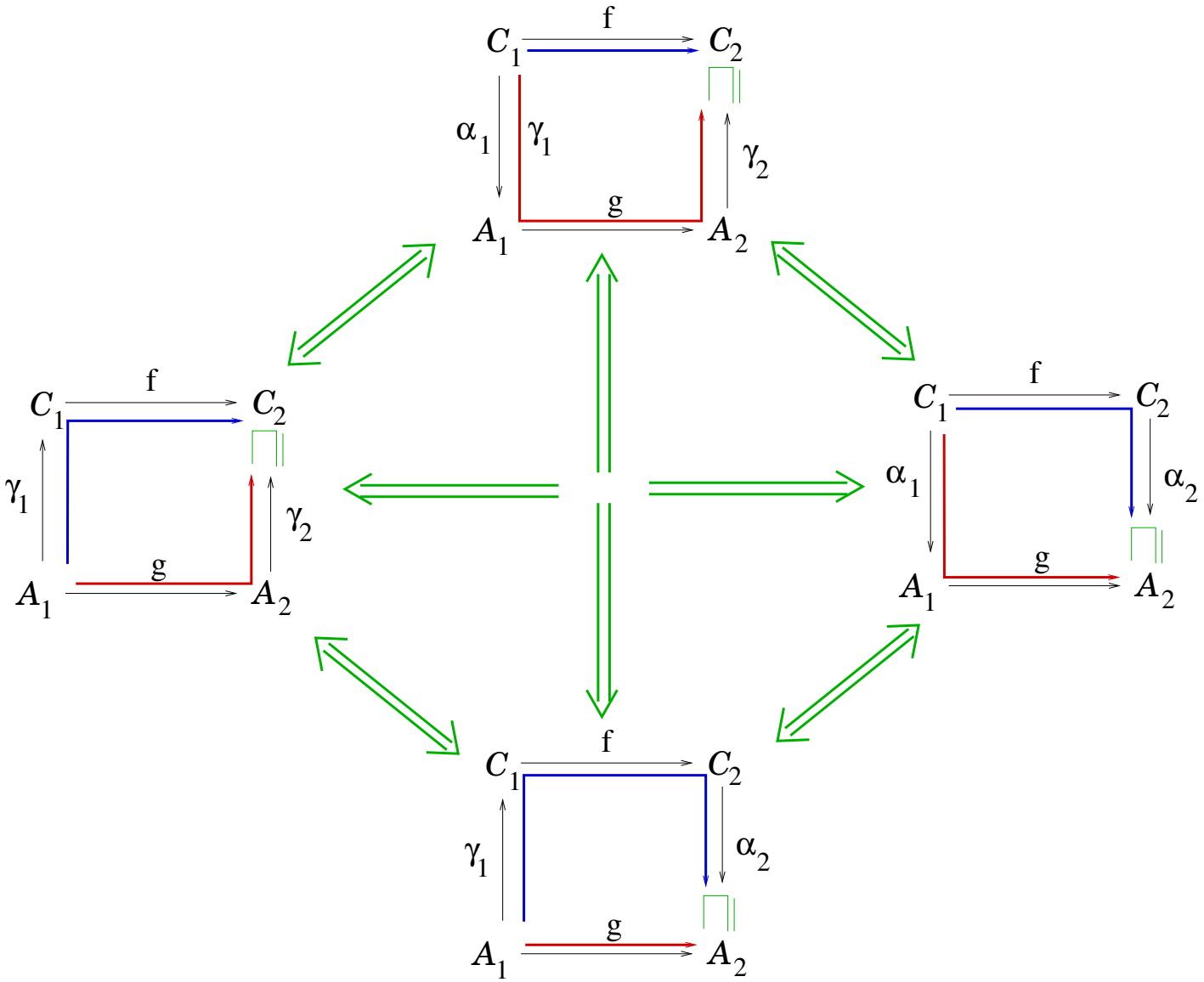
Sei $(f, g, (\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1), (\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2))$ ein Galois-System. Dann sind folgende Aussagen äquivalent:

1. $f \sqsubseteq_{\mathcal{C}_2} \gamma_2 \circ g \circ \alpha_1$
2. $\alpha_2 \circ f \sqsubseteq_{\mathcal{A}_2} g \circ \alpha_1$
3. $\alpha_2 \circ f \circ \gamma_1 \sqsubseteq_{\mathcal{A}_2} g$
4. $f \circ \gamma_1 \sqsubseteq_{\mathcal{C}_2} \gamma_2 \circ g$

Lemma 18.4.3 (Charakterisierung)

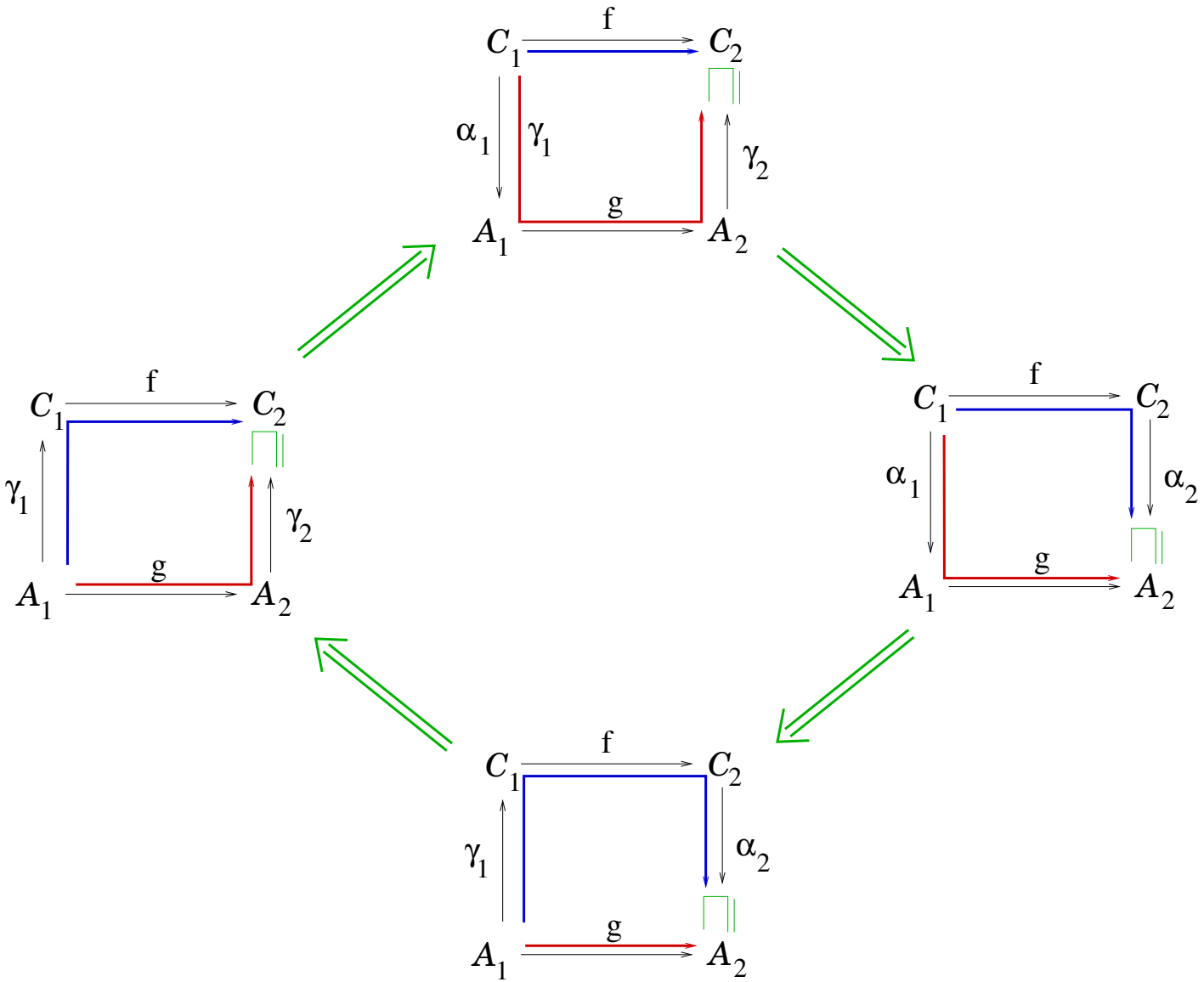
Lemma 18.4.2 gilt analog, wenn überall $\sqsubseteq$ durch $=$ ersetzt wird.

Illustration der Aussage von Lemma 18.4.2



Übungsaufgabe 18.4.4

Beweise Lemma 18.4.2 und 18.4.3 unter Ausnutzung folgender Idee:



Kapitel 18.5

Systeme abstrakter Interpretationen

Idee

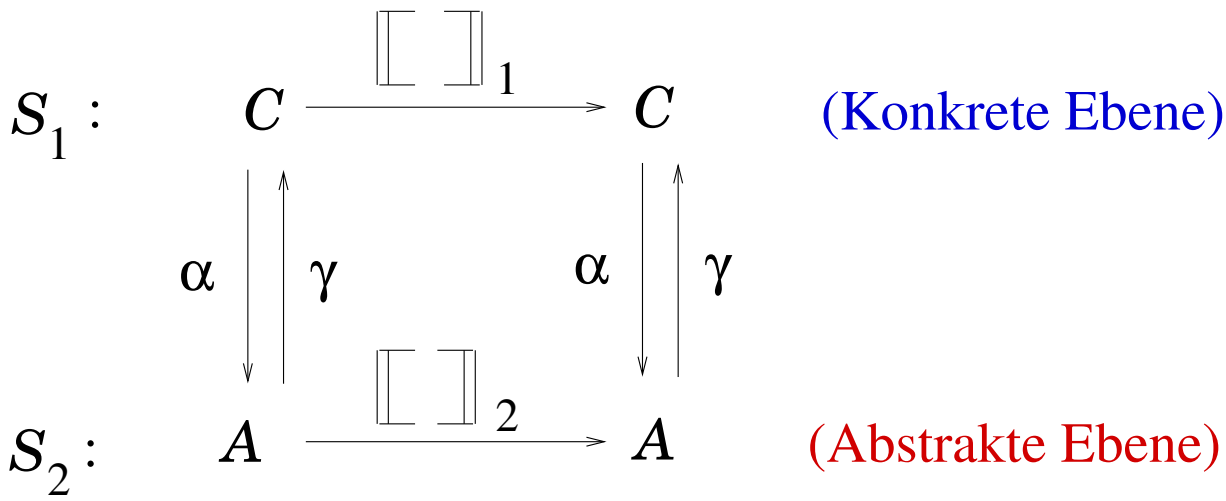
...ersetzen wir in einem Galois-System

$$(f, g, (\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1), (\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2))$$

die Abbildungen f und g durch abstrakte lokale Semantiken $[\![\ ]\!]_1 : \mathcal{C} \to \mathcal{C}$ und $[\![\ ]\!]_2 : \mathcal{A} \to \mathcal{A}$, so erhalten wir ein System abstrakter Interpretationen

$$([\![\ ]\!]_1, [\![\ ]\!]_2, (\mathcal{C}, \alpha, \gamma, \mathcal{A}))$$

das folgende (einfachere) Situation beschreibt:

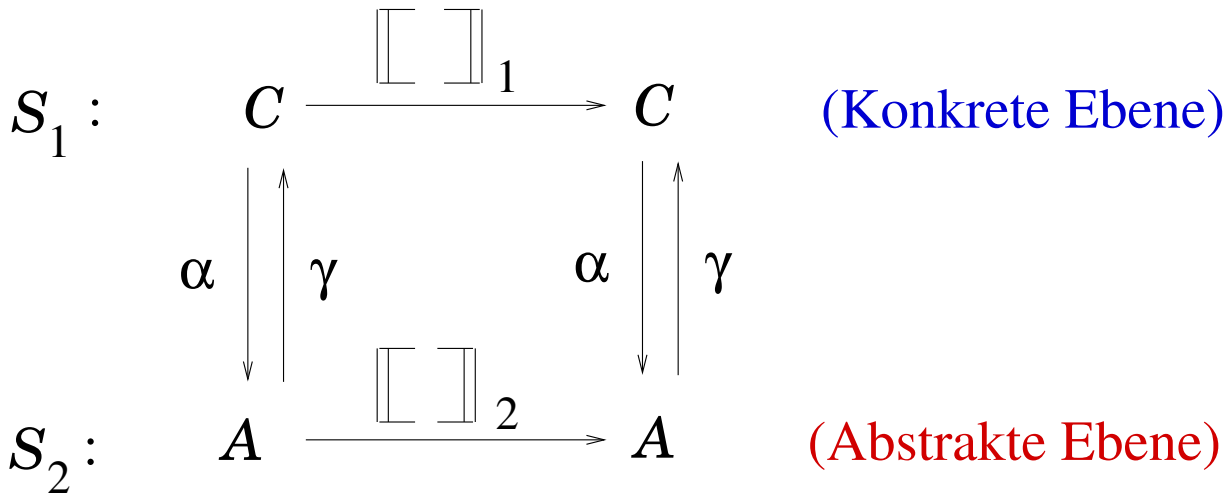


Systeme abstrakter Interpretationen

Definition 18.5.1 (System abstr. Interpretationen)

Sei $(\mathcal{C}, \alpha_1, \gamma_1, \mathcal{A})$ eine Galois-Verbindung und $[\![\ ]\!]_1 : \mathcal{C} \to \mathcal{C}$ und $[\![\ ]\!]_2 : \mathcal{A} \to \mathcal{A}$ zwei abstrakte lokale Semantiken.

Dann heißt das Tupel $([\![\ ]\!]_1, [\![\ ]\!]_2, (\mathcal{C}, \alpha, \gamma, \mathcal{A}))$ ein System abstrakter Interpretationen, das folgende Situation beschreibt:



Datenflussanalyse und abstrakte Interpretation

...ist $(\llbracket\ \rrbracket_1, \llbracket\ \rrbracket_2, (\mathcal{C}, \alpha, \gamma, \mathcal{A}))$ ein System abstrakter Interpretationen, so spezifizieren $\mathcal{S}_1 = (\mathcal{C}, \llbracket\ \rrbracket_1)$ und $\mathcal{S}_2 = (\mathcal{A}, \llbracket\ \rrbracket_2)$ abstrakte Programmsemantiken, z.B. im Sinn der

- Vereinigung/Schnitt-über-alle-Pfade-Semantik

die (unter geeigneten Voraussetzungen) approximativ oder akkurat als

- minimale/maximale Fixpunktsemantik

effektiv berechnet werden können (s. Kapitel 8).

Datenflussanalyse und abstrakte Interpretation

...dabei ist die abstrakte Semantik eines Programms G durch die globale abstrakte Semantik am Endknoten von G bestimmt:

$$[\![G]\!]_{\mathcal{S}}^{VUP/SUP} =_{df} [\![\mathbf{e}]\!]_{\mathcal{S}}^{VUP/SUP} \sqsubseteq_{MinFP}^{VUP} / \sqsupseteq_{MaxFP}^{SUP} [\![\mathbf{e}]\!]_{\mathcal{S}}^{MinFP/MaxFP}$$

wobei (unter geeigneten Voraussetzungen, s. Kapitel 8) gilt:

$$[\![G]\!]_{\mathcal{S}}^{VUP/SUP} = [\![\mathbf{e}]\!]_{\mathcal{S}}^{MinFP/MaxFP}$$

Diese Beobachtung verknüpft die Theorie (und Praxis) der Datenflussanalyse (s. Kapitel 7, 8) mit der Theorie (und Praxis) der abstrakten Interpretationen (s. Kapitel 18).

Entwicklung von Programmanalysen (1)

...im Rahmen von Systemen abstrakter Interpretationen erfolgt typischerweise iterativ.

Initial-Iteration:

- Wähle eine Analysespezifikation $\mathcal{S} = (\mathcal{C}, [\![\ ]\!])$ mit $\mathcal{C}$ vollständiger Verband und $[\![\ ]\!] : E \to (C \to C)$ lokale abstrakte Semantik, die noch eng mit der tatsächlichen Programmsemantik verbunden sind, z.B. die (nicht-deterministische) Aufsammelsemantik:

  $$\mathcal{S} = (\mathcal{C}, [\![\ ]\!]) =_{df} (\mathcal{P}(\Sigma_\perp^\top), [\![\ ]\!]_{\text{WHILE}})$$

  Die Analysespezifikation

  $$\mathcal{S} \ _{df}= \mathcal{S}_0 = (\mathcal{A}_0, [\![\ ]\!]_0)$$

  legt die sog. konkrete Ebene, die Referenzebene aller weiteren Analysen fest.

Entwicklung von Programmanalysen (2)

Folge-Iterationen:

- Ausgehend von $\mathcal{S}_i = (\mathcal{A}_i, [\![\ ]\!]_i)$, führe einen stärker approximativen Verband zusammen mit einer darauf abgestimmten lokalen abstrakten Semantik

$$\mathcal{S}_{i+1} = (\mathcal{A}_{i+1}, [\![\ ]\!]_{i+1})$$

und einem Paar aus Abstraktions- und Konkretisierungsfunktion

$$\alpha_{i+1} : A_i \to A_{i+1}, \quad \gamma_{i+1} : A_{i+1} \to A_i$$

ein, so dass

$$(\mathcal{A}_i, \alpha_{i+1}, \gamma_{i+1}, \mathcal{A}_{i+1})$$

eine Galois-Verbindung oder Galois-Passung bilden.

...bis ein für Analysefrage und -berechnung angemessenes und zweckmäßiges Abstraktionsniveau erreicht ist.

Turm von Galois-Verbindungen/-Passungen (1)

...durch wiederholte Anwendung des Iterationsschritts entsteht ein

- Turm (oder Hierarchie) von Systemen abstrakter Interpretationen

dessen Ebenen durch

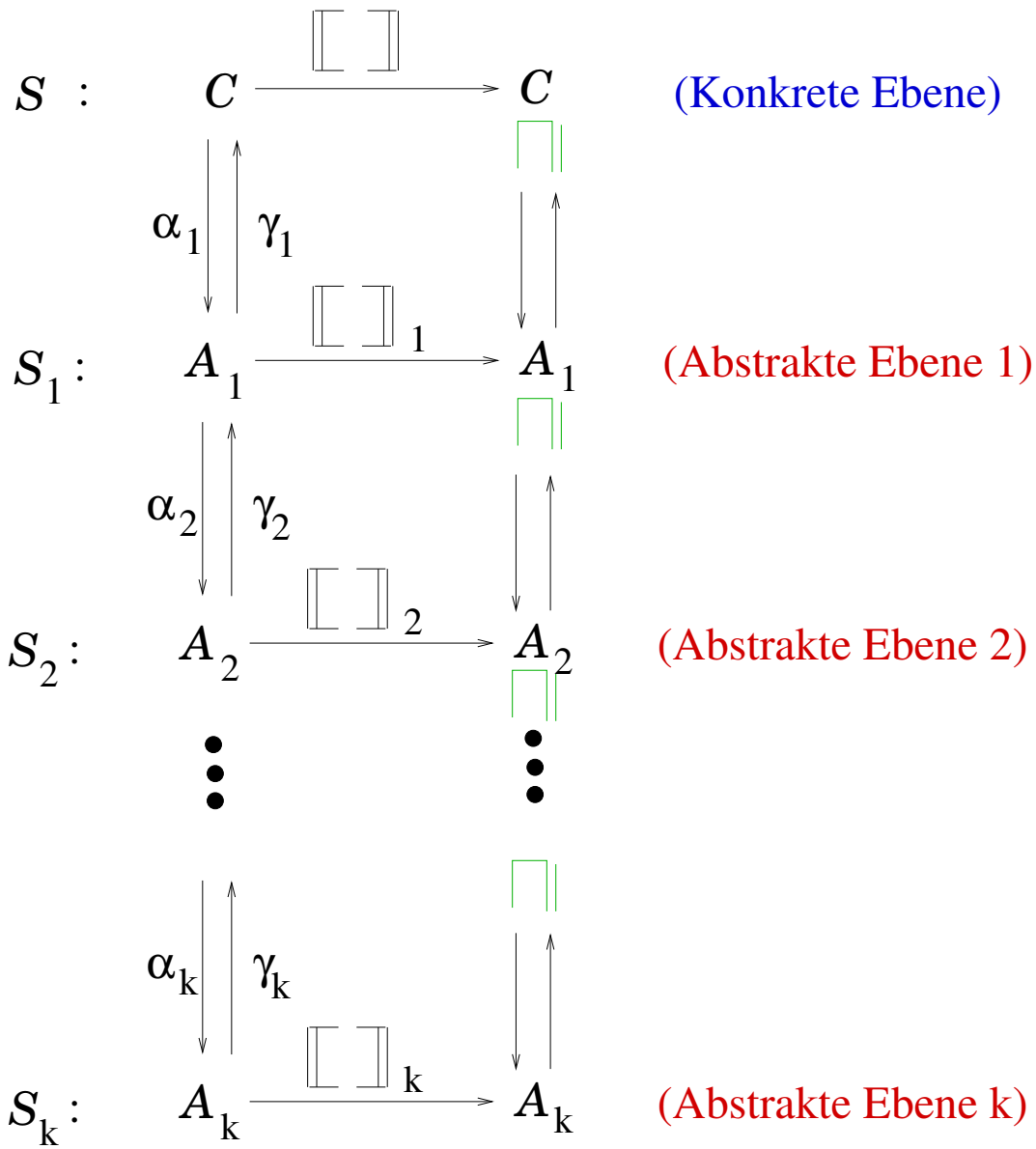
- Galois-Verbindungen oder (sogar) Galois-Passungen

verknüpft sind, so dass abstrakte Interpretationen niedrigerer Ebenen

- korrekt und (idealerweise) vollständig und optimal

bezüglich abstrakter Interpretationen höherer Ebenen sind (s. Kapitel 18.6 und 18.7).

Turm von Galois-Verbindungen/-Passungen (2)



Grundlage für Galois-Verb./-Passungstürme

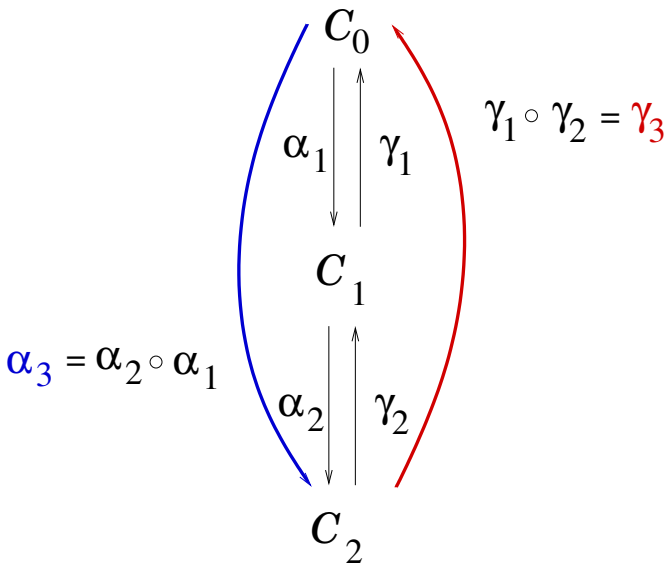
...ist die induktive Ausdehnung von:

Lemma 18.5.2 (Komposition v. Galois-Verb./Pass.)

Seien $(\mathcal{C}_0, \alpha_1, \gamma_1, \mathcal{C}_1)$ und $(\mathcal{C}_1, \alpha_2, \gamma_2, \mathcal{C}_2)$ zwei Galois-Verbindungen (Galois-Passungen). Dann ist auch

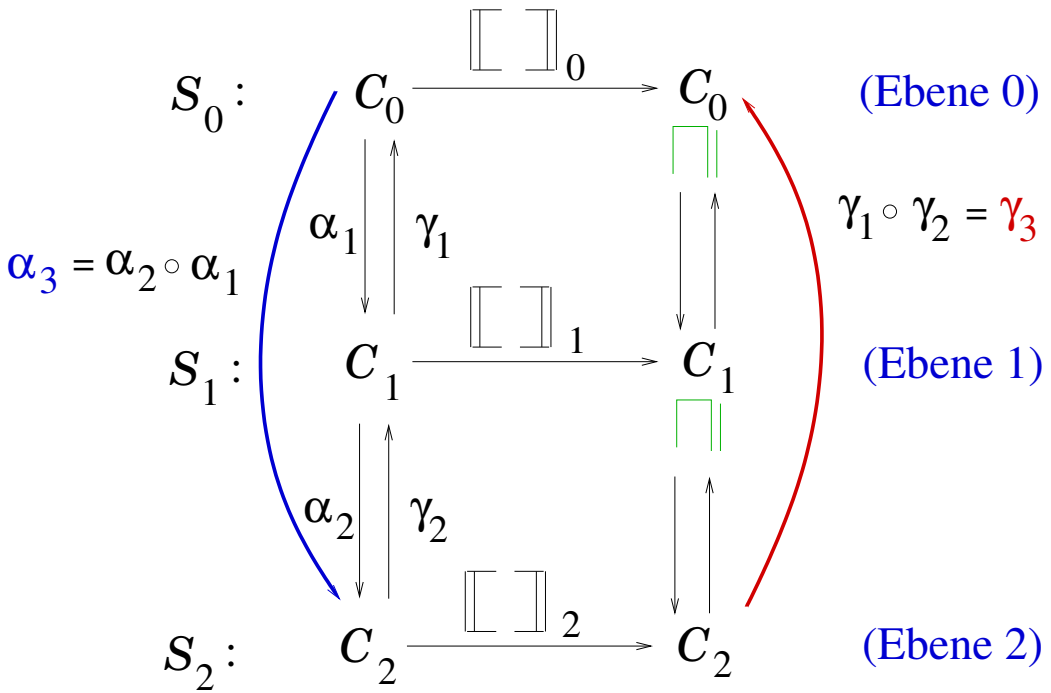
$$(\mathcal{C}_0, \alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2, \mathcal{C}_2)$$

eine Galois-Verbindung (Galois-Passung).



Anwendung

...von Lemma 18.5.2 auf Systeme abstrakter Interpretationen:



Kapitel 18.6

Korrektheit und Vollständigkeit abstrakter Interpretationen

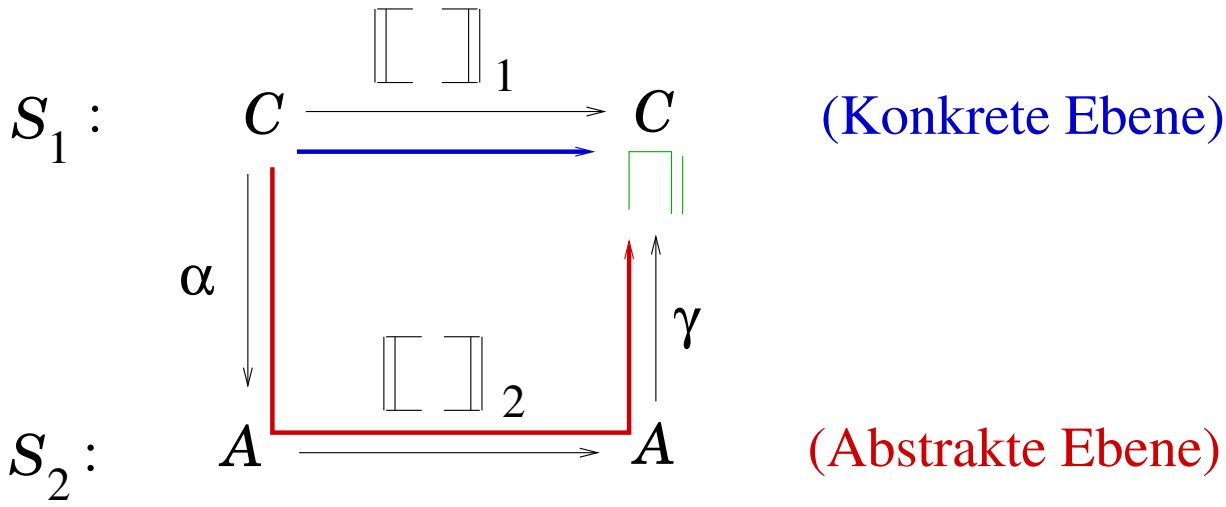
Korrektheits- und Vollständigkeitsbegriff

...in der Theorie abstrakter Interpretationen setzen an der Einbettung abstrakter Interpretationen in

- Systeme abstrakter Interpretationen an.

Informell bedeuten Korrektheit und Vollständigkeit in Systemen abstrakter Interpretationen, speziellen Galois-Systemen

- Korrektheit und Vollständigkeit einer abstrakten Interpretation einer niedrigeren Ebene relativ zu einer abstrakten Interpretation einer höheren Ebene.



Korrektheitheit und Vollständigkeit

...einer abstrakten Interpretation in einem Galois-System.

Definition 18.6.1 (Korrektheit, Vollständigkeit)

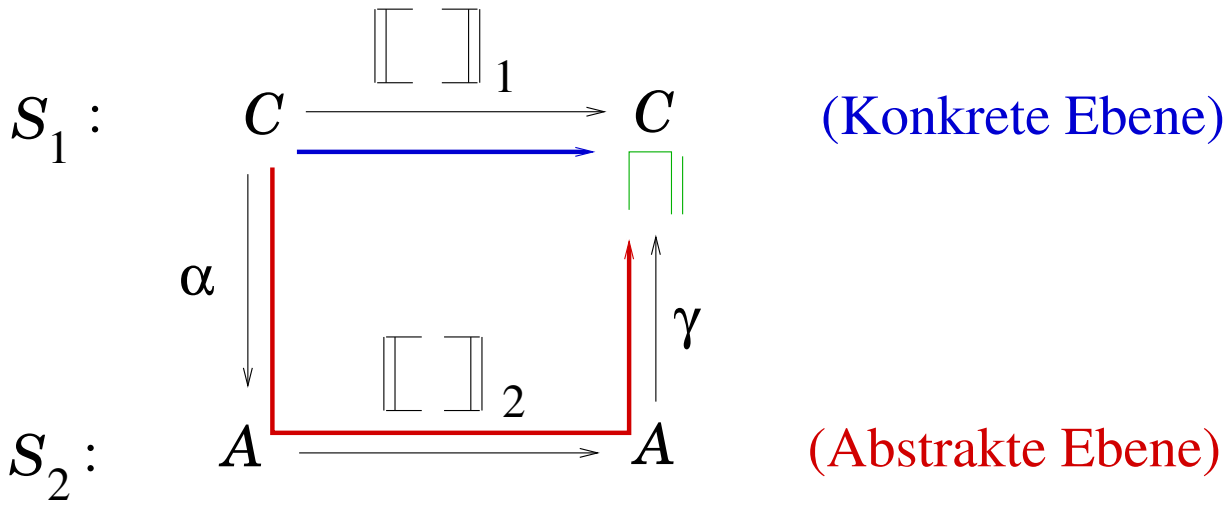
Sei $(\llbracket\ \rrbracket_1, \llbracket\ \rrbracket_2, (\mathcal{C}, \alpha, \gamma, \mathcal{A}))$ ein System abstrakter Interpretationen, wobei $\mathcal{S}_1 = (\mathcal{C}, \llbracket\ \rrbracket_1)$ und $\mathcal{S}_2 = (\mathcal{A}, \llbracket\ \rrbracket_2)$ zwei abstrakte Semantiken auf den vollständigen (Vereinigungs-) Halbverbänden $\mathcal{C}$ bzw. $\mathcal{A}$ sind.

Dann heißt $\mathcal{S}_2 = (\mathcal{A}, \llbracket\ \rrbracket_2)$

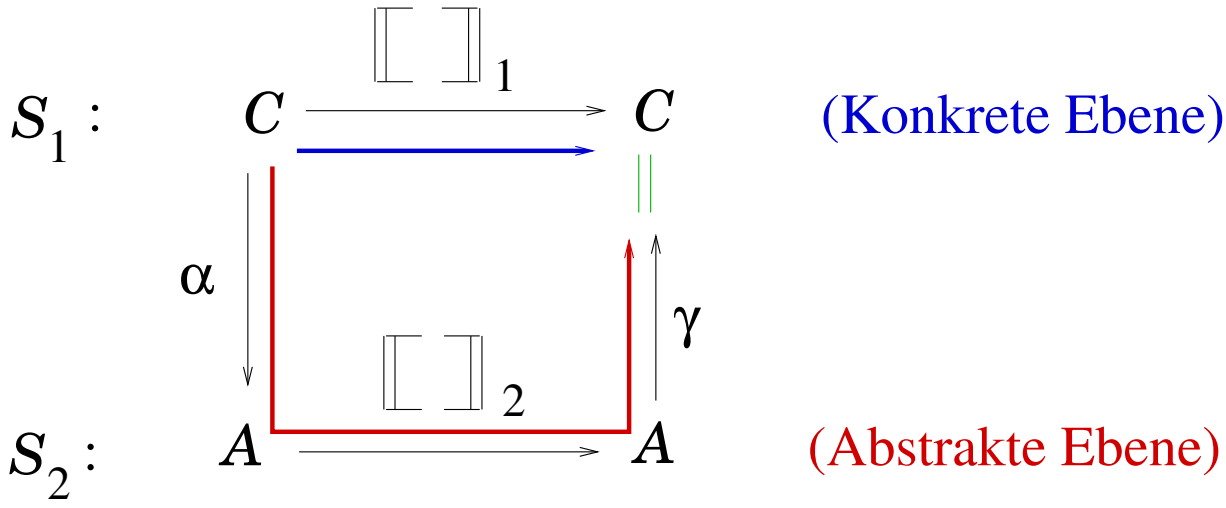
1. korrekt bzgl. $\mathcal{S}_1 = (\mathcal{C}, \llbracket\ \rrbracket_1)$, wenn das nachstehende Diagramm schwach kommutativ ist, d.h., für (mindestens) einige Elemente aus $\mathcal{C}$ die Inklusion echt ist.
2. vollständig bzgl. $\mathcal{S}_1 = (\mathcal{C}, \llbracket\ \rrbracket_1)$, wenn das nachstehende Diagramm (stark) kommutativ ist, d.h. die Inklusion für alle Elemente aus $\mathcal{C}$ unecht ist (also Gleichheit gilt).

Diagramme zu Definition 18.6.1

Korrektheitheit von $\mathcal{S}_2 = (\mathcal{A}, [\![\ ]\!]_2)$ bzgl. $\mathcal{S}_1 = (\mathcal{C}, [\![\ ]\!]_1)$:



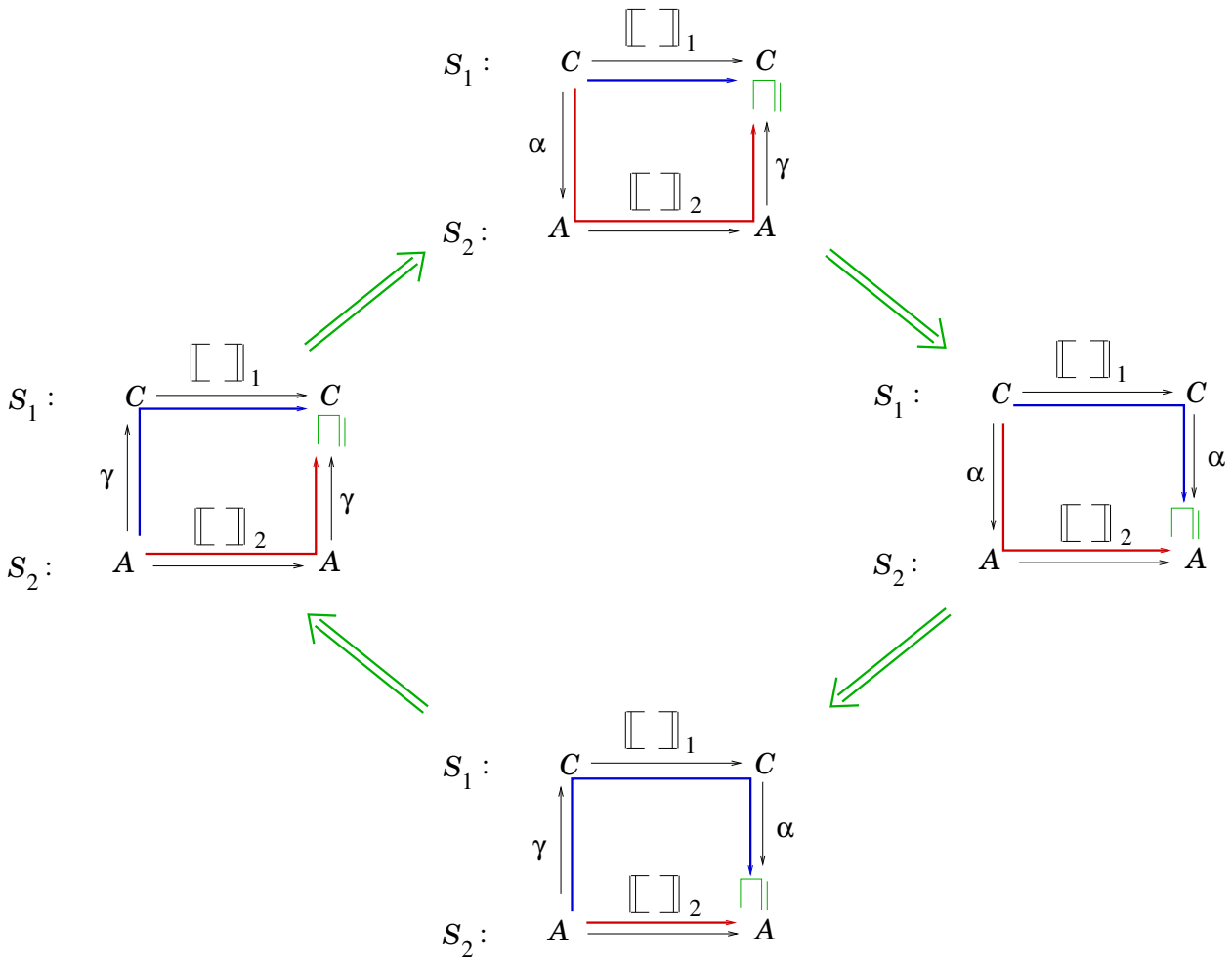
Vollständigkeit von $\mathcal{S}_2 = (\mathcal{A}, [\![\ ]\!]_2)$ bzgl. $\mathcal{S}_1 = (\mathcal{C}, [\![\ ]\!]_1)$:



Als unmittelbare Folgerung (1)

...aus Lemma 18.4.2 f. allgemeine Galois-Systeme erhalten wir:

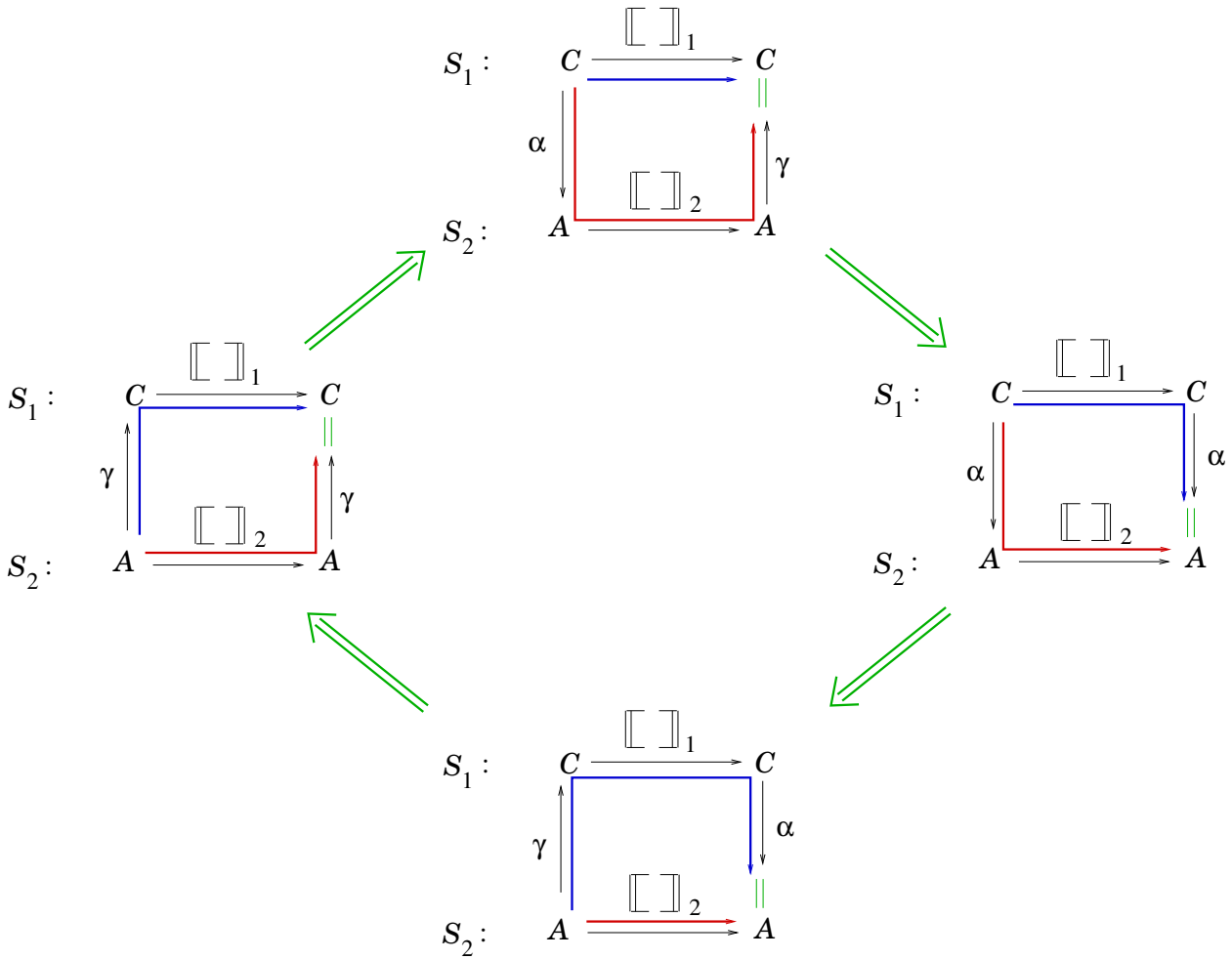
Korollar 18.6.2 (Korrektheit)



Als unmittelbare Folgerung (2)

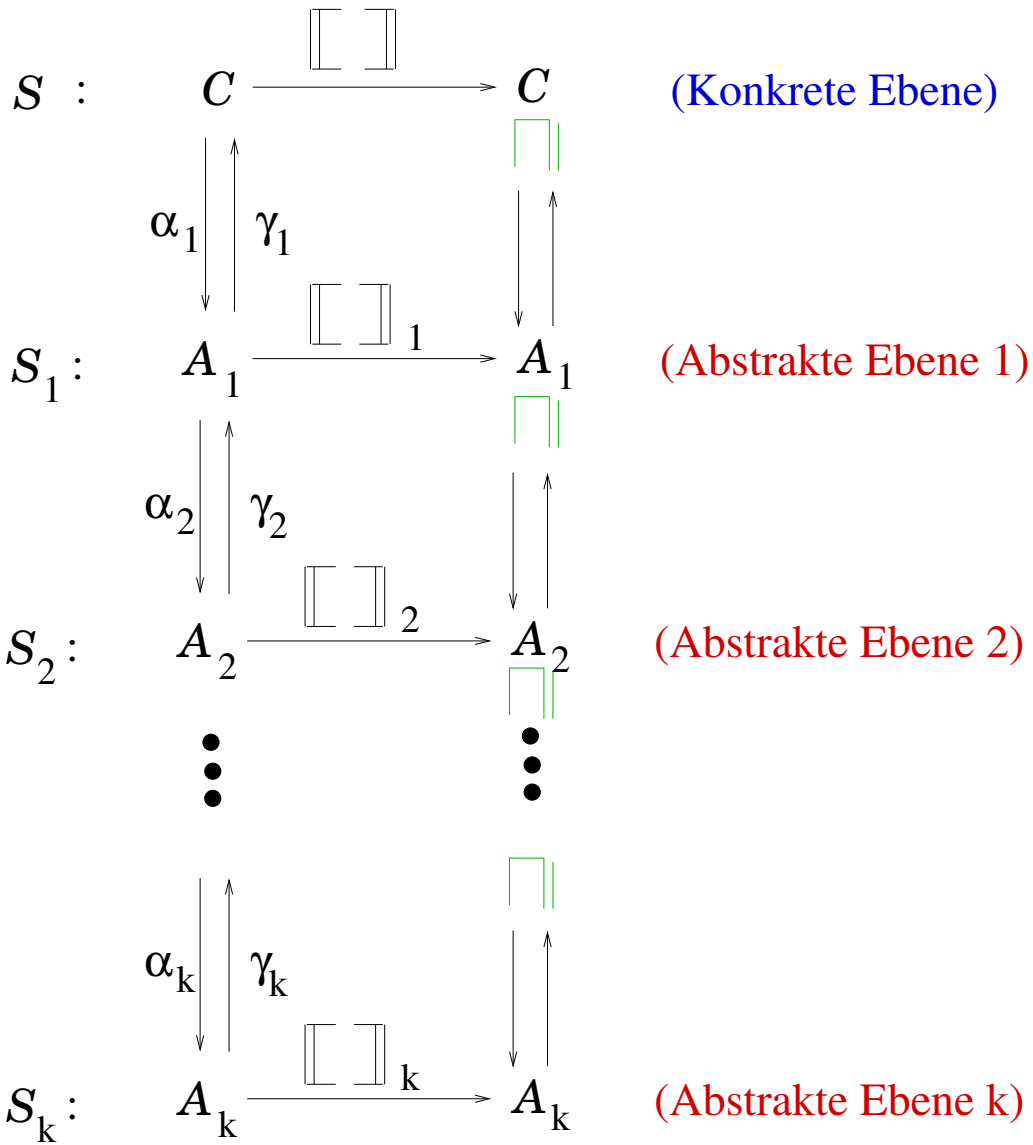
...aus Lemma 18.4.3 f. allgemeine Galois-Systeme erhalten wir:

Korollar 18.6.3 (Vollständigkeit)



Verallg. von Korrektheit und Vollständigkeit

...auf Türme von Systemen abstrakter Interpretationen:



Kapitel 18.7

Optimalität abstrakter Interpretationen

Optimalität

...fragt intuitiv nach der

- 'einfachsten', 'abstraktesten', 'niedrigstebenigen' abstrakten Semantik

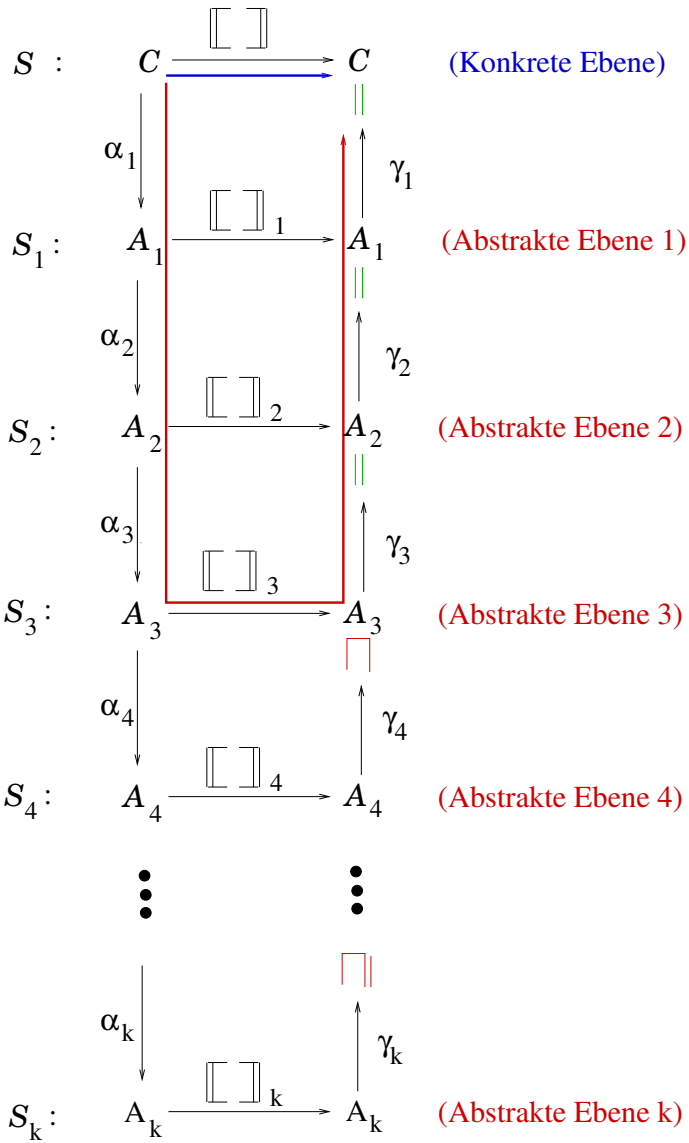
die eine Klasse $\mathcal{K}$ von Analysefrage zu beantworten erlaubt.

Dabei können intuitiv zwei Sichten unterschieden werden:

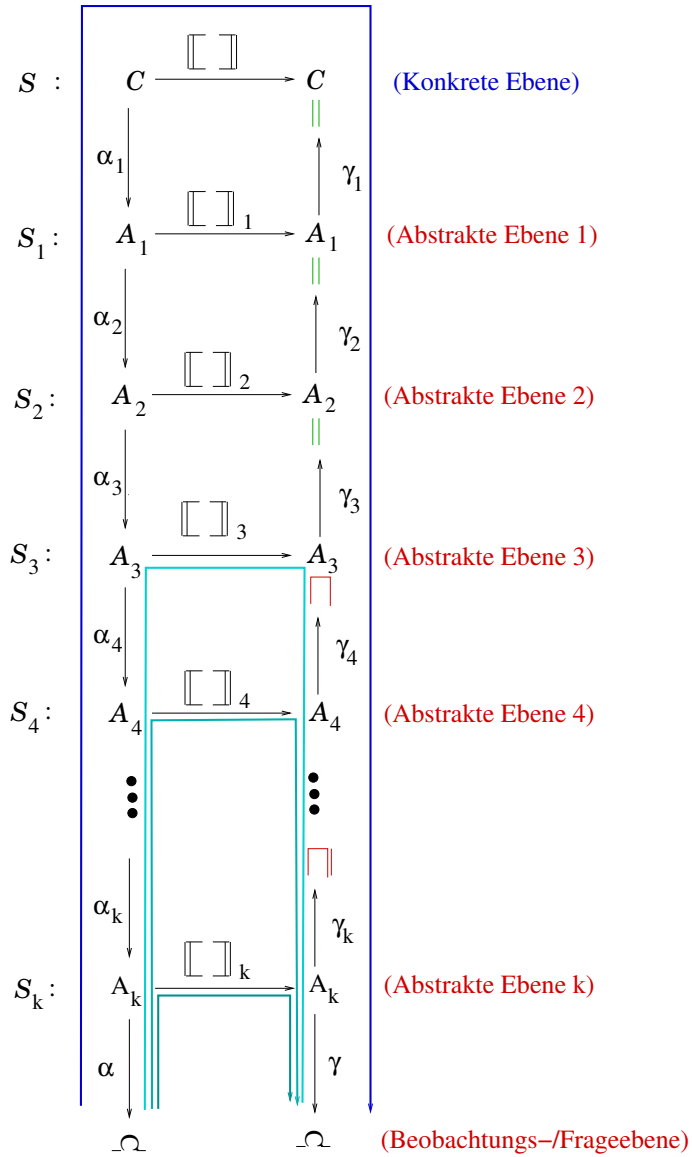
- Von oben nach unten: Wie weit kann abgestiegen werden, ohne Ausdruckskraft für $\mathcal{K}$ zu verlieren?
- Von unten nach oben: Wie weit muss aufgestiegen werden, um eine genügend ausdrucksstarke Analyse zu erreichen, um Analysefragen aus $\mathcal{K}$ zu beantworten?

Illustration

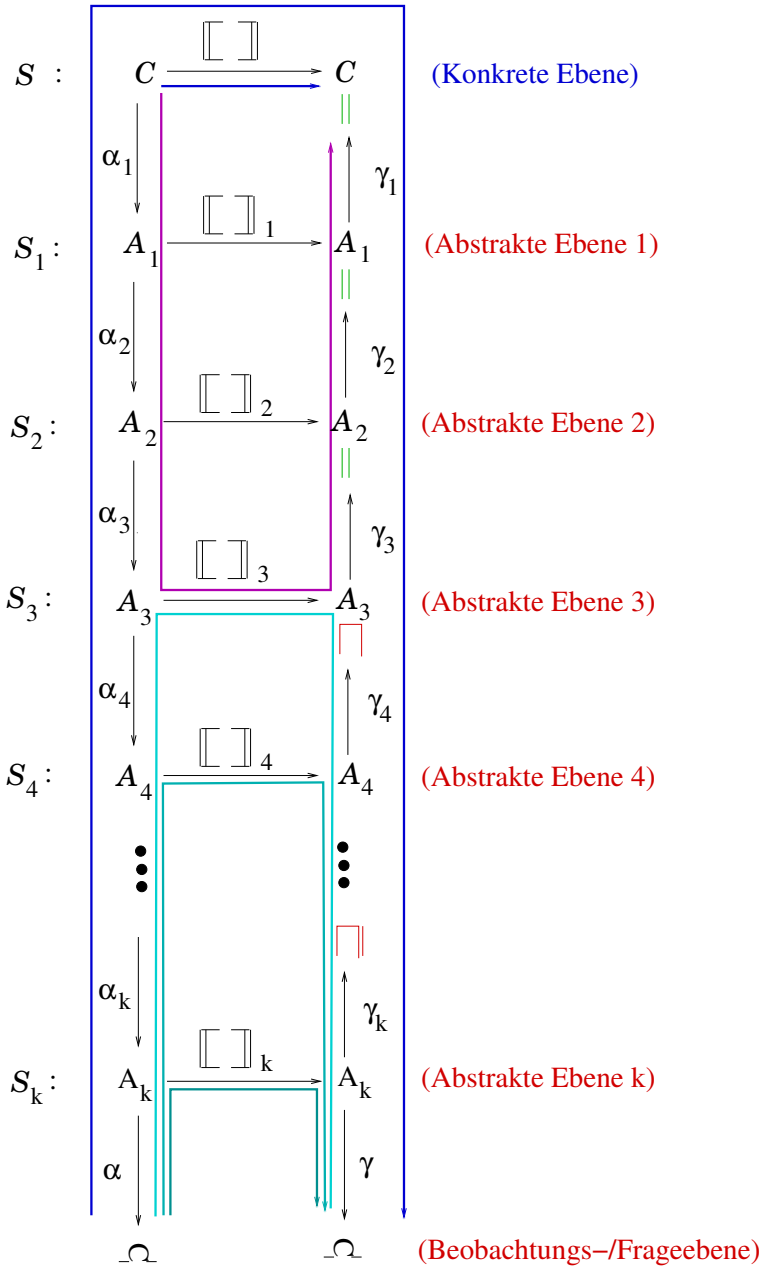
Von oben nach unten:



Von unten nach oben:



Beide Sichten vereint in einem Diagramm



Optimalität in der ‘oben-nach-unten’-Sicht

...informell ist eine abstrakte Interpretation optimal für eine Klasse von Analysefragen, wenn es keine echte Abstraktion von ihr gibt, die diese Fragen in gleicher Weise zu beantworten erlaubt.

Definition 18.7.1 (onu-Optimalität)

Sei $\mathcal{K}$ eine Klasse von Analysefragen und $\mathcal{S}_1 = (\mathcal{C}, [\![\ ]\!]_1)$ ausdruckskräftig für $\mathcal{K}$.

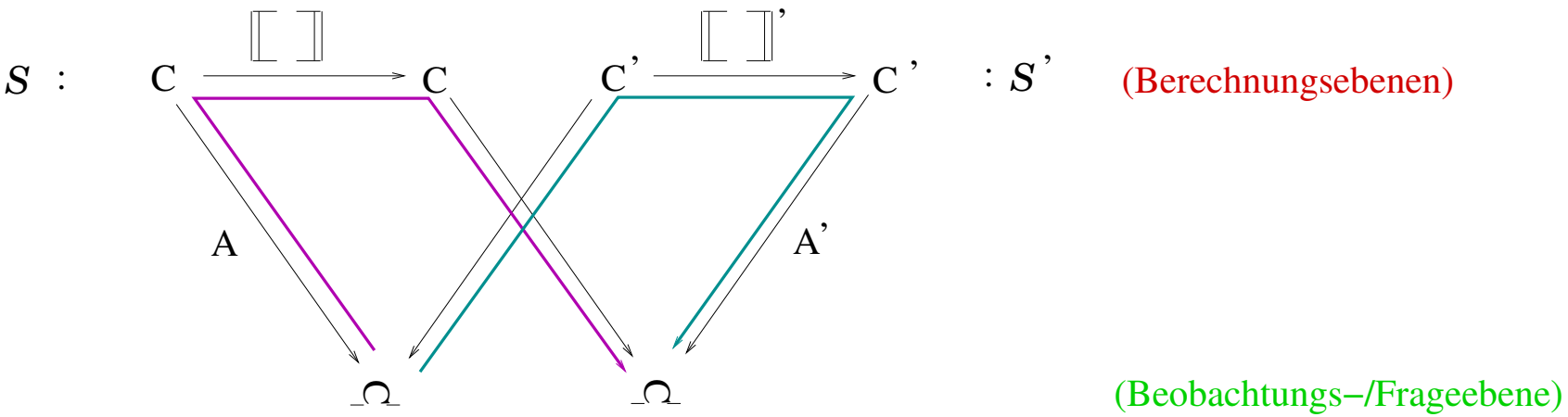
Dann ist $\mathcal{S}_1$ onu-optimal für $\mathcal{K}$, wenn alle für ‘$\mathcal{K}$ ausdruckskräftigen Abstraktionen von $\mathcal{S}_1$ zu $\mathcal{S}_1$ isomorph’ sind.

Optimalität in der ‘unten-nach-oben’-Sicht

...um den Optimalitätsbegriff in der ‘unten-nach-oben’-Sicht zu fassen, gehen wir von der streng hierarchischen zu einer allgemeineren sich auf einen Begriff von

- Beobachungsäquivalenz relativ zu einem Niveau Ω

stützenden Sicht über, die auf Bernhard Steffen (MFCS’89, TAPSOFT’87) zurückgeht und auch die ‘oben-nach-unten’-Sicht in generellerer Weise beleuchtet.



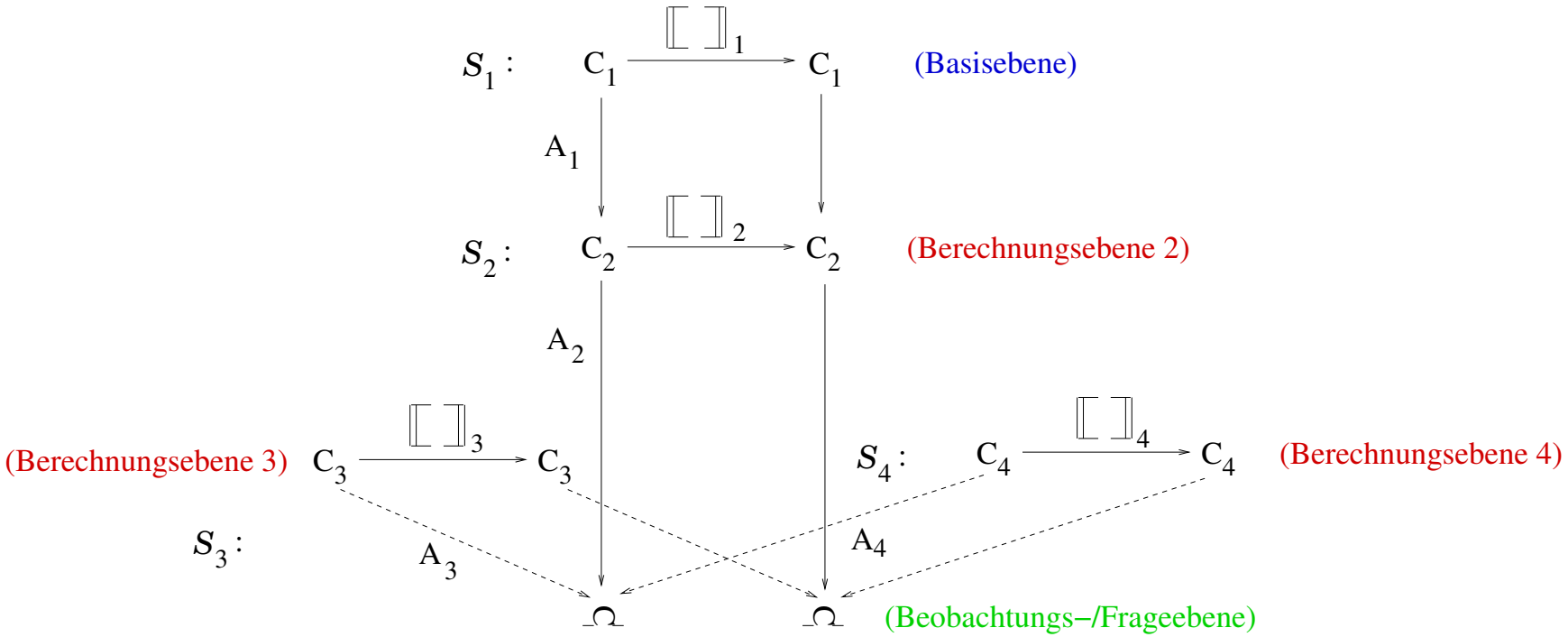
(Berechnungsebenen)

(Beobachtungs–/Frageebene)

...$\mathcal{S}$ und $\mathcal{S}'$ induzieren beide ein Berechnungsverfahren für Ω.

Illustration

...der Idee von Beobachungsäquivalenz durch induzierte Berechnungsebenen für eine gegebene Anfrage- (oder Beobachtungs-) Ebene.



...es ist nicht *per se* klar, dass ein 'einfachstes ausreichendes' Berechnungsniveau innerhalb eines Turms liegen muss.

In der Folge

...bezeichnen:

- **N** eine Menge von Knoten, die für die Menge der Vorkommen elementarer Anweisungen steht.
- $G =_{df} (N, E, \mathbf{s}, \mathbf{e})$ einen Flussgraphen mit Knotenmenge $N \subseteq \mathbf{N}$, Kantenmenge $E \sqsubseteq N \times N$, Startknoten $\mathbf{s} \in N$ und Endknoten $\mathbf{e} \in N$, wobei **s** keine Vorgänger, **e** keine Nachfolger hat; $\mathbf{P}(G)$ die Menge aller Pfadevon **s** nach **e** in G.
- **FG** die Menge aller Flussgraphen über **N** und **LFG** die Menge aller linearen Flussgraphen über **N**, d.h. die Menge aller Flussgraphen mit genau einem Pfad von **s** und **e**.
- $\mathcal{C} =_{df} (C, \sqcup, \sqsubseteq, \bot, \top)$ einen vollständigen Halbverband mit kleinstem Element $\bot$ und größtem Element $\top$.

Lokale abstrakte Semantik

Definition 18.7.2 (Lokale abstrakte Semantik)

Sei $[\![\ ]\!]_l : N \to (\mathcal{C} \to \mathcal{C})$ eine Funktion, die jedem Knoten $n \in N$ eine additive Funktion auf $\mathcal{C}$ zuordnet, d.h.

$$\forall\, n \in \mathbf{N}\ \forall\, C' \subseteq C.\ [\![\, n \,]\!]_l(\bigsqcup C') = \bigsqcup\{[\![\, n \,]\!]_l(c) \mid c \in C'\}.$$

Dann heißt das Paar $([\![\ ]\!]_l, \mathcal{C})$ eine lokale abstrakte Semantik (oder lokale abstrakte Interpretation).

Globale abstrakte Semantik

Definition 18.7.3 (Globale abstrakte Semantik)

Sei $(\llbracket\ \rrbracket_l, \mathcal{C})$ abstrakte Interpretation und $\llbracket\ \rrbracket : \mathbf{FG} \to (\mathcal{C} \to \mathcal{C})$ die Globalisierung von $\llbracket\ \rrbracket_l$, d.h. $\forall\, G \in \mathbf{FG}\ \forall\, c \in \mathcal{C}$ gilt:

$$\llbracket\, G \,\rrbracket(c) =_{df}$$

$$\begin{cases} \llbracket\, n_k \,\rrbracket_l \circ \ldots \circ \llbracket\, n_1 \,\rrbracket_l(c) & \text{falls } G = (n_1, \ldots, n_k) \in \mathbf{LFG} \\ \bigsqcup\{\llbracket\, P \,\rrbracket(c) \mid P \in \mathbf{P}(G)\} & \text{sonst} \end{cases}$$

Dann heißt das Paar $(\llbracket\ \rrbracket, \mathcal{C})$ die von $(\llbracket\ \rrbracket_l, \mathcal{C})$ induzierte (globale) abstrakte Semantik.

Abstraktion und Konkretisierung

Definition 18.7.4 (Abstraktion und Konkretisierung)

Seien $\mathcal{S}_1 = (\llbracket\ \rrbracket_1, \mathcal{C}_1)$ und $\mathcal{S}_2 = (\llbracket\ \rrbracket_2, \mathcal{C}_2)$ zwei abstrakte Semantiken.

1. Eine Funktion $A : \mathcal{C}_1 \to \mathcal{C}_2$ heißt Abstraktionsfunktion, in Zeichen $\mathcal{S}_2 \leq_A \mathcal{S}_1$, falls A additiv und surjektiv ist und die (lokale) Korrektheitsbedingung

$$\forall\, n \in \mathbf{N}.\ A \circ \llbracket\, n \,\rrbracket_1 \sqsubseteq \llbracket\, n \,\rrbracket_2 \circ A$$

   erfüllt.

2. Die Funktion $A^a : \mathcal{C}_2 \to \mathcal{C}_1$ definiert durch

$$\forall\, c \in \mathcal{C}_2.\ A^a(c) =_{df} \bigsqcup \{c' \mid A(c') = c\}$$

   heißt adjungierte (oder Konkretisierungs-) funktion zu A.

Anmerkungen zu Abstraktion/Konkretisierung

- Additivität ist eine wesentliche Anforderung an eine abstrakte Interpretation. Die meisten der folgenden Ergebnisse gelten nur unter dieser Voraussetzung.
- Surjektivität ist keine wesentliche Voraussetzung, erleichtert aber die formale Argumentation.
- Paare aus Abstraktions- und Konkretisierungsfunktion (A, A^a) sind Paare adjungierter Funktionen im Sinne von Cousot und Cousot (POPL'77) (ebenso in Kapitel 9 und 10 die Paare aus Datenfluss- und reversen Datenflussanalysefunktionen).
- Die Konkretisierungsfunktion A^a ist monoton, i.a. aber nicht additiv.
- Mit den Bezeichnungen aus Kapitel 18.2 entsprechen sich α und A und γ und A^a.

Isomorphie abstrakter Semantiken

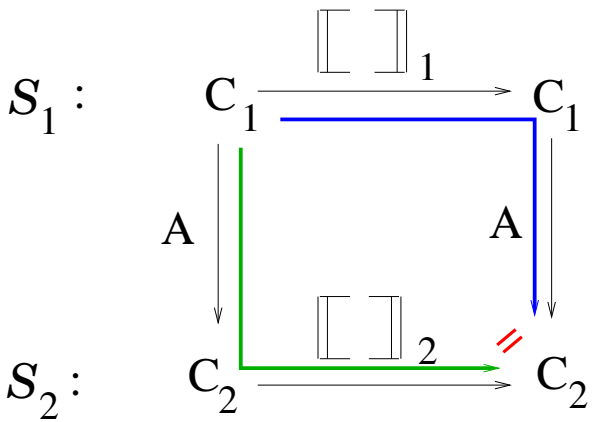
Definition 18.7.5 (Isomorphie)

Seien $\mathcal{S}_1 = (\llbracket\ \rrbracket_1, \mathcal{C}_1)$ und $\mathcal{S}_2 = (\llbracket\ \rrbracket_2, \mathcal{C}_2)$ zwei abstrakte Semantiken.

$\mathcal{S}_1$ und $\mathcal{S}_2$ heißen isomorph, in Zeichen $S_1 \approx_A S_2$ oder $S_1 \approx S_2$, wenn es eine additive und bijektive Abstraktionsfunktion $A : \mathcal{C}_1 \to \mathcal{C}_2$ gibt, so dass für alle $G \in \mathbf{FG}$ gilt:

$$A \circ \llbracket\ G\ \rrbracket_1 = \llbracket\ G\ \rrbracket_2 \circ A$$

Veranschaulichung:



Beobachtungsniveau und Beobachtung

Definition 18.7.6 (Beobachtung(sniveau))

Sei $\mathcal{S}$ eine abstrakte Semantik, Ω ('Ω' für Beobachtung) ein vollständiger Halbverband und $A : \mathcal{C} \to \Omega$ eine additive und surjektive Funktion.

Dann induziert $\mathcal{S}$ ein Semantikfunktional oder Verhalten $[\![\ ]\!]_A : \mathbf{FG} \to (\Omega \to \Omega)$ auf Ω durch

$$\forall\, G \in \mathbf{FG}.\ [\![\ G\ ]\!]_A =_{df} A \circ [\![\ G\ ]\!] \circ A^a$$

Wir bezeichnen diese Situation mit $\mathcal{S} \to_A \Omega$ und nennen Ω ein Beobachtungsniveau.

Modelle

Definition 18.7.7 (Modelle)

Sei Ω ein Beobachtungsniveau und $\mathcal{S}$ eine abstrakte Semantik mit $\mathcal{S} \to_A \Omega$.

Dann heißt das Paar $(\mathcal{S}, A)$ ein Modell von Ω.

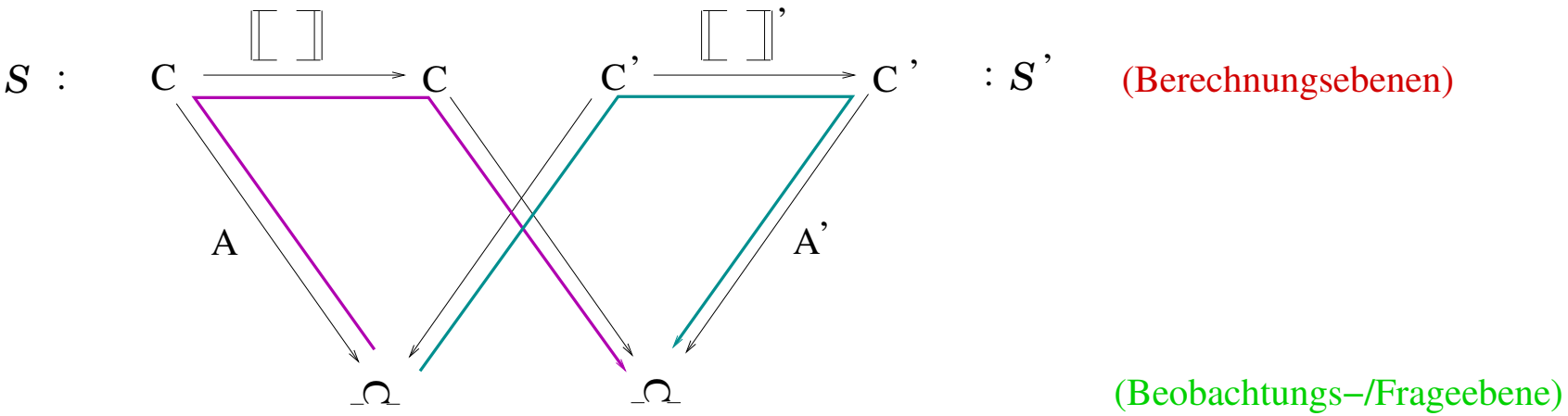
Beobachtungsäquivalenz

Definition 18.7.8 (Beobachtungsäquivalenz)

Seien $(\mathcal{S}, A)$ und $(\mathcal{S}', A')$ zwei Modelle von Ω.

Dann heißen $(\mathcal{S}, A)$ und $(\mathcal{S}', A')$ Ω-äquivalent (oder beobachtungsäquivalent für Ω), in Zeichen $(\mathcal{S}, A) \approx_\Omega (\mathcal{S}', A')$ gdw sie dasselbe Verhalten auf Ω induzieren, d.h. gdw $[\![\ ]\!]_A = [\![\ ]\!]_{A'}$.

Veranschaulichung:



(Berechnungsebenen)

(Beobachtungs–/Frageebene)

Eigenschaften der Relation $\approx_\Omega$

Lemma 18.7.9 (Äquivalenzrelation)

Die Relation $\approx_\Omega$ ist eine Äquivalenzrelation auf der Menge aller Modelle von Ω.

Verbandshüllen

Definition 18.7.10 (Verbandshüllen)

Sei $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$ und $\mathcal{S}_2 \to_{A_2} \Omega$. Dann definieren wir:

1. $RI(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega) =_{df} \{c \in \mathcal{C}_2 \mid \exists\, G \in \mathbf{FG}\ \exists\, c' \in \Omega.\ c = A_1 \circ [\![\ G\ ]\!]_1 \circ A_1^a \circ A_2^a\}(c')\ \}$
2. $RL(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)$ bezeichnet die vollständige Halbverbandshülle von $RI(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)$ in $\mathcal{C}_2$.
3. $RS(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega) =_{df} ([\![\ ]\!], RL(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega))$, wobei $[\![\ ]\!]$ folgendermaßen definiert ist:

   $\forall\, G \in \mathbf{FG}.\ [\![\ G\ ]\!] =_{df}$

$$\begin{cases} [\![\ G\ ]\!]_2 \mid_{RL(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)} & \text{falls } RL(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega) \\ & \text{abgeschlossen ist unter } [\![\ ]\!]_2 \\ \bot & \text{sonst} \end{cases}$$

Lokale Optimalität

Definition 18.7.11 (Lokale Optimalität)

Sei $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$ und $\mathcal{S}_2 \to_{A_2} \Omega$. Dann heißt $\mathcal{S}_2$ lokal optimal für $\mathcal{S}_1$ und A gdw für alle $n \in \mathbf{N}$ gilt:

$$A_1 \circ [\![n]\!]_1 \circ A_1^a = [\![n]\!]_2$$

Voll abstrakte Modelle

Definition 18.7.12 (Voll abstrakte Modelle)

Seien $\mathcal{S}_1$, Ω und A mit $\mathcal{S}_1 \to_A \Omega$.

Ein Paar $(\mathcal{S}_2, A_2)$ mit $\mathcal{S}_2 \to_A \Omega$ heißt voll abstraktes Modell für $\mathcal{S}_1$ bezüglich Ω gdw eine Abstraktionsfunktion A_1 mit $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$ existiert, die folgende 4 Eigenschaften erfüllt:

1. $A = A_2 \circ A_1$
2. $\mathcal{S}_2 = RS(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)$
3. $\mathcal{S}_2$ ist lokal optimal für $\mathcal{S}_1$ und A_1
4. $\forall\, c, c' \in RI(\mathcal{S}_1, ID, \mathcal{S}_1, A, \Omega).\ A_1(c) = A_1(c') \Longleftrightarrow \forall\, G \in$ **LFG**. $A \circ [\![\, G \,]\!]_1(c) = A \circ [\![\, G \,]\!]_1(c')$, wobei ID die Identität auf dem semantischen Bereich von $\mathcal{S}_1$ ist.

Wir bezeichnen die Menge aller voll abstrakten Modelle für $\mathcal{S}_1$, Ω und A, die in Beziehung $\mathcal{S}_1 \to_A \Omega$ stehen, mit $\Phi(\mathcal{S}_1, A, \Omega)$.

Existenz und Eindeutigkeit

Theorem 18.7.13 (Existenz und Eindeutigkeit)

Seien $\mathcal{S}_1$, Ω und A mit $\mathcal{S}_1 \to_A \Omega$.

Dann gibt es ein voll abstraktes Modell $(\mathcal{S}_2, A_2)$ für $\mathcal{S}_1$ bezüglich Ω mit folgender Eindeutigkeitseigenschaft:

$$\Phi(\mathcal{S}_1, A, \Omega) = \{(\mathcal{S}'_2, A'_2) \mid \exists\, A'.\ \mathcal{S}_2 \approx_{A'} \mathcal{S}'_2 \wedge A_2 = A'_2 \circ A'\}$$

Voll abstrakt ist ‘gut genug’

Theorem 18.7.14 (Abschneidetheorem)

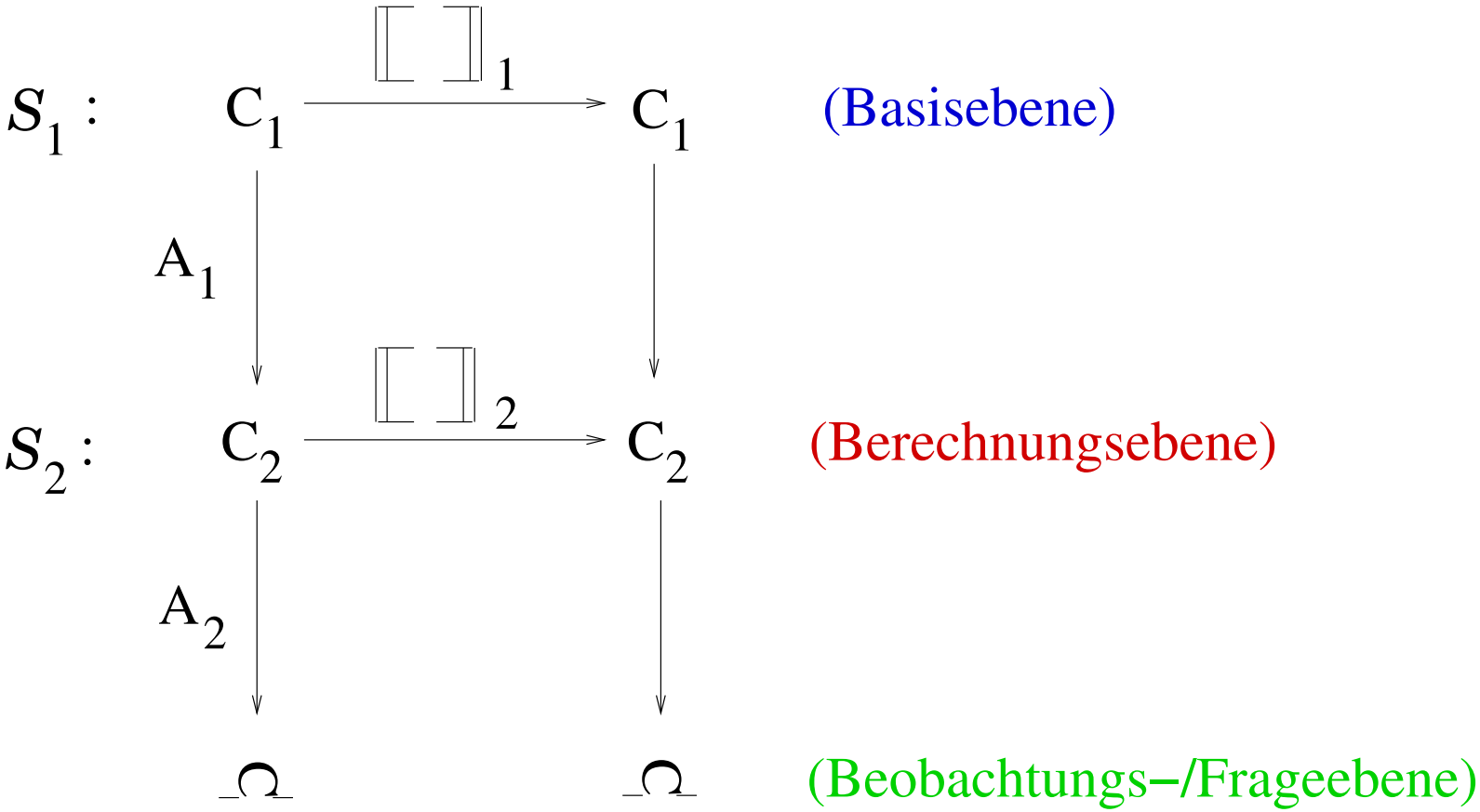
Sei $(\mathcal{S}_2, A_2) \in \Phi(\mathcal{S}_1, A_2 \circ A_1, \Omega)$ mit $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$. Dann gilt:

$$\forall\, G \in \mathbf{FG}.\ A_1 \circ [\![\ G\ ]\!]_1 \circ A_1^a = [\![\ G\ ]\!]_2$$

Insbesondere gilt weiters:

$$(\mathcal{S}_1, A_2 \circ A_1) \approx_\Omega (\mathcal{S}_2, A_2)$$

Veranschaulichung



Äquivalenz

Theorem 18.7.15 (Äquivalenz)

Seien $(\mathcal{S}, A)$ und $(\mathcal{S}', A')$ zwei Modelle von Ω. Dann gilt:

$$(\mathcal{S}, A) \approx_\Omega (\mathcal{S}', A') \Longleftrightarrow \Phi(\mathcal{S}, A, \Omega) = \Phi(\mathcal{S}', A', \Omega)$$

Intuitiv: Zusammen mit Existenz- und Eindeutigkeitstheorem 18.5.12 und Abschneidetheorem 18.5.13 liefert Äquivalenztheorem 18.5.14, dass voll abstrakte Modelle (bis auf Isomorphie) die 'abstraktesten' Repräsentanten ihrer Beobachtungsäquivalenzklasse sind.

Interpretation und Folgerung (1)

Zu vorgegebenem Beobachtungsniveau Ω und Modell $(\mathcal{S}, A)$ von Ω gibt es ein

- beobachtungsäquivalentes 'abstraktestes' Berechnungsniveau.

Dieses 'abstrakteste' Berechnungsniveau ist das bis auf Isomorphie

- eindeutig bestimmte voll abstrakte Modell.

Das voll abstrakte Modell ist das gesuchte

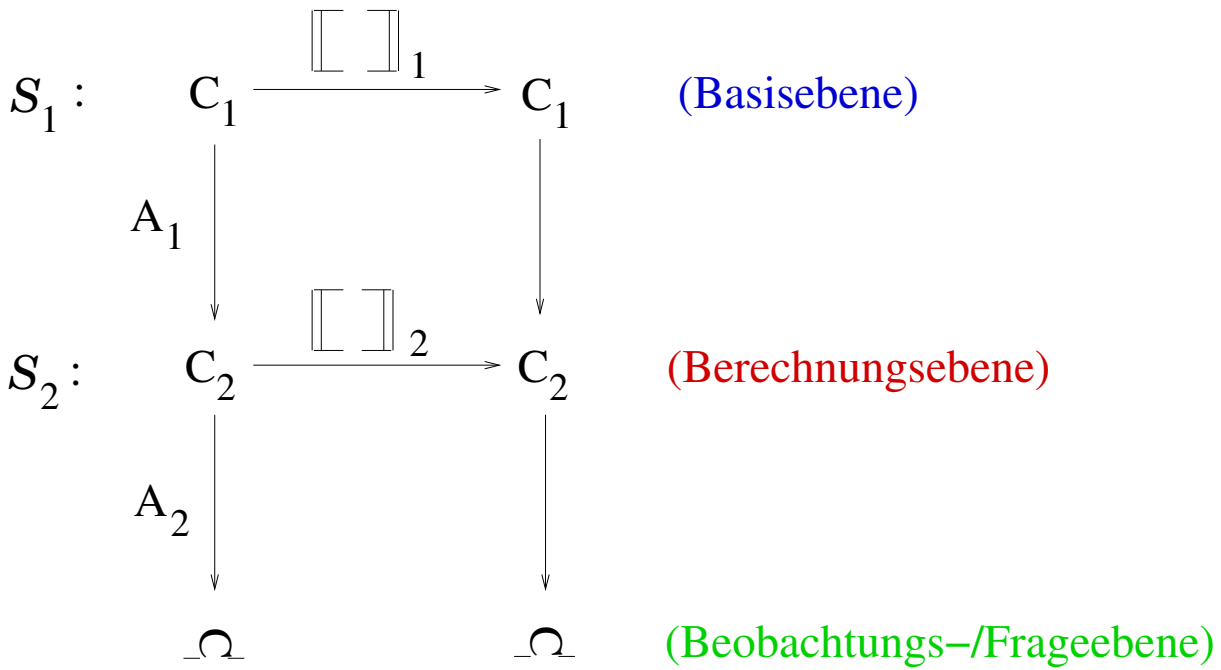
- korrekte, vollständige und optimale Modell.

Interpretation und Folgerung (2)

Das voll abstrakte Modell liegt hierarchisch eingebettet innerhalb des

- 3-stufigen Modells.

In diesem Sinn ist das 3-stufige Modell hinreichend allgemein für den nicht auf Hierarchien abstrakter Interpretationen beschränkten Begriff der Beobachtungsäquivalenz.



Kapitel 18.8

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (1)

- Samson Abramsky, Chris Hankin. *An Introduction to Abstract Interpretation*. In Abstract Interpretation of Declarative Languages, Samson Abramsky, Chris Hankin (Hrsg.), Prentice Hall, 63-102, 1987.
- Patrick Cousot. *Methods and Logics for Proving Programs*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Hrsg.), Elsevier Science Publishers B. V., Chapter 15, 841-993, 1990.
- Patrick Cousot. *Abstract Interpretation*. ACM Computing Surveys 28(2):324-328, 1996.
- Patrick Cousot. *Refining Model-Checking by Abstract Interpretation*. Automated Software Engineering 6(1):69-95, 1999.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (2)


Patrick Cousot. *Design of Syntactic Program Transformations by Abstract Interpretation of Semantic Transformations*. In Proceedings of the 17th International Conference on Logic Programming (ICLP 2001), Springer-V., LNCS 2237, 4-5, 2001.


Patrick Cousot. *The Verification Grand Challenge and Abstract Interpretation*. In Proceedings of Verified Software: Theories, Tools, Experiments (VSTTE 2005), Springer-V, LNCS 4171, 189-201, 2005.


Patrick Cousot. *Verification by Abstract Interpretation*. In Proceedings of the 4th International Conference on Verification, Model-Checking, Abstract Interpretation (VMCAI 2003), Springer-V., LNCS 2575, 20-24, 2003.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (3)

Patrick Cousot. *Verification by Abstract Interpretation*. In Verification: Theory and Practice, Essays dedicated to Zohar Manna on the Occasion of His 64th Birthday. Springer-V., LNCS 2772, 243-268, 2003.

Patrick Cousot, Radhia Cousot. *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In Conference Record of the 4th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77), 238-252, 1977.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (4)


Patrick Cousot, Radhia Cousot. *Systematic Design of Program Analysis Frameworks*. In Conference Record of the 6th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'79), 269-282, 1979.


Patrick Cousot, Radhia Cousot. *Abstract Interpretation Frameworks*. Journal of Logic and Computation 2(4):511-547, 1992.


Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (5)


Patrick Cousot, Radhia Cousot. *A Gentle Introduction to Formal Verification of Computer Systems by Abstract Interpretation*. In Logics and Languages for Reliability and Security. NATO Science for Peace and Security - D; Information and Communication Security, Vol. 25, IOS Press, 2010. ISBN 978-1-60750–099-5.


Patrick Cousot, Radhia Cousot, Laurent Mauborgne. *Theories, Solvers and Static Analysis by Abstract Interpretation*. Journal of the ACM 59(6), Article 31, 56 Seiten, 2012.


Patrick Cousot, Michael Monerau. *Probabilistic Abstract Interpretation*. In Proceedings of the 21st Symposium on Programming (ESOP 2012), Springer-V., LNCS 7211, 169-193, 2012.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (6)


Nevin Heintze, Joxan Jaffar, Răzvan Voicu. *A Framework for Combining Analysis and Verification*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 26-39, 2000.


Neil D. Jones, Flemming Nielson. *Abstract Interpretation: A Semantics-based Tool for Program Analysis*. In Handbook of Logic in Computer Science, Volume 4, Oxford University Press, 1995.


Kim Marriot. *Frameworks for Abstract Interpretation*. Acta Informatica 30:103-129, 1993.


Flemming Nielson. *A Bibliography on Abstract Interpretations*. ACM SIGPLAN Notices 21:31-38, 1986.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (7)


Flemming Nielson. *A Bibliography on Abstract Interpretation*. EATCS Bulletin 28:42-52, 1986.


Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. 2nd edition, Springer-V., 2005. (Chapter 1.5, Abstract Interpretation; Chapter 4, Abstract Interpretation)


Bernhard Steffen. *Optimal Run Time Optimization – Proved by a New Look at Abstract Interpretation*. In Proceedings of the 2nd Joint International Conference on the Theory and Practice of Software Development (TAPSOFT'87), Springer-V., LNCS 249, 52-68, 1987.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (8)

Bernhard Steffen. *Optimal Data Flow Analysis via Observational Equivalence*. In Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS'89), Springer-V., LNCS 379, 492-502, 1989.

Kapitel 19

Modellprüfung und Datenflussanalyse

Kapitel 19.1

Motivation

Motivation

...das Grundproblem der Modellprüfung (engl. model-checking).

Gegeben:

- Ein Modell $\mathcal{M}$
- Eine Eigenschaft $\mathcal{E}$ in Form einer Formel ϕ

Modellprüfungsfrage:

- Ist $\mathcal{M}$ ein Modell für Eigenschaft $\mathcal{E}$, erfüllt $\mathcal{M}$ Eigenschaft ϕ?

$$\mathcal{M} \models \phi$$

Kapitel 19.2

Modellprüfer, Modellprüfung

Modellprüfer und Modellprüfung

Modellprüfer: Eine Methode, ein Werkzeug zur Beantwortung von Modellprüfungsfragen.

Modellprüfung: Ansetzen eines Modellprüfers MP auf ein Paar $\mathcal{M}, \phi$ aus Modell $\mathcal{M}$ und Formel ϕ:

- $\mathcal{M} \models_{MP} \phi$ wird nachgewiesen: $\mathcal{M}$ ist bezüglich ϕ verifiziert (oder ϕ ist für $\mathcal{M}$ verifiziert).
- $\mathcal{M} \models_{MP} \phi$ wird widerlegt: $\mathcal{M}$ ist bezüglich ϕ falsifiziert (oder ϕ ist für $\mathcal{M}$ falsifiziert).

  Wünschenswert: Ausgabe eines (minimalen) Gegenbeispiels, das die Verletzung der Formel zeigt (CEGAR ('counter-example-guided abstraction/refinement'-Ansatz).
- $\mathcal{M} \models_{MP} \phi$ wird weder noch nachgewiesen oder widerlegt: Modellprüfer ist unvollständig für Modell- und Formelsprache.

Kapitel 19.3

Modell- und Formelsprachen

Modellsprachen

...typischerweise:

- Transitionssysteme (kantenbenannt)
- Kripke-Strukturen (knotenbenannt)

Spezielle Ausprägungen:

- Automaten
- Flussgraphen (kanntenbenannt: Transitionssystem;
  knotenbenannt: Kripke-Struktur)
- Zustandsgraphen (z.B. Programmzustandsgraphen)
- ...

Modelle

...können sein:

- endlich: Endliche Modellprüfung
- unendlich: Unendliche Modellprüfung

Herausforderung für Modellprüferbau und Modellprüfung:

...die Meisterung der Explosion des Zustandsraums, eines notorischen (auch im Fall endlicher Modellprüfung schwierig zu handhabenden) Problems, kurz:

- Zustandsraumexplosion

Formelsprachen

...sind typischerweise:

- Temporale, modale Logiken (Linearzeitlogik (engl. linear time logics), Verzweigungszeitlogik (engl. branching time logics)
  - LTL, CTL, CTL*
  - μ-Kalkül
  - ...

Herausforderung:

...Ausdruckskraft und Entscheidbarkeit, vor allem effiziente Entscheidbarkeit der Formelsprache

- ausgewogen auszubalancieren.

Beispiel: Modaler μ-Kalkül: Syntax

...hier erweitert um sog. Rückwärtsmodalitäten erweitert, wobei $\mathcal{S}$ die Knotenmenge eines Modells bezeichnet, λ eine Abbildung, die jedem Knoten aus $\mathcal{S}$ eine Menge von Benennungen (aus der von β erzeugten Sprache) zuordnet, die von X und α erzeugten Sprachen eine Variablenmenge bzw. eine Menge von Transitionsbenennungen (d.h. Kantenbenennungen) sind.

Syntax:

$$\Phi \ ::= \ tt \mid X \mid \Phi \wedge \Phi \mid \neg\Phi \mid \beta \mid [\alpha]\,\Phi \mid \overline{[\alpha]}\,\Phi \mid \nu X.\,\Phi$$

Modaler μ-Kalkül: Semantik

Semantik:

$$
\begin{aligned}
[\![tt]\!]\, e &= \mathcal{S} \\
[\![X]\!]\, e &= e(X) \\
[\![\Phi_1 \wedge \Phi_2]\!]\, e &= [\![\Phi_1]\!]\, e \wedge [\![\Phi_2]\!]\, e \\
[\![\neg\Phi]\!]\, e &= \mathcal{S} \backslash [\![\Phi]\!]\, e \\
[\![\beta]\!]\, e &= \{p \in \mathcal{S} \mid \beta \in \lambda(p)\} \\
[\![[\alpha]\Phi]\!]\, e &= \{p \in \mathcal{S} \mid \forall\, q \in \mathit{Succ}_\alpha.\ q \in [\![\Phi]\!]\, e\} \\
[\![\overline{[\alpha]}\,\Phi]\!]\, e &= \{p \in \mathcal{S} \mid \forall\, p \in \mathit{Pred}_\alpha.\ p \in [\![\Phi]\!]\, e\} \\
[\![\nu X.\Phi]\!]\, e &= \bigcup \{S' \subseteq \mathcal{S} \mid S' \subseteq [\![\Phi]\!]\, e[S'/X]\}
\end{aligned}
$$

wobei e für eine Umgebung (oder Variablenbelegung) (engl. environment) steht: '$e : X \to \mathcal{P}(\mathcal{S})$'

Modaler μ-Kalkül: Informelle Interpretation

...der (allquantifizierten) modalen Operatoren:

- $[\![\, [\alpha]\, \Phi \,]\!]\, e$: Die Menge aller Knoten n, für die gilt: Ausgewertet in Umgebung e gilt für alle α-Nachfolger von n Eigenschaft ϕ.
- $[\![\, \overline{[\alpha]}\, \Phi \,]\!]\, e$: Die Menge aller Knoten n, für die gilt: Ausgewertet in Umgebung e gilt für alle α-Vorgänger von n Eigenschaft ϕ.

Modaler μ-Kalkül: Abgeleitete Operatoren (1)

Abgeleitete Operatoren:

$$
\begin{aligned}
ff &= \neg tt \\
\Phi_1 \vee \Phi_2 &= \neg(\neg\Phi_1 \wedge \neg\Phi_2) \\
\langle\alpha\rangle\Phi &= \neg[\alpha](\neg\Phi) \\
\overline{\langle\alpha\rangle}\Phi &= \neg\overline{[\alpha]}(\neg\Phi) \\
\mu X.\,\Phi &= \nu X.\,\neg(\Phi[\neg X/X]) \\
\Phi \succ \Psi &= \neg\Phi \vee \Psi
\end{aligned}
$$

...informelle Interpretation der (existentiell quantifizierten) modalen Operatoren:

- $[\![\langle\alpha\rangle \Phi]\!]\, e$ ($[\![\overline{\langle\alpha\rangle} \Phi]\!]\, e$): Die Menge aller Knoten n, für die gilt: Ausgewertet in Umgebung e gibt es einen α-Nachfolger (α-Vorgänger) von n, für den Eigenschaft ϕ gilt.

Modaler μ-Kalkül: Abgeleitete Operatoren (2)

Höher-abstrakte abgeleitete Operatoren:

$$\begin{aligned} \mathbf{AG}\ \Phi &= \nu X.(\Phi \wedge [.]X) \\ \Phi\ \mathbf{U}\ \Psi &= \nu X.(\Psi \vee (\Phi \wedge [.]X) \end{aligned}$$

$$\begin{aligned} \overline{\mathbf{AG}}\ \Phi &= \nu X.(\Phi \wedge \overline{[.]}X) \\ \Phi\ \overline{\mathbf{U}}\ \Psi &= \nu X.(\Psi \vee (\Phi \wedge \overline{[.]}X) \end{aligned}$$

Modaler μ-Kalkül: Informelle Interpretation

...der höher-abstrakten abeleiteten modalen Operatoren:

- **AG** Φ: Eigenschaft ϕ gilt in der Zukunft immer und überall (engl. always generally (forward)), d.h. jeder von einem Knoten (einschließlich des Knotens selbst) vorwärts erreichbare Knoten erfüllt ϕ.
- $\overline{\textbf{AG}}$ Φ: Eigenschaft ϕ hat in der Vergangenheit immer und überall gegolten (engl. always generally (backward)), d.h. jeder von einem Knoten (einschließlich des Knotens selbst) rückwärts erreichbare Knoten erfüllt ϕ.
- Φ **U** Ψ: Φ gilt vorwärts bis Ψ eintritt (engl. until (forward)).
- Φ $\overline{\textbf{U}}$ Ψ: Φ gilt rückwärts bis Ψ eintritt (engl. until (backward)).

Modaler μ-Kalkül: Zwei Ausprägungen

...des bis-Operators als sog.:

- starkes bis (engl. strong until): Φ gilt bis schließlich (engl. eventually) (d.h. in jedem Fall) Ψ eintritt.
- schwaches bis (engl. weak until): Φ gilt bis Ψ eintritt (möglicherweise nie; in diesem Fall gilt Φ für alle Zeit).

Übungsaufgabe 19.3.1

1. Sind die Operatoren **U** und $\overline{\mathbf{U}}$ vorstehend im Sinne eines starken oder schwachen bis definiert?
2. Wie müssen die Semantikdefinitionen von **U** und $\overline{\mathbf{U}}$ geändert werden, um bis-Operatoren im jeweils anderen Sinn zu erhalten?

Kapitel 19.4

Modellprüfung und DFA: Eine Analogie

Analogie Modellprüfung – Datenflussanalyse

Datenflussanalyse (als Abbildung verstanden):

DFA-Algorithmus für Eigenschaft ϕ :

Programm $\rightarrow$ Menge der ϕ erfüllenden Programmpunkte

Modellprüfung (als Abbildung verstanden):

Modellprüfer :

Formel $\times$ Modell $\rightarrow$ Menge der die Formel erfüllenden Zustände

Informell

...und holzschnittartig:

Ein DFA-Algorithmus für ϕ ist ein

- bezüglich ϕ partiell ausgewerteter (oder bezüglich ϕ spezialisierter) Modellprüfer!

Umgekehrt:

Ein Modellprüfer ist ein

- in einer Menge von Programmeigenschaften (d.h. ausdrückbar in der Formelsprache) parametrisierter (generischer) DFA-Algorithmus.

Anwendung: PREE für einen Term t

Sicherheit (Notwendigkeit der Berechnung):

$$NOTW =_{df} (\neg(Mod \vee end)) \ \mathbf{U} \ Used$$

Frühestheit (Wert kann nicht früher bereitgestellt werden):

$$FRUEH =_{df} start \vee \neg(\overline{[.]}((\neg(Mod \vee start)) \ \overline{\mathbf{U}} \ (NOTW \wedge \neg Mod)))$$

Berechnungspunkte:

$$BP =_{df} NOTW \wedge FRUEH$$

PREE-Optimierungstransformation OT_{PREE}:

1. Deklariere eine frische Hilfsvariable h_t für t.
2. Initialisiere h_t an allen Punkten in BP mit $h_t := t$.
3. Ersetze alle Vorkommen von t im Programm durch h_t.

Korrektheit und Optimalität

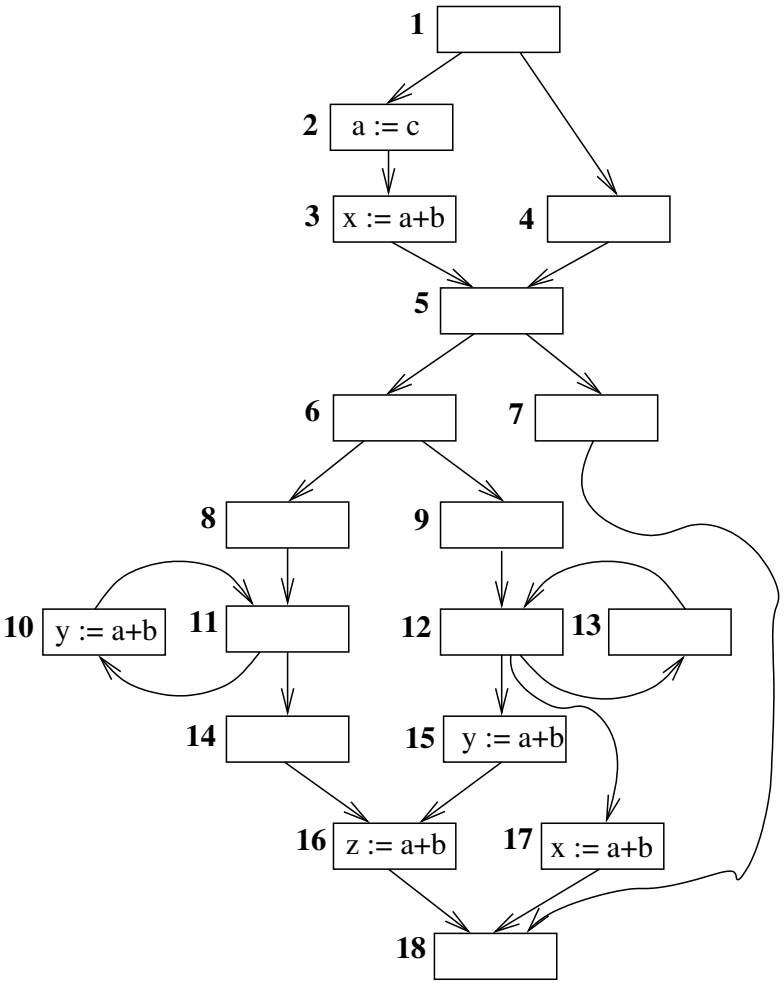
Theorem 19.4.1 (Korrektheit und Optimalität)

OT_{PREE} ist korrekt (d.h. semantikerhaltend) und optimal (d.h. mindestens so gut wie jede andere korrekte Platzierung der Berechnungen von t).

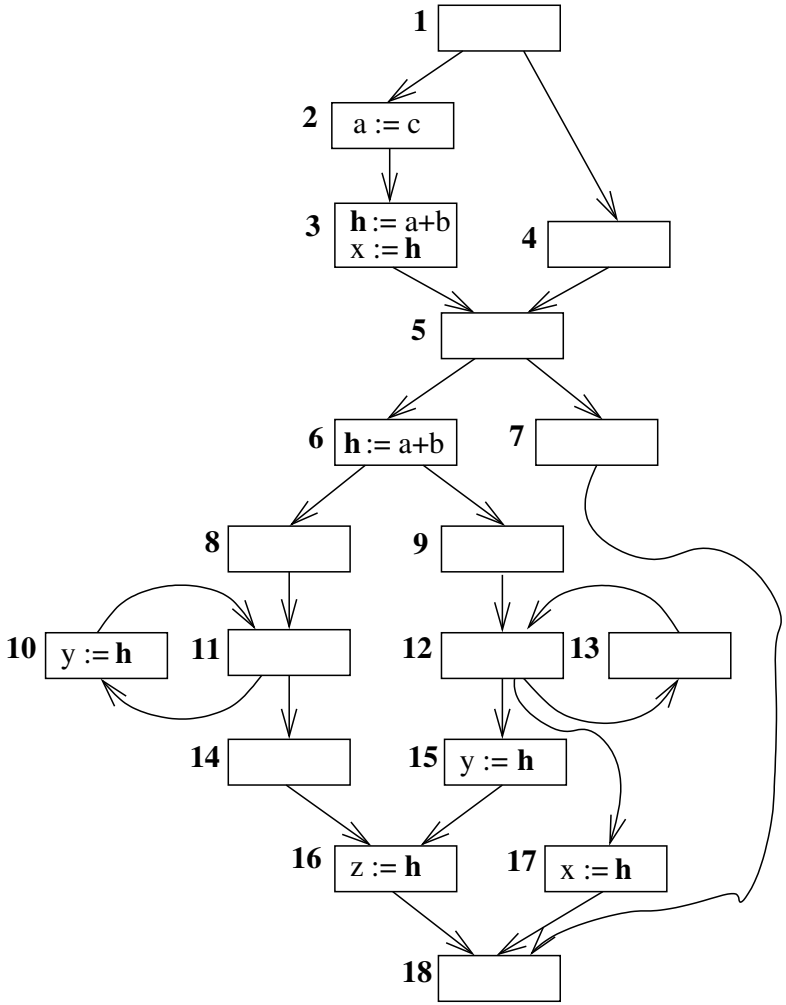
Beachte: OT_{PREE} entspricht der *busy code motion*-Transformation für die Elimination partiell redundanter Berechnungen (s. LVA 185.A04 Optimierende Compiler, Kapitel 7; auch zur formalen Definition von 'korrekt' und 'optimal').

Beispiel: Anwendung von OT_{PREE}

Ausgangsprogramm



OT_{PREE}-optimiertes Programm



Bem.: OT_{PREE} nimmt kanten-, nicht knotenben. Graphen an.

Kapitel 19.5

Zusammenfassung, Hinweise

Zusammenfassung, Hinweise (1)

...die vorgestellte Charakterisierung des Zusammenhangs von DFA und Modellprüfung und die PREE-Anwendung geht zurück auf:

- Bernhard Steffen. Data Flow Analysis as Model Checking. In Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'91), Springer-V., LNCS 526, 346-365, 1991.
- Bernhard Steffen. Generating Data Flow Analysis Algorithms from Modal Specifications. International Journal on Science of Computer Programming 21:115-139, 1993.

Zusammenfassung, Hinweise (2)

...ist aufgegriffen worden von:

- David A. Schmidt. Data Flow is Model Checking of Abstract Interpretations. In Conference Record of the 25th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'98), 38-48, 1998.

...und hat in weiterer Folge geführt zu:

- David A. Schmidt, Bernhard Steffen. Program Analysis *as* Model Checking of Abstract Interpretations. In Proceedings of the 5th Static Analysis Symposium (SAS'98), Springer-V., LNCS 1503, 351-380, 1998.

Kapitel 19.6

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 19 (1)

- Christel Baier, Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
- Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Peit, Laure Petrucci, Philippe Schnoebelen with Pierre McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-V., 2001.
- Francesco Buccafurri, Thomas Eiter, Georg Gottlob, Nicola Leone. *Enhancing Model Checking in Verification by AI Techniques*. Artificial Intelligence 112(1-2):57-104, 1999.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 19 (2)


Edmund M. Clarke. *The Birth of Model Checking*. In *25 Years of Model Checking*. Orna Grumberg, Helmut Veith (Hrsg.), Springer-V., LNCS 5000, 1-26, 2008.


Edmund M. Clarke, Orna Grumberg, Doron Peled. *Model Checking*. MIT Press, 2001.


Edmund M. Clarke, H. Schlingloff. *Model Checking*. In Handbook of Automated Reasoning, John Alan Robinson, Andrei Voronkov (Hrsg.), Vol. II, Elsevier, 1635-1790, 2000.


Patrick Cousot, Radhia Cousot. *Temporal Abstract Interpretation*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 12-25, 2000.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 19 (3)

- E. Allen Emerson. *Temporal and Modal Logic*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Hrsg.), Elsevier, 995-1072, 1990.
- Orna Grumberg, Helmut Veith (Hrsg.). *25 Years of Model Checking: History, Achievements, Perspectives*. Springer-V., LNCS 5000, 2008.
- George E. Hughes, Max J. Cresswell. *An Introduction to Modal Logic*. Methuan, 1968.
- George E. Hughes, Max J. Cresswell. *A Companion to Modal Logic*. Methuan, 1986.
- George E. Hughes, Max J. Cresswell. *A New Introduction to Modal Logic*. Routledge, 1996.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 19 (4)

Fred Kröger, Stephan Merz. *Temporal Logic and State Systems*. Springer-V., 2008. (Chapter 3, Extensions of Linear Time Logic; Chapter 5, First-Order Linear Time Logic; Chapter 10, Other Temporal Logics; Chapter 11, System Verification by Model Checking)

Janusz Laski, William Stanley. *Software Verification and Analysis: An Integrated, Hands-On Approach*. Springer-V., 2009.

Robert Lover. *Elementary Logic for Software Development*. Springer-V., 2008. (Chapter 20.2.2, Temporal, Modal, and Dynamic Logics)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 19 (5)

- Markus Müller-Olm, David A. Schmidt, Bernhard Steffen. *Model-Checking: A Tutorial Introduction*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 330-354, 1999.
- Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019. (Chapter 6, Model Checking)
- Doron A. Peled. *Software Reliability Methods*. Springer-V., 2001.
- Dirk Richter. *Programmanalysen zur Verbesserung der Softwaremodellprüfung*. Dissertation, Universität Halle-Wittenberg, Deutschland, 2012.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 19 (6)


David A. Schmidt. *Data Flow Analysis is Model Checking of Abstract Interpretations*. In Conference Record of the 25th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'98), 38-48, 1998.


David A. Schmidt, Bernhard Steffen. *Program Analysis as Model Checking of Abstract Interpretations*. In Proceedings of the 5th Static Analysis Symposium (SAS'98), Springer-V., LNCS 1503, 351-380, 1998.


Bernhard Steffen. *Data Flow Analysis as Model Checking*. In Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'91), Springer-V., LNCS 526, 346-365, 1991.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 19 (7)


Bernhard Steffen. *Generating Data Flow Analysis Algorithms from Modal Specifications*. International Journal on Science of Computer Programming 21:115-139, 1993.


Bernhard Steffen. *Property-Oriented Expansion*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 22-41, 1996.

Kapitel 20

Modellprüfung und Abstrakte Interpretation

Kapitel 20.1

Eine Symbiose

Zustandsexplosionsproblem

...zentrale Herausforderung für Modellprüfung in praktischen Anwendungen:

- Bändigung des Zustandsexplosionsproblems.

Beachte: Die Zahl der Zustände im Zustandsraum wächst

- exponentiell in der Zahl paralleler/nebenläufiger Komponenten.
- exponentiell in der Zahl von Fallunterscheidungen schleifenfreier (!) sequentieller Programme.

Vielfältige Ansätze

...zur Zähmung des Zustandsexplosionsproblems:

- Reduktionstechniken basierend auf Prozessäquivalenzen, z.B. Bouajjani et al., 1990; Graf et al., 1996.
- Symbolische Modellprüfungstechniken, z.B. McMillan, 1993.
- On-the-fly Techniken, z.B. Jard et al., 1992.
- Lokale Modellprüfungstechniken, z.B. Stirling et al., 1991.
- Partielle Ordnungs-Techniken, z.B. Godefroid, 1996; Peled, 1993; Valmari, 1992.
- Kompositionelle Techniken, Clarke et al., 1989; Santone, 2002.
- Abstraktions-Techniken, Barbuti et al., 1999; Clarke et al., 1994.
- ...

In unserem Zusammenhang

...besonders interessant:

- Dirk Richter. Programmanalysen zur Verbesserung der Softwaremodellprüfung. Dissertation, Universität Halle-Wittenberg, Deutschland, 2012.

zur Verknüpfung und Verzahnung von Programmanalyse und Modellprüfung, speziell durch Vorschaltung

- DFA-basierter Optimierungen zur Modellverkleinerung

und damit zur

- Effizienzverbesserung anschließender Modellprüfungen.

Kapitel 20.2

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 20 (1)

Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, Irene Finocchi. *A Survey of Symbolic Execution Techniques*. ACM Computing Surveys 51(3):50:1-39, 2018.

Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, Gigliola Vaglini. *Selective Mu-Calculus and Formula-based Equivalence of Transition Systems*. Journal of Computer and System Sciences 59(3):537-556, 1999.

Ahmed Bouajjani, Jean-Claude Fernandez, Nicolas Halbwachs. *Minimal Model Generation*. In Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV'90), Springer-V., LNCS 531, 197-203, 1990.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 20 (2)


J.-H. Chow, W. L. Harrison. *State Space Reduction in Abstract Interpretation of Parallel Programs*. In Proceedings of the International Conference on Computer Languages (ICCL'94), 277-288, 1994.


Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, Roderick Bloem (Hrsg.). *Handbook of Model Checking*. Springer-V., 2018.


Edmund M. Clarke, David E. Long, Kenneth L. MacMillan. *Compositional Model Checking*. In Proceedings of the 4th Annual Symposium on Logic in Computer Science (LICS'89), IEEE Computer Society, 353-362, 1989.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 20 (3)


Edmund M. Clarke, Orna Grumberg, David E. Long. *Model Checking and Abstraction*. ACM Transactions on Programming Languages and Systems 16(5):1512-1542, 1994.


Edmund M. Clarke, Qinsi Wang: 2^5 *Years of Model Checking*. Ershov Memorial Conference 2014, 26-40, 2014.


Patrice Godefroid. *Between Testing and Verification: Dynamic Software Model Checking*. Dependable Software Systems Engineering 2016, NATO Science for Peace and Security Series - D: Information and Communication Security 45, IOS Press, 99-116, 2016.


Patrice Godefroid (Hrsg). *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*. Springer-V., LNCS 1032, 1996.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 20 (4)


Patrice Godefroid, Koushik Sen. *Combining Model Checking and Testing*. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, Roderick Bloem (Hrsg.), 613-649, 2018.


Susanne Graf, Bernhard Steffen, Gerald Lüttgen. *Compositional Minimization of Finite State Systems using Interface Specifications*. Formal Aspects of Computing 8(5):607-616, 1996.


Claude Jard, Thierry Jéron. *Bounded-memory Algorithms for Verification On-the-fly*. In Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91), Springer-V., LNCS 575, 192-202, 1992.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 20 (5)

- Kenneth L. MacMillan. *Symbolic Model Checking*. Kluwer, 1993.
- Doron Peled. *All from One, One for All: On Model Checking Using Representatives*. In Proceedings of the 5th International Workshop on Computer Aided Verification (CAV'93), Springer-V., LNCS 697, 409-423, 1993.
- Dirk Richter. *Programmanalysen zur Verbesserung der Softwaremodellprüfung*. Dissertation, Universität Halle-Wittenberg, Deutschland, 2012.
- Antonella Santone. *Automatic Verification of Concurrent Systems Using a Formula-based Compositional Approach*. Acta Informatica 38(8):531-564, 2002.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 20 (6)


Colin Stirling, David Walker. *Local Model Checking in the Modal Mu-Calculus*. Theoretical Computer Science 89(1):161-177, 1991.


Antti Valmari. *A Stubborn Attack on State Explosion*. Formal Methods in System Design 1(4):297-322, 1992.

Teil VII

Abschluss und Ausblick

Kapitel 21

Resümee, Perspektiven

Kapitel 21.1

Rückschau, Vorschau

Analyse und Verifikation (1)

...'geschafft': Ausdrucksstarke, wohlfundierte, in Werkzeuge umgesetzte, vielfach erprobte, bewährte und etablierte

- Theorie(n) für Analyse und Verifikation

mit einer Vielzahl von Ausprägungen, darunter:

- Axiomatische Verifikation
- Datenflussanalyse
- Abstrakte Interpretation
- Modellprüfung
- Theorembeweiser
- Symbolische Analyse
- Konkolische Analyse
- ...

Analyse und Verifikation (2)

...und umfangreichen und vielfältigen Erfahrungsberichten, z.B.:

- Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler. A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World. Communications of the ACM 53(2):66-75, 2010.
- Cristian Cadar, Koushik Sen. Symbolic Execution for Software Testing: Three Decades Later. Communications of the ACM 56(2):82-90, 2013.
- Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, Ciera Jaspan. Lessons from Building Static Analysis Tools at Google. Communications of the ACM 61(4):58-66, 2018.

Transformationen

...wünschenswert: Vergleichbar reiche, fundierte, praktikable

- Theorie(n) für Transformationen

im Hinblick auf

- Korrektheit, Vollständigkeit, Wirksamkeit, Optimalität

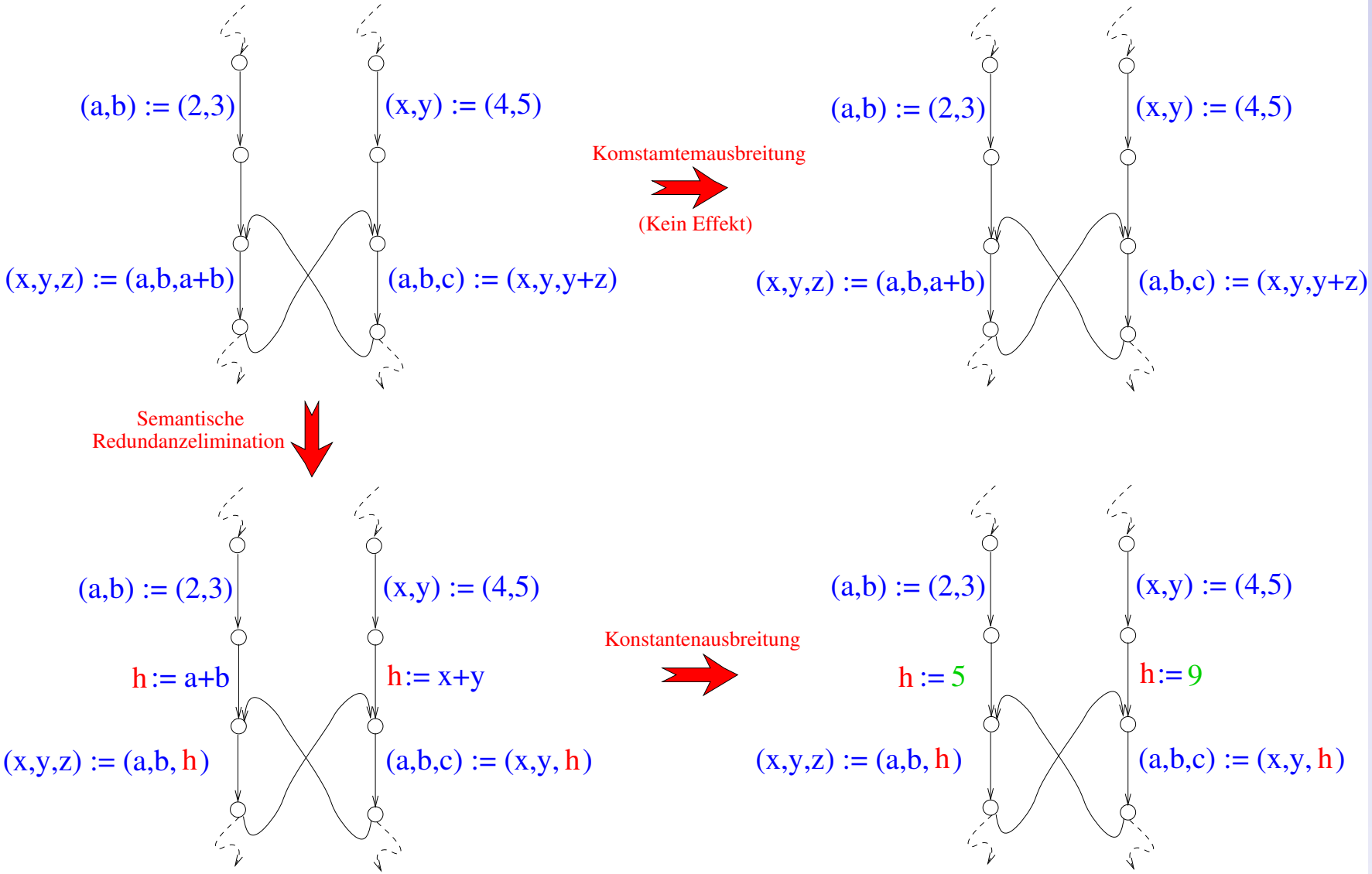
von Transformationen einschließlich Theorien und Techniken zur Evaluation mit Anwendungen insbesondere im

- Übersetzerbau (Optimierung, Parallelisierung, Portabilität, Mehrfachziele (Performanz, Speicher, Energie), Sicherheit, Schutz, Privatsphäre,...)
- Software-Technik (Modellbasierte Entwicklung und Code-Erzeugung, Refaktorisierung,...)

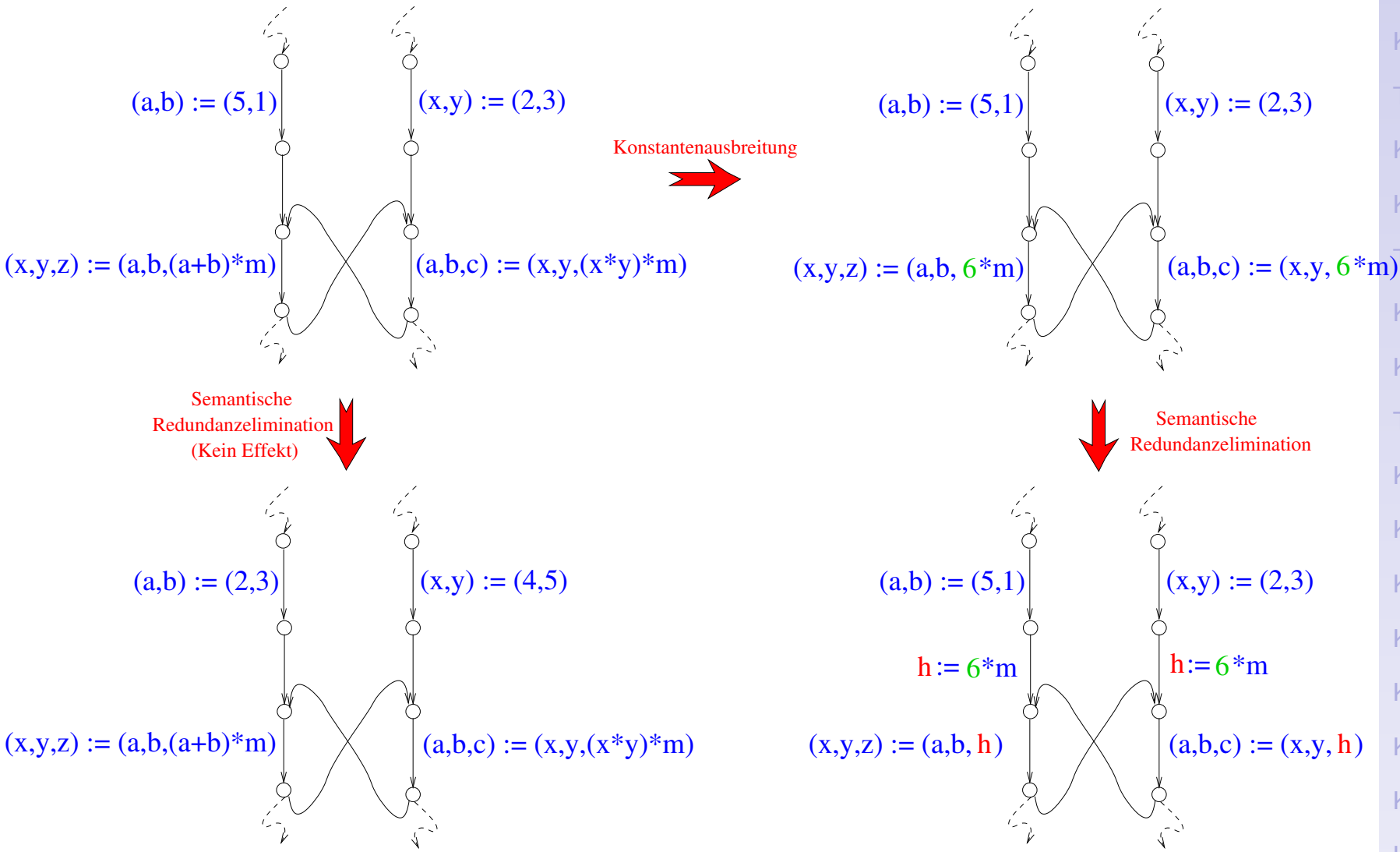
...und darüber hinaus.

Illustriert anhand v. Programmoptimierung (1)

...am Beispiel des Zusammenspiels von Optimierungen:

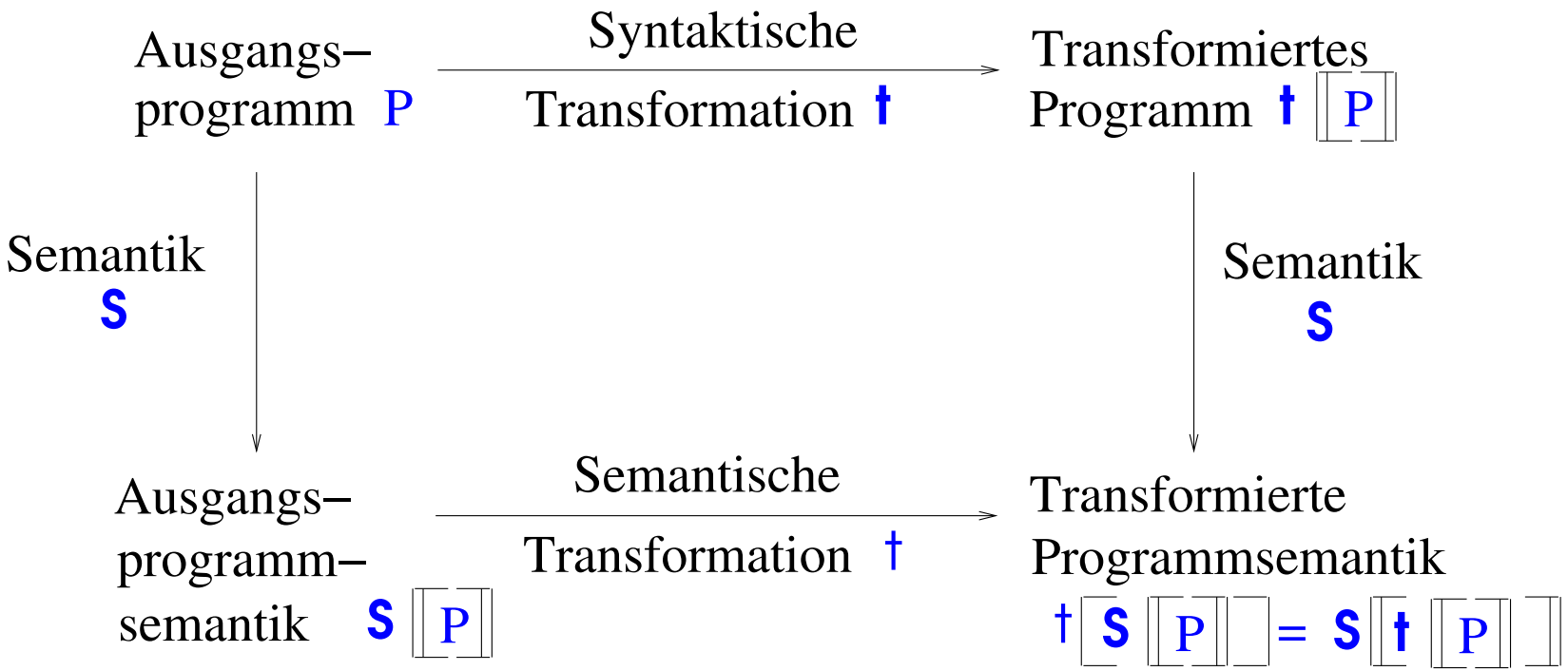


Illustriert anhand v. Programmoptimierung (2)



Analyse, Verifikation und Transformation

...bewiesen (beweisbar) korrekt und vollständig/optimal in einem einheitlichen Rahmen:



Patrick und Radhia Cousot, POPL 2002

Analyse, Verifikation, Transformation

...wichtig für Soft- und Hardware:

This software comes "without warranty of any kind, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose."

Quelle: Prototypisch

"Motorola, Inc. general policy does not recommend the use of its components in life support applications where a failure or malfunction of the component may directly threaten life or injury. Per Motorola Terms and Conditions of Sale, the user of Motorola components in life support applications assumes all risk of such use and indemnifies Motorola against all damages."

Quelle: MOTOROLA MC68020 32-BIT MICROPROCESSOR USER'S MANUAL

Wichtige Anwendungsfelder (1)

...für Analyse, Verifikation, Transformation im Bereich und Umfeld von Hard- und Software-Entwicklung.

Entwicklungsfelder

- Übersetzerbau
  - Optimierung (Performanz, Speicher, Energie, Parallelität, Portabilität,...)
  - Verifikation (Verifizierte, verifizierende Übersetzer)
- Software-Technik (dito Hardware-Technik)
  - Spezifikation, Generierung, Konfigurierung, Spezialisierung (Kundenanpassung), Verstehen (Refaktorisierung, Re-Entwurf, Rückentwurf, Dokumentation,...), Analyse, Validierung, Verifikation, Zertifizierung,...

  ...für Prozesse und Prozesskonstrukte
  - Besonders kritisch: System- und Infrastruktur-Software (Betriebssysteme, Übersetzer, Laufzeitsysteme, Dateisysteme, Browser, Kommunikationsdienste,...)

Wichtige Anwendungsfelder (2)

Anwendungsfelder

- Künstliche Intelligenz, Datenförderung und -ausbeutung, autonome Systeme, selbstorganisierende Systeme, sicherheitskritische (Echtzeit-) Systeme,...

Querschnittsfelder

- Sicherheit, Schutz, Privatsphäre, gesetzliche Vorgaben (DSGVO, finanz-, wirtschafts-, steuerrechtl. Vorgaben),...
- ...
- Grüne Informationstechnologie

All dies

...auf

- Programm-Ebene im Kleinen, System-Ebene im Großen
  - Systeme von Systemen
  - Hard- und Software-Systeme von Systemen
    - Verteilt (Service-orientiert, wolkig, mehrkernig, echtzeitig,...)
    - Eingebettet
    - Cyberphysikalisch
    - ...
- Spezifikations-, Modellierungs-, Programmier-, Zwischensprach- und Binärcode-Ebene.
- Statisch und dynamisch.

Zielführend und unverzichtbar

...fundierte

- formale (aber auch pragmatische) Methoden

wirksam unterstützt durch

- vollautomatische
  - Knopfdruckanalyse, -verifikation und -transformation
- halbautomatische
  - Interaktive, benutzergeleitete, systemgestützte, -unterstützte Analyse, Verifikation und Transformation
- hochskalierende

'Denk'-Werkzeuge auch zur

- Orchestrierung u. Ordnung von Zusammenspiel/-wirkung

über Methoden- und Aufgabengrenzen hinweg (z.B. abstrakte Interpretation, axiomatische Verifikation, Modellprüfung, Theorembeweiser, Transformatoren,...)

Sowie

...eine gute und möglichst nahtlose Verbindung verschieden-(st)er Analyse- und Verifikationstechniken unter einem gemeinsamen 'Dach':

Mario Gleirscher, Simon Foster, Jim Woodcock. *New Opportunities for Integrated Formal Methods*. ACM Computing Surveys 52(6), Article 117, 36 Seiten.

Keijiro Araki, Andy Galloway, Kenji Taguchi (Hrsg.). *Proceedings of the 1st International Conference on Integrated Formal Methods*. Springer-V., 1999.

Bernhard K Aichernig, Tom Maibaum (Hrsg.). *Formal Methods at the Crossroad. From Panacea to Foundational Support*. Springer-V., 2003.

Wichtige Konferenzen und Zeitschriften (1)

- Annual International Conference on Verification, Model-Checking, Abstract Interpretation (VMCAI) Series, Springer-V., LNCS series, seit 2000.
- Annual International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Springer-V., LNCS series, seit 1995.
- Annual International Conference on Computer-Aided Verification (CAV) Series, Springer-V., LNCS series, since 1989.
- Annual International Conference on Software Testing, Verification and Validation (ICST) Series, IEEE, seit 2008.

Wichtige Konferenzen und Zeitschriften (2)

- Annual International Symposium on Formal Methods (FM) Series, Springer-V., LNCS series, seit 1995.
- Biennial International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA) Series, Springer-V., LNCS series, seit 2004.
- International Journal on Software Tools for Technology Transfer (STTT), Springer-V, seit 1999.

...und viele mehr.

Kapitel 21.2

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 21 (1)

Uwe Aßmann. *How to Uniformly Specify Program Analysis and Transformation*. In Proceedings of the 6th International Conference on Compiler Construction (CC'96), Springer-V., LNCS 1060, 121-135, 1996.

Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis and Verification of Aerospace Software by Abstract Interpretation*. In Proceedings AIAA Infotech@Aerospace (AIAA I@A 2010), AIAA-2010-3385, American Institute of Aeronautics and Astronautics, 1-38, April 2010.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 21 (2)


Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis by Abstract Interpretation of Embedded Critical Software*. ACM Software Engineering Notes 36(1):1-8, 2011.


Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler. *A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World*. Communications of the ACM 53(2):66-75, 2010.


Cristian Cadar, Koushik Sen. *Symbolic Execution for Software Testing: Three Decades Later*. Communications of the ACM 56(2):82-90, 2013.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 21 (3)

Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.

Chris Cummins, Pavlos Petoumenos, Zheng Wang, Hugh Leather. *End-to-end Deep Learning of Optimization Heuristics*. In Proceedings of the 26th ACM/IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT 2017), 219-232, 2017.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 21 (4)


Nevin Heintze, Joxan Jaffar, Răzvan Voicu. *A Framework for Combining Analysis and Verification*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 26-39, 2000.


Sameer Kulkarni, John Cavazos. *Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning*. Proceedings of the ACM International Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2012), 147-162, 2012.


Flemming Nielson. *Program Transformations in a Denotational Setting*. ACM Transactions on Programming Languages and Systems 7:359-379, 1985.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 21 (5)


Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019. (Chapter 5, Language-Based Security)


Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, Ciera Jaspan. *Lessons from Building Static Analysis Tools at Google*. Communications of the ACM 61(4):58-66, 2018.


Ashish Tiwari, Sumit Gulwani. *Static Program Analysis Using Theorem Proving*. In Proceedings of the 21st Conference on Automated Deduction (CADE-21), LNCS 4603, Springer-V., 147-166, 2007.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 21 (6)

Franklyn Turbak, David Gifford with Mark A. Sheldon. *Design Concepts in Programming Languages*. MIT Press, 2008. (Kapitel 105, Software Testing; Kapitel 106, Formal Methods; Kapitel 107, Verification and Validation)

Daniel Weise. *Static Analysis of Mega-Programs (Invited Paper)*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 300-302, 1999.

Literaturverzeichnis

Literaturhinweise, Leseempfehlungen

.....zum vertiefenden und weiterführenden Selbststudium.

- I Lehrbücher
- II Handbücher
- III Sammelbände
- IV Dissertationen
- V Artikel
- VI Web-Ressourcen

I Lehrbücher (1)


Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. 2. Auflage, Addison-Wesley, 2007.


Randy Allen, Ken Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufman Publishers, 2002.


Krzysztof R. Apt, Ernst-Rüdiger Olderog. *Programmverifikation – Sequentielle, parallele und verteilte Programme*. Springer-V., 1994.


Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. 3. Auflage, Springer-V., 2009.


André Arnold, Irène Guessarian. *Mathematics for Computer Science*. Prentice Hall, 1996.

I Lehrbücher (2)


Christel Baier, Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.


Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 2. Auflage, Springer-V., 2001.


Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Peit, Laure Petrucci, Philippe Schnoebelen with Pierre McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-V., 2001.


Rudolf Berghammer. *Ordnungen, Verbände und Relationen mit Anwendungen*. Springer-V., 2012.


Rudolf Berghammer. *Ordnungen und Verbände: Grundlagen, Vorgehensweisen und Anwendungen*. Springer-V., 2013.

I Lehrbücher (3)


Garret Birkhoff. *Lattice Theory*. American Mathematical Society, 3. Auflage, 1967.


Edmund M. Clarke, Orna Grumberg, Doron Peled. *Model Checking*. MIT Press, 2001.


Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004.


Peter Crawley, Robert P. Dilworth. *Algebraic Theory of Lattices*. Prentice Hall, 1973.


Jaco W. De Backer. *Mathematical Theory of Program Correctness*. Prentice-Hall, 1980.


Brian A. Davey, Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge University Press, 2. Auflage, 2002.

I Lehrbücher (4)


Martin Davis. *Computability and Unsolvability*. Dover Publications, 1982.


Gilles Dowek. *Principles of Programming Languages*. Springer-V, 2009.


Marcel Erné. *Einführung in die Ordnungstheorie*. Bibliographisches Institut, 2. Auflage, 1982.


Helmuth Gericke. *Theorie der Verbände*. Bibliographisches Institut, 2. Auflage, 1967.


Michael J.C. Gordon. *The Denotational Description of Programming Languages*. Springer-V., 1979.


George Grätzer. *General Lattice Theory*. Birkhäuser, 2nd edition, 1998.

I Lehrbücher (5)


George Grätzer. *Lattice Theory: Foundation*. Birkhäuser, 2011.


Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, 1992.


Paul R. Halmos. *Naive Set Theory*. Springer-V., Wiederauflage, 2001.


Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.


Matthew Hennessey. *The Semantics of Programming Languages: An Elementary Introduction using Structural Operational Semantics*. Wiley, 1991.


Hans Hermes. *Einführung in die Verbandstheorie*. Springer-V., 2. Auflage, 1967.

I Lehrbücher (6)


John E. Hopcroft, Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.


John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 3. Auflage, Pearson, 2013.


George E. Hughes, Max J. Cresswell. *An Introduction to Modal Logic*. Methuan, 1968.


George E. Hughes, Max J. Cresswell. *A Companion to Modal Logic*. Methuan, 1986.


George E. Hughes, Max J. Cresswell. *A New Introduction to Modal Logic*. Routledge, 1996.

I Lehrbücher (7)


Richard Johnsonbaugh. *Discrete Mathematics*. Pearson, 7. Auflage, 2009.


Uday P. Khedker, Amitabha Sanyal, Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press, 2009.


Stephen C. Kleene. *Introduction to Metamathematics*. North Holland, 1952. (Wiederauflage, North Holland, 1980)


Fred Kröger, Stephan Merz. *Temporal Logic and State Systems*. Springer-V., 2008.


Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009.

I Lehrbücher (8)


Seymour Lipschutz. *Set Theory and Related Topics*. McGraw Hill Schaum's Outline Series, 2. Auflage, 1998.


Jacques Loeckx, Kurt Sieber. *The Foundations of Program Verification*. Wiley, 1984.


Robert Lover. *Elementary Logic for Software Development*. Springer-V., 2008.


Kenneth L. MacMillan. *Symbolic Model Checking*. Kluwer, 1993.


David Makinson. *Sets, Logic and Maths for Computing*. Springer-V., 2008.


Robert Morgan. *Building an Optimizing Compiler*. Digital Press, 1998.

I Lehrbücher (9)


Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997.


Flemming Nielson, Hanne Riis Nielson. *Formal Methods: An Appetizer*. Springer-V., 2019.


Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. Springer-V., 2. Auflage, 2005.


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992.


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007.


Doron A. Peled. *Software Reliability Methods*. Springer-V., 2001.

I Lehrbücher (10)


Steven Roman. *Lattices and Ordered Sets*. Springer-V., 2008.


David A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn & Bacon, 1986.


Bernhard Steffen, Oliver Rüthing, Malte Isberner. *Grundlagen der höheren Informatik: Induktives Vorgehen*. Springer-V., 2014.


Bernhard Steffen, Oliver Rüthing, Michael Huth. *Mathematical Foundations of Advanced Informatics: Inductive Approaches*. Springer-V., 2018.


Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1981.

I Lehrbücher (11)


Robert D. Tennent. *Semantics of Programming Languages*. Prentice Hall, 1991.


Franklyn Turbak, David Gifford with Mark A. Sheldon. *Design Concepts in Programming Languages*. MIT Press, 2008.


Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

II Handbücher


Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, Roderick Bloem (Hrsg.). *Handbook of Model Checking*. Springer-V., 2018.


Jan van Leeuwen (Hrsg.). *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B. V., 1990.


Peter Rechenberg, Gustav Pomberger (Hrsg.). *Informatik-Handbuch*. 4. Auflage, Carl Hanser Verlag, 2006.


John Alan Robinson, Andrei Voronkov (Hrsg.). *Handbook of Automated Reasoning*. Vol. II, Elsevier, 2000.


Samson Abramsky, Dov M. Gabbay, Thomas S.E. Maibaum (Hrsg.). *Handbook of Logic in Computer Science*. Volume 4, Oxford University Press, 1995.

III Sammelbände (1)


Samson Abramsky, Chris Hankin (Hrsg.). *Abstract Interpretation of Declarative Languages*. Prentice Hall, 1987.


Roland Backhouse, Roy Crole, Jeremy R. Gibbons (Hrsg.). *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*. International Summer School and Workshop, Oxford, UK, April 10-14, 2000, Revised Lectures, Springer-V., LNCS 2297, 2002.


Bernhard Beckert, Reiner Hähnle, Peter H. Schmitt (Hrsg.). *Verification of Object-Oriented Software: The KeY Approach*. LNCS 4334, Springer-V., 2007.


Patrice Godefroid (Hrsg.). *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*. Springer-V., LNCS 1032, 1996.

III Sammelbände (2)


George Grätzer, Friedrich Wehrung (Hrsg.). *Lattice Theory: Special Topics and Applications, Vol. I*. Birkhäuser, 2014.


George Grätzer, Friedrich Wehrung (Hrsg.). *Lattice Theory: Special Topics and Applications, Vol. II*. Birkhäuser, 2016.


Orna Grumberg, Helmut Veith (Hrsg.). *25 Years of Model Checking: History, Achievements, Perspectives*. Springer-V., LNCS 5000, 2008.


Nachum Dershowitz (Hrsg.). *Verification: Theory and Practice*. Essays dedicated to Zohar Manna on the Occasion of His 64th Birthday. Springer-V., LNCS 2772, 243-268, 2003.

IV Dissertationen


Evelyn Duesterwald. *A Demand-driven Approach for Efficient Interprocedural Data-Flow Analysis*. PhD thesis, University of Pittsburgh, PA, USA, 1996.


Hanne Riis Nielson. *Hoare Logic's for Run-time Analysis of Programs*. PhD thesis, Edinburgh University, UK, 1984.


Dirk Richter. *Programmanalysen zur Verbesserung der Softwaremodellprüfung*. Dissertation, Universität Halle-Wittenberg, Deutschland, 2012.

V Artikel (1)


Samson Abramsky, Chris Hankin. *An Introduction to Abstract Interpretation*. In 'Abstract Interpretation of Declarative Languages,' Samson Abramsky, Chris Hankin (Hrsg.). Prentice Hall, 63-102, 1987.


Gagan Agrawal. *Demand-driven Construction of Call Graphs*. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 125-140, 2000.


Frances E. Allen, John A. Cocke. *A Program Data Flow Analysis Procedure*. Communications of the ACM 19(3):137-147, 1976.

V Artikel (2)


Bowen Alpern, Mark N. Wegman, F. Kenneth Zadeck. *Detecting Equality of Variables in Programs*. In Conference Record of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'88), 1-11, 1988.


Krzysztof R. Apt. *Ten Years of Hoare's Logic: A Survey – Part 1*. ACM Transactions on Programming Languages and Systems 3(4):431-483, 1981.


Krzysztof R. Apt. *Ten Years of Hoare's Logic: A Survey – Part II: Nondeterminism*. Theoretical Computer Science 28(1-2):83-109, 1984.

V Artikel (3)


Uwe Aßmann. *How to Uniformly Specify Program Analysis and Transformation*. In Proceedings of the 6th International Conference on Compiler Construction (CC'96), Springer-V., LNCS 1060, 121-135, 1996.


Philip Axer, Rolf Ernst, Heiko Falk, Alain Girault, Daniel Grund, Nan Guan, Bengt Jonsson, Peter Marwedel, Jan Reineke, Christine Rochange, Maurice Sebastian, Reinhard von Hanxleden, Reinhard Wilhelm, Wang Yi. *Building Timing Predictable Embedded Systems*. ACM Transactions on Embedded Computing Systems 13(4):82, 2014.


Wayne A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part I - Exhaustive Analysis*. Acta Informatica 10(3):245-264, 1978.

V Artikel (4)


Wayne A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part II - Demand Analysis*. Acta Informatica 10(3):265-272, 1978.


Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, Irene Finocchi. *A Survey of Symbolic Execution Techniques*. ACM Computing Surveys 51(3):50:1-39, 2018.


Clément Ballabriga, Hugues Cassé, Christine Rochange, Pascal Sainrat. *OTAWA: An Open Toolbox for Adaptive WCET Analysis*. In Proceedings SEUS 2010, Springer-V., 35-46, 2010.

V Artikel (5)

Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, Gigliola Vaglini. *Selective Mu-Calculus and Formula-based Equivalence of Transition Systems*. Journal of Computer and System Sciences 59(3):537-556, 1999.

Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis and Verification of Aerospace Software by Abstract Interpretation*. In Proceedings AIAA Infotech@Aerospace (AIAA I@A 2010), AIAA-2010-3385, American Institute of Aeronautics and Astronautics, 1-38, April 2010.

V Artikel (6)


Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis by Abstract Interpretation of Embedded Critical Software*. ACM Software Engineering Notes 36(1):1-8, 2011.


Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler. *A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World*. Communications of the ACM 53(2):66-75, 2010.


Garret Birkhoff. *Applications of Lattice Algebra*. Mathematical Proceedings of the Cambridge Philosophical Society 30(2):115-122, 1934.

V Artikel (7)

Stephen M. Blackburn, Amer Diwan, Matthias Hauswirth, Peter F. Sweeny, José Nelson Amaral, Tim Brecht, Lubomír Bulej, Cliff Click, Lieven Eeckhout, Sebastian Fischmeister, Daniel Frampton, Laurie J. Hendren, Michael Hind, Antony L. Hosking, Richard E. Jones, Tomas Kalibera, Nathan Keynes, Nathaniel Nystrom, Andreas Zeller. *The Truth, The Whole Truth, and Nothing But the Truth: A Pragmatic Guide to Assessing Empirical Evaluations*. ACM Transactions on Programming Languages and Systems 38(4), Article 15:1-20, 2016.

V Artikel (8)


Ras Bodik, Rajiv Gupta. *Partial Dead Code Elimination using Slicing Transformations*. In Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation (PLDI'97), ACM SIGPLAN Notices 32(6):159-170, 1997.


Ras Bodík, Rajiv Gupta, Vivek Sarkar. *ABCD: Eliminating Array Bounds Check on Demand*. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'00), ACM SIGPLAN Notices 35(5):321-333, 2000.


Armelle Bonenfant, Hugues Cassé, Marianne De Michiel, Jens Knoop, Laura Kovács, Jakob Zwirchmayr. *FFX: A Portable WCET Annotation Language*. In Proceedings of the 20th International Conference on Real-Time and Network Systems (RTNS 2012), ACM, 91-100, 2012.

V Artikel (9)


Ahmed Bouajjani, Jean-Claude Fernandez, Nicolas Halbwachs. *Minimal Model Generation*. In Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV'90), Springer-V., LNCS 531, 197-203, 1990.


Nicolas Bourbaki. *Sur la théorème de Zorn*. Archiv der Mathematik 2:434-437, 1949/50.


Francesco Buccafurri, Thomas Eiter, Georg Gottlob, Nicola Leone. *Enhancing Model Checking in Verification by AI Techniques*. Artificial Intelligence 112(1-2):57-104, 1999.


Cristian Cadar, Koushik Sen. *Symbolic Execution for Software Testing: Three Decades Later*. Communications of the ACM 56(2):82-90, 2013.

V Artikel (10)


Larry Carter, Jeanne Ferrante, Clark Thomborson. *Folklore Confirmed: Reducible Flow Graphs are Exponentially Larger*. In Conference Record of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2003), 106-114, 2003.


Jyh-Herng Chow, William L. Harrison. *Compile Time Analysis of Parallel Programs that share Memory*. In Conference Record of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'92), 130-141, 1992.


Jyh-Herng Chow, William L. Harrison. *State Space Reduction in Abstract Interpretation of Parallel Programs*. In Proceedings of the International Conference on Computer Languages (ICCL'94), 277-288, 1994.

V Artikel (11)


Edmund M. Clarke. *Programming Language Constructs for which it is Impossible to Obtain Good Hoare Axiom Systems*. Journal of the ACM 26(1):129-147, 1979.


Edmund M. Clarke. *The Birth of Model Checking*. In *25 Years of Model Checking*. Orna Grumberg, Helmut Veith (Hrsg.), Springer-V., LNCS 5000, 1-26, 2008.


Edmund M. Clarke, Stephen M. German, Joseph Y. Halpern. *Effective Axiomatizations of Hoare Logics*. Journal of the ACM 30(1):612-636, 1983.


Edmund M. Clarke, Orna Grumberg, David E. Long. *Model Checking and Abstraction*. ACM Transactions on Programming Languages and Systems 16(5):1512-1542, 1994.

V Artikel (12)


Edmund M. Clarke, David E. Long, Kenneth L. MacMillan. *Compositional Model Checking*. In Proceedings of the 4th Annual Symposium on Logic in Computer Science (LICS'89), IEEE Computer Society, 353-362, 1989.


Edmund M. Clarke, H. Schlingloff. *Model Checking*. In *Handbook of Automated Reasoning*, John Alan Robinson, Andrei Voronkov (Hrsg.), Vol. II, Elsevier, 1635-1790, 2000.


Edmund M. Clarke, Qinsi Wang: 2^5 *Years of Model Checking*. Ershov Memorial Conference 2014, 26-40, 2014.


Dominique Clemént, Joëlle Despeyroux, Thierry Despeyroux, L. Hascoet, Gilles Kahn. *Natural Semantics on the Computer*. INRIA Research Report RR 416, INRIA, Sophia-Antipolis, June 1985.

V Artikel (13)


Dominique Clemént, Joëlle Despeyroux, Thierry Despeyroux, Gilles Kahn. *A Simple Applicative Language: Mini-ML*. In Proceedings of the International ACM Conference on Lisp and Functional Programming (LFP'86), 13-27, 1986.


Ernie Cohen, Dexter Kozen. *A Note on the Complexity of Propositional Hoare Logic*. ACM Transactions on Computational Logic 1(1):171-174, 2000.


Stephen A. Cook. *Soundness and Completeness of an Axiom System for Program Verification*. SIAM Journal on Computing 7(1):70-90, 1978.


Patrick Cousot. *Methods and Logics for Proving Programs*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Hrsg.), Elsevier Science Publishers B. V., Kapitel 15, 841-993, 1990.

V Artikel (14)


Patrick Cousot. *Abstract Interpretation*. ACM Computing Surveys 28(2):324-328, 1996.


Patrick Cousot. *Refining Model-Checking by Abstract Interpretation*. Automated Software Engineering 6(1):69-95, 1999.


Patrick Cousot. *Design of Syntactic Program Transformations by Abstract Interpretation of Semantic Transformations*. In Proceedings of the 17th International Conference on Logic Programming (ICLP 2001), Springer-V., LNCS 2237, 4-5, 2001.


Patrick Cousot. *The Verification Grand Challenge and Abstract Interpretation*. In Proceedings of Verified Software: Theories, Tools, Experiments (VSTTE 2005), Springer-V, LNCS 4171, 189-201, 2005.

V Artikel (15)


Patrick Cousot. *Verification by Abstract Interpretation*. In Proceedings of the 4th International Conference on Verification, Model-Checking, Abstract Interpretation (VMCAI 2003), Springer-V., LNCS 2575, 20-24, 2003.


Patrick Cousot. *Verification by Abstract Interpretation*. In Verification: Theory and Practice, Essays dedicated to Zohar Manna on the Occasion of His 64th Birthday. Springer-V., LNCS 2772, 243-268, 2003.


Patrick Cousot, Radhia Cousot. *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In Conference Record of the 4th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77), 238-252, 1977.

V Artikel (16)


Patrick Cousot, Radhia Cousot. *Constructive Versions of Tarski's Fixed Point Theorems*. Pacific Journal of Mathematics 82(1):43-57, 1979.


Patrick Cousot, Radhia Cousot. *Systematic Design of Program Analysis Frameworks*. In Conference Record of the 6th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'79), 269-282, 1979.


Patrick Cousot, Radhia Cousot. *Invariance Proof Methods and Analysis Techniques for Parallel Programs*. In Automatic Program Construction Techniques, A. W. Biermann, G. Guiho, Y. Kodratoff (Hrsg.), Macmillan Publishing Company, Kapitel 12, 243-271, 1984.

V Artikel (17)


Patrick Cousot, Radhia Cousot. *Abstract Interpretation Frameworks*. Journal of Logic and Computation 2(4):511-547, 1992.


Patrick Cousot, Radhia Cousot. *Temporal Abstract Interpretation*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 12-25, 2000.


Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.

V Artikel (18)


Patrick Cousot, Radhia Cousot. *A Gentle Introduction to Formal Verification of Computer Systems by Abstract Interpretation*. In Logics and Languages for Reliability and Security. NATO Science for Peace and Security - D; Information and Communication Security, Vol. 25, IOS Press, 2010. ISBN 978-1-60750–099-5.


Patrick Cousot, Radhia Cousot, Laurent Mauborgne. *Theories, Solvers and Static Analysis by Abstract Interpretation*. Journal of the ACM 59(6), Artikel 31, 56 Seiten, 2012.


Patrick Cousot, Michael Monerau. *Probabilistic Abstract Interpretation*. In Proceedings of the 21st Symposium on Programming (ESOP 2012), Springer-V., LNCS 7211, 169-193, 2012.

V Artikel (19)

Chris Cummins, Pavlos Petoumenos, Zheng Wang, Hugh Leather. *End-to-end Deep Learning of Optimization Heuristics*. In Proceedings of the 26th ACM/IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT 2017), 219-232, 2017.

Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck. *An Efficient Method of Computing Static Single Assignment Form*. In Conference Record of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'89), 25-35, 1989.

V Artikel (20)


Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck. *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph*. ACM Transactions on Programming Languages and Systems 13(4):451-490, 1991.


Marvin Damschen, Lars Bauer, Jörg Henkel. *Timing Analysis of Tasks on Runtime Reconfigurable Processors*. In IEEE Transactions on Very Large Scale Integration Systems 25(1):294-307, 2017.


Anne C. Davis. *A Characterization of Complete Lattices*. Pacific Journal of Mathematics 5(2):311-319, 1955.


Martin Davis. *Hilbert's Tenth Problem is Unsolvable*. American Mathematical Monthly 80:33-269, 1973.

V Artikel (21)


Martin Davis, Yuri Matijasevič, Julia Robinson. *Hilbert's Tenth Problem. Diophantine Equations: Positive Aspects of a Negative Solution*. In Proceedings of the Symposium on the Hilbert Problems (De Kalb, Illinois), May 1974, American Mathematical Society, Providence, R.I., 323-378, 1976.


Joëlle Despeyroux. *Proof of Translation in Natural Semantics*. In Proceedings of the 2nd International IEEE Symposium on Logic in Computer Science (LICS'86), 193-205, 1986.


Thierry Despeyroux. *Typol: A Formalism to Implement Natural Semantics*. INRIA Research Report 94, Roquencourt, France, 1988.

V Artikel (22)


Dhananjay M. Dhamdhere. *Register Assignment using Code Placement Techniques*. Journal of Computer Languages 13(2):75-93, 1988.


Dhananjay M. Dhamdhere. *A usually linear Algorithm for Register Assignment using Edge Placement of Load and Store Instructions*. Journal of Computer Languages 15(2):83-94, 1990.


Dhananjay M. Dhamdhere. *Practical Adaptation of the Global Optimization Algorithm of Morel and Renvoise*. ACM Transactions on Programming Languages and Systems 13(2):291-294, 1991. Technical Correspondence.

V Artikel (23)


Dhananjay M. Dhamdhere, Barry K. Rosen, F. Kenneth Zadeck. *How to Analyze Large Programs Efficiently and Informatively*. In Proceedings of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation (PLDI'92), ACM SIGPLAN Notices 27(7):212-223, 1992.


Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *Demand-driven Computation of Interprocedural Data Flow*. In Conference Record of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95), 37-48, 1995.


Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Demand-driven Analyzer for Data Flow Testing at the Integration Level*. In Proceedings of the IEEE Conference on Software Engineering (CoSE'96), 575-586, 1996.

V Artikel (24)


Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Practical Framework for Demand-driven Interprocedural Data Flow Analysis*. ACM Transactions on Programming Languages and Systems 19(6):992-1030, 1997.


Matthew B. Dwyer, Lori A. Clarke. *Data Flow Analysis for Verifying Properties of Concurrent Programs*. In Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering (SFSE'94), Software Engineering Notes 19(5):62-75, 1994.


Matthew B. Dwyer, Lori A. Clarke, Jamieson M. Cobleigh, Gleb Naumovich. *Flow Analysis for Verifying Properties of Concurrent Software Systems*. ACM Transactions on Software Engineering Methodology 13(4):359-430, 2004.

V Artikel (25)


Stephen A. Edwards, Edward A. Lee. *The Case for the Precision-timed (PRET) Machine*. In Proceedings of the 44th Design Automation Conference (DAC 2007), 264-265, 2007.


E. Allen Emerson. *Temporal and Modal Logic*. In *Handbook of Theoretical Computer Science*, Jan van Leeuwen (Hrsg.), Elsevier, 995-1072, 1990.


Leandro Faccinetti, Zachary Palmer, Scott Smith. *Higher-order Demand-driven Program Analysis*. ACM Transactions on Programming Languages and Systems (TOPLAS) 41(3):14:Computing Surveys 51(3):14:1-53, 2019.

V Artikel (26)


Christian Fecht, Helmut Seidl. *An Even Faster Solver for General Systems of Equations*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 189-204, 1996.


Christian Fecht, Helmut Seidl. *Propagating Differences: An Efficient New Fixpoint Algorithm for Distributive Constraint Systems*. In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 90-104, 1998.


Christian Fecht, Helmut Seidl. *A Faster Solver for General Systems of Equations*. Science of Computer Programming 35(2):137-161, 1999.

V Artikel (27)


L. Feigen, D. Klappholz, R. Casazza, X. Xue. *The Revival Transformation*. In Conference Record of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'94), 1994.


Jeanne Ferrante, Dirk Grunwald, Harini Srinivasan. *Compile-time Analysis and Optimization of Explicitly Parallel Programs*. Parallel Algorithms and Applications 12(1-3):21-56, 1997.


Robert W. Floyd. *Assigning Meaning to Programs*. In Proceedings of Symposium on Applied Mathematics, Mathematical Aspects of Computer Science, American Mathematical Society, New York, 19:19-32, 1967.

V Artikel (28)


Emily P. Friedman. *Relationships between Monadic Recursion Schemes and Deterministic Context-free Languages*. In Conference Record of the 15th Annual IEEE Symposium on Switching and Automata Theory (SWAT'74), 43-51, 1974.


Emily P. Friedman. *Equivalence Problems for Deterministic Context-free Languages and Monadic Recursion Schemes*. Journal of Computer and System Sciences 14(3):344-359, 1977.


Stephen J. Garland, David C. Luckham. *Program Schemes, Recursion Schemes, and Formal Languages*. Journal of Computer and System Sciences 7(2):119-160, 1973.

V Artikel (29)


Alfons Geser, Jens Knoop, Gerald Lüttgen, Oliver Rüthing, Bernhard Steffen. *Non-monotone Fixpoint Iterations to Resolve Second Order Effects*. In Proceedings of the 6th International Conference on Compiler Construction (CC'96), Springer-V., LNCS 1060, 106-120, 1996.


Seymour Ginsburg, Sheila Greibach. *Deterministic Context Free Languages*. Information and Control 9(6):620-648, 1966.


Patrice Godefroid. *Between Testing and Verification: Dynamic Software Model Checking*. Dependable Software Systems Engineering 2016, NATO Science for Peace and Security Series - D: Information and Communication Security 45, IOS Press, 99-116, 2016.

V Artikel (30)

Patrice Godefroid, Koushik Sen. *Combining Model Checking and Testing*. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, Roderick Bloem (Hrsg.), 613-649.

Susanne Graf, Bernhard Steffen, Gerald Lüttgen. *Compositional Minimization of Finite State Systems using Interface Specifications*. Formal Aspects of Computing 8(5):607-616, 1996.

V Artikel (31)


Dirk Grunwald, Harini Srinivasan. *Data Flow Equations for Explicitely Parallel Programs*. In Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'93), ACM SIGPLAN Notices 28(7):159-168, 1993.


Jan Gustafsson. *Usability Aspects of WCET Analysis*. In Proceedings of the 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2008), 346-352, 2008.


Jan Gustafsson, Adam Betts, Andreas Ermedahl, Björn Lisper. *The Mälardalen WCET Benchmarks: Past, Present, and Future*. In Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010), 136-146, 2010.

V Artikel (32)


Nevin Heintze, Joxan Jaffar, Răzvan Voicu. *A Framework for Combining Analysis and Verification*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 26-39, 2000.


Charles A.R. Hoare. *An Axiomatic Basis for Computer Programming*. Communications of the ACM 12(10):576-580, 583, 1969.


Charles A.R. Hoare. *The Emperor's Old Clothes*. Communications of the ACM 24(2):75-83, 1981. DOI: 10.1145/358549.358561


Charles A.R. Hoare. *The Ideal of Program Correctness*. The Computer Journal 50(3):254-260, 2007.

V Artikel (33)


Charles A.R. Hoare. *Retrospective: An Axiomatic Basis for Computer Programming*. Communications of the ACM 52(10):30-32, 2009. DOI: 10.1145/1562764.1562779


Susan Horwitz, Alan J. Demers, Tim Teitelbaum. *An Efficient General Iterative Algorithm for Dataflow Analysis*. Acta Informatica 24(6):679-694, 1987.


Susan Horwitz, Thomas Reps, Mooly Sagiv. *Demand Interprocedural Data Flow Analysis*. In Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-3), 104-115, 1995.


John Hughes, John Launchbury. *Reversing Abstract Interpretations*. In Proceedings of the 4th European Symposium on Programming (ESOP'92), Springer-V., LNCS 582, 269-286, 1992.

V Artikel (34)


John Hughes, John Launchbury. *Reversing Abstract Interpretations*. Science of Computer Programming 22:307-326, 1994.


Claude Jard, Thierry Jéron. *Bounded-memory Algorithms for Verification On-the-fly*. In Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91), Springer-V., LNCS 575, 192-202, 1992.


Tudor Jebelean, Laura Kovács, Nikolaj Popov. *Experimental Program Verification in the Theorema System*. In Proceedings of the International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004), 92-99, 2004. `www.risc.jku.at/publications/download/risc_2243/KoPoJeb.pdf`

V Artikel (35)


Neil D. Jones, Flemming Nielson. *Abstract Interpretation: A Semantics-based Tool for Program Analysis*. In Handbook of Logic in Computer Science, Volume 4, Oxford University Press, 1995.


Gilles Kahn. *Natural Semantics*. In Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS'87), Springer-V., LNCS 247, 22-39, 1987.


John B. Kam, Jeffrey D. Ullman. *Global Data Flow Analysis and Iterative Algorithms*. Journal of the ACM 23:158-171, 1976.


John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.

V Artikel (36)


Gary A. Kildall. *A Unified Approach to Global Program Optimization*. In Conference Record of the 1st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'73), 194-206, 1973.


Raimund Kirner, Jens Knoop, Adrian Prantl, Markus Schordan, Albrecht Kadlec. *Beyond Loop Bounds: Comparing Annotation Languages for Worst-Case Execution Time Analysis*. Journal of Software and Systems Modeling 10(3):411-437, Springer-V., 2011.


Marion Klein, Jens Knoop, Dirk Koschützki, Bernhard Steffen. *DFA&OPT-METAFrame: A Toolkit for Program Analysis and Optimization*. In Proceedings of the 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96), Springer-V., LNCS 1055, 422-426, 1996.

V Artikel (37)


Jens Knoop. *Parallel Constant Propagation*. In Proceedings of the 4th European Conference on Parallel Processing (Europar'98), Springer-V., LNCS 1470, 445-455, 1998.


Jens Knoop. *From DFA-frameworks to DFA-generators: A Unifying Multiparadigm Approach*. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Springer-V., LNCS 1579, 360-374, 1999.


Jens Knoop. *Demand-driven Analysis of Explicitly Parallel Programs: An Approach based on Reverse Data-Flow Analysis*. In Proceedings of the 9th International Workshop on Compilers for Parallel Computers (CPC 2001), 151-162, 2001.

V Artikel (38)


Jens Knoop. *Data-Flow Analysis for Hot-Spot Program Optimization*. In Proceedings of the 14th Biennial Workshop on 'Programmiersprachen und Grundlagen der Programmierung' (KPS 2007). Bericht A-07-07 der Institute für Mathematik und Informatik, Universität Lübeck, Deutschland, 124-131, 2007.


Jens Knoop, Dirk Koschützki, Bernhard Steffen. *Basic-block Graphs: Living Dinosaurs?* In Proceedings of the 7th International Conference on Compiler Construction (CC'98), Springer-V., LNCS 1383, 65-79, 1998.


Jens Knoop, Eduard Mehofer. *Distribution Assignment Placement: Effective Optimization of Redistribution Costs*. IEEE Transactions on Parallel and Distributed Systems 13(6):628-647, 2002.

V Artikel (39)


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Lazy Code Motion*. In Proceedings of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation (PLDI'92), ACM SIGPLAN Notices 27(7):224-234, 1992.


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Partial Dead Code Elimination*. In Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI'94), ACM SIGPLAN Notices 29(6):147-158, 1994.


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Optimal Code Motion: Theory and Practice*. ACM Transactions on Programming Languages and Systems 16(4):1117-1155, 1994.

V Artikel (40)


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *The Power of Assignment Motion*. In Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI'95), ACM SIGPLAN Notices 30(6):233-245, 1995.


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Code Motion and Code Placement: Just Synonyms?* In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 154-169, 1998.


Jens Knoop, Oliver Rüthing. *Constant Propagation on the Value Graph: Simple Constants and Beyond*. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 94-109, 2000.

V Artikel (41)


Jens Knoop, Oliver Rüthing. *Constant Propagation on Predicated Code*. Journal of Universal Computer Science 9(8):829-850, 2003. (special issue devoted to SBLP'03).


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Retrospective: Lazy Code Motion*. In '20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation (1979 - 1999): A Selection,' ACM SIGPLAN Notices 39(4):460-461&462-472, 2004.


Jens Knoop, Bernhard Steffen. *The Interprocedural Coincidence Theorem*. In Proceedings of the 4th International Conference on Compiler Construction (CC'92), Springer-V., LNCS 641, 125-140, 1992.

V Artikel (42)


Jens Knoop, Bernhard Steffen. *Code Motion for Explicitly Parallel Programs*. In Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99), ACM SIGPLAN Notices 34(8):13-24, 1999.


Jens Knoop, Bernhard Steffen, Jürgen Vollmer. *Parallelism for Free: Bitvector Analyses* $\rightarrow$ *No State Explosion!* In Proceedings of the 1st International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95), LNCS 1019, Springer-V., 264-289, 1995.


Jens Knoop, Bernhard Steffen, Jürgen Vollmer. *Parallelism for Free: Efficient and Optimal Bitvector Analyses for Parallel Programs*. ACM Transactions on Programming Languages and Systems 18(3):268-299, 1996.

V Artikel (43)


Laura Kovács, Tudor Jebelean. *Practical Aspects of Imperative Program Verification using Theorema*. In Proceedings of the 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2003), 317-320, 2003. www.risc.jku.at/publications/download/risc_464/synasc03.pdf


Laura Kovács, Tudor Jebelean. *Generation of Invariants in Theorema*. In Proceedings of the 10th International Symposium of Mathematics and its Applications, 407-415, 2003. www.risc.jku.at/publications/download/risc_2053/2003-11-06-A.pdf


Dexter Kozen, Jerzy Tiuryn. *On the Completeness of Propositional Hoare Logic*. Information Sciences 139(3-4):187-195, 2001.

V Artikel (44)


Sameer Kulkarni, John Cavazos. *Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning*. Proceedings of the ACM International Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2012), 147-162, 2012.


William Landi. *Undecidability of Static Analysis*. ACM Letters on Programming Languages and Systems 1(4):323-337, 1992.


Jean-Louis Lassez, V.L. Nguyen, Elizabeth A. Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.

V Artikel (45)


Thomas Leveque, Etienne Borde, Amine Marref, Jan Carlson. *Hierarchical Composition of Parametric WCET in a Component Based Approach*. In Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2011), 261-268, 2011.


Yau-Tsun Steven Li, Sharad Malik. *Performance Analysis of Embedded Software using Implicit Path Enumeration*. ACM SIGPLAN Notices 30(11):88-98, 1995.


Yuan Lin, David A. Padua. *Demand-driven Interprocedural Array Property Analysis*. In Proceedings of the 12th International Conference on Languages and Compilers for Parallel Computing (LCPC'99), Springer-V., LNCS 1863, 303-317, 1999.

V Artikel (46)


Björn Lisper, Andreas Ermedahl, Dietmar Schreiner, Jens Knoop, Peter Gliwa. *Practical Experiences of Applying Source-level WCET Flow Analysis to Industrial Code*. Journal of Software Tools for Technology Transfer (STTT) 15(1):53-63, Springer-V., 2013.


Konstantinos Mamouras. *On the Hoare Theory of Monadic Recursion Schemes*. In Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (CSL-LICS'14), Article 69, 69.1-69.10, 2014.


Konstantinos Mamouras. *The Hoare Logic of Deterministic and Nondeterministic Monadic Recursion Schemes*. ACM Transactions on Computational Logic 17(2):13.1-13.30, 2016.

V Artikel (47)


George Markowsky. *Chain-complete Posets and Directed Sets with Applications*. Algebra Universalis 6(1):53-68, 1976.


Thomas J. Marlowe, Barbara G. Ryder. *Properties of Data Flow Frameworks*. Acta Informatica 20:121-163, 1990.


Kim Marriot. *Frameworks for Abstract Interpretation*. Acta Informatica 30:103-129, 1993.


Florian Martin. *PAG - An Efficient Program Analyzer Generator*. Journal of Software Tools for Technology Transfer 2(1):46-67, 1998.


Stephen P. Masticola, Thomas J. Marlowe, Barbara G. Ryder. *Lattice Frameworks for Multisource and Bidirectional Data Flow Problems*. ACM Transactions on Programming Languages and Systems (TOPLAS) 17(5):777-803, 1995.

V Artikel (48)

- Yuri V. Matijasevič. *Enumerable Sets are Diophantine (auf Russisch)*. Doklady Akademii Nauk SSSR 191:279-282, 1970 (englische Übersetzung: Soviet Mathematics Doklady 11:354-357, 1970).
- Yuri V. Matijasevič. *On Recursive Unsolvability of Hilbert's Tenth Problem*. In Proceedings of the 4th International Congress on Logic, Methodology and Philosophy of Science (Bucharest 1971), North-Holland, Amsterdam, 89-110, 1973.
- Yuri V. Matijasevič. *What Should We Do Having Proved a Decision Problem to be Unsolvable?* In Proceedings of Algorithms in Modern Mathematics and Computer Science, Springer-V., LNCS 122, 441-448, 1979.

V Artikel (49)

- Yuri V. Matijasevič. *Hilbert's Tenth Problem*. MIT Press, 1993.
- Samuel P. Midkiff, José E. Moreira, Marc Snir. *A Constant Propagation Algorithm for Explicitly Parallel Programs*. International Journal of Computer Science 26(5):563-589, 1998.
- Samuel P. Midkiff, David A. Padua. *Issues in the Optimization of Parallel Programs*. In Proceedings of the 18th International Conference on Parallel Processing (ICPP'90), Vol. II., 105-113, 1990.
- Steve P. Miller, Michael W. Whalen, Darren D. Cofer. *Software Model Checking Takes Off*. Communications of the ACM 53(2):58-64, 2010.

V Artikel (50)


Ronald J. Mintz, Gerald A. Fisher, Micha Sharir. *The Design of a Global Optimizer*. In Proceedings of the ACM SIGPLAN'79 Symposium on Compiler Construction (SoCC'79), ACM SIGPLAN Notices 14(8):226-234, 1979.


Markus Müller-Olm, David A. Schmidt, Bernhard Steffen. *Model-Checking: A Tutorial Introduction*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 330-354, 1999.


Markus Müller-Olm, Helmut Seidl. *Polynomial Constants are Decidable*. In Proceedings of the 9th Static Analysis Symposium (SAS 2002), Springer-V., LNCS 2477, 4-19, 2002.

V Artikel (51)


Flemming Nielson. *Program Transformations in a Denotational Setting*. ACM Transactions on Programming Languages and Systems 7(3):359-379, 1985.


Flemming Nielson. *A Bibliography on Abstract Interpretation*. ACM SIGPLAN Notices 21:31-38, 1986.


Flemming Nielson. *A Bibliography on Abstract Interpretation*. EATCS Bulletin 28:42-52, 1986.


Flemming Nielson. *Semantics-directed Program Analysis: A Tool-maker's Perspective*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 2-21, 1996.

V Artikel (52)


Flemming Nielson, Hanne Riis Nielson. *Finiteness Conditions for Fixed Point Iteration*. In Proceedings of the 7th ACM Conference on LISP and Functional Programming (LFP'92), 96-108, 1992.


Hanne Riis Nielson. *A Hoare-like Proof System for Run-Time Analysis of Programs*. Science of Computer Programming 9(2):107-136, 1987.


David von Oheimb. *Hoare Logic for Java in Isabelle/HOL*. Concurrency and Computation: Practice and Experience 13(13):1173-1214, 2001.


Ernst-Rüdiger Olderog. *Correctness of Programs with Pascal-like Procedures without Global Variables*. Theoretical Computer Science 30(1):49-90, 1984.

V Artikel (53)


Ernst-Rüdiger Olderog, Bernhard Steffen. *Formale Semantik und Programmverifikation*. In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 145-166, 2006.


Ernst-Rüdiger Olderog, Reinhard Wilhelm. *Turing und die Verifikation*. Informatik Spektrum 35(4):271-279, 2012.


Greger Ottosson, Mikael Sjödin. *Worst-Case Execution Time Analysis for Modern Hardware Architectures*. In Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems (LCT-RTS'97), 1997.

V Artikel (54)


Doron Peled. *All from One, One for All: On Model Checking Using Representatives*. In Proceedings of the 5th International Workshop on Computer Aided Verification (CAV'93), Springer-V., LNCS 697, 409-423, 1993.


Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Lecture notes, DAIMI FN-19, Aarhus University, Dänemark, 1981 (als Nachdruck von 1991).


Gordon D. Plotkin. *An Operational Semantics for CSP*. In Proceedings of the TC-2 Working Conference on Formal Description of Programming Concepts II, Dines Bjørner (Hrsg.), North-Holland, Amsterdam, 199-226, 1982.


Gordon D. Plotkin. *The Origins of Structural Operational Semantics*. Journal of Logic and Algebraic Programming 60-61:3-15, 2004.

V Artikel (55)


Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Journal of Logic and Algebraic Programming 60-61:17-139, 2004.


Vaughan R. Pratt. *Semantical Considerations of Floyd-Hoare Logic*. In Proceedings of the 17th IEEE Annual Symposium on Foundations of Computer Science (FOCS'76), 109-121, 1976.


Peter Puschner, Raimund Kirner, Robert G. Pettit. *Towards Composable Timing for Real-Time Programs*. Software Technologies for Future Dependable Distributed Systems, 1-5, 2009.

V Artikel (56)


Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, Gernot Gebhard. *The T-CREST Approach of Compiler and WCET-Analysis Integration*. In Proceedings of the 9th International Workshop on Software Technologies for Future Embedded and Ubiquitious Systems (SEUS 2013), 1-8, 2013.


John H. Reif, Harry R. Lewis. *Symbolic Evaluation and the Global Value Graph*. In Conference Record of the 4th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77), 104-118, 1977.


Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, Bernd Becker. *A Definition and Classification of Timing Anomalies*. In Proceedings of the 6th International Workshop on Worst-Case Execution Time Analysis (WCET 2006), 2006.

V Artikel (57)


Thomas Reps. *Solving Demand Versions of Interprocedural Analysis Problems*. In Proceedings of the 5th International Conference on Compiler Construction (CC'95), Springer-V., LNCS 786, 389-403, 1994.


Thomas Reps. *Demand Interprocedural Program Analysis using Logic Databases*. In Applications of Logic Databases, R. Ramakrishnan (Hrsg.), Kluwer Academic Publishers, 1994.


F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, Frankreich, Dez. 1976.


Oliver Rüthing, Jens Knoop, Bernhard Steffen. *Detecting Equalities of Variables: Combining Efficiency with Precision*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 232-247, 1999.

V Artikel (58)


Oliver Rüthing, Markus Müller-Olm. *On the Complexity of Constant Propagation*. In Proceedings of the 10th European Symposium on Programming (ESOP 2001), Springer-V., LNCS 2028, 190-205, 2001.


Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, Ciera Jaspan. *Lessons from Building Static Analysis Tools at Google*. Communications of the ACM 61(4):58-66, 2018.


Antonella Santone. *Automatic Verification of Concurrent Systems Using a Formula-based Compositional Approach*. Acta Informatica 38(8):531-564, 2002.

V Artikel (59)


David A. Schmidt. *Data Flow Analysis is Model Checking of Abstract Interpretations*. In Conference Record of the 25th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'98), 38-48, 1998.


David A. Schmidt, Bernhard Steffen. *Program Analysis as Model Checking of Abstract Interpretations*. In Proceedings of the 5th Static Analysis Symposium (SAS'98), Springer-V., LNCS 1503, 351-380, 1998.


Mary Lou Soffa. *Tutorial: Techniques to improve the Scalability and Precision of Data Flow Analysis*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 355-356, 1999.

V Artikel (60)


Harini Srinivasan, Michael Wolfe. *Analyzing Programs with Explicit Parallelism*. In Proceedings of the 4th International Conference on Languages and Compilers for Parallel Computing (LCPC'91), Springer-V., LNCS 589, 405-419, 1991.


Bernhard Steffen. *Optimal Run Time Optimization – Proved by a New Look at Abstract Interpretation*. In Proceedings of the 2nd Joint International Conference on the Theory and Practice of Software Development (TAPSOFT'87), Springer-V., LNCS 249, 52-68, 1987.


Bernhard Steffen. *Optimal Data Flow Analysis via Observational Equivalence*. In Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS'89), Springer-V., LNCS 379, 492-502, 1989.

V Artikel (61)


Bernhard Steffen. *Data Flow Analysis as Model Checking*. In Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'91), Springer-V., LNCS 526, 346-365, 1991.


Bernhard Steffen. *Generating Data Flow Analysis Algorithms from Modal Specifications*. International Journal on Science of Computer Programming 21:115-139, 1993.


Bernhard Steffen. *Property-Oriented Expansion*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 22-41, 1996.


Bernhard Steffen, Andreas Claßen, Marion Klein, Jens Knoop, Tiziana Margaria. textcolorblack*The Fixpoint Analysis Machine*. In Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95), Springer-V., LNCS 962, 72-87, 1995.

V Artikel (62)


Bernhard Steffen, Jens Knoop. *Finite Constants: Characterizations of a New Decidable Set of Constants*. Theoretical Computer Science 80(2):303-318, 1991.


Colin Stirling, David Walker. *Local Model Checking in the Modal Mu-Calculus*. Theoretical Computer Science 89(1):161-177, 1991.


Munehiro Takimoto, Kenichi Harada. *Effective Partial Redundancy Elimination based on Extended Value Graph*. Information Processing Society of Japan 38(11):2237-2250, 1990.


Munehiro Takimoto, Kenichi Harada. *Partial Dead Code Elimination Using Extended Value Graph*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 179-193, 1999.

V Artikel (63)


Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5(2):285-309, 1955.


Henrik Theiling. *ILP-based Interprocedural Path Analysis*. In Proceedings of the International Workshop on Embedded Software (EMSOFT 2002), Springer-V., LNCS 2491, 349-363, 2002.


Lothar Thiele, Reinhard Wilhelm. *Design for Timing Predictability*. Real-Time Syst. 28(2-3):157-177, 2004.


Ashish Tiwari, Sumit Gulwani. *Static Program Analysis Using Theorem Proving*. In Proceedings of the 21st Conference on Automated Deduction (CADE-21), LNCS 4603, Springer-V., 147-166, 2007.

V Artikel (64)


Peng Tu, David A. Padua. *Gated SSA-based Demand-driven Symbolic Analysis for Parallelizing Computers*. In Proceedings of the International Conference on Supercomputing (SC'95), 414-423, 1995.


Antti Valmari. *A Stubborn Attack on State Explosion*. Formal Methods in System Design 1(4):297-322, 1992.


Jürgen Vollmer. *Data Flow Analysis of Parallel Programs*. In Proceedings of the IFIP WG 10.3 Working Conference on Parallel Architectures and Compilation Techniques (PACT'95), 168-177, 1995.


Mark N. Wegman, F. Kenneth Zadeck. *Constant Propagation with Conditional Branches*. In Conference Record of the 12th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'85), 291-299, 1985.

V Artikel (65)


Mark N. Wegman, F. Kenneth Zadeck. *Constant Propagation With Conditional Branches*. ACM Transactions on Programming Languages and Systems 13(2):181-201, 1991.


Daniel Weise. *Static Analysis of Mega-Programs (Invited Paper)*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 300-302, 1999.


Reinhard Wilhelm. *Real Time spent on Real Time*. Communications of the ACM 63(10):54-60, 2020.

V Artikel (66)

Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström. *The Worst-case Execution Time Problem – Overview of Methods and Survey of Tools*. ACM Transactions on Embedded Computing Systems 7(3):36.1-53, 2008.

Reinhard Wilhelm, Daniel Grund. *Computation takes Time, but How Much?* Communications of the ACM 57(2):94-103, 2014.

V Artikel (67)


Michael Wolfe, Harini Srinivasan. *Data Structures for Optimizing Programs with Explicit Parallelism*. In Proceedings of the 1st International Conference of the Austrian Center for Parallel Computation, Springer-V., LNCS 591, 139-156, 1991.


Xin Yuan, Rajiv Gupta, Rami Melhem. *Demand-driven Data Flow Analysis for Communication Optimization*. Parallel Processing Letters 7(4):359-370, 1997.


Xin Zheng, Radu Rugina. *Demand-driven Alias Analysis for C*. In Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008), 197-208, 2008.

VI Web-Ressourcen

aiT Worst-Case Execution Time Analyzers. Website: `http://www.absint.com/ait`, 2016. [Online; accessed 1-August-2016]

Reiner Hähnle, Richard Bubel. *A Hoare-Style Calculus with Explicit State Updates*. Handout in the course 'Program Verification' at the Department of Computer Science at the Chalmers University of Technology, 19 Seiten. `i12www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf`

Anhänge

Anhang A

Mathematische Grundlagen

A.1

Relationen

Relationen

Seien M_i, $1 \leq i \leq k$, Mengen.

Definition A.1.1 (k-stellige Relation)

Eine (k-stellige) Relation ist eine Menge R geordneter Tupel von Elementen aus $M_1, \ldots, M_k$, d.h. $R \subseteq M_1 \times \ldots \times M_k$ ist eine Teilmenge des kartesischen Produkts der Mengen M_i, $1 \leq i \leq k$.

Beispiele

- $\emptyset$ ist die kleinste Relation auf $M_1 \times \ldots \times M_k$.
- $M_1 \times \ldots \times M_k$ ist die größte Relation auf $M_1 \times \ldots \times M_k$.

Binäre Relationen

Seien M, N Mengen.

Definition A.1.2 (Binäre Relation)

Eine (binäre) Relation auf M und N ist eine Menge R geordneter Paare von Elementen aus M und N, d.h. R ist eine Teilmenge des kartesischen Produkts von M und N, $R \subseteq M \times N$.

Beispiele

- $\emptyset$ ist die kleinste Relation auf M und N.
- $M \times N$ ist die größte Relation auf M und N.

Bemerkung

- Ist R eine Relation auf M und N, ist es üblich $m\,R\,n$, $R(m,n)$ oder $R\,m\,n$ zu schreiben anstelle von $(m,n) \in R$.

Zwischen, auf

Definition A.1.3 (Zwischen, auf)

Eine Relation R auf M und N heißt Relation zwischen M und N (oder: Relation auf $M \times N$).

Gilt M gleich N, heißt R Relation auf M, in Zeichen: (M, R).

Argument- u. Wertebereich binärer Relationen

Definition A.1.4 (Argument- und Wertebereich)

Sei R eine Relation auf M und N.

Die Mengen

- $dom(R) =_{df} \{m \mid \exists\, n \in N.\ (m, n) \in R\}$
- $ran(R) =_{df} \{n \mid \exists\, m \in M.\ (m, n) \in R\}$

heißen Argumentbereich (engl. domain) bzw. Wertebereich (engl. range) von R.

Eigensch. von Relationen auf einer Menge M

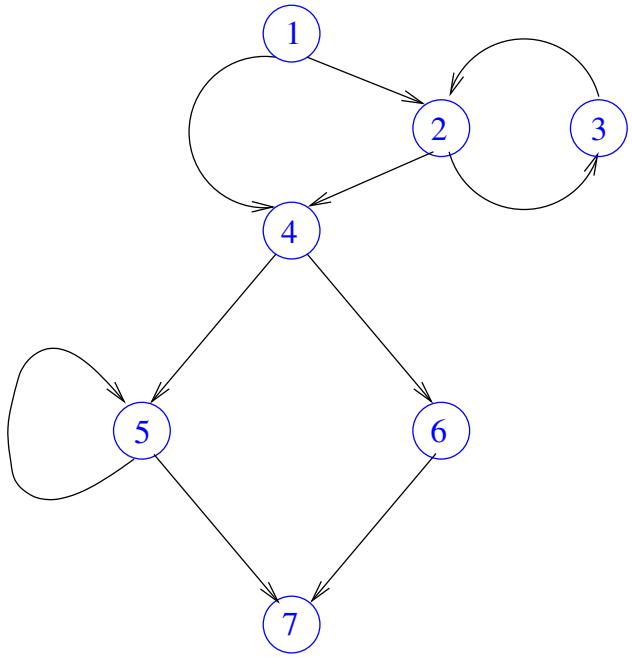
Definition A.1.5 (Eigensch. von Relationen auf M)

Eine Relation R auf einer Menge M heißt

1. reflexiv gdw $\forall\, m \in M.\ m\, R\, m$
2. irreflexiv gdw $\forall\, m \in M.\ \neg m\, R\, m$
3. transitiv gdw $\forall\, m, n, p \in M.\ m\, R\, n\ \wedge\ n\, R\, p\ \Rightarrow\ m\, R\, p$
4. intransitiv gdw $\forall\, m, n, p \in M.\ m\, R\, n \wedge n\, R\, p \Rightarrow \neg m\, R\, p$
5. symmetrisch gdw $\forall\, m, n \in M.\ m\, R\, n\ \Longleftrightarrow\ n\, R\, m$
6. antisymmetrisch gdw $\forall\, m, n \in M.\ m\, R\, n \wedge n\, R\, m \Rightarrow m = n$
7. asymmetrisch gdw $\forall\, m, n \in M.\ m\, R\, n \Rightarrow \neg n\, R\, m$
8. linear gdw $\forall\, m, n \in M.\ m\, R\, n\ \vee\ n\, R\, m\ \vee\ m = n$
9. total gdw $\forall\, m, n \in M.\ m\, R\, n\ \vee\ n\, R\, m$

(Anti-) Beispiel

Sei $G = (N, E, \mathbf{s} \equiv 1, \mathbf{e} \equiv 7)$ der nachstehende (Fluss-) Graph und R die Relation ‘Knoten $\cdot$ ist über eine (gerichtete) Kante aus E von G verbunden mit Knoten $\cdot$ (z.B. ist Knoten 4 verbunden mit Knoten 6, aber nicht umgekehrt).



Die Relation R ist nicht reflexiv, nicht irreflexiv, nicht transitiv, nicht intransitv, nicht symmetrisch, nicht antisymmetrisch, nicht asymmetrisch, nicht linear und nicht total.

Äquivalenzrelation

Sei R eine Relation auf M.

Definition A.1.6 (Äquivalenzrelation)

R heißt Äquivalenzrelation (oder Äquivalenz) gdw R ist reflexiv, transitiv und symmetrisch.

Übungsaufgabe A.1.7

Bezeichne $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \mid 35$.

Beweise oder widerlege durch Angabe eines Gegenbeispiels:

Die Teilbarkeitsrelation $|$ auf $\mathbb{N}_0$ ist

1. reflexiv
2. irreflexiv
3. transitiv
4. intransitiv
5. symmetrisch
6. antisymmetrisch
7. asymmetrisch
8. linear
9. total
10. Äquivalenz(relation)

A.2

Geordnete Mengen, Ordnungen

A.2.1

Halbordnungen, partielle Ordnungen

Geordnete Mengen

Sei R eine Relation auf M.

Definition A.2.1.1 (Halbordnung)

R ist eine Halbordnung (oder: Quasiordnung) (engl. pre-order (oder: quasi-order)) gdw R ist reflexiv und transitiv.

Definition A.2.1.2 (Partielle Ordnung)

R ist eine partielle Ordnung (engl. partial order (oder: poset oder: order)) gdw R ist reflexiv, transitiv und antisymmetrisch.

Definition A.2.1.3 (Strikte partielle Ordnung)

R ist eine strikte partielle Ordnung (engl. strict partial order) gdw R ist asymmetrisch und transitiv.

Beispiele geordneter Mengen

Halbordnung (reflexiv, transitiv)

- Die Relation $\Rightarrow$ auf logischen Formeln.

Partielle Ordnung (reflexiv, transitiv, antisymmetrisch)

- Die Relationen $=$, $\leq$ und $\geq$ auf $\mathbb{N}$.
- Die Relation $m \mid n$ (m ist Teiler von n) auf $\mathbb{N}$.

Strikte partielle Ordnungen (asymmetrisch, transitiv)

- Die Relationen $<$ und $>$ auf $\mathbb{N}$.
- Die Relationen $\subset$ und $\supset$ auf Mengen.

Äquivalenzrelation (reflexiv, transitiv, symmetrisch)

- Die Relation $\Longleftrightarrow$ auf logischen Formeln.
- Die Relation 'haben die selben Primzahlfaktoren' auf $\mathbb{N}$.
- Die Relation 'sind Bürger desselben Landes' auf Leuten.

Beachte

- Eine antisymmetrische Halbordnung ist eine partielle Ordnung; eine symmetrische Halbordnung ist eine Äquivalenzrelation.
- Der Einfachheit halber wird auch das Paar (M, R) eine Halbordnung, partielle Ordnung bzw. strikte partielle Ordnung genannt.
- Präziser können wir vom Paar (M, R) als einer durch R halbgeordneten, partiell geordneten bzw. strikt partiell geordneten Menge sprechen.
- Synonym sprechen wir von M auch als von einer halbgeordneten, partiell geordneten bzw. strikt partiell geordneten Menge, oder als einer Menge mit einer Halbordnung, partiellen Ordnung bzw. strikten partiellen Ordnung.
- Unabhängig von der Grundmenge ist die Gleichheitsrelation $=$ stets eine partielle Ordnung, die sog. diskrete (partielle) Ordnung.

Der strikte Teil einer Ordnung

Sei $\sqsubseteq$ eine Halbordnung (reflexiv, transitiv) auf P.

Definition A.2.1.4 (Strikter Teil von $\sqsubseteq$)

Die Relation $\sqsubset$ auf P definiert durch

$$\forall\, p, q \in P.\ p \sqsubset q \iff_{df} p \sqsubseteq q \ \wedge\ p \neq q$$

heißt strikter Teil von $\sqsubseteq$.

Korollar A.2.1.5 (Strikte partielle Ordnung)

Sei $(P, \sqsubseteq)$ eine partielle Ordnung und $\sqsubset$ der strikte Teil von $\sqsubseteq$.

Dann gilt: $(P, \sqsubset)$ ist eine strikte partielle Ordnung.

Nützliche Resultate

Sei $\sqsubset$ eine strikte partielle Ordnung (asymmetrisch, transitiv) auf P.

Lemma A.2.1.6

Die Relation $\sqsubset$ ist irreflexiv.

Lemma A.2.1.7

Das Paar $(P, \sqsubseteq)$, wobei $\sqsubseteq$ definiert ist durch

$$\forall\, p, q \in P.\ p \sqsubseteq q \iff_{df} p \sqsubset q \ \vee \ p = q$$

ist eine partielle Ordnung.

Induzierte (oder: ererbte) partielle Ordnung

Definition A.2.1.8 (Induzierte partielle Ordnung)

Sei $(P, \sqsubseteq_P)$ eine partiell geordnete Menge, $Q \subseteq P$ eine Teilmenge von P und $\sqsubseteq_Q$ eine Relation auf Q definiert durch

$$\forall\, q, r \in Q.\ q \sqsubseteq_Q r \iff_{df} q \sqsubseteq_P r$$

Dann heißt $\sqsubseteq_Q$ die induzierte partielle Ordnung auf Q (oder: ererbte Ordnung von P auf Q).

Übungsaufgabe A.2.1.9

Bezeichne $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \mid 35$.

Beweise oder widerlege durch Angabe eines Gegenbeispiels:

Die Teilbarkeitsrelation $|$ auf $\mathbb{N}_0$ ist eine

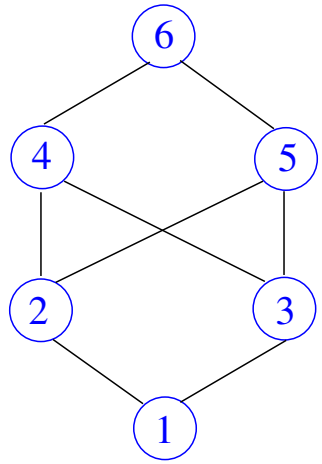
1. Halbordnung
2. partielle Ordnung
3. strikte partielle Ordnung
4. Äquivalenz(relation)

A.2.2

Hasse-Diagramme

Hasse-Diagramme

...sind kompakte graphische Darstellungen partieller Ordnungen.



Die Kanten eines Hasse-Diagramms

- werden von unten nach oben gelesen (tiefer bedeutet dabei kleiner bzgl. der Ordnung, höher bedeutet größer).
- stellen die Relation R '$\cdot$ ist ein unmittelbarer Vorgänger von $\cdot$' einer partiellen Ordnung $(P, \sqsubseteq)$ definiert durch $p\,R\,q \iff_{df} p \sqsubset q \wedge \nexists r \in P.\ p \sqsubset r \sqsubset q$ dar, wobei $\sqsubset$ die strikte partielle Ordnung von $\sqsubseteq$ ist.

Lesen von Hasse-Diagrammen

Die Hasse-Diagramm-Darstellung einer partiellen Ordnung

- verzichtet auf Kanten zur Darstellung von Reflexivität und Transitivität der Relation.
- konzentriert sich auf die Darstellung der Relation 'unmittelbarer Vorgänger'.

Die Hasse-Diagramm-Darstellung einer partiellen Ordnung

- ist kompakt und (gemessen an der Zahl der Kanten) ökonomisch.
- erhält alle relevanten Informationen über die dargestellte partielle Ordnung:
  - $p \sqsubseteq q \wedge p = q$ (Reflexivität): trivialerweise dargestellt (implizit, also ohne explizite Kante).
  - $p \sqsubseteq q \wedge p \neq q$ (Transitivität): dargestellt durch aufsteigende Pfade (mit mindestens einer Kante) von p nach q.

Übungsaufgabe A.2.2.1

Welche der folgenden Diagramme sind Hasse-Diagrammdarstellungen einer partiellen Ordnung?

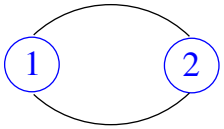
a)

{ }

b)

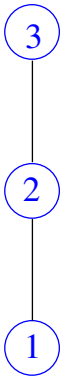
c)

d)

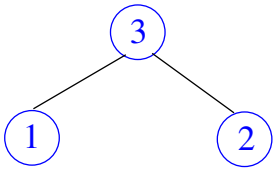


e)

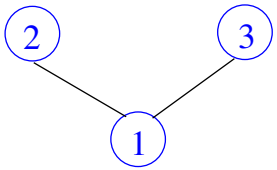
f)



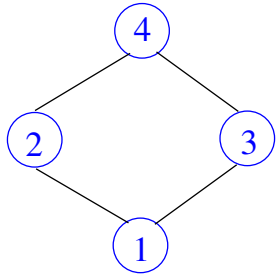
g)



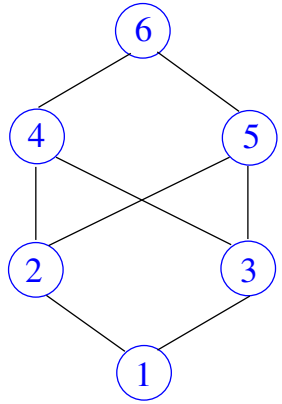
h)



i)



j)



Übungsaufgabe A.2.2.2

Bezeichne $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \mid 35$.

Gib einen aussagekräftigen Ausschnitt des Hasse-Diagramms der Teilbarkeitsrelation $|$ auf $\mathbb{N}_0$ an.

A.2.3

Schranken und extreme Elemente

Schranken in Halbordnungen

Definition A.2.3.1 (Schranken in Halbordnungen)

Sei $(Q, \sqsubseteq)$ eine Halbordnung, $q \in Q$ und $Q' \subseteq Q$.

q heißt

1. untere Schranke von Q', in Zeichen: $q \sqsubseteq Q'$, wenn $\forall\, q' \in Q'.\ q \sqsubseteq q'$
2. obere Schranke von Q', in Zeichen: $Q' \sqsubseteq q$, wenn $\forall\, q' \in Q'.\ q' \sqsubseteq q$
3. größte untere Schranke (gus) (oder Infimum) von Q', in Zeichen: $\bigsqcap Q'$, wenn q eine untere Schranke von Q' ist und für jede andere untere Schranke $\hat{q}$ von Q' gilt: $\hat{q} \sqsubseteq q$.
4. kleinste obere Schranke (kos) (oder Supremum) von Q', in Zeichen: $\bigsqcup Q'$, wenn q eine obere Schranke von Q' ist und für jede andere obere Schranke $\hat{q}$ von Q' gilt: $q \sqsubseteq \hat{q}$.

Extreme Elemente in Halbordnungen

Definition A.2.3.2 (Extreme Elemente in Halbordn.)

Sei $(Q, \sqsubseteq)$ eine Halbordnung, $\sqsubset$ der strikte Teil von $\sqsubseteq$ sowie $Q' \subseteq Q$ und $q \in Q'$.

q heißt

1. minimales Element von Q', wenn es kein $q' \in Q'$ gibt mit $q' \sqsubset q$.
2. maximales Element von Q', wenn es kein $q' \in Q'$ gibt mit $q \sqsubset q'$.
3. kleinstes (oder Minimum-) Element von Q', wenn $q \sqsubseteq Q'$.
4. größtes (oder Maximum-) Element von Q', wenn $Q' \sqsubseteq q$.

Bemerkung: Kleinste und größte Elemente von Q selbst werden gewöhnlich mit $\bot$ bzw. $\top$ (Tief, Hoch (engl. bottom, top)) bezeichnet, wenn sie existieren. Kleinste (größte) Elemente von Q sind stets auch minimale (maximale) Elemente von Q.

Existenz und Eindeutigkeit

...von Schranken und extremen Elementen in partiell geordneten Mengen.

Sei $(P, \sqsubseteq)$ eine partielle Ordnung und $Q \subseteq P$ eine Teilmenge von P.

Lemma A.2.3.3 (kos/gus: Eindeutig, wenn existent)

Kleinste obere Schranken, größte untere Schranken, kleinste Elemente und größte Elemente in Q sind eindeutig, wenn sie existieren.

Lemma A.2.3.4 (Min./Max. Elemente: Nicht eind.)

Minimale u. maximale Elemente in Q sind i.a. nicht eindeutig.

Beachte: Lemma A.2.3.3 legt nahe, $\bigsqcup$ und $\bigsqcap$ als partielle Abbildungen $\bigsqcup, \bigsqcap : \mathcal{P}(P) \to P$ von der Potenzmenge $\mathcal{P}(P)$ von P nach P zu betrachten. Lemma A.2.3.3 gilt nicht für Halbordnungen.

Charakterisierung kleinster, größter Elemente

...in Form von Infima und Suprema von Mengen.

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Lemma A.2.3.5 (Charakterisierung von $\bot$ und $\top$)

Das kleinste Element $\bot$ und das größte Element $\top$ von P sind durch das Supremum bzw. das Infimum der leeren Menge und das Infimum bzw. das Supremum von P gegeben, d.h.:

$$\bot = \bigsqcup \emptyset = \bigsqcap P \quad \text{und} \quad \top = \bigsqcap \emptyset = \bigsqcup P$$

wenn sie existieren.

Untere und obere Schranken von Mengen

Betrachtet man $\bigsqcup$ und $\bigsqcap$ als partielle Funktionen $\bigsqcup, \bigsqcap : \mathcal{P}(P) \to P$ auf der Potenzmenge einer partiellen Ordnung $(P, \sqsubseteq)$, legt das nahe, zwei weitere Abbildungen $US, OS : \mathcal{P}(P) \to \mathcal{P}(P)$ auf $\mathcal{P}(P)$ einzuführen:

Definition A.2.3.6 (Untere, obere Schranken v. M.)

Sei $(P, \sqsubseteq)$ eine partielle Ordnung. Dann bezeichnen $US, OS : \mathcal{P}(P) \to \mathcal{P}(P)$ zwei Abbildungen, die eine Teilmenge $Q \subseteq P$ auf die Menge ihrer unteren Schranken bzw. oberen Schranken abbilden:

1. $\forall\, Q \subseteq P.\ US(Q) =_{df} \{us \in P \mid\ us \sqsubseteq Q\}$
2. $\forall\, Q \subseteq P.\ OS(Q) =_{df} \{os \in P \mid Q \sqsubseteq os\}$

Eigensch. unterer, oberer Schranken v. Mengen

Lemma A.2.3.7

Sei $(P, \sqsubseteq)$ eine partielle Ordnung und $Q \subseteq P$. Dann gilt:

$$\bigsqcup Q = \bigsqcap OS(Q) \quad \text{und} \quad \bigsqcap Q = \bigsqcup US(Q)$$

wenn das Supremum und das Infimum von Q existieren.

Lemma A.2.3.8

Sei $(P, \sqsubseteq)$ eine partielle Ordnung, $Q, Q_1, Q_2 \subseteq P$. Dann gilt:

1. $Q_1 \subseteq Q_2 \Rightarrow US(Q_1) \supseteq US(Q_2) \wedge OS(Q_1) \supseteq OS(Q_2)$
2. $OS(US(OS(Q))) = OS(Q)$
3. $US(OS(US(Q))) = US(Q)$

Beachte: Lemma A.2.3.8(1) sagt, dass US und OS antitonische Abbildungen sind (s. Kapitel A.2.7).

Übungsaufgabe A.2.3.9

Welche Elemente der folgenden Diagramme sind minimal, maximal, kleinst, größt?

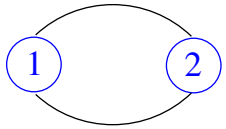
a)

{ }

b)

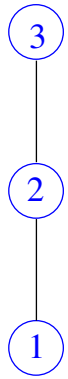
c)

d)

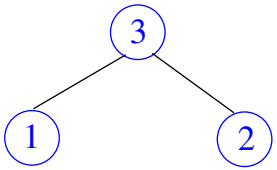


e)

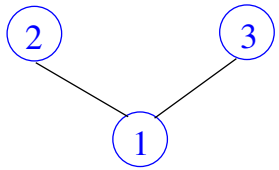
f)



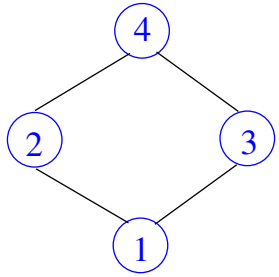
g)



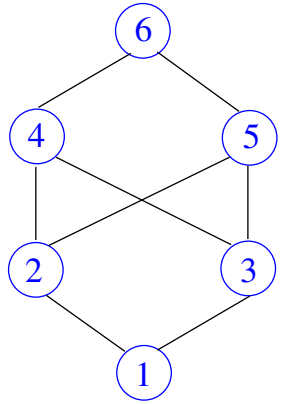
h)



i)



j)



Übungsaufgabe A.2.3.10

Bezeichne $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \mid 35$.

Gib die Menge derjenigen Elemente aus $\mathbb{N}_0$ an, die

1. minimal
2. maximal
3. kleinst
4. größt

bzgl. der Teilbarkeitsrelation $|$ auf $\mathbb{N}_0$ sind.

A.2.4

Noethersche und Artinsche Ordnungen

Noethersche und Artinsche Ordnungen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Definition A.2.4.1 (Noethersche Ordnung)

$(P, \sqsubseteq)$ heißt Noethersche Ordnung (engl. Noetherian order), wenn jede nichtleere Teilmenge $\emptyset \neq Q \subseteq P$ ein minimales Element enthält.

Definition A.2.4.2 (Artinsche Ordnung)

$(P, \sqsubseteq)$ heißt Artinsche Ordnung (engl. Artinian order), wenn die duale Ordnung $(P, \sqsupseteq)$ von $(P, \sqsubseteq)$ eine Noethersche Ordnung ist.

Lemma A.2.4.3

$(P, \sqsubseteq)$ ist eine Artinsche Ordnung gdw jede nichtleere Teilmenge $\emptyset \neq Q \subseteq P$ enthält ein maximales Element.

Wohlfundierte Ordnungen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Definition A.2.4.4 (Wohlfundierte Ordnung)

$(P, \sqsubseteq)$ heißt wohlfundierte Ordnung (engl. well-founded order), wenn $(P, \sqsubseteq)$ eine total geordnete Noethersche Ordnung ist.

Lemma A.2.4.5

$(P, \sqsubseteq)$ ist eine wohlfundierte Ordnung gdw jede nichtleere Teilmenge $\emptyset \neq Q \subseteq P$ enthält ein kleinstes Element.

Noethersche Induktion

Theorem A.2.4.6 (Noethersche Induktion)

Sei $(N, \sqsubseteq)$ eine Noethersche Ordnung, $N_{min} \subseteq N$ die Menge der minimalen Elemente von N und $\phi : N \to \mathbb{B}$ ein Prädikat auf N. Dann:

Wenn

1. $\forall\, n \in N_{min}.\ \phi(n)$ (Induktionsanfang)
2. $\forall\, n \in N \backslash N_{min}.\ (\forall\, m \sqsubset n.\ \phi(m)) \Rightarrow \phi(n)$ (Induk.-Schritt)

dann gilt:

$$\forall\, n \in N.\ \phi(n)$$

A.2.5

Ketten

Ketten, Antiketten

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Definition A.2.5.1 (Kette)

Eine Menge $K \subseteq P$ heißt Kette (engl. chain), wenn die Elemente von K total geordnet sind, d.h. wenn gilt:
$\forall\, k_1, k_2 \in K.\ k_1 \sqsubseteq k_2 \vee k_2 \sqsubseteq k_1$.

Definition A.2.5.2 (Antikette)

Eine Menge $K \subseteq P$ heißt Antikette (engl. antichain), wenn gilt: $\forall\, k_1, k_2 \in K.\ k_1 \sqsubseteq k_2 \Rightarrow k_1 = k_2$.

Definition A.2.5.3 (Endliche, unendl. (Anti-) Kette)

Sei $K \subseteq P$ eine Kette oder Antikette. K heißt endlich, wenn die Zahl der Elemente endlich ist; K heißt unendlich sonst.

Beachte: Jede Menge P wird durch Überstülpen der diskreten Ordnung $(P, =)$ zu einer Antikette.

Aufsteigende, absteigende Ketten

Definition A.2.5.4 (Aufsteigende, absteigende Kette)

Sei $K \subseteq P$ eine Kette. K dargestellt in der Form

- $K = \{k_0 \sqsubseteq k_1 \sqsubseteq k_2 \sqsubseteq \ldots\}$
- $K = \{k_0 \sqsupseteq k_1 \sqsupseteq k_2 \sqsupseteq \ldots\}$

heißt aufsteigende Kette bzw. absteigende Kette.

Beispiele für Ketten

Die Menge

- $M =_{df} \{n \in \mathbb{N} \mid n \text{ gerade}\}$ ist eine Kette in $\mathbb{N}$.
- $M =_{df} \{z \in \mathbb{Z} \mid z \text{ ungerade}\}$ ist eine Kette in $\mathbb{Z}$.
- $M =_{df} \{\{k \in \mathbb{N} \mid k < n\} \mid n \in \mathbb{N}\}$ ist eine Kette in der Potenzmenge $\mathcal{P}(\mathbb{N})$ von $\mathbb{N}$.

Beachte: Eine Kette kann stets als aufsteigende und absteigende Kette aufgefasst werden.

- $\{0 \leq 2 \leq 4 \leq 6 \leq \ldots\}$: $\mathbb{N}$ als aufsteigende Kette.
- $\{\ldots \geq 6 \geq 4 \geq 2 \geq 0\}$: $\mathbb{N}$ als absteigende Kette.
- $\{\ldots \leq -3 \leq -1 \leq 1 \leq 3 \leq \ldots\}$: $\mathbb{Z}$ als aufsteig. Kette.
- $\{\ldots \geq 3 \geq 1 \geq -1 \geq -3 \geq \ldots\}$: $\mathbb{Z}$ als absteig. Kette.
- ...

Schließlich stationäre Folgen

Definition A.2.5.5 (Schließlich stationäre Folge)

1. Eine aufsteigende Folge der Form

$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots$$

heißt schließlich stationär, wenn

$$\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$$

2. Eine absteigende Folge der Form

$$p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$

heißt schließlich stationär, wenn

$$\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$$

Ketten und Folgen

Lemma A.2.5.6

Eine aufsteigende oder absteigende Folge der Form

$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots \quad \text{oder} \quad p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$

1. ist eine endliche Kette gdw sie ist schließlich stationär.
2. ist eine unendliche Kette gdw sie ist nicht schließlich stationär.

Beachte den feinen Unterschied zwischen der Sichtweise einer Kette als Menge

$$\{p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots\} \quad \text{or} \quad \{p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots\}$$

und als Folge

$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots \quad \text{or} \quad p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$

Anders als Mengen können Folgen Duplikate enthalten, was einer Definition von Ketten als Multimengen entspräche.

Aufsteigende, absteigende Kettenbedingung

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Definition A.2.5.7 (Aufst./abst. Kettenbedingung)

$(P, \sqsubseteq)$ erfüllt die

1. aufsteigende Kettenbedingung (engl. ascending chain condition), wenn jede aufsteigende Kette schließlich stationär ist, d.h. für jede Kette $p_1 \sqsubseteq p_2 \sqsubseteq \ldots \sqsubseteq p_n \sqsubseteq \ldots$ gibt es einen Index $m \geq 1$ mit $p_m = p_{m+j}$ für alle $j \in \mathbb{N}$.
2. absteigende Kettenbedingung (engl. descending chain condition), wenn jede absteigende Kette schließlich stationär ist, d.h. für jede Kette $p_1 \sqsupseteq p_2 \sqsupseteq \ldots \sqsupseteq p_n \sqsupseteq \ldots$ gibt es einen Index $m \geq 1$ mit $p_m = p_{m+j}$ für alle $j \in \mathbb{N}$.

Ketten und Noethersche Ordnungen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Lemma A.2.5.8 (Noethersche Ordnung)

Folgende Aussagen sind äquivalent:

1. $(P, \sqsubseteq)$ ist eine Noethersche Ordnung.
2. $(P, \sqsubseteq)$ erfüllt die absteigende Kettenbedingung.
3. Jede Kette der Form

   $$p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \dots$$

   ist schließlich stationär, d.h.: $\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$.
4. Jede Kette der Form

   $$p_0 \sqsupset p_1 \sqsupset p_2 \sqsupset \dots$$

   ist endlich.

Ketten und Artinsche Ordnungen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Lemma A.2.5.9 (Artinsche Ordnung)

Folgende Aussagen sind äquivalent:

1. $(P, \sqsubseteq)$ ist eine Artinsche Ordnung.
2. $(P, \sqsubseteq)$ erfüllt die aufsteigende Kettenbedingung.
3. Jede Kette der Form

   $$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \dots$$

   ist schließlich stationär, d.h.: $\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$.
4. Jede Kette der Form

   $$p_0 \sqsubset p_1 \sqsubset p_2 \sqsubset \dots$$

   ist endlich.

Ketten und Noethersche, Artinsche Ordnungen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Lemma A.2.5.10 (Noethersche, Artinsche Ordnung)

Folgende Aussagen sind äquivalent:

1. $(P, \sqsubseteq)$ ist eine Noethersche und eine Artinsche Ordnung.
2. $(P, \sqsubseteq)$ erfüllt die absteigende und die aufsteigende Kettenbedingung.
3. Jede Kette $K \subseteq P$ ist endlich.

A.2.6

Gerichtete Mengen

Gerichtete Mengen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung und $\emptyset \neq G \subseteq P$.

Definition A.2.6.1 (Gerichtete Menge)

G $(\neq \emptyset)$ heißt gerichtete Menge (engl. directed set), wenn

$$\forall\, g, h \in G.\ \exists\, k \in G.\ k \in OS(\{g, h\})$$

d.h. für je zwei Elemente g und h gibt es eine gemeinsame obere Schranke von g und h in G, d.h. $OS(\{g, h\}) \cap G \neq \emptyset$.

Eigenschaften gerichteter Mengen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung und $G \subseteq P$.

Lemma A.2.6.2

G ist eine gerichtete Menge gdw jede endliche Teilmenge $G' \subseteq G$ hat eine obere Schranke in G, d.h.
$\exists g \in G.\ g \in OS(G')$, d.h. $OS(G') \cap G \neq \emptyset$.

Lemma A.2.6.3

Hat G ein größtes Element, so ist G eine gerichtete Menge.

Eigenschaften endlicher gerichteter Mengen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung und $G \subseteq P$.

Korollar A.2.6.4

Sei G eine endliche gerichtete Menge. Dann gilt:
$\bigsqcup G$ *existiert* $\in G$ und ist das größte Element von G.

Beweis. Weil G eine gerichtete Menge ist, gilt:

$$\exists g \in G.\ g \in OS(G),\ \text{d.h. } OS(G) \cap G \neq \emptyset.$$

Das bedeutet $G \sqsubseteq g$. Die Antisymmetrie von $\sqsubseteq$ liefert, dass das Element mit dieser Eigenschaft eindeutig bestimmt ist. Folglich ist g das (eindeutig bestimmte) größte Element von G gegeben durch $\bigsqcup G$, d.h. $g = \bigsqcup G$.

Beachte: Ist G unendlich, gilt die Aussage von Korollar A.2.6.4 i.a. nicht.

Stark gerichtete Mengen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung mit kleinstem Element $\bot$ und $G \subseteq P$.

Definition A.2.6.5 (Stark gerichtete Menge)

$G \neq \emptyset$ heißt stark gerichtete Menge (engl. strongly directed set), wenn

1. $\bot \in G$
2. $\forall\, g, h \in G.\ \exists\, k \in G.\ k = \bigsqcup\{g, h\}$, d.h. für je zwei Elemente g und h existiert das Supremum $\bigsqcup\{g, h\}$ von g und h in G.

Eigenschaften stark gerichteter Mengen

Sei $(P, \sqsubseteq)$ eine partielle Ordnung mit kleinstem Element $\bot$ und $G \subseteq P$.

Lemma A.2.6.6

G ist eine stark gerichtete Menge gdw jede endliche Teilmenge $G' \subseteq G$ hat ein Supremum in G, d.h. $\exists\, g \in G.\ d = \bigsqcup G'$.

Lemma A.2.6.7

Sei G eine endliche stark gerichtete Menge. Dann gilt: $\bigsqcup G$ *existiert* $\in G$ und ist das größte Element von G.

Beachte: Die Aussage von Lemma A.2.6.7 gilt i.a. nicht, wenn G unendlich ist.

Gerichtete Mengen, stark ger. Mengen, Ketten

Sei $(P, \sqsubseteq)$ eine partielle Ordnung mit kleinstem Element $\bot$.

Lemma A.2.6.8

Sei $\emptyset \neq G \subseteq P$ eine nichtleere Teilmenge von P. Dann gilt:

1. G ist eine gerichtete Menge, wenn G eine stark gerichtete Menge ist.
2. G ist eine stark gerichtete Menge, wenn $\bot \in G$ und G eine Kette ist.

Korollar A.2.6.9

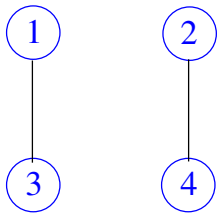
Sei $\emptyset \neq G \subseteq P$ eine nichtleere Teilmenge von P. Dann gilt:

$\bot \in G \wedge G$ Kette $\Rightarrow G$ stark gerichtete Menge $\Rightarrow G$ gerichtete Menge

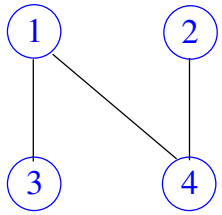
Übungsaufgabe A.2.6.10

Welche der folgenden partiellen Ordnungen sind (stark) gerichtete Mengen? Welche ihrer Teilmengen sind (stark) gerichtete Mengen?

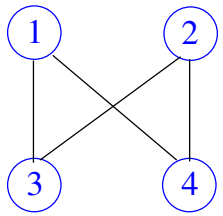
a)



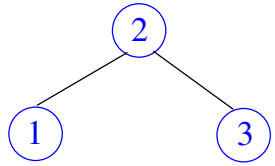
b)



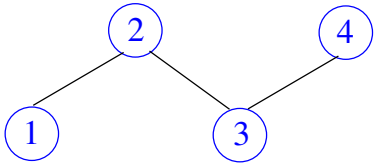
c)



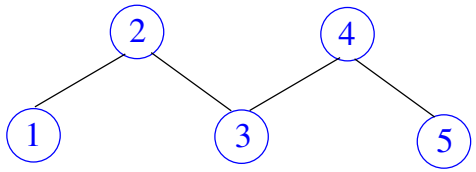
d)



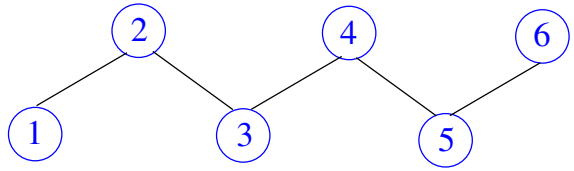
e)



f)



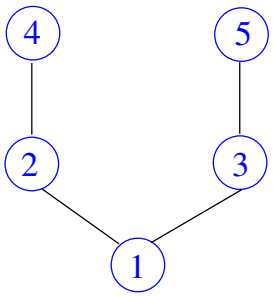
g)



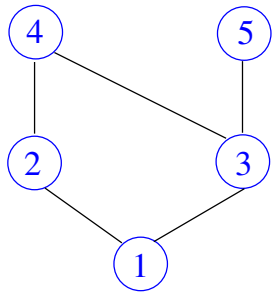
Übungsaufgabe A.2.6.11

Welche der folgenden partiellen Ordnungen sind (stark) gerichtete Mengen? Welche ihrer Teilmengen sind (stark) gerichtete Mengen?

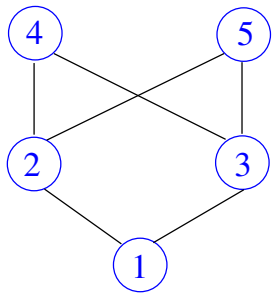
a)



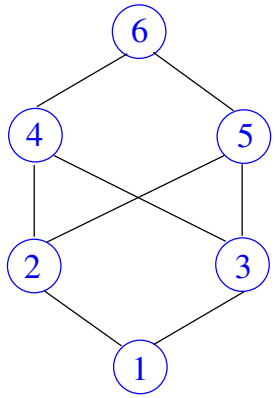
b)



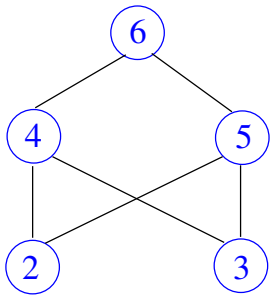
c)



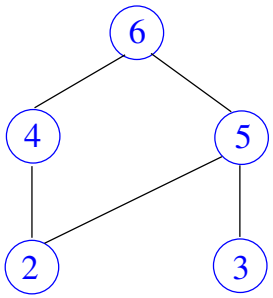
d)



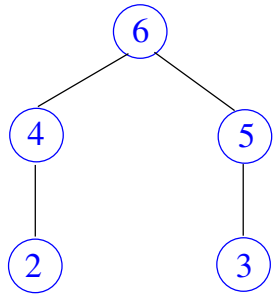
e)



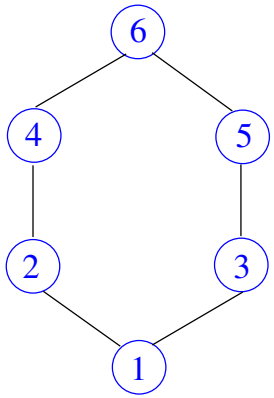
f)



g)



h)



Übungsaufgabe A.2.6.12

Betrachte die partielle Ordnung $(\mathbb{N}_0, \sqsubseteq)$ mit $\sqsubseteq =_{df} |$, wobei $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$ bezeichne, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \,|\, 35$.

Ist die Menge $\mathbb{N}_0$

1. gerichtet?
2. stark gerichtet?

Welche Teilmengen von $\mathbb{N}_0$ sind

1. gerichtet?
2. stark gerichtet?

Beweis oder Gegenbeispiel.

A.2.7

Abbildungen auf partiellen Ordnungen

Monotone und antitone Abbildungen auf POs

Seien $(C, \sqsubseteq_C)$, $(D, \sqsubseteq_D)$ partielle Ordnungen und $f \in [C \to D]$ eine Abbildung von C nach D.

Definition A.2.7.1 (Monotone Abbildungen auf POs)

f heißt monoton (oder: ordnungserhaltend) gdw

$$\forall\, c, c' \in C.\ c \sqsubseteq_C c' \Rightarrow f(c) \sqsubseteq_D f(c')$$

(Erhalt der Ordnung von Elementen)

Definition A.2.7.2 (Antitone Abbildungen auf POs)

f heißt antiton (oder: ordnungsumkehrend) gdw

$$\forall\, c, c' \in C.\ c \sqsubseteq_C c' \Rightarrow f(c') \sqsupseteq_D f(c)$$

(Umkehrung der Ordnung von Elementen)

Expandierende u. kontrahierende Abb. auf POs

Sei $(C, \sqsubseteq_C)$ eine partielle Ordnung, $f \in [C \to C]$ eine Abbildung auf C und $\hat{c} \in C$ ein Element von C.

Definition A.2.7.3 (Expandierende Abb. auf POs)

f heißt

1. expandierend (oder: inflationär) für $\hat{c}$ gdw $\hat{c} \sqsubseteq f(\hat{c})$
2. expandierend (oder: inflationär) gdw $\forall\, c \in C.\ c \sqsubseteq f(c)$

Definition A.2.7.4 (Kontrahierende Abb. auf POs)

f heißt

1. kontrahierend (oder: deflationär) für $\hat{c}$ gdw $f(\hat{c}) \sqsubseteq \hat{c}$
2. kontrahierend (oder: deflationär) gdw $\forall\, c \in C.\ f(c) \sqsubseteq c$

A.2.8

Ordnungshomomorphismen und -isomorphismen

Ordnungshomomorphismen, -isomorphismen

Seien $(P, \sqsubseteq_P)$, $(R, \sqsubseteq_R)$ partielle Ordnungen und $f \in [P \to R]$ eine Abbildung von P nach R.

Definition A.2.8.1 (Ord.-Hom. & -Isomorphismus)

f heißt

1. Ordnungshomomorphismus zwischen P und R, wenn f monoton (oder ordnungserhaltend) ist, d.h.:
$$\forall\, p, q \in P.\ p \sqsubseteq_P q \Rightarrow f(p) \sqsubseteq_R f(q)$$
2. Ordnungsisomorphismus zwischen P und R, wenn f ein bijektiver Ordnungshomomorphismus zwischen P und R und die inverse Abbildung f^{-1} von f ein Ordnungshomomorphismus zwischen R und P ist.

Definition A.2.8.2 (Ordnungsisomorph)

$(P, \sqsubseteq_P)$ und $(R, \sqsubseteq_R)$ heißen ordnungsisomorph, wenn es einen Ordnungsisomorphismus zwischen P und R gibt.

Ordnungseinbettungen

Seien $(P, \sqsubseteq_P)$, $(R, \sqsubseteq_R)$ partielle Ordnungen und $f \in [P \to R]$ eine Abbildung von P nach R.

Definition A.2.8.3 (Ordnungseinbettung)

f heißt Ordnungseinbettung von P in R gdw

$$\forall\, p, q \in P.\ p \sqsubseteq_P q \iff f(p) \sqsubseteq_R f(q)$$

Lemma A.2.8.4 (Ord.-Einbett. und -Isomorphismen)

f ist ein Ordnungsisomorphismus zwischen P und R gdw f ist eine Ordnungseinbettung von P in R und f ist surjektiv.

Intuitiv: Ordnungsisomorphe partielle Ordnungen sind 'im wesentlichen gleich'.

A.3

Vollständige partielle Ordnungen

A.3.1

Kettenvollständige, gerichtete vollständige partielle Ordnungen

Vollständige partielle Ordnungen

...oder vollständige partiell geordnete Mengen:

- ein etwas schwächerer Ordungsbegriff als der eines Verbands (s. Anhang A.4), der aber in der Informatik oft ausreichend und in diesem Sinn angemessener zur Modellierung von Problemen ist, wenn (wie oft) die vollen Verbandseigenschaften nicht benötigt werden.
- gibt es in zwei Sichtweisen als sogenannte
  - kettenvollständige partielle Ordnungen (KVPOs)
  - gerichtete vollständige partielle Ordnungen (GVPOs)

  die sich konzeptuell auf Ketten bzw. gerichtete Mengen abstützen, dabei aber äquivalent sind (s. Theorem 3.1.7).

Vollständige partielle Ordnungen: KVPO-Sicht

Definition A.3.1.1 (Kettenvollständige part. Ordn.)

Eine partielle Ordnung $(P, \sqsubseteq)$ ist eine

1. kettenprävollständige partielle Ordnung (prä-KVPO) (engl. chain complete partial order (pre-CCPO), wenn jede nichtleere (aufsteigende) Kette $\emptyset \neq K \subseteq P$ eine kleinste obere Schranke $\bigsqcup K$ in P hat, d.h. $\bigsqcup K$ *existiert* $\in P$.
2. (geerdete) kettenvollständige partielle Ordnung (KVPO) (engl. pointed chain complete partial order (CCPO)), wenn jede (aufsteigende) Kette $K \subseteq P$ eine kleinste obere Schranke $\bigsqcup K$ in P hat, d.h. $\bigsqcup K$ *existiert* $\in P$.

Vollständige partielle Ordnungen: GVPO-Sicht

Definition A.3.1.2 (Gerichtete vollst. part. Ordnung)

Eine partielle Ordnung $(P, \sqsubseteq)$ ist eine

1. gerichtete prävollständige partielle Ordnung (prä-GVPO) (engl. directedly complete partial order (pre-DCPO)), wenn jede gerichtete Teilmenge $G \subseteq P$ eine kleinste obere Schranke $\bigsqcup G$ in P hat, d.h. $\bigsqcup G$ *existiert* $\in P$.
2. (geerdete) gerichtete vollständige partielle Ordnung (GVPO) (engl. pointed directedly complete partial order (DCPO)), wenn sie eine prä-GVPO ist und ein kleinstes Element $\bot$ hat.

Anmerkungen zur KVPOs und GVPOs

Zu KVPOs:

- Eine KVPO wird oft als Bereich (engl. domain) bezeichnet.
- 'Aufsteigende Kette' and 'Kette' können in Definition A.3.1.1 gleichbedeutend verwendet werden, da Ketten stets aufsteigend angeordnet angegeben werden können. Der genauere Gebrauch 'aufsteigende Kette' macht die Forderung allerdings augenfälliger.

Zu GVPOs:

- In gerichteten Mengen M hat definitionsgemäß jede endliche Teilmenge eine obere Schranke in M. Ist M endlich, so gilt dies auch für M selbst, d.h. M hat ein Supremum in M; für unendliche M gilt das nicht. Die GVPO-Eigenschaft folgt deshalb nicht trivial aus der Eigenschaft gerichtet von Mengen (s. Korollar A.2.6.4).

Existenz kleinster Elemente in VPOs

Lemma A.3.1.3 (Ex. kleinster Elemente in VPOs)

Sei $(P, \sqsubseteq)$ eine VPO, d.h. eine KVPO oder GVPO. Dann gibt es ein eindeutig bestimmtes kleinstes Element in P, bezeichnet mit $\bot$, das durch das Supremum der leeren Kette bzw. Menge gegeben ist, d.h.: $\bot = \bigsqcup \emptyset$.

Korollar A.3.1.4 (Nichtleerheit von VPOs)

Sei $(P, \sqsubseteq)$ eine VPO, d.h. eine KVPO oder GVPO. Dann gilt: $P \neq \emptyset$.

Beachte: Lemma A.3.1.3 gilt nicht für prä-VPOs, d.h. $(P, \sqsubseteq)$ eine prä-VPOs besitzt nicht notwendig ein kleinstes Element.

Beziehungen endlicher POs, KVPOs, GVPOs

Sei P eine endliche Menge und $\sqsubseteq$ eine Relation auf P.

Lemma A.3.1.5 (Endliche POs, prä-VPOs)

Folgende Aussagen sind äquivalent:

1. $(P, \sqsubseteq)$ ist eine partielle Ordnung.
2. $(P, \sqsubseteq)$ ist eine prä-KVPO.
3. $(P, \sqsubseteq)$ ist eine prä-GVPO.

Lemma A.3.1.6 (Endliche POs, KVPOs, GVPOs)

Sei $p \in P$ mit $p \sqsubseteq P$. Dann sind folgende Aussagen äquivalent:

1. $(P, \sqsubseteq)$ ist eine partielle Ordnung.
2. $(P, \sqsubseteq)$ ist eine KVPO.
3. $(P, \sqsubseteq)$ ist eine GVPO.

Äquivalenz von KVPOs und GVPOs

Theorem A.3.1.7 (Äquivalenz)

Sei $(P, \sqsubseteq)$ eine partielle Ordnung. Dann sind folgende Aussagen äquivalent:

1. $(P, \sqsubseteq)$ ist eine KVPO.
2. $(P, \sqsubseteq)$ ist eine GVPO.

Beachte: Spielt die ketten- oder gerichtete-Mengen-Sicht einer vollständigen partiellen Ordnung keine Rolle, so sprechen wir einfacher von einer VPO anstatt genauer von einer KVPO oder GVPO; analog gilt dies für prä-VPOs.

Beispiele für prä-VPOs und VPOs (1)

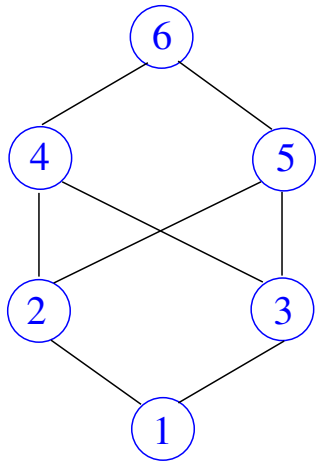
- $(\mathcal{P}(\mathbb{N}), \subseteq)$ ist eine VPO (d.h. eine KVPO und GVPO).
  - Kleinstes Element: $\emptyset$
  - Kleinste obere Schranke $\bigsqcup K$ von K *Kette* $\subseteq \mathcal{P}(\mathbb{N})$: $\bigcup_{K' \in K} K'$
- Die Menge endlicher und unendlicher Zeichenreichen Z, partiell geordnet durch die Präfixrelation $\sqsubseteq_{pfx}$ mit:

$$\forall\, z, z'' \in Z.\ z \sqsubseteq_{pfx} z'' \iff_{df} z = z'' \vee (z \textit{ endlich } \wedge \exists\, z' \in Z.\ z +\!\!+ z' = z'')$$

  ist eine VPO (d.h. eine KVPO und GVPO).
- $(\{-n \mid n \in \mathbb{N}\}, \leq)$ ist eine prä-VPO (d.h. eine prä-KVPO und prä-GVPO), aber keine VPO (d.h. keine KVPO und GVPO).

Beispiele für prä-VPOs und VPOs (2)

- $(\emptyset, \emptyset)$ ist eine prä-VPO (d.h. eine prä-KVPO und prä-GVPO), aber keine VPO (d.h. keine KVPO und GVPO). (Die prä-KVPO- (da keine nichtleeren Ketten in $\emptyset$) und prä-GVPO- (da $\emptyset$ einzige Teilmenge von $\emptyset$ ist und diese definitionsgemäss nicht gerichtet ist) Eigenschaften gelten beide in trivialerweise. Beachte dabei auch, dass aus $P = \emptyset$ auch die Gleichheit $\sqsubseteq = \emptyset \subseteq P \times P$ folgt.)
- Die durch folgendes Hasse-Diagramm gegebene partielle Ordnung $(P, \sqsubseteq)$ ist eine VPO (d.h. eine KVPO u. GVPO).



Beispiele für prä-VPOs und VPOs (3)

- Die Menge endlicher und nichtendlicher Zeichenreihen Z, partiell geordnet durch die folgendermaßen definierte lexikographische Ordnung $\sqsubseteq_{lex}$:

  $$\forall\, s, t \in Z.\ s \sqsubseteq_{lex} t \iff_{df}$$
  $$s = t \vee (\exists\, p\ \mathit{endlich}, s', t' \in Z.\ s = p +\!\!+ s' \wedge t = p +\!\!+ t' \wedge (s' = \varepsilon \ \vee\ s'_1 < t'_1))$$

  wobei ε die leere Zeichenreihe bezeichnet, $w\downarrow_1$ das erste Zeichen einer Zeichenreihe w und $<$ die lexikographische Ordnung auf Zeichen, ist eine VPO (d.h. eine KVPO und GVPO).

(Anti-) Beispiele für VPOs

- $(\mathbb{N}, \leq)$ ist keine VPO (d.h. keine KVPO und GVPO).
- Die Menge endlicher Zeichenreihen Z_{endl}, partiell geordnet durch die
  - Präfixrelation $\sqsubseteq_{pfx}$ definiert durch:

    $$\forall\, s, s' \in Z_{endl}.\ s \sqsubseteq_{pfx} s' \iff_{df} \exists\, s'' \in Z_{endl}.\ s +\!\!+ s'' = s'$$

    ist keine VPO.
  - lexikographische Ordnung $\sqsubseteq_{lex}$ definiert durch:

    $$\forall\, s, t \in Z_{endl}.\ s \sqsubseteq_{lex} t \iff_{df} \exists\, p, s', t' \in Z_{endl}.\ s = p +\!\!+ s' \ \wedge\ t = p +\!\!+ t' \ \wedge\ (s' = \varepsilon \ \vee\ s'\downarrow_1 < t'\downarrow_1)$$

    wobei ε die leere Zeichenreihe bezeichnet, $w\downarrow_1$ das erste Zeichen einer Zeichenreihe w und $<$ die lexikographische Ordnung auf Zeichen ist keine VPO.
- $(\mathcal{P}_{endl}(\mathbb{N}), \subseteq)$ ist keine VPO.

Übungsaufgabe A.3.1.8

Welche der durch folgende Hasse-Diagramme gegebenen partiellen Ordnungen sind (prä-) KVPOs? Welche (prä-) GVPOs?

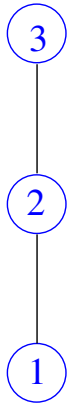
a)

b)

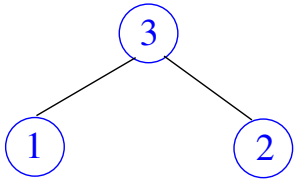
c)

d)

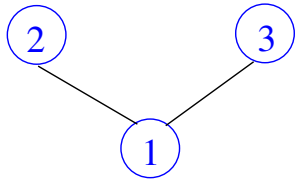
e)



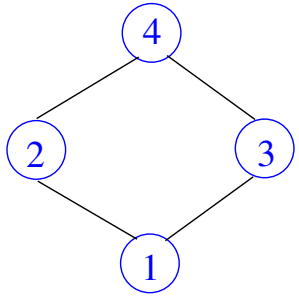
f)



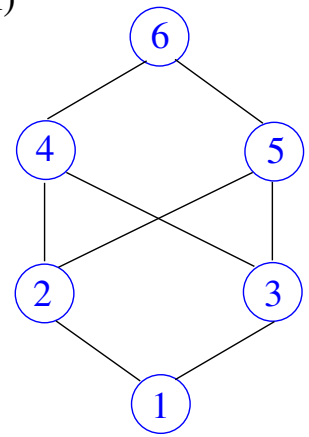
g)



h)



i)



Stark gerichtete VPOs: Eine GVPO-Variante

Zu GVPOs auf Grundlage stark gerichteter Mengen

- Ersetzt man in Definition A.3.1.2 gerichtete durch stark gerichtete Mengen, so erhält man SGVPOs.
- Eingedenk, dass eine stark gerichtete Menge nie leer ist (s. Definition A.2.6.5), gibt es auf Grundlage stark gerichteter Mengen kein Analogon zu prä-GVPOs.
- Eine stark gerichtete Menge M, in der definitionsgemäß jede endliche Teilmenge ein Supremum in M hat, muss selbst kein Supremum in M haben, wenn M unendlich ist. Die SGVPO-Eigenschaft folgt deshalb nicht in trivialer Weise aus der Eigenschaft stark gerichtet von Mengen (s. Korollar A.2.6.4).

Übungsaufgabe A.3.1.9

Betrachte die partielle Ordnung $(\mathbb{N}_0, \sqsubseteq)$ mit $\sqsubseteq =_{df} |$, wobei $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$ bezeichne, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \mid 35$.

Beweise oder widerlege durch Angabe eines Gegenbeispiels:

Das Paar $(\mathbb{N}_0, \sqsubseteq)$ ist eine

1. prä-KVPO
2. KVPO
3. prä-GVPO
4. GVPO
5. SGVPO

A.3.2

Abbildungen auf vollständigen partiellen Ordnungen

Stetige Abbildungen auf KVPOs

Seien $(K, \sqsubseteq_K)$ und $(L, \sqsubseteq_L)$ KVPOs und sei $f \in [K \to L]$ eine Abbildung von K nach L.

Definition A.3.2.1 (Stetige Abbildungen auf KVPOs)

f heißt stetig gdw f ist monoton und

$$\forall\, K' \neq \emptyset \; \mathit{Kette} \subseteq K.\; f(\bigsqcup_K K') =_L \bigsqcup_L f(K')$$

(Erhalt kleinster oberer Schranken)

Beachte: $\forall\, T \subseteq K.\; f(T) =_{df} \{\, f(t) \mid t \in T \}$

Stetige Abbildungen auf GVPOs

Seien $(G, \sqsubseteq_G)$ und $(H, \sqsubseteq_H)$ GVPOs und sei $f \in [G \to H]$ eine Abbildung von G nach H.

Definition A.3.2.2 (Stetige Abbildungen a. GVPOs)

f heißt stetig gdw

$$\forall\, G' \neq \emptyset \textit{ gerichtet } \subseteq G.\ f(G') \text{ gerichtet } \subseteq H \wedge f(\bigsqcup_G G') =_H \bigsqcup_H f(G')$$

(Erhalt kleinster oberer Schranken)

Beachte: $\forall\, T \subseteq G.\ f(T) =_{df} \{ f(t) \mid t \in T \}$

Monotoniecharakterisierung

Seien $(K, \sqsubseteq_K), (L, \sqsubseteq_L)$ KVPOs und $(G, \sqsubseteq_G), (H, \sqsubseteq_H)$ GVPOs.

Lemma A.3.2.3 (Monotoniecharakterisierung)

1. $f : K \to L$ ist monoton

   gdw $\forall\, K' \neq \emptyset$ *Kette* $\subseteq K$.

   $$f(K') \text{ Kette} \subseteq L \wedge f(\bigsqcup_K K') \sqsupseteq_L \bigsqcup_L f(K')$$

2. $g : G \to H$ ist monoton

   gdw $\forall\, G' \neq \emptyset$ *gerichtet* $\subseteq G$.

   $$g(G') \text{ gerichtet} \subseteq H \ \wedge \ g(\bigsqcup_G G') \sqsupseteq_H \bigsqcup_H g(G')$$

Strikte Funktionen auf KVPOs und GVPOs

Seien $(K, \sqsubseteq_K), (L, \sqsubseteq_L)$ KVPOs mit kleinsten Elementen $\bot_K$ bzw. $\bot_L$, seien $(G, \sqsubseteq_G), (H, \sqsubseteq_H)$ GVPOs mit kleinsten Elementen $\bot_G$ bzw. $\bot_H$ und seien $f \in [K \overset{stet}{\to} L]$, $g \in [G \overset{stet}{\to} H]$ stetige Funktionen.

Definition A.3.2.4 (Strikte Funktionen auf VPOs)

f und g heißen strikt, wenn die Gleichheiten

- $f(\bigsqcup_K K') =_L \bigsqcup_L f(K')$, $g(\bigsqcup_G G') =_H \bigsqcup_H g(G')$

auch für $K' = \emptyset$ und $G' = \emptyset$ gelten, d.h., wenn die Gleichheiten

- $f(\bigsqcup_K \emptyset) =_K f(\bot_K) =_L \bot_L =_L \bigsqcup \emptyset$
- $f(\bigsqcup_G \emptyset) =_G g(\bot_G) =_H \bot_H =_H \bigsqcup \emptyset$

erfüllt sind.

A.3.3

Konstruktionsmechanismen für vollständige partielle Ordnungen

Typische KVPO- und GVPO-Konstruktionen

Die im folgenden angebenen Konstruktionsprinzipien gelten für

- KVPOs
- GVPOs

in gleicher Weise; deshalb schreiben wir einfacher stets VPO (statt genauer KVPO und GVPO).

Typische VPO-Konstrukt.: Flache prä-VPOs

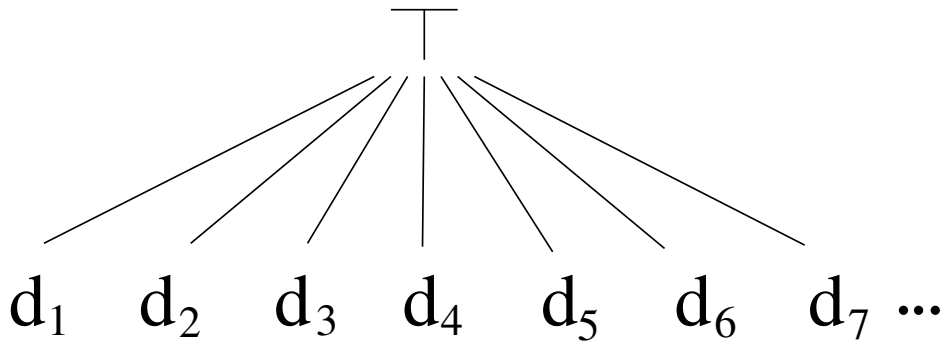
Lemma A.3.3.1 (Flache prä-VPO-Konstruktion)

Sei D eine Menge. Dann gilt:

$(D \dot\cup \{\top\}, \sqsubseteq_{flach})$ mit $\sqsubseteq_{flach}$ definiert durch

$$\forall\, d, e \in D \dot\cup \{\top\}.\ d \sqsubseteq_{flat} e \iff_{df} e = \top \ \vee \ d = e$$

ist eine prä-VPO, eine sog. flache prä-VPO (engl. flat pre-CPO).



Typische VPO-Konstruktionen: Flache VPOs

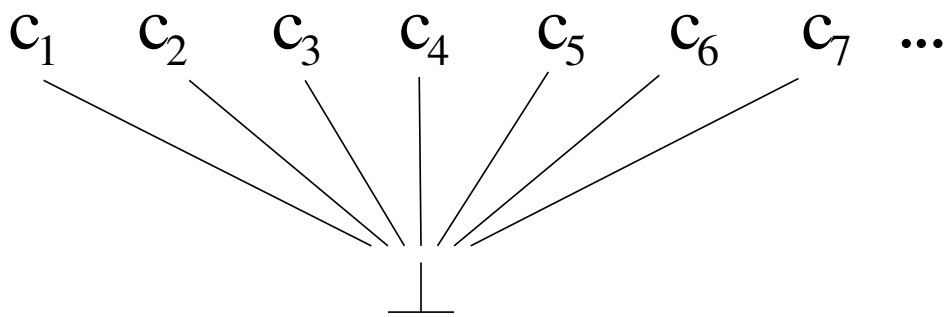
Lemma A.3.3.2 (Flache VPO-Konstruktion)

Sei C eine Menge. Dann gilt:

$(C \dot\cup \{\bot\}, \sqsubseteq_{flach})$ mit $\sqsubseteq_{flach}$ definiert durch

$$\forall\, c, d \in C \dot\cup \{\bot\}.\ c \sqsubseteq_{flach} d \iff_{df} c = \bot \ \vee \ c = d$$

ist eine VPO, eine sog. flache VPO (engl. flat CPO).



Typische VPO-Konstruktionen: Produkte (1)

Lemma A.3.3.3 (Nichtstrikte Produktkonstruktion)

Seien $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ VPOs. Dann gilt:

Das nichtstrikte Produkt (engl. non-strict product) $(\times P_i, \sqsubseteq_\times)$, wobei

- $\times P_i =_{df} P_1 \times P_2 \times \ldots \times P_n$ das kartesische Produkt aller P_i, $1 \leq i \leq n$, ist
- $\sqsubseteq_\times$ punktweise definiert ist durch

$$\forall\, (p_1, \ldots, p_n), (q_1, \ldots, q_n) \in \times P_i.$$
$$(p_1, \ldots, p_n) \sqsubseteq_\times (q_1, \ldots, q_n) \iff_{df} \forall\, i \in \{1, \ldots, n\}.\ p_i \sqsubseteq_i q_i$$

ist eine VPO.

Typische VPO-Konstruktionen: Produkte (2)

Lemma A.3.3.4 (Strikte Produktkonstruktion)

Seien $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ VPOs. Dann gilt:

Das strikte Produkt (engl. strict (or: smash) product) $(\bigotimes P_i, \sqsubseteq_\otimes)$, wobei

- $\bigotimes P_i =_{df} \times P_i$ das kartesische Produkt aller P_i ist
- $\sqsubseteq_\otimes =_{df} \sqsubseteq_\times$ punktweise definiert ist mit folgender zusätzlicher (identifizierender) Setzung:

$$(p_1, \ldots, p_n) = \bot \iff_{df} \exists\, i \in \{1, \ldots, n\}.\ p_i = \bot_i$$

ist eine VPO.

Typische VPO-Konstruktionen: Summen (1)

Lemma A.3.3.5 (Direkte Summenkonstruktion)

Seien $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ VPOs. Dann gilt:

Die direkte Summe (engl. separated (or: direct) sum) $(\bigoplus_\perp P_i, \sqsubseteq_{\oplus_\perp})$, wobei

- $\bigoplus_\perp P_i =_{df} P_1 \dot\cup P_2 \dot\cup \ldots \dot\cup P_n \dot\cup \{\perp\}$ die disjunkte Vereinigung aller P_i, $1 \leq i \leq n$, und $\perp$ ein frisches in keinem P_i enthaltenem Element ist
- $\sqsubseteq_{\oplus_\perp}$ definiert ist durch

  $$\forall\, p, q \in \bigoplus_\perp P_i.\ p \sqsubseteq_{\oplus_\perp} q \iff_{df} p = \perp \ \vee\ (\exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge\ p \sqsubseteq_i q)$$

ist eine VPO.

Typische VPO-Konstruktionen: Summen (2)

Lemma A.3.3.6 (Vereinigungssummenkonstruktion)

Seien $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ VPOs. Dann gilt:

Die Vereinigungssumme (engl. coalesced sum) $(\bigoplus_\vee P_i, \sqsubseteq_{\oplus_\vee})$, wobei

- $\bigoplus_\vee P_i =_{df} P_1 \setminus \{\bot_1\} \dot\cup P_2 \setminus \{\bot_2\} \dot\cup \ldots \dot\cup P_n \setminus \{\bot_n\} \dot\cup \{\bot\}$ die disjunkte Vereinigung aller P_i, $1 \leq i \leq n$, und $\bot$ ein frisches in keinem P_i enthaltenem kleinsten Element ist, das mit jedem der ursprünglichen kleinsten Elemente $\bot_i$ der Mengen P_i identifiziert wird und diese ersetzt, d.h., $\bot =_{df} \bot_i$, $i \in \{1, \ldots, n\}$
- $\sqsubseteq_{\oplus_\vee}$ ist definiert durch

  $\forall\, p, q \in \bigoplus_\vee P_i.\ p \sqsubseteq_{\oplus_\vee} q \iff_{df}$

  $$p = \bot \ \vee\ (\exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge\ p \sqsubseteq_i q)$$

ist eine VPO.

Typ. VPO-Konstruktionen: Funktionenraum

Lemma A.3.3.7 (Stetige Funktionenraumkonstrukt.)

Seien $(P, \sqsubseteq_P)$ und $(Q, \sqsubseteq_Q)$ pVPOs. Dann gilt:

Der Raum stetiger Funktionen (oder: stetige Funktionenraum) (engl. continuous function space) $([P \overset{stet}{\to} Q], \sqsubseteq_{sfr})$, wobei

- $[P \overset{stet}{\to} Q]$ die Menge stetiger Abbildungen von P auf Q ist
- $\sqsubseteq_{sfr}$ punktweise definiert ist durch

$$\forall f, g \in [P \overset{stet}{\to} Q]. f \sqsubseteq_{sfr} g \iff_{df} \forall p \in P. f(p) \sqsubseteq_Q g(p)$$

ist eine prä-VPO. Der stetige Funktionenraum ist eine VPO, wenn $(Q, \sqsubseteq_Q)$ eine VPO ist.

Beachte: Die Definition von $\sqsubseteq_{sfr}$ nutzt nicht aus, dass P eine prä-VPO ist. Diese Forderung ist nur gestellt, um die Definition auf stetige Funktionen zuschneiden zu können.

A.4

Verbände

A.4.1

Verbände, vollständige Verbände

Verbände und vollständige Verbände

Sei $(P, \sqsubseteq)$ eine partielle Ordnung, $P \neq \emptyset$.

Definition A.4.1.1 (Verband)

$(P, \sqsubseteq)$ ist ein Verband (engl. lattice), wenn jede nichtleere endliche Teilmenge P' von P eine kleinste obere und größte untere Schranke in P besitzt.

Definition A.4.1.2 (Vollständiger Verband)

$(P, \sqsubseteq)$ ist ein vollständiger Verband (engl. complete lattice), wenn jede Teilmenge P' von P eine kleinste obere und größte untere Schranke in P besitzt.

Beachte: Verbände und vollständige Verbände sind spezielle partielle Ordnungen.

Eigenschaften vollständiger Verbände

Lemma A.4.1.3 (Existenz extremer Elemente)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband. Dann gibt es ein

1. kleinstes Element in P, bezeichnet mit $\bot$, für das gilt: $\bot = \bigsqcup \emptyset = \bigsqcap P$.
2. größtes Element in P, bezeichnet mit $\top$, für das gilt: $\top = \bigsqcap \emptyset = \bigsqcup P$.

Lemma A.4.1.4 (Charakterisierungslemma)

Sei $(P, \sqsubseteq)$ eine partielle Ordnung. Dann sind folgende Aussagen äquivalent:

1. $(P, \sqsubseteq)$ ist ein vollständiger Verband.
2. Jede Teilmenge von P hat eine kleinste obere Schranke.
3. Jede Teilmenge von P hat eine größte untere Schranke.

Eigenschaften endlicher Verbände

Lemma A.4.1.5 (Endlich impliziert vollständig)

Jeder endliche Verband $(P, \sqsubseteq)$ ist vollständig.

Korollar A.4.1.6 (Endlich impl. Ex. kl./gr. Elem.)

Jeder endliche Verband besitzt ein kleinstes und ein größtes Element.

Vollständige Halbverbände

Sei $(P, \sqsubseteq)$ eine partielle Ordnung, $P \neq \emptyset$.

Definition A.4.1.7 (Vollständiger Halbverband)

$(P, \sqsubseteq)$ ist ein vollständiger

1. Vereinigungshalbverband (engl. join semi-lattice) gdw
   $\forall\, \emptyset \neq Q \subseteq P.\ \bigsqcup Q$ *existiert* $\in P$.
2. Schnitthalbverband (engl. meet semi-lattice) gdw
   $\forall\, \emptyset \neq Q \subseteq P.\ \bigsqcap Q$ *existiert* $\in P$.

Eigenschaften vollständiger Halbverbände (1)

Proposition A.4.1.8 (Extr. Schrank. in vollst. Halbv.)

Ist $(P, \sqsubseteq)$ ein vollständiger

1. Vereinigungshalbverband, so gilt: $\bigsqcup P$ *existiert* $\in P$ (wohingegen $\bigsqcup \emptyset$ ($\widehat{=} \bot$) i.a. nicht in P existiert).
2. Schnitthalbverband, so gilt: $\bigsqcap P$ *existiert* $\in P$ (wohingegen $\bigsqcap \emptyset$ ($\widehat{=} \top$) i.a. nicht in P existiert).

Informell: In einem vollständigen Vereinigungshalbverband muss nicht notwendig ein kleinstes Element existieren, in einem vollständigen Schnitthalbverband nicht notwendig ein größtes Element.

Eigenschaften vollständiger Halbverbände (2)

Lemma A.4.1.9 (Ex. gr. Elem. in vollst. Verein.halbv.)

Sei $(P, \sqsubseteq)$ ein vollständiger Vereinigungshalbverband. Dann gilt:

$\bigsqcup P$ *existiert* $\in P$ und ist das (eindeutig bestimmte i.a. mit $\top$ bezeichnete) größte Element in P, d.h.: $\top = \bigsqcup P$.

Lemma A.4.1.10 (Ex. kl. El. in vollst. Schnitthalbv.)

Sei $(P, \sqsubseteq)$ ein vollständiger Schnitthalbverband. Dann gilt:

$\bigsqcap P$ *existiert* $\in P$ und ist das (eindeutig bestimmte i.a. mit $\bot$ bezeichnete) kleinste Element in P, d.h.: $\bot = \bigsqcap P$.

Charakterisier. oberer u. unterer Schranken (1)

...in vollständigen Halbverbänden.

Lemma A.4.1.11 (Ch. o./u. Schrank. in vollst. Halbv.)

1. Sei $(P, \sqsubseteq)$ ein vollständiger Vereinigungshalbverband und $Q \subseteq P$ eine Teilmenge von P.

   Hat Q untere Schranken in P, d.h. ist $\{p \in P \mid p \sqsubseteq Q\} \neq \emptyset$, so gilt: $\bigsqcap Q$ *existiert* $\in P$ und

   $$\bigsqcap Q = \bigsqcup \{p \in P \mid p \sqsubseteq Q\}$$

2. Sei $(P, \sqsubseteq)$ ein vollständiger Schnitthalbverband und $Q \subseteq P$ eine Teilmenge von P.

   Hat Q obere Schranken in P, d.h. ist $\{p \in P \mid Q \sqsubseteq p\} \neq \emptyset$, so gilt: $\bigsqcup Q$ *existiert* $\in P$ und

   $$\bigsqcup Q = \bigsqcap \{p \in P \mid Q \sqsubseteq p\}$$

Charakterisier. oberer u. unterer Schranken (2)

Lemma A.4.1.12 (Kl./gr. Elem. in vollst. Halbverb.)

Ist $(P, \sqsubseteq)$ ein vollständiger

1. Vereinigungshalbverband und $\bigsqcup \emptyset$ *existiert* $\in P$, dann ist $\bigsqcup \emptyset$ das (eindeutig bestimmte mit $\bot$ bezeichnete) kleinste Element in P, d.h.: $\bot = \bigsqcup \emptyset$.
2. Schnitthalbverband und $\bigsqcap \emptyset$ *existiert* $\in P$, dann ist $\bigsqcap \emptyset$ das (eindeutig bestimmte mit $\top$ bezeichnete) größte Element in P, d.h.: $\top = \bigsqcap \emptyset$.

Bezieh. zw. vollst. Halbverb. und Verbänden

Lemma A.4.1.13 (Vollst. Halbverbände u. Verbände)

Ist $(P, \sqsubseteq)$ ein vollständiger

1. Vereinigungshalbverband und $\bigsqcup \emptyset$ *existiert* $\in P$
2. Schnitthalbverband und $\bigsqcap \emptyset$ *existiert* $\in P$

so ist $(P, \sqsubseteq)$ ein vollständiger Verband.

Übungsaufgabe A.4.1.14

Zeige oder widerlege durch Angabe eines Gegenbeispiels:

Ist $(P, \sqsubseteq)$ ein vollständiger Verband, so ist

1. $(P\backslash\{\bot\}, \sqsubseteq_{\backslash\bot})$ ein vollständiger Vereinigungshalbverband.
2. $(P\backslash\{\top\}, \sqsubseteq_{\backslash\top})$ ein vollständiger Schnitthalbverband.

wobei $\sqsubseteq_{\backslash\bot}$ und $\sqsubseteq_{\backslash\top}$ die Einschränkungen von $\sqsubseteq$ von P auf $P\backslash\{\bot\}$ bzw. $P\backslash\{\top\}$ bezeichnen.

Bezieh. zw. Verbänden und vollst. part. Ordn.

Lemma A.4.1.15 (Vollständige Verbände und VPOs)

Ist $(P, \sqsubseteq)$ ein vollständiger Verband, so ist $(P, \sqsubseteq)$ eine VPO (d.h. eine KVPO und GVPO).

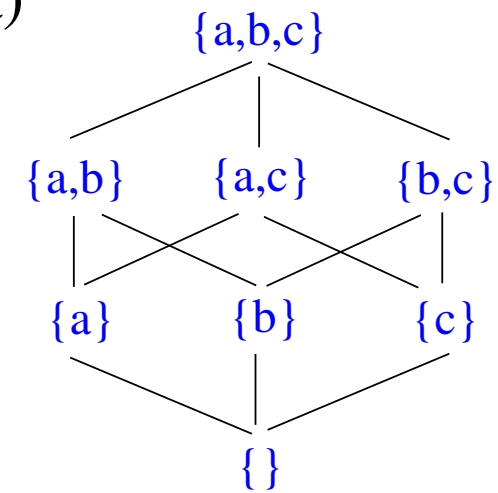
Korollar A.4.1.16 (Endliche Verbände und VPOs)

Ist $(P, \sqsubseteq)$ ein endlicher Verband, so ist $(P, \sqsubseteq)$ eine VPO (d.h. eine KVPO und GVPO).

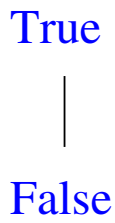
Beachte: Lemma A.4.1.15 gilt nicht für Verbände.

Beispiele vollständiger Verbände

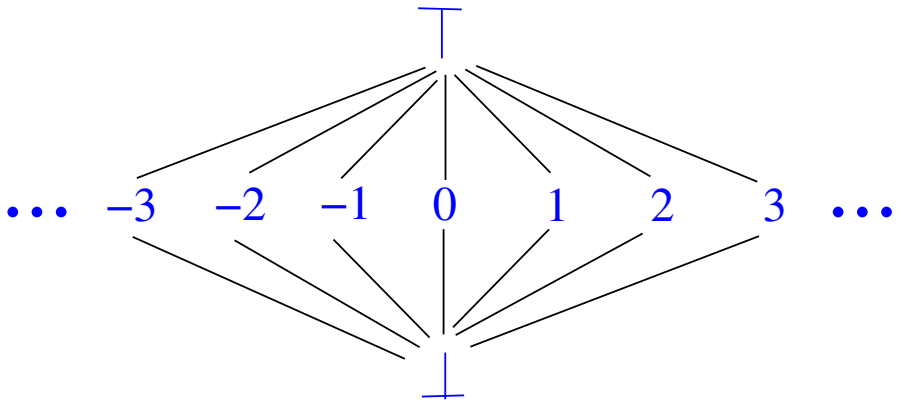
a)



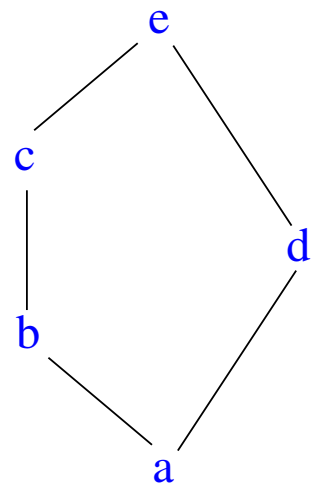
b)



c)

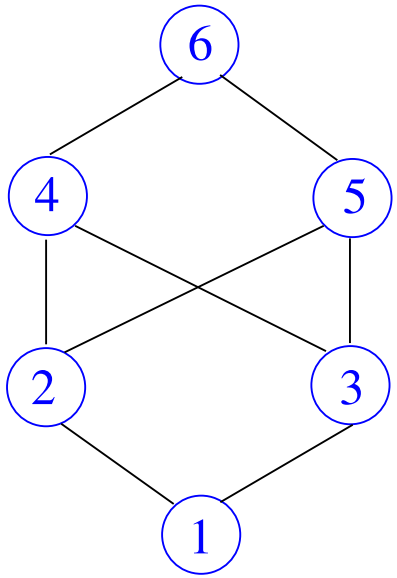


d)



(Anti-) Beispiele

- Die durch folgendes Hasse-Diagramm gegebene partielle Ordnung $(P, \sqsubseteq)$ ist kein Verband (wohl aber eine VPO).



- $(\mathcal{P}_{fin}(\mathbb{N}), \subseteq)$ ist kein vollständiger Verband (und auch keine VPO).

Übungsaufgabe A.4.1.17

Welche der durch die folgenden Hasse-Diagramme gegebenen partiellen Ordnungen sind Verbände? Welche sind vollständige Verbände?

a)

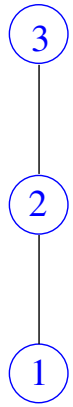
{ }

b)

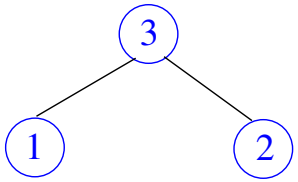
c)

d)

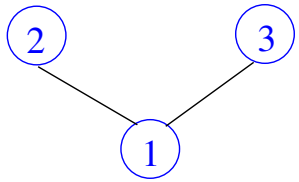
e)



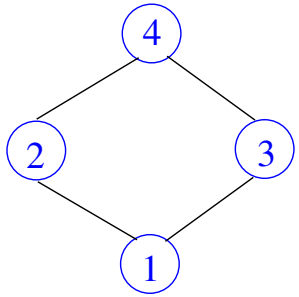
f)



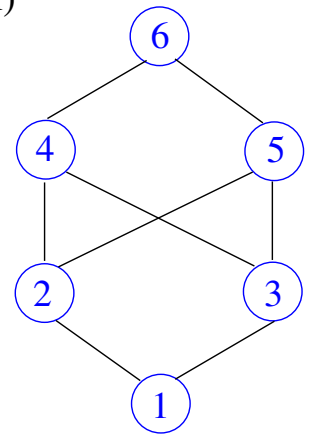
g)



h)



i)



Übungsaufgabe A.4.1.18

Betrachte die partielle Ordnung $(\mathbb{N}_0, \sqsubseteq)$ mit $\sqsubseteq =_{df} |$, wobei $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$ bezeichne, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \,|\, 35$.

Beweise oder widerlege durch Angabe eines Gegenbeispiels:

$(\mathbb{N}_0, \sqsubseteq)$ ist ein

1. Verband
2. vollständiger Verband
3. vollständiger Vereinigungshalbverband
4. vollständiger Schnitthalbverband

Zusammenfassung, Überblick

Korollar A.4.1.19

Sei $P \neq \emptyset$ nichtleere Menge und $\sqsubseteq$ Relation auf P. Dann gilt:

$(P, \sqsubseteq)$ endlicher Verband (L. A.4.1.5) $\vee$

$(P, \sqsubseteq)$ vollständiger Vereinigungshalbverband und

$\bigsqcup \emptyset$ *existiert* $\in P$ (L. A.4.1.13(1)) $\vee$

$(P, \sqsubseteq)$ vollständiger Schnitthalbverband und

$\bigsqcap \emptyset$ *existiert* $\in P$ (L. A.4.1.13(2))

$\Rightarrow$ $(P, \sqsubseteq)$ vollständiger Verband

(D. A.4.1.2 & L. A.4.1.14) $\Rightarrow$ $(P, \sqsubseteq)$ Verband & vollständige partielle Ordnung

(D. A.4.1.1 & D. A.3.1.1/2) $\Rightarrow$ $(P, \sqsubseteq)$ partielle Ordnung

(D. A.2.1.2) $\Rightarrow$ $(P, \sqsubseteq)$ Halbordnung

Übungsaufgabe A.4.1.20

Bezeichne mit

$$\mathcal{HO}, \mathcal{PO}, \mathcal{V}, \mathcal{VPO}, \mathcal{VV}, \mathcal{EV}, \mathcal{VVHV}, \mathcal{VVHV}_\perp, \mathcal{VSHV}, \mathcal{VSHV}^\top$$

die Mengen aller Halbordnungen $\mathcal{HO}$, partiellen Ordnungen $\mathcal{PO}$, Verbände $\mathcal{V}$, vollständigen partiellen Ordnungen $\mathcal{VPO}$, vollständigen Verbände $\mathcal{VV}$, endlichen Verbände $\mathcal{EV}$, vollständigen Vereinigungshalbverbände ohne/mit kleinstem Element $\mathcal{VVHV}/\mathcal{VVHV}_\perp$ und Schnitthalbverbände ohne/mit größtem Element $\mathcal{VSHV}/\mathcal{VSHV}^\top$.

1. Welche weiteren Implikationen oder Äquivalenzen gelten über die in Korollar A.4.1.19 genannten hinaus? (Beweis oder Gegenbeispiel)
2. Welche Inklusionen oder (Mengen-) Gleichheiten gelten zwischen $\mathcal{HO}$, $\mathcal{PO}$, $\mathcal{V}$, usw.? (Beweis oder Gegenbeispiel)

A.4.2

Distributive, additive Abbildungen auf Verbänden

Distributive, additive Abb. auf Verbänden

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $f \in [P \to P]$ eine Abbildung auf P.

Definition A.4.2.1 (Distributive, additive Abbildung)

f heißt

1. distributiv (oder: $\sqcap$-stetig) (engl. distributive, $\sqcap$-continuous) gdw
$$\forall \emptyset \neq P' \subseteq P.\ f(\bigsqcap P') = \bigsqcap f(P')$$
(Erhalt größter unterer Schranken)

2. additiv (oder: $\sqcup$-stetig) (engl. additive, $\sqcup$-continuous) gdw
$$\forall \emptyset \neq P' \subseteq P.\ f(\bigsqcup P') = \bigsqcup f(P')$$
(Erhalt kleinster oberer Schranken)

Beachte: $\forall\, T \subseteq P.\ f(T) =_{df} \{\, f(t) \mid t \in T \}$

Monotoniecharakterisierung

...über den Erhalt größter unterer und kleinster oberer Schranken:

Lemma A.4.2.2 (Monotoniecharakterisierung)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $f \in [P \to P]$ eine Abbildung auf P. Dann gilt:

$$f \text{ ist monoton} \iff \forall\, P' \subseteq P.\ \ f(\bigsqcap P') \sqsubseteq \bigsqcap f(P')$$
$$\iff \forall\, P' \subseteq P.\ \ f(\bigsqcup P') \sqsupseteq \bigsqcup f(P')$$

Beachte: $\forall\, T \subseteq P.\ f(T) =_{df} \{\, f(t) \mid t \in T \,\}$

Nützl. Resultate über Mon., Distr., Additivität

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $f \in [P \to P]$ eine Abbildung auf P.

Lemma A.4.2.3

f ist monoton, wenn f distributiv oder additiv ist.

(d.h., Distributivität/Additivität implizieren Monotonie)

Beachte: Distributivität und Additivität sind unabhängig voneinander; keine Eigenschaft impliziert die andere.

A.4.3

Verbandshomomorphismen und -isomorphismen

Verbandshomomorphismen u. -isomorphismen

Seien $(P, \sqsubseteq_P)$, $(R, \sqsubseteq_R)$ zwei Verbände und $f \in [P \to R]$ eine Abbildung von P nach R.

Definition A.4.3.1 (Verbandshomomorphismus)

f heißt Verbandshomomorphismus, wenn gilt:

$$\forall\, p, q \in P.\ f(p \sqcup_P q) = f(p) \sqcup_Q f(q) \wedge$$
$$f(p \sqcap_P q) = f(p) \sqcap_Q f(q)$$

Definition A.4.3.2 (Verbandsisomorphismus)

1. f heißt Verbandsisomorphismus, wenn f ein Verbandshomomorphismus und bijektiv ist.
2. $(P, \sqsubseteq_P)$ und $(R, \sqsubseteq_R)$ heißen isomorph, wenn es einen Verbandsisomorphismus zwischen P und R gibt.

Nützliche Resultate (1)

Seien $(P, \sqsubseteq_P)$, $(R, \sqsubseteq_R)$ zwei Verbände und $f \in [P \to R]$ eine Abbildung von P nach R.

Lemma A.4.3.3

$$f \in [P \overset{hom}{\to} R] \;\Rightarrow\; f \in [P \overset{mon}{\to} R]$$

Die Rückrichtung der Implikation aus Lemma A.4.3.3 gilt nicht, aber folgende schwächere Beziehung ist gültig:

Lemma A.4.3.4

$$f \in [P \overset{mon}{\to} R] \;\Rightarrow$$
$$\forall\, p, q \in P.\; f(p \sqcup_P q) \sqsupseteq_Q f(p) \sqcup_Q f(q) \;\wedge$$
$$f(p \sqcap_P q) \sqsubseteq_Q f(p) \sqcap_Q f(q)$$

Nützliche Resultate (2)

Seien $(P, \sqsubseteq_P)$, $(R, \sqsubseteq_R)$ zwei Verbände und $f \in [P \to R]$ eine Abbildung von P nach R.

Lemma A.4.3.5

$$f \in [P \overset{iso}{\to} R] \;\Rightarrow\; f^{-1} \in [R \overset{iso}{\to} P]$$

Lemma A.4.3.6

$$f \in [P \overset{iso}{\to} R] \Longleftrightarrow f \in [P \overset{po-hom}{\to} R] \text{ wrt } \sqsubseteq_P \text{ and } \sqsubseteq_Q$$

A.4.4

Modulare, distributive und Boolesche Verbände

Modulare Verbände

Sei $(P, \sqsubseteq)$ ein Verband mit Schnittoperation $\sqcap$ und Vereinigungsoperation $\sqcup$.

Lemma A.4.4.1

$$\forall\, p, q, r \in P.\ p \sqsubseteq r \Rightarrow p \sqcup (q \sqcap r) \sqsubseteq (p \sqcup q) \sqcap r$$

Definition A.4.4.2 (Modularer Verband)

$(P, \sqsubseteq)$ heißt modular, wenn gilt:

$$\forall\, p, q, r \in P.\ p \sqsubseteq r \Rightarrow p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap r$$

Charakterisierung modularer Verbände

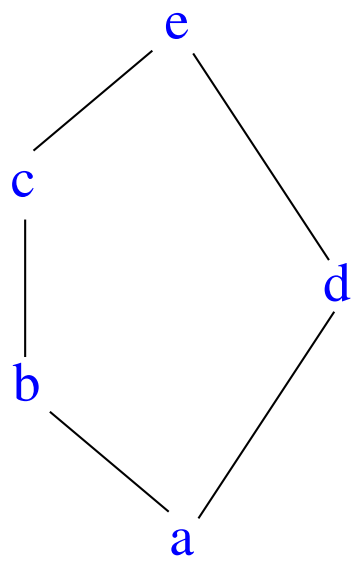
Theorem A.4.4.3 (Charakt. modularer Verbände)

Ein Verband $(P, \sqsubseteq)$ ist

1. modular gdw

   $$\forall\, p, q, r \in P.\ p \sqsubseteq q,\ p \sqcap r = q \sqcap r,\ p \sqcup r = q \sqcup r \Rightarrow p = q$$

2. nicht modular gdw $(P, \sqsubseteq)$ enthält einen Unterverband, der isomorph ist zum Verband:



Distributive Verbände

Sei $(P, \sqsubseteq)$ ein Verband mit Schnittoperation $\sqcap$ und Vereinigungsoperation $\sqcup$.

Lemma A.4.4.4

1. $\forall\, p, q, r \in P.\ p \sqcup (q \sqcap r) \sqsubseteq (p \sqcup q) \sqcap (p \sqcup r)$
2. $\forall\, p, q, r \in P.\ p \sqcap (q \sqcup r) \sqsupseteq (p \sqcap q) \sqcup (p \sqcap r)$

Definition A.4.4.5 (Distributiver Verband)

$(P, \sqsubseteq)$ heißt distributiv, wenn gilt:

1. $\forall\, p, q, r \in P.\ p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap (p \sqcup r)$
2. $\forall\, p, q, r \in P.\ p \sqcap (q \sqcup r) = (p \sqcap q) \sqcup (p \sqcap r)$

Hin zur Charakt. distributiver Verbände

Lemma A.4.4.6

Folgende Aussagen sind äquivalent:

1. $\forall\, p, q, r \in P.\ p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap (p \sqcup r)$
2. $\forall\, p, q, r \in P.\ p \sqcap (q \sqcup r) = (p \sqcap q) \sqcup (p \sqcap r)$

Somit reicht es aus, in Definition A.4.4.5 die Gültigkeit von Eigenschaft (1) oder von Eigenschaft (2) zu fordern.

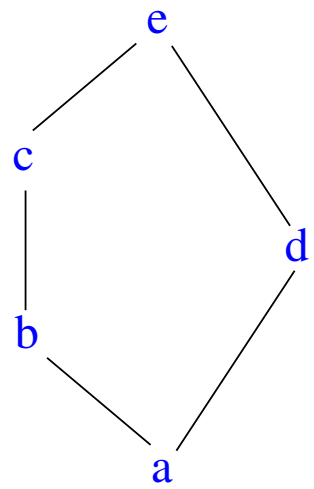
Charakterisierung distributiver Verbände

Sei $(P, \sqsubseteq)$ ein Verband.

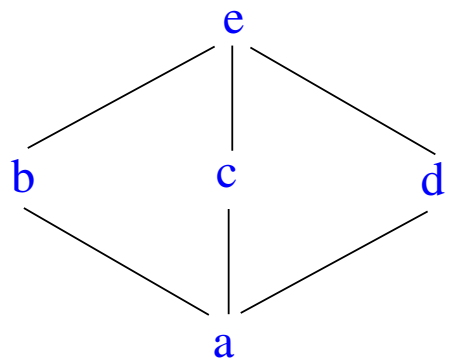
Theorem A.4.4.7 (Charakt. distributiver Verbände)

$(P, \sqsubseteq)$ ist nicht distributiv gdw $(P, \sqsubseteq)$ enthält einen Unterverband, der isomorph ist zu einem der folgenden zwei Verbände:

a)



b)



Korollar A.4.4.8

Ist $(P, \sqsubseteq)$ distributiv, so ist $(P, \sqsubseteq)$ auch modular.

Boolesche Verbände

Sei $(P, \sqsubseteq)$ ein Verband mit Schnittoperation $\sqcap$, Vereinigungsoperation $\sqcup$, kleinstem Element $\bot$ und größtem Element $\top$.

Definition A.4.4.9 (Komplement)

Seien $p, q \in P$. Dann gilt:

1. q heißt ein Komplement von p, wenn gilt: $p \sqcup q = \top$ und $p \sqcap q = \bot$.
2. P heißt komplementär, wenn alle Elemente in P ein Komplement besitzen.

Definition A.4.4.10 (Boolescher Verband)

$(P, \sqsubseteq)$ heißt Boolesch, wenn $(P, \sqsubseteq)$ komplementär und distributiv ist und $\bot \neq \top$ gilt.

Beachte: Ist $(P, \sqsubseteq)$ Boolesch, so hat jedes Element $p \in P$ ein eindeutig bestimmtes Komplement in P, bezeichnet mit $\bar{p}$.

Nützliche Resultate

Lemma A.4.4.11

Seien $(P, \sqsubseteq)$ ein Boolescher Verband und $p, q, r \in P$. Dann gilt:

1. $\bar{\bar{p}} = p$ (Involutionsgesetz)
2. $\overline{p \sqcup q} = \bar{p} \sqcap \bar{q}, \quad \overline{p \sqcap q} = \bar{p} \sqcup \bar{q}$ (De Morgansche Gesetze)
3. $p \sqsubseteq q \iff \bar{p} \sqcup q = \top \iff p \sqcap \bar{q} = \bot$
4. $p \sqsubseteq q \sqcup r \iff p \sqcap \bar{q} \sqsubseteq r \iff \bar{q} \sqsubseteq \bar{p} \sqcup r$

Boolescher Verbandshomo-, -isomorphismus

Seien $(P, \sqsubseteq_P)$, $(Q, \sqsubseteq_Q)$ zwei Boolesche Verbände und $f \in [P \to Q]$ eine Funktion von P nach Q.

Definition A.4.4.12 (Boolescher V.-Homorphismus)

f heißt Boolescher Verbandshomomorphismus, wenn f ein Ver- bandshomomorphismus ist und es gilt:

$$\forall\, p \in P.\ f(\bar{p}) = \overline{f(p)}$$

Definition A.4.4.13 (Boolescher V.-Isomorphismus)

f heist Boolescher Verbandsisomorphismus, wenn f ein Boolescher Verbandshomomorphismus und bijektiv ist.

Nützliche Resultate

Seien $(P, \sqsubseteq_P)$, $(Q, \sqsubseteq_Q)$ zwei Boolesche Verbände und $f \in [P \overset{bhom}{\to} Q]$ ein Boolescher Verbandshomomorphismus von P nach Q.

Lemma A.4.4.14

$$f(\bot) = \bot \wedge f(\top) = \top$$

Lemma A.4.4.15

f ist ein Boolescher Verbandsisomorphismus gdw

$$f(\bot) = \bot \wedge f(\top) = \top$$

Zusammenfassung, Überblick

Korollar A.4.4.16

Sei $P \neq \emptyset$ nichtleere Menge und $\sqsubseteq$ Relation auf P. Dann gilt:

$$
\begin{array}{rcl}
 & & (P, \sqsubseteq) \text{ Boolescher Verband} \\
(\text{Def. A.4.4.10}) & \Rightarrow & (P, \sqsubseteq) \text{ Distributiver Verband} \\
(\text{Kor. A.4.4.8}) & \Rightarrow & (P, \sqsubseteq) \text{ Modularer Verband} \\
(\text{Def. A.4.4.2}) & \Rightarrow & (P, \sqsubseteq) \text{ Verband} \\
(\text{Def. A.4.1.1}) & \Rightarrow & (P, \sqsubseteq) \text{ partielle Ordnung} \\
(\text{Def. A.2.1.2}) & \Rightarrow & (P, \sqsubseteq) \text{ Halbordnung}
\end{array}
$$

Korollar A.4.4.17

$$\mathcal{HO} \supset \mathcal{PO} \supset \mathcal{V} \supset \mathcal{MV} \supset \mathcal{DV} \supset \mathcal{BV}$$

wobei alle Inklusionen echt sind und $\mathcal{HO}$, $\mathcal{PO}$, $\mathcal{V}$, $\mathcal{MV}$, $\mathcal{DV}$ und $\mathcal{BV}$ die Mengen aller Halbordnungen, partiellen Ordnungen, Verbände, modularen, distributiven und Booleschen Verbände bezeichnen.

Übungsaufgabe A.4.4.18

Betrachte die partielle Ordnung $(\mathbb{N}_0, \sqsubseteq)$ mit $\sqsubseteq =_{df} |$, wobei $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$ bezeichne, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \mid 35$.

Beweise oder widerlege durch Angabe eines Gegenbeispiels:

$(\mathbb{N}_0, \sqsubseteq)$ ist ein

1. modularer Verband
2. distributiver Verband
3. Boolescher Verband

A.4.5

Konstruktionsmechanismen für Verbände

Typ. Verbandskonstrukt.: Flache Verbände

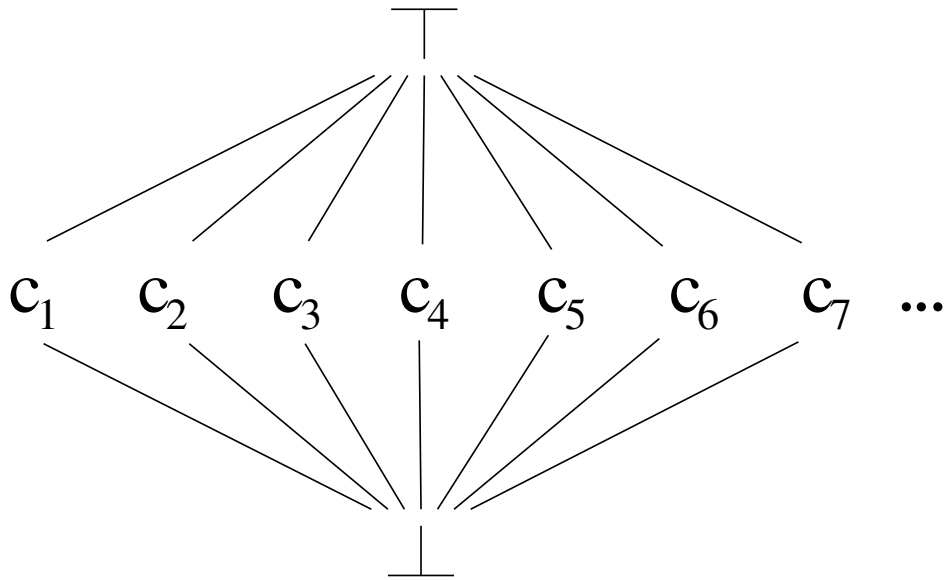
Lemma A.4.5.1 (Flache Verbandskonstruktion)

Sei C eine Menge. Dann gilt:

$(C \dot\cup \{\bot, \top\}, \sqsubseteq_{flach})$ mit $\sqsubseteq_{flach}$ definiert durch

$$\forall\, c, d \in C \dot\cup \{\bot, \top\}.\ c \sqsubseteq_{flach} d \iff_{df} c = \bot \vee c = d \vee d = \top$$

ist ein vollständiger Verband, ein sog. flacher Verband (engl. flat lattice or diamond lattice).



Typ. Verbandskonstrukt.: Produkte, Summen,...

Analog zum Konstruktionsmechanismus für flache VPOs übertragen sich auch die Konstruktionsmechanismen für

- nichtstrikte Produkte
- strikte Produkte
- direkte Summen
- Vereinigungssummen
- stetige (genauer: additive, distributive) Funktionenräume

von VPOs auf (vollständige) Verbände (s. Anhang A.3.3).

A.4.6

Ordnungstheoretische und algebraische Verbandssicht

Motivation

Definition A.4.1.1 führt Verbände als spezielle

- geordnete Mengen $(P, \sqsubseteq)$

ein, was einer

- ordnungstheoretischen Verbandssicht entspricht.

Alternativ können Verbände als spezielle

- algebraische Strukturen $(P, \sqcap, \sqcup)$

eingeführt werden, was einer

- algebraischen Verbandssicht entspricht.

In der Folge zeigen wir, dass beide Sichten gleichwertig sind:

- Ordnungstheoretisch eingeführte Verbände können algebraisch aufgefasst werden und umgekehrt.

Verbände als algebraische Strukturen

Definition A.4.6.1 (Algebraischer Verband)

Ein algebraischer Verband ist eine algebraische Struktur $(P, \sqcap, \sqcup)$, wobei

- $P \neq \emptyset$ eine nichtleere Menge ist.
- $\sqcap, \sqcup : P \times P \to P$ zwei Verknüpfungen sind, so dass für alle Elemente $p, q, r \in P$ folgende Rechengesetze gelten (Infix-Notation):
  - Kommutativgesetze: $p \sqcap q = q \sqcap p$, $p \sqcup q = q \sqcup p$
  - Assoziativgesetze: $(p \sqcap q) \sqcap r = p \sqcap (q \sqcap r)$, $(p \sqcup q) \sqcup r = p \sqcup (q \sqcup r)$
  - Absorptionsgesetze: $(p \sqcap q) \sqcup p = p$, $(p \sqcup q) \sqcap p = p$

Eigenschaften algebraischer Verbände

Sei $(P, \sqcap, \sqcup)$ ein algebraischer Verband.

Lemma A.4.6.2 (Idempotenzgesetze)

Für alle $p \in P$ erfüllen die Verknüpfungen $\sqcap, \sqcup : P \times P \to P$ folgende Gesetze:

- Idempotenzgesetze: $p \sqcap p = p$
  $p \sqcup p = p$

Lemma A.4.6.3

Für alle $p, q \in P$ erfüllen die Verknüpfungen $\sqcap, \sqcup : P \times P \to P$ folgende Äquivalenzen:

1. $p \sqcap q = p \iff p \sqcup q = q$
2. $p \sqcap q = p \sqcup q \iff p = q$

Induzierte (partielle) Ordnung

Sei $(P, \sqcap, \sqcup)$ ein algebraischer Verband.

Lemma A.4.6.4

Die Relation $\sqsubseteq \subseteq P \times P$ auf P definiert durch

$$\forall\, p, q \in P.\ p \sqsubseteq q \iff_{df} p \sqcap q = p$$

ist eine partielle Ordnung auf P, d.h., $\sqsubseteq$ ist reflexiv, transitiv und antisymmetrisch.

Definition A.4.6.5 (Induzierte partielle Ordnung)

Die Relation $\sqsubseteq$ aus Lemma A.4.6.4 heißt induzierte (partielle) Ordnung auf $(P, \sqcap, \sqcup)$.

Eigenschaften induzierter partieller Ordnungen

Sei $(P, \sqcap, \sqcup)$ ein algebraischer Verband und $\sqsubseteq$ die induzierte partielle Ordnung auf $(P, \sqcap, \sqcup)$.

Lemma A.4.6.6

Für alle $p, q \in P$ existieren Infimum ($\widehat{=}$ größte untere Schranke) und Supremum ($\widehat{=}$ kleinste obere Schranke) der Menge $\{p, q\}$ und sind durch die Bilder der Verknüpfungen $\sqcap$ bzw. $\sqcup$ angewendet auf p und q gegeben, d.h.:

$$\forall\, p, q \in P.\ \bigsqcap\{p, q\} = p \sqcap q \ \wedge\ \bigsqcup\{p, q\} = p \sqcup q$$

Lemma A.4.6.6 kann induktiv ausgedehnt werden:

Lemma A.4.6.7

Sei $\emptyset \neq Q \subseteq P$ eine nichtleere endliche Teilmenge von P. Dann gilt:

$$\exists\, gus, kos \in P.\ gus = \bigsqcap Q \ \wedge\ kos = \bigsqcup Q$$

Algebraische Verbände ordnungstheoretisch

Korollar A.4.6.8 (Von $(P, \sqcap, \sqcup)$ zu $(P, \sqsubseteq)$)

Sei $(P, \sqcap, \sqcup)$ ein algebraischer Verband. Dann gilt:

$(P, \sqsubseteq)$, wobei $\sqsubseteq$ die induzierte partielle Ordnung auf $(P, \sqcap, \sqcup)$ ist, ist ein ordnungstheoretischer Verband im Sinn von Definition A.4.1.1.

Induzierte algebraische Verknüpfungen

Sei $(P, \sqsubseteq)$ ein ordnungstheoretischer Verband.

Definition A.4.6.9 (Induzierte algebraische Verkn.)

Die partielle Ordnung $\sqsubseteq$ von $(P, \sqsubseteq)$ induziert zwei Verknüpfungen $\sqcap$ und $\sqcup$ auf $P \times P$ und P definiert durch:

1. $\forall\, p, q \in P.\ p \sqcap q =_{df} \bigsqcap\{p, q\}$
2. $\forall\, p, q \in P.\ p \sqcup q =_{df} \bigsqcup\{p, q\}$

Eigensch. der induz. algebraischen Verkn. (1)

Seien $(P, \sqsubseteq)$ ein ordnungstheoretischer Verband und $\sqcap$ und $\sqcup$ die von $(P, \sqsubseteq)$ induzierten algebraischen Verknüpfungen.

Lemma A.4.6.10

Seien $p, q \in P$. Dann sind folgende Aussagen äquivalent:

1. $p \sqsubseteq q$
2. $p \sqcap q = p$
3. $p \sqcup q = q$

Eigensch. der induz. algebraischen Verkn. (2)

Seien $(P, \sqsubseteq)$ ein ordnungstheoretischer Verband und $\sqcap$ und $\sqcup$ die von $(P, \sqsubseteq)$ induzierten algebraischen Verknüpfungen.

Lemma A.4.6.11

Für alle $p, q, r \in P$ erfüllen die induzierten Verknüpfungen $\sqcap$ und $\sqcup$ folgende Gesetze:

1. Kommutativgesetze: $p \sqcap q = q \sqcap p$
   $p \sqcup q = q \sqcup p$
2. Assoziativgesetze: $(p \sqcap q) \sqcap r = p \sqcap (q \sqcap r)$
   $(p \sqcup q) \sqcup r = p \sqcup (q \sqcup r)$
3. Absorptionsgesetze: $(p \sqcap q) \sqcup p = p$
   $(p \sqcup q) \sqcap p = p$
4. Idempotenzgesetze: $p \sqcap p = p$
   $p \sqcup p = p$

Ordnungstheoretische Verbände algebraisch

Korollar A.4.6.12 (Von $(P, \sqsubseteq)$ zu $(P, \sqcap, \sqcup)$)

Sei $(P, \sqsubseteq)$ ein ordnungstheoretischer Verband. Dann gilt:

$(P, \sqcap, \sqcup)$, wobei $\sqcap$ und $\sqcup$ die von $(P, \sqsubseteq)$ induzierten Verknüpfungen sind, ist ein algebraischer Verband im Sinn von Definition A.4.6.1.

Äquivalenz (1)

...der ordnungtheoretischen und algebraischen Sicht von Verbänden.

Von ordnungsheoretischen zu algebraischen Verbänden:

- Ein ordnungstheoretischer Verband $(P, \sqsubseteq)$ kann durch den Übergang von $(P, \sqsubseteq)$ zu $(P, \sqcap, \sqcup)$, wobei $\sqcap$ und $\sqcup$ die induzierten Verknüpfungen von $(P, \sqsubseteq)$ sind, als algebraischer Verband aufgefasst werden

Von algebraischen zu ordnungstheoretischen Verbänden:

- Ein algebraischer Verband $(P, \sqcap, \sqcup)$ kann durch den Übergang von $(P, \sqcap, \sqcup)$ zu $(P, \sqsubseteq)$, wobei $\sqsubseteq$ die von $(P, \sqcap, \sqcup)$ induzierte partielle Ordnung ist, als ordnungstheoretischer Verband aufgefasst werden.

Äquivalenz (2)

Zusammen erlaubt uns das, einfach(er) von einem Verband P zu sprechen und lediglich präziser von P als

- ordnungstheoretischem Verband $(P, \sqsubseteq)$
- algebraischem Verband $(P, \sqcap, \sqcup)$

um herauszustreichen, dass wir P abhängig vom Kontext als spezielle geordnete Menge oder algebraische Struktur sehen.

Tief und Hoch vs. Null und Eins (1)

Sei P ein Verband mit kleinstem und größtem Element.

Betrachten wir P

- ordnungstheoretisch in der Form $(P, \sqsubseteq)$, ist es zweckmässig von seinem kleinsten und größten Element konzeptuell als Tief $\bot$ und Hoch $\top$ bzgl. $\sqsubseteq$ zu denken mit:
  - Tief $\bot \in P$: $\bot = \bigsqcup \emptyset$
  - Hoch $\top \in P$: $\top = \bigsqcap \emptyset$
- algebraisch in der Form $(P, \sqcap, \sqcup)$, ist es zweckmäßig von seinem kleinsten und größten Element konzeptuell als Null $\mathbf{0}$ und Eins $\mathbf{1}$ bzgl. $\sqcap$ und $\sqcup$ zu denken, wobei $(P, \sqcap, \sqcup)$ (bei Existenz eindeutig bestimmte) Null- und Einselemente hat, für die gilt:
  - Null-Element $\mathbf{0} \in P$: $\forall\, p \in P.\ p \sqcup \mathbf{0} = p$
  - Eins-Element $\mathbf{1} \in P$: $\forall\, p \in P.\ p \sqcap \mathbf{1} = p$

Tief und Hoch vs. Null und Eins (2)

Lemma A.4.6.13

Sei P ein Verband. Dann gilt:

1. $(P, \sqsubseteq)$ besitzt ein Tief-Element $\bot$ gdw $(P, \sqcap, \sqcup)$ besitzt ein Null-Element $\mathbf{0}$; existieren $\bot$ und $\mathbf{0}$ gilt:

$$\left(\bigsqcup \emptyset = \right) \bot = \mathbf{0}$$

2. $(P, \sqsubseteq)$ besitzt ein Hoch-Element $\top$ gdw $(P, \sqcap, \sqcup)$ besitzt ein Eins-Element $\mathbf{1}$; existieren $\top$ und $\mathbf{1}$ gilt:

$$\left(\bigsqcap \emptyset = \right) \top = \mathbf{1}$$

Zur Angemessenheit d. beiden Verbandssichten

In der Mathematik ist häufig die

- algebraische Verbandssicht geeigneter als sie konform zu anderen algebraischen Strukturen ist ('eine Menge mit bestimmten Gesetzen genügenden Verknüpfungsvorschriften') wie z.B. Gruppen, Ringen, Körpern, Vektorräumen, Kategorien, usw., die in der Mathematik untersucht und behandelt werden.

In der Informatik ist häufig die

- ordnungstheoretische Verbandssicht geeigneter, da sich die Ordnungsrelation häufig als '· trägt mehr/weniger Information als ·', '· ist mehr/weniger definiert als ·,' '· ist stärker/schwächer als ·', usw. interpretieren und verstehen lässt, was oft sehr natürlich zu Problemen passt, die in der Informatik untersucht und behandelt werden.

Übungsaufgabe A.4.6.14

Betrachte den Verband $(\mathbb{N}_0, \sqsubseteq)$ mit $\sqsubseteq =_{df} |$, wobei $|$ die Teilbarkeitsrelation auf den natürlichen Zahlen $\mathbb{N}_0$ bezeichne, d.h. die Relation '$\cdot$ teilt $\cdot$' (ohne Rest), z.B. $5 \,|\, 35$.

Definiere $(\mathbb{N}_0, \wedge, \vee)$, d.h. gib das algebraisch definierte Gegenstück zu $(\mathbb{N}_0, \sqsubseteq)$ an. Definiere dazu die Schnitt- und Vereinigungsoperation auf $\mathbb{N}_0 \times \mathbb{N}_0$:

1. $\wedge : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$
2. $\vee : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$

Welches ist das

1. Null-Element $\mathbf{0}$
2. Eins-Element $\mathbf{1}$

von $(\mathbb{N}_0, \wedge, \vee)$?

A.5

Fixpunkttheoreme

A.5.1

Fixpunkte, Türme

Fixpunkte von Funktionen

Definition A.5.1.1 (Fixpunkt)

Sei M eine Menge, $f \in [M \to M]$ eine Funktion auf M und $m \in M$ ein Element von M. Wir legen fest:

m heißt Fixpunkt von f gdw $f(m) = m$.

Kleinste, größte Fixpunkte in part. Ordnungen

Definition A.5.1.2 (Kleinster, größter Fixpunkt)

Sei $(P, \sqsubseteq)$ eine partielle Ordnung, $f \in [P \to P]$ eine Funktion auf P und p ein Fixpunkt von f, d.h. $f(p) = p$. Wir legen fest:

p heißt

1. kleinster Fixpunkt von f, bezeichnet mit μf,
   gdw $\forall\, q \in P.\ f(q) = q \Rightarrow p \sqsubseteq q$
2. größter Fixpunkt von f, bezeichnet mit νf,
   gdw $\forall\, q \in P.\ f(q) = q \Rightarrow q \sqsubseteq p$

Türme in kettenvollständigen part. Ordnungen

Definition A.5.1.3 (f-Turm in C)

Sei $(C, \sqsubseteq)$ eine KVPO, $f \in [C \to C]$ eine Funktion auf C und $T \subseteq C$ eine Teilmenge von C. Wir legen fest:

T heißt f-Turm in C gdw

1. $\bot \in T$.
2. Wenn $t \in T$, dann auch $f(t) \in T$.
3. Wenn $T' \subseteq T$ Kette in C, dann $\bigsqcup T' \in T$.

Kleinste Türme in kettenv. part. Ordnungen

Lemma A.5.1.4 (Kleinster f-Turm in C)

Der Schnitt

$$S =_{df} \bigcap \{T \mid T \text{ } f\text{-tower in } C\}$$

aller f-Türme in C ist der kleinste f-Turm in C, d.h.

1. S ist ein f-Turm in C.
2. $\forall\, T$ f-Turm in C. $S \subseteq T$.

Lemma A.5.1.5 (Kleinster f-Türme und Ketten)

Der kleinste f-Turm in C ist eine Kette in C, wenn f expandierend ist.

A.5.2

Fixpunkttheoreme für vollständige partielle Ordnungen

Fixpunkte expandierender/monotoner Funkt.

Fixpunkttheorem A.5.2.1 (Expandierende Funkt.)

Sei $(C, \sqsubseteq)$ eine KVPO und $f \in [C \stackrel{exp}{\to} C]$ eine expandierende Funktion auf C. Dann gilt:

Das Supremum des kleinsten f-Turms in C ist ein Fixpunkt von f.

Fixpunkttheorem A.5.2.2 (Monotone Funktionen)

Sei $(C, \sqsubseteq)$ eine KVPO und $f \in [C \stackrel{mon}{\to} C]$ eine monotone Funktion auf C. Dann gilt:

f hat einen eindeutig bestimmten kleinsten Fixpunkt μf, der durch das Supremum des kleinsten f-Turms in C gegeben ist.

Beachte

- Theorem A.5.2.1 und Theorem A.5.2.2 sichern für expandierende Funktionen die Existenz eines Fixpunkts und für monotone Funktionen die Existenz eines eindeutig bestimmten kleinsten Fixpunkts zu, sie liefern jedoch kein konstruktives Verfahren zur Berechnung oder zur näherungsweisen Berechnung dieser Fixpunkte.
- Das ist anders für Theorem A.5.2.3, das für stetige Funktionen ein (Näherungs-) Verfahren zur Berechnung des eindeutig bestimmten kleinsten Fixpunkts liefert. Daraus erklärt sich die größere Bedeutung stetiger gegenüber expandierender und monotoner Funktionen in der Praxis, und dass man, wo immer möglich, stetige Funktionen wählt.

Kleinste Fixpunkte stetiger Funktionen

Fixpunkttheorem A.5.2.3 (Knaster, Tarski, Kleene)

Sei $(C, \sqsubseteq)$ eine KVPO und $f \in [C \stackrel{stet}{\to} C]$ eine stetige Funktion auf C. Dann gilt:

f hat einen eindeutig bestimmten kleinsten Fixpunkt $\mu f \in C$, der durch das Supremum der (sog.) Kleene-Kette $\{\bot, f(\bot), f^2(\bot), f^3(\bot), \ldots\}$ gegeben ist, d.h.:

$$\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) = \bigsqcup \{\bot, f(\bot), f^2(\bot), \ldots\}$$

Erinnerung: $f^0 =_{df} Id_C$; $f^i =_{df} f \circ f^{i-1}$, $i > 0$.

Beweis von Fixpunkttheorem A.5.2.3 (1)

Wir müssen zeigen:

$$\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) = \bigsqcup \{ f^i(\bot) \mid i \geq 0 \}$$

1. existiert,
2. ist ein Fixpunkt von f,
3. ist der kleinste Fixpunkt von f.

Beweis von Fixpunkttheorem A.5.2.3 (2)

1. Existenz
   - Nach Definition von $\bot$ als kleinstem Element von C und von f^0 als Identität auf C erhalten wir: $\bot = f^0(\bot) \sqsubseteq f^1(\bot) = f(\bot)$.
   - Da f stetig und daher auch monoton ist, erhalten wir mit mithilfe vollständiger Induktion: $\forall\, i, j \in \mathbb{N}_0.\ i < j \Rightarrow f^i(\bot) \sqsubseteq f^{i+1}(\bot) \sqsubseteq f^j(\bot)$.
   - Somit ist die Menge $\{f^i(\bot) \mid i \geq 0\}$ eine (möglicherweise unendliche) Kette in C.
   - Da $(C, \sqsubseteq)$ eine KVPO ist und $\{f^i(\bot) \mid i \geq 0\}$ eine Kette in C ist, impliziert dies nach Definition einer KVPO, dass die kleinste obere Schranke der Kette $\{f^i(\bot) \mid i \geq 0\}$

$$\bigsqcup \{f^i(\bot) \mid i \geq 0\} = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \text{ existiert.}$$

Beweis von Fixpunkttheorem A.5.2.3 (3)

2. Fixpunkteigenschaft

$$
\begin{aligned}
& & & f(\bigsqcup_{i\in\mathbb{N}_0} f^i(\bot)) \\
& (f \text{ stetig}) & = & \bigsqcup_{i\in\mathbb{N}_0} f(f^i(\bot)) \\
& & = & \bigsqcup_{i\in\mathbb{N}_1} f^i(\bot) \\
(C' =_{df} \{f^i \bot \mid i \geq 1\} \text{ ist eine Kette} & \Rightarrow & & \\
\bigsqcup C' \text{ existiert } = \bot \sqcup \bigsqcup C') & = & \bot \sqcup \bigsqcup_{i\in\mathbb{N}_1} f^i(\bot) & \\
(f^0(\bot) =_{df} \bot) & = & \bigsqcup_{i\in\mathbb{N}_0} f^i(\bot) &
\end{aligned}
$$

Beweis von Fixpunkttheorem A.5.2.3 (4)

3. Kleinste Fixpunkteigenschaft

- Sei c ein beliebiger Fixpunkt von f. Dann gilt: $\bot \sqsubseteq c$.
- Da f stetig und daher auch monoton ist, erhalten wir mithilfe vollständiger Induktion:
  $\forall\, i \in \mathbb{N}_0.\ f^i(\bot) \sqsubseteq f^i(c)\ (= c)$.
- Da c ein Fixpunkt von f ist, impliziert das:
  $\forall\, i \in \mathbb{N}_0.\ f^i(\bot) \sqsubseteq c\ (= f^i(c))$.
- Somit ist c eine obere Schranke der Menge
  $\{f^i(\bot) \mid i \in \mathbb{N}_0\}$.
- Da $\{f^i(\bot) \mid i \in \mathbb{N}_0\}$ eine Kette ist und $\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)$ nach Definition die kleinste obere Schranke dieser Kette ist, erhalten wir die gewünschte noch fehlende Inklusionsbeziehung:

$$\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \sqsubseteq c.$$

□

Kleinste bedingte Fixpunkte

Sei $(C, \sqsubseteq)$ eine KVPO, $f \in [C \to C]$ eine Funktion auf C und $d, c_d \in C$ zwei Elemente von C.

Definition A.5.2.4 (Kleinster bedingter Fixpunkt)

c_d heißt kleinster bedingter Fixpunkt von f bezüglich d (engl. least conditional fixed point) gdw c_d ist der kleinste Fixpunkt von C mit $d \sqsubseteq c_d$, d.h.:

$$\forall x \in C.\ f(x) = x \ \wedge \ d \sqsubseteq x \Rightarrow c_d \sqsubseteq x$$

Kleinste bedingte Fixpunkte stetiger Funkt.

Theorem A.5.2.5 (Bedingtes Fixpunkttheorem)

Sei $(C, \sqsubseteq)$ eine KVPO, $d \in C$ und $f \in [C \overset{stet}{\to} C]$ eine stetige Funktion auf C, die für d expandierend ist, d.h. $d \sqsubseteq f(d)$. Dann gilt:

f hat einen kleinsten bedingten Fixpunkt $\mu f_d \in C$, der durch das Supremum der (verallgemeinerten) Kleene-Kette $\{d, f(d), f^2(d), \ldots\}$ gegeben ist, d.h.:

$$\mu f_d = \bigsqcup_{i \in \mathbb{N}_0} f^i(d) = \bigsqcup \{d, f(d), f^2(d), \ldots\}$$

Endliche Fixpunkte

Sei $(C, \sqsubseteq)$ eine KVO, $d \in C$ und $f \in [C \overset{mon}{\to} C]$ eine monotone Funktion auf C.

Theorem A.5.2.6 (Endliches Fixpunkttheorem)

Wenn zwei aufeinanderfolgende Elemente der Kleene-Kette von f gleich sind, d.h., gibt es ein $i \in \mathbb{N}$ mit $f^i(\bot) = f^{i+1}(\bot)$, so gilt: $\mu f = f^i(\bot)$.

Theorem A.5.2.7 (Endliches bedingtes Fixpunktth.)

Ist f expandierend für d, d.h. $d \sqsubseteq f(d)$, und sind zwei aufeinanderfolgende Elemente der (verallgemeinerten) Kleene-Kette von f bezüglich d gleich, d.h., gibt es ein $i \in \mathbb{N}$ mit $f^i(d) = f^{i+1}(d)$, so gilt: $\mu f_d = f^i(d)$.

Beachte: Theorem A.5.2.6 und A.5.2.7 setzen keine Stetigkeit von f voraus. Monotonie (und Expansion) von f reichen.

Hin zur Existenz endlicher Fixpunkte

Sei $(P, \sqsubseteq)$ eine partielle Ordnung und $p, r \in P$.

Definition A.5.2.8 (Kettenendliche part. Ordnung)

$(P, \sqsubseteq)$ heißt kettenendlich (engl. chain-finite) gdw P enthält keine unendlichen Ketten.

Definition A.5.2.9 (Endliches Element)

p heißt

1. endlich (engl. finite) gdw die Menge $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$ enthält keine unendliche Kette.
2. endlich relativ zu r gdw die Menge $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$ enthält keine unendliche Kette.

Existenz endlicher Fixpunkte

...es gibt zahlreiche hinreichende Bedingungen für die Existenz kleinster bedingter Fixpunkte einer Funktion f, die in der Praxis oft erfüllt sind (s. Nielson/Nielson 1992), z.B.:

- der Definitions- und Wertebereich von f sind endlich oder kettenendlich,
- der kleinste Fixpunkt von f ist endlich,
- f ist von der Form $f(c) = c \sqcup g(c)$, wobei g eine monotone Funktion auf einem kettenendlichen (Daten-) Bereich ist.

Fixpunkttheoreme, Verbände und GVPOs

Beachte: Vollständige Verbände (s. Lemma A.4.1.13) und GVPOs mit einem kleinsten Element (s. Lemma A.3.1.5) sind auch KVPOs.

Daraus können wir schließen:

Korollar A.5.2.10 (Fixpunkte, Verbände, GVPOs)

Die Fixpunkttheoreme aus Kapitel A.5.2 gelten auch für Funktionen auf vollständigen Verbänden und GVPOs mit einem kleinsten Element.

A.5.3

Fixpunkttheoreme für Verbände

Fixpunkte monotoner Funktionen

Fixpunkttheorem A.5.3.1 (Knaster, Tarski)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $f \in [P \overset{mon}{\to} P]$ eine monotone Funktion auf P. Dann gilt:

1. f hat einen eindeutig bestimmten kleinsten Fixpunkt $\mu f \in P$, der gegeben ist durch: $\mu f = \bigsqcap \{p \in P \mid f(p) \sqsubseteq p\}$.
2. f hat einen eindeutig bestimmten größten Fixpunkt $\nu f \in P$, der gegeben ist durch $\nu f = \bigsqcup \{p \in P \mid p \sqsubseteq f(p)\}$.

Charakterisierungstheorem A.5.3.2 (Davis)

Sei $(P, \sqsubseteq)$ ein Verband. Dann gilt:

$(P, \sqsubseteq)$ ist vollständig gdw jedes $f \in [P \overset{mon}{\to} P]$ hat einen Fixpunkt.

Der Fixpunktverband monotoner Funktionen

Theorem A.5.3.3 (Fixpunktverband)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband, $f \in [P \overset{mon}{\to} P]$ eine monotone Funktion auf P und $Fix(f) =_{df} \{p \in P \mid f(p) = p\}$ die Menge aller Fixpunkte von f. Dann gilt:

Jede Teilmenge $F \subseteq Fix(f)$ hat ein Supremum und ein Infimum in $Fix(f)$, d.h. $(Fix(f), \sqsubseteq_{\mid Fix(f)})$ ist ein vollständiger Verband.

Theorem A.5.3.4 (Fixpunktordnung)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $f \in [P \overset{mon}{\to} P]$ eine monotone Funktion auf P. Dann gilt:

$$\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \sqsubseteq \mu f \sqsubseteq \nu f \sqsubseteq \bigsqcap_{i \in \mathbb{N}_0} f^i(\top)$$

Fixpunkte additiver/distributiver Funktionen

Für additive und distributive Funktionen werden die linke und die rechte Ungleichheit in Theorem A.5.3.4 zu Gleichheiten:

Fixpunkttheorem A.5.3.5 (Knaster, Tarski, Kleene)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $f \in [P \to P]$ eine Funktion auf P. Dann gilt: f hat einen eindeutig bestimmten

1. kleinsten Fixpunkt $\mu f \in P$ gegeben durch $\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)$, wenn f additiv ist, d.h. $f \in [P \stackrel{add}{\to} P]$.
2. größten Fixpunkt $\nu f \in P$ gegeben durch $\nu f = \bigsqcap_{i \in \mathbb{N}_0} f^i(\top)$, wenn f distributiv ist, d.h. $f \in [P \stackrel{dis}{\to} P]$.

Erinnerung: $f^0 =_{df} Id_C$; $f^i =_{df} f \circ f^{i-1}$, $i > 0$.

A.6

Fixpunktinduktion

Zulässige Prädikate

Fixpunktinduktion erlaubt Eigenschaften über Fixpunkte stetiger Funktionen zu beweisen. Wesentlich dafür ist der Begriff zulässiger Prädikate:

Definition A.6.1 (Zulässiges Prädikat)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $\phi : P \to \mathbb{B}$ ein Prädikat auf P:

ϕ heißt zulässig (oder $\sqcup$-zulässig) (engl. admissible, $\sqcup$-admissible) gdw für jede Kette $C \subseteq P$ gilt:

$$(\forall\, c \in C.\ \phi(c)) \Rightarrow \phi(\bigsqcup C)$$

Lemma A.6.2

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $\phi : P \to \mathbb{B}$ ein zulässiges Prädikat auf P. Dann gilt: $\phi(\bot) = \mathbf{wahr}$.

Beweis. Aus der Zulässigkeit von ϕ folgt $\phi(\bigsqcup \emptyset) = \mathbf{wahr}$. Zusätzlich gilt $\bot = \bigsqcup \emptyset$, was den Beweis vervollständigt.

Hinreichende Bedingungen für Zulässigkeit

Theorem A.6.3 (Zulässigkeitsbedingung 1)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $\phi : P \to \mathbb{B}$ ein Prädikat auf P. Dann gilt:

ϕ ist zulässig, wenn es einen vollständigen Verband $(Q, \sqsubseteq_Q)$ und zwei additive Funktionen $f, g \in [P \overset{add}{\to} Q]$ gibt, so dass gilt:

$$\forall\, p \in P.\ \phi(p) \Longleftrightarrow f(p) \sqsubseteq_Q g(p)$$

Theorem A.6.4 (Zulässigkeitsbedingung 2)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband und $\phi, \psi : P \to \mathbb{B}$ zwei zulässige Prädikate auf P. Dann gilt:

Die Konjunktion von ϕ und ψ, das Prädikat $\phi \ \wedge \ \psi$ definiert durch

$$\forall\, p \in P.\ (\phi \ \wedge \ \psi)(p) =_{df} \phi(p) \wedge \psi(p)$$

ist zulässig.

Fixpunktinduktion auf vollständigen Verbänden

Theorem A.6.5 (Fixpunktinduktion auf vollst. Verb.)

Sei $(P, \sqsubseteq)$ ein vollständiger Verband, $f \in [P \overset{add}{\to} P]$ eine additive Funktion auf P und $\phi : P \to \mathbb{B}$ ein zulässiges Prädikat auf P. Dann gilt:

Die Gültigkeit von

- $\forall\, p \in P.\ \phi(p) \Rightarrow \phi(f(p))$ (Induktionsschritt)

impliziert die Gültigkeit von $\phi(\mu f)$.

Beachte: Der Induktionsanfang, d.h. die Gültigkeit von $\phi(\bot)$, folgt aus der Zulässigkeit von ϕ (vgl. Lemma A.6.2) und ist daher bereits mit dem Beweis der Zulässigkeit von ϕ bewiesen.

Fixpunktinduktion auf KVPOs (= CCPOs)

Der Begriff der Zulässigkeit von Prädikaten überträgt sich in natürlicher Weise von vollständigen Verbänden auf kettenvollständige partielle Ordnungen (KVPOs (= CCPOs)).

Theorem A.6.6 (Fixpunktinduktion auf KVPOs)

Sei $(C, \sqsubseteq)$ eine KVPO, $f \in [C \stackrel{mon}{\to} C]$ eine monotone Funktion auf C und $\phi : C \to \mathbb{B}$ ein zulässiges Prädikat auf C. Dann gilt:

Die Gültigkeit von

- $\forall\, c \in C.\ \phi(c) \Rightarrow \phi(f(c))$ (Induktionsschritt)

impliziert die Gültigkeit von $\phi(\mu f)$.

Beachte: Theorem A.6.6 gilt (natürlich auch) für einen vollständigen Verband $(P, \sqsubseteq)$ anstelle einer KVPO $(C, \sqsubseteq)$.

A.7

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Anhang A (1)


André Arnold, Irène Guessarian. *Mathematics for Computer Science*. Prentice Hall, 1996.


Roland Backhouse, Roy Crole, Jeremy R. Gibbons (Hrsg.). *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*. International Summer School and Workshop, Oxford, UK, April 10-14, 2000, Revised Lectures, Springer-V., LNCS 2297, 2002. (Chapter 1, Ordered Sets and Complete Lattices by Hilary A. Priestley; Chapter 2, Algebras and Coalgebras by Peter Aczel; Chapter 4, Calculating Functional Programs by Jeremy Gibbons)


Rudolf Berghammer. *Ordnungen, Verbände und Relationen mit Anwendungen*. Vieweg+Teubner, 2008.

Vertiefende und weiterführende Leseempfehlungen für Anhang A (2)

Rudolf Berghammer. *Ordnungen, Verbände und Relationen mit Anwendungen*. Springer-V., 2012. (Kapitel 1, Ordnungen und Verbände; Kapitel 2.4, Vollständige Verbände; Kapitel 3, Fixpunkttheorie mit Anwendungen; Kapitel 4, Vervollständigung und Darstellung mittels Vervollständigung; Kapitel 5, Wohlgeordnete Mengen und das Auswahlaxiom)

Rudolf Berghammer. *Ordnungen und Verbände: Grundlagen, Vorgehensweisen und Anwendungen*. Springer-V., 2013. (Kapitel 2, Verbände und Ordnungen; Kapitel 3.4, Vollständige Verbände; Kapitel 4, Fixpunkttheorie mit Anwendungen; Kapitel 5, Vervollständigung und Darstellung mittels Vervollständigung; Kapitel 6, Wohlgeordnete Mengen und das Auswahlaxiom)

Vertiefende und weiterführende Leseempfehlungen für Anhang A (3)

- Garret Birkhoff. *Applications of Lattice Algebra*. Mathematical Proceedings of the Cambridge Philosophical Society 30(2):115-122, 1934.
- Garret Birkhoff. *Lattice Theory*. American Mathematical Society, 3rd edition, 1967.
- Peter Crawley, Robert P. Dilworth. *Algebraic Theory of Lattices*. Prentice Hall, 1973.
- Brian A. Davey, Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge University Press, 2nd edition, 2002. (Chapter 1, Ordered Sets; Chapter 2, Lattices and Complete Lattices; Chapter 8, CPOs and Fixpoint Theorems)

Vertiefende und weiterführende Leseempfehlungen für Anhang A (4)


Anne C. Davis. *A Characterization of Complete Lattices*. Pacific Journal of Mathematics 5(2):311-319, 1955.


Marcel Erné. *Einführung in die Ordnungstheorie*. Bibliographisches Institut, 2. Auflage, 1982.


Helmuth Gericke. *Theorie der Verbände*. Bibliographisches Institut, 2. Auflage, 1967.


George Grätzer. *General Lattice Theory*. Birkhäuser, 2nd edition, 1998. (Chapter 1, First Concepts; Chapter 2, Distributive Lattices; Chapter 3, Congruences and Ideals; Chapter 5, Varieties of Lattices)


George Grätzer. *Lattice Theory: Foundation*. Birkhäuser, 2011.

Vertiefende und weiterführende Leseempfehlungen für Anhang A (5)


George Grätzer, Friedrich Wehrung (Hrsg.). *Lattice Theory: Special Topics and Applications, Vol. I*. Birkhäuser, 2014.


George Grätzer, Friedrich Wehrung (Hrsg.). *Lattice Theory: Special Topics and Applications, Vol. II*. Birkhäuser, 2016.


Paul R. Halmos. *Naive Set Theory*. Springer-V., Reprint, 2001. (Chapter 6, Ordered Pairs; Chapter 7, Relations; Chapter 8, Functions)


Hans Hermes. *Einführung in die Verbandstheorie*. Springer-V., 2. Auflage, 1967.

Vertiefende und weiterführende Leseempfehlungen für Anhang A (6)

Richard Johnsonbaugh. *Discrete Mathematics*. Pearson, 7th edition, 2009. (Chapter 3, Functions, Sequences, and Relations)

Stephen C. Kleene. *Introduction to Metamathematics*. North Holland, 1952. (Reprint, North Holland, 1980)

Seymour Lipschutz. *Set Theory and Related Topics*. McGraw Hill Schaum's Outline Series, 2nd edition, 1998. (Chapter 4, Functions; Chapter 6, Relations)

David Makinson. *Sets, Logic and Maths for Computing*. Springer-V., 2008. (Chapter 1, Collecting Things Together: Sets; Chapter 2, Comparing Things: Relations)

Vertiefende und weiterführende Leseempfehlungen für Anhang A (7)


George Markowsky. *Chain-complete Posets and Directed Sets with Applications*. Algebra Universalis 6(1):53-68, 1976.


Flemming Nielson, Hanne Riis Nielson. *Finiteness Conditions for Fixed Point Iteration*. In Proceedings of the 7th ACM Conference on LISP and Functional Programming (LFP'92), 96-108, 1992.


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 4, Denotational Semantics)

Vertiefende und weiterführende Leseempfehlungen für Anhang A (8)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 5, Denotational Semantics)


Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. Springer-V., 2nd edition, 2005. (Appendix A, Partially Ordered Sets)


Steven Roman. *Lattices and Ordered Sets*. Springer-V., 2008.


Bernhard Steffen, Oliver Rüthing, Malte Isberner. *Grundlagen der höheren Informatik. Induktives Vorgehen*. Springer-V., 2014. (Kapitel 5.1, Ordnungsrelationen; Kapitel 5.2, Ordnungen und Teilstrukturen)

Vertiefende und weiterführende Leseempfehlungen für Anhang A (9)

Bernhard Steffen, Oliver Rüthing, Michael Huth. *Mathematical Foundations of Advanced Informatics: Inductive Approaches*. Springer-V., 2018. (Chapter 5.1, Order Relations; Chapter 5.2, Orders and Substructures)

Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5(2):285-309, 1955.

Franklyn Turbak, David Gifford with Mark A. Sheldon. *Design Concepts in Programming Languages*. MIT Press, 2008. (Chapter 5, Fixed Points; Chapter 105, Software Testing; Chapter 106, Formal Methods; Chapter 107, Verification and Validation)

Vertiefende und weiterführende Leseempfehlungen für Anhang A (10)

Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993. (Chapter 1, Basic set theory; Chapter 8, Introduction to domain theory; Chapter 9, Recursion equations; Chapter 10, Recursion techniques; Chapter 10.2, Fixed-point induction)

Anhang B

Pragmatik: Flussgraphvarianten

B.1

Motivation

B.1.1

Flussgraphvarianten

Anweisungsrepräsentation in Flussgraphen

...werden Programme als Flussgraphen dargestellt, können Anweisungen (Zuweisungen, Tests)

- Knoten
- Kanten

zugeordnet werden als

- einzelne Anweisungen
- Basisblöcke (sequentielle Anweisungsfolgen max. Länge)

Flussgraphvarianten

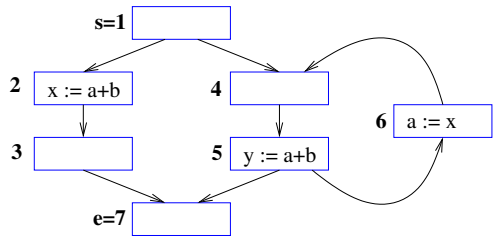
Diese Wahlmöglichkeiten führen auf vier Flussgraphvarianten:

- Knotenbenannte Flussgraphen
  (im Stil von Kripke-Strukturen)
  1) Einzelanweisungsgraphen (EA-Graphen)
  2) Basisblockgraphen (BB-Graphen)
- Kantenbenannte Flussgraphen
  (im Stil von Transitionssystemen)
  3) Einzelanweisungsgraphen (EA-Graphen)
  4) Basisblockgraphen (BB-Graphen)

Knotenbenannte Flussgraphvarianten

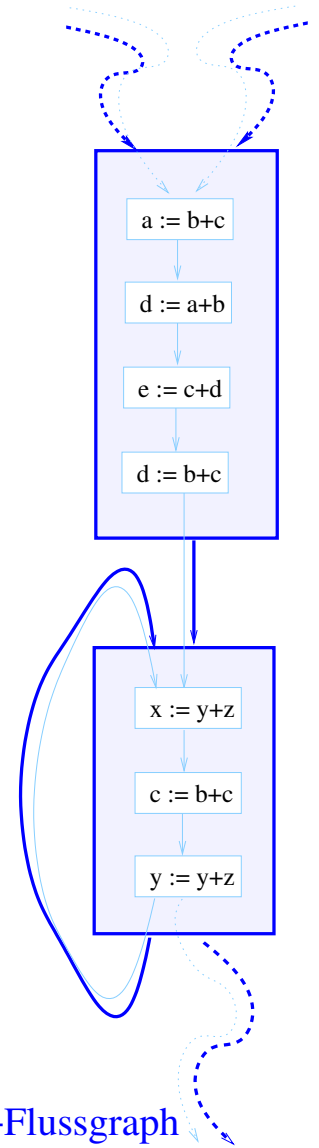
a) Einzelanweisungs- vs. b) Basisblockflussgrahen:

a)



Knotenbenannter EA–Flussgraph

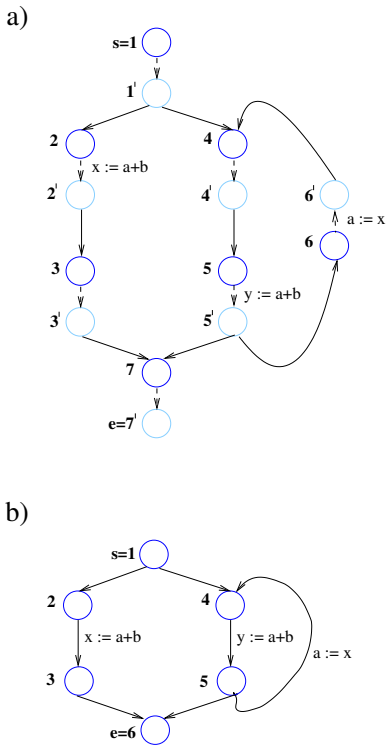
b)



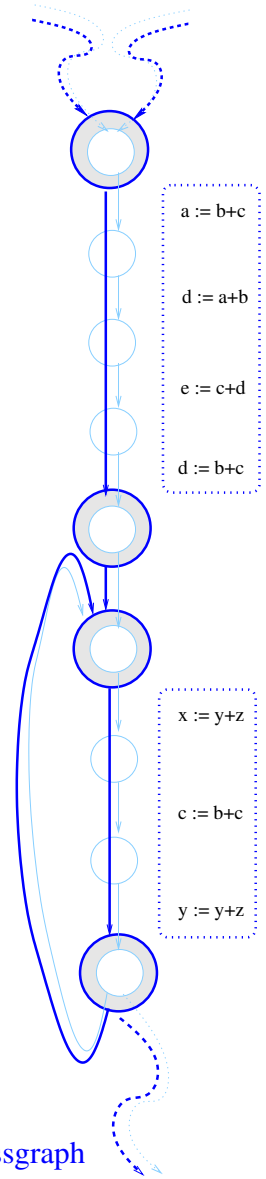
Knotenbenannter BB–Flussgraph

Kantenbenannte Flussgraphvarianten

a), b) Einzelanweisungs- vs. c) Basisblockflussgrahen:



Kantenbenannte EA–Flussgraphen



Kantenbenannter BB–Flussgraph

Welche Flussgraphvariante sollten wir wählen?

Konzeptuell

- besteht kein wesentlicher Unterschied zwischen den verschiedenen Flussgraphvarianten, was die Wahl einer bestimmten Variante zu einer Geschmacksfrage macht.

Pragmatisch

- unterscheiden sich die Flussgraphvarianten jedoch in der Einfachheit und damit ihrer Angemessenheit zur Spezifikation und Implementierung von Programmanalysen und Optimierungen.

Das werden wir in der Folge genauer herausarbeiten.

B.1.2

Flussgraphvarianten: Welche sollten wir wählen?

Basisblock- vs. Einzelanweisungsgraphen

...wir untersuchen und vergleichen unter pragmatischen Gesichtspunkten die Zweckmäßigkeit verschiedener Flussgraphvarianten als Programmrepräsentation für Programmanalyse.

Dazu betrachten wir knoten- und kantenbenannte Flussgraphen, die mit Basisblöcken bzw. Einzelanweisungen benannt sind, und untersuchen ihre jeweiligen

- Vor- und Nachteile für die Programmanalyse

...für eine Antwort auf die Frage:

- Knoten- oder kantenbenannte Basisblock- oder Einzelanweisungsgraphen: (Nur) eine Geschmacksfrage?

En passant werden wir dabei weitere praktisch relevante

- DFA-Probleme und -Analysen

kennenlernen.

Von Basisblockgraphen erhoffte Vorteile

...allgemein wird Basisblockgraphen ‘folkloristisch’ (engl. folk knowledge) vor allem folgender Anwendungsvorteil zugeschrieben :

Bessere Skalierungseigenschaften und Performanzvorteile, da

- weniger Knoten in die (potentiell) berechnungsaufwändige iterative Fixpunktberechnung involviert sind.
- größere Programme im Hauptspeicher gehalten werden können.

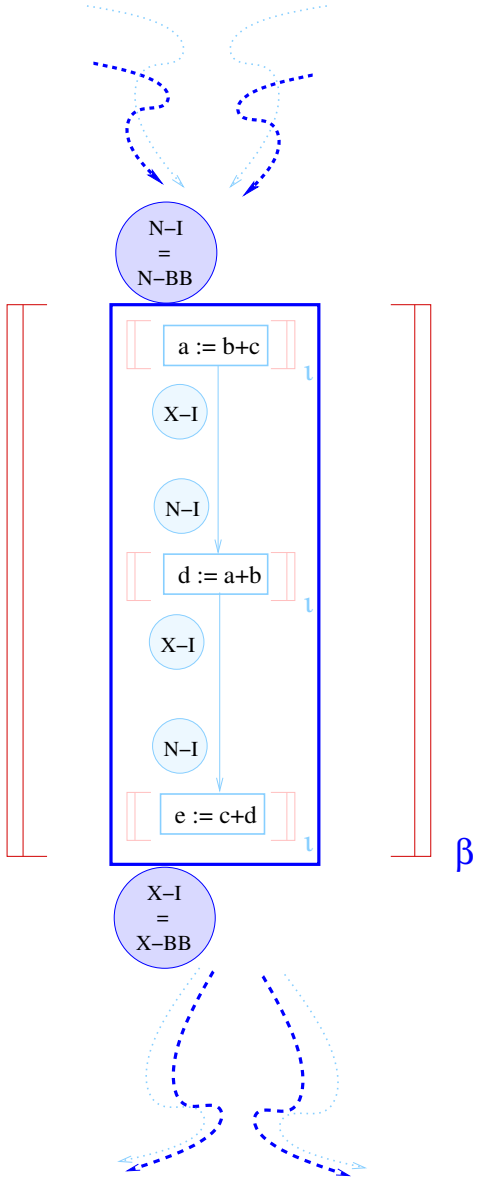
Mit Basisblockgraphen verbundene Nachteile

...sind definitiv auch gegeben, besonders folgende:

- Höhere konzeptuelle Komplexität: Basisblöcke führen zu einer unerwünschten Hierarchisierung, die sowohl theoretische Überlegungen wie praktische Implementierungen erschwert.
- Notwendigkeit von Prä- und Postprozessen: Sind i.a. erforderlich, um hierarchie-induzierte Zusatzprobleme zu behandeln (z.B. für Elimination toter Anweisungen (engl. dead code elimination), Konstantenanalyse (engl. constant propagation and folding),...); oder 'trickhafte', problemspezifische Formulierungen nötig machen, sie zu vermeiden (z.B. für partielle Redundanzelimination (engl. partial redundancy elimination)).
- Eingeschränkte Allgemeinheit: Bestimmte praktisch relevante Analysen und Optimierungen sind nur schwer oder gar nicht auf der Ebene von Basisblöcken auszudrücken (z.B. Geisteranweisungsanalyse und -elimination (engl. faint variable elimination)).

Kernproblem

...Basisblöcke führen zu einer hierarchischen Graphstruktur:



In der Folge

...Gegenüberstellung von Vor- und Nachteilen von

- Basisblock- und Einzelanweisungsgraphen

anhand von Beispielen von uns bereits betrachteter:

- Verfügbare Ausdrücke (engl. available expressions)
- Einfache Konstanten (engl. simple constants)

und neuer DFA-Probleme:

- Tote Anweisungen (engl. dead variables)
- Geisteranweisungen (engl. faint variables)

B.2

SUP- und *MaxFP*-Ansatz

B.2.1

Kantenbenannte Einzelanweisungsgraphen

SUP_{EAG}- und $MaxFP_{EAG}$-Ansatz

...für kantenbenannte Einzelanweisungsgraphen.

Die *SUP*-Lösung:

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ SUP_{(\llbracket\ \rrbracket_\alpha, c_s)}(n) =_{df} \bigsqcap \{\, \llbracket\, p\,\rrbracket_\alpha(c_s) \mid p \in \mathbf{P}_G[s,n] \,\}$$

Die *MaxFP*-Lösung:

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket_\alpha, c_s)}(n) =_{df} \nu\text{-}inf(n)$$

wobei ν-*inf* die größte Lösung des *MaxFP*-Gleichungssystems für Einzelanweisungsgraphen bezeichnet (α für Anweisung):

$$inf(n) = \begin{cases} c_s & \text{falls } n = s \\ \bigsqcap \{\, \llbracket\, (m,n)\,\rrbracket_\alpha(inf(m)) \mid m \in pred_G(n) \,\} & \text{sonst} \end{cases}$$

B.2.2

Knotenbenannte Basisblockgraphen

Bezeichnungen

...in der Folge bezeichnen:

- fett gesetzte Buchstaben Basisblockknoten: **m**, **n**,...
- normal gesetzte Buchstaben Einzelanweisungsknoten: m, n,...

Weiters bezeichnen:

- $[\![\ ]\!]_\beta$
- $[\![\ ]\!]_\alpha$

(lokale) abstrakte DFA-Funktionale für Basisblock- bzw. Einzelanweisungsknoten sowie

- *bb*, *start* und *end* drei Abbildungen, die angewendet auf einen Einzelanweisungsknoten *n* bzw. einen Basisblock **n** den Basisblock liefern, zu dem *n* gehört, bzw. den Start- und Endanweisungsknoten von **n**.

Hierarchischer SUP_{BBG}-Ansatz: Stufe I

...für knotenbenannte Basisblockgraphen.

Stufe I: Die *SUP*-Lösung auf Basisblockebene

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, \mathbf{n} \in \mathbf{N}.\ SUP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\ E\text{-}SUP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ A\text{-}SUP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n})\ )$$

mit

$$E\text{-}SUP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\, [\![\ p\ ]\!]_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s}, \mathbf{n}[\,\}$$

$$A\text{-}SUP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\, [\![\ p\ ]\!]_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s}, \mathbf{n}]\,\}$$

...wobei *E* und *A* für Basisblock-Eingang und -Ausgang stehen.

Hierarchischer SUP_{BBG}-Ansatz: Stufe II

Stufe II: Die *SUP*-Lösung auf Einzelanweisungsebene

...hineintreiben der DFA-Information in die Basisblöcke.

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ SUP_{(\![\ ]\!]_\alpha, c_s)}(n) =_{df} \\ (\ E\text{-}SUP_{(\![\ ]\!]_\alpha, c_s)}(n),\ A\text{-}SUP_{(\![\ ]\!]_\alpha, c_s)}(n)\ )$$

mit

$$E\text{-}SUP_{(\![\ ]\!]_\alpha, c_s)}(n) =_{df} \begin{cases} E\text{-}SUP_{(\![\ ]\!]_\beta, c_s)}(bb(n)) \\ \quad \text{falls } n = start(bb(n)) \\ [\![\ p\ ]\!]_\alpha(E\text{-}SUP_{(\![\ ]\!]_\beta, c_s)}(bb(n))) \\ \quad \text{sonst } (p \text{ Präfixpfad von } start(bb(n)) \\ \qquad \text{bis (ausschließlich) } n) \end{cases}$$

$$A\text{-}SUP_{(\![\ ]\!]_\alpha, c_s)}(n) =_{df} [\![\ p\ ]\!]_\alpha(E\text{-}SUP_{(\![\ ]\!]_\beta, c_s)}(bb(n))) \\ (p \text{ Präfixpfad von } start(bb(n)) \text{ bis (ein-} \\ \text{-schließlich) } n)$$

Hierarchischer $MaxFP_{BBG}$-Ansatz: Stufe I

...für knotenbenannte Basisblockgraphen:

Stufe I: Die *MaxFP*-Lösung auf Basisblockebene

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\, \forall\, \mathbf{n} \in \mathbf{N}.\ MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\ E\text{-}MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ A\text{-}MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n})\ )$$

mit

$$E\text{-}MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \nu\text{-}E\text{-}inf(\mathbf{n})$$
$$A\text{-}MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \nu\text{-}A\text{-}inf(\mathbf{n})$$

wobei ν-*E-inf* und ν-*A-inf* die größten Lösungen des *MaxFP*-Gleichungssystems für Basisblockknoten bezeichnen:

$$E\text{-}inf(\mathbf{n}) = \begin{cases} c_{\mathbf{s}} & \text{falls } \mathbf{n} = \mathbf{s} \\ \bigsqcap \{\, A\text{-}inf(\mathbf{m}) \mid \mathbf{m} \in pred_{\mathbf{G}}(\mathbf{n}) \,\} & \text{sonst} \end{cases}$$
$$A\text{-}inf(\mathbf{n}) = \llbracket\, \mathbf{n} \,\rrbracket_\beta(E\text{-}inf(\mathbf{n}))$$

Hierarchischer $MaxFP_{BBG}$-Ansatz: Stufe II

Stufe II: Die *MaxFP*-Lösung auf Einzelanweisungsebene

...hineintreiben der DFA-Information in die Basisblöcke.

$$\forall\, c_s \in \mathcal{C}\, \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket_\alpha, c_s)}(n) =_{df} \\ (\ E\text{-}MaxFP_{(\llbracket\ \rrbracket_\alpha, c_s)}(n),\ A\text{-}MaxFP_{(\llbracket\ \rrbracket_\alpha, c_s)}(n)\ )$$

mit

$$E\text{-}MaxFP_{(\llbracket\ \rrbracket_\alpha, c_s)}(n) =_{df} \nu\text{-}E\text{-}inf(n)$$
$$A\text{-}MaxFP_{(\llbracket\ \rrbracket_\alpha, c_s)}(n) =_{df} \nu\text{-}A\text{-}inf(n)$$

...wobei ν-*E-inf* und ν-*A-inf* die größten Lösungen des *MaxFP*-Gleichungssystems für Anweisungsknoten bezeichnen:

$$E\text{-}inf(n) = \begin{cases} \nu\text{-}E\text{-}inf(bb(n)) & \text{falls } n = start(bb(n)) \\ A\text{-}inf(m) & \text{sonst (wobei } m \text{ der eindeutig bestimmte Vorgänger von } n \text{ in } bb(n) \text{ ist)} \end{cases}$$

$$A\text{-}inf(n) = \llbracket\ n\ \rrbracket_\alpha(E\text{-}inf(n))$$

Kapitel B.3

Verfügbare Ausdrücke

B.3.1

Knotenbenannte Basisblockgraphen

Verfügbare Ausdrücke: Stufe I

...für knotenbenannte Basisblockgraphen.

Stufe I: Die Basisblockebene

Lokale Prädikate (assoziiert mit Basisblockknoten):

- BB-XComp$_\beta(t)$: t wird von einer Anweisung α in β berechnet und weder α noch eine auf α folgende Anweisung in β modifizieren einen Operanden von t.
- BB-Transp$_\beta(t)$: t ist transparent für β, d.h. keine Anweisung in β modifiziert einen Operanden von t.

Das Basisblock-Gleichungssystem von Stufe I:

$$\text{BB-N-Avail}_\beta = \begin{cases} \textbf{falsch} & \text{falls } \beta = \textbf{s} \\ \prod_{\hat{\beta} \in \mathit{pred}(\beta)} \text{BB-X-Avail}_{\hat{\beta}} & \text{sonst} \end{cases}$$

$$\text{BB-X-Avail}_\beta = \text{BB-N-Avail}_\beta \cdot \text{BB-Transp}_\beta + \text{BB-XComp}_\beta$$

Verfügbare Ausdrücke: Stufe II

Stufe II: Die Einzelanweisungsebene

Lokale Prädikate (assoziiert mit Einzelanweisungsknoten):

- $\text{Comp}_\alpha(t)$: α berechnet t.
- $\text{Transp}_\alpha(t)$: α modifiziert keinen Operanden von t.
- ν-BB-N-Avail, ν-BB-X-Avail: größte Lösungen des BB-Gleichungssystem von Stufe I.

Das Einzelanweisungs-Gleichungssystem von Stufe II:

$$\text{N-Avail}_\alpha = \begin{cases} \nu\text{-BB-N-Avail}_{bb(\alpha)} & \text{falls } \alpha = start(bb(\alpha)) \\ \text{X-Avail}_{pred(\alpha)} & \text{sonst (da gilt: } |pred(\alpha)| = 1) \end{cases}$$

$$\text{X-Avail}_\alpha = \begin{cases} \nu\text{-BB-X-Avail}_{bb(\alpha)} & \text{falls } \alpha = end(bb(\alpha)) \\ (\text{N-Avail}_\alpha + \text{Comp}_\alpha) \cdot \text{Transp}_\alpha & \text{sonst} \end{cases}$$

B.3.2

Knotenbenannte Einzelanweisungsgraphen

Verfügbare Ausdrücke

...für knotenbenannte Einzelanweisungsgraphen (einstufig, hierarchiefrei).

Lokale Prädikate (assoziiert mit Einzelanweisungsknoten):

- $\text{Comp}_\alpha(t)$: α berechnet t.
- $\text{Transp}_\alpha(t)$: α modifiziert keinen Operanden von t.

Gleichungssystem f. knotenbenannte Einzelanweisungsgraphen:

$$\text{N-Avail}_\iota = \begin{cases} \textbf{falsch} & \text{falls } \alpha = s \\ \prod_{\hat{\alpha} \in \mathit{pred}(\alpha)} \text{X-Avail}_{\hat{\alpha}} & \text{sonst} \end{cases}$$

$$\text{X-Avail}_\alpha = (\text{N-Avail}_\alpha + \text{Comp}_\alpha) \cdot \text{Transp}_\alpha$$

B.3.3

Kantenbenannte Einzelanweisungsgraphen

Verfügbare Ausdrücke

...für kantenbenannte Einzelanweisungsgraphen (einstufig, hierarchiefrei).

Lokale Prädikate (assoziiert mit Instruktionskanten):

- $\mathrm{Comp}_{\varepsilon}(t)$: Anweisung α von Kante ε berechnet t.
- $\mathrm{Transp}_{\varepsilon}(t)$: Anweisung α von Kante ε modifiziert keinen Operanden von t.

Gleichungssystem f. kantenbenannte Einzelanweisungsgraphen:

$$\mathrm{Avail}_n = \begin{cases} \textbf{falsch} & \text{falls } n = s \\ \prod_{m \in pred(n)} (\mathrm{Avail}_m + \mathrm{Comp}_{(m,n)}) \cdot \mathrm{Transp}_{(m,n)} & \text{sonst} \end{cases}$$

B.3.4

Zwischenfazit

Beobachtung

...kantenbenannte Einzelanweisungsgraphen sind konzeptuell und formulierungstechnisch am wenigsten aufwändig und deshalb am

- günstigsten.

...knotenbenannte Basisblockgraphen am aufwändigsten und deshalb am

- ungünstigsten.

In der Folge

...zwei weitere Beispiele dazu und zur Veranschaulichung des Einflusses von Flussgraphvarianten auf den konzeptuellen und technischen Aufwand der Formulierung von Programmanalysen:

- Konstantenanalyse (engl. constant propagation and folding)
- Geistervariablenanalyse (engl. faint variables analysis)

Dabei betrachten wir Analyseformulierungen für:

- knotenbenannte Basisblockgraphen
- kantenbenannte Einzelanweisungsgraphen

als die beiden antagonistischen Pole der Graphvarianten.

Kapitel B.4

Konstantenanalyse

Konstantenanalyse

... am Beispiel einfacher Konstanten (engl. simple constants).

Wir benötigen zwei Hilfsfunktionen für Anweisungen:

1. Rückwärtssubstitution
2. Zustandstransformation

sowie deren Ausdehnungen auf Anweisungssequenzen, speziell Pfadanweisungssequenzen.

B.4.1

Kantenbenannte Einzelanweisungsgraphen

Rückwärtssubstitution, Zustandstransformation

...für Anweisungen $\alpha \equiv (x := t')$.

Wir definieren für α:

1. Rückwärtssubstitution

$$\delta_\alpha : \mathbf{T} \to \mathbf{T}$$
$$\forall\, t \in \mathbf{T}.\ \delta_\alpha(t) =_{df} t[t'/x]$$

   wobei $t[t'/x]$ die simultane Ersetzung aller Vorkommen von x in t durch t' bezeichnet (syntaktische Substitution).

2. Zustandstransformation

$$\theta_\alpha : \Sigma \to \Sigma$$
$$\theta_\alpha(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

   wobei $\mathcal{E} : \mathbf{T} \to \Sigma \to \mathbb{Z}$ die Evaluation von Termen entsprechend ihrer Semantik leistet (z.B. $\mathcal{E} = [\![\ ]\!]_A$).

Der Zusammenhang von δ und θ

...ist beschrieben durch das Substitutionslemma B.4.1.1, wobei $\mathcal{A}$ die Menge aller Anweisungen bezeichne.

Lemma B.4.1.1 (Substitutionslemma)

$$\forall\, t \in \mathbf{T}\ \forall\, \sigma \in \Sigma\ \forall\, \alpha \in \mathcal{A}.\ \mathcal{E}(\delta_\alpha(t))(\sigma) = \mathcal{E}(t)(\theta_\alpha(\sigma))$$

Beweis induktiv über den Aufbau von t.

Einfache Konstanten auf Anweisungsgraphen

Bezeichnen:

- $\mathsf{eK}_n \in \Sigma =_{df} \{\sigma \mid \sigma : \mathbf{V} \to \mathbb{Z}^{\top}_{\bot}\}$
- $\sigma_s \in \Sigma \backslash \{\sigma_{\top}\}$ Anfangszustand (oder Anfangszusicherung)

...wobei eK von 'einfache Konstanten' abgeleitet ist.

Das eK-Gleichungssystem für Einzelanweisungsgraphen:

$$\mathsf{eK}_n = \begin{cases} \sigma_{\mathbf{s}} & \text{falls } n = s \\ \lambda\, v.\ \bigsqcap\{\, \mathcal{E}(\delta_{(m,n)}(v))(\mathsf{eK}_m) \mid m \in \mathit{pred}(n) \,\} & \text{sonst} \end{cases}$$

B.4.2

Knotenbenannte Basisblockgraphen

Rückwärtssubstitution, Zustandstransformation

...auf Anweisungssequenzen von Pfaden, speziell damit auch auf Anweisungssequenzen von Basisblöcken.

Ausdehnung von δ und θ zur:

1. Rückwärtssubstitution auf Pfadanweisungssequenzen

$$\Delta_p : \mathbf{T} \to \mathbf{T}$$

$$\Delta_p =_{df} \begin{cases} \delta_{n_q} & \text{falls } q = 1 \\ \Delta_{(n_1,\ldots,n_{q-1})} \circ \delta_{n_q} & \text{falls } q > 1 \end{cases}$$

2. Zustandstransformation auf Pfadanweisungssequenzen

$$\Theta_p : \Sigma \to \Sigma$$

$$\Theta_p =_{df} \begin{cases} \theta_{n_1} & \text{falls } q = 1 \\ \Theta_{(n_2,\ldots,n_q)} \circ \theta_{n_1} & \text{falls } q > 1 \end{cases}$$

Der Zusammenhang von Δ und Θ

...ist beschrieben durch das Verallgemeinerte Substitutionslemma B.4.2.1, wobei $\mathcal{B}$ die Menge aller Basisblöcke bezeichne.

Lemma B.4.2.1 (Verallg. Substitutionslemma)

$$\forall\, t \in \mathbf{T}\ \forall\, \sigma \in \Sigma\ \forall\, \beta \in \mathcal{B}.\ \mathcal{E}(\Delta_\beta(t))(\sigma) = \mathcal{E}(t)(\Theta_\beta(\sigma))$$

Beweis induktiv über die Länge von p.

Einfache Konstanten auf BB-Graphen: Stufe I

Stufe I: Die Basisblockebene

Bezeichnen:

- $\Sigma =_{df} \{\sigma \mid \sigma : \mathbf{V} \to \mathbb{Z}_{\perp}^{\top}\}$
- $\Delta_{\beta}(v) =_{df} \delta_{\alpha_1} \circ \ldots \circ \delta_{\alpha_q}(v)$, wobei $\beta \equiv \alpha_1; \ldots; \alpha_q$.
- BB-N-eK$_{\beta}$, BB-X-eK$_{\beta}$, N-eK$_{\alpha}$, X-eK$_{\alpha} \in \Sigma$
- $\sigma_s \in \Sigma$ Anfangszustand (oder: Anfangszusicherung)

Das BB-Gleichungssystem für einf. Konstanten von Stufe I:

$$\text{BB-N-eK}_{\beta} = \begin{cases} \sigma_s & \text{falls } \beta = \mathbf{s} \\ \bigsqcap\{\text{BB-X-eK}_{\hat{\beta}} \mid \hat{\beta} \in \mathit{pred}(\beta)\} & \text{sonst} \end{cases}$$

$$\text{BB-X-eK}_{\beta} = \lambda\, v.\, \mathcal{E}(\Delta_{\beta}(v))(\text{BB-N-eK}_{\beta})$$

Einfache Konstanten auf BB-Graphen: Stufe II

Stufe II: Die Anweisungsebene

Vorberechnete Resultate (von Stufe I):

- ν-BB-N-eK, ν-BB-X-eK: die größten Lösung des Gleichungssystems von Stufe I.

Das Anw.-Gleichungssystem für einf. Konstanten von Stufe II:

$$\text{N-eK}_\alpha = \begin{cases} \nu\text{-BB-N-eK}_{bb(\alpha)} & \text{falls } \alpha = start(bb(\alpha)) \\ \text{X-eK}_{pred(\alpha)} & \text{sonst (da gilt: } |pred(\alpha)| = 1) \end{cases}$$

$$\text{X-eK}_\alpha = \begin{cases} \nu\text{-BB-X-eK}_{bb(\alpha)} & \text{falls } \alpha = end(bb(\alpha)) \\ \lambda v.\, \mathcal{E}(\delta_\alpha(v))(\text{N-eK}_\alpha) & \text{sonst} \end{cases}$$

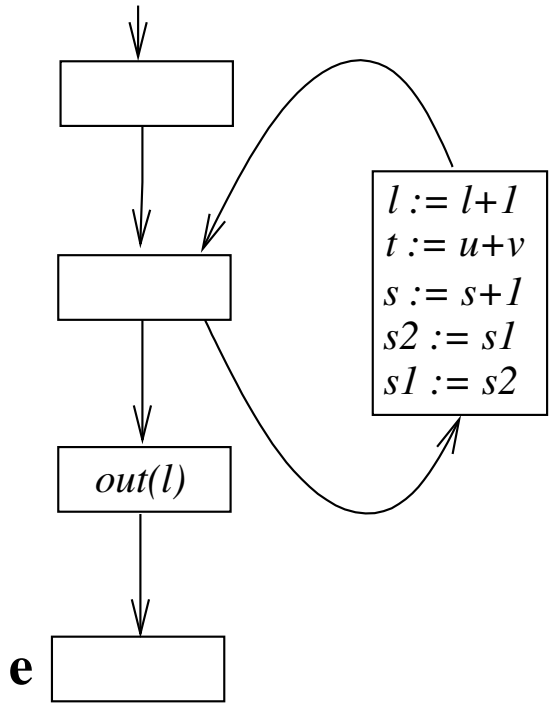
...wobei *bb*, *start* und *end* drei Abbildungen sind, die angewendet auf eine Anweisung α bzw. einen Basisblock β den Basisblock liefern, zu dem α gehört, bzw. den Start- oder Endanweisungsknoten von β.

B.5

Geistervariablenanalyse

Geistervariablen

...betrachte folgendes Programm:



Es gilt: Anweisung

- $l := l + 1$ ist lebendig.
- $t := u + v$ ist tot.
- $s := s + 1$ und $s1 := s2$; $s2 := s1$ sind lebendig, aber geisterhaft (engl. faint).

Geistervariablenanalyse (1)

...für kantenbenannte Einzelanweisungsgraphen.

Lokale Prädikate (assoziiert mit Einzelanweisungskanten):

- $\mathrm{LifeEnforcingUse}_{\varepsilon}^{v}$: Variable v kommt in der Anweisung α von Kante ε vor und wird von ihr 'zu leben gezwungen' (z.B. wenn α eine Ausgabeanweisung, Verzweigungsbedingung oder Schleifenabbruchbedingung ist).
- $\mathrm{Mod}_{\varepsilon}^{v}$: Anweisung α von Kante ε modifiziert Variable v.
- $\mathrm{AssUse}_{\varepsilon}^{v}$: Variable v kommt rechtsseitig in der Zuweisung α von Kante ε vor.
- $\mathit{LhsVar}_{\varepsilon}$: Bezeichnet die linksseitige Variable der Zuweisung α an Kante ε.

Geistervariablenanalyse (2)

Das GV-Gleichungssystem für Einzelanweisungsgraphen:

$$\begin{aligned} \mathrm{FAINT}_n^v = \prod_{m \in succ(n)} & \left(\overline{\mathrm{LifeEnforcingUse}_{(n,m)}^v} \; * \right. \\ & (\mathrm{FAINT}_m^v \; + \; \mathrm{Mod}_{(n,m)}^v) \; * \\ & \left. (\mathrm{FAINT}_m^{LhsVar_{(n,m)}} + \overline{\mathrm{AssUse}_{(n,m)}^v}) \right) \end{aligned}$$

Intuitiv: Eine Variable v ist geisterhaft am Knoten n, wenn v

- von keiner Anweisung einer in n eingehenden Kante zu leben gezwungen wird (1-tes Konjunktionsglied).
- am Knoten n bereits geisterhaft ist oder durch die Anweisung an einer eingehenden Kante modifiziert und dadurch geisterhaft wird (2-tes Konjunktionsglied).
- von keiner Anweisung auf einer eingehenden Kante benutzt wird oder höchstens der Wertzuweisung an eine andere geisterhafte Variable dient (3-tes Konjunktionsglied).

Geistervariablen

...sind ein Beispiel für ein DFA-Problem, dessen Formulierung für knoten- und kantenbenannte

- Instruktionsgraphen offensichtlich ist.
- Basisblockgraphen alles andere als ersichtlich, nicht möglich ist.

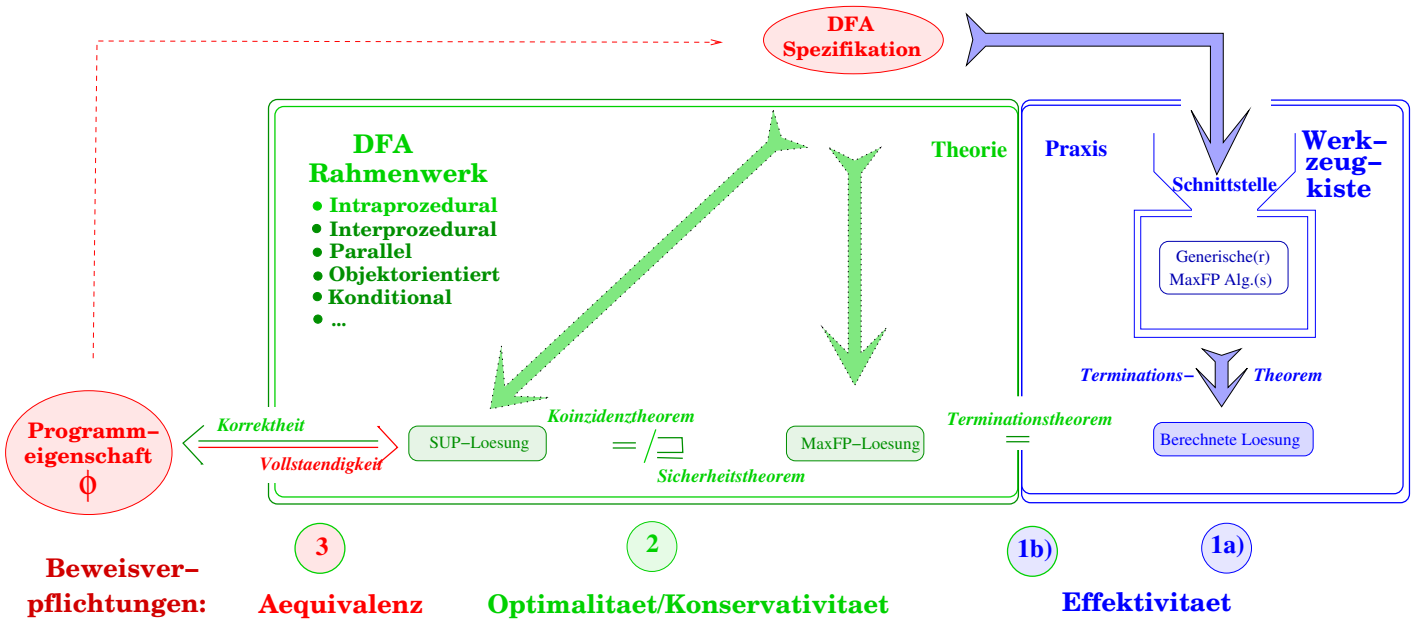
B.6

Zusammenfassung, Schlussfolgerungen

Zusammenfassung, Schlussfolgerungen

Alle 4 Flussgraphrepräsentationen sind grundsätzlich gleichwertig.

Konzeptuell reicht deshalb eine einzige gemeinsame Rahmen- bzw. Werkzeugkistensicht:



im Wissen, dass sie je nach Aufgabe unterschiedlich zweckmässig sind und unterschiedlich aufwändige Spezifikations-, Implementierungs- und Beweisverpflichtungen zur Folge haben.

Kapitel B.7

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Anhang B


Larry Carter, Jeanne Ferrante, Clark Thomborson. *Folklore Confirmed: Reducible Flow Graphs are Exponentially Larger*. In Conference Record of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2003), 106-114, 2003.


Jens Knoop. *From DFA-Frameworks to DFA-Generators: A Unifying Multiparadigm Approach*. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Springer-V., LNCS 1579, 360-374, 1999.


Jens Knoop, Dirk Koschützki, Bernhard Steffen. *Basic-block graphs: Living dinosaurs?* In Proceedings of the 7th International Conference on Compiler Construction (CC'98), Springer-V., LNCS 1383, 65 - 79, 1998.