Analyse und Verifikation

LVA 185.276, VU 2.0, ECTS 3.0
SS 2018

(Stand: 07.07.2018)

Jens Knoop

Technische Universität Wien
Information Systems Engineering
Compilers and Languages

Inhaltsverzeichnis

Inhaltsverzeichnis (1)

Inhaltsverzeichnis (2)

Inhaltsverzeichnis (3)

Teil II: Verifikation

Inhaltsverzeichnis (4)

Inhaltsverzeichnis (5)

Teil III: Analyse

Inhaltsverzeichnis (6)

Inhaltsverzeichnis (7)

Inhaltsverzeichnis (8)

Inhaltsverzeichnis (9)

Inhaltsverzeichnis (10)

Inhaltsverzeichnis (11)

Inhaltsverzeichnis (12)

Inhaltsverzeichnis (13)

Inhaltsverzeichnis (14)

Teil V: Abstrakte Interpretation und Modellprüfung

Inhaltsverzeichnis (15)

Inhaltsverzeichnis (16)

Inhaltsverzeichnis (17)

Teil I

Motivation

Kapitel 1

Grundlagen

Kapitel 1.1

Motivation

Programmiersprachen

...festgelegt durch Angabe von Syntax und Semantik.

- 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: Festlegung von Syntax und Semantik durch

- formale Regelwerke.

Vorteile formaler Regelwerke

...zur Festlegung von Syntax und Semantik von Programmiersprachen: Rigorisität!

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.

Programmiersprache WHILE

...als Modellsprache anhand derer wir die Angabe formaler Regelwerke zur Festlegung von Syntax und Semantik einer Programmiersprache beispielhaft illustrieren und demonstrieren.

Besondere Wichtigkeit kommt dabei der Semantik zu, die sich wie die Syntax einer Programmiersprache auf verschiedene Weise festlegen lässt. Wir sprechen von unterschiedlichen Definitionsstilen. Sie gewähren eine unterschiedliche Sicht auf die Bedeutung der Sprache und richten sich daher implizit an andere Adressaten.

Besonders grundlegend und wichtig sind der

- operationelle
- denotationelle

und mit abweichendem Fokus

- axiomatische

Semantikdefinitionsstil.

Semantikdefinitionsstile

- **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.

Adressaten/besondere Eignung

...von Semantikdefinitionsstilen.

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

Literaturhinweise (1)

...als Textbücher für Kapitel 1 bis 5:

- 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

Literaturhinweise (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.

Literaturhinweise (3)

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

Kapitel 1.2

Modellsprache WHILE

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

Die Menge wohlgeformter WHILE-Programme ist beschrieben durch folgende Backus-Naur-Regel (BNF):

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

wobei

- a für arithmetische Ausdrücke über Numeralen
- b für Wahrheitswertausdrücke

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

...wir bezeichnen mit:

- **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**

Ausblick: Semantik von WHILE (1)

Die Bedeutung eines WHILE-Programms π ist gegeben durch (partielle) Zustandstransformationen

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

wobei

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

die Menge aller Zustände über der Variablenmenge **Var** und einem geeigneten Datenbereich ID bezeichnet (in der Folge werden wir für ID meist die Menge der ganzen Zahlen $\mathbb{Z}$ betrachten).

Notationelle Konventionen: Der Pfeil $\to$ bezeichnet totale Funktionen, der Pfeil $\hookrightarrow$ partielle Funktionen; das Zeichen $=_{df}$ steht für 'definitionsgemäß gleich'.

Ausblick: Semantik von WHILE (2)

Im einzelnen legen wir auf drei Arten für WHILE eine Semantik fest:

- Operationelle Semantik
  - Natürliche Semantik:
    $[\![\ ]\!]_{ns} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$
  - Strukturell operationelle Semantik:
    $[\![\ ]\!]_{sos} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$
- Denotationelle Semantik: $[\![\ ]\!]_{ds} : \mathbf{Prg} \to (\Sigma \hookrightarrow \Sigma)$

...und deren Beziehungen zueinander untersuchen, d.h. die Beziehungen zwischen $[\![\ ]\!]_{ns}$, $[\![\ ]\!]_{sos}$ und $[\![\ ]\!]_{ds}$.

Anschließend betrachten wir die

- Axiomatische Semantik

die einen abweichenden Fokus auf Programmverifikation hat.

Ausblick: Semantik von WHILE (3)

Dabei stützen sich die Semantikfestlegungen für WHILE auf eine Semantik für

- Zahlwörter
- arithmetische Ausdrücke
- Wahrheitswertausdrücke

und den Begriff der

- Speicherzustände

ab. Mit deren Festlegung werden wir daher beginnen.

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:

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

wobei

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

Bemerkungen

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: $\mathbb{Z} =_{df} \{\ldots, \mathbf{-2}, \mathbf{-1}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$.
- $\mathit{plus}, \mathit{mal} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, $\mathit{minus} : \mathbb{Z} \to \mathbb{Z}$ 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 $\mathbb{Z}$

Semantik arithmetischer Ausdrücke (2)

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

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

...weitere Operatoren (*div*, *mod*, ^, ...) analog.

Beachte auch hier den Unterschied zwischen syntaktischen und semantischen Entitäten:

- $+$, $*$, $-$ bezeichnen syntaktische Entitäten (kurz: Operatoren).
- *plus*, *mal*, *minus* bezeichnen semantische Entitäten (kurz: 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 $[\![\, . \,]\!]_B : \textbf{Bexpr} \to (\Sigma \hookrightarrow \mathbb{B})$ definiert ist durch:

$$
\begin{array}{lll}
[\![\mathit{true}]\!]_B(\sigma) & =_{df} & \textbf{wahr} \\
[\![\mathit{false}]\!]_B(\sigma) & =_{df} & \textbf{falsch} \\
[\![a_1 = a_2]\!]_B(\sigma) & =_{df} & \begin{cases} \textbf{wahr} & \text{falls } \mathit{gleich}\,([\![a_1]\!]_A(\sigma), [\![a_2]\!]_A(\sigma)) \\ \textbf{falsch} & \text{sonst} \end{cases} \\
[\![a_1 > a_2]\!]_B(\sigma) & =_{df} & \begin{cases} \textbf{wahr} & \text{falls } \mathit{größer}\,([\![a_1]\!]_A(\sigma), [\![a_2]\!]_A(\sigma)) \\ \textbf{falsch} & \text{sonst} \end{cases}
\end{array}
$$

...weitere Relatoren $(\geq, \; <, \leq, \neq, \ldots)$ analog.

$$
\begin{array}{lll}
[\![b_1 \wedge b_2]\!]_B(\sigma) & =_{df} & \mathit{und}\,([\![b_1]\!]_B(\sigma), [\![b_2]\!]_B(\sigma)) \\
[\![b_1 \vee b_2]\!]_B(\sigma) & =_{df} & \mathit{oder}\,([\![b_1]\!]_B(\sigma), [\![b_2]\!]_B(\sigma)) \\
[\![\neg b]\!]_B(\sigma) & =_{df} & \mathit{neg}\,([\![b]\!]_B(\sigma))
\end{array}
$$

Semantik Boolescher Ausdrücke (3)

Dabei bezeichnen:

$$
\begin{array}{lcll}
\mathit{und} & : & \mathbb{B} \times \mathbb{B} \to \mathbb{B} & \text{(Logische Konjunktion)} \\
\mathit{oder} & : & \mathbb{B} \times \mathbb{B} \to \mathbb{B} & \text{(Logische Disjunktion)} \\
\mathit{neg} & : & \mathbb{B} \to \mathbb{B} & \text{(Logische Negation)} \\
\mathit{gleich} & : & \mathbb{Z} \times \mathbb{Z} \to \mathbb{B} & \text{(Gleichheitsrelation auf } \mathbb{Z}\text{)}
\end{array}
$$

Beachte: Wir werden später das Symbol $=$ überladen und statt *gleich* kürzer auch $=$ schreiben; aus dem Kontext geht jeweils hervor, ob das Symbol $=$ als

- Operatorsymbol wie in $[\![a_1 = a_2]\!]_B(\sigma)$
- Operationssymbol wie in $[\![a_1]\!]_A(\sigma) = [\![a_2]\!]_A(\sigma)$

verwendet wird.

Semantik Boolescher Ausdrücke (3)

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

Syntaktische Entitäten:

- *true* und *false* bezeichnen syntaktische Entitäten.
- $=$, $>$, $<$ bezeichnen syntaktische Entitäten (kurz: Relatoren).
- $\neg$, $\wedge$, $\vee$ bezeichnen syntaktische Entitäten (kurz: Operatoren).

Semantische Entitäten:

- **wahr** und **falsch** bezeichnen semantische Entitäten.
- *gleich*, *größer*, *kleiner* bezeichnen semantische Entitäten (kurz: Relationen (auf $\mathbb{Z}$)).
- *neg*, *und*, *oder* bezeichnen semantische Entitäten (kurz: Operationen (auf $\mathbb{B}$)).

Kapitel 1.6

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

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 $[\![\ ]\!]_N$, $[\![\ ]\!]_A$ und $[\![\ ]\!]_B$

Lemma 1.6.1

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

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: Das Symbol $=$ ist hier bereits überladen verwendet zur Bezeichnung der semantischen Gleichheitsrelation auf $\mathbb{Z}$ und $\mathbb{B}$.

Kapitel 1.7

Syntaktische und semantische Substitution

Substitution

...ein Begriff von zentraler Bedeutung, der in zwei Varianten auftritt:

- Syntaktische Substitution
- Semantische Substitution

Der Zusammenhang zwischen syntaktischer und semantischer Substitution wird beschrieben durch:

- Substitutionslemma 1.7.3

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:

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

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

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:

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

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 und
- $[[\![\ a'\ ]\!]_A(\sigma)/x]$ 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 $[\![\ ]\!]_B$)

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

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

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:

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

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

Beweise durch vollständige Induktion:

Lemma 1.8.1.2

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:

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

(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 ist definiert durch:

$$fib : \mathbb{N}_0 \to \mathbb{N}_0$$

$$fib(n) =_{df} \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2) & \text{if } n \geq 2 \end{cases}$$

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 zum Beweis von Lemma 1.8.2.1

...für $n \in \mathbb{N}_0$, $n \geq 2$, ist die Ausnutzung der Gleichheit

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

die aufgrund der Induktionshypothese (IH) für $m = n - 1$ und $m = n - 2$ angenommen werden kann.

(Beachte: Wir könnten im Fall von $n \geq 2$ diese Gleichheit aufgrund der Induktionshypothese 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:

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

(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 $n \in \mathbb{N}_0$, $n \geq 2$. Mithilfe der IH für $n-2$ u. $n-1$ 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 $(*)$

Die mit $(*)$ markierte Gleichheit gilt aufgrund folgender zwei Gleichheiten, die sich unter Ausnutzung der Binomialformeln zeigen lassen:

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

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

Übungsaufgabe

Sei die Funktion f definiert durch:

$$f : \mathbb{N}_0 \to \mathbb{N}_0$$

$$f(n) =_{df} \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ \sum_{k=0}^{n-1} f(k) & \text{falls } n \geq 2 \end{cases}$$

Beweise durch verallgemeinerte Induktion (unter Abstützung auf Lemma 1.8.2.3):

Lemma 1.8.2.2

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

Beweise durch vollständige Induktion:

Lemma 1.8.2.3

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

Kapitel 1.8.3

Strukturelle Induktion

Das Prinzip struktureller Induktion

Sei M eine induktiv aus einfacheren/einfachsten Elementen aufgebaute Menge, bezeichne $sub(m) \subseteq M$, $m \in M$, die endliche Menge einfacherer Elemente aus M, aus denen m aufgebaut ist, und sei E eine Eigenschaft der Elemente von M.

Das Prinzip struktureller Induktion:

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

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

Beispiel: Beweis von Lemma 1.6.2 (1)

Lemma 1.6.2 (wiederholt)

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).

Beispiel: Beweis von Lemma 1.6.2 (2)

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

Fall 1: Sei $a \equiv n$, $n \in$ **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 $a \equiv x$, $x \in$ **Var**. Mithilfe der Definition von $[\![\ ]\!]_A$ erhalten wir auch in diesem Fall unmittelbar die Gleichheit der rechten und linken Seite der Aussage des Lemmas:

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

Beispiel: Beweis von Lemma 1.6.2 (3)

Fall 3: Sei $a \equiv a_1 + a_2$, $a_1, a_2 \in$ **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. Damit können wir den Beweis wie folgt abschließen:

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

Fälle für weitere arithmetische Operatoren: Analog. □

Übungsaufgabe (1)

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 (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 $[\![\ ]\!]_B$)

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

Kapitel 1.8.4

Zusammenfassung

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.

Beachte

Abhängig vom Anwendungsfall

- ist oft eines der Induktionsprinzipien unmittelbarer, einfacher und damit zweckmäßiger anzuwenden.

Zum Beweis von Aussagen oder Eigenschaften

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

Kapitel 1.9

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)


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

Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (4)


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)


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.

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

Strukturell operationelle Semantik

...beschreibt den Ablauf der einzelnen Berechnungsschritte, die stattfinden; daher auch die Bezeichnung Kleinschritt-Semantik.

Grundlegende Arbeiten

...zur strukturell operationellen Semantik:

- Gordon D. Plotkin. A Structural Approach to Operational Semantics. Journal of Logic and Algebraic Programming 60-61:17-139, 2004.
- 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, 1982.
- Gordon D. Plotkin. A Structural Approach to Operational Semantics. Lecture notes, DAIMI FN-19, Aarhus University, Dänemark, 1981, Nachdruck von 1991.

Strukturell operationelle Semantik von WHILE

...die strukturell operationelle Semantik (oder SO-Semantik) von WHILE (im Sinne von Gordon D. Plotkin) ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation zu:

$$\Sigma \hookrightarrow \Sigma$$

Die SO-Semantik von WHILE ist demnach durch ein Zustandstransformationsfunktional $[\![\, . \,]\!]_{sos}$

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

gegeben, das wir in der Folge definieren.

Speicherzustandsübergänge, Konfigurationen

...die SO-Semantik von WHILE beschreibt den Berechnungsvorgang von Programmen $\pi \in \mathbf{Prg}$ als Folge elementarer

- Speicherzustandsübergänge.

Zentral: Der Begriff der

- Konfiguration

als Paar von (Rest-) Programm und Zustand bzw. Endzustand.

Konfigurationen

Definition 2.1.1 (Konfigurationen)

- Eine Konfiguration ist ein Element der Form $\langle \pi, \sigma \rangle$ oder σ, wobei $\pi \in \mathbf{Prg}$ ein WHILE-Programm und $\sigma \in \Sigma$ ein Zustand ist.
- $\Gamma =_{df} (\mathbf{Prg} \times \Sigma) \cup \Sigma$ bezeichnet die Menge aller Konfigurationen, γ eine einzelne Konfiguration.
- Eine Konfiguration der Form
  - $\langle \pi, \sigma \rangle$ heißt nichtterminal (oder Zwischenkonfiguration):

    ...das (Rest-) Programm π ist auf den (Zwischen-) Zustand σ anzuwenden.
  - σ heißt terminal (oder final):

    ...der Zustand σ ist der Resultatzustand nach Ende einer (regulären) Berechnung.

SOS-Regelwerk von WHILE: Axiome (1)

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

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

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

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

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

SOS-Regelwerk von WHILE: Regeln (2)

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

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

Regelwerke: Axiome und Regeln

..wir unterscheiden in Regelwerken:

- Prämissenlose Regeln, sog. Axiome, der Form

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

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

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

jeweils mit optionalen Randbedingungen (oder Seitenbedingungen) wie z.B. im Axiom $[\text{if}^{ff}_{sos}]$ in Form von $[\![\, b \,]\!]_B(\sigma) = \textbf{falsch}$.

Das Regelwerk der SO-Semantik von WHILE

...besteht aus:

- 5 Axiomen
  ...für die leere Anweisung (1), Zuweisung (1), Fallunterscheidung (2) und while-Schleife (1).
- 2 Regeln
  ...für die sequentielle Komposition (2).

Berechnungsschritt, Berechnungsfolge

Definition 2.1.2 (Berechnungsschritt)

Ein (SOS-) Berechnungsschritt ist von der Form

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

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

Definition 2.1.3 (Berechnungsfolge)

Eine (SOS-) Berechnungsfolge eines Programms π angesetzt auf einen (Start-) Zustand $\sigma \in \Sigma$ ist eine

- endliche Folge $\gamma_0, \ldots, \gamma_k$ von Konfigurationen mit $\gamma_0 = \langle \pi, \sigma \rangle$ und $\gamma_i \Longrightarrow \gamma_{i+1}$ 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 $\gamma_i \Longrightarrow \gamma_{i+1}$ für alle $i \in \mathbb{N}$.

Eigenschaften maximaler Berechnungsfolgen

...reguläre und irreguläre Terminierung, Divergenz:

Definition 2.1.4 (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 $\pi \in \mathbf{Prg}$ und Anfangszustand $\sigma \in \Sigma$ mit

- $\pi \equiv y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$
  (Bemerkung: $\equiv$ steht für 'syntaktisch identisch')
- $\sigma(x) = \mathbf{3}$, $\sigma(y)$ beliebig $\in \mathbb{Z}$ für $y \neq x$.

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

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

induzierte Berechnungsfolge.

Beispiel: Illustration der SO-Semantik (2)

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

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

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

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

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

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

Beispiel: Illustration der SO-Semantik (3)

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

Beispiel: Illustration der SO-Semantik (4)

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

Detailbetrachtung: Korrektheitsargument (1)

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

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

...steht vereinfachend und abkürzend für:

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

Detailbetrachtung: Korrektheitsargument (2)

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

...steht vereinfachend und abkürzend für:

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

Detailbetrachtung: Korrektheitsargument (3)

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

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

...steht vereinfachend und abkürzend für:

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

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

SOS-Regeln: Determiniertheit, Determinismus

Lemma 2.1.5

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

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

Korollar 2.1.6

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 2.1.6 können wir festlegen:

Definition 2.1.7 (SO-Semantik von WHILE)

Die strukturell operationelle Semantik (oder SO-Semantik) von WHILE ist gegeben durch das Funktional

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

das definiert ist durch:

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

wobei $\Longrightarrow\!\!\!\!\Longrightarrow^{*}$ die reflexiv-transitive Hülle von $\Longrightarrow\!\!\!\!\Longrightarrow$ bezeichnet.

Induktion über Berechnungsfolgenlänge

...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 zum Nachweis von Aussagen oder Eigenschaften im Zusammenhang mit strukturell operationeller Semantik.

Ein typisches Beispiel hierfür ist der Beweis der Aussage von:

Lemma 2.1.8

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

Kapitel 2.2

Natürliche Semantik

Natürliche Semantik

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

Natürliche Semantik von WHILE

...die natürliche Semantik (oder N-Semantik) von WHILE ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation zu:

$$\Sigma \hookrightarrow \Sigma$$

Die N-Semantik von WHILE ist demnach durch ein Zustandstransformationsfunktional $[\![.]\!]_{ns}$

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

gegeben, das wir in der Folge definieren.

Initialer/finaler Zustand, Konfigurationen

...die N-Semantik ist am Zusammenhang zwischen

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

Speicherzustand einer Berechnung eines Programms interessiert.

Zentral auch hier: Der bereits von der SO-Semantik bekannte Begriff der

- Konfiguation (s. Definition 2.1.1).

NS-Regelwerk von WHILE: Axiome (1)

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

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

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

NS-Regelwerk von WHILE: Regeln (2)

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

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

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

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

Das Regelwerk der N-Semantik von WHILE

...besteht aus:

- 3 Axiomen
  ...für die leere Anweisung (1), Zuweisung (1), und while-Schleife (1).
- 4 Regeln
  ...für die while Schleife (1), Fallunterscheidung (2) und sequentielle Komposition (1).

Beispiel: Illustration der N-Semantik (1)

Gegeben: Programm $\pi \in \mathbf{Prg}$ und Anfangszustand $\sigma \in \Sigma$ mit

- $\pi \equiv y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$
  (Bemerkung: $\equiv$ steht für 'syntaktisch identisch')
- $\sigma(x) = \mathbf{3}$, $\sigma(y)$ beliebig $\in \mathbb{Z}$ für $y \neq x$.

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

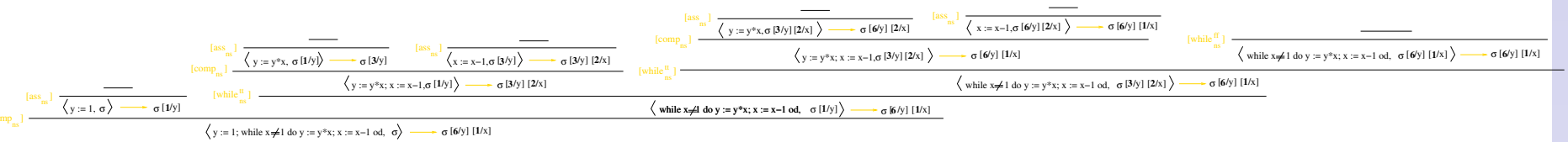
- $\langle y := 1;\ \text{while } x /= 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma \rangle$

Behauptung:

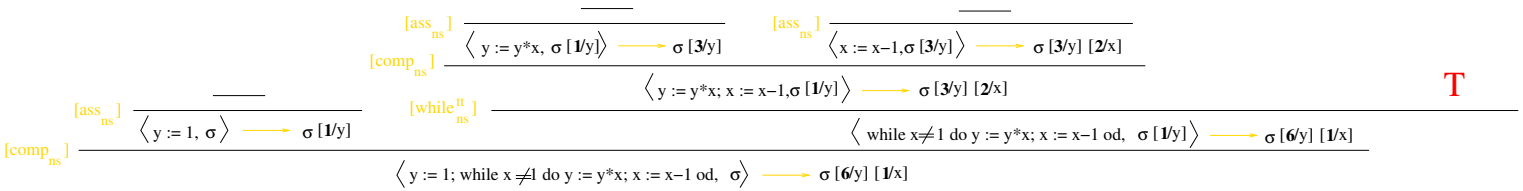
$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma \rangle \longrightarrow \sigma[\mathbf{6}/y][\mathbf{3}/x]$$

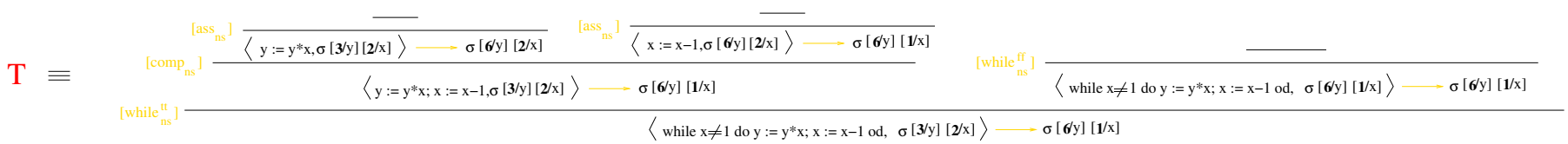
Beispiel: Illustration der N-Semantik (2)

...der die Behauptung beweisende Ableitungsbaum gemäß des Regelwerks der N-Semantik:



...der gleiche Ableitungsbaum geringfügig größer durch Einführung des benannten Teilbaums T:





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 können wir festlegen:

Definition 2.2.3 (N-Semantik von WHILE)

Die natürliche Semantik (oder N-Semantik) von WHILE ist gegeben durch das Funktional

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

das definiert ist durch:

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

Induktion über Ableitungsbäume

....als Variante induktiver Beweisführung.

Induktion über die Form 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 die Form von Ableitungsbäumen ist typisch zum Nachweis von Aussagen oder Eigenschaften im Zusammenhang mit natürlicher Semantik.

Ein typisches Beispiel hierfür ist der Beweis der Aussage von Lemma 2.2.1 aus diesem Kapitel:

Lemma 2.2.1 (wiederholt)

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

Kapitel 2.3

Äquivalenz strukturell operationeller und natürlicher Semantik

Strukturell operationelle Semantik

Der Fokus liegt auf den

- individuellen Schritten einer Berechnungsfolge, d.h. auf der Ausführung von Zuweisungen und Tests.

Intuitiv haben die Transitionen $\langle \pi, \sigma \rangle \ggg \gamma$ der Transitionsrelation $\ggg$ folgende Bedeutung:

- Eine Transition beschreibt den ersten Schritt der Berechnungsfolge von π angesetzt auf σ.

Dabei sind folgende Übergänge möglich: γ ist von der Form

- $\langle \pi', \sigma' \rangle$: 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.

Natürliche Semantik

Der Fokus liegt auf dem

- Zusammenhang von initialem und finalem Zustand einer Berechnungsfolge.

Intuitiv haben die Transitionen $\langle \pi, \sigma \rangle \to \sigma'$ der Transitionsrelation $\to$ folgende Bedeutung:

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

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 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 strukturell operationeller und natürlicher Semantik von WHILE.

Aus Lemma 2.3.1 und Lemma 2.3.2 folgt unmittelbar:

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

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

Kapitel 2.4

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 2 (1)

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 (2)


Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Lecture notes, DAIMI FN-19, Aarhus University, Dänemark, 1981, 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, 1982.


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

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.*

Fokus operationeller, denotationeller Semantik

...der Fokus

- operationeller Semantik liegt darauf, wie ein Programm ausgeführt wird, nicht auf dem (Gesamt-) Effekt, den es (im Sinn einer Abbildung) bewirkt.
- denotationeller Semantik liegt auf dem (Gesamt-) Effekt (im Sinn einer Abbildung), den die Ausführung eines Programms hat, nicht darauf, wie dieser Effekt erreicht wird.

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

Schlüsseleigenschaft

...Kompositionalität der semantischen Funktionen.

Intuitiv: Für jedes

- elementare syntaktische Konstrukt gibt es eine semantische Funktion, 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.

Kapitel 3.1

Denotationelle Semantik

Denotationelle Semantik von WHILE

...die denotationelle Semantik (oder D-Semantik) von WHILE ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation zu:

$$\Sigma \hookrightarrow \Sigma$$

Die D-Semantik von WHILE ist demnach durch ein Zustandstransformationsfunktional $[\![\, . \,]\!]_{ds}$

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

gegeben, das wir in der Folge definieren.

D-Regelwerk von WHILE: Definierende Abb.

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

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

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

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

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

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

Dabei bezeichnen:

- *id*: Identische Zustandstransformation.
- *cond*: Fallunterscheidungsfunktional.
- *FIX*: Fixpunkt-Zustandstransformationsfunktional.

Zu den (Hilfs-) Funktionen und -funktionalen

...*id*, *cond* und *FIX*:

- $id : \Sigma \to \Sigma$: Identische Zustandstransformation definiert durch: $\forall\, \sigma \in \Sigma.\ id(\sigma) =_{df} \sigma$.
- $cond : (\Sigma \hookrightarrow \mathbb{B}) \times (\Sigma \hookrightarrow \Sigma) \times (\Sigma \hookrightarrow \Sigma) \to (\Sigma \hookrightarrow \Sigma)$: Fallunterscheidungsfunktional.
- $FIX : ((\Sigma \hookrightarrow \Sigma) \to (\Sigma \hookrightarrow \Sigma)) \to (\Sigma \hookrightarrow \Sigma)$: Fixpunkt-Zustandstransformationsfunktional, das den kleinsten Fixpunkt des Argumentfunktionals liefert.

Das Funktional *cond*

Fallunterscheidungsfunktional

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

definiert durch

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

Anmerkung zu den Argumenten und zum Resultat von *cond*:

- 1. Argument: Ein partiell definiertes Prädikat.
- 2. Argument, 3. Argument: Je eine partiell definierte Zustandstransformation.
- Resultat: Eine partiell definierte Zustandstransformation.

Mithilfe von *cond*

... ergibt sich für die denotationelle Bedeutung des Fallunterscheidungskonstrukts 'if-then-else-fi':

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

$$= \quad \mathit{cond}\,([\![b]\!]_B, [\![\pi_1]\!]_{ds}, [\![\pi_2]\!]_{ds})(\sigma)$$

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

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

Das Funktional *FIX*

Fixpunkt-Zustandstransformationsfunktional

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

Im Zusammenhang mit der D-Semantik des while-Konstrukts

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

wird *FIX* angewendet auf ein ganz bestimmtes Zustandstransformationsfunktional, das Funktional

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

das definiert ist durch:

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

Zur D-Semantik des while-Konstrukts (1)

Jede der Intuition der Bedeutung von Fallunterscheidung und while-Schleife entsprechende Festlegung der D-Semantik von Fallunterscheidungs- und while-Konstrukt muss erfüllen:

A) Die Anweisungen while b do π od und if b then (π; while b do π od) else *skip* fi haben denselben Effekt, d.h. ihre D-Semantik muss übereinstimmen:

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

Zusammen mit A) liefern die D-Regeln für das Fallunterscheidungskonstrukt, das sequentielle Kompositions- und das *skip*-Konstrukt damit folgende Gleichheit:

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

Zur D-Semantik des while-Konstrukts (2)

Die Definition des Funktionals F liefert zusammen mit der D-Regel für das while-Konstrukt folgende Gleichheit:

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

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

Zusammen implizieren B) und C) die Gleichheit:

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

Das heißt: Die D-Semantik des while-Konstrukts ist *ein* Fixpunkt des Funktionals F.

Die D-Regel des while-Konstrukts legt fest:

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

Das heißt: Die D-Semantik des while-Konstrukts ist *der eindeutig bestimmte* kleinste Fixpunkt des Funktionals F.

Hauptresultat

...die definierenden Gleichungen des D-Regelwerks von WHILE sind 'vernünftig':

Theorem 3.1.1 (Determiniertheit, Determinismus)

Für alle WHILE-Programme $\pi \in \mathbf{Prg}$ ist durch das D-Regelwerk von WHILE in eindeutiger Weise eine partielle Zustandstransformation festgelegt.

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

...dank Theorem 3.1.1 können wir festlegen:

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 $\pi \in \mathbf{Prg}$ die partielle Zustandstransformation $[\![\ \pi\ ]\!]_{ds}$ diejenige gemäß Theorem 3.1.1 gegebene und eindeutig bestimmte partielle Zustandstransformation ist.

Kapitel 3.2

Fixpunktfunktional und Wohldefiniertheit

Zur Wohldefiniertheit

...von *FIX* und *FIX F*.

Arbeitsplan:

Wir stellen zunächst einige Resultate aus der

- Fixpunkttheorie

zusammen, die wir benötigen.

Anschließend zeigen wir, dass diese Resultate auf das D-Regelwerk

- anwendbar sind.

Dabei setzen wir voraus: Mathematische Grundlagen über

- Ordnungen, vollständige partielle Ordnungen (CPOs), Stetigkeit von Funktionen auf CPOs, das Fixpunkttheorem von Knaster, Tarski und Kleene (siehe Anhang A 'Mathematische Grundlagen' für Details).

Folgende Resultate

...sind zentral:

1. Die Menge der partiellen Zustandstransformationen $ZT =_{df} [\Sigma \hookrightarrow \Sigma]$ kann vollständig partiell geordnet werden, d.h. das Paar $([\Sigma \hookrightarrow \Sigma], \sqsubseteq)$ ist für eine geeignete Relation $\sqsubseteq \subseteq ZT \times ZT$ eine CPO.
2. Das Funktional F ist im Kontext des D-Regelwerks stetig.
3. Fixpunktbildung wird im Kontext des D-Regelwerks ausschließlich auf stetige Funktionen angewendet.

Insgesamt folgt aus 1.), 2.) und 3.) die Wohldefiniertheit von

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

Ordnung auf Zustandstransformationen

Bezeichne $ZT =_{df} [\Sigma \hookrightarrow \Sigma]$ die Menge der

- partiellen Zustandstransformationen.

Wir definieren folgende Ordnung(srelation) $\sqsubseteq$ auf ZT:

$$\forall\, g_1, g_2 \in ZT.\ g_1 \sqsubseteq g_2 \iff_{df}$$
$$\forall\, \sigma \in \Sigma.\ g_1(\sigma)\ \mathit{definiert} = \sigma' \Rightarrow g_2(\sigma)\ \mathit{definiert} = \sigma'$$

Lemma 3.2.1 (Partielle Ordnung, kleinstes Element)

1. Das Paar $(ZT, \sqsubseteq)$ ist eine partielle Ordnung.
2. Die total undefinierte (d.h. nirgends definierte) Zustandstransformation $\bot \in ZT$ mit $\bot(\sigma) = \mathit{undef}$ für alle $\sigma \in \Sigma$ ist das eindeutig bestimmte kleinste Element in ZT bezüglich der Ordnungsrelation $\sqsubseteq$.

Der Graph totaler Funktionen

Definition 3.2.2 (Graph einer totalen Funktion)

Der Graph einer totalen Funktion $f : M \to N$ ist eine Relation *graph* auf $M \times N$ definiert durch:

$$graph(f) =_{df} \{\langle m, n\rangle \mid f(m) = n\} \subseteq M \times N$$

Lemma 3.2.3

Die Relation *graph* einer totalen Funktion $f : M \to N$ ist:

1. rechtseindeutig, d.h. $\forall\, m \in M.\ \forall\, n \in N.\ \langle m, n\rangle \in graph(f) \ \wedge\ \langle m, n'\rangle \in graph(f) \Rightarrow n = n'$
2. linkstotal, d.h. $\forall\, m \in M.\ \exists\, n \in N.\ \langle m, n\rangle \in graph(f)$.

Der Graph partieller Funktionen

Definition 3.2.4 (Graph einer partiellen Funktion)

Der Graph einer partiellen Funktion $f : M \hookrightarrow N$ mit Definitionsbereich $M_f \subseteq M$ ist eine 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 $f : M \hookrightarrow N$ partiell definierte Funktion auf $M_f \subseteq M$ schreiben wir:

- $f(m) = n$, falls $\langle m, n\rangle \in graph(f)$
- $f(m) = undef$, falls $m \notin M_f$

Ordnungen, Ketten, Schranken

Wir definieren:

- $\forall f, g : M \to N.\ f \sqsubseteq g \iff_{df} graph(f) \subseteq graph(g)$
- Eine Menge $G =_{df} \{g_1, g_2, g_3, \ldots\} \subseteq [M \to N]$ von Funktionen heißt eine Kette, wenn G total geordnet ist bezüglich $\sqsubseteq$, d.h. die Elemente von G lassen sich anordnen in der Form $g_1 \sqsubseteq g_2 \sqsubseteq g_3 \sqsubseteq \ldots$.
- Eine Funktion $g : M \to N$ heißt obere Schranke einer Menge $G =_{df} \{g_1, g_2, g_3, \ldots\} \subseteq [M \to N]$ von Funktionen, wenn gilt: $\forall g' \in G.\ g' \sqsubseteq g$.
- Eine Funktion $g : M \to N$ heißt kleinste obere Schranke von $G =_{df} \{g_1, g_2, g_3, \ldots\} \subseteq [M \to N]$, wenn gilt:
  1. g ist obere Schranke von G.
  2. $\forall g' : [M \to N].$ (g' obere Schranke von G). $g \sqsubseteq g'$.

Beachte: Die obigen Festlegungen können in natürlicher Weise auf partielle Funktionen ausgedehnt werden

Schranken von Ketten (partieller) Funktionen

Lemma 3.2.5

Sei $Y \subseteq [M \hookrightarrow N]$ eine Kette (partieller) Funktionen. Dann gilt: Die kleinste obere Schranke von Y, in Zeichen: $\bigsqcup Y$, existiert und ist gegeben durch:

$$\mathit{graph}(\bigsqcup Y) \ \mathit{existiert} \ = \ \bigcup \{\mathit{graph}(g) \mid g \in Y\}$$

Beachte: Eine Funktion $g : M \hookrightarrow N$ kann mit ihrem Graphen $\mathit{graph}(g)$ identifiziert werden und umgekehrt.

Bezogen auf Lemma 3.2.5 heißt das:

$$\forall\, m \in M.\ (\bigsqcup Y)(m) \ = \ n \iff \exists\, g \in Y.\ g(m) \ = \ n$$

CPO auf Zustandstransformationen

...die Anwendung der vorherigen Ergebnisse auf die Menge ZT partieller Zustandstransformationen liefert:

Lemma 3.2.6 (Vollständige partielle Ordnung)

Das Paar $(ZT, \sqsubseteq)$ ist eine vollständige partielle Ordnung (oder CPO) mit kleinstem Element $\bot$.

Definition 3.2.7 (Vollständige partielle Ordnung)

Eine partielle Ordnung $(P, \sqsubseteq)$ heißt eine vollständige partielle Ordnung (engl. complete partial order (CPO)), falls jede (aufsteigende) Kette $C \subseteq P$ eine kleinste obere Schranke $\bigsqcup C$ in P besitzt, d.h. $\bigsqcup C$ *exists* $\in P$.

Monotonie und Stetigkeit

...von Funktionen auf vollständigen partiellen Ordnungen.

Definition 3.2.8 (Monotonie, Stetigkeit)

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

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

Stetigkeitsresultate

Lemma 3.2.9

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

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

Dann gilt: F ist stetig.

Lemma 3.2.10

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

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

Dann gilt: F ist stetig.

Summa summarum

Zusammen mit:

Lemma 3.2.11

Das D-Regelwerk von WHILE definiert eine totale Funktion:

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

...sind wir durch. Wir können zeigen:

Theorem 3.2.12 (Wohldefiniertheit von $[\![\ ]\!]_{ds}$)

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

Beweisskizze

Aus

1. Die Menge $ZT =_{df} [\Sigma \hookrightarrow \Sigma]$ der partiellen Zustandstransformationen bildet zusammen mit der Ordnung $\sqsubseteq$ eine CPO.
2. Das Funktional F defniert durch $F\ g = \mathit{cond}\,(p, g, g_0)$ und die sequentielle Komposition $g \circ g_0$ von g und g_0 sind stetig.
3. Bei der Definition von $[\![\ ]\!]_{ds}$ wird Fixpunktbildung ausschließlich auf stetige Funktionen auf einer CPO angewendet.

...folgt insgesamt die Wohldefiniertheit der D-Semantik:

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

Kapitel 3.3

Äquivalenz denotationeller und operationeller Semantik

Äquivalenz von $[\![\ ]\!]_{ds}$ und $[\![\ ]\!]_{sos}$ (1)

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 $[\![\ ]\!]_{ds}$ und $[\![\ ]\!]_{sos}$ (2)

Lemma 3.3.2

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

Beweis von

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

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

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

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

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

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

Aus Lemma 3.3.1 und Lemma 3.3.2 folgt:

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:

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

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

Kapitel 3.4

Schlussfolgerung zur Semantik von WHILE

Schlussfolgerung

...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.) durch:

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

Kapitel 3.5

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 3


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)


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

Teil II

Verifikation

Kapitel 4

Axiomatische Semantik von WHILE

Axiomatische Semantik von WHILE

...bestimmte Eigenschaften des Effekts der Ausführung eines Konstrukts werden als Zusicherungen ausgedrückt. Bestimmte andere Aspekte der Ausführung werden dabei i.a. ignoriert.

Kapitel 4.1

Direkte Programmverifikation

Programmverifikation

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

- Eigenschaften

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

Dabei lässt sich grob unterscheiden zwischen

- partiellen
- totalen

Korrektheitseigenschaften.

Partielle, totale Korrektheitseigenschaften

...für ein Programm π.

Partielle Korrektheitseigenschaften von π garantieren:

- Wenn π angesetzt auf einen Zustand σ regulär terminiert in einem Zustand σ', dann stehen die Werte der Variablen von σ und σ' in einer bestimmten Beziehung zueinander.

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 zueinander.

Informell:

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

Beispiel: Programm und Programmeigenschaft

Betrachte das (kanonische) Fakultätsprogramm:

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

und folgende Eigenschaft

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

definiert durch:

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

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

$$! : \mathbb{N}_0 \to \mathbb{N}_1$$

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

Partielle, totale Korrektheit von π bzgl. E

Lemma 4.1.1 (Partielle Korrektheit von π bzgl. E)

Wenn π angesetzt auf einen Zustand $\sigma \in \Sigma$ regulär in einem Zustand $\sigma' \in \Sigma$ terminiert, dann gilt $E(\sigma, \sigma')$, d.h.:

$$\sigma'(y) = \sigma(a)!$$

d.h. π ist partiell korrekt für E (für jeden Anfangszustand σ).

Lemma 4.1.2 (Totale Korrektheit von π bzgl. E)

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.:

$$\sigma'(y) = \sigma(a)!$$

d.h. π ist total korrekt für E (für jeden Anfangszustand σ mit $\sigma(a) \geq 0$).

Beweisskizze für Lemma 4.1.1 (1)

Wegen $[\![\ ]\!]_{sos} = [\![\ ]\!]_{ns} = [\![\ ]\!]_{ds}$ (Äquivalenztheorem 3.3.4) reicht es bei Wahl der

- strukturell operationellen Semantik zu zeigen:

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

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

  $$\langle \pi, \sigma \rangle \Longrightarrow^* \sigma'$$

  bewiesen werden kann.
- natürlichen Semantik zu zeigen:

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

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

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

  bewiesen werden kann.

Beweisskizze für Lemma 4.1.1 (2)

...reicht es bei Wahl der

- denotationellen Semantik 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:

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

wobei $(***)$ nach Beweis der Zulässigkeit von ψ (s. Definition A.6.1) mittels Fixpunktinduktion (s. Anhang A.6) bewiesen werden kann.

Übungsaufgabe

Vervollständige den Beweis von Lemma 4.1.1 mit Wahl der

1. strukturell operationellen
2. natürlichen
3. denotationellen

Semantik für π.

Bobachtung

...anhand der (hier nicht) vollständig ausgeführten Beweise von Lemma 4.1.1:

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

Kapitel 4.2

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:

Zentral die 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.

Bestandteile von Zusicherungen

In Zusicherungen der Form

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

...ist

- π ein Programm

...heißen

- p Vorbedingung
- q Nachbedingung

der Zusicherungen.

Für die Wahl der Grundmenge von Vor- und Nachbedingungen gibt es verschiedene Möglichkeiten, die auf extensionale und intensionale Ansätze axiomatischer Programmverifikation führen.

Extensionale vs. intensionale Ansätze

...axiomatischer Programmverifikation.

Extensionale Ansätze: Prädikate als Grundmenge

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

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:

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

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

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

Sprechweisen von ‘erfüllt in’

...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 axiomatischer Semantik und Programmverifikation:

- 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 4.2.1

Partielle und totale Korrektheit

Partielle Korrektheit

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

Definition 4.2.1.1 (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 zugehörige Berechnung von π angesetzt auf σ regulär in einem 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 4.2.1.2 (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 zugehörige Berechnung von π angesetzt auf σ regulär mit einem 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!

Informell

Totale Korrektheit “gleich”

Partielle Korrektheit “plus” Reguläre Termination

Charakterisierung logischer Formeln, Prädikate

...durch erfüllende Zustandsmengen.

Definition 4.2.1.3 (Charakterisierung)

1. Sei p eine logische Formel. Die Zustandsmenge

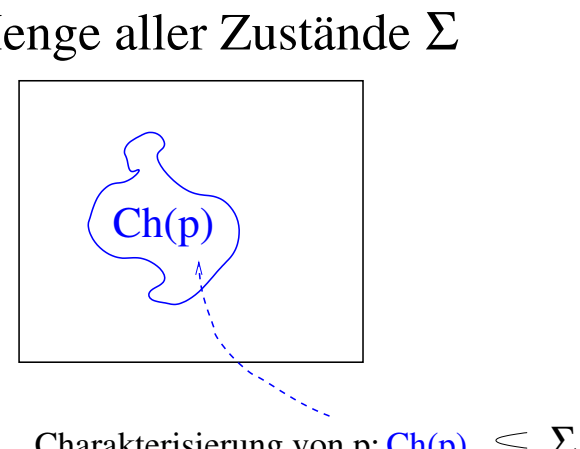
$$Ch(p) =_{df} \{\sigma \in \Sigma \mid [\![p]\!]_B(\sigma) = \textbf{wahr}\}$$

2. Sei p eine Prädikat. Die Zustandsmenge

$$Ch(p) =_{df} \{\sigma \in \Sigma \mid p(\sigma) = \textbf{wahr}\}$$

heißt Charakterisierung von p.

Veranschaulichung:



Partielle und totale Korrektheit

...ausgedrückt über die Charakterisierung von Vor- und Nachbedingungen von Korrektheitsformeln.

Lemma 4.2.1.4 (Charakterisierungslemma)

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

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

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

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

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

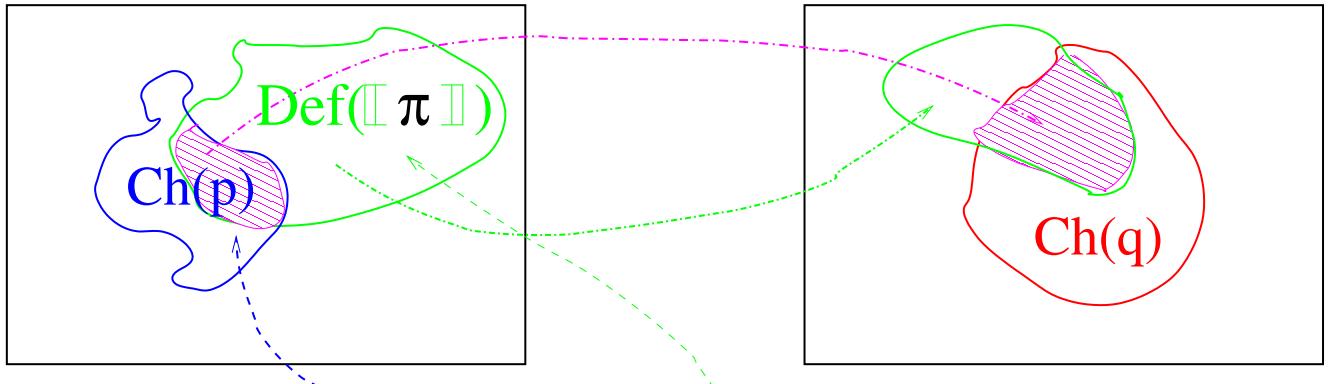
$$\{p\}\ \pi\ \{q\} \text{ partiell korrekt } \wedge\ Ch(p) \subseteq Def([\![\ \pi\ ]\!])$$

   wobei $Def([\![\ \pi\ ]\!])$ den Definitionsbereich von π bezeichnet, die Menge der Zustände, für die π regulär terminiert.

Veranschaulichung von Lemma 4.2.1.4(1)

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

Menge aller Zustände Σ



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

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

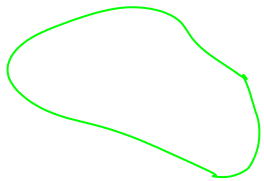


Bild von $\llbracket \pi \rrbracket$ für $Def(\llbracket \pi \rrbracket)$

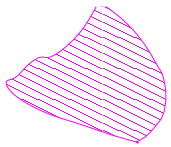
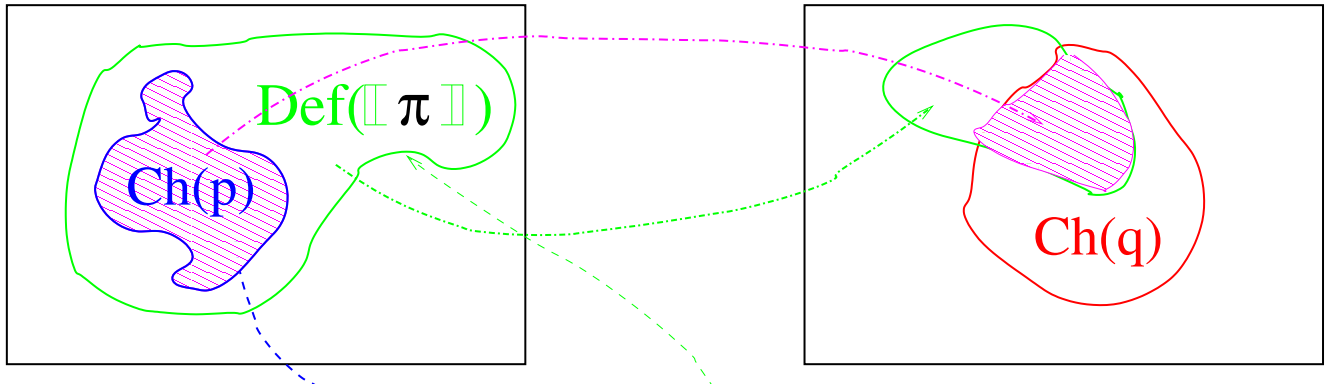


Bild von $\llbracket \pi \rrbracket$ für $Def(\llbracket \pi \rrbracket) \wedge Ch(p)$

Veranschaulichung von Lemma 4.2.1.4(2)

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

Menge aller Zustände Σ



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

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

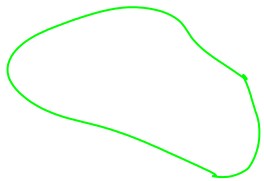


Bild von $\llbracket \pi \rrbracket$ für $Def(\llbracket \pi \rrbracket)$

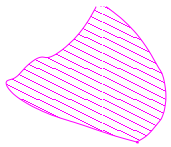


Bild von $\llbracket \pi \rrbracket$ für $Def(\llbracket \pi \rrbracket) \wedge Ch(p)$

Zusammenhang

...partieller und totaler Korrektheit:

Lemma 4.2.1.5

Für WHILE-Programme π gilt:

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

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

Beispiele Hoarescher Zusicherungen

...für partielle Korrektheit:

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

...für totale Korrektheit:

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

des WHILE-Programms zur Berechnung der Fakultätsfunktion.

Lesart von Vor- und Nachbedingungen

...als Prädikat (extensional) oder logische Formel, hier Boolesche Ausdrücke (intensional):

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

Vorbedingungen $true$, $a \geq 0$ und Nachbedingung $y = a!$ können gelesen werden

- intensional als Boolesche Ausdrücke
  $true, (a \geq 0), (y = a!) \in \mathbf{Bexpr}$.
- extensional als Prädikate
  $true, (a \geq 0), (y = a!) \in [\Sigma \to \mathbb{B}]$ definiert durch
  - $\forall\, \sigma \in \Sigma.\ (true)(\sigma) =_{df} \mathbf{wahr}$
  - $\forall\, \sigma \in \Sigma.\ (a \geq 0)(\sigma) =_{df} \mathit{größergleich}(\sigma(a), \mathbf{0})$
  - $\forall\, \sigma \in \Sigma.\ (y = a!)(\sigma) =_{df} \mathit{gleich}(\sigma(y), \sigma(a)!)$

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

Programmvariablen vs. logische Variablen

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

- Programmvariablen
  ...Variablen, die in π vorkommen (und deren Wert möglicherweise durch π verändert wird).
- logischen Variablen
  ...Variablen, auf die in 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

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

...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 (1)

...Hoaresche Zusicherungen (oder Korrektheitsformeln) Tripel sind von der Form

- $\{p\}\ \pi\ \{q\}$ im Sinn partieller Korrektheit.
- $[p]\ \pi\ [q]$ im Sinn totaler Korrektheit.

Dabei sind

- π ein Programm.
- p, q als Vor- und Nachbedingung bezeichnete Prädikate (extensional) oder logische Formeln (intensional), meist prädikatenlogische Formeln 1. Stufe.
- p, q können logische und Programmvariablen enthalten.

Zusammenfassung (2)

...für Tripel der Form

- $\{p\}\ \pi\ \{q\}$, $[p]\ \pi\ [q]$

betont die Sprechweise

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

jeweils die

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

auf die Tripel.

Kapitel 4.2.2

Stärkste Nachbedingungen, schwächste Vorbedingungen, schwächste liberale Vorbedingungen

Stärkere, schwächere

...Formeln und Prädikate.

Definition 4.2.2.1 (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 4.2.2.2

$$p \text{ stärker als } q \iff q \text{ schwächer als } p$$

Stärkste, schwächste

...Formeln oder Prädikate relativ zu einer Eigenschaft.

Seien p, q logische Formeln oder Prädikate, E eine Eigenschaft logischer Formeln oder Prädikate.

Definition 4.2.2.3 (Relativ stärkst, relativ schwächst)

- p heißt stärkst relativ zu E (oder relativ E-stärkst) für q gdw
  1. p erfüllt E.
  2. p stärker als q.
  3. $\forall\, p'.\, (p'$ erfüllt $E \wedge p'$ stärker als $q).\ p$ stärker als p'.
- p heißt schwächst relativ zu E (oder relativ E-schwächst) für q gdw
  1. p erfüllt E.
  2. p schwächer als q.
  3. $\forall\, p'.\, (p'$ erfüllt $E \wedge p'$ schwächer als $q).\ p$ schwächer als p'.

Schwächste liberale Vorbedingungen

...im Kontext mit partieller Korrektheit.

Definition 4.2.2.4 (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

$$\{wlp(\pi, q)\}\ \pi\ \{q\}$$

partiell korrekt ist und $wlp(\pi, q)$ die schwächste Formel oder Prädikat mit dieser Eigenschaft ist.

...d.h. $wlp(\pi, q)$ ist schwächst für q relativ zu E mit

$$\forall\, p.\ E(p) =_{df} \models_{pk} \{p\}\ \pi\ \{q\}.$$

Schwächste Vorbedingungen

...im Kontext mit totaler Korrektheit.

Definition 4.2.2.5 (Schwächste Vorbedingung)

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

$$[wp(\pi, q)]\ \pi\ [q]$$

total korrekt ist und $wp(\pi, q)$ die schwächste Formel oder Prädikat mit dieser Eigenschaft ist.

...d.h. $wp(\pi, q)$ ist schwächst für q relativ zu E mit

$$\forall\, p.\ E(p) =_{df} \models_{tk} [p]\ \pi\ [q].$$

Stärkste Nachbedingungen

...im Kontext partieller Korrektheit.

Definition 4.2.2.6 ((Partiell) stärkste Nachbeding.)

Sei π ein Programm, p eine Formel oder Prädikat. Dann heißt $sp_{pk}(p, \pi)$ (partiell) stärkste Nachbedingung von π bezüglich Vorbedingung p, wenn

$$\{p\}\ \pi\ \{sp(p, \pi)\}$$

partiell korrekt ist und $sp_{pk}(p, \pi)$ die stärkste Formel mit dieser Eigenschaft ist.

...d.h. $sp_{pk}(p, \pi)$ ist stärkst für p relativ zu E mit

$$\forall\, q.\ E(q) =_{df} \models_{pk} \{p\}\ \pi\ \{q\}.$$

Stärkste Nachbedingungen

...im Kontext totaler Korrektheit.

Definition 4.2.2.7 ((Total) stärkste Nachbedingung)

Sei π ein Programm, p eine Formel oder Prädikat. Dann heißt $sp_{tk}(p, \pi)$ (total) stärkste Nachbedingung von π bezüglich Vorbedingung p, wenn

$$[p]\ \pi\ [sp(p, \pi)]$$

total korrekt ist und $sp_{tk}(p, \pi)$ die stärkste Formel mit dieser Eigenschaft ist.

...d.h. $sp_{tk}(p, \pi)$ ist stärkst für p relativ zu E mit

$$\forall\, q.\ E(q) =_{df} \models_{tk} [p]\ \pi\ [q].$$

Stärkste Nach-, schwächste Vorbedingungen

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

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

Lemma 4.2.2.8 (Charakterisierungsmengen)

1. $[\![\pi]\!](Ch(p))$ ist die Charakterisierungsmenge der (partiell und total) stärksten Nachbedingung von π bezüglich p.
2. $[\![\pi]\!]^{-1}(Ch(q))$ ist die Charakterisierungsmenge der schwächsten Vorbedingung von π bezüglich q, wobei $[\![\pi]\!]^{-1}(\Sigma') =_{df} \{\sigma \in \Sigma \mid [\![\pi]\!](\sigma) \in \Sigma'\}$.
3. $[\![\pi]\!]^{-1}(Ch(q)) \cup \mathcal{C}(Def([\![\pi]\!]))$ ist die Charakterisierungsmenge der schwächsten liberalen Vorbedingung von π bezüglich q, wobei $\mathcal{C}$ den Mengenkomplementoperator (bzgl. Grundmenge Σ) bezeichnet: $\forall \Sigma' \subseteq \Sigma.\ \mathcal{C}(\Sigma') =_{df} \Sigma \backslash \Sigma'$.

Bemerkung

Die Definitionen

- schwächster, schwächster liberaler Vorbedingungen
- partiell, total 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 4.2.2.8) 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. 4.4 zur Vollständigkeit intensionaler Ableitungskalküle.

Zusammenhang

...stärkster Nach- und schwächster Vorbedingungen:

Lemma 4.2.2.9

Ist $[\![\pi]\!]$ total definiert, d.h. $Def([\![\pi]\!]) = \Sigma$, dann gilt für alle p, q Formeln oder Prädikate:

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

Beweis: Übungsaufgabe.

Kapitel 4.2.3

Korrektheit, Vollständigkeit von Ableitungskalkülen

Korrektheits- und Vollständigkeitsfrage

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

Korrektheitsfrage:

- Ist jede mithilfe von K ableitbare (oder 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 beweisbar)?

Korrektheit und Vollständigkeit eines Kalküls

...für partielle Korrektheit.

Definition 4.2.3.1 (Korrektheit und Vollständigkeit)

Ein Beweiskalkül K_{pk} für partielle Korrektheit heißt

- 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.:

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

- 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.:

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

Korrektheit und Vollständigkeit eines Kalküls

...für totale Korrektheit.

Definition 4.2.3.2 (Korrektheit und Vollständigkeit)

Ein Beweiskalkül K_{tk} für totale Korrektheit heißt

- 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.:

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

- 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.:

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

Kapitel 4.3

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.

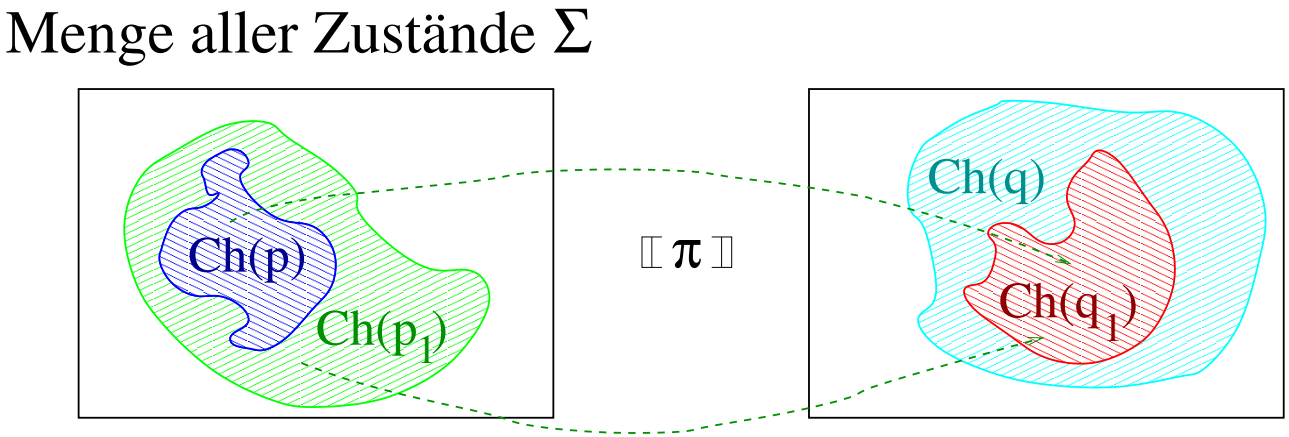
Sie erlaubt

- 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:



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

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

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:

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

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

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 Vorwärtszuweisungsregel(n)

Die Vorwärtsregel für die Zuweisung

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

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

Beachte: Folgende scheinbar naheliegende quantorfreie Variante einer Vorwärtszuweisungsregel ist nicht korrekt:

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

Beweis: Übungsaufgabe.

Kapitel 4.4

Korrektheit und Vollständigkeit von HK_{pk}

Korrektheit von HK_{pk}

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

Theorem 4.4.1 (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.:

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

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 4.4.2 (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.:

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

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 4.4.2 nicht.

Wichtig ist folgende Beobachtung:

Lemma 4.4.3

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 4.4.3(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 4.4.2 für den intensionalen Ansatz mit **Bexpr** als Zusicherungssprache und $[\![\ ]\!]_B$ als Semantik.

Zu Lem. 4.4.3(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 4.2.2.4 und 4.2.1.1 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. 4.4.3(2): Halteproblemreduktion (2)

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

$$\forall \sigma \in \Sigma.\ [\![f_\pi]\!]_B(\sigma) = \textbf{wahr} \iff$$

π terminiert nicht regulär angesetzt auf σ $\quad (b)$

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

$$\forall \sigma \in \Sigma.\ [\![\neg f_\pi]\!]_B(\sigma) = \textbf{wahr} \iff$$

π terminiert regulär angesetzt auf σ $\quad (c)$

Aus (b), (c) folgt zusammen mit (a) und Definition 4.2.2.4:

$$Ch(f_\pi) = Ch(wlp(\pi, false))$$

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

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

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 4.4.3(2)?

Kapitel 4.5

Partielle Korrektheitsbeweise

Kapitel 4.5.1

Beispiele: Fakultät und Division mit Rest

Fakultäts- und Divisionsprogramm

Lemma 4.5.1.1 (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 4.5.1.2 (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: Beweise Lemma 4.5.1.1 und 4.5.1.2.

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

Wir zeigen die Beweise in zwei notationellen Varianten. Als

- Ableitungsbaum (kanonische Variante).
- lineare Beweisskizze (pragmatische Variante).

Kapitel 4.5.2

Ableitungsbäume

Fakultätsprogramm: Festlegung der Invariante

Schritt 1:

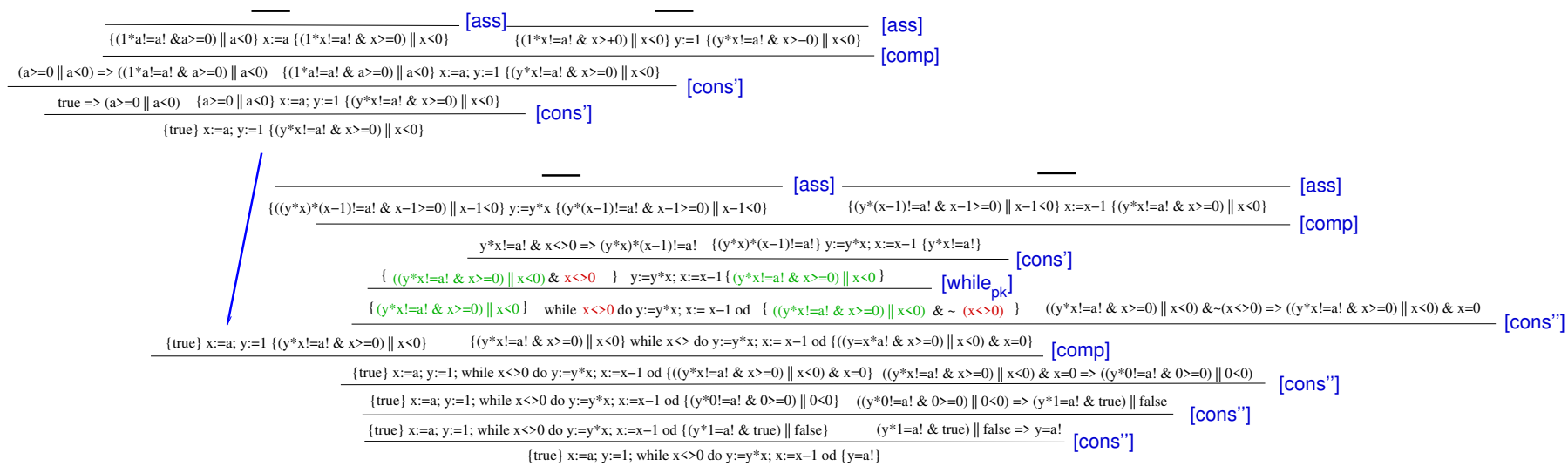
"Träumen" von

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

...als (geeignete) Invariante, um die Regel $[\text{while}_{pk}]$ anwenden und den Beweis erfolgreich abschließen zu können.

Fakultätsprogramm: Ableitungsbaum

Schritt 2: Angabe des Ableitungsbaums



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

Fakultätsprogramm: Zusammenfassung

Durch Angabe eines 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 4.4.1 ist die Zusicherung damit gültig im Sinn partieller Korrektheit. Damit ist der Beweis von Lemma 4.5.1.1 erbracht.

Divisionsprogramm: Festlegung der Invariante

Schritt 1:

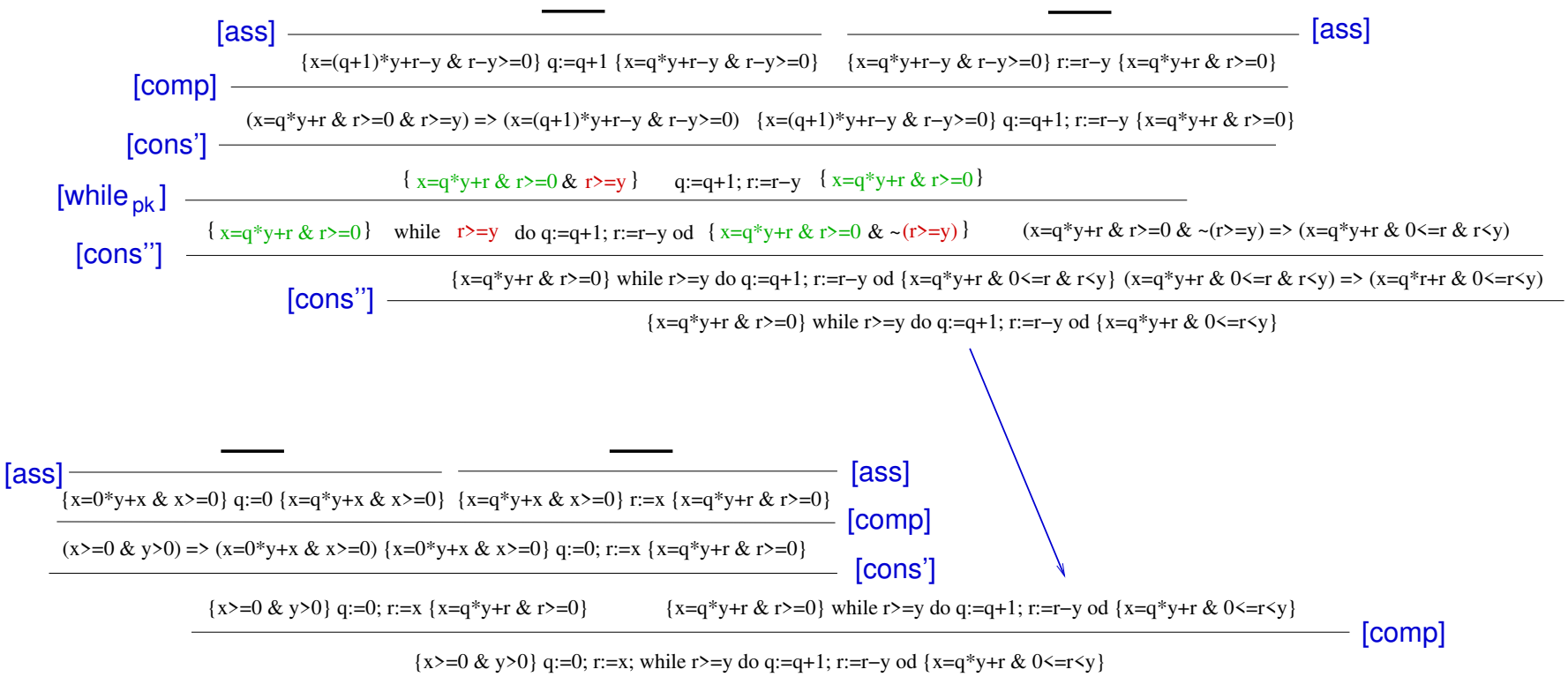
"Träumen" von

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

...als (geeignete) Invariante, um die Regel $[\text{while}_{pk}]$ anwenden und den Beweis erfolgreich abschließen zu können.

Divisionsprogramm: Ableitungsbaum

Schritt 2: Angabe des Ableitungsbaums



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

Divisionsprogramm: Zusammenfassung

Durch Angabe eines 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 4.4.1 ist die Zusicherung damit gültig im Sinn partieller Korrektheit. Damit ist der Beweis von Lemma 4.5.1.2 erbracht.

Ableitungsbäume vs. lineare Beweisskizzen

Die von den Kalkülregeln induzierte 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 in den Programmtext Zusicherungen als Annotationen eingestreut werden.

Man spricht von sog.

- linearen Beweisen oder linearen Beweisskizzen.

Vorteil

...des linearen gegenüber des baumartigen Notationsstils:

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

In der Folge

- demonstrieren wir diesen Notationsstil am Beispiel des Beweises von Lemma 4.5.1.1 zum Nachweis der partiellen Korrektheit des Fakultätsprogramms bezüglich des angegebenen Paars von Vor- und Nachbedingung.

Kapitel 4.5.3

Lineare Beweisskizzen

Lemma 4.5.1.1: Fakultätsprogramm

...Beweis von Lemma 4.5.1.1:

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.

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

Fakultätsprogramm: Festlegung der Invariante

Schritt 1:

"Träumen" von

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

...als (geeignete) Invariante, um die Regel $[\text{while}_{pk}]$ anwenden und den Beweis erfolgreich abschließen zu können.

Fakultätsprogramm: Lineare Beweisskizze (1)

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:

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

while $x \neq 0$ do

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

$$y := y * x;$$

$$x := x - 1;$$

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

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

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

Fakultätsprogramm: Lineare Beweisskizze (2)

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 (3)

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 (4)

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 (5)

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 (6)

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 (7)

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 (8)

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

$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
while $x \neq 0$ do
$$\{((y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0) \ \wedge \ x \neq 0\}$$
$$\Downarrow \ [\text{cons}']$$
$$\{((y * x) * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$
$y := y * x$; [ass]
$$\{(y * (x - 1)! = a! \ \wedge \ x - 1 \geq 0) \ \vee \ x - 1 < 0\}$$
$x := x - 1$; [ass]
$$\{(y * x! = a! \ \wedge \ x \geq 0) \ \vee \ x < 0\}$$
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 (9)

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 (10)

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 (11)

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 (12)

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 (13)

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 Angabe einer 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 4.4.1 ist die Zusicherung damit gültig im Sinn partieller Korrektheit. Damit ist der Beweis von Lemma 4.5.1.1 erbracht.

Übungsaufgabe 1: Fakultätsprogramm

Zeige durch Angabe

- eines Ableitungsbaums
- einer linearen 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. zeige

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

Lässt sich die Invariante aus dem Beweis partieller Korrektheit zur Vorbedingung *true* für den Beweis zur Vorbedingung $a \geq 0$ abschwächen? Begründe die Antwort.

Übungsaufgabe 2: Divisionsprogramm

...ganzzahlige Division mit Rest.

Beweise Lemma 4.5.1.2.

Zeige durch Angabe 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. zeige

$$\models_{pk} \{x \geq 0 \wedge y > 0\}\ \pi\ \{x = q * y + r \wedge 0 \leq r < y\}$$

Übungsaufgabe 3: Divisionsprogramm (1)

Überlege, ob $x \geq 0 \wedge y > 0$ 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 die Nachbedingung

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

ist?

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 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. zeige

$$\models_{pk} \{wlp\}\ \pi\ \{x = q * y + r \wedge 0 \leq r < y\}$$

durch Angabe

1. eines Ableitungsbaums.
2. einer linearen 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?

Beweise die zusätzlichen Eigenschaften.

Kapitel 4.6

Ableitungskalküle HK'_{tk}, HK_{tk} für totale Korrektheit

Die Hoare-Kalküle

...für partielle und totale Korrektheit für WHILE sind nahezu

- identisch.

Einziger Unterschied: Die Regel $[\text{while}_{pk}]$ zur Behandlung der

- while-Schleife

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 der

- Regel (Variante V1).
- Regelanwendung (Variante V2).

V1: Hoare-Kalkül HK_{tk} für totale Korrektheit

Variante 1: Regeleinfachheit vor Regelanwendungseinfachheit

$$[\text{while}'_{tk}] \quad \frac{I \Rightarrow t{\geq}0,\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]} \qquad (I \text{ Invariante})$$

wobei

- 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: $(\mathbb{N}, \textit{kleiner})$ (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 Element der durch die Relation *kleiner* $(<)$ 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 des Schleifenrumpfs 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 *kleiner* $(<)$ 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, woraus die Terminierung der while-Schleife folgt.

V2: Hoare-Kalkül HK_{tk} für totale Korrektheit

Variante 2: Regelanwendungseinfachheit vor Regeleinfachheit

$$[\text{while}_{tk}] \quad \frac{I \wedge b \Rightarrow u[t/v],\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [I \wedge \neg b]} \qquad (I \text{ Invariante})$$

wobei

- 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, dass $u[t/v]$ wahr ist.

Zusammen mit der Definition von M folgt daraus, dass der Wert des Terminierungsterms t vor jeder Ausführung des Schleifenrumpfs 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 des Schleifenrumpfs 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] $$\frac{}{[p]\ \mathit{skip}\ [p]}$$

[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_{tk}] $$\frac{I \wedge b \Rightarrow u[t/v],\ [I \wedge b \wedge t{=}w]\ \pi\ [I \wedge t{<}w]}{[I]\ \text{while } b \text{ do } \pi \text{ od } [I \wedge \neg b]}$$

(I Invariante)

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

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 bietet $[\text{while}_{tk}]$ pragmatische Vorteile im Vergleich zu $[\text{while}'_{tk}]$.

Irreflexive partielle Ordnungen

Definition 4.6.1 (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 $<$, $>$).

Wohlfundierte Ordnungen

Definition 4.6.2 (Wohlfundierte Ordnung)

Sei $(P, \sqsubset)$ eine irreflexive partielle Ordnung und W eine Teilmenge von P.

- $\sqsubset$ 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.
- 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 $(\mathbb{N}, >)$.

Konstruktion wohlfundierter Ordnungen

...aus gegebenen wohlfundierten Ordnungen:

Lemma 4.6.3

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 identisch:

- 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, dass der Beweis totaler Korrektheit einer Hoareschen Zusicherung besteht aus dem Nachweis

- partieller Korrektheit.
- Regulärer Terminierung des Programms.

Totale Korrektheit "gleich"
Partielle Korrektheit "plus" Reguläre Terminierung

Diese Trennung kann im Beweis explizit vollzogen werden. Der Gesamtbeweis wird dadurch modular, wobei der Terminationsbeweis zudem oft einfach ist.

Bemerkung: Die obige Zerlegung kann in gleicher Weise für die Schleifenregel $[\text{while}_{tk}]$ erfolgen.

Kapitel 4.7

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 4.7.1 (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 4.7.2 (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 4.8

Totale Korrektheitsbeweise

Kapitel 4.8.1

Beispiele: Fakultät und Division mit Rest

Fakultäts- und Divisionsprogramm

Lemma 4.8.1.1 (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 4.8.1.2 (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: Beweise Lemma 4.8.1.1 und 4.8.1.2.

...d.h., zeige, dass die Hoareschen Tripel für die Berechung der Fakultätsfunktion und der ganzzahligen Division mit Rest gültig sind im Sinn totaler Korrektheit.

Führe die Beweise in zwei notationellen Varianten. Als

- Ableitungsbaum (kanonische Variante).
- lineare Beweisskizze (pragmatische Variante).

Kapitel 4.8.2

Ableitungsbäume

Übungsaufgabe

Beweise die Gültigkeit der Hoareschen Zusicherungen aus

- Lemma 4.8.1.1 (Fakultät)
- Lemma 4.8.1.2 (Division mit Rest)

mithilfe der Axiome und Regeln von

1. HK'_{tk}
2. HK_{tk}

durch Ableitung und Angabe entsprechender Ableitungsbäume.

Kapitel 4.8.3

Lineare Beweisskizzen

Lemma 4.8.1.1: Fakultätsprogramm

...Beweis von Lemma 4.8.1.1:

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.

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

Fakultätsprogramm: Invariante, Term.-Term

Schritt 1:

"Träumen" von

- $I \equiv y * x! = a! \wedge x \geq 0$ als Invariante
- $t \equiv x$ als Terminierungsterm
- $u \equiv v > 0$ als Boolescher Ausdruck über v

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

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:

$$
\begin{aligned}
M &= \\
&=_{df} \{\, \sigma(v) \mid \sigma \in Ch(u) \,\} \\
&= \{\, \sigma(v) \mid \sigma \in Ch(v > 0) \,\} \\
&= \{\, \sigma(v) \mid \sigma \in \Sigma \wedge \mathit{größer}([\![\, v \,]\!]_A(\sigma), [\![\, 0 \,]\!]_A(\sigma)) \,\} \\
&= \{\, \sigma(v) \mid \sigma \in \Sigma \wedge \mathit{größer}(\sigma(v), \mathbf{0}) \,\} \\
&= \mathbb{N}_1
\end{aligned}
$$

Damit gilt: M ist bzgl. der Relation *größer* auf $\mathbb{N}_1$ Noethersch geordnet, d.h.

$$(M, \sqsubseteq) = (\mathbb{N}_1, \mathit{größer})$$

ist Noethersche Ordnung.

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 (1)

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 (2)

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 (3)

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 (4)

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 (5)

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 (6)

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

$$[y * x! = a! \wedge x \geq 0]$$

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 [while$_{tk}$]

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

Fakultätsprogramm: Lineare Beweisskizze (7)

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 (8)

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 (9)

Aus Platzgründen etwas verkürzt dargestellt:

$$[y * x! = a! \wedge x \geq 0]$$

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

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$

$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

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

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

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

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

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

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

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

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow 3x [\text{cons}'']$$

$$[y = a!]$$

Fakultätsprogramm: Lineare Beweisskizze (10)

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 (11)

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 (12)

Nochmalige Anwendung der [ass]-Regel liefert:

$$[a \geq 0]$$

$$[1 * a! = a! \wedge a \geq 0]$$

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

$$[1 * x! = a! \wedge x \geq 0]$$

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

$$[y * x! = a! \wedge x \geq 0]$$

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

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

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

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

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

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

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

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

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

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

$$\Downarrow \text{ 3x [cons}''\text{]}$$

$$[y = a!]$$

Fakultätsprogramm: Lineare Beweisskizze (13)

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

$$[a \geq 0]$$

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

$$[1 * a! = a! \wedge a \geq 0]$$

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

$$[1 * x! = a! \wedge x \geq 0]$$

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

$$[y * x! = a! \wedge x \geq 0]$$

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

$$y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x > 0$$
$$[y * x! = a! \wedge x \geq 0 \wedge x \neq 0 \wedge x = w]$$

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

$$[(y * x) * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

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

$$[y * (x - 1)! = a! \wedge x - 1 \geq 0 \wedge x - 1 < w]$$

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

$$[y * x! = a! \wedge x \geq 0 \wedge x < w]$$

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

$$[y * x! = a! \wedge x \geq 0 \wedge \neg(x \neq 0)]$$

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

$$[y * x! = a! \wedge x \geq 0 \wedge x = 0]$$

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

$$[y * 0! = a!]$$

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

$$[y = a!]$$

Zusammenfassung

Durch Angabe einer 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 4.7.1 ist die Zusicherung damit gültig im Sinn totaler Korrektheit. Damit ist der Beweis von Lemma 4.8.1.1 erbracht.

Übungsaufgabe

...ganzzahlige Division mit Rest.

Beweise Lemma 4.8.1.2.

Zeige durch Angabe 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.

Kapitel 4.9

Ansätze und 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 4.10

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 4.11

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`

Charles A.R. Hoare. *An Axiomatic Basis for Computer Programming*. Communications of the ACM 12(10):576-580, 583, 1969.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (7)

Charles A.R. Hoare. *The Ideal of Program Correctness*. The Computer Journal 50(3):254-260, 2007.

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`

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (8)

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`

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (9)


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)


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

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (10)

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)

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 6, Axiomatic Program Verification)

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)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (12)

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.

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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (13)


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.

Kapitel 5

Axiomatische Ausführungszeitanalyse

Kapitel 5.1

Motivation

Übergang

...von axiomatischer Programmverifikation zu axiomatischer Programmanalyse.

...am Beispiel

- axiomatischer asymptotischer Ausführungszeitanalyse

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.

Ausführungszeitanalyse

Hintergrund: In vielen Anwendungsbereichen sind

- (zumindest weiche) Aussagen über die Ausführungszeit von Programmen erforderlich, z.B. Antwortzeiten von Buchungsportalen.
- sogar harte Aussagen über die Ausführungs- bzw. Antwortzeiten von Programmen erforderlich, besonders für sicherheitskritische Echtzeitanwendungen (sog. Schlechtester-Fall-Ausführungszeitanalyse (engl. worst-case execution time (WCET) analysis).

Der Nachweis totaler Korrektheit mittels axiomatischer Programmverifikation garantiert zwar

- Terminierung eines Programms

sagt aber nichts über den tatsächlichen Ressourcen-, insbesondere Laufzeitbedarf aus.

In diesem Kapitel

...Erweiterung und Adaptierung des Ableitungskalküls für totale Korrektheit, um Aussagen über den

- asymptotischen Laufzeitbedarf

zu ermöglichen.

Grundidee (1)

...Zuordnung von Auswertungszeiten zu Ausdrücken:

- Numerale, Wahrheitswertkonstanten
  ...Auswertungszeit in konstanter Zeit, d.h. in Größenordnung $\mathcal{O}(\mathbf{1})$.
- Variablen
  ...Auswertungs- bzw. Zugriffszeit (lesen, schreiben) in konstanter Zeit, d.h. in Größenordnung $\mathcal{O}(\mathbf{1})$.
- Zusammengesetzte Ausdrücke
  ...Auswertungszeit in linearer Zeit abhängig von der Zahl **n** der Operatoren und Relatoren im Ausdruck, d.h. in Größenordnung $\mathcal{O}(\mathbf{n})$.

Grundidee (2)

...Zuordnung von Ausführungszeiten zu Programmkonstrukten:

- Leere Anweisung
  ...Ausführungszeit in konstanter Zeit, d.h. in Größenordnung $\mathcal{O}(\mathbf{1})$.
- Zuweisung
  ...Ausführungszeit in linearer Zeit in Größenordnung der Auswertungszeit des rechtsseitigen Ausdrucks.
- (Sequentielle) Komposition
  ...Ausführungszeit in Größenordnung (d.h. gleich bis auf einen konstanten Faktor) der Summe der Ausführungszeiten der Komponenten.

Grundidee (3)

- Fallunterscheidung
  ...Ausführungszeit in Größenordnung der Summe der Auswertungszeit der Bedingung und der größeren der Ausführungszeiten der beiden Zweige.
- while-Schleife
  ...Ausführungszeit in Größenordnung der Summe der wiederholten Auswertungszeiten der Abbruchbedingung und Ausführungszeiten 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 (Kapitel 5.2).
2. Ausführungszeitbewusste (abstrakte) Programmsemantik:
   ...Erweiterung und Adaption der natürlichen Semantik von WHILE zu einer ausführungszeitbewussten abstrakten Programmsemantik (Kapitel 5.3).
3. Ableitungskalkül für totale Korrektheit mit asymptotischen Ausführungszeitaussagen:
   Erweiterung und Adaption des Ableitungskalküls für totale Korrektheit zur Ableitung von Aussagen zur asymptotischen Ausführungszeit von Programmen (Kapitel 5.4).

Kapitel 5.2

Zeitbewusste Ausdruckssemantik

Zeitbewusste Ausdruckssemantik

...Einführung und Definition zeitbewusster (abstrakter) Semantikfunktionen für arithmetische und Boolesche Ausdrücke:

- $[\![.]\!]_{ZA} : \mathbf{Aexpr} \to \mathbb{Z}$
- $[\![.]\!]_{ZB} : \mathbf{Bexpr} \to \mathbb{Z}$

die als Bedeutung arithmetischen und Booleschen Ausdrücken ihre Auswertungszeit (in Zeiteinheiten einer hier nicht näher spezifizierten abstrakten Maschine *AM*) geben.

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 und b benötigt.

Zeitbewusste Semantik arithmet. Ausdrücke

$[\![\, . \,]\!]_{ZA} : \mathbf{Aexpr} \to \mathbb{Z}$ (zustandsunabhängig) 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.

Zeitbewusste Semantik Boolescher Ausdrücke

$[\![.]\!]_{ZB} : \mathbf{Bexpr} \to \mathbb{Z}$ (zustandsunabhängig) 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.

Kapitel 5.3

Zeitbewusste natürliche Semantik

Erweiterung und Anpassung

...der

- natürlichen Semantik $[\![\ ]\!]_{ns}$ von WHILE

zur Bestimmung der

- Ausführungszeit 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

$$[\text{skip}_{nzs}] \quad \frac{}{\langle \mathit{skip}, \sigma \rangle \ \to^{1} \ \sigma}$$

$$[\text{ass}_{nzs}] \quad \frac{}{\langle x := t, \sigma \rangle \ \to^{[\![t]\!]_{ZA}+1} \ \sigma[[\![t]\!]_A(\sigma)/x]}$$

$$[\text{while}^{ff}_{nzs}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \ \to^{[\![b]\!]_{ZB}+3} \ \sigma} \qquad [\![b]\!]_B(\sigma) = \textbf{falsch}$$

NZS-Regelwerk von WHILE: Regeln

$$[\text{while}^{tt}_{nzs}] \quad \frac{\langle \pi, \sigma \rangle \to^{t} \sigma', \langle \text{while } b \text{ do } \pi \text{ od}, \sigma' \rangle \to^{t'} \sigma''}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \to^{[\![b]\!]_{ZB}+t+t'+2} \sigma''} [\![b]\!]_B(\sigma) = \textbf{wahr}$$

$$[\text{if}^{tt}_{nzs}] \quad \frac{\langle \pi_1, \sigma \rangle \to^{t} \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \to^{[\![b]\!]_{ZB}+t+1} \sigma'} [\![b]\!]_B(\sigma) = \textbf{wahr}$$

$$[\text{if}^{ff}_{nzs}] \quad \frac{\langle \pi_2, \sigma \rangle \to^{t} \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \to^{[\![b]\!]_{ZB}+t+1} \sigma'} [\![b]\!]_B(\sigma) = \textbf{falsch}$$

$$[\text{comp}_{nzs}] \quad \frac{\langle \pi_1, \sigma \rangle \to^{t_1} \sigma', \langle \pi_2, \sigma' \rangle \to^{t_2} \sigma''}{\langle \pi_1; \pi_2, \sigma \rangle \to^{t_1+t_2} \sigma''}$$

Beispiel: Illustration der NZ-Semantik (1)

...anhand des (kanonischen) Fakultätsprogramms.

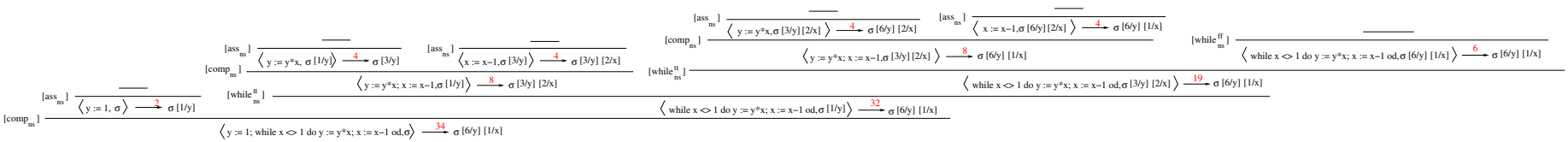
Sei $\sigma \in \Sigma$ mit $\sigma(x) = \mathbf{3}$.

Dann gilt:

$$\langle y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma \rangle \rightarrow^{34} \sigma[\mathbf{6}/y][\mathbf{1}/x]$$

...terminiert in 34 Zeiteinheiten im Zustand $\sigma[\mathbf{6}/y][\mathbf{1}/x]$.

Zugehöriger Ableitungsbaum:



Beispiel: Illustration der NZ-Semantik (2)

...der gleiche Ableitungsbaum in "etwas" größerer Darstellung durch Einführung eines benannten Teilbaums T:

$$\dfrac{\dfrac{}{\langle y := 1, \sigma\rangle \xrightarrow{2} \sigma\,[1/y]}[\mathrm{ass}_{ns}] \qquad \dfrac{\dfrac{\dfrac{}{\langle y := y*x, \sigma\,[1/y]\rangle \xrightarrow{4} \sigma\,[3/y]}[\mathrm{ass}_{ns}] \qquad \dfrac{}{\langle x := x-1, \sigma\,[3/y]\rangle \xrightarrow{4} \sigma\,[3/y]\,[2/x]}[\mathrm{ass}_{ns}]}{\langle y := y*x;\ x := x-1, \sigma\,[1/y]\rangle \xrightarrow{8} \sigma\,[3/y]\,[2/x]}[\mathrm{comp}_{ns}] \qquad T\ ^{19}}{\langle \text{while } x <> 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\,[1/y]\rangle \xrightarrow{32} \sigma\,[6/y]\,[1/x]}[\mathrm{while}^{tt}_{ns}]}{\langle y := 1;\ \text{while } x <> 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\rangle \xrightarrow{34} \sigma\,[6/y]\,[1/x]}[\mathrm{comp}_{ns}]$$

$T \equiv$

$$\dfrac{\dfrac{\dfrac{}{\langle y := y*x, \sigma\,[3/y]\,[2/x]\rangle \xrightarrow{4} \sigma\,[6/y]\,[2/x]}[\mathrm{ass}_{ns}] \qquad \dfrac{}{\langle x := x-1, \sigma\,[6/y]\,[2/x]\rangle \xrightarrow{4} \sigma\,[6/y]\,[1/x]}[\mathrm{ass}_{ns}]}{\langle y := y*x;\ x := x-1, \sigma\,[3/y]\,[2/x]\rangle \xrightarrow{8} \sigma\,[6/y]\,[1/x]}[\mathrm{comp}_{ns}] \qquad \dfrac{}{\langle \text{while } x <> 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\,[6/y]\,[1/x]\rangle \xrightarrow{6} \sigma\,[6/y]\,[1/x]}[\mathrm{while}^{ff}_{ns}]}{\langle \text{while } x <> 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\,[3/y]\,[2/x]\rangle \xrightarrow{19} \sigma\,[6/y]\,[1/x]}[\mathrm{while}^{tt}_{ns}]$$

Kapitel 5.4

Zeitbewusste axiomatische Semantik

Erweiterung und Anpassung

...des

- Ableitungskalküls HK_{tk} für totale Korrektheit

von Programmen um den Aspekt ihrer asymptotischen Ausführungszeit.

Methode: Übergang von zeitunbewussten Korrektheitsformeln der Form

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

zu zeitbewussten Korrektheitsformeln der Form

$$[p]\ \pi\ [e \Downarrow q]$$

wobei

- π WHILE-Programm.
- p, q logische Formeln oder Prädikate als Vor- und Nachbedingung (wie bisher!).
- $e \in \mathbf{Aexp}$ arithmetischer Ausdruck als Ausführungszeitabschätzung.

Semantik zeitbewusster Korrektheitsformeln

Sei π ein WHILE-Programm, p, q zwei logische Formeln oder Prädikate, e ein arithmetischer Ausdruck.

Definition 5.1.1 (Gültigkeit zeitbew. Korrektheitsf.)

Die zeitbewusste Korrektheitsformel

$$[p]\ \pi\ [e \Downarrow q]$$

ist gültig im Sinn zeitbewusster 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 die benötigte Ausführungszeit von π ist durch e beschränkt, d.h. von der Grössenordnung $\mathcal{O}(e)$.

Charakterisierung zeitbew. totaler Korrektheit

Lemma 5.1.2 (Charakterisierung)

Die zeitbewusste Korrektheitsformel

$$[p]\ \pi\ [e \Downarrow q]$$

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).

Beachte

..der Ausdruck e zur Größenordnungsabschätzung wird im Anfangszustand σ ausgewertet, nicht im Endzustand σ':

- $t \leq \mathbf{k} * [\![e]\!]_A(\sigma)$

Diesem sinnvollen Umstand (Übungsaufgabe: Warum?) ist geschuldet, dass die Festlegung der Regeln

- $[while_e]$ und $[comp_e]$

des Hoare-artigen Ausführungszeitableitungskalküls AK_{ztk} komplizierter ausfällt als möglicherweise zunächst vermutet.

Laufzeitabschätzungskalkül AK_{ztk}: Axiome

$$[\text{skip}_e] \quad \frac{}{\{p\}\ \mathit{skip}\ \{\mathbf{1} \Downarrow p\}}$$

$$[\text{ass}_e] \quad \frac{}{\{p[t\backslash x]\}\ x{:=}t\ \{\mathbf{1} \Downarrow p\}}$$

Laufzeitabschätzungskalkül AK_{ztk}: Regeln

$$[\text{comp}_e] \quad \frac{[p \wedge e_2' = u]\ \pi_1\ [e_1 \Downarrow r \wedge e_2 \leq u],\ [r]\ \pi_2\ [e_2 \Downarrow q]}{[p]\ \pi_1;\pi_2\ [e_1 + e_2' \Downarrow q]}$$

wobei u frische logische Variable.

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

$$[\text{while}_e] \quad \frac{[p(z+1) \wedge e' = u]\ \pi\ [e_1 \Downarrow p(z) \wedge e \leq u]}{[\exists z.\ p(z)]\ \text{while}\ b\ \text{do}\ \pi\ \text{od}\ [e \Downarrow p(0)]}$$

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.

$$[\text{cons}_e] \quad \frac{p \Rightarrow p_1 \quad [p_1]\ \pi\ [e' \Downarrow q_1] \quad q_1 \Rightarrow q}{[p]\ \pi\ [e \Downarrow q]} \qquad \exists\, \mathbf{k} \in \mathbb{N}.\ e' \leq \mathbf{k} * e$$

Anmerkungen zur AK_{ztk}-Kalkülregel [$comp_e$]

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 erzwungen.

Anmerkungen zur AK_{ztk}-Kalkülregel [$while_e$]

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}-Kalkülregel $[\text{cons}_e]$

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.

Beispiele: Fakultätsprogramm (1)

1) Die zeitbewusste Korrektheitsformel

$$[a = 3]$$
$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$[1 \Downarrow \mathit{true}]$$

ist gültig im Sinn zeitbewusster totaler Korrektheit und beschreibt, dass die Zahl der Ausführungsschritte des Fakultätsprogramms angesetzt auf einen Zustand σ mit $\sigma(a) = \mathbf{3}$ durch $\mathcal{O}(\mathbf{1})$ beschränkt ist, also durch eine Konstante.

Beispiele: Fakultätsprogramm (2)

2) Die zeitbewusste Korrektheitsformel

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

ist gültig im Sinn zeitbewusster totaler Korrektheit und beschreibt, dass die Zahl der Ausführungsschritte des Fakultätsprogramms angesetzt auf einen Zustand σ mit $\sigma(x) > \mathbf{0}$ durch $\mathcal{O}(a)$ beschränkt ist, also linear in der Größe von a und damit des Anfangswerts von x ist.

Kapitel 5.5

Zeitbewusste totale Korrektheitsbeweise

Fakultätsprogramm variiert

...Terminierung plus Terminierungsabschätzung:

Lemma 5.5.1 (Fakultät)

1. Die zeitbewusste Korrektheitsformel

$$[x = 3]$$
$$y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$[1 \Downarrow \mathit{true}]$$

   ist gültig im Sinn zeitbewusster totaler Korrektheit.

2. Die zeitbewusste Korrektheitsformel

$$[x > 0]$$
$$y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x\ x := x - 1 \text{ od}$$
$$[a \Downarrow \mathit{true}]$$

   ist gültig im Sinn zeitbewusster totaler Korrektheit.

Lemma 5.5.1(2): Lineare Beweisskizze (1)

$$[x > 0]$$
$$\vdots$$
$$y := 1;$$
$$\vdots$$
$$[\exists\, z \in \mathbb{N}_0.\ (x > 0 \wedge x = z + 1)] \quad (\equiv\ [\exists\, z \in \mathbb{N}_0.\ \mathit{INV}(z)])$$
$$\text{while } x \neq 1 \text{ do}$$
$$[(x > 0 \wedge x = z + 1) \wedge x - 1 = u_1] \quad (\equiv\ [\mathit{INV}(z+1) \wedge x - 1 = u_1])$$
$$[(x > 0 \wedge x = (z + 1) + 1) \wedge x - 1 = u_1]$$
$$[((x > 0 \wedge x = (z + 1) + 1) \wedge x - 1 = u_1) \wedge 1 = u_2]$$
$$\Downarrow [\mathrm{cons}'_e]$$
$$[((x - 1 > 0 \wedge x - 1 = z + 1) \wedge x - 1 \leq u_1) \wedge 1 \leq u_2]$$
$$y := y * x;\ [\mathrm{ass}_e],\ [\mathrm{comp}_e]$$
$$[1 \Downarrow ((x - 1 > 0 \wedge x - 1 = z + 1) \wedge x - 1 \leq u_1) \wedge 1 \leq u_2]$$
$$[(x - 1 > 0 \wedge x - 1 = z + 1) \wedge x - 1 \leq u_1]$$
$$x := x - 1;\ [\mathrm{ass}_e]$$
$$[1 \Downarrow (x > 0 \wedge x = z + 1) \wedge x \leq u_1]$$
$$[1 + 1 \Downarrow (x > 0 \wedge x = z + 1) \wedge x \leq u_1]$$
$$\Downarrow [\mathrm{cons}''_e]$$
$$[1 \Downarrow (x > 0 \wedge x = z + 1) \wedge x \leq u_1] \quad (\equiv\ [1 \Downarrow\ \mathit{INV}(z) \wedge x \leq u_1])$$
$$(x > 0 \wedge x = (z + 1) + 1) \Rightarrow \neg(x = 1) \wedge x \geq 1 + (x - 1)\ (\equiv \mathit{INV}(z+1) \Rightarrow \neg(x = 1) \wedge x \geq 1 + (x - 1))$$
$$(x > 0 \wedge x = 0 + 1) \Rightarrow \neg(\neg(x = 1)) \wedge 1 \leq x \quad (\equiv\ \mathit{INV}(0) \Rightarrow \neg(\neg(x = 1)) \wedge 1 \leq x)$$
$$\text{od } [\mathrm{while}_e]$$
$$[x \Downarrow (x > 0 \wedge x = 0 + 1)] \quad (\equiv\ [x \Downarrow\ \mathit{INV}(0)])$$
$$\vdots$$
$$[x \Downarrow \mathit{true}]$$

$\mathit{INV}(z) = x > 0 \wedge x = z + 1,\ z \in \mathbb{N}_0$ (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)

$$[x > 0]$$

$$[x > 0 \wedge 1 = u3]$$

$$\Downarrow [\mathrm{cons}'_e]$$

$$[\exists\, z\, \mathbb{N}_0.\ (x > 0 \wedge x = z + 1) \wedge 1 \leq u_3]$$

$$y := 1;\ [\mathrm{ass}_e],\ [\mathrm{comp}_e]$$

$$[1 \Downarrow \exists\, z \in \mathbb{N}_0.\ (x > 0 \wedge x = z + 1) \wedge 1 \leq u_3]$$

$$[\exists\, z \in \mathbb{N}_0.\ (x > 0 \wedge x = z + 1)] \quad (\equiv\ [\exists\, z \in \mathbb{N}_0.\ \mathit{INV}(z)])$$

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: Fakultätsprogramm (1)

...Terminierung plus Terminierungsabschätzung:

Lemma 5.5.2 (Fakultät)

1. Die zeitbewusste Korrektheitsformel

$$[a = 3]$$

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

$$[1 \Downarrow \mathit{true}]$$

ist gültig im Sinn zeitbewusster totaler Korrektheit.

2. Die zeitbewusste Korrektheitsformel

$$[a \geq 0]$$

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

$$[a \Downarrow \mathit{true}]$$

ist gültig im Sinn zeitbewusster totaler Korrektheit.

Übungsaufgabe: Fakultätsprogramm (2)

Beweise Terminierung(sabschätzung) plus funkt. Korrektheit:

Lemma 5.5.3 (Fakultät)

1. Die zeitbewusste Korrektheitsformel

$$[a = 3]$$
$$x := a;\ y := 1;\ \text{while } x \neq 0 \text{ do } y := y * x;\ x := x - 1 \text{ od}$$
$$[1 \Downarrow y = 6 = a!]$$

ist gültig im Sinn zeitbewusster totaler Korrektheit.

2. Die zeitbewusste Korrektheitsformel

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

ist gültig im Sinn zeitbewusster totaler Korrektheit.

Übungsaufgabe: Divisionsprogramm (1)

Beweise Terminierung plus Terminierungsabschätzung:

Lemma 5.5.4 (Ganzzahlige Division mit Rest)

1. Die zeitbewusste Korrektheitsformel

$$[x = 17 \wedge y = 4]$$

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

$$[1 \Downarrow \mathit{true}]$$

ist gültig (im Sinn zeitbewusster totaler Korrektheit).

2. Die zeitbewusste Korrektheitsformel

$$[x \geq 0 \wedge y > 0]$$

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

$$[x - y \Downarrow \mathit{true}]$$

ist gültig (im Sinn zeitbewusster totaler Korrektheit).

Übungsaufgabe: Divisionsprogramm (2)

Beweise Terminierung(sabschätzung) plus funkt. Korrektheit:

Lemma 5.5.5 (Ganzzahlige Division mit Rest)

1. Die zeitbewusste Korrektheitsformel

$$[x = 17 \wedge y = 4]$$

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

$$[1 \Downarrow x = q * y + r \wedge 0 \leq r < y \wedge q = 4 \wedge r = 1]$$

ist gültig (im Sinn zeitbewusster totaler Korrektheit).

2. Die zeitbewusste Korrektheitsformel

$$[x \geq 0 \wedge y > 0]$$

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

$$[x - y \Downarrow x = q * y + r \wedge 0 \leq r < y\}$$

ist gültig (im Sinn zeitbewusster totaler Korrektheit).

Kapitel 5.6

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (1)

Axiomatische asymptotische Ausführungszeitanalyse


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: Überblicksarbeit

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.

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 III

Analyse

Kapitel 6

Programmanalyse

Kapitel 6.1

Motivation

Motivation

...in Kapitel 2 und 3 haben wir uns mit verschiedenen Methoden zur Festlegung einer

- konkreten Semantik $[\![\ ]\!]_{\text{While}}$ der Sprache While

und darauf aufbauend in Kapitel 4 und 5 mit

- axiomatischer Verifikation

zum Beweis von Eigenschaften eines Programms relativ zur konkreten Semantik $[\![\ ]\!]_{\text{While}}$ beschäftigt.

In der Folge

...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 $[\![\ ]\!]_{\mathrm{While}}$ von While korrekt sein müssen.

Dabei gilt

...(fast) alle interessanten Eigenschaften eines Programms sind bezüglich der konkreten wie (auch vieler) abstrakter Programmsemantiken

- unentscheidbar.

Glücklicherweise: Einige interessante Eigenschaften sind

- entscheidbar und
- nützlich

und ermöglichen Verfahren zur

- manuellen/semiautomatischen Programmverifikation
- semiautomatischen/vollautomatischen Programmanalyse.

Kapitel 6.2

Ausblick

Besonders

...die Entwicklung von Analyse- und Verifikationsverfahren relativ zu einer abstrakten Programmsemantik kann sich dabei zunutze machen, die fast immer konfliktären Ziele von

- Performanz/Effizienz/Skalierbarkeit und
- Akkuratheit/Vollständigkeit

gegeneinander abzuwiegen und abzutauschen, solange die erzielten Resultate noch

- nützlich

sind.

Mit der Aufgabe, Verfahren und Methoden zu entwickeln, die im Spannungsdreieck von

- Vollständigkeit, Nützlichkeit u. Performanz/Skalierbarkeit

eine gute Balance bewahren, beginnt Informatik.

Analyse- und Verifikationsverfahren

...zur Programmanalyse gibt es (deshalb) in verschiedensten Zugängen und Ausprägungen:

- Datenflussanalyse (Kap. 7)
- Reverse Datenflussanalyse (Kap. 8)
- Parallele Datenflussanalyse (Kap. 9)
- Abstrakte Interpretation (Kap. 15)
- Modellprüfung (Kap. 16)
- Symbolische Analyse
- Konkolische Analyse
- ...

von denen wir einige beginnend mit der sog. Datenflussanalyse in den folgenden Kapiteln genauer betrachten werden...

Anwendungen und Wechselbeziehungen

...exemplarisch auch hinsichtlich ihrer Nützlichkeit für Anwendungen am Beispiel der:

- Elimination unnötiger Anweisungen in Programmen (Kap. 12 und 13)
- Ersetzung von Ausdrücken durch ihre Werte (Kap. 14)

sowie ihrer Gemeinsamkeiten, Unterschiede, Verbindungen und Wechselbeziehungen untereinander:

- Abstrakte Interpretation und Datenflussanalyse (Kap. 15)
- Modellprüfung und Datenflussanalyse (Kap. 16)
- Modellprüfung und Abstrakte Interpretation (Kap. 17)

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)

Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (2)

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)

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.


Susan Horwitz, Alan J. Demers, Tim Teitelbaum. *An Efficient General Iterative Algorithm for Dataflow Analysis*. Acta Informatica 24(6):679-694, 1987.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (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.

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 und 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

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

Datenflussanalyse

Kapitel 7.1

Vorbereitung

Kapitel 7.1.1

Flussgraphen

Flow Graphs

...as representations of WHILE programs.

Definition 7.1.1.1 (Flow Graph)

A flow graph is a quadruple $G = (N, E, \mathbf{s}, \mathbf{e})$ with

- N, set of nodes.
- $E \subseteq N \times N$, set of edges.
- $\mathbf{s}$, distinguished start node w/out any predecessors.
- $\mathbf{e}$, distinguished end node w/out any successors.

Nodes represent program points, edges the branching structure of G. Every node of G is assumed to lie on a path from $\mathbf{s}$ to $\mathbf{e}$.

Node-labelled vs. Edge-labelled Flow Graphs

Given a flow graph, instructions (i.e., assignments, tests) can be represented by

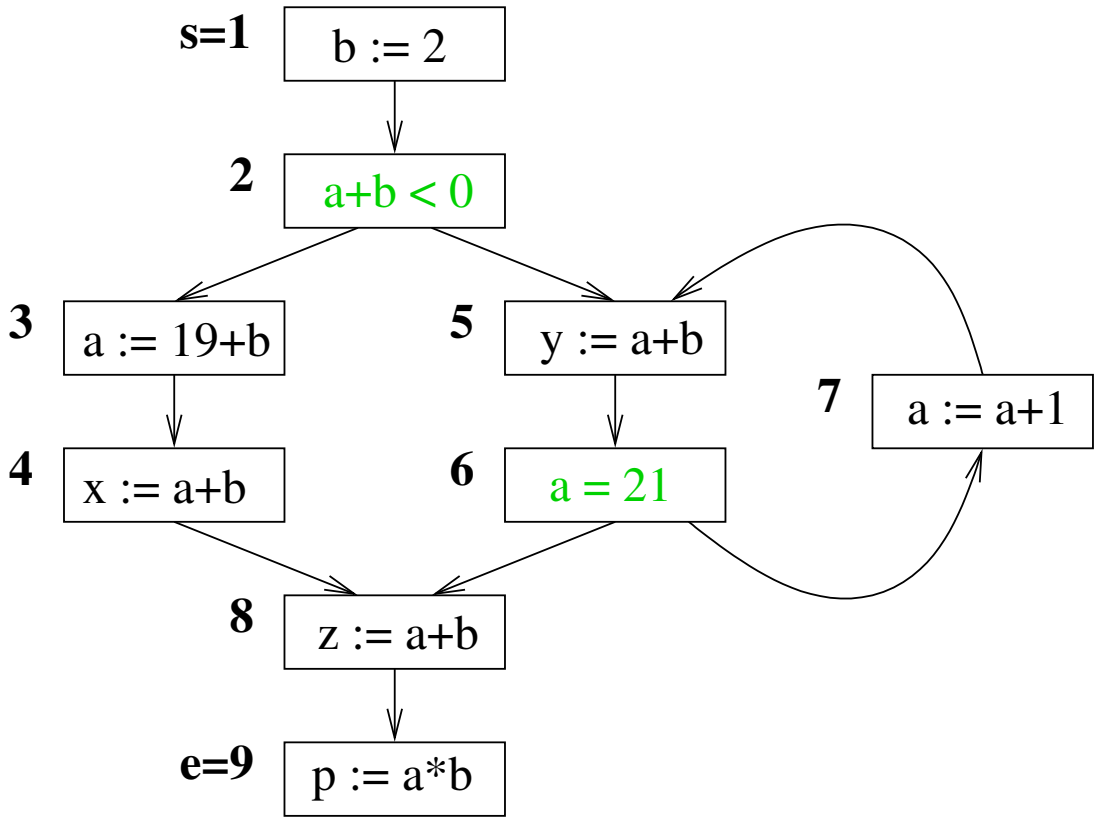
- nodes
- edges

leading to

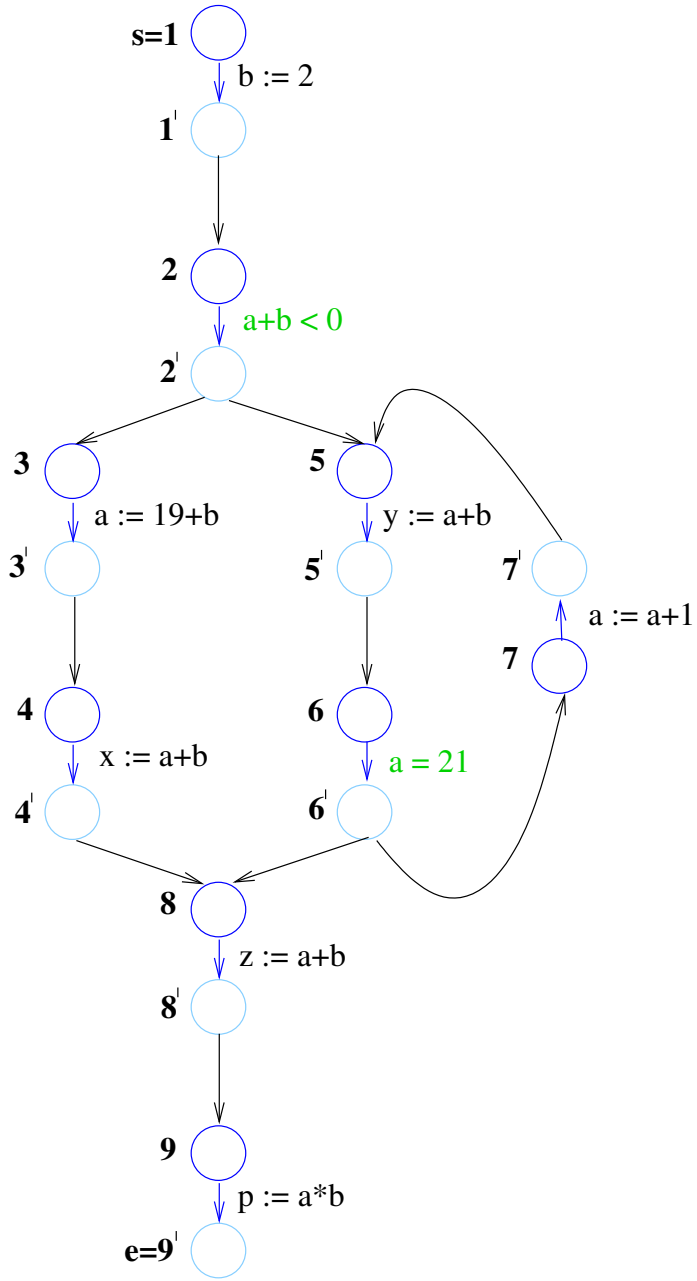
- node-labelled
- edge-labelled

flow graphs, respectively.

Example: A Node-Labelled Flow Graph

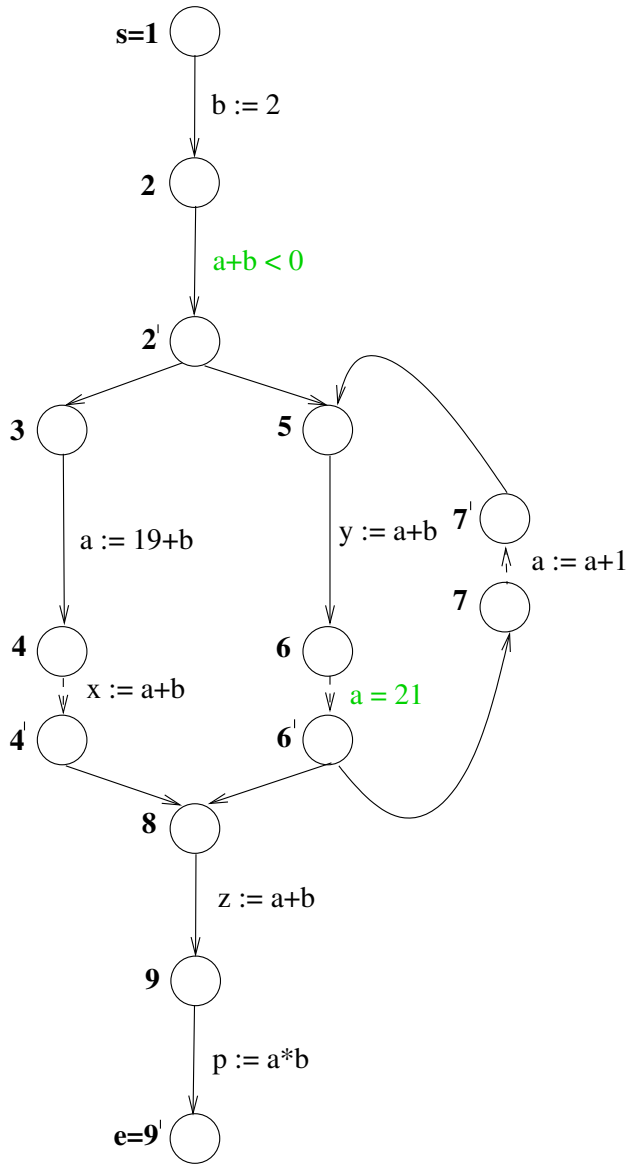


Example: An Edge-Labelled Flow Graph

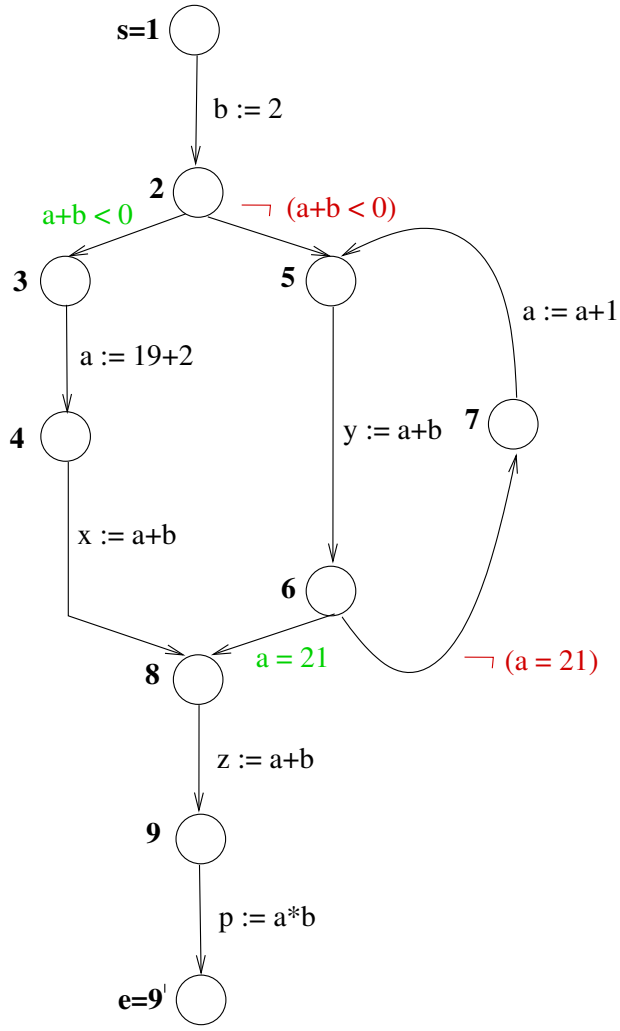


Edge-Labelled Flow Graph after Cleaning Up

a)



b)



Predecessor Nodes, Successor Nodes, Paths

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be a flow graph, m, n be two nodes of N.

Definition 7.1.1.2 (Predecessor, Successor Nodes)

- $pred_G(n) =_{df} \{ m \mid (m, n) \in E \}$ denotes the set of predecessor nodes of n.
- $succ_G(n) =_{df} \{ m \mid (n, m) \in E \}$ denotes the set of successor nodes of n.

Definition 7.1.1.3 (Paths)

- A sequence of edges $\langle (n_1, m_1), (n_2, m_2), \ldots, (n_k, m_k) \rangle$ with $m_i = n_{i+1}$, $1 \leq i < k$ is called a path from n_1 to m_k.
- $\mathbf{P}_G[m, n]$ denotes the set of all paths from m to n.

Note, if G is obvious from the context, we drop index G and write $pred$, $succ$, and $\mathbf{P}$ instead of $pred_G$, $succ_G$, and $\mathbf{P}_G$, resp.

In the following

...we consider

- edge-labelled

flow graphs, which are pragmatically advantageous by requiring less notational overhead. Moreover, we do not evaluate tests in order to avoid (some) undecidabilities, leading us to so-called non-deterministic flow graphs.

Note, advantages and disadvantages of particular flow graph variants as program representations are discussed in

- Appendix B: Pragmatics of Flow Graph Representations

Kapitel 7.1.2

Vollständige Verbände

Partially Ordered Sets, Complete Lattices

Definition 7.1.2.1 (Partially Ordered Set)

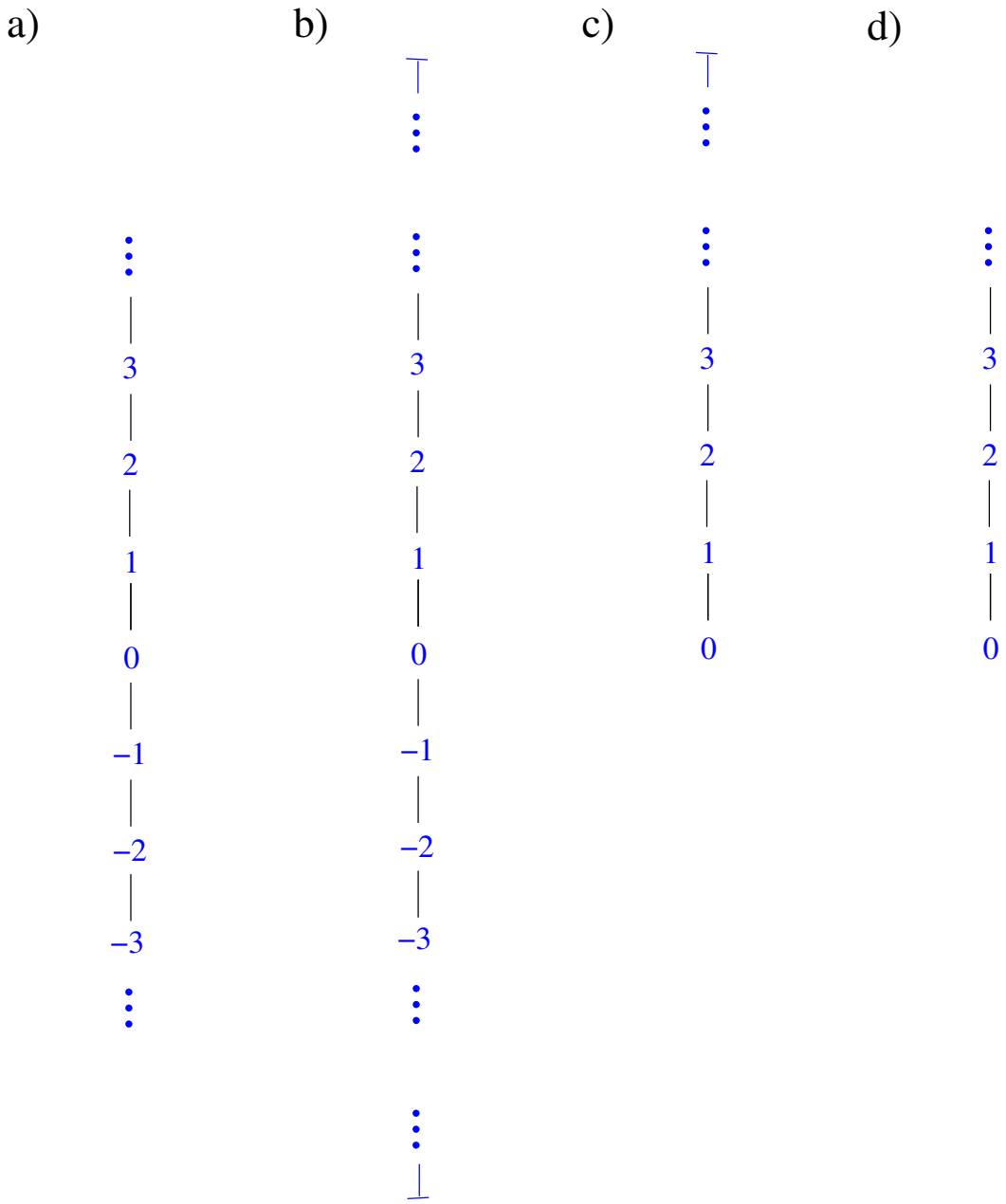
Let S be a set and $R \subseteq S \times S$ be a relation on S. Then (S, R) is called a partially ordered set (dtsch. partiell geordnete Menge) iff R is reflexive, transitive, and anti-symmetric.

Definition 7.1.2.2 (Lattice, Complete Lattice)

Let $(P, \sqsubseteq)$ be a partially ordered set. Then $(P, \sqsubseteq)$ is a

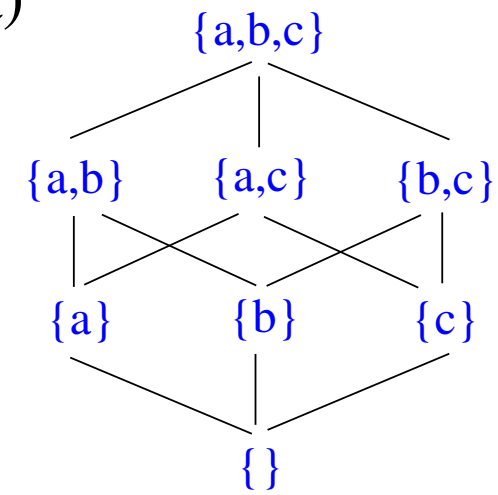
- lattice (dtsch. Verband), if every finite nonempty subset P' of P has a least upper bound and a greatest lower bound in P.
- complete lattice (dtsch. vollständiger Verband), if every subset P' of P has a least upper bound and a greatest lower bound in P.

Examples: Partially Ordered Sets and Lattices

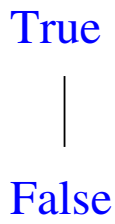


Examples: Complete Lattices

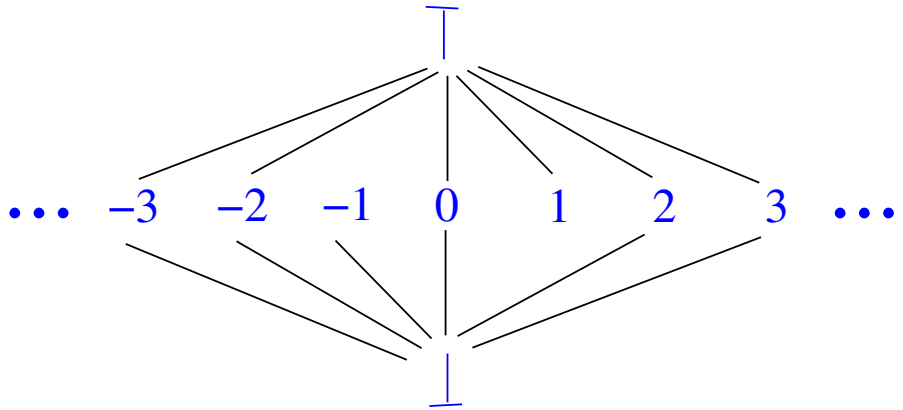
a)



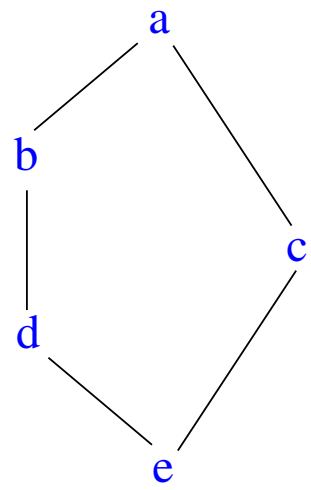
b)



c)



d)



Notions, Notations for Lattices

Let $(C, \sqsubseteq)$ be a complete lattice, $C' \subseteq C$ a subset of C. Then

- $\bigsqcap C'$ denotes the greatest lower bound of C'.
- $\bigsqcup C'$ denotes the least upper bound of C'.
- $\bigsqcap C = \bigsqcup \emptyset$ is the least element of C, denoted by $\bot$.
- $\bigsqcup C = \bigsqcap \emptyset$ is the greatest element of C, denoted by $\top$.

This gives rise to write a complete lattice as a six-tuple

- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$

where $\sqcap$, $\sqcup$, $\bot$, and $\top$ are read as meet, join, bottom, and top, respectively.

Descending, Ascending Chain Condition

Definition 7.1.2.3 (Chain Condition)

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a lattice. $\widehat{\mathcal{C}}$ satisfies the

1. descending chain condition (dtsch. absteigende Kettenbedingung), if every descending chain gets stationary, i.e., for every chain $c_1 \sqsupseteq c_2 \sqsupseteq \ldots \sqsupseteq c_n \sqsupseteq \ldots$ there is an index $m \geq 1$ with $c_m = c_{m+j}$ for all $j \in \mathbb{N}$.
2. ascending chain condition (dtsch.aufsteigende Kettenbedingung), if every ascending chain gets stationary, i.e., for every chain $c_1 \sqsubseteq c_2 \sqsubseteq \ldots \sqsubseteq c_n \sqsubseteq \ldots$ there is an index $m \geq 1$ with $c_m = c_{m+j}$ for all $j \in \mathbb{N}$.

Monotonicity, Distributivity, and Additivity

...are important properties of functions on lattices:

Definition 7.1.2.4 (Monotonicity)

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice and $f : \mathcal{C} \to \mathcal{C}$ be a function on $\mathcal{C}$. Then f is called

- monotonic iff $\forall\, c, c' \in \mathcal{C}.\ c \sqsubseteq c' \Rightarrow f(c) \sqsubseteq f(c')$
  (Preservation of the order of elements)

Definition 7.1.2.5 (Distributivity, Additivity)

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice and $f : \mathcal{C} \to \mathcal{C}$ be a function on $\mathcal{C}$. Then f is called

- distributive iff $\forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcap C') = \bigsqcap \{f(c) \mid c \in C'\}$
  (Preservation of greatest lower bounds)
- additive iff $\forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcup C') = \bigsqcup \{f(c) \mid c \in C'\}$
  (Preservation of least upper bounds)

Monotonicity

...characterized in terms of the preservation of greatest lower and least upper bounds:

Lemma 7.1.2.6

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice, $f : \mathcal{C} \to \mathcal{C}$ a function on $\mathcal{C}$. Then:

$$
\begin{aligned}
& f \text{ is monotonic} \\
\iff & \forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcap C') \sqsubseteq \bigsqcap \{f(c) \mid c \in C'\} \\
\iff & \forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcup C') \sqsupseteq \bigsqcup \{f(c) \mid c \in C'\}
\end{aligned}
$$

Relating Monotonicity, Distributivity, Additivity

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice, $f : \mathcal{C} \to \mathcal{C}$ a function on $\mathcal{C}$.

Lemma 7.1.2.7

1. f is distributive iff f is additive.
2. f is monotonic if f is distributive or additive.

Kapitel 7.2

Lokale DFA-Semantik

Local DFA Semantics

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be an edge-labelled flow graph.

Definition 7.2.1 (Local DFA Semantics)

A local abstract DFA semanctics for G is a map

$$[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$$

where $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ is a complete lattice.

Note: The elements of $\widehat{\mathcal{C}}$ are the mathematical objects modeling and representing the data flow information of interest.

Kapitel 7.3

DFA-Spezifikation

DFA Specification

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be an edge-labelled flow graph.

Definition 7.3.1 (DFA Specification)

A DFA specification for G is a triple $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ with

- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ a complete lattice.
- $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ a local abstract semantics.
- $c_{\mathbf{s}} \in \mathcal{C}$ an initial information (or start assertion).

Definition 7.3.2 (DFA Problem)

A DFA specification $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ defines a DFA problem for G.

Note

Let $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification for G. Then:

- The elements of $\mathcal{C}$ represent the data flow information of interest.
- The functions $[\![\, e \,]\!]$, $e \in E$, abstract the concrete semantics of instructions to the level of the analysis.
- $c_{\mathbf{s}} \in \mathcal{C}$ is the data flow information assumed to be valid at the startnode $\mathbf{s}$ of G.

Overall, this gives rise to call

- $\widehat{\mathcal{C}}$ a DFA lattice.
- $[\![\ ]\!]$ a local abstract DFA semantics (or DFA semantics).
- $[\![\, e \,]\!]$, $e \in E$, a local semantic DFA function (or DFA function).
- $c_{\mathbf{s}} \in \mathcal{C}$ a DFA start assertion.

Example: Availability of a Term t (1)

...a term t is available at a program point n, if t is computed along every path p from $\mathbf{s}$ to n without that any operand of t is modified after the last computation of t on p.

DFA Specification for the Availability of a Term t:

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $(\mathbb{B}, \wedge, \vee, \leq, \mathbf{falsch}, \mathbf{wahr}) = \widehat{\mathbb{B}}$
- DFA semantics
  $[\![\ ]\!]^t_{av} : E \to (\ \mathbb{B} \to \mathbb{B}\ )$ where
  $\forall\, e \in E.\ \forall\, b \in \mathbb{B}.\ [\![\, e \,]\!]^t_{av}(b) =_{df} (b \vee \mathit{Comp}^t_e) \wedge \mathit{Transp}^t_e$
- DFA start assertion: $b_{\mathbf{s}} \in \mathbb{B}$

Overall:

- Availability Specification: $\mathcal{S}^{av,t}_G = (\widehat{\mathbb{B}}, [\![\ ]\!]^t_{av}, b_{\mathbf{s}})$

Example: Availability of a Term t (2)

...where $\widehat{\mathbb{B}}$ denotes the data flow lattice and $Comp_e^t$, Mod_e^t, and $Transp_e^t$ three local predicates associated with edges and their instructions:

- $\widehat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr})$

  ...lattice of Boolean truth values: least element **falsch**, greatest element **wahr**, **falsch** $\leq$ **wahr**, logical $\wedge$ and logical $\vee$ as meet and join operation, respectively.

- $Comp_e^t$ ...**wahr**, if t is computed by the instruction at edge e, otherwise **falsch**.

- $Transp_e^t$ ...**wahr**, if e is transparent for t (i.e., no operand of t is assigned a new value by the instruction at edge e), otherwise **falsch**.

DFA Problems

...are practically relevant, if their underlying local DFA semantics are

- monotonic
- distributive/additive

and their data flow lattices satisfy the

- descending/ascending chain condition.

Properties of DFA Semantics, DFA Problems

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ be a DFA specification for G.

Definition 7.3.3 (Properties of DFA Semantics)

The local DFA semantics $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ of $\mathcal{S}_G$ is monotonic/distributive/additive iff all DFA functions $[\![e]\!]$, $e \in E$, are monotonic/distributive/additive, respectively.

Definition 7.3.4 (Properties of DFA Problems)

The DFA problem specified by $\mathcal{S}_G$

- is monotonic/distributive/additive iff the local DFA semantics $[\![\ ]\!]$ of $\mathcal{S}_G$ is monotonic/distributive/additive, respectively.
- satisfies the descending/ascending chain condition iff the DFA lattice $\widehat{C}$ of $\mathcal{S}_G$ satisfies the descending/ascending chain condition, respectively.

Example: Availability of a Term t (1)

Lemma 7.3.5 (DFA Functions)

$$\forall\, e \in E.\ [\![\, e \,]\!]^t_{av} = \begin{cases} Cst_{\mathbf{wahr}} & \text{if } Comp^t_e \wedge Transp^t_e \\ Id_{\mathbb{B}} & \text{if } \neg Comp^t_e \wedge Transp^t_e \\ Cst_{\mathbf{falsch}} & \text{otherwise} \end{cases}$$

where

- $Cst_{\mathbf{wahr}}, Cst_{\mathbf{falsch}} : \mathbb{B} \to \mathbb{B}$ (constant functions on $\mathbb{B}$)

  $\forall\, b \in \mathbb{B}.\ Cst_{\mathbf{wahr}}(b) =_{df} \mathbf{wahr}$

  $\forall\, b \in \mathbb{B}.\ Cst_{\mathbf{falsch}}(b) =_{df} \mathbf{falsch}$

- $Id_{\mathbb{B}} : \mathbb{B} \to \mathbb{B}$ (identity on $\mathbb{B}$)

  $\forall\, b \in \mathbb{B}.\ Id_{\mathbb{B}}(b) =_{df} b$

Example: Availability of a Term t (2)

Lemma 7.3.6 (Chain Condition)

$\widehat{\mathbb{B}}$ satisfies the descending and ascending chain condition.

Lemma 7.3.7 (Distributivity, Additivity)

$[\![e]\!]^t_{av}$, $e \in E$, is distributive and additive (and hence, also monotonic).

Proof. Immediately with Lemma 7.3.5 and Lemma 7.1.2.7(2).

Corollary 7.3.8 (Availability of a Term t)

The DFA problem specified by $\mathcal{S}^{av,t}_G = (\widehat{\mathbb{B}}, [\![\ ]\!]^t_{av}, b_{\mathbf{s}})$ is distributive and additive and satisfies the descending and ascending chain condition.

Towards a Global Abstract Semantics

...by globalizing a local abstract semantics for instructions to a global abstract semantics for flow graphs.

This leads to the nondeterministic operational

- collecting (*CS*) semantics

from which we derive two deterministic operational variants:

- The meet over all paths (*MOP*) semantics
- The join over all paths (*JOP*) semantics

...together with two computational deterministic denotational variants:

- The maximum fixed point (*MaxFP*) semantics
- The minimum fixed point (*MinFP*) semantics

which induce computation procedures for computing or approximating their operational counterparts.

Kapitel 7.4

Operationelle globale DFA-Semantik

Kapitel 7.4.1

Aufsammelsemantik

Extending DFA Functions from Edges to Paths

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ be a DFA specification.

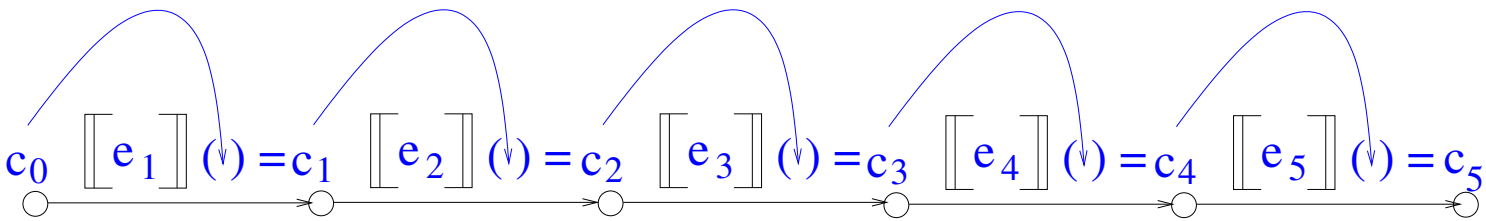
Definition 7.4.1.1 (Extending $[\![\ ]\!]$ to Paths)

The DFA semantics $[\![\ e\ ]\!]$, $e \in E$, is extended from edges onto paths $p = \langle e_1, e_2, \ldots, e_q \rangle$ by defining:

$$[\![\ p\ ]\!] =_{df} \begin{cases} Id_{\mathcal{C}} & \text{if } \lambda_p < 1 \\ [\![\ \langle e_2, \ldots, e_q \rangle\ ]\!] \circ [\![\ e_1\ ]\!] & \text{otherwise} \end{cases}$$

where $Id_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$ denotes the identity on $\mathcal{C}$, i.e., $Id_{\mathcal{C}}(c) = c$, $c \in \mathcal{C}$.

Illustrating the extension of $[\![\ ]\!]$ from edges to paths:



The Collecting DFA Semantics

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Definition 7.4.1.2 (Collecting DFA Semantics)

The (nondeterministic) collecting DFA semantics (or global abstract semantics) induced by $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]^{CS}_{\mathcal{S}_G} : N \to \mathcal{P}(\mathcal{C})$$

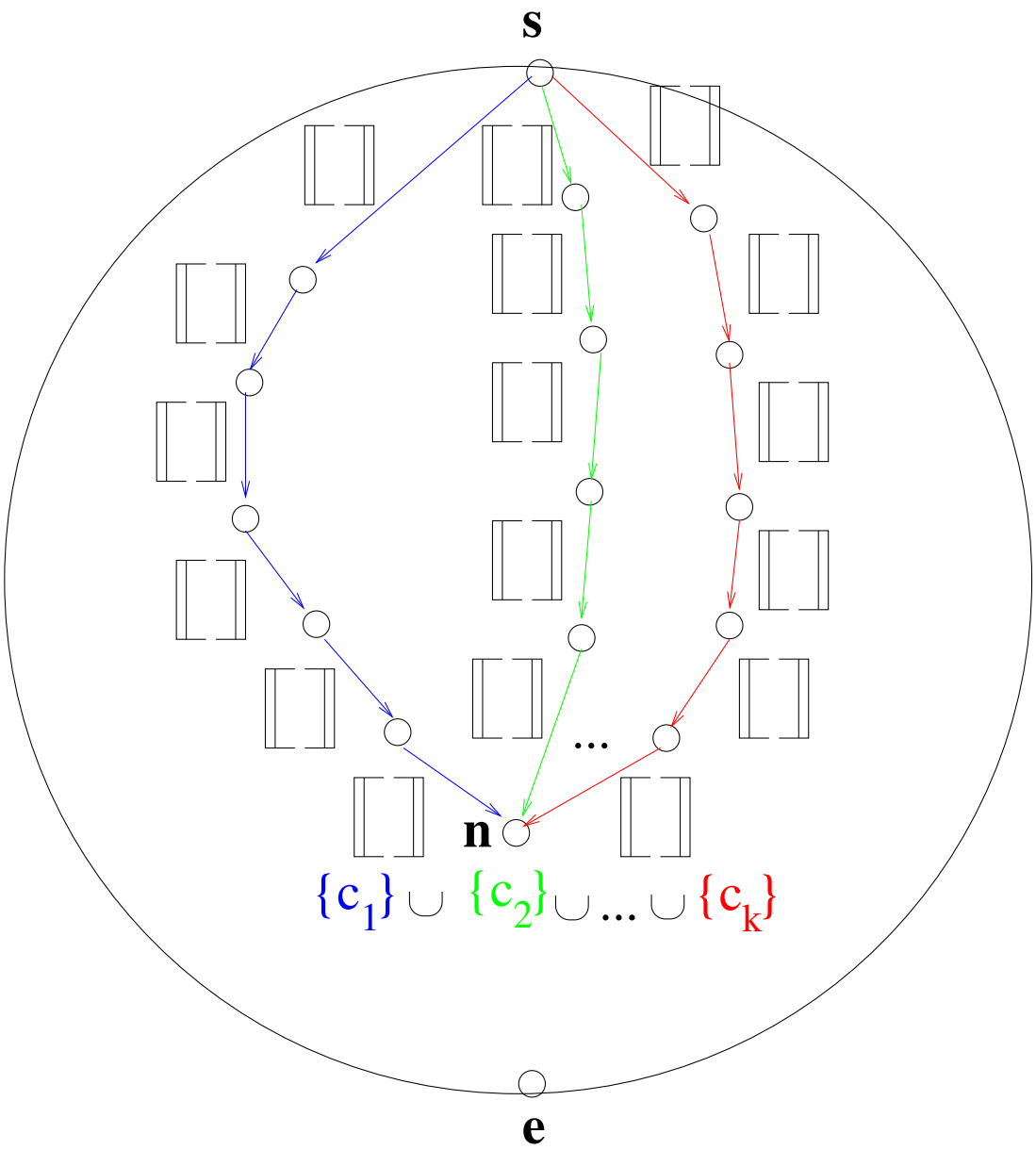
$$\forall\, n \in N.\ [\![\, n \,]\!]^{CS}_{\mathcal{S}_G} =_{df} \{\, [\![\, p \,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\}$$

where $\mathcal{P}$ denotes the powerset operator.

Note:

$$[\![\, \mathbf{s} \,]\!]^{CS}_{\mathcal{S}_G} = \{c_{\mathbf{s}}\}$$

Illustrating the Collecting DFA Semantics



Note

...if π is a WHILE program, G its flow graph representation, and $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], \bot)$ a DFA specification for G wrt the start assertion $\bot$, then the global DFA semantics at the program end node **e**

$$[\![\, \mathbf{e} \,]\!]^{CS}_{\mathcal{S}_G} \;=\; \{\, [\![\, p \,]\!](\bot) \,|\, p \in \mathbf{P}[\mathbf{s}, \mathbf{e}] \,\}$$

can be considered the nondeterministic abstract counterpart of the deterministic WHILE semantics of π for Σ:

$$[\![\, \pi \,]\!]_{\mathrm{WHILE}}(\Sigma) \;= \{\, [\![\, \pi \,]\!]_{\mathrm{WHILE}}(\sigma) \,|\, \sigma \in \Sigma \}$$

Kapitel 7.4.2

Schnitt-über-alle-Pfade-Semantik

The Meet Over All Paths (*MOP*) Semantics

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Definition 7.4.2.1 (*MOP* Semantics)

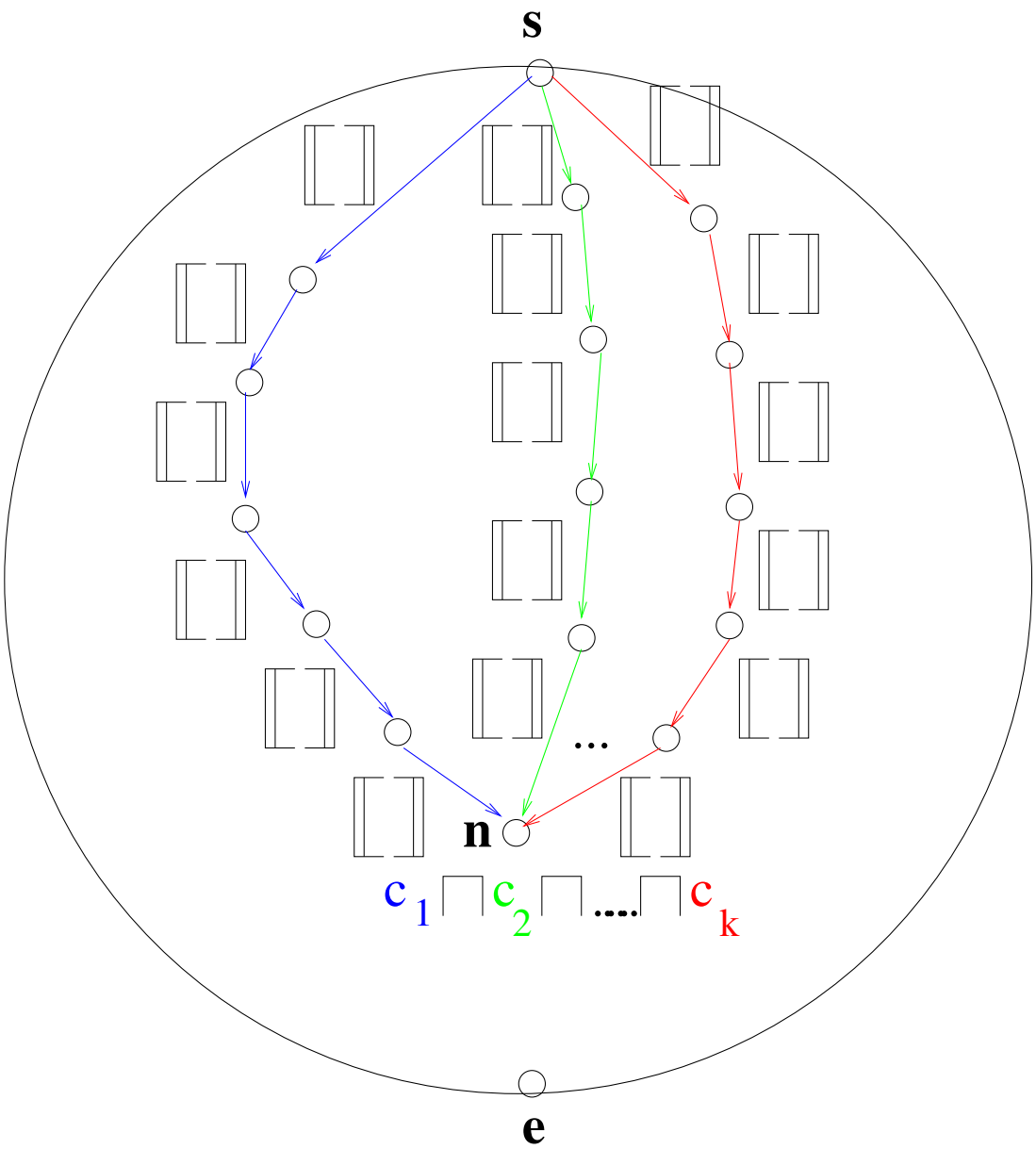
The (deterministic) *MOP* semantics of $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]^{MOP}_{\mathcal{S}_G} : N \to \mathcal{C}$$

$$\begin{aligned} \forall\, n \in N.\ [\![\, n \,]\!]^{MOP}_{\mathcal{S}_G} &=_{df} \bigsqcap [\![\, n \,]\!]^{CS}_{\mathcal{S}_G} \\ &= \bigsqcap \{\, [\![\, p \,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\} \end{aligned}$$

Note: $\bigsqcap [\![\, n \,]\!]^{CS}_{\mathcal{S}_G}$ and hence $[\![\, n \,]\!]^{MOP}_{\mathcal{S}_G}$, $n \in N$, exists, since $\widehat{\mathcal{C}}$ is a complete lattice.

Illustrating the *MOP* Semantics



Kapitel 7.4.3

Vereinigung-über-alle-Pfade-Semantik

The Join Over All Paths (*JOP*) Semantics

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Definition 7.4.3.1 (*JOP* Semantics)

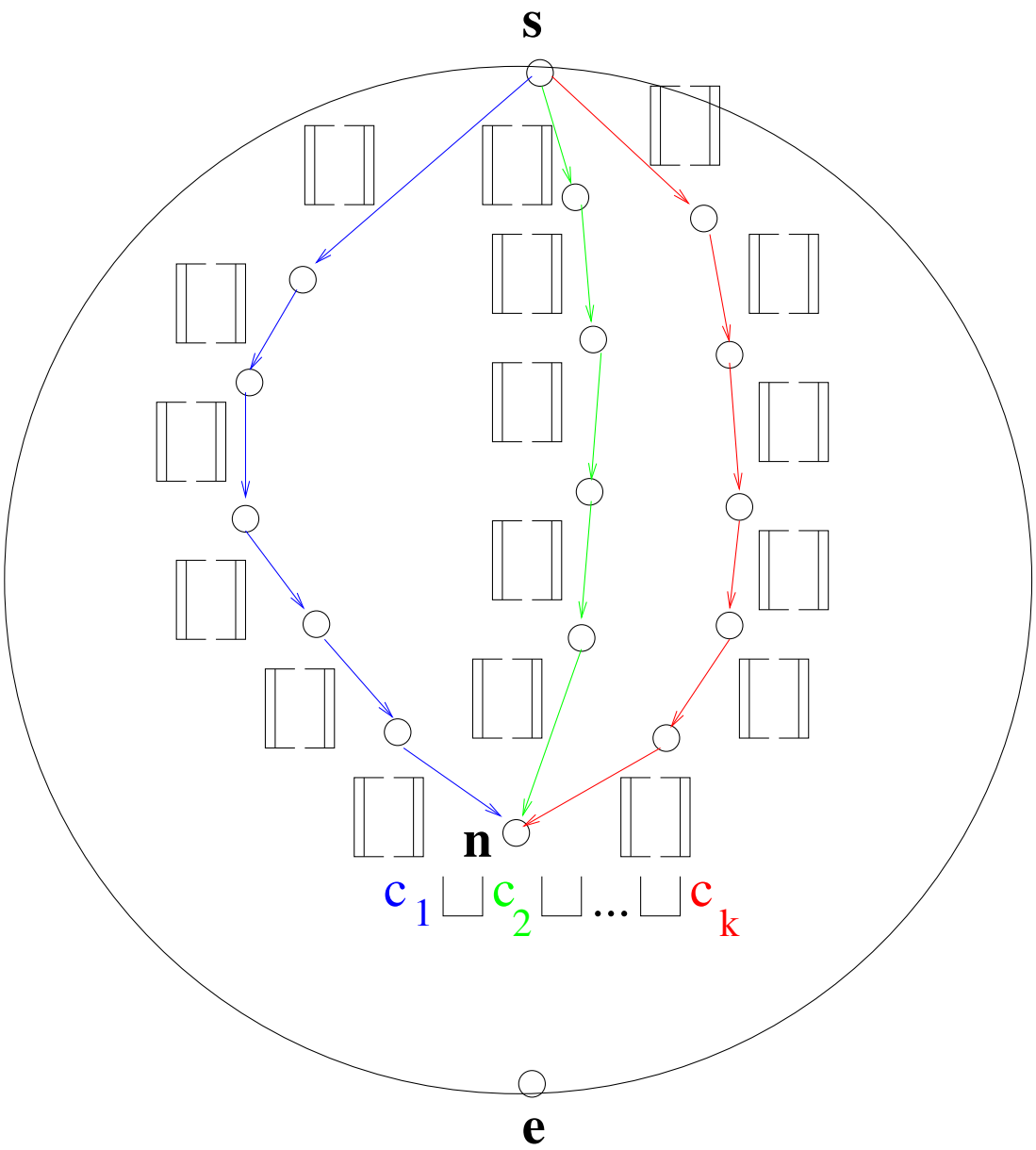
The (deterministic) *JOP* semantics of $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]^{JOP}_{\mathcal{S}_G} : N \to \mathcal{C}$$

$$\begin{aligned} \forall\, n \in N.\ [\![\, n \,]\!]^{JOP}_{\mathcal{S}_G} &=_{df} \bigsqcup [\![\, n \,]\!]^{CS}_{\mathcal{S}_G} \\ &= \bigsqcup \{\, [\![\, p \,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\} \end{aligned}$$

Note: $\bigsqcup [\![\, n \,]\!]^{CS}_{\mathcal{S}_G}$ and hence $[\![\, n \,]\!]^{JOP}_{\mathcal{S}_G}$, $n \in N$, exists, since $\widehat{\mathcal{C}}$ is a complete lattice.

Illustrating the *JOP* Semantics



Kapitel 7.4.4

SUP- und *VUP*-Semantik als spezifizierende Lösungen von DFA-Problemen

As illustrated by the Figures

...of Chapter 7.4.2 and 7.4.3, the *MOP* and the *JOP* semantics bound for program point n the DFA information

- possible at n wrt $\mathcal{S}_G$:

Independently of the path $p \in \mathbf{P}[\mathbf{s}, n]$ along which node n is reached, the information provided by p at n is

- at least as large as the *MOP* semantics at n (it can not be worse, only better):

$$[\![p]\!](c_{\mathbf{s}}) \sqsupseteq [\![n]\!]^{MOP}_{\mathcal{S}_G}$$

- at most as large as the *JOP* semantics at n (it can not be better, only worse):

$$[\![p]\!](c_{\mathbf{s}}) \sqsubseteq [\![n]\!]^{JOP}_{\mathcal{S}_G}$$

This means:

$$\forall\, p \in \mathbf{P}[\mathbf{s}, n].\ [\![n]\!]^{MOP}_{\mathcal{S}_G} \sqsubseteq [\![p]\!](c_{\mathbf{s}}) \sqsubseteq [\![n]\!]^{JOP}_{\mathcal{S}_G}$$

In other words

...the *MOP* and the *JOP* semantics provide for every program point n the DFA informations which are

- the best possible valid ones at n wrt $\mathcal{S}_G$

in the following sense:

- $[\![n]\!]^{MOP}_{\mathcal{S}_G}$ is the minimum information valid at n (it can not be worse, only better): $\forall\, p \in \mathbf{P}[\mathbf{s}, n].\ [\![n]\!]^{MOP}_{\mathcal{S}_G} \sqsubseteq [\![p]\!](c_{\mathbf{s}})$.
- $[\![n]\!]^{JOP}_{\mathcal{S}_G}$ is the maximum information valid at n (it can not be better, only worse): $\forall\, p \in \mathbf{P}[\mathbf{s}, n].\ [\![n]\!]^{JOP}_{\mathcal{S}_G} \sqsupseteq [\![p]\!](c_{\mathbf{s}})$.

This means, the *MOP* and *JOP* semantics ensure the absence of 'surprises': Independently of the path $p \in \mathbf{P}[\mathbf{s}, n]$ along which node n is reached, we always have:

$$[\![n]\!]^{MOP}_{\mathcal{S}_G} \sqsubseteq [\![p]\!](c_{\mathbf{s}}) \sqsubseteq [\![n]\!]^{JOP}_{\mathcal{S}_G}$$

The Specifying Solutions of a DFA Problem

This gives rise to the following definition:

Definition 7.4.4.1 (Specifying Solutions of a DFA P.)

The *MOP* and the *JOP* semantics of a flow graph define the specifying solutions of a DFA problem, its so-called *MOP* and *JOP* solutions.

Conservative DFA Algorithms

Definition 7.4.4.2 (Conservative DFA Algorithm)

A DFA algorithm A is

- *MOP* conservative
- *JOP* conservative

for $\mathcal{S}_G$, if A terminates with

- a lower approximation of the *MOP* semantics
- an upper approximation of the *JOP* semantics

of $\mathcal{S}_G$, respectively.

Tight DFA Algorithms

Definition 7.4.4.3 (Tight DFA Algorithm)

A DFA algorithm A is

- *MOP* tight
- *JOP* tight

for $\mathcal{S}_G$, if A terminates with

- the *MOP* semantics
- the *JOP* semantics

of $\mathcal{S}_G$, respectively.

Kapitel 7.4.5

Unentscheidbarkeit von *SUP*- und *VUP*-Semantik

Unfortunately

...the definitions of the *MOP* and *JOP* semantics do not directly induce

- effective computation procedures

for computing them (think of loops in a non-deterministically interpreted flow graph causing the number of paths reaching a node to be infinite).

Even worse, the *MOP* and *JOP* semantics of a flow graph are

- not decidable!

Undecidability of the *MOP* Semantics

Theorem 7.4.5.1 (Undecidability of the *MOP* Sem.)

There is no algorithm A such that:

- The input of A is
  - a DFA specification $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$.
  - algorithms for computing of the meet, the equality test, and the application of monotonic functions on $\widehat{\mathcal{C}}$.
- The output of A is the *MOP* semantics of $\mathcal{S}_G$.

(John B. Kam, Jeffrey D. Ullman. Monotone Data Flow Analysis Frameworks. Acta Informatica 7:305-317, 1977)

Undecidability of the *JOP* Semantics

Corollary 7.4.5.2 (Undecidability of the *JOP* Sem.)

There is no algorithm A such that:

- The input of A is
  - a DFA specification $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$.
  - algorithms for the computing the meet, the equality test, and the application of monotonic functions on $\widehat{\mathcal{C}}$.
- The output of A is the *JOP* semantics of $\mathcal{S}_G$.

Towards Conservative and Tight DFA Alg's

...because of the preceding negative results we complement the operational approach underlying the *MOP* and *JOP* semantics an orthogonal denotational globalization approach of a local abstract semantics leading to the

- Maximum fixed point (*MaxFP*) semantics
- Minimum fixed point (*MinFP*) semantics

of a flow graph, respectively.

The *MaxFP* and *MinFP* semantics are also called the

- Maximum fixed point (*MaxFP*) solution
- Maximum fixed point (*MaxFP*) solution

of a DFA problem, respectively, which (under certain conditions) can

- effectively be computed.

Kapitel 7.5

Denotationelle globale DFA-Semantik

Kapitel 7.5.1

Maximale Fixpunktsemantik

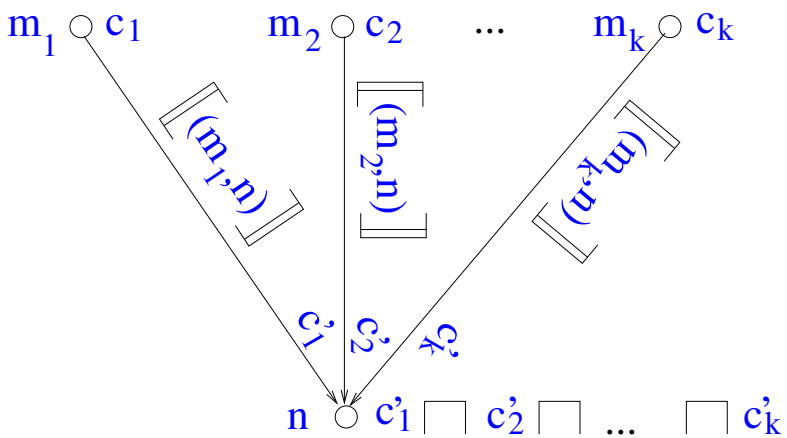
The Maximum Fixed Point (*MaxFP*) Approach

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Equation System 7.5.1.1 (*MaxFP* Equation System)

$$inf(n) \quad = \quad \begin{cases} c_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigsqcap \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{otherwise} \end{cases}$$

Illustrating the *MaxFP* Approach ($n \neq \mathbf{s}$):



The *MaxFP* Semantics

Let

- $inf^{\star}_{c_s}(n),\ n \in N$

denote the greatest solution of Equation System 7.5.1.1.

Definition 7.5.1.2 (*MaxFP* Semantics)

The (deterministic) *MaxFP* semantics of $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]^{MaxFP}_{\mathcal{S}_G} : N \to \mathcal{C}$$

$$\forall\, n \in N.\ [\![\ n\ ]\!]^{MaxFP}_{\mathcal{S}_G} =_{df} inf^{\star}_{c_s}(n)$$

Note:

$$[\![\ \mathbf{s}\ ]\!]^{MaxFP}_{\mathcal{S}_G} = c_{\mathbf{s}}$$

Kapitel 7.5.2

Minimale Fixpunktsemantik

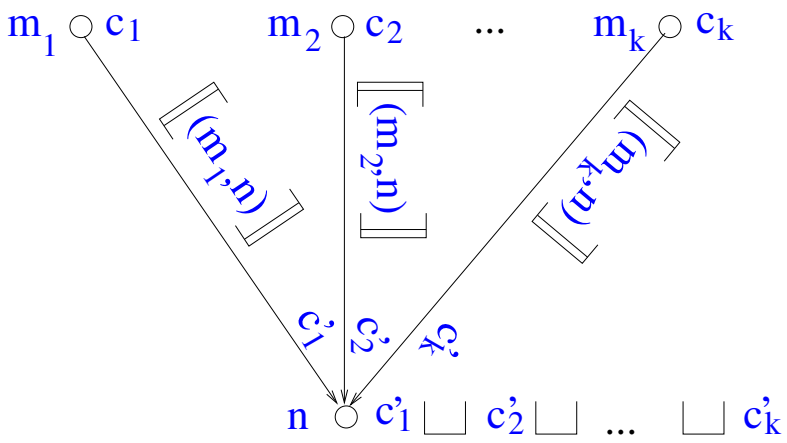
The Minimum Fixed Point (*MinFP*) Approach

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ be a DFA specification.

Equation System 7.5.2.1 (*MinFP* Equation System)

$$inf(n) = \begin{cases} c_s & \text{if } n = \mathbf{s} \\ \bigsqcup \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{otherwise} \end{cases}$$

Illustrating the *MinFP* Approach ($n \neq \mathbf{s}$):



The *MinFP* Semantics

Let

- $inf^{\star}_{c_s}(n),\ n \in N$

denote the least solution of Equation System 7.5.2.1.

Definition 7.5.2.2 (*MinFP* Semantics)

The *MinFP* semantics of $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]^{MinFP}_{\mathcal{S}_G} : N \to \mathcal{C}$$

$$\forall\, n \in N.\ [\![\, n \,]\!]^{MinFP}_{\mathcal{S}_G} =_{df} inf^{\star}_{c_s}(n)$$

Note:

$$[\![\, \mathbf{s} \,]\!]^{MinFP}_{\mathcal{S}_G} = c_{\mathbf{s}}$$

Kapitel 7.6

Generischer Fixpunktalgorithmus

The *MaxFP* and *MinFP* Semantics

...are practically relevant because *MaxFP* Equation System 7.5.1.1 and *MinFP* Equation System 7.5.2.1 induce a generic

- iterative computation procedure (Algorithm 7.6.1.1)

approximating their greatest and smallest solutions, respectively, i.e., the *MaxFP* and *MinFP* semantics.

Kapitel 7.6.1

Algorithmus

The Generic Fixed Point Algorithm 7.6.1.1 (1)

Input: A DFA specification $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$.

Output: On termination of the algorithm (cf. Termination Theorem 7.6.2.1), variable $inf[n]$ stores the *MaxFP* solution of $\mathcal{S}_G$ at node n.

Additionally (cf. Safety Theorem 7.7.1 and Coincidence Theorem 7.7.2): If $[\![\ ]\!]$ is

- distributive: $inf[n]$ stores
- monotonic: $inf[n]$ stores a lower approximation of

the *MOP* solution of $\mathcal{S}_G$ at node n.

Remark: The variable *workset* controls the iterative process. It temporarily stores a set of nodes of G, whose annotations have recently been changed and thus can impact the annotations of their neighbouring nodes.

The Generic Fixed Point Algorithm 7.6.1.1 (2)

(Prologue: Initializing *inf* and *workset*)
FORALL $n \in N \backslash \{\mathbf{s}\}$ DO $inf[n] := \top$ OD;
$inf[\mathbf{s}] := c_{\mathbf{s}}$;
$workset := N$;
(Main loop: The iterative fixed point computation)
WHILE $workset \neq \emptyset$ DO
    CHOOSE $m \in workset$;
        $workset := workset \backslash \{ m \}$;
        (Updating the annotations of all successors of node 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 7.6.2

Terminierung

Termination

Theorem 7.6.2.1 (Termination)

The Generic Fixed Point Algorithm 7.6.1.1 terminates w/ the

1. *MaxFP* semantics of $\mathcal{S}_G$, if
   - 1.1 $[\![\ ]\!]$ is monotonic
   - 1.2 $\widehat{\mathcal{C}}$ satisfies the descending chain condition.
2. *MinFP* semantics of $\mathcal{S}_G$, if
   - 2.1 $[\![\ ]\!]$ is monotonic
   - 2.2 $\widehat{\mathcal{C}^{usd}}$ satisfies the ascending chain condition, where

$$\widehat{\mathcal{C}^{usd}} =_{df} (\mathcal{C}, \sqcup, \sqcap, \sqsupseteq, \top, \bot)$$

is lattice $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ put up-side down.

The Computable Solutions of a DFA Problem

...together the Generic Fixed Point Algorithm 7.6.1.1 and Termination Theorem 7.6.2.1 give rise to the following definition:

Definition 7.6.2.2 (Computable Solutions of a DFA P.)

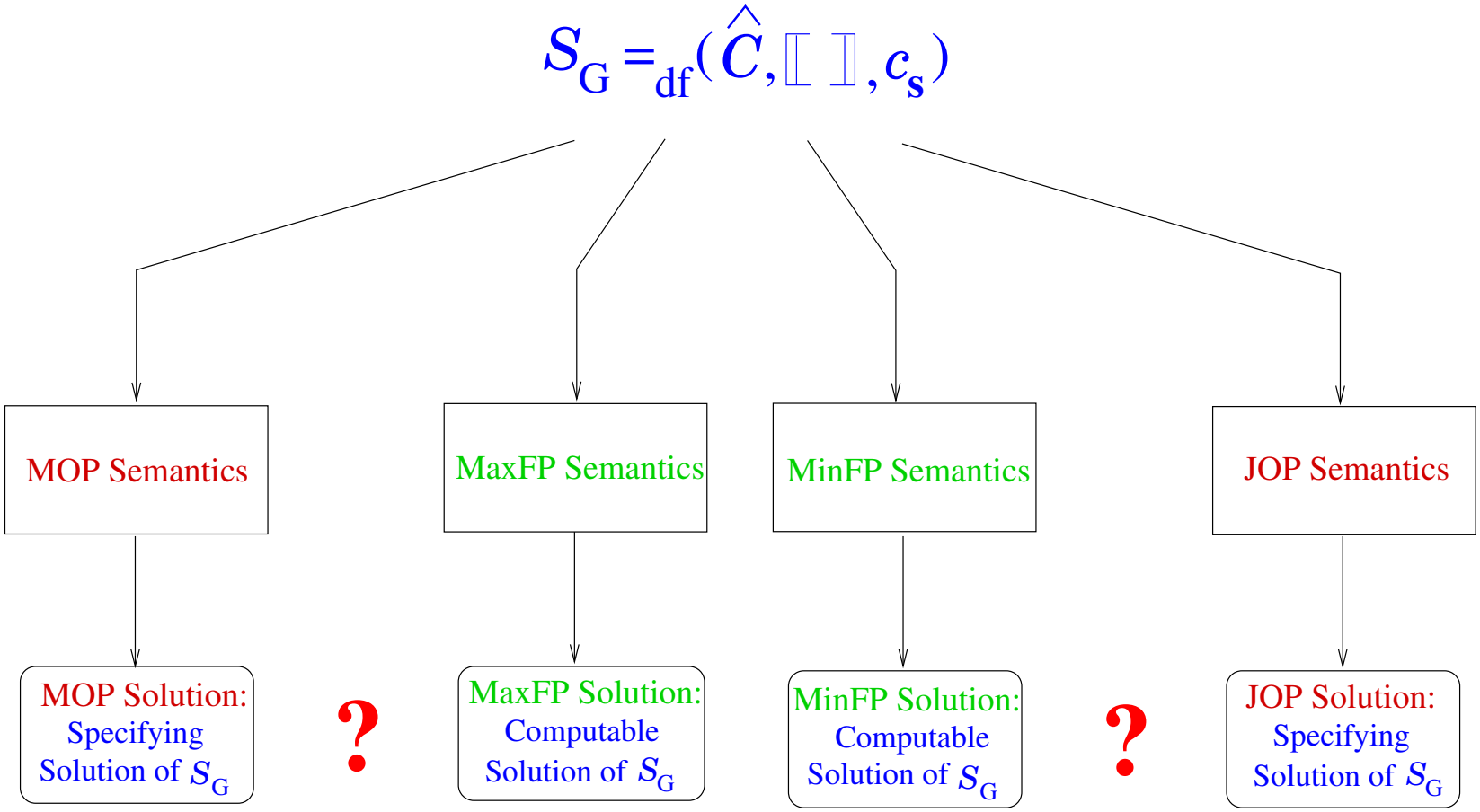
The *MaxFP* and the *MinFP* semantics of a flow graph define the computable solutions of a DFA problem, its so-called *MaxFP* and *MinFP* solutions.

Kapitel 7.7

Sicherheit und Koinzidenz

MOP/*MaxFP*- and *JOP*/*MinFP* Semantics

...of a DFA Specification and the question of their relationship:



Safety

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ be a DFA specification.

Theorem 7.7.1 (Safety)

1. The *MaxFP* semantics of $\mathcal{S}_G$ is a safe (i.e., lower) approximation of the *MOP* semantics of $\mathcal{S}_G$, i.e.,

$$\forall\, n \in N.\ [\![\, n \,]\!]^{MaxFP}_{\mathcal{S}_G} \sqsubseteq [\![\, n \,]\!]^{MOP}_{\mathcal{S}_G}$$

2. The *MinFP* semantics of $\mathcal{S}_G$ is a safe (i.e., upper) approximation of the *JOP* semantics of $\mathcal{S}_G$, i.e.,

$$\forall\, n \in N.\ [\![\, n \,]\!]^{MinFP}_{\mathcal{S}_G} \sqsupseteq [\![\, n \,]\!]^{JOP}_{\mathcal{S}_G}$$

if the DFA semantics $[\![\ ]\!]$ is monotonic.

Coincidence

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ be a DFA specification.

Theorem 7.7.2 (Coincidence)

1. The *MaxFP* and the *MOP* semantics of $\mathcal{S}_G$ coincide, i.e.,

$$\forall\, n \in N.\ [\![\, n \,]\!]^{MaxFP}_{\mathcal{S}_G} = [\![\, n \,]\!]^{MOP}_{\mathcal{S}_G}$$

2. The *MinFP* and the *JOP* semantics of $\mathcal{S}_G$ coincide, i.e.,

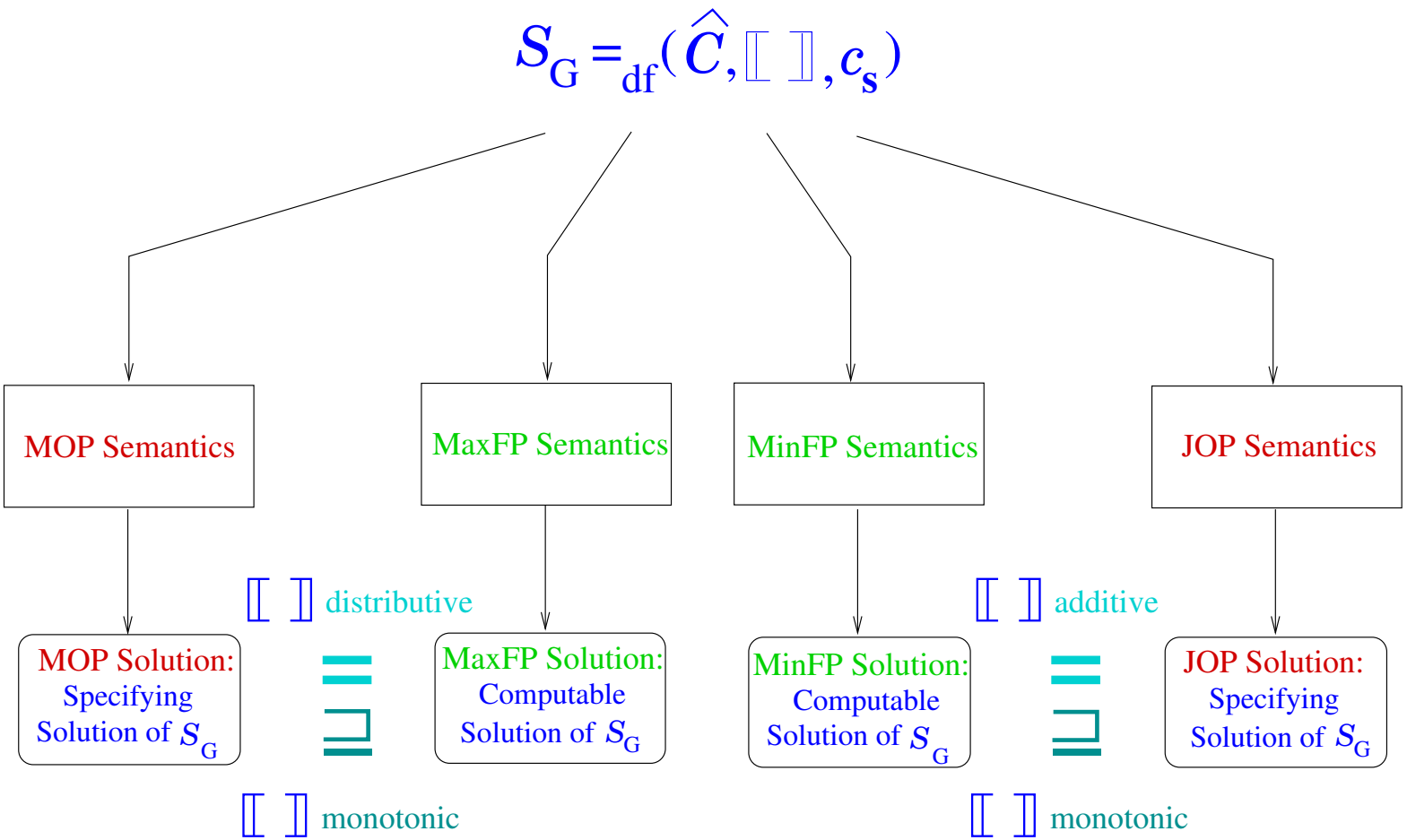
$$\forall\, n \in N.\ [\![\, n \,]\!]^{MinFP}_{\mathcal{S}_G} = [\![\, n \,]\!]^{JOP}_{\mathcal{S}_G}$$

if the DFA semantics $[\![\ ]\!]$ is distributive or additive, respectively.

Recall Lemma 7.1.2.7(1): $[\![\ ]\!]$ is distributive iff $[\![\ ]\!]$ is additive.

MOP / *MaxFP*- and *JOP* / *MinFP* Semantics

...of a DFA Specification and their relationship:



Conservativity of Algorithm 7.6.1.1

Corollary 7.7.3 (*MOP*/*JOP* Conservativity)

Algorithm 7.6.1.1 is

- *MOP* (*JOP*) conservative

for $\mathcal{S}_G$, i.e., it terminates with a lower (upper) approximation of the *MOP* (*JOP*) semantics of $\mathcal{S}_G$, if

1. $[\![\ ]\!]$ is monotonic
2. $\widehat{\mathcal{C}}$ satisfies the descending (ascending) chain condition, respectively.

Tightness of Algorithm 7.6.1.1

Corollary 7.7.4 (*MOP*/*JOP* Tightness)

Algorithm 7.6.1.1 is

- *MOP* (*JOP*) tight

for $\mathcal{S}_G$, i.e., it terminates with the *MOP* (*JOP*) semantics of $\mathcal{S}_G$, if

1. $[\![\ ]\!]$ is distributive (additive)
2. $\widehat{\mathcal{C}}$ satisfies the descending (ascending) chain condition

respectively.

Kapitel 7.8

Korrektheit und Vollständigkeit

Soundness and Completeness (1)

Analysis Scenario:

- Let ϕ be a program property of interest (e.g., availability of a term, liveness of a variable, etc.).
- Let $\mathcal{S}_G^\phi$ be a DFA specification designed for ϕ.

Definition 7.8.1 (Soundness)

$\mathcal{S}_G^\phi$ is *MOP* sound (*JOP* sound) for ϕ, if, whenever the *MOP* semantics (*JOP* semantics) of $\mathcal{S}_G^\phi$ indicates that ϕ is valid, then ϕ is valid.

Definition 7.8.2 (Completeness)

$\mathcal{S}_G^\phi$ is *MOP* complete (*JOP* complete) for ϕ, if, whenever ϕ is valid, then the *MOP* semantics (*JOP* semantics) of $\mathcal{S}_G^\phi$ indicates that ϕ is valid.

Soundness and Completeness (2)

Intuitively

- *MOP* soundness means: $[\![\ ]\!]^{MOP}_{\mathcal{S}^{\phi}_{G}}$ 'implies' ϕ.
- *MOP* completeness means: ϕ 'implies' $[\![\ ]\!]^{MOP}_{\mathcal{S}^{\phi}_{G}}$.

and

- *JOP* soundness means: ϕ 'implies' $[\![\ ]\!]^{JOP}_{\mathcal{S}^{\phi}_{G}}$.
- *JOP* completeness means: $[\![\ ]\!]^{JOP}_{\mathcal{S}^{\phi}_{G}}$ 'implies' ϕ.

Soundness and Completeness (3)

Intuitively, if $\mathcal{S}_G^\phi$ is *MOP* (*JOP*) sound and complete for ϕ, this means:

We compute

- the property of interest,
- the whole property of interest,
- and only the property of interest.

In other words, we compute

- the program property of interest accurately!

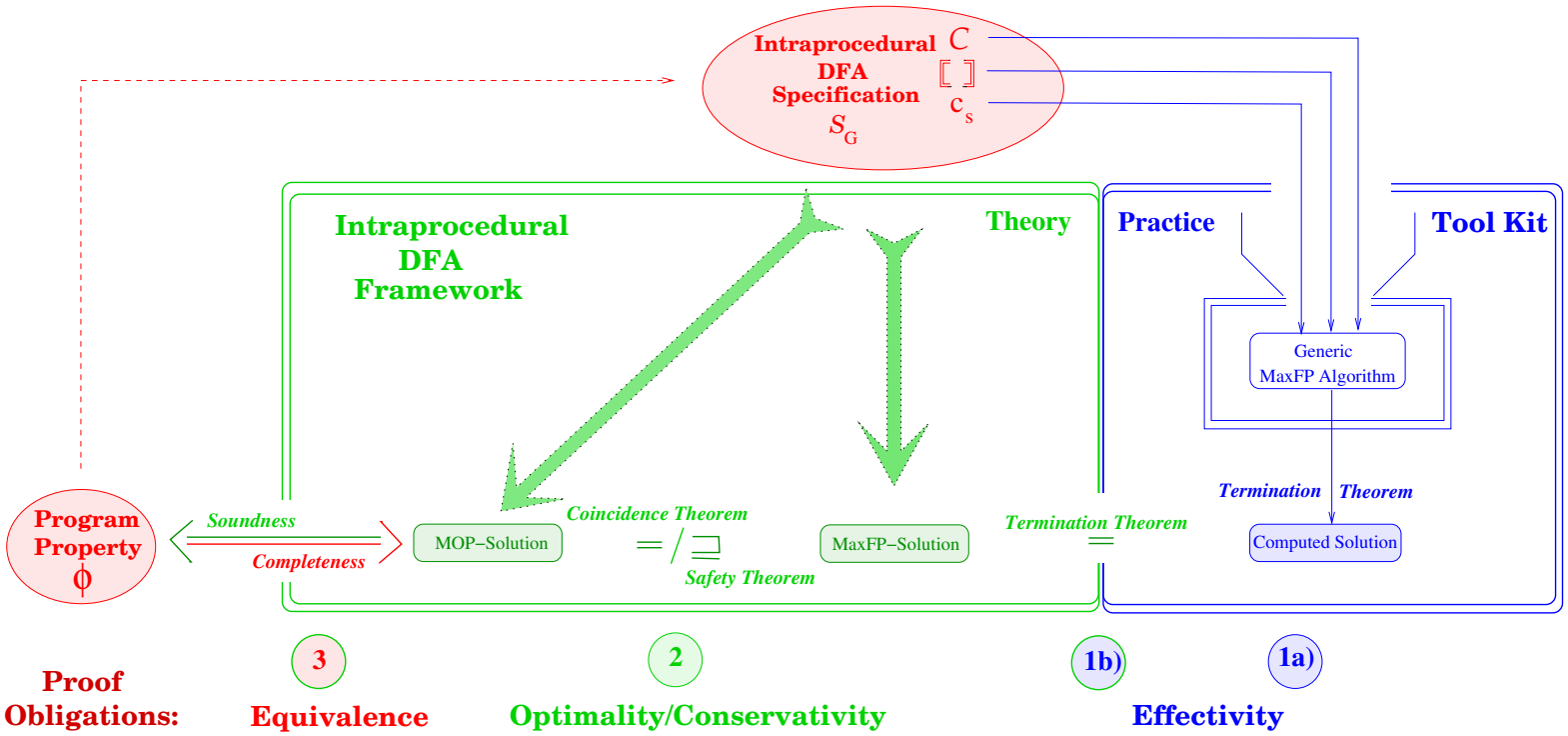
Kapitel 7.9

DFA-Rahmen- und Werkzeugkastensicht

Data Flow Analysis: A Holistic View

...considering (intraprocedural) DFA from a holistic angle: The

- Framework and Toolkit (*MOP*/*MaxFP*) View of DFA



Data Flow Analysis in Practice

...working with framework and toolkit is a three-stage process:

The Three-Stage Process

1. **Identifying a Program Property of Interest**

   Identify a program property of interest (e.g., availability of a term, liveness of a variable, etc.), say ϕ, and define ϕ formally.

2. **Designing a DFA Specification**

   Design a DFA specification $\mathcal{S}_G^\phi = (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ for ϕ.

3. **Accomplishing Proof Obligations, Obtaining Guarantees**

   Prove a fixed set of proof obligations about the components of $\mathcal{S}_G^\phi$ and the relation of its *MOP* (*JOP*) solution and ϕ to obtain guarantees that its *MaxFP* (*MinFP*) solution is sound or even sound and complete for ϕ.

Proof Obligations, Implied Guarantees (1)

Proof obligations and guarantees in detail:

- Proof Obligations 1a), 1b): Descending (ascending) chain condition for $\widehat{\mathcal{C}}$, monotonicity for $[\![\ ]\!]$

  Guarantees:

  - Effectivity: Termination of Algorithm 7.6.1.1 with the *MaxFP* (*MinFP*) semantics of $\mathcal{S}_G^\phi$.
  - Conservativity: The *MaxFP* (*MinFP*) solution of $\mathcal{S}_G^\phi$ is *MOP* (*JOP*) conservative.

- Proof Obligation 2): Distributivity (additivity) for $[\![\ ]\!]$

  Guarantee:

  - Tightness: The *MaxFP* (*MinFP*) semantics of $\mathcal{S}_G^\phi$ is *MOP* (*JOP*) tight.

Proof Obligations, Implied Guarantees (2)

- Proof Obligation 3): Equivalence of $MOP_{\mathcal{S}_G^\phi}$ ($JOP_{\mathcal{S}_G^\phi}$) and ϕ

  Guarantees:

  - Whenever the *MOP* solution of $\mathcal{S}_G^\phi$ indicates the validity of ϕ, then it is valid: Soundness.

    $\rightsquigarrow$ We compute the property of interest, and only the property of interest.
  - Whenever ϕ is valid, this is indicated by the *MOP* solution of $\mathcal{S}_G^\phi$: Completeness.

    $\rightsquigarrow$ We compute the whole property of interest.
  - Vice versa for the *JOP* solution of $\mathcal{S}_G^\phi$.

  Guarantee of combined Soundness and Completeness:

  - We compute program property ϕ accurately!

Kapitel 7.10

Anwendungsbeispiele

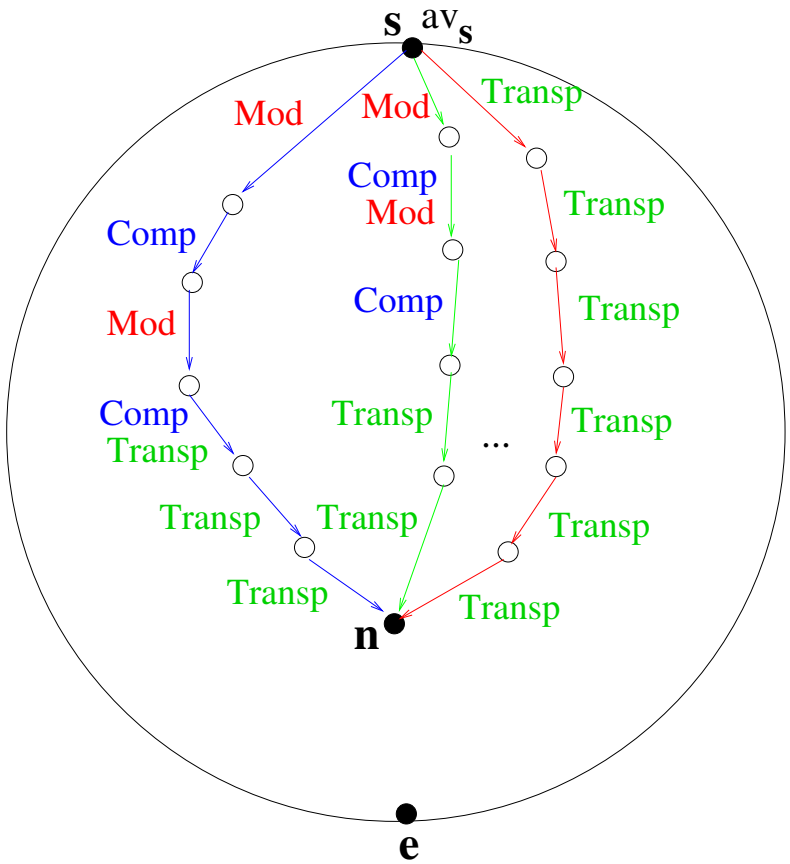
Kapitel 7.10.1

Distributive DFA: Verfügbare Ausdrücke

Intuitively

...a term is available at a node if, no matter which path is taken from the entry of the program to that node, the term is computed without that any of the variables occurring in it is redefined before reaching this node.

Illustration:



Availability

...we will specify the availability problem in four different variants in order to illustrate the

- usage of different lattices for DFA
- class of so-called
  - bitvector
  - Gen/Kill

  DFA problems availability is a typical representative of.

...computing available terms is a

- canonical example of a distributive DFA problem.

Preliminaries

Let $\iota_e \equiv x := exp$ (or $\iota_e \equiv exp$) be the instruction (or the condition) at edge e, t a term.

Local Predicates (for edges)

- $Comp_e^t$

  ...**wahr**, if t is computed by ι_e (i.e., t is a subterm of the right-hand side expression exp of ι_e), otherwise **falsch**.

- Mod_e^t

  ...**wahr**, if t is modified by ι_e (i.e., ι_e assigns a new value to some operand of t), otherwise **falsch**.

- $Transp_e^t =_{df} \neg Mod_e^t$

  ...**wahr**, if e is transparent for t (i.e., ι_e does not assign a new value to any operand of t), otherwise **falsch**.

Variant 1: Fixing the Setting

...availability for a single term t.

Lattice

- $\widehat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, \leq, \mathbf{falsch}, \mathbf{wahr})$

  ...lattice of Boolean truth values: least element **falsch**, greatest element **wahr**, **falsch** $\leq$ **wahr**, logical $\wedge$ and logical $\vee$ as meet and join operation, respectively.

Utility Functions

- Constant Functions $Cst_{\mathbf{wahr}}, Cst_{\mathbf{falsch}} : \mathbb{B} \to \mathbb{B}$

  $\forall\, b \in \mathbb{B}.\ Cst_{\mathbf{wahr}}(b) =_{df} \mathbf{wahr}$
  $\forall\, b \in \mathbb{B}.\ Cst_{\mathbf{falsch}}(b) =_{df} \mathbf{falsch}$

- Identity $Id_{\mathbb{B}} : \mathbb{B} \to \mathbb{B}$

  $\forall\, b \in \mathbb{B}.\ Id_{\mathbb{B}}(b) =_{df} b$

Variant 1: Specifying the DFA

DFA Specification

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $(\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr}) = \widehat{\mathbb{B}}$
- DFA semantics
  $[\![\ ]\!]^t_{av} : E \to (\ \mathbb{B} \to \mathbb{B}\ )$ where
  $\forall\, e \in E\ \forall\, b \in \mathbb{B}.\ [\![\ e\ ]\!]^t_{av}(b) =_{df} (b \vee \mathit{Comp}^t_e) \wedge \mathit{Transp}^t_e$
- Start assertion: $b_{\mathbf{s}} \in \mathbb{B}$

Availability Specification for t

- Specification: $\mathcal{S}^{av,t}_G = (\widehat{\mathbb{B}}, [\![\ ]\!]^t_{av}, b_{\mathbf{s}})$

Variant 1: Fulfilling the Proof Obligations

Lemma 7.10.1.1 (DFA Functions)

$$\forall\, e \in E.\ [\![\, e \,]\!]^t_{av} = \begin{cases} Cst_{\mathbf{wahr}} & \text{if } Comp^t_e \wedge Transp^t_e \\ Id_{\mathbb{IB}} & \text{if } \neg Comp^t_e \wedge Transp^t_e \\ Cst_{\mathbf{falsch}} & \text{otherwise} \end{cases}$$

Lemma 7.10.1.2 (Chain Conditions)

$\widehat{\mathbb{IB}}$ satisfies the descending and ascending chain condition.

Lemma 7.10.1.3 (Distributivity, Additivity)

$[\![\ ]\!]^t_{av}$ is distributive and additive.

Proof. Immediately with Lemma 7.10.1.1.

Corollary 7.10.1.4 (Monotonicity)

$[\![\ ]\!]^t_{av}$ is monotonic.

Variant 1: Collecting the Guarantees

...on termination, tightness.

Theorem 7.10.1.5 (Termination)

Applied to $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\ ]\!]_{av}^t, b_{\mathbf{s}})$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}_G^{av,t}$.

Proof. Immediately with Lemma 7.10.1.2, Corollary 7.10.1.4, and Termination Theorem 7.6.2.1.

Theorem 7.10.1.6 (Tightness)

Applied to $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\ ]\!]_{av}^t, b_{\mathbf{s}})$, Algorithm 7.6.1.1 is *MOP*/*JOP* tight for $\mathcal{S}_G^{av,t}$ (i.e., terminates with the *MOP*/*JOP* semantics of $\mathcal{S}_G^{av,t}$).

Proof. Immediately with Lemma 7.10.1.3, Coincidence Theorem 7.7.2, and Termination Theorem 7.6.2.1.

Variant 2: Fixing the Setting

...availability for a finite set of terms T.

Lattice

- $\widehat{\mathcal{P}(T)} =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T)$

  ...power set lattice of T: least element $\emptyset$, greatest element T, subset relation $\subseteq$ as ordering relation, set intersection $\cap$ and set union $\cup$ as meet and join operation, respectively.

Variant 2: Specifying the DFA

DFA Specification

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T) = \widehat{\mathcal{P}(T)}$
- DFA semantics
  $[\![\ ]\!]^T_{av} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$ where
  $\forall\, e \in E\ \forall\, T' \in \mathcal{P}(T).\ [\![e]\!]^T_{av}(T') =_{df}$
  $\{t \in T \mid (t \in T' \vee Comp^t_e) \wedge Transp^t_e\}$
- Start assertion: $T_s \in \mathcal{P}(T)$

Availability Specification for T

- Specification: $\mathcal{S}^{av,T}_G = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]^T_{av}, T_s)$

Variant 2: Fulfilling the Proof Obligations

Lemma 7.10.1.7 (Chain Conditions)

$\widehat{\mathcal{P}(T)}$ satisfies the descending and ascending chain condition.

Lemma 7.10.1.8 (Distributivity, Additivity)

$[\![\ ]\!]^T_{av}$ is distributive and additive.

Corollary 7.10.1.9 (Monotonicity)

$[\![\ ]\!]^T_{av}$ is monotonic.

Variant 2: Collecting the Guarantees

...on termination, tightness.

Theorem 7.10.1.10 (Termination)

Applied to $\mathcal{S}_G^{av,T} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av}^T, T_{\mathbf{s}})$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}_G^{av,T}$.

Proof. Immediately with Lemma 7.10.1.7, Corollary 7.10.1.9, and Termination Theorem 7.6.2.1.

Theorem 7.10.1.11 (Tightness)

Applied to $\mathcal{S}_G^{av,T} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av}^T, T_{\mathbf{s}})$, Algorithm 7.6.1.1 is *MOP*/*JOP* tight for $\mathcal{S}_G^{av,T}$ (i.e., it terminates with the *MOP*/*JOP* semantics of $\mathcal{S}_G^{av,T}$).

Proof. Immediately with Lemma 7.10.1.8, Coincidence Theorem 7.7.2, and Termination Theorem 7.6.2.1.

Variant 3: Fixing the Setting (1)

...availability for a finite set of terms T, $|T| = n$.

Lattice

- $\widehat{\mathbb{B}^n} =_{df} (\mathbb{B}^n, \wedge_{pw}, \vee_{pw}, <_{pw}, \overline{\mathbf{falsch}}, \overline{\mathbf{wahr}})$

  ...n-ary cross-product lattice over $\mathbb{B}$: least element $\overline{\mathbf{falsch}} =_{df} (\mathbf{falsch}, \ldots, \mathbf{falsch}) \in \mathbb{B}^n$, greatest element $\overline{\mathbf{wahr}} =_{df} (\mathbf{wahr}, \ldots, \mathbf{wahr}) \in \mathbb{B}^n$, ordering relation $<_{pw}$ as pointwise extension of $<$ from $\widehat{\mathbb{B}}$ to $\widehat{\mathbb{B}^n}$, $\wedge_{pw}$ and $\vee_{pw}$ as pointwise extensions of logical $\wedge$ and logical $\vee$ from $\widehat{\mathbb{B}}$ to $\widehat{\mathbb{B}^n}$ as meet and join operation, respectively.

Variant 3: Fixing the Setting (2)

Utility Functions

- $ix : T \to \{1, \ldots, n\},\ ix^{-1} : \{1, \ldots, n\} \to T$

  ...bijective index mappings which uniquely associates every term $t \in T$ with a number in $\{1, \ldots, n\}$ and vice versa.

  The $ix(t)^{th}$ element of an element
  $\bar{b} = (b_1, \ldots, b_{ix(t)}, \ldots, b_n) \in \mathbb{B}^n$
  is the availability information for t stored in $\bar{b}$.

- $\cdot\downarrow_\cdot : \mathbb{B}^n \to \{1, \ldots, n\} \to \mathbb{B}$

  ...projection function which yields the i^{th} element of an element $\bar{b} \in \mathbb{B}^n$, i.e., $\forall\, i \in \{1, \ldots, n\}.\ \bar{b}\downarrow_i =_{df} b_i$.

Variant 3: Specifying the DFA (cross-pr. view)

DFA Specification (cross-product view (cpv))

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $(\mathbb{B}^n, \wedge_{pw}, \vee_{pw}, <_{pw}, \overline{\textbf{falsch}}, \overline{\textbf{wahr}}) = \widehat{\mathbb{B}^n}$
- DFA semantics
  $[\![\ ]\!]^T_{av,cpv} : E \to (\ \mathbb{B}^n \to \mathbb{B}^n\ )$ where
  $\forall\, e \in E\ \forall\, v \in \mathbb{B}^n.\ [\![\ e\ ]\!]^T_{av,cpv}(\bar{b}) =_{df} \bar{b}'$
  where $\forall\, i \in \{1, \ldots, n\}.\ \bar{b}'\downarrow_i\ =_{df}$
  $(\bar{b}\downarrow_i\ \vee\ Comp_e^{ix^{-1}(i)}) \wedge Transp_e^{ix^{-1}(i)}$
- Start assertion: $\bar{b}_{\mathbf{s}} \in \mathbb{B}^n$

Availability Specification for T

- Specification: $\mathcal{S}_G^{av,T,cpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]^T_{av,cpv}, \bar{b}_{\mathbf{s}})$

Variant 3: Towards the Bitvector View (1)

...as implementation of $\mathcal{S}_G^{av,T,cpv}$:

- $\widehat{\mathbb{B}^n}$ can efficiently be implemented in terms of bitvectors $\vec{bv} = [d_1, \ldots, d_n]$, $d_i \in \{0,1\}$, $1 \leq i \leq n$, of length n.
- Let $\mathcal{BV}^n$ denote the set of all bitvectors of length n.
- Let $\vec{bv}[i] = d_i$ for all $\vec{bv} = [d_1, \ldots, d_n] \in \mathcal{BV}^n$, $1 \leq i \leq n$.
- Let $\vec{0} =_{df} [0, \ldots, 0] \in \mathcal{BV}^n$ and $\vec{1} =_{df} [1, \ldots, 1] \in \mathcal{BV}^n$.
- Let $min_{\mathcal{BV}}$ and $max_{\mathcal{BV}}$ be the bitwise minimum ("logical $\wedge$") and the bitwise maximum function ("logical $\vee$") on bitvectors, i.e., $\forall\, \vec{bv_1}, \vec{bv_2} \in \mathcal{BV}^n \; \forall\, i \in \{1, \ldots, n\}$.
  - $(\vec{bv_1}\ min_{\mathcal{BV}}\ \vec{bv_2})[i] =_{df} min(\vec{bv_1}[i], \vec{bv_2}[i])$
  - $(\vec{bv_1}\ max_{\mathcal{BV}}\ \vec{bv_2})[i] =_{df} max(\vec{bv_1}[i], \vec{bv_2}[i])$

Variant 3: Towards the Bitvector View (2)

Utility Functions:

- $ix : T \to \{1, \ldots, n\},\ ix^{-1} : \{1, \ldots, n\} \to T$

  ...bijective index mappings which associate every term $t \in T$ uniquely with a number in $\{1, \ldots, n\}$ and vice versa.

  The $ix(t)^{th}$ element of a bitvector
  $\vec{bv} = [d_1, \ldots, d_{ix(t)}, \ldots, d_n)] \in \mathcal{BV}^n$
  is the availability information for t stored in $\vec{bv}$.

Variant 3: Towards the Bitvector View (3)

...extending and adapting local predicates to bitvectors:

- $\overrightarrow{Comp}_e^T \in \mathcal{BV}^n$

  $$\forall\, i \in \{1, \ldots, n\}.\ \overrightarrow{Comp}_e^T\ [i] =_{df} \begin{cases} 1 & \text{if } Comp_e^{ix^{-1}(i)} \\ 0 & \text{otherwise} \end{cases}$$

- $\overrightarrow{Transp}_e^T \in \mathcal{BV}^n$

  $$\forall\, i \in \{1, \ldots, n\}.\ \overrightarrow{Transp}_e^T\ [i] =_{df} \begin{cases} 1 & \text{if } Transp_e^{ix^{-1}(i)} \\ 0 & \text{otherwise} \end{cases}$$

Variant 3: Specifying the DFA (bitvector view)

DFA Specification (bitvector view (bvv))

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $$(\mathcal{BV}^n, min_{\mathcal{BV}}, max_{\mathcal{BV}}, <_{\mathcal{BV}}, \vec{0}, \vec{1}) = \widehat{\mathcal{BV}^n}$$
- DFA semantics
  $[\![\ ]\!]^T_{av,bvv} : E \to (\ \mathcal{BV}^n \to \mathcal{BV}^n\ )$ where
  $\forall\, e \in E\ \forall\, \vec{bv} \in \mathcal{BV}^n.\ [\![\ e\ ]\!]^T_{av,bvv}(\vec{bv}) =_{df}$
  $$(\vec{bv}\ max_{\mathcal{BV}}\ \overrightarrow{Comp}^T_e)\ min_{\mathcal{BV}}\ \overrightarrow{Transp}^T_e$$
- Start assertion: $\vec{bv_s} \in \mathcal{BV}^n$

Availability Specification for T

- Specification: $\mathcal{S}^{av,T,bvv}_G = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]^T_{av,bvv}, \vec{bv_s})$

Variant 3: Fulfilling the Proof Obligations

Lemma 7.10.1.12 (Chain Conditions)

$\widehat{\mathbb{B}^n}$ and $\widehat{\mathcal{BV}^n}$ satisfy the descending and ascending chain condition.

Lemma 7.10.1.13 (Distributivity, Additivity)

$[\![\ ]\!]^T_{av,cpv}$ and $[\![\ ]\!]^T_{av,bvv}$ are distributive and additive.

Corollary 7.10.1.14 (Monotonicity)

$[\![\ ]\!]^T_{av,cpv}$ and $[\![\ ]\!]^T_{av,bvv}$ are monotonic.

Variant 3: Collecting the Guarantees (1)

...on termination.

Theorem 7.10.1.15 (Termination)

Applied to $\mathcal{S}_G^{av,T,cpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]_{av,cpv}^T, \bar{b}_s)$ or $\mathcal{S}_G^{av,T,bvv} = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]_{av,bvv}^T, \vec{bv}_s)$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}_G^{av,T,cpv}$ or $\mathcal{S}_G^{av,T,bvv}$.

Proof. Immediately with Lemma 7.10.1.12, Corollary 7.10.1.14, and Termination Theorem 7.6.2.1.

Variant 3: Collecting the Guarantees (2)

...on tightness.

Theorem 7.10.1.16 (Tightness)

Applied to $\mathcal{S}_G^{av,T,cpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]^T_{av,cpv}, \bar{b}_s)$ or $\mathcal{S}_G^{av,T,bvv} = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]^T_{av,bvv}, \vec{bv}_s)$, Algorithm 7.6.1.1 is *MOP*/ *JOP* tight for $\mathcal{S}_G^{av,T,cpv}$ or $\mathcal{S}_G^{av,T,bvv}$, respectively (i.e., it terminates with the *MOP*/*JOP* semantics of $\mathcal{S}_G^{av,T,cpv}$ or $\mathcal{S}_G^{av,T,bvv}$, respectively).

Proof. Immediately with Lemma 7.10.1.13, Coincidence Theorem 7.7.2, and Termination Theorem 7.6.2.1.

Note:

- Applied to $\mathcal{S}_G^{av,T,bvv}$ instead of its cross-product counterpart, Algorithm 7.6.1.1 can take advantage of the efficient bitvector operations available on actual processors.
- This gives rise to call availability a bitvector problem.

Variant 4: Fixing the Setting

...availability for a finite set of terms T.

Introducing Gen/Kill Predicates for edges

- $Gen_e^T =_{df} \{t \in T \mid Comp_e^t \wedge \neg Mod_e^t\}$
  $= \{t \in T \mid Comp_e^t \wedge Transp_e^t\}$
- $Kill_e^T =_{df} \{t \in T \mid Mod_e^t\}$
  $= \{t \in T \mid \neg Transp_e^t\}$

Variant 4: Specifying the DFA (gen/kill view)

DFA Specification (gen/kill view (gkv))

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T) = \widehat{\mathcal{P}(T)}$
- DFA semantics
  $[\![\ ]\!]^T_{av,gkv} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$ where
  $\forall e \in E\ \forall T' \in \mathcal{P}(T).\ [\![e]\!]^T_{av,gkv}(T') =_{df} (T' \setminus Kill^T_e) \cup Gen^T_e$
- Start assertion: $T_{\mathbf{s}} \in \mathcal{P}(T)$

Availability Specification for T

- Specification: $\mathcal{S}^{av,T,gkv}_G = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]^T_{av,gkv}, T_{\mathbf{s}})$

Variant 4: Fulfilling the Proof Obligations

Comparing

- $[\![\ ]\!]^T_{av,gkv} : E \to (\,\mathcal{P}(T) \to \mathcal{P}(T)\,)$ where
  $\forall e \in E\, \forall T' \in \mathcal{P}(T).\, [\![\, e \,]\!]^T_{av,gkv}(T') =_{df} (T' \setminus Kill^T_e) \cup Gen^T_e$

with

- $[\![\ ]\!]^T_{av} : E \to (\,\mathcal{P}(T) \to \mathcal{P}(T)\,)$ where
  $\forall\, e \in E\ \forall\, T' \in \mathcal{P}(T).\, [\![\, e \,]\!]^T_{av}(T') =_{df}$
  $\{t \in T \mid (t \in T' \vee Comp^t_e) \wedge Transp^t_e\}$

...we get:

Lemma 7.10.1.17 (Equality)

$$[\![\ ]\!]^T_{av} = [\![\ ]\!]^T_{av,gkv}$$

Variant 4: Collecting the Guarantees

...on termination, tightness.

Theorem 7.10.1.18 (Termination)

Applied to $\mathcal{S}_G^{av,T,gkv} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]^T_{av,gkv}, T_s)$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}_G^{av,T,gkv}$.

Proof. Immediately with Lemma 7.10.1.7, Lemma 7.10.1.8, Corollary 7.10.1.9, and Termination Theorem 7.6.2.1.

Theorem 7.10.1.19 (Tightness)

Applied to $\mathcal{S}_G^{av,T,gkv} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]^T_{av,gkv}, T_s)$, Algorithm 7.6.1.1 is *MOP*/*JOP* tight for $\mathcal{S}_G^{av,T,gk}$ (i.e., it terminates with the *MOP*/ *JOP* semantics of $\mathcal{S}_G^{av,T,gkv}$).

Proof. Immediately with Lemma 7.10.1.7, Lemma 7.10.1.8, Coincidence Theorem 6.7.2, and Termination Theorem 7.6.2.1.

Availability again as a Gen/Kill-Problem (1)

...specializing the generic *MaxFP* Equation System 7.5.1.1:

Equation System 7.5.1.1 (*MaxFP* Equation System)

$$inf(n) = \begin{cases} c_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigsqcap \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{otherwise} \end{cases}$$

...for the availability problem yields:

Equation System 7.10.1.20 (Availability)

$$Available(n) = \begin{cases} T_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigcap \{ [\![(m,n)]\!]^{T}_{av,gk}(Available(m)) \mid m \in pred(n) \} & \text{otherwise} \end{cases}$$

Availability again as a Gen/Kill Problem (2)

...expanding additionally $[\![\ ]\!]^T_{av,gkv}$ we get:

Equation System 7.10.1.21 (Availability)

$$Available(n) =$$

$$\begin{cases} T_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigcap \{ (Available(m) \setminus Kill^T_{(m,n)}) \cup Gen^T_{(m,n)} \mid m \in pred(n) \} & \text{otherwise} \end{cases}$$

Note: Both Equation System 7.10.1.21 and the definition of the DFA semantics

- $[\![\ ]\!]^T_{av,gkv} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$ where
  $\forall e \in E\ \forall T' \in \mathcal{P}(T).\ [\![e]\!]^T_{av,gkv}(T') =_{df} (T' \setminus Kill^T_e) \cup Gen^T_e$

give rise to call availability a Gen/Kill problem.

Gen/Kill (or Bitvector) Problems

...including properties like

- availability and very busyness of terms, liveness and reaching definitions of variables, etc.

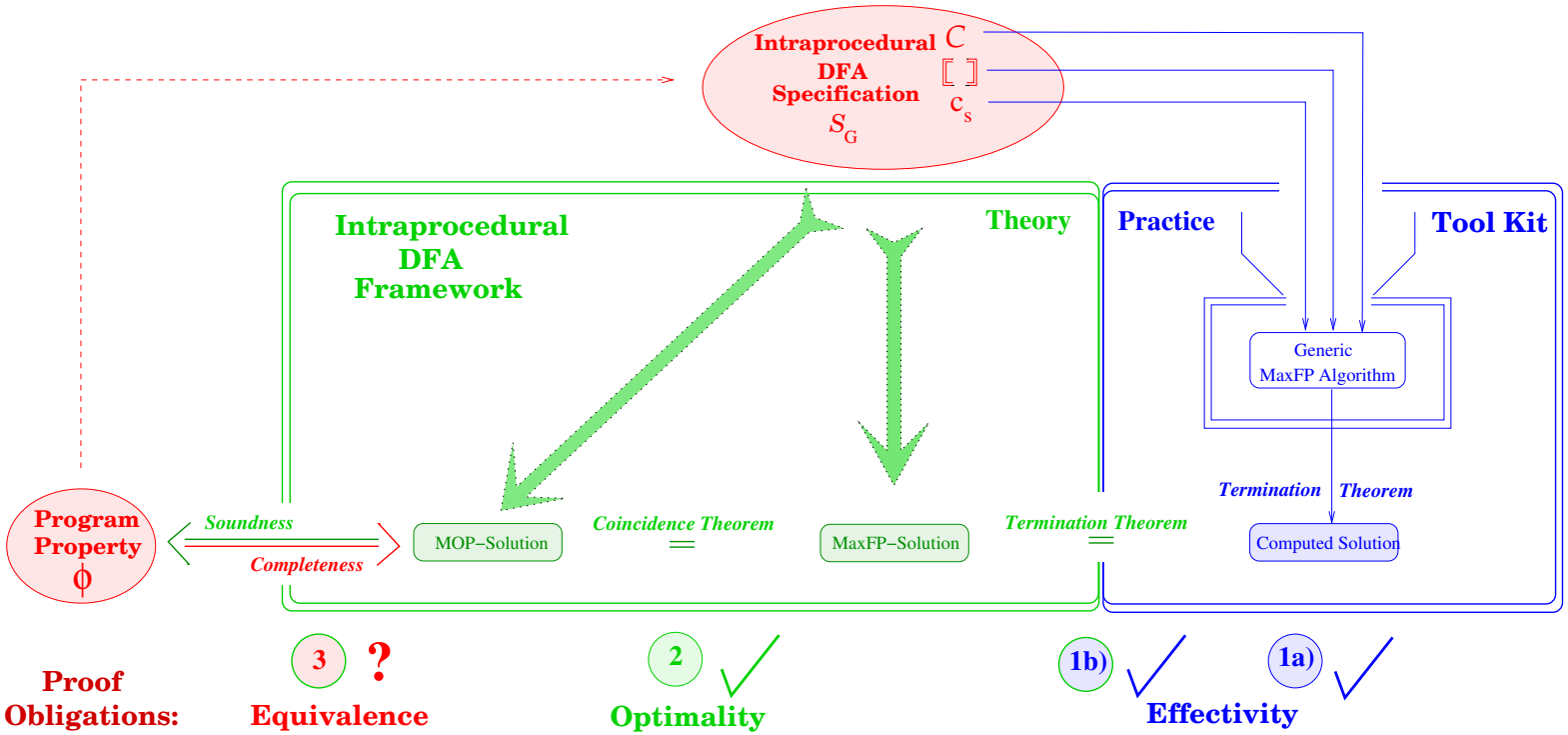
form despite their conceptual simplicity a most important class of DFA problems with numerous applications in program optimization including:

- Partially redundant expression elimination (busy/lazy code motion)
- Strength reduction (busy/lazy strength reduction)
- Partial dead-code elimination
- Partially redundant assignment elimination
- Assignment motion
- ...

...see course notes of LVA 185.A04 Optimizing Compilers for further details.

Variants 1 Thru 4: Closing the Final Proof Gap

...proving soundness and completeness for the *MOP* view of the availability property using $\mathcal{S}_G^{av,t}$ (Variant 1) as example:



Recall

...informally, a term is available at a node

- if, no matter which path is taken from the entry of the program to that node, the term is computed without that any of the variables occurring in it is redefined before reaching this node.

Note

- If entry of the program is replaced by entry of the procedure, the informal 'definition' of availibility does not foresee the possibility of the availability of an expression at the procedure entry itself.
- Situations where this availability is ensured by the calling context of the procedure, are thus not captured and can not be dealt with.

Towards defining Availability Formally

...useful notation.

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be a flow graph, and *Predicate* a predicate defined for edges $e \in E$.

For paths $p = \langle e_1, \ldots, e_q \rangle \in \mathbf{P}[m, n]$, we define:

- p_i, $1 \leq i \leq q$, denotes the i^{th} edge e_i of p.
- $p_{[k,l]}$ denotes the subpath $\langle e_k, \ldots, e_l \rangle$ of p.
- $\lambda_p = q$ denotes the length of p, i.e., the number of edges of p.

For predicates along paths, we define:

- $Predicate_p^{\forall} \iff_{df} \forall\, 1 \leq i \leq \lambda_p.\ Predicate_{p_i}$
- $Predicate_p^{\exists} \iff_{df} \exists\, 1 \leq i \leq \lambda_p.\ Predicate_{p_i}$

Availability

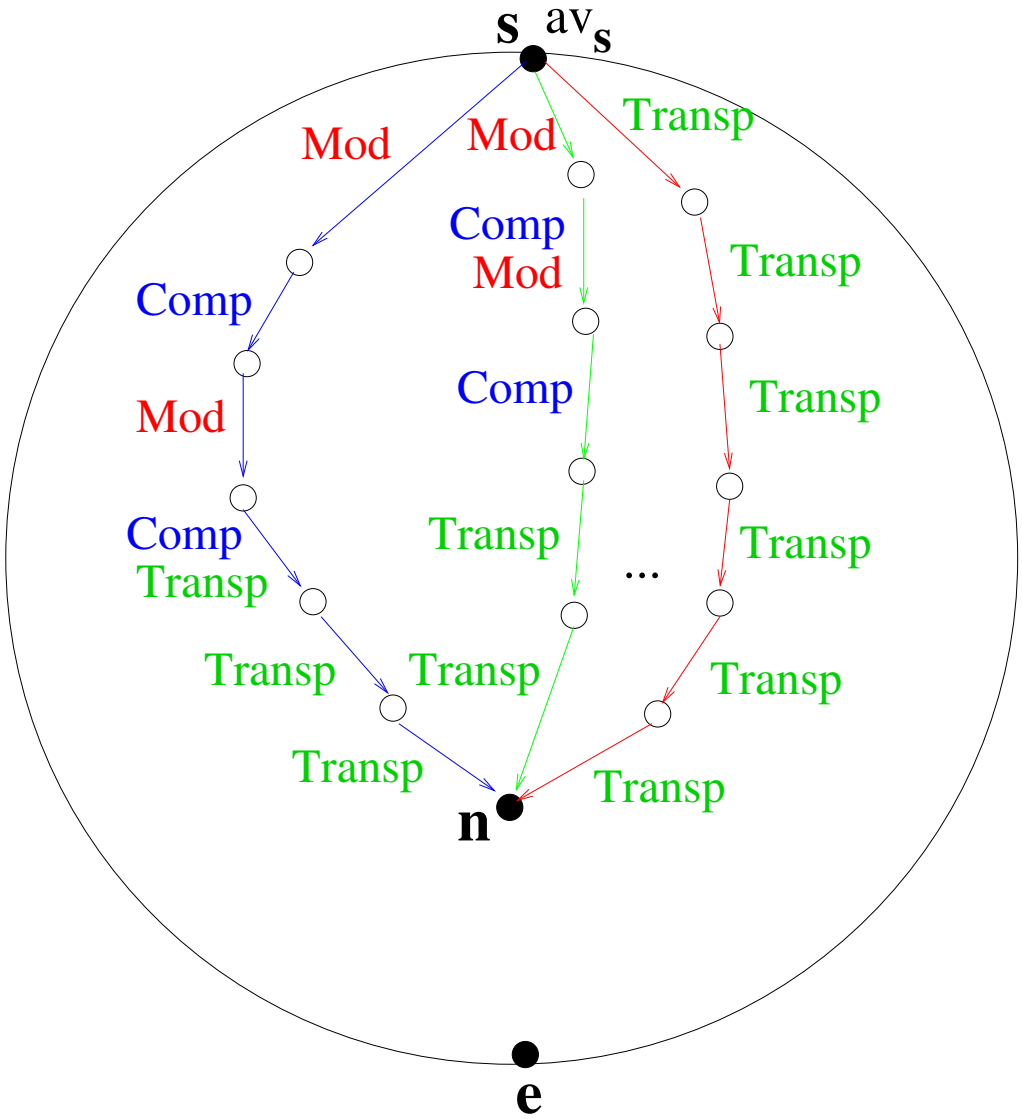
...defined formally:

Definition 7.10.1.22 (Availability)

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be a flow graph, t a term, and $av_{\mathbf{s}} \in \mathbb{B}$ the availability information for t at $\mathbf{s}$ ensured by the calling context of G. Then:

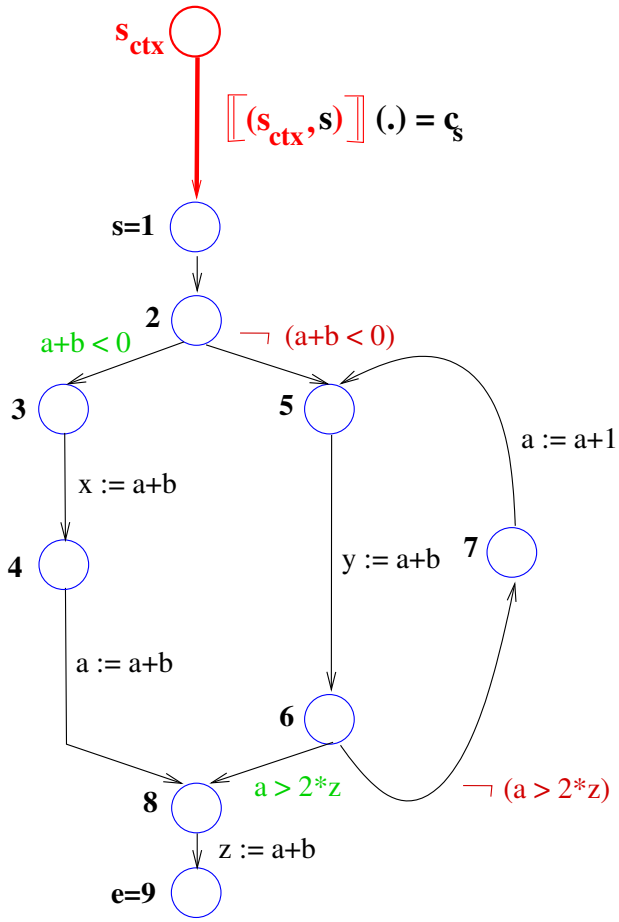
$$Available^t(n) \iff_{df} \begin{cases} av_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \forall\, p \in \mathbf{P}[\mathbf{s}, n].\ (av^t_{\mathbf{s}} \wedge Transp^{t\,\forall}_{p}) \ \vee & \\ \qquad \exists\, i \leq \lambda_p.\ Comp^t_{p_i} \wedge Transp^{t\,\forall}_{p[i,\lambda_p]} & \text{otherwise} \end{cases}$$

Illustrating the Essence of Definition 7.10.1.22

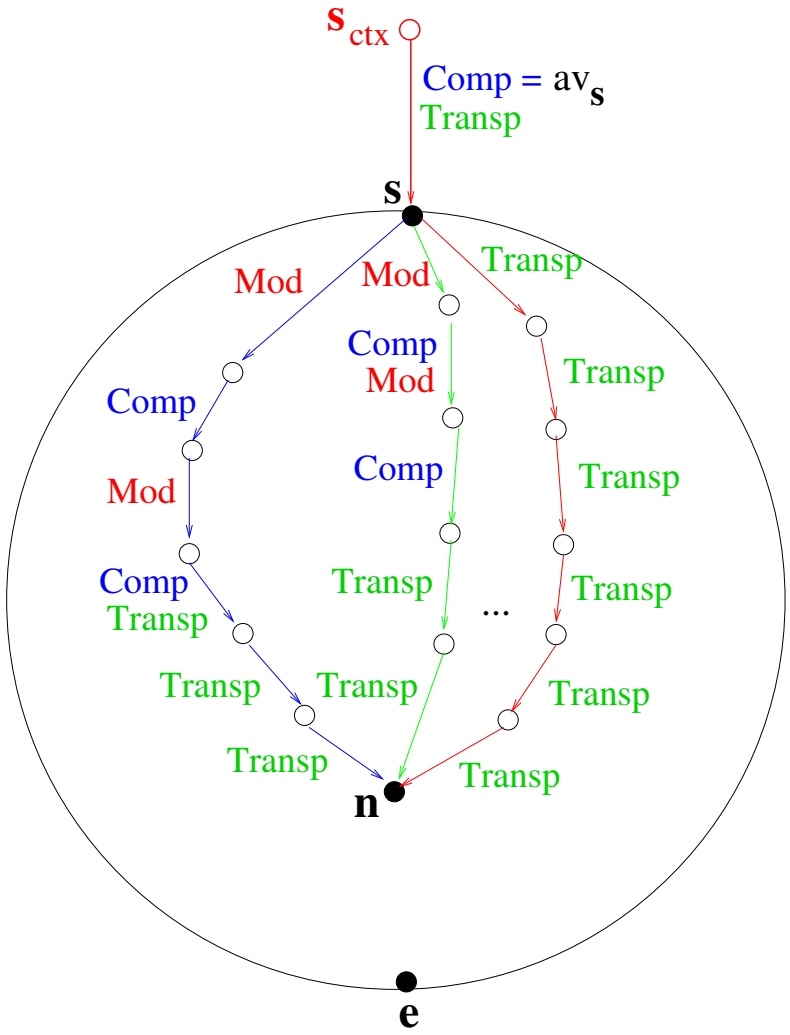


Context Edges

...allow a simpler case-free definition of availability:



Using Context Edges



...availability can be defined without cases:

$$\forall\, n \in N \backslash \{\mathbf{s}_{ctx}\}.\ \mathit{Available}^t(n) \iff_{df} \forall\, p \in \mathbf{P}[\mathbf{s}_{ctx}, n].\ \exists\, i \leq \lambda_p.\ \mathit{Comp}^t_{p_i} \wedge \mathit{Transp}^{t\,\forall}_{p[i,\lambda_p]}$$

Closing the Final Proof Gap

Theorem 7.10.1.23 (Soundness and Completeness)

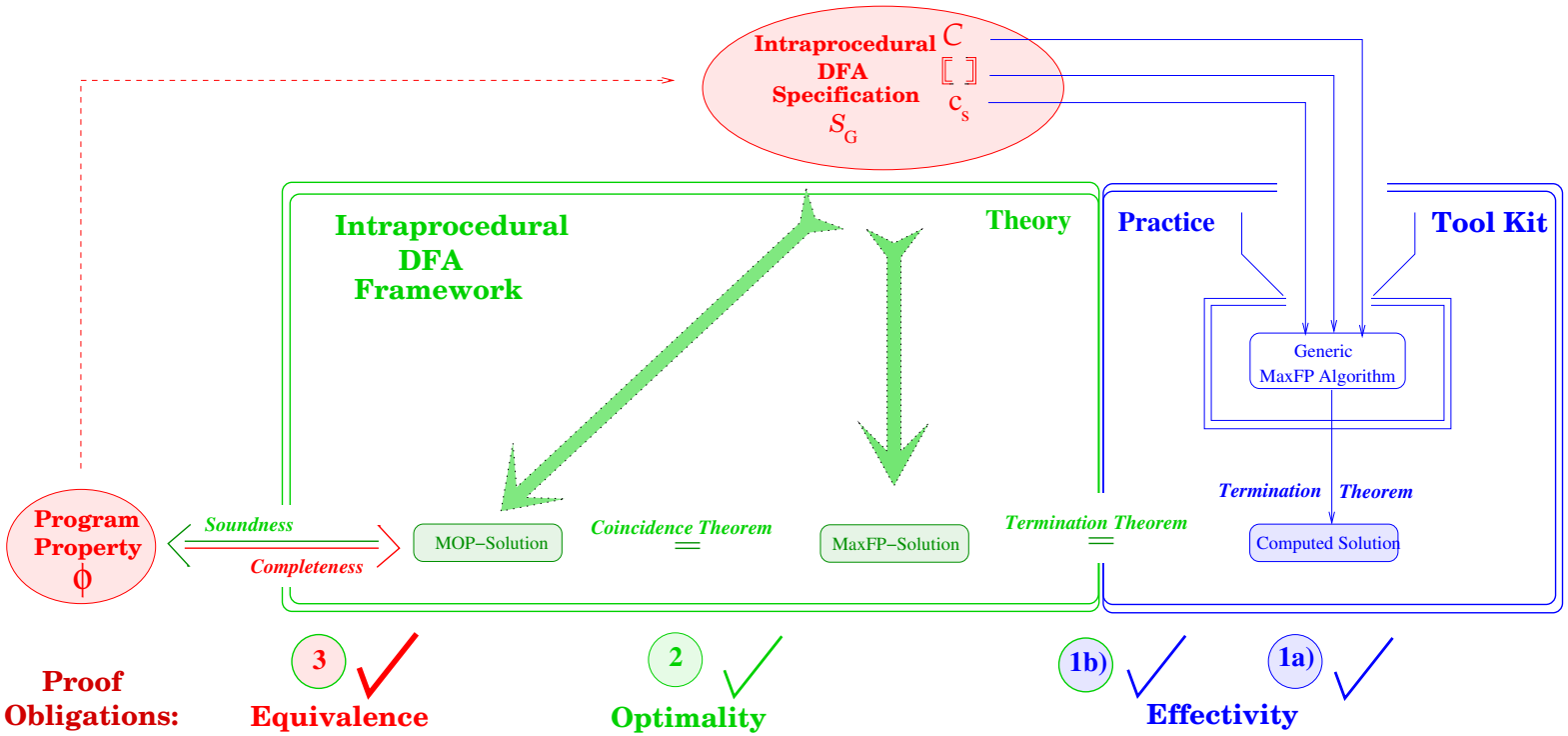
Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be a flow graph, t an expression, $av_{\mathbf{s}} \in \mathbb{IB}$ the availability information for t at $\mathbf{s}$ ensured by the calling context of G, and let $[\![\ ]\!]^{MOP}_{\mathcal{S}^{av,t}_G}$ be the *MOP* semantics of G for the DFA specification $\mathcal{S}^{av,t}_G = (\widehat{\mathbb{IB}}, [\![\ ]\!]^t_{av}, av_{\mathbf{s}}, fw)$.

Then:

$$\forall\, n \in N.\ Available^t(n) \iff [\![\, n \,]\!]^{MOP}_{\mathcal{S}^{av,t}_G}$$

Gap Closed: Soundness and Completeness

...for the *MOP* view of $\mathcal{S}_G^{av,t}$ for term availability proven:



Homework

1. What does soundness and completeness mean
2. How can soundness and completeness be proven

...for the *JOP* view of $\mathcal{S}_G^{av,t}$ for term availability?

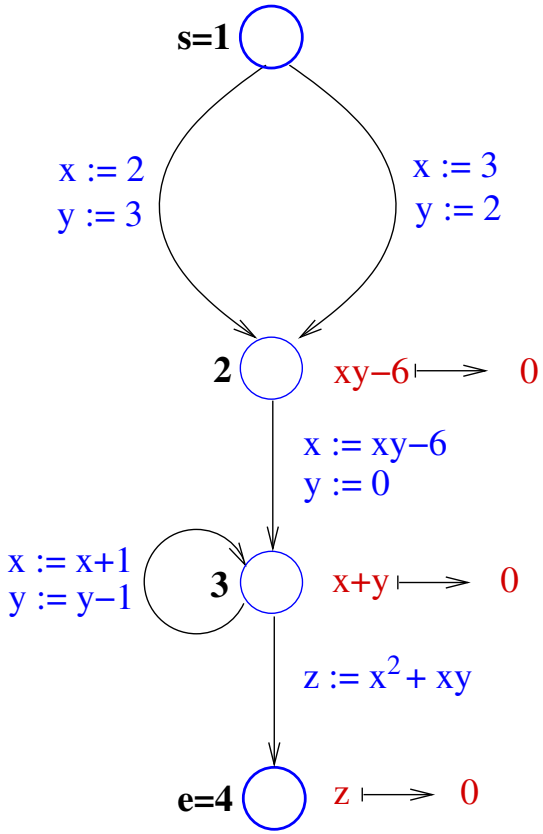
Kapitel 7.10.2

Monotone DFA: Einfache Konstanten

Intuitively

...a term is a constant of value c at a node, if, no matter which path is taken from the entry of the program to that node, the evaluation of this term at the node yields value c.

Illustration:



Example by Markus Müller-Olm, Helmut Seidl (SAS 2002)

Constant Propagation and Folding

...terms of a constant value can be replaced at compile time by this value effectively moving computational effort from the run time of a program to its compile time improving its run time performance, a so-called program optimization known as constant propagation and folding.

Unfortunately, there is no algorithm which always succeeds in determining if a term is a constant of some value at a node or not.

Undecidability of Constant Propagation

Theorem 7.10.2.1 (Undecidability, Reif&Lewis 1977)

In the arithmetic domain, the problem of discovering all text expressions covered by constant signs is undecidable.

(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)

Proof Sketch of Theorem 7.10.2.1 (1)

The proof of Theorem 7.10.2.1 works by reducing Hilbert's 10th problem to the problem of discovering all text expressions covered by constant signs:

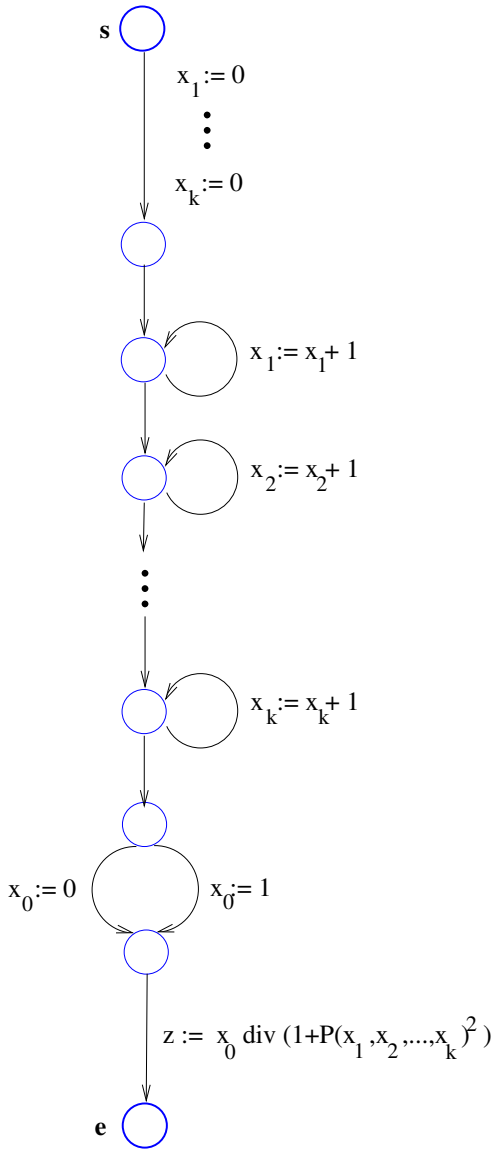
- Hilbert's 10th Problem

  Let $\{x_1, \ldots, x_k\}$ be a set of variables, $k > 5$, and let $P(x_1, \ldots, x_k)$ be a (multivariate) polynomial.

  It is not decidable, if $P(x_1, \ldots, x_k)$ has a root in the natural numbers (Matijasevic 1970).

Proof Sketch of Theorem 7.10.2.1 (2)

...consider program G given by its below flow graph:



Proof Sketch of Theorem 7.10.2.1 (3)

Then: Proving the equivalence

P has no root in the natural numbers iff

z is of a constant value at node $\mathbf{e}$ of G

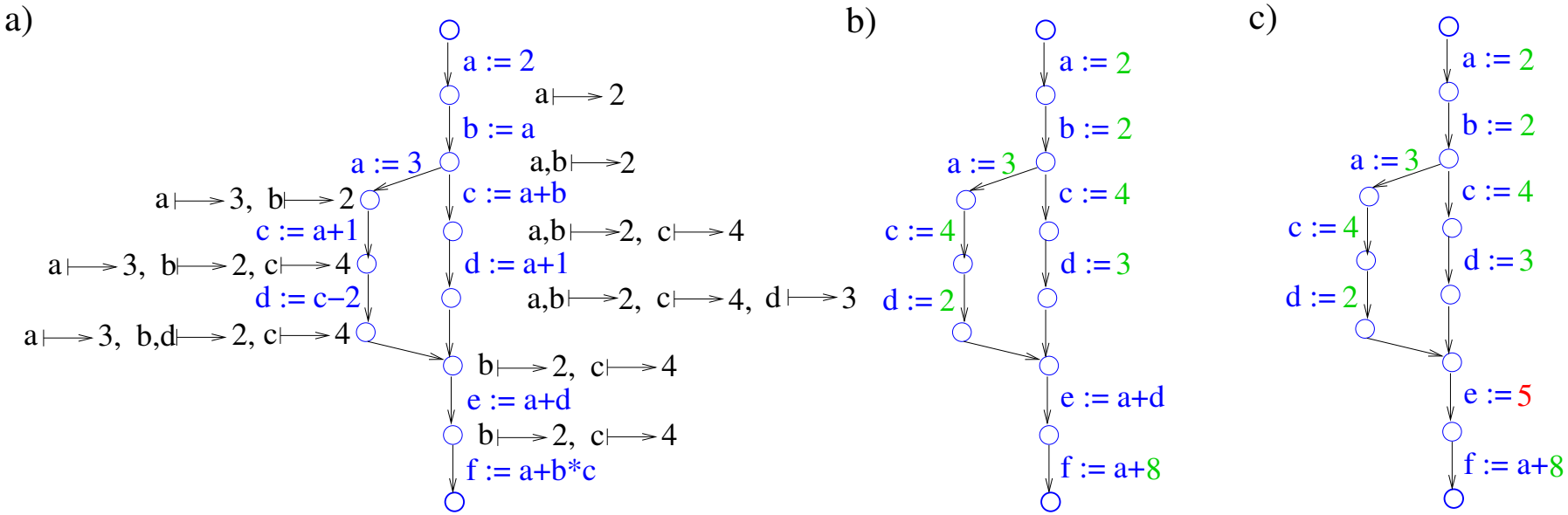
completes the proof. □

Simple Constants

...due to this negative result, in practice simpler decidable versions of the constant propagation and folding problem are considered, one of which is the class of so-called simple constants.

Informally, a term is a simple constant at a node, if every operand of the term has a unique constant value at this node, no matter, which path is taken from the entry of the program to the node.

Illustrating Simple Constants (1)



Note:

- All terms except of $a + d$ and $a + 8$ are simple constants (Figure b)).
- $a + d$ is a constant of value 5 but not a simple constant (Figure c)).

Illustrating Simple Constants (2)

- None of the (none-trivial) terms in the initial example of Müller-Olm and Seidl is a simple constant.
- $a + d$ as well as all terms in the example of Müller-Olm and Seidl can be detected to be constants by more sophisticated (and in the latter case computationally considerably more complex) constant propagation algorithms (cf. course notes of LVA 185.A04 Optimizing Compilers for details).

...computing simple constants is

- a canonical example of a monotonic (non distributive) DFA problem.
- an example of an incomplete analysis algorithm, which fails to dectect many terms to be constant, which could be detected so, but which is efficient w/ still useful results for optimization: Trading completeness for efficiency!

Computing Simple Constants: Preliminaries

...from data domains to DFA lattices.

Let $\mathbb{ID}$ be the

- data domain of interest (e.g., the set of natural numbers $\mathbb{IN}$, the set of integers $\mathbb{Z}$, the set of Boolean truth values $\mathbb{IB}$, etc.) with a distinguished element $\bot$ representing the value *undefined*.

We extend $\mathbb{ID}$ by adding

- a new element $\top$ not in $\mathbb{ID}$, i.e., $\top \notin \mathbb{ID}$

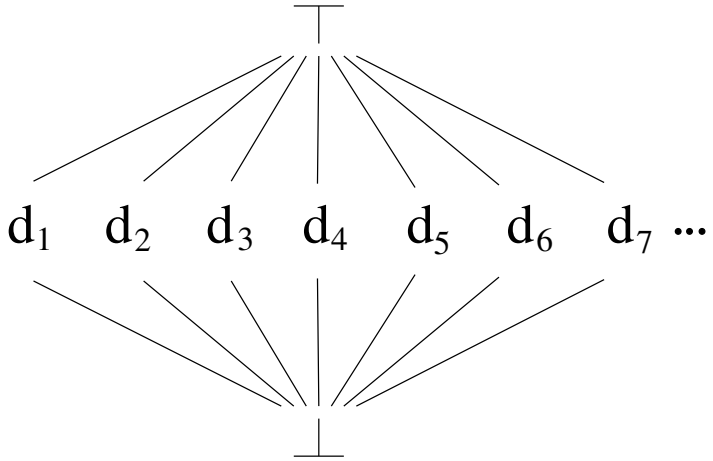
...and denote the extended domain by

- $\mathbb{ID}' =_{df} \mathbb{ID} \cup \{\top\}$.

Note: Assuming $\bot$ an element of the underlying data domain, whereas $\top$ not, might appear arbitrary. It is motivated by the fact that data types implemented on machines often contain a representation considered the undefined value of the data type.

Constructing DFA Lattices

...given an extended data domain $\mathbb{D}'$, we construct the flat lattice $\mathcal{FL}_{\mathbb{D}'}$ (cf. Appendix A.4)



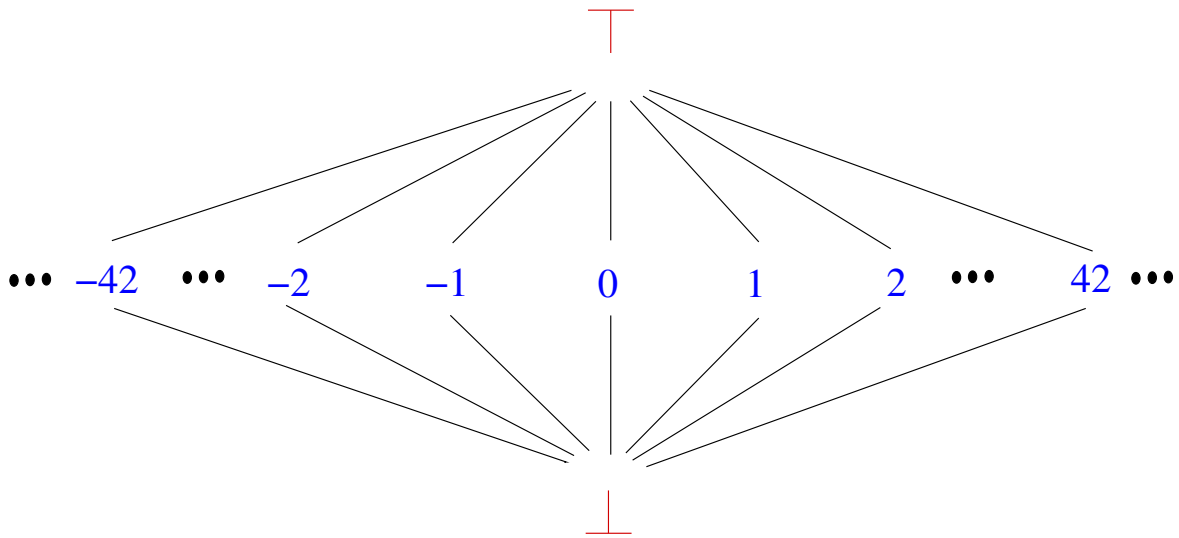
which is the basic DFA lattice of the DFA analysis for simple constants.

Intuitively

- $\top$ represents complete but inconsistent information.
- d_i, $i \geq 1$, represents accurate information.
- $\bot$ represents no information, the 'empty' information.

The Basic DFA Lattice over $\mathbb{Z}$

...is given by $\mathcal{FL}_{\mathbf{Z}}$:



...which is used for computing the class of simple constants over $\mathbb{Z}$.

Abstract Program States: DFA States

Definition 7.10.2.2 (DFA States)

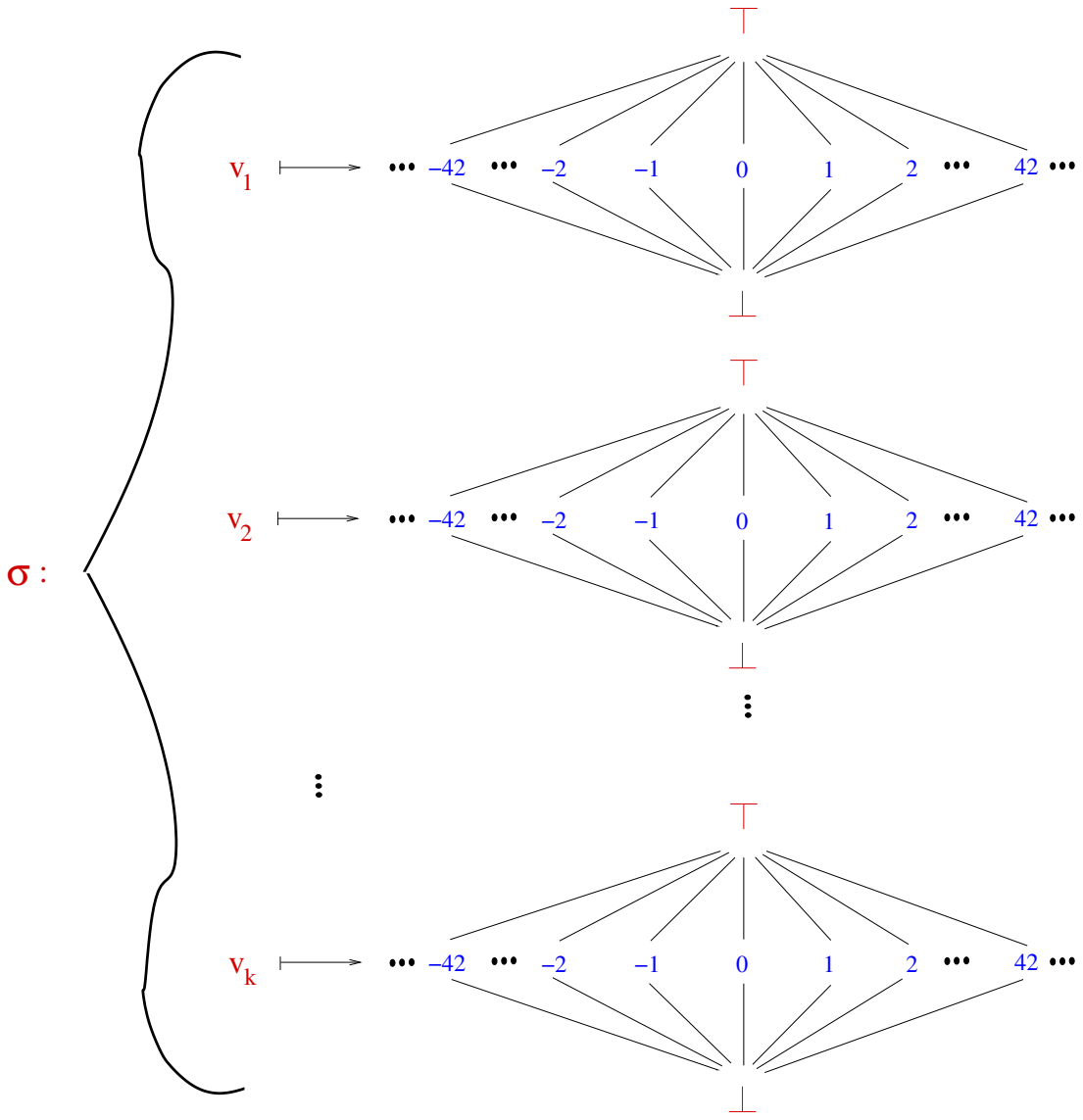
1. A DFA state is a total mapping $\sigma : \mathbf{V} \to \mathbb{ID}'$, which maps every variable to a datum $d \in \mathbb{ID}'$.
2. The set of all DFA states is defined by

$$\Sigma' =_{df} \{ \sigma \mid \sigma : \mathbf{V} \to \mathbb{ID}' \}.$$

3. $\sigma_\bot$ and $\sigma_\top$ denote two distinguished DFA states of Σ', which are defined by:

$$\forall\, v \in \mathbf{V}.\ \sigma_\bot(v) = \bot,\ \sigma_\top(v) = \top.$$

Illustrating a DFA State σ over $\mathbb{Z}$



Initial DFA States

...for an initial DFA state, we require that no variable is mapped to the special value $\top$, i.e, we require to either have accurate information of the value of a variable, when entering a procedure, or no information at all. We define:

Definition 7.10.2.3 (Initial DFA States over ID′)

The set of initial DFA states is defined by

$$\Sigma'_{Init} =_{df} \{ \sigma \in \Sigma' \mid \forall v \in \mathbf{V}.\ \sigma(v) \neq \top \}$$

Extending the Interpretation

...of constant and operator symbols from ID to ID′.

Definition 7.10.2.4 (Extending the Interpretation)

Let $I =_{df} (\mathrm{ID}, I_0)$ be an interpretation of constant and operator symbols over the data domain ID.

Then $I' =_{df} (\mathrm{ID}', I_0')$ is an interpretation over ID′ which extends I by defining

- $I_0'(c) =_{df} I_0(c)$ for every constant symbol $c \in \mathbf{C}$
- $I_0'(op) : \mathrm{ID}'^k \to \mathrm{ID}'$ for every k-ary operator symbol $op \in \mathbf{O}$:

$$\forall (d_1, \ldots, d_k) \in \mathrm{ID}'^k.\ I_0'(op)(d_1, \ldots, d_k) =_{df}$$

$$\begin{cases} I_0(op)(d_1, \ldots, d_k) & \text{if } d_i = \bot \text{ for some } 1 \leq i \leq k, \text{ or} \\ & \quad d_j \neq \top,\ 1 \leq j \leq k \\ \top & \text{if } d_i \neq \bot,\ 1 \leq i \leq k, \text{ and} \\ & \quad d_j = \top \text{ for some } 1 \leq j \leq k \end{cases}$$

The Abstract Term Semantics over ID'

Definition 7.10.2.5 (Abstract Term Semantics)

The abstract semantics of terms $t \in \mathbf{T}$ is defined by the evaluation function

$$\mathcal{E}\colon \mathbf{T} \to (\Sigma' \to \mathrm{ID}')$$

defined by

$$\forall t \in \mathbf{T}\ \forall \sigma \in \Sigma'.\ \mathcal{E}(t)(\sigma) =_{df} \begin{cases} \sigma(x) & \text{if } t \equiv x \in \mathbf{V} \\ I'_0(c) & \text{if } t \equiv c \in \mathbf{C} \\ I'_0(op)(\mathcal{E}(t_1)(\sigma), \ldots, \mathcal{E}(t_k)(\sigma)) & \\ & \text{if } t \equiv (op, t_1, \ldots, t_k) \end{cases}$$

The Abstract Instruction Semantics

Definition 7.10.2.6 (Abstract Instruction Semantics)

The abstract semantics of

- an assignment instruction $\iota \equiv x := t$ is given by the state transformer $\theta_\iota \colon \Sigma' \to \Sigma'$ defined by

$$\forall\, \sigma \in \Sigma'\ \forall\, y \in \mathbf{V}.\ \theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{if } y = x \\ \sigma(y) & \text{otherwise} \end{cases}$$

- the empty instruction $\iota \equiv \mathit{skip}$ and a condition $\iota \equiv \mathit{cond}$ is given by the identical state transformer $\mathit{Id}_{\Sigma'}$, i.e., $\theta_\iota =_{df} \mathit{Id}_{\Sigma'}$ with $\mathit{Id}_{\Sigma'} \colon \Sigma' \to \Sigma'$ defined by $\forall\, \sigma \in \Sigma'.\ \mathit{Id}_{\Sigma'}(\sigma) =_{df} \sigma$.

Note: Executing *skip* and evaluating conditions do not have side effects.

The DFA Lattice for Simple Constants

...the set of DFA states together with the pointwise ordering of states, $\sqsubseteq_{\Sigma'}$, forms a complete lattice (cf. Appendix A.4):

$$\forall\, \sigma, \sigma' \in \Sigma'.\ \sigma \sqsubseteq_{\Sigma'} \sigma' \ \text{ iff } \ \forall\, v \in \mathbf{V}.\ \sigma(v) \sqsubseteq_{\mathcal{FL}_{\mathbb{D}'}} \sigma'(v)$$

Lemma 7.10.2.7 (Lattice of DFA States)

$\widehat{\Sigma'} =_{df} (\Sigma', \sqcap_{\Sigma'}, \sqcup_{\Sigma'}, \sqsubseteq_{\Sigma'}, \sigma_\bot, \sigma_\top)$ is a complete lattice with

- least element $\sigma_\bot$,
- greatest element $\sigma_\top$,
- pointwise meet $\sqcap_{\Sigma'}$ and join $\sqcup_{\Sigma'}$ as meet and join operation, respectively.

Simple Constants: Specifying the DFA

DFA Specification

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $(\Sigma', \sqcap_{\Sigma'}, \sqcup_{\Sigma'}, \sqsubseteq_{\Sigma'}, \sigma_\bot, \sigma_\top) = \widehat{\Sigma}'$
  with Σ' set of DFA states over $\mathbb{Z}$.
- DFA semantics
  $[\![\ ]\!]_{sc} : E \to (\Sigma' \to \Sigma')$ where $\forall\, e \in E.\ [\![e]\!]_{sc} =_{df} \theta'_{\iota_e}$
- Start assertion: $\sigma_{\mathbf{s}} \in \Sigma'_{Init}$

Simple Constants Specification

- Specification: $\mathcal{S}^{sc}_G = (\widehat{\Sigma}', [\![\ ]\!]_{sc}, \sigma_{\mathbf{s}})$

Simple Constants: Fulfilling the Proof Oblig.

Lemma 7.10.2.8 (Chain Conditions)

$\widehat{\Sigma}'$ satisfies the descending and ascending chain condition.

Note. The set of variables occurring in a program is finite.

Lemma 7.10.2.9 (Monotonicity)

$[\![\ ]\!]_{sc}$ is monotonic.

Lemma 7.10.2.10 (Non-Distributivity/Addititivity)

$[\![\ ]\!]_{sc}$ is (in general) not distributive and not additive.

Simple Constants: Collecting the Guarantees

...on termination, conservativity.

Theorem 7.10.2.11 (Termination)

Applied to $\mathcal{S}^{sc}_G = (\widehat{\Sigma}', [\![\ ]\!]_{sc}, \sigma_{\mathbf{s}})$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}^{sc}_G$.

Proof. Immediately with Lemma 7.10.2.8 Lemma 7.10.2.9, and Termination Theorem 7.6.2.1.

Theorem 7.10.2.12 (Safety, Conservativity)

Applied to $\mathcal{S}^{sc}_G = (\widehat{\Sigma}', [\![\ ]\!]_{sc}, \sigma_{\mathbf{s}})$, Algorithm 7.6.1.1 is *MOP*/*JOP* conservative for $\mathcal{S}^{sc}_G$ (i.e., it terminates with a lower (upper) approximation of the *MOP*/*JOP* semantics of $\mathcal{S}^{sc}_G$, resp.).

Proof. Immediately with Lemma 7.10.2.8, Lemma 7.10.2.9, Safety Theorem 7.7.1, and Termination Theorem 7.6.2.1.

Simple Constants: Negative Result

...on tightness.

Theorem 7.10.2.13 (Non-Tightness)

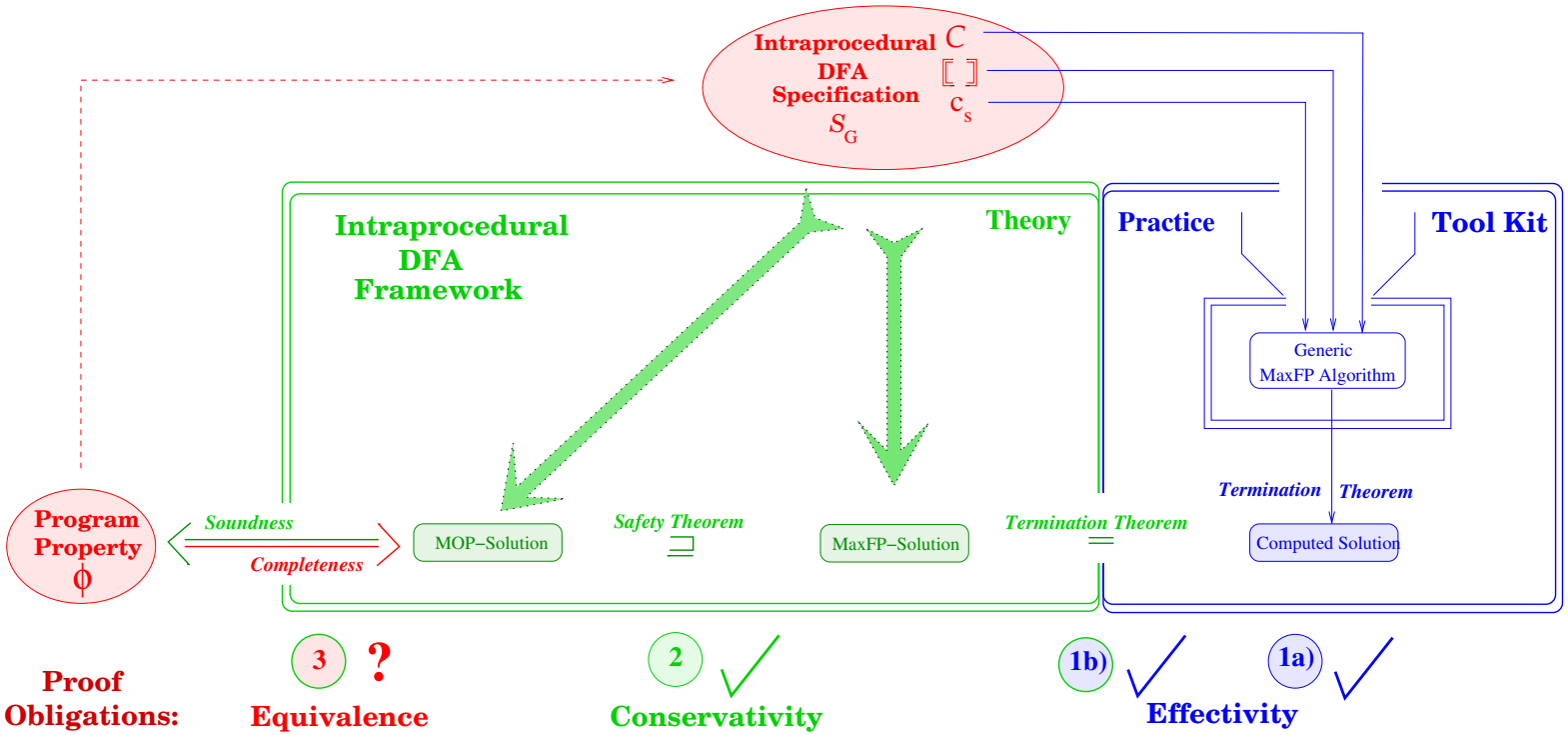
Applied to $\mathcal{S}^{sc}_G = (\widehat{\Sigma}', [\![\ ]\!]_{sc}, \sigma_{\mathbf{s}})$, Algorithm 7.6.1.1 is in general not *MOP*/*JOP* tight for $\mathcal{S}^{sc}_G$ (i.e., it terminates with a proper approximation of the *MOP*/*JOP* solution of $\mathcal{S}^{sc}_G$, respectively).

Proof. Immediately with Lemma 7.10.2.8, Lemma 7.10.2.9, Lemma 7.10.2.10, Coincidence Theorem 7.7.2, and Termination Theorem 7.6.2.1.

In closing: The *MaxFP*/*MinFP* solutions of $\mathcal{S}^{sc}_G$ are always safe approximations of the *MOP*/*JOP* solutions of $\mathcal{S}^{sc}_G$. In general, the operational *MOP*/*JOP* solutions of $\mathcal{S}^{sc}_G$ and their denotational *MaxFP*/*MinFP* counterparts do not coincide.

Simple Constants: Closing the Final Proof Gap

...proving soundness and completeness for the *MOP* view of $\mathcal{S}_G^{sc}$ for the simple constant property:



Simple Constants: Soundness, Completeness

...for the *MOP* semantics.

Theorem 7.10.2.14 (Soundness and Completeness)

The *MOP* semantics of $\mathcal{S}_G^{sc}$ is

1. sound and complete for variables.
2. sound but not complete for (non-trivial) terms (i.e., for terms containing at least one (non-unary) operator symbol).

...for Theorem 7.10.2.14(2), note that the *MOP* solution at every node can be considered a state, i.e., a map from variables to values, allowing an evaluation of terms.

Simple Constants: Soundness, Completeness

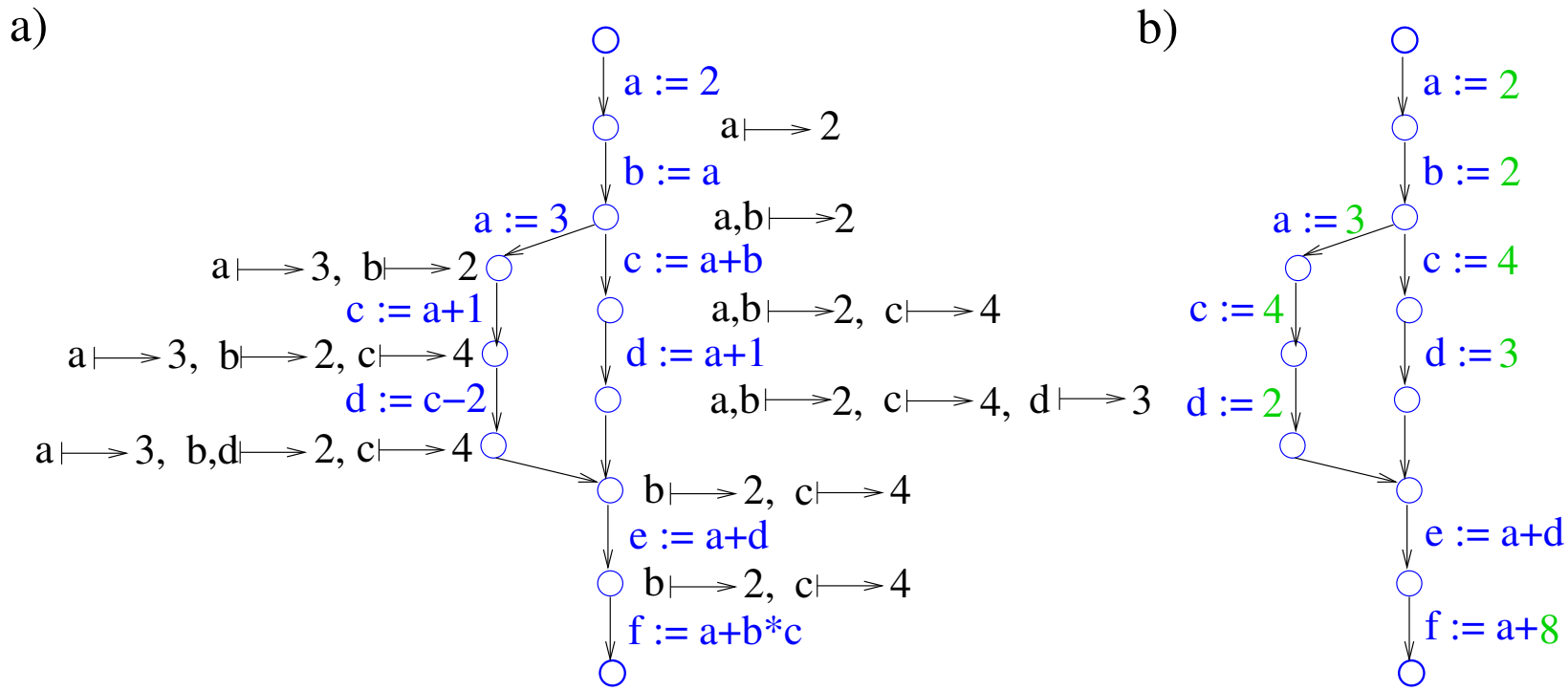
...for the *MaxFP* semantics.

Theorem 7.10.2.15 (Soundness and Completeness)

The *MaxFP* semantics of $\mathcal{S}_G^{sc}$ is sound but not complete (for both variables and terms).

...see course notes of LVA 185.A04 Optimizing Compilers for further details.

Simple Constants: Illustrating Example

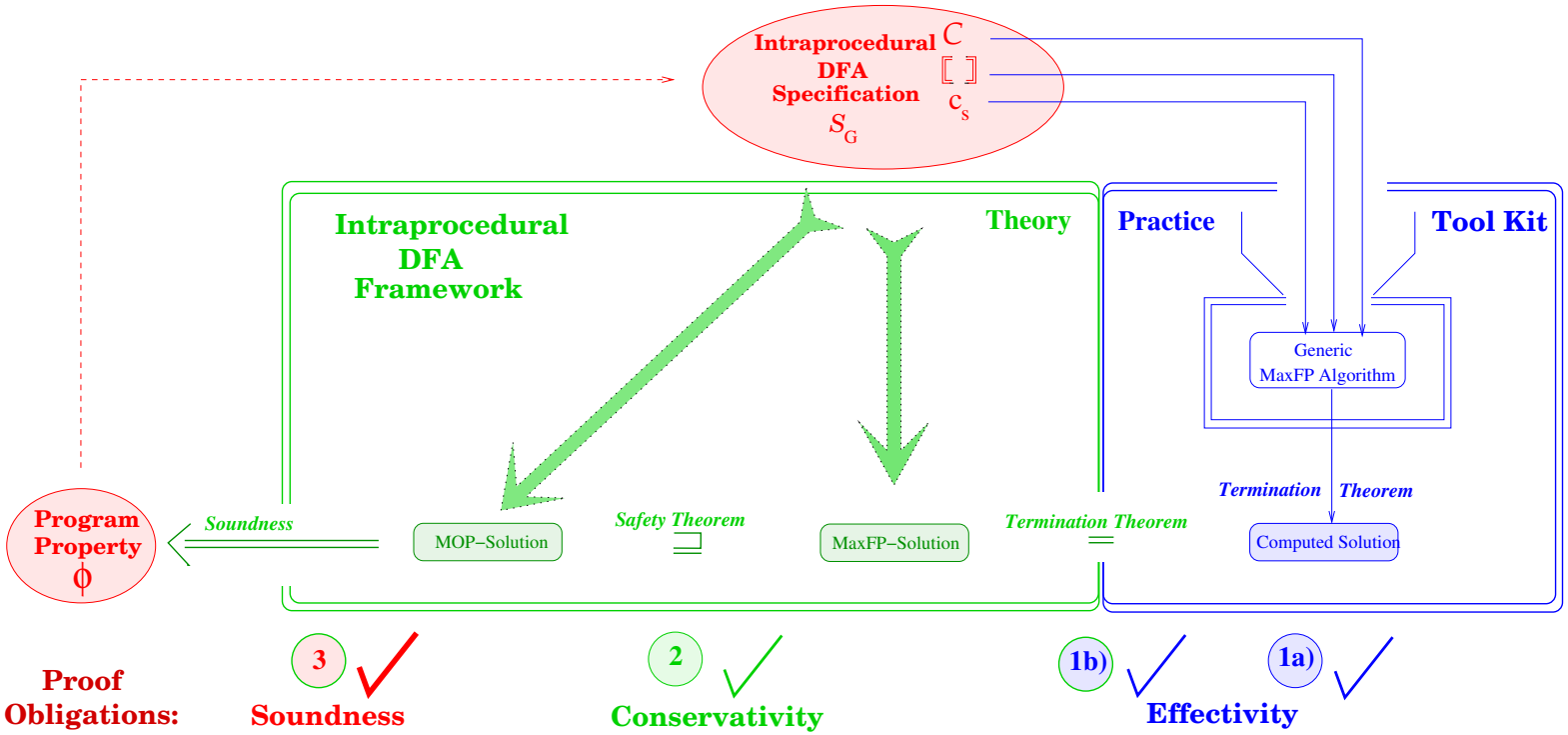


...all terms except of $a + d$ and $a + 8$ are simple constants.

Recall: $a + d$ is a constant of value 5 but not a simple constant; $a + 8$ is not a constant.

Gap Partially Closed: Soundness

...for the *MOP* view of $\mathcal{S}_G^{sc}$ for simple constants proven:



Homework

1. What does soundness and completeness mean
2. How can soundness and completeness be proven

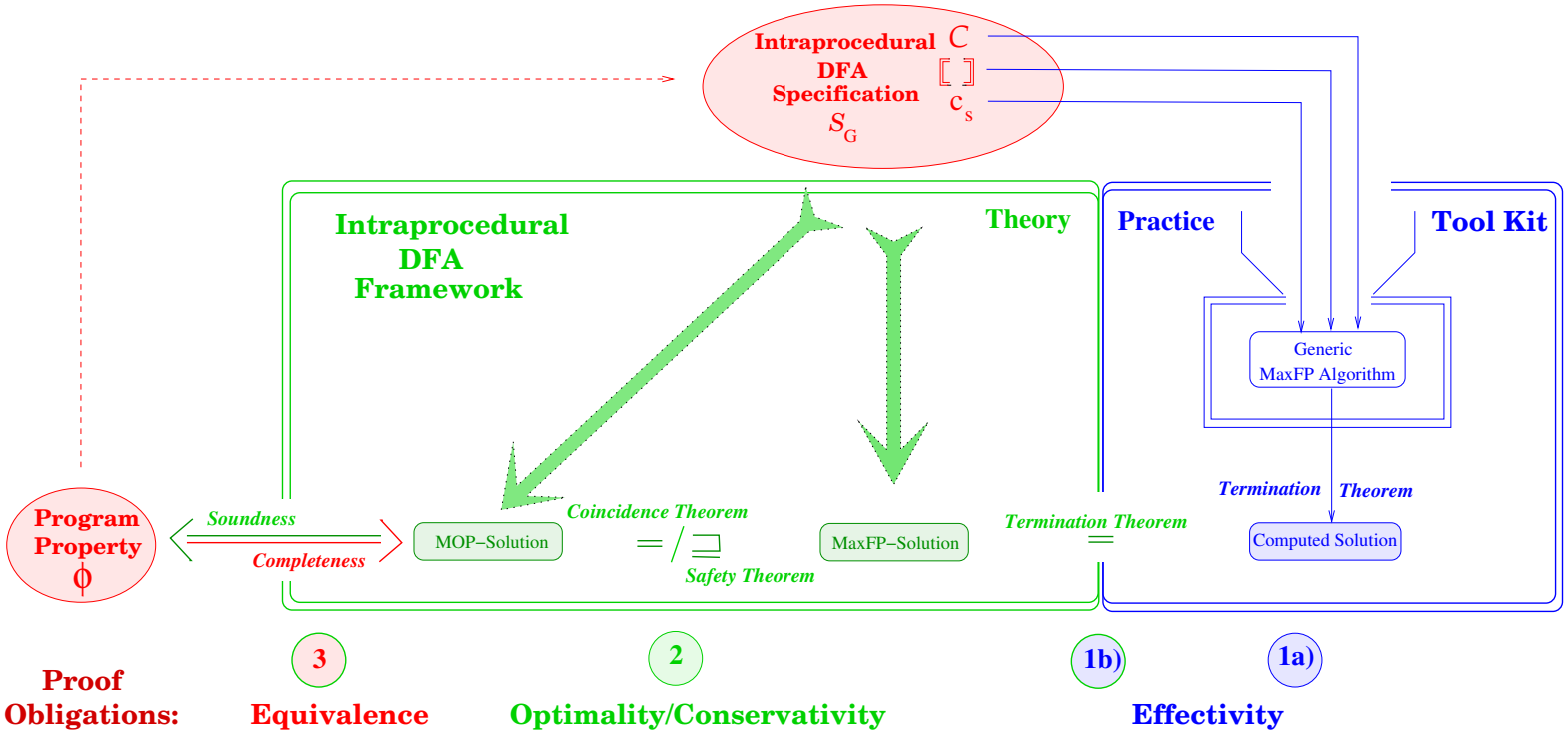
...for the *JOP* view of $\mathcal{S}_G^{sc}$ for the simple constants property?

Kapitel 7.11

Zusammenfassung, Ausblick

The Framework/Toolkit View

...of data flow analysis.



...reconsidered from the angle of correctness, accuracy, and the kind of ϕ.

Kinds of Properties: Must vs. May

...basically, we can distinguish two kinds of properties ϕ:

- Universally quantified (or must) properties $\phi^{\forall}$: $\phi^{\forall}$ holds at a node n, if it holds along all paths from $\mathbf{s}$ to n at n.
- Existentially quantified (or may) properties $\phi^{\exists}$: $\phi^{\exists}$ holds at a node n, if it holds along some paths from $\mathbf{s}$ to n at n.

Must-properties $\phi^{\forall}$ are related to the

- operational *MOP* semantics of a program and its computational denotational counterpart, the *MaxFP* semantics.

May-properties $\phi^{\exists}$ are related to the

- operational *JOP* semantics of a program and its computational denotational counterpart, the *MinFP* semantics.

Correctness and Accuracy

...essentially, there are two places where correctness and accuracy issues are handled in the framework/toolkit view of DFA:

Framework/Toolkit **internally**: captured by

- **Safety** $\rightsquigarrow$ Conservativity
- **Coincidence** $\rightsquigarrow$ Tightness

...relating *MaxFP*/*MinFP* and *MOP*/*JOP* solution, respectively.

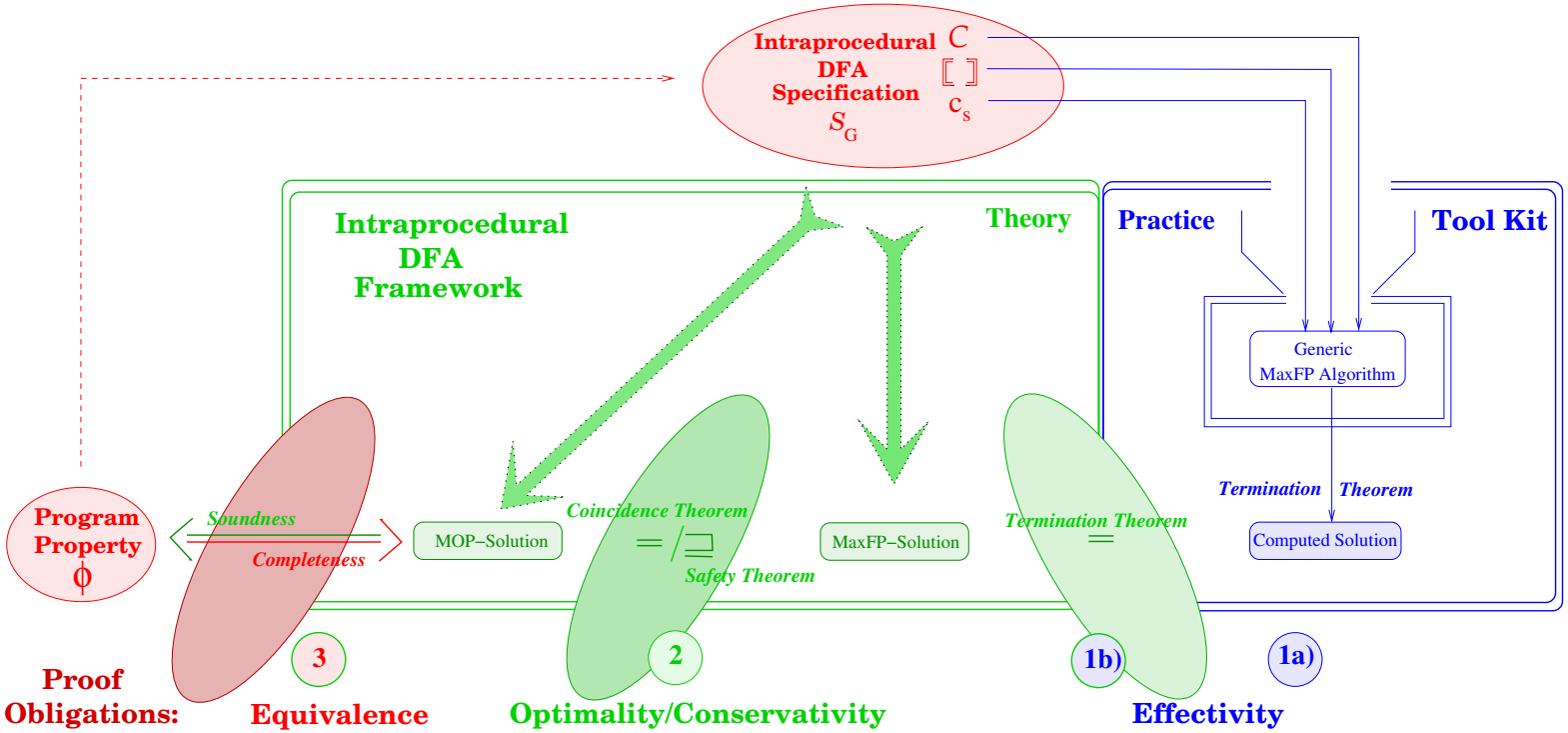
Framework/Toolkit **externally**: captured by

- **Soundness** $\rightsquigarrow$ No false positives
- **Completeness** $\rightsquigarrow$ No false negatives

...relating *MOP*/*JOP* solution and $\phi^{\forall}/\phi^{\exists}$, respectively.

Illustrating

...the places of internal and external correctness and accuracy handling:

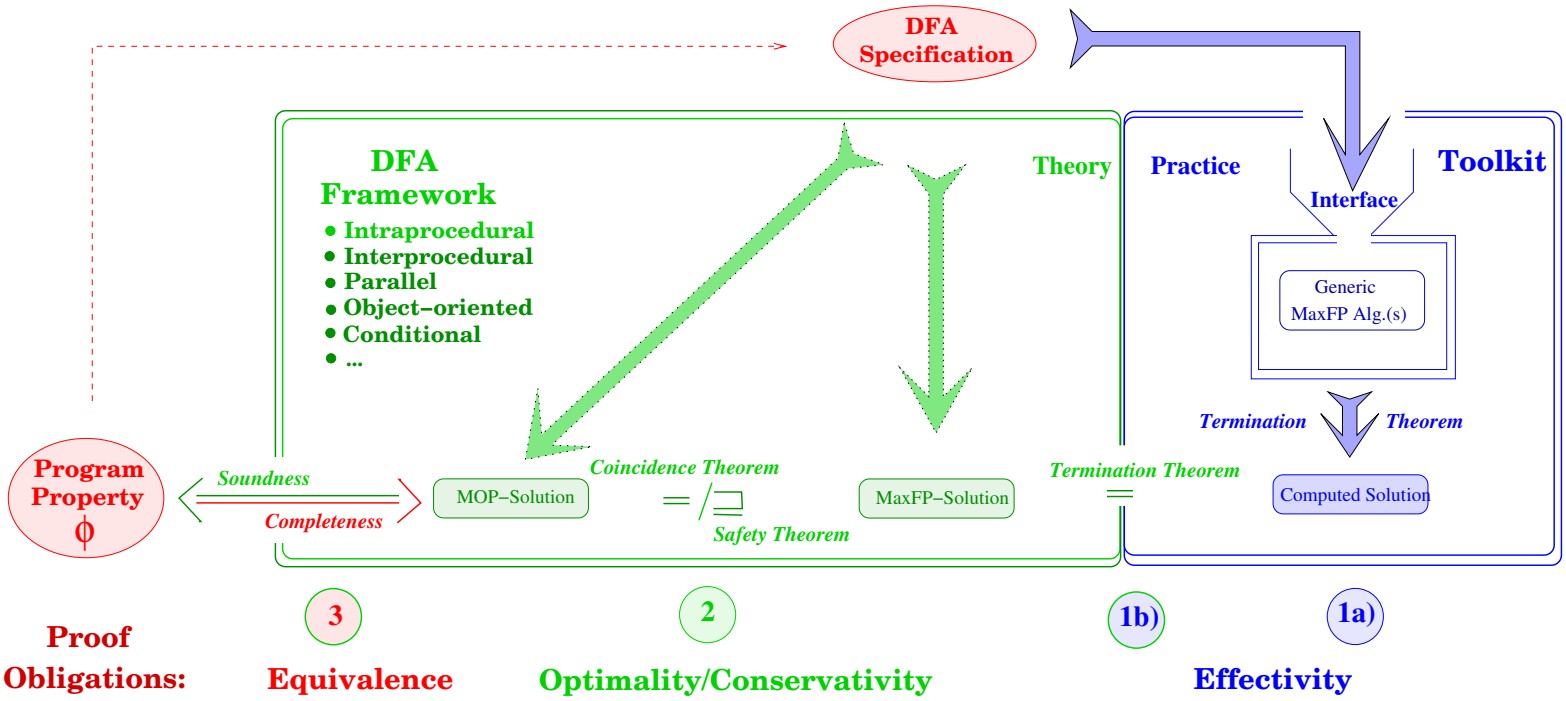


Looking ahead: The Uniform View of DFA

...in the course of this lecture course (and of LVA 185.A04 Optimizing Compilers), we will see:

- The Framework and Toolkit View of DFA

is achievable beyond the base case of intraprocedural DFA providing a uniform view of DFA:



Kapitel 7.12

Literaturverzeichnis, Leseempfehlungen

Further Reading for Chapter 7 (1)

Textbook Representations

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)

Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.

Further Reading for Chapter 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; 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)

Further Reading for Chapter 7 (3)

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)

Further Reading for Chapter 7 (4)

Pioneering, Groundbreaking Articles


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.


John B. Kam, Jeffrey D. Ullman. *Global Data Flow Analysis and Iterative Algorithms*. Journal of the ACM 23:158-171, 1976.

Further Reading for Chapter 7 (5)


John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.

Frameworks and Toolkits


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.

Further Reading for Chapter 7 (6)


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.


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.

Further Reading for Chapter 7 (7)

Solving Equation Systems, Computing Fixed Points


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.

Further Reading for Chapter 7 (8)

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.

Flow Graph Pragmatics

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.

Further Reading for Chapter 7 (9)

Miscellaneous

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.

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)

Further Reading for Chapter 7 (10)


Yuri V. Matijasevic. *Enumerable Sets are Diophantine (In Russian)*. Dodl. Akad. Nauk SSSR 191, 279-282, 1970.


Yuri V. Matijasevic. *What Should We Do Having Proved a Decision Problem to be Unsolvable?* Algorithms in Modern Mathematics and Computer Science 1979:441-448, 1979.


Yuri V. Matijasevic. *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 8

Reverse Datenflussanalyse

Motivation: DFA vs. reverse DFA (1)

...intuitiv:

Datenflussanalyse zielt

- auf die Berechnung eines stärkst möglichen Datenflussfakts für jede Programmstelle relativ zu einer gegebenen Startzusicherung.

Reverse Datenflussanalyse zielt

- auf die Berechnung eines schwächst möglichen Datenflussfakts für jede Programmstelle relativ zu einem angefragten Datenflussfakt an einer bestimmten Programmstelle, dem sog. Anfrageknoten, so dass der angefragte Datenflussfakt am Anfrageknoten gültig ist.

  Von besonderem Interesse ist dabei der schwächst mögliche Datenflussfakt am Startknoten, der die Gültigkeit des angefragten Datenflussfakts am Anfrageknoten garantiert.

Motivation: DFA vs. reverse DFA (2)

...die reversen Gegenstücke der

- Schnitt/Vereinigung-über-alle-Pfade (*SUP*/*VUP*) Semantiken einer lokalen abstrakten DFA-Semantik $[\![\ ]\!]$

sind daher die

- reversen Vereinigung/Schnitt-über-alle-Pfade (*RVUP*/*RSUP*) Semantiken

der von $[\![\ ]\!]$ induzierten reversen lokalen abstrakten DFA-Semantik $[\![\ ]\!]_R$ sowie die

- berechenbaren denotationellen reversen Entsprechungen der operationellen reversen Vereinigung/Schnitt-über-alle-Pfade Semantiken, die *RMinFP*- und *RMaxFP*-Semantiken.

Kapitel 8.1

Vorbereitung

DFA-Fehlschlagsverbandserweiterung

...sei $(\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ eine DFA-Spezifikation, *failure* ein neues Element nicht in $\mathcal{C}$ und $\mathcal{C}_f =_{df} \mathcal{C} \dot{\cup} \{failure\}$.

Definition 8.1.1 (Fehlschlagserweiterter DFA-Verb.)

Die *failure*-Erweiterung von $\widehat{\mathcal{C}}$ ist der vollständige Verband $\widehat{\mathcal{C}}_f$ definiert durch:

$$\widehat{\mathcal{C}}_f =_{df} (\mathcal{C}_f, \sqsubseteq_f, \sqcap_f, \sqcup_f, \bot, failure)$$

mit *failure* größtes Element in $\widehat{\mathcal{C}}_f$ ist, d.h.:

1. $\forall\, c \in \mathcal{C}_f.\ c \sqsubseteq_f failure$
2. $\forall\, c, c' \in \mathcal{C}.\ c \sqsubseteq_f c'$ gdw $c \sqsubseteq c'$

Beachte:

- Mit $\sqsubseteq_f$ sind auch $\sqcap_f$ und $\sqcup_f$ eindeutig festgelegt.
- Das Element *failure* repräsentiert nicht erfüllbare, nicht zusicherbare DFA-Information.

Fehlschlagserweiterung lokaler DFA-Semantik

Definition 8.1.2 (Fehlschlagserweiterte DFA-Sem.)

Wir legen fest:

$$\forall\, c \in \mathcal{C}_f\ \forall\, e \in E.\ [\![\, e \,]\!]_f(c) =_{df} \begin{cases} [\![\, e \,]\!](c) & \text{falls } c \neq \mathit{failure} \\ \mathit{failure} & \text{sonst} \end{cases}$$

Erste Ergebnisse

Lemma 8.1.3

Der Verband $\widehat{\mathcal{C}_f}$ erfüllt die aufsteigende (absteigende) Kettenbedingung gdw der Verband $\widehat{\mathcal{C}}$ die aufsteigende (absteigende) Kettenbedingung erfüllt.

Lemma 8.1.4

1. $[\![\ ]\!]_f$ monoton gdw $[\![\ ]\!]$ monoton.
2. $[\![\ ]\!]_f$ distributiv gdw $[\![\ ]\!]$ distributiv.
3. $[\![\ ]\!]_f$ additiv gdw $[\![\ ]\!]$ additiv.

Kapitel 8.2

Induzierte reverse lokale DFA-Semantik

Induzierte reverse lokale DFA-Semantik

...sei $(\widehat{\mathcal{C}_f}, [\![\ ]\!]_f)$ eine fehlschlagserweiterte DFA-Spezifikation (ohne Startzusicherung).

Definition 8.2.1 (Reverse lokale abstrakte Semantik)

Die von $[\![\ ]\!]_f$ induzierte reverse lokale abstrakte Semantik

$$[\![\ ]\!]_R : E \to (\mathcal{C}_f \to \mathcal{C}_f)$$

ist definiert durch:

$$\forall\, e \in E\ \forall\, c \in \mathcal{C}_f.\ [\![\, e \,]\!]_R(c) =_{df} \bigsqcap \{\, c' \mid [\![\, e \,]\!]_f(c') \sqsupseteq c \,\}$$

Beachte: Die Gültigkeit von $[\![\, e \,]\!]_f(c') \sqsupseteq c$ bedeutet, dass c' am Eingang von e mindestens c oder eine größere DFA-Information am Ausgang von e garantiert.

Eigenschaften von $[\![\ ]\!]$, $[\![\ ]\!]_f$ und $[\![\ ]\!]_R$

...sei $[\![\ ]\!]_f$ die fehlschlagserweiterte lokale abstrakte DFA-Semantik zur lokalen abstrakten DFA-Semantik $[\![\ ]\!]$; $[\![\ ]\!]_R$ die induzierte reverse DFA-Semantik.

Lemma 8.2.2

1. $\forall\, e \in E.\ [\![\ e\ ]\!]_R$ ist wohldefiniert und monoton.
2. $\forall\, e \in E.\ [\![\ e\ ]\!]_R$ ist additiv, falls $[\![\ e\ ]\!]_f$ distributiv ist.

Lemma 8.2.3

1. $\forall\, e \in E.\ [\![\ e\ ]\!]_R \circ [\![\ e\ ]\!]_f \sqsubseteq Id_{C_f}$, falls $[\![\ e\ ]\!]$ monoton ist.
2. $\forall\, e \in E.\ [\![\ e\ ]\!]_f \circ [\![\ e\ ]\!]_R \sqsupseteq Id_{C_f}$, falls $[\![\ e\ ]\!]$ distributiv ist.

...in der Sprechweise der Theorie 'Abstrakter Interpretation':

- $[\![\ e\ ]\!]_f$ und $[\![\ e\ ]\!]_R$ bilden eine Galois-Verbindung.

Kapitel 8.3

Reverse DFA-Spezifikation

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 Datenflussfaktanfrage (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 der Anfrageknoten explizit dargestellt ist, wenn modellierungstechnisch nötig.

Wir definieren: Ist q

- gleich $\mathbf{s}'$ 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. $\mathit{pred}(\mathbf{q}) = \mathit{pred}(q)$ und $\mathit{succ}(\mathbf{q}) = \emptyset$.

Es gilt

...die Hinzunahme von $\mathbf{q}$ hat keinen Einfluss auf die

- *MOP*/*MaxFP*-Semantik
- *JOP*/*MinFP*-Semantik

irgendeines der ursprünglichen Knoten von G' (s. Korollar 8.3.2(1)).

Für $\mathbf{q}$ und q stimmen die

- *MOP*-Semantik
- *JOP*-Semantik

jeweils überein (s. Korollar 8.3.2(2)). Für die zugehörigen Fixpunktsemantiken (*MaxFP*, *MinFP*) gilt dies nicht.

Das folgende Lemma und Korollar fassen dies zusammen.

DFA-Zusammenhang von G' und G

Lemma 8.3.1

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 8.3.2

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 $X \in \{\mathit{MOP}, \mathit{MaxFP}, \mathit{JOP}, \mathit{MinFP}\}$, $Y \in \{\mathit{MOP}, \mathit{JOP}\}$.

...wobei $\mathbf{q} \in N$ die zu einem Anfrageknoten $q \in N'$ gehörige Kopie bezeichnet mit $\mathit{pred}_G(\mathbf{q}) = \mathit{pred}_{G'}(q)$ und $\mathit{succ}_G(\mathbf{q}) = \emptyset$.

Reverse DFA-Spezifikation und -Problem

...mit den vorherigen Festlegungen und Beobachtungen können wir definieren:

Definition 8.3.3 (Reverse DFA-Spezifikation)

Eine reverse DFA-Spezifikation zu einer DFA-Spezifikation $(\widehat{\mathcal{C}}, [\![\ ]\!])$ ist ein Tripel $(\widehat{\mathcal{C}}_f, [\![\ ]\!]_R, c_q)$ mit

- $\widehat{\mathcal{C}}_f =_{df} (\mathcal{C}_f, \sqsubseteq_f, \sqcap_f, \sqcup_f, \bot, \mathit{failure})$ DFA-Verbandserweiterung zu $\widehat{\mathcal{C}}$ (gemäß Definition 8.1.1).
- $[\![\ ]\!]_R : E \to (\mathcal{C}_f \to \mathcal{C}_f)$ induzierte reverse lokale DFA-Semantik (gemäß Definition 8.2.1).
- $c_q \in \mathcal{C}$ eine DFA-Anfrage.

Definition 8.3.4 (Reverses DFA-Problem)

Eine reverse DFA-Spezifikation legt ein reverses DFA-Problem fest.

Kapitel 8.4

Reverse operationelle globale DFA-Semantik

Kapitel 8.4.1

Reverse Aufsammelsemantik

Ausdehnung von $[\![\ ]\!]_R$ von Kanten auf Pfade

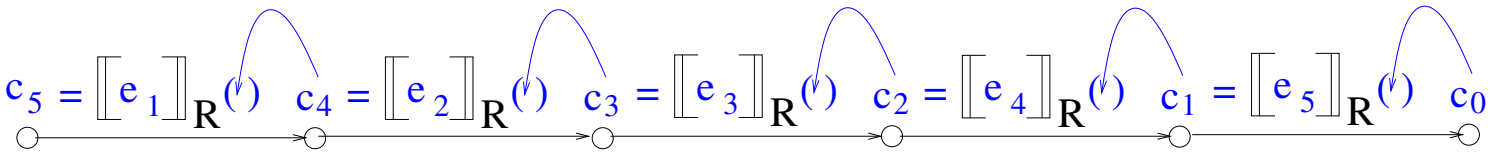
Sei $\mathcal{S}^R_G =_{df} (\widehat{\mathcal{C}}_f, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Definition 8.4.1.1 (Pfadausdehnung von $[\![\ ]\!]_R$)

Die Ausdehnung einer reversen (lokalen) DFA-Semantik auf Pfade $p = \langle e_1, \ldots, e_{q-1}, e_q \rangle$ ist definiert durch:

$$[\![\, p \,]\!]_R =_{df} \begin{cases} Id_{\mathcal{C}} & \text{falls } \lambda_p < 1 \\ [\![\, \langle e_1, \ldots, e_{q-1} \rangle \,]\!]_R \circ [\![\, e_q \,]\!]_R & \text{sonst} \end{cases}$$

Illustrating the extension of $[\![\ ]\!]_R$ from edges to paths:



Beachte: Die obige Ausdehnung bedeutet einen Rückwärtsdurchlauf von Pfad p.

Reverse Aufsammelsemantik

Sei $\mathcal{S}^R_G =_{df} (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Definition 8.4.1.2 (Reverse Aufsammelsemantik)

Die von $\mathcal{S}^R_G$ induzierte (nichtdeterministische) reverse Aufsammelsemantik (oder globale abstrakte reverse Semantik) ist definiert durch:

$$[\![\ ]\!]^{CRS}_{\mathcal{S}^R_G} : N \to \mathcal{P}(\mathcal{C}_f)$$

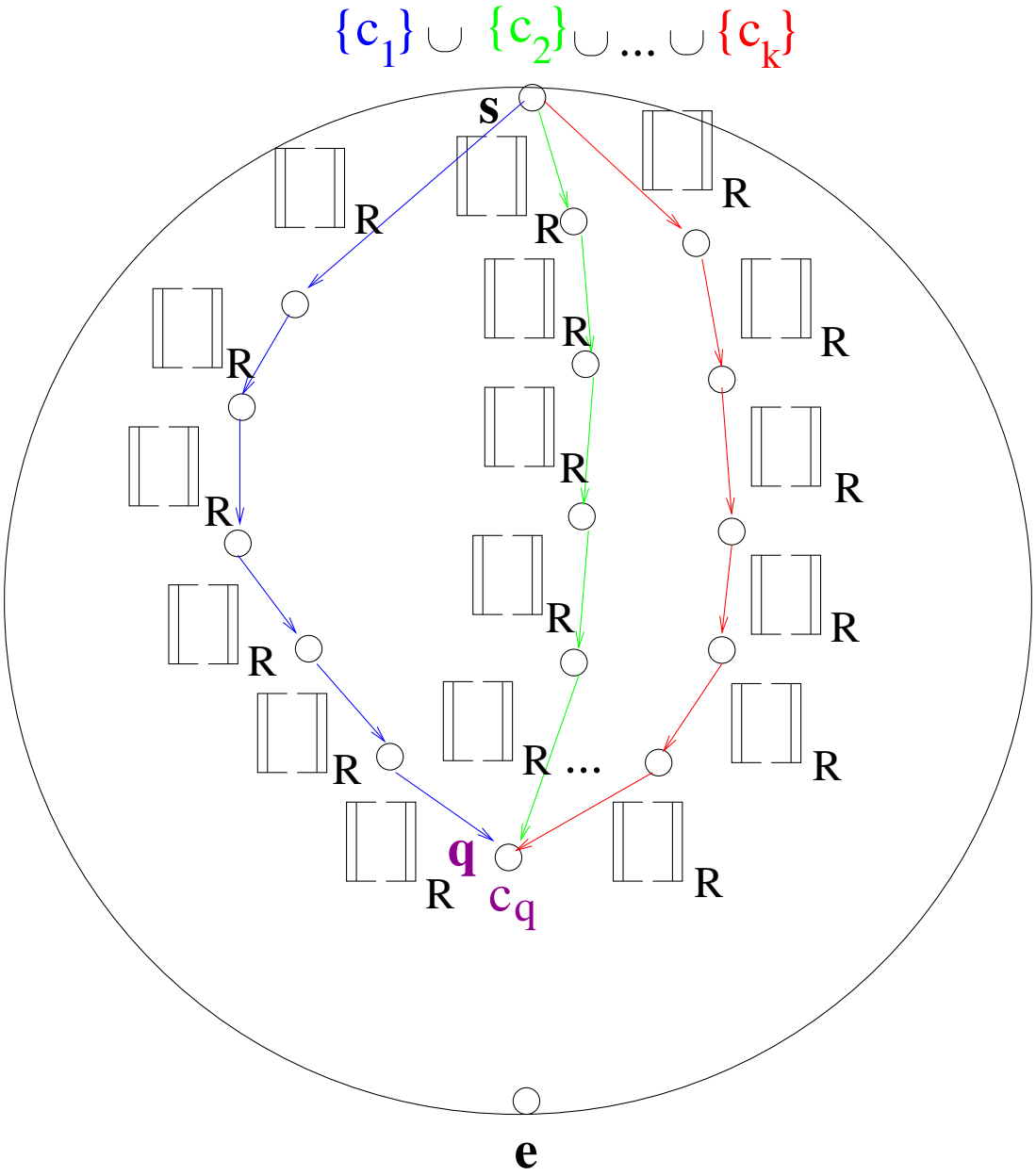
$$\forall\, n \in N.\ [\![\ n\ ]\!]^{CRS}_{\mathcal{S}^R_G} =_{df} \{\, [\![\ p\ ]\!]_R(c_{\mathbf{s}}) \mid p \in \mathbf{P}[n, \mathbf{q}] \,\}$$

wobei $\mathcal{P}$ den Potenzmengenoperator bezeichnet.

Beachte:

$$[\![\ \mathbf{q}\ ]\!]^{CRS}_{\mathcal{S}^R_G} = \{c_q\}$$

Illustration der reversen Aufsammelsemantik



Kapitel 8.4.2

Reverse Vereinigung-über-alle-Pfade-Semantik

Die *RJOP*-Semantik

Sei $\mathcal{S}_G^R =_{df} (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Definition 8.4.2.1 (*RJOP*-Semantik)

Die (deterministische) *RJOP*-Semantik von $\mathcal{S}_G^R$ ist definiert durch:

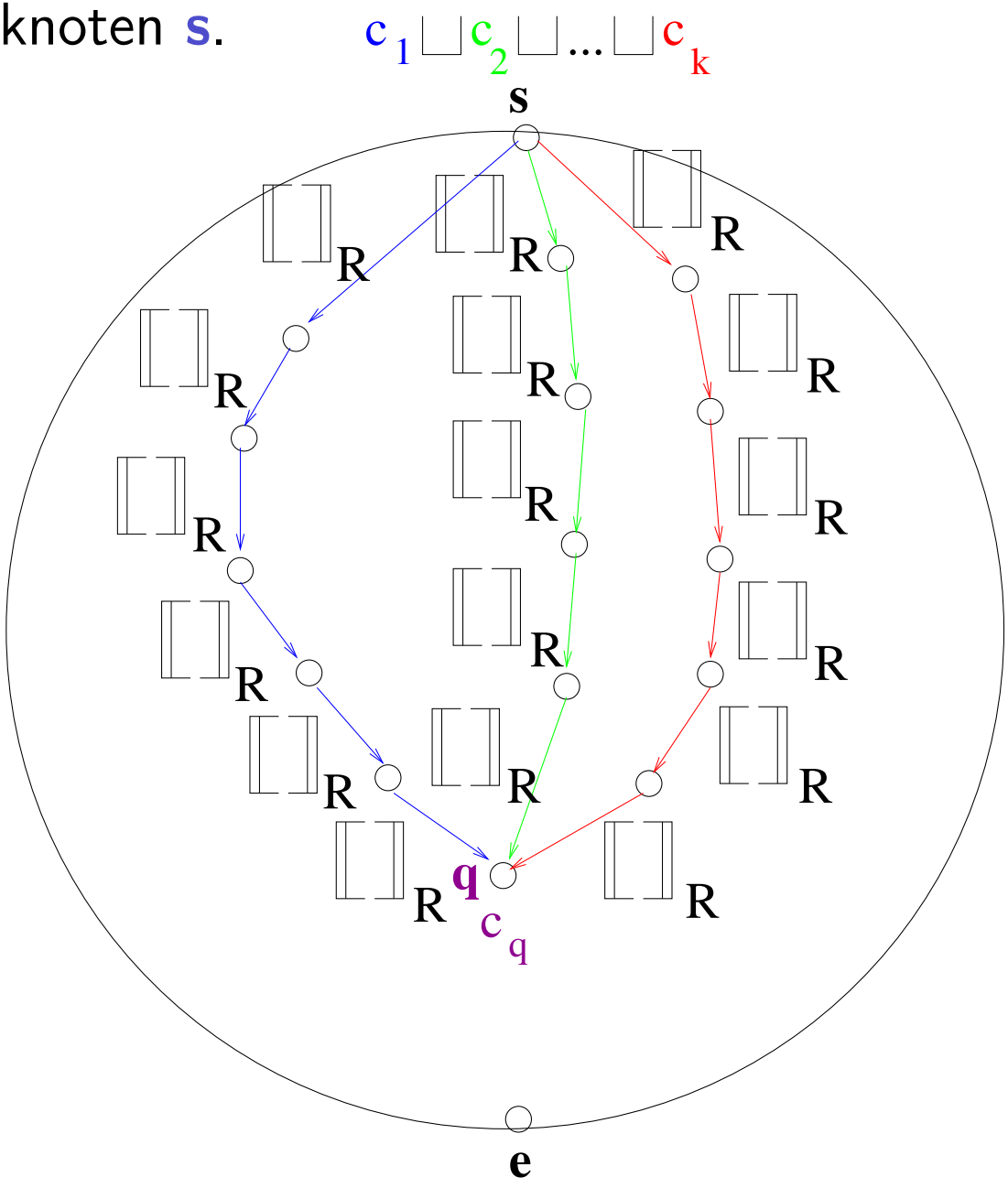
$$[\![\ ]\!]^{RJOP}_{\mathcal{S}_G^R} : N \to \mathcal{C}_f$$

$$\forall\, n \in N.\ [\![\, n \,]\!]^{RJOP}_{\mathcal{S}_G^R} =_{df} \bigsqcup [\![\, n \,]\!]^{CRS}_{\mathcal{S}_G^R}$$

$$= \bigsqcup \{\, [\![\, p \,]\!]_R(c_q) \mid p \in \mathbf{P}[n, \mathbf{q}] \,\}$$

Veranschaulichung der *RJOP*-Semantik

...am Startknoten **s**.



Kapitel 8.4.3

Reverse Schnitt-über-alle-Pfade-Semantik

Die *RMOP*-Semantik

Sei $\mathcal{S}^R_G =_{df} (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Definition 8.4.3.1 (*RMOP*-Semantik)

Die (deterministische) *RMOP*-Semantik von $\mathcal{S}^R_G$ ist definiert durch:

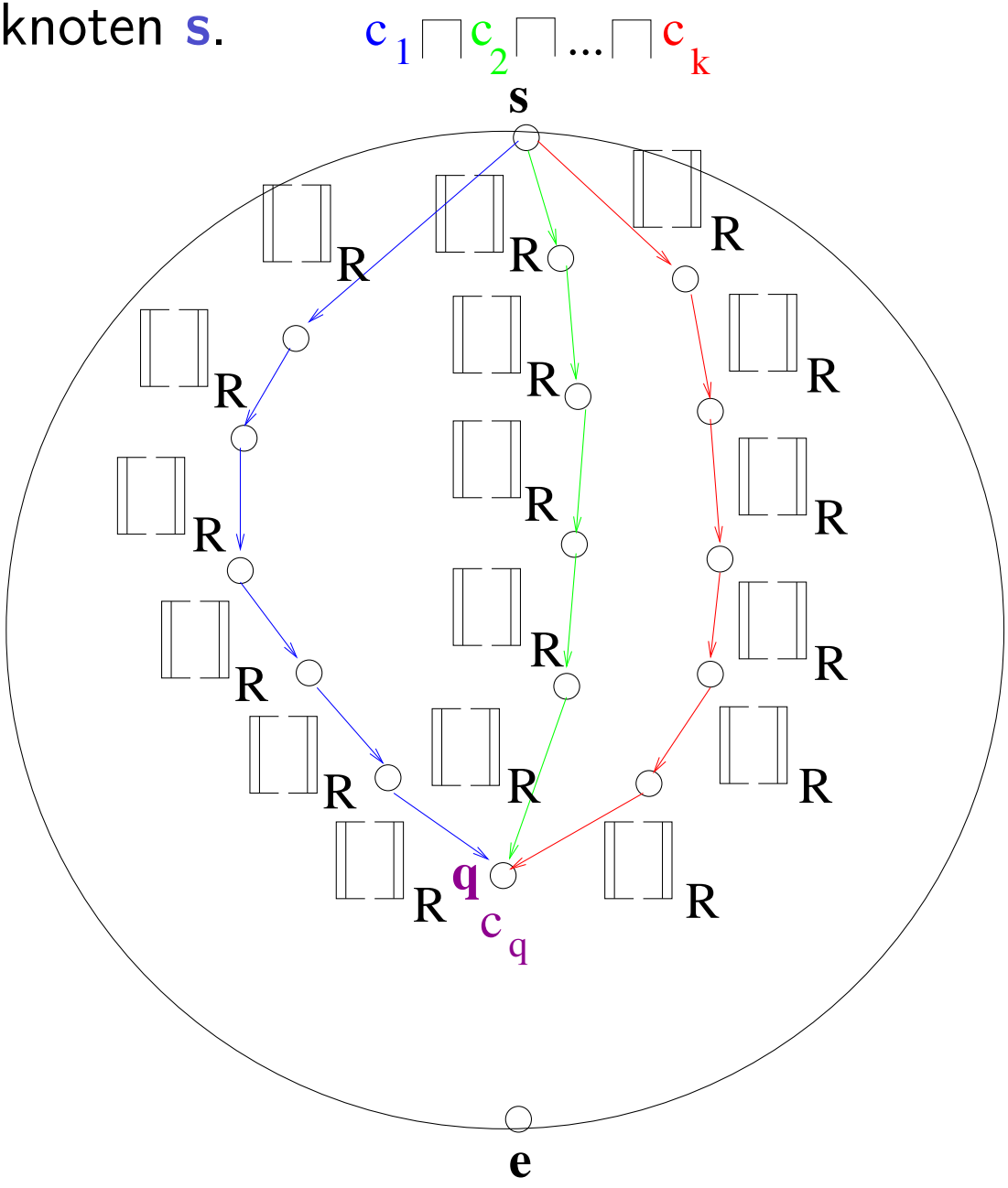
$$[\![\ ]\!]^{RMOP}_{\mathcal{S}^R_G} : N \to \mathcal{C}_f$$

$$\forall\, n \in N.\ [\![\, n \,]\!]^{RMOP}_{\mathcal{S}^R_G} =_{df} \bigsqcap [\![\, n \,]\!]^{CRS}_{\mathcal{S}^R_G}$$

$$= \bigsqcap \{\, [\![\, p \,]\!]_R(c_q) \mid p \in \mathbf{P}[n, \mathbf{q}] \,\}$$

Veranschaulichung der *RMOP*-Semantik

...am Startknoten **s**.



Kapitel 8.4.4

RVUP- und *RSUP*-Semantik als spezifizierende Lösungen reverser DFA-Probleme

Spezifizierende Lösungen reverser DFA-Prob.

...nach dem Vorbild der *SUP*- und *VUP*-Semantik für DFA-Probleme definieren wir:

Definition 8.4.4.1 (Spezifizierende Lsg. v. RDFA-P.)

Die *RJOP*- und *RMOP*-Semantik eines Flussgraphen definieren die spezifizierenden Lösungen eines reversen DFA-Problems, seine sog. *RJOP*- und *RMOP*-Lösungen.

Kapitel 8.5

Reverse denotationelle globale DFA-Semantik

Kapitel 8.5.1

Reverse minimale Fixpunktsemantik

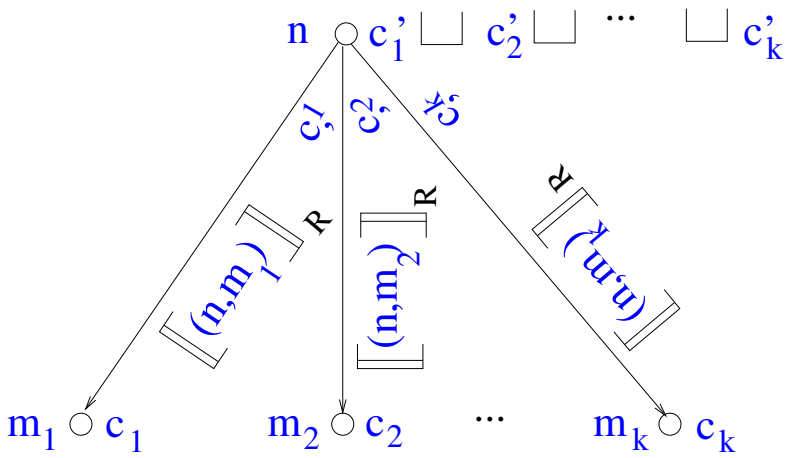
Reverser minimaler (*RMinFP*) Fixpunktansatz

Sei $\mathcal{S}^R_G =_{df} (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Gleichungssystem 8.5.1.1 (*RMinFP*-Ansatz)

$$reqInf(n) = \begin{cases} c_q & \text{falls } n = \mathbf{q} \\ \bigsqcup \{ [\![(n,m)]\!]_R(reqInf(m)) \mid m \in succ(n) \} & \text{sonst} \end{cases}$$

Illustration des *RMinFP*-Ansatzes ($n \neq \mathbf{e}$):



The *RMinFP*-Semantik

Bezeichne

- $reqInf^{\star}_{c_q}(n),\ n \in N$

die kleinste Lösung von Gleichungssystem 8.5.1.1.

Definition 8.5.1.2 (*RMinFP*-Semantik)

Die *RMinFP*-Semantik von $\mathcal{S}^R_G$ ist definiert durch:

$$[\![\ ]\!]^{RMinFP}_{\mathcal{S}^R_G} : N \to \mathcal{C}_f$$

$$\forall\, n \in N.\ [\![\ n\ ]\!]^{RMinFP}_{\mathcal{S}^R_G} =_{df} reqInf^{\star}_{c_q}(n)$$

Beachte:

$$[\![\ \mathbf{q}\ ]\!]^{RMinFP}_{\mathcal{S}^R_G} = c_q$$

Kapitel 8.5.2

Reverse maximale Fixpunktsemantik

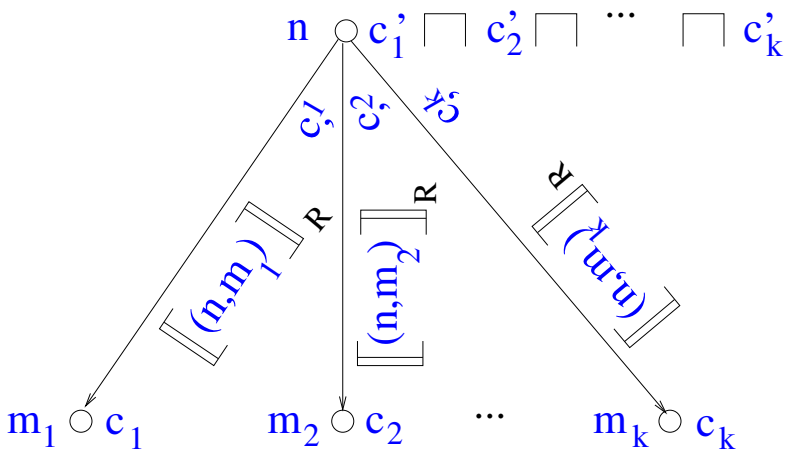
Reverser maximaler (*RMaxFP*) Fixpunktansatz

Sei $\mathcal{S}^R_G =_{df} (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Gleichungssystem 8.5.2.1 (*RMaxFP*-Ansatz)

$$reqInf(n) = \begin{cases} c_q & \text{falls } n = \mathbf{q} \\ \bigsqcap \{ [\![(n,m)]\!]_R(reqInf(m)) \mid m \in succ(n) \} & \text{sonst} \end{cases}$$

Illustration des *RMaxFP*-Ansatzes ($n \neq \mathbf{e}$):



The *RMaxFP*-Semantik

Bezeichne

- $reqInf^{\star}_{c_q}(n),\ n \in N$

die größte Lösung von Gleichungssystem 8.5.2.1.

Definition 8.5.2.2 (*RMaxFP*-Semantik)

Die *RMaxFP*-Semantik von $\mathcal{S}^R_G$ ist definiert durch:

$$[\![\ ]\!]^{RMaxFP}_{\mathcal{S}^R_G} : N \to \mathcal{C}_f$$

$$\forall\, n \in N.\ [\![\ n\ ]\!]^{RMaxFP}_{\mathcal{S}^R_G} =_{df} reqInf^{\star}_{c_q}(n)$$

Beachte:

$$[\![\ \mathbf{q}\ ]\!]^{RMaxFP}_{\mathcal{S}^R_G} = c_q$$

Kapitel 8.6

Generischer reverser Fixpunktalgorithmus

Die *RMinFP*- und *RMaxFP*-Semantik

...sind praktisch relevant, weil das *RMinFP*-Gleichungssystem 8.5.1.1 und das *RMaxFP*-Gleichungssystem 8.5.2.1 ein generisches

- iteratives Berechnungsverfahren (Algorithmus 8.6.1.1)

induzieren, das ihre kleinste und größte Lösung zu approximieren erlaubt, d.h. die *RMinFP*- und *RMaxFP*-Semantik.

Kapitel 8.6.1

Algorithmus

Gen. reverser Fixpunktalgorithmus 8.6.1.1 (1)

Eingabe: Reverse DFA-Spezifikation $\mathcal{S}_G^R =_{df} (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$.

Ausgabe: Bei Terminierung des Algorithmus (s. Terminierungstheorem 8.6.2.1) enthält die Variable *reqInf*[n] die *RMinFP*-Lösung von $\mathcal{S}_G^R$ am Knoten n.

Zusätzlich (s. Reverses Sicherheitstheorem 8.7.1 und Reverses Koinzidenztheorem 8.7.2) gilt: Wenn $[\![\ ]\!]_R$

- additiv: *reqInf*[**s**] enthält die
- monoton: *reqInf*[**s**] enthält eine obere Approximation der

der *RJOP*-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.

Gen. reverser Fixpunktalgorithmus 8.6.1.1 (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
            $join := [\![(n, m)]\!]_R(reqInf[m]) \sqcup_f reqInf[n]$;
            IF $reqInf[n] \sqsubset_f join$
                THEN
                    $reqInf[n] := join$;
                    $workset := workset \cup \{n\}$ FI
        OD ESOOHC OD.

Kapitel 8.6.2

Terminierung

Terminierung

...mit Lemma 8.2.2(1) ("die von einer lokalen DFA-Semantik induzierten reversen Semantikfkt. sind monoton") erhalten wir:

Theorem 8.6.2.1 (Terminierung)

Der generische reverse Fixpunktalgorithmus 8.6.1.1 terminiert mit der

1. *RMinFP*-Semantik von $\mathcal{S}_G^R$, wenn $\widehat{\mathcal{C}_f}$ die aufsteigende Kettenbedingung erfüllt.
2. *RMaxFP*-Semantik von $\mathcal{S}_G^R$, wenn $\widehat{\mathcal{C}_f^{usd}}$ die aufsteigende Kettenbedingung erfüllt, wobei

$$\widehat{\mathcal{C}_f^{usd}} =_{df} (\mathcal{C}_f, \sqcup_f, \sqcap_f, \sqsupseteq_f, \mathit{failure}, \bot_f)$$

der auf den Kopf gestellte Verband

$$\widehat{\mathcal{C}_f} = (\mathcal{C}_f, \sqcap_f, \sqcup_f, \sqsubseteq_f, \bot_f, \mathit{failure})$$

ist.

Terminierungskorollar

...mit Lemma 8.1.3 ("der fehlschlagserweiterte DFA-Verband $\widehat{\mathcal{C}_f}$ 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 8.6.2.1 auf die entsprechende Eigenschaft des Ursprungsproblems zurückführen:

Korollar 8.6.2.2 (Terminierung)

Der generische reverse Fixpunktalgorithmus 8.6.1.1 terminiert mit der *RMinFP*-Semantik (*RMaxFP*-Semantik) von $\mathcal{S}_G^R$, wenn $\widehat{\mathcal{C}}$ ($\widehat{\mathcal{C}^{usd}}$) die aufsteigende Kettenbedingung erfüllt, wobei $\widehat{\mathcal{C}^{usd}}$ der der auf den Kopf gestellte Verband $\widehat{\mathcal{C}}$ ist.

Berechenbare Lösungen eines RDFA-Problems

...zusammen legen der generische reverse Fixpunktalgorithmus 8.6.1.1 und das Terminierungstheorem 8.6.2.1 folgende Definition nahe:

Definition 8.6.2.2 (Berechenbare RDFA-Lösungen)

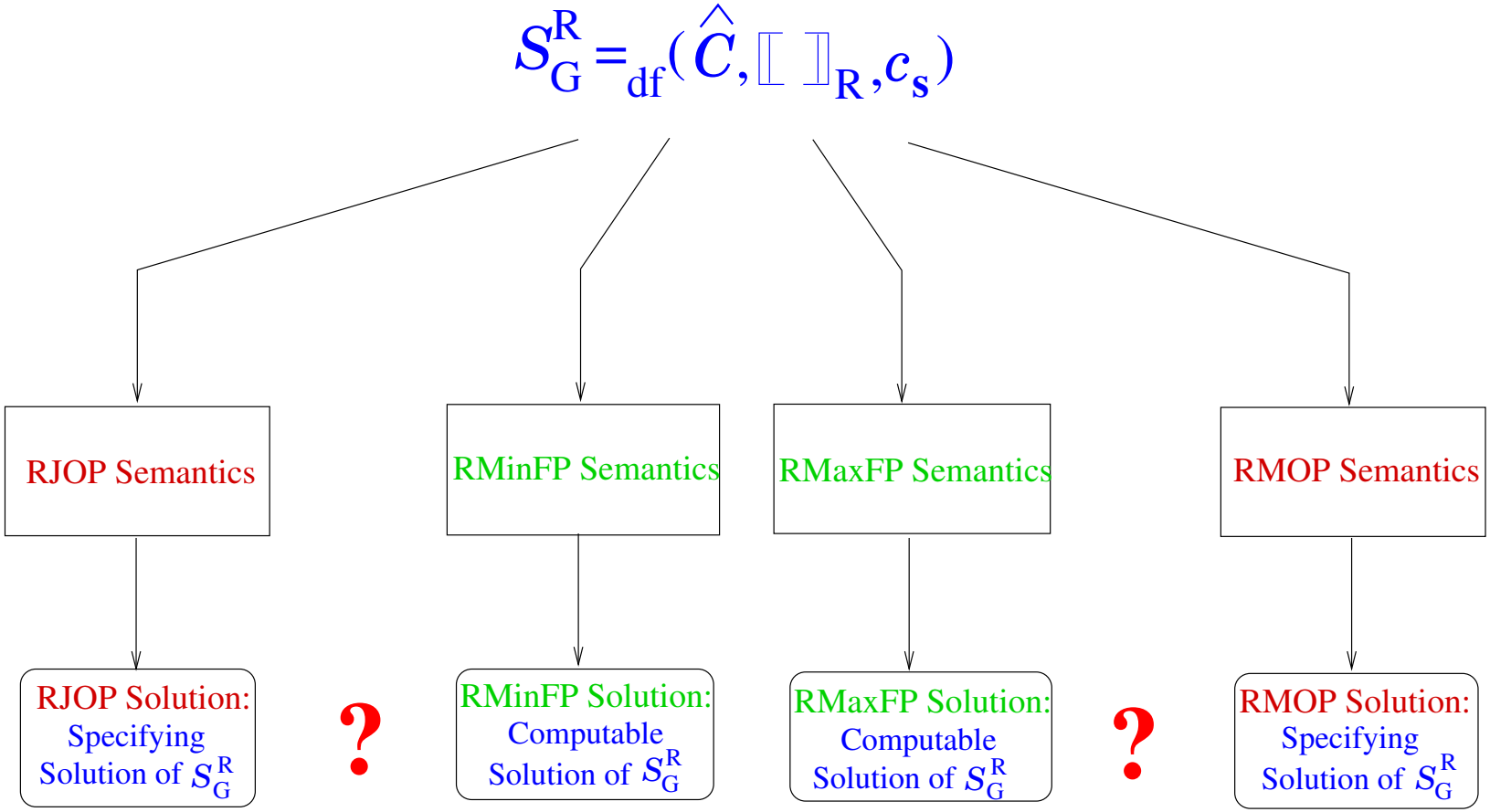
Die *RMinFP-* und *RMaxFP*-Semantik eines Flussgraphen definieren die berechenbaren Lösungen eines reversen DFA-Problems, seine sog. *RMinFP-* und *RMaxFP*-Lösungen.

Kapitel 8.7

Reverse Sicherheit und Koinzidenz

RJOP / *RMinFP*- u. *RMOP* / *RMaxFP*-Semantik

...einer RDFA-Spezifikation und die Frage ihrer Beziehung zueinander:



Reverse Sicherheit

Sei $\mathcal{S}_G^R =_{df} (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

...mit Lemma 8.2.2(1) ("die von einer lokalen DFA-Semantik induzierten reversen Semantikfkt. sind monoton") erhalten wir:

Theorem 8.7.1 (Reverse Sicherheit)

1. Die *RMinFP*-Semantik von $\mathcal{S}_G^R$ ist eine sichere (d.h. obere) Approximation der *RJOP*-Semantik von $\mathcal{S}_G^R$, d.h.,

$$\forall\, n \in N.\ [\![\, n \,]\!]^{RMinFP}_{\mathcal{S}_G^R} \sqsupseteq [\![\, n \,]\!]^{RJOP}_{\mathcal{S}_G^R}$$

2. Die *RMaxFP*-Semantik von $\mathcal{S}_G^R$ ist eine sichere (d.h. untere) Approximation der *RMOP*-Semantik von $\mathcal{S}_G^R$, d.h.,

$$\forall\, n \in N.\ [\![\, n \,]\!]^{RMaxFP}_{\mathcal{S}_G^R} \sqsubseteq [\![\, n \,]\!]^{RMOP}_{\mathcal{S}_G^R}$$

Reverse Koinzidenz

Sei $\mathcal{S}_G^R =_{df} (\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ eine reverse DFA-Spezifikation.

Theorem 8.7.2 (Reverse Koinzidenz)

1. Die *RMinFP*- und *RJOP*-Semantik von $\mathcal{S}_G^R$ stimmen überein, d.h.,
$$\forall\, n \in N.\ [\![\, n \,]\!]^{RMinFP}_{\mathcal{S}_G^R} = [\![\, n \,]\!]^{RJOP}_{\mathcal{S}_G^R}$$

2. Die *RMaxFP*- und *RMOP*-Semantik von $\mathcal{S}_G^R$ stimmen überein, d.h.,
$$\forall\, n \in N.\ [\![\, n \,]\!]^{RMaxFP}_{\mathcal{S}_G^R} = [\![\, n \,]\!]^{RMOP}_{\mathcal{S}_G^R}$$

wenn die reverse DFA-Semantik $[\![\ ]\!]_R$ additiv bzw. distributiv ist.

Reverses Koinzidenzkorollar

...mit Lemma 7.1.2.7(1) und 8.1.4 ("$[\![\ ]\!]_R$ ist distributiv/additiv, wenn $[\![\ ]\!]$ distributiv ist") können wir die verbleibenden Voraussetzungen von Theorem 8.7.2 auf eine einzelne Eigenschaft des induzierenden Ursprungsproblems zurückführen:

Korollar 8.7.3 (Reverse Koinzidenz)

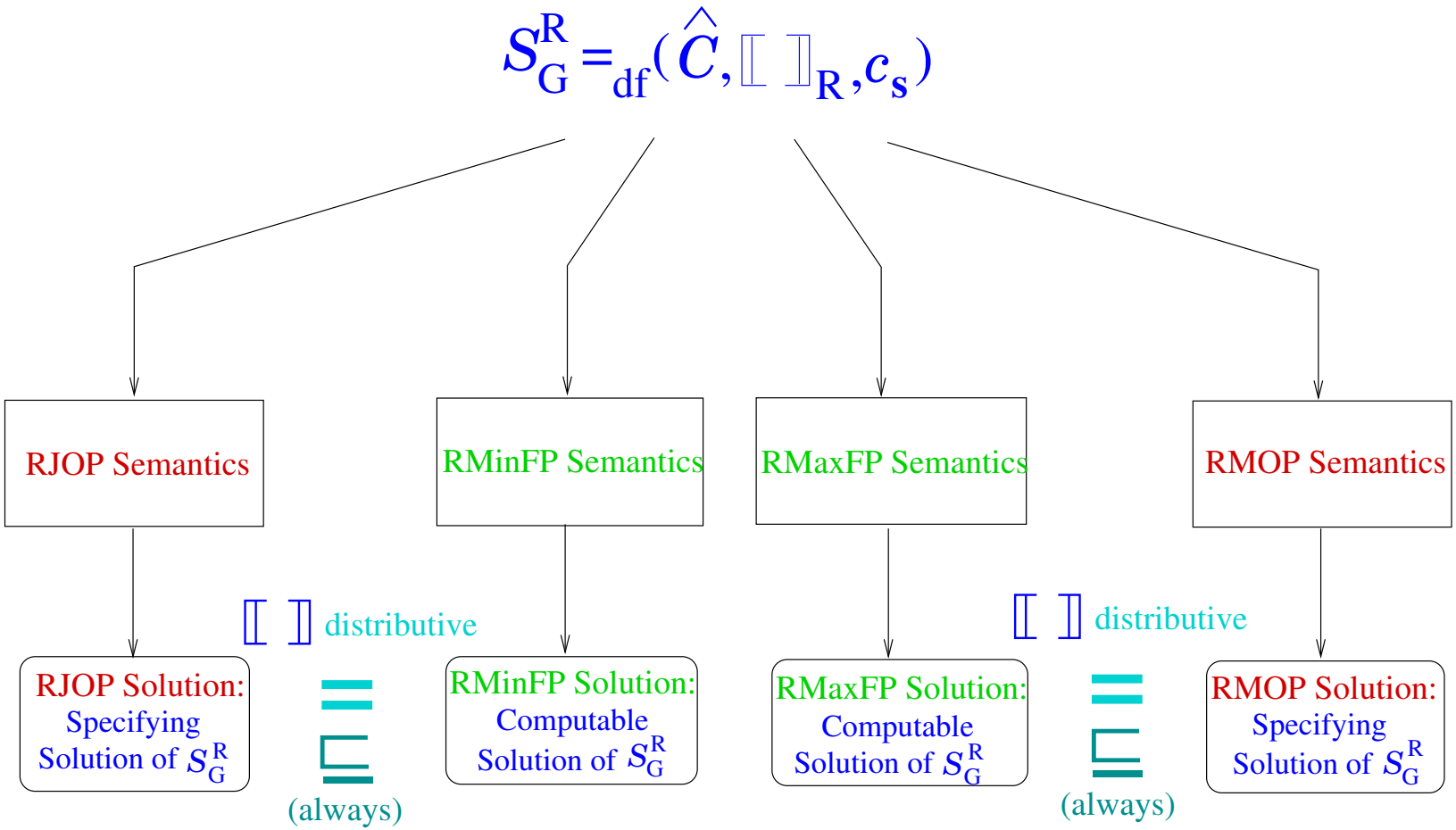
1. Die *RMinFP*- und *RJOP*-Semantik von $\mathcal{S}_G^R$ stimmen überein, d.h.,
$$\forall\, n \in N.\ [\![\, n \,]\!]^{RMinFP}_{\mathcal{S}_G^R} = [\![\, n \,]\!]^{RJOP}_{\mathcal{S}_G^R}$$

2. Die *RMaxFP*- und *RMOP*-Semantik von $\mathcal{S}_G^R$ stimmen überein, d.h.,
$$\forall\, n \in N.\ [\![\, n \,]\!]^{RMaxFP}_{\mathcal{S}_G^R} = [\![\, n \,]\!]^{RMOP}_{\mathcal{S}_G^R}$$

wenn die lokale DFA-Semantik $[\![\ ]\!]$ der induzierenden DFA-Spezifikation $\mathcal{S}_G$ distributiv ist.

RJOP / *RMinFP*- u. *RMOP* / *RMaxFP*-Semantik

...einer RDFA-Spezifikation und ihre Beziehung zueinander:



Konservativität von Algorithm 8.6.1.1

...mit Lemma 8.1.3 und Theorem 8.7.1 erhalten wir:

Corollary 8.7.4 (*RJOP*/*RMOP*-Konservativität)

Algorithmus 8.6.1.1 ist

- *RJOP*- (*RMOP*-) konservativ

für $\mathcal{S}_G^R$, d.h. terminiert mit einer oberen (unteren) Approximation der *RJOP*- (*RMOP*-) Semantik von $\mathcal{S}_G^R$, wenn der Verband $\widehat{\mathcal{C}}$ der induzierenden DFA-Spezifikation $\mathcal{S}_G$ (und damit auch $\widehat{\mathcal{C}_f}$) die aufsteigende (absteigende) Kettenbedingung erfüllt.

Straffheit von Algorithm 8.6.1.1

...Straffheit (engl. tightness).

Korollar 8.7.5 (*RJOP*/*RMOP*-Straffheit)

Algorithmus 8.6.1.1 ist

- *RJOP*- (*RMOP*-) straff

für $\mathcal{S}_G^R$, d.h. terminiert mit der *RJOP*- (*RMOP*-) 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 $[\![\ ]\!]_f$) impliziert Addivität von $[\![\ ]\!]_R$ (siehe Lemma 7.1.2.7(1) und 8.1.4(2)). Somit sind Distributivität und Addivität von $[\![\ ]\!]_R$ implizit in Korollar 8.7.5 gefordert, werden aber auf die Distributivitätseigenschaft des Ursprungsproblems zurückgeführt.

Kapitel 8.8

RDFA-Anwendungsbeispiele

Reverse Datenflussanalyse

...hat eine Vielzahl von Anwendungen, darunter

- anforderungsgetriebene Datenflussanalyse (engl. demand-driven data-flow analysis)

oder den Bau von

- 'Hot Spot'-Programmanalysatoren und -optimierern
- Fehlersuchern (engl. Debugger)
- ...

Kapitel 8.8.1

Verfügbare Ausdrücke

Verfügbarkeit: Reverse DFA-Spezifikation

...reverse Verfügbarkeit für einen einzelnen Term t.

Reverse DFA-Spezifikation für Verfügbarkeit:

1. DFA-Verband:
   $(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathbb{B}_f, \wedge_f, \vee_f, \leq_f, \textbf{falsch}, \textit{failure})$
   mit $\textbf{falsch} \leq_f \textbf{wahr} \leq_f \textit{failure}$

2. Reverse DFA-Semantik:
   $[\![\ ]\!]^t_{av_R} : E \to (\ \mathbb{B}_f \to \mathbb{B}_f\ )$ definiert durch
   $\forall\, e \in E.\ \forall\, b \in \mathbb{B}_f.\ [\![\ e\ ]\!]^t_{av_R}(b) =_{df}$
   $$\bigsqcap \{ b' \in \mathbb{B}_f \mid [\![\ e\ ]\!]^t_{av,f}(b') \geq_f b \}$$
   wobei $[\![\ e\ ]\!]^t_{av,f} : E \to (\ \mathbb{B}_f \to \mathbb{B}_f\ )$ die Ausdehnung der lokalen DFA-Semantik $[\![\ e\ ]\!]^t_{av} : E \to (\ \mathbb{B} \to \mathbb{B}\ )$ von Variante 1 aus Kapitel 7.10.1 von $\mathbb{B}$ auf $\mathbb{B}_f$ gemäß Definition 8.1.2 ist.

Charakterisierung der rev. DFA-Semantikfkt.

...mit Hilfe der Funktionen $Cst^R_{\mathbf{wahr}}$, $Cst^R_{\mathbf{falsch}}$ und $Id^R_{\mathbb{B}_f}$ über $\mathbb{B}_f =_{df} \{\mathbf{falsch}, \mathbf{wahr}, \mathit{failure}\}$, die definiert sind durch:

$$\forall\, b \in \mathbb{B}_f.\ Cst^R_{\mathbf{wahr}}(b) =_{df} \begin{cases} \mathbf{falsch} & \text{falls } b \in \mathbb{B} \\ \mathit{failure} & \text{sonst (d.h. falls } b = \mathit{failure}) \end{cases}$$

$$\forall\, b \in \mathbb{B}_f.\ Cst^R_{\mathbf{falsch}}(b) =_{df} \begin{cases} \mathbf{falsch} & \text{falls } b = \mathbf{falsch} \\ \mathit{failure} & \text{sonst} \end{cases}$$

$$Id^R_{\mathbb{B}_f} =_{df} Id_{\mathbb{B}_f}$$

...können wir die induzierten reversen DFA-Semantikfunktionen $[\![\, e \,]\!]^t_{av_R}(b)$, $e \in E$, direkt charakterisieren.

Charakterisierungslemma

Charakterisierungslemma 8.8.1.1

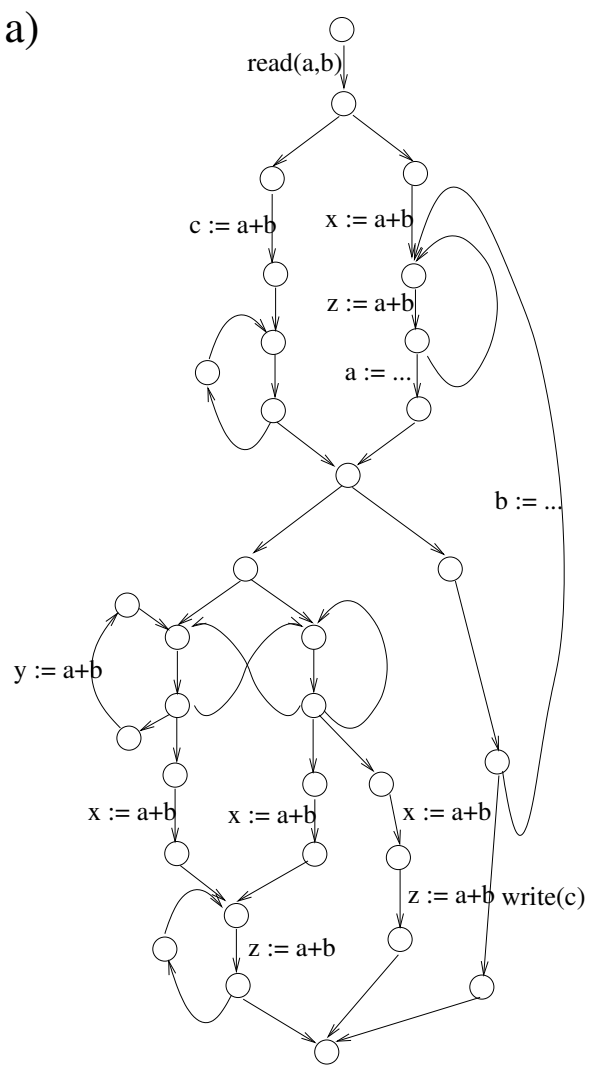
$$\forall\, e \in E.\ [\![\, e \,]\!]^t_{av_R} = \begin{cases} Cst^R_{\mathbf{wahr}} & \text{falls } [\![\, e \,]\!]^t_{av} = Cst_{\mathbf{wahr}} \\ Id^R_{\mathbb{B}_f} & \text{falls } [\![\, e \,]\!]^t_{av} = Id_{\mathbb{B}} \\ Cst^R_{\mathbf{falsch}} & \text{falls } [\![\, e \,]\!]^t_{av} = Cst_{\mathbf{falsch}} \end{cases}$$

wobei $[\![\, e \,]\!]^t_{av} : E \to (\, \mathbb{B} \to \mathbb{B}\,)$ die lokale DFA-Semantik von Variante 1 aus Kapitel 7.10.1 ist.

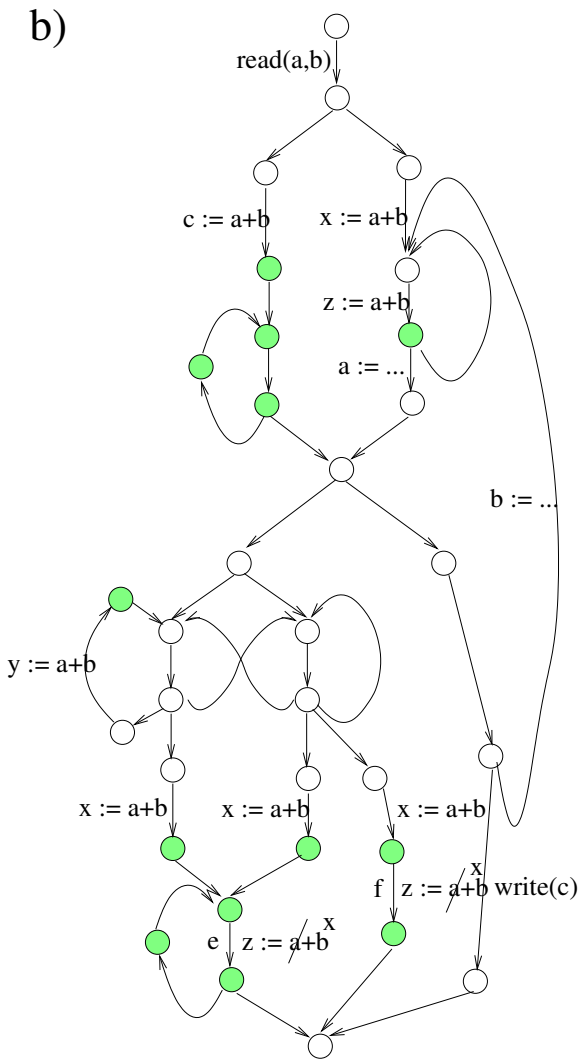
Kapitel 8.8.2

‘Hot Spot’-Analysatoren, -optimierer

Einfacher Analysator und Optimierer



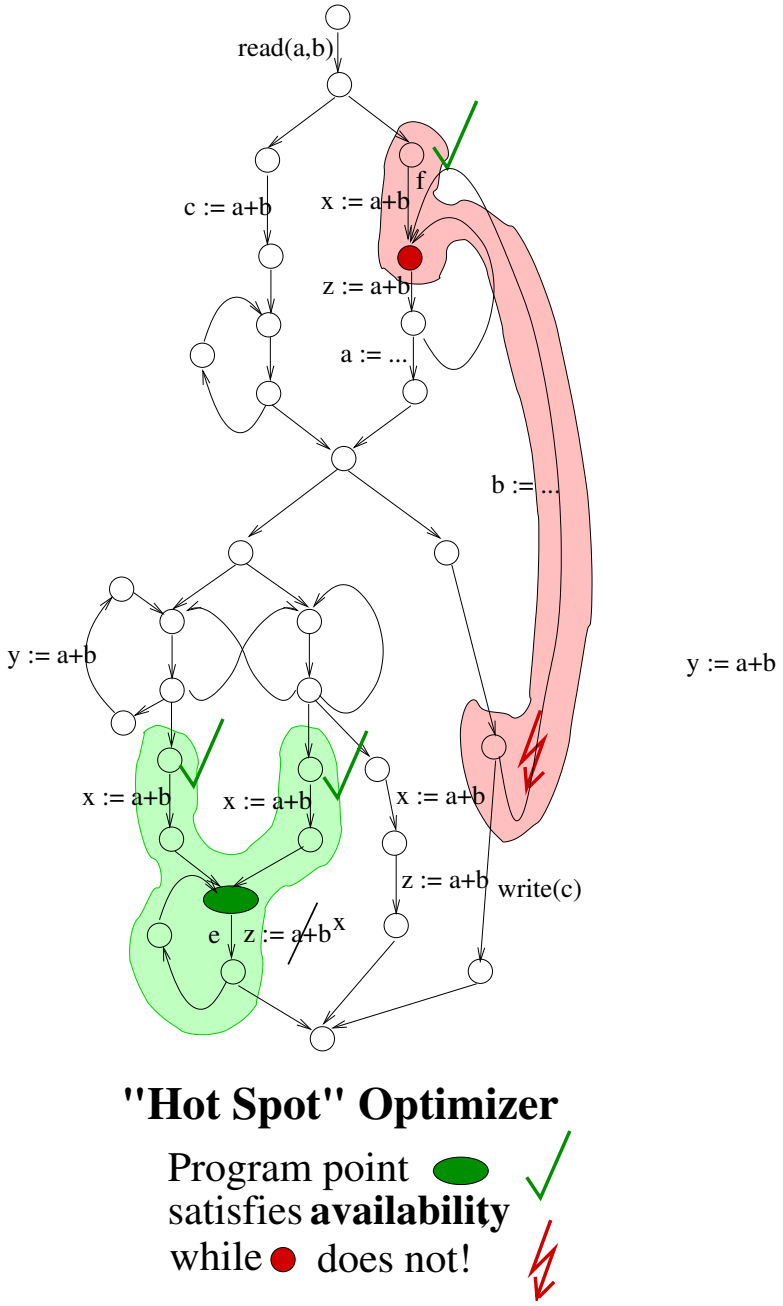
Flow graph



Data−flow information

Program points satisfying **availability**

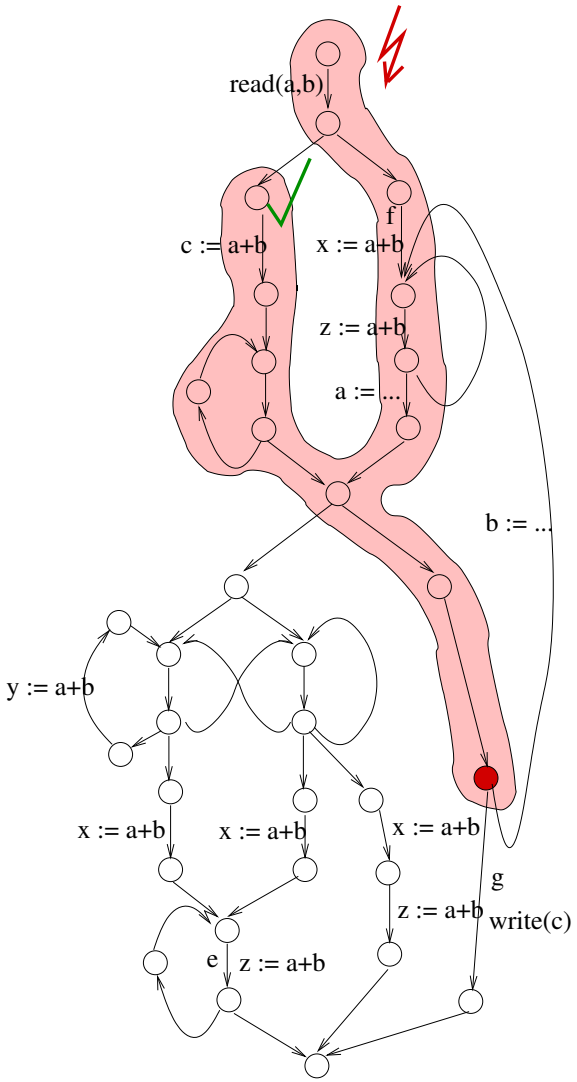
‘Hot Spot’-Analysator und -optimierer



Kapitel 8.8.3

Fehlersucher

Fehlersucher



Debugger

Variable c is not initialized along some paths reaching program point ●

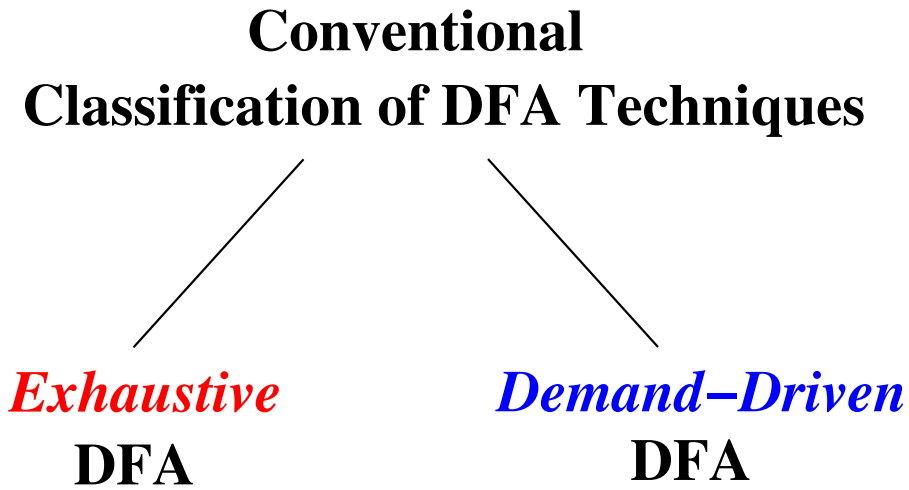
Kapitel 8.8.4

Anforderungsgetriebene Datenflussanalyse

Erschöpfende vs. anforderungsgetriebene DFA

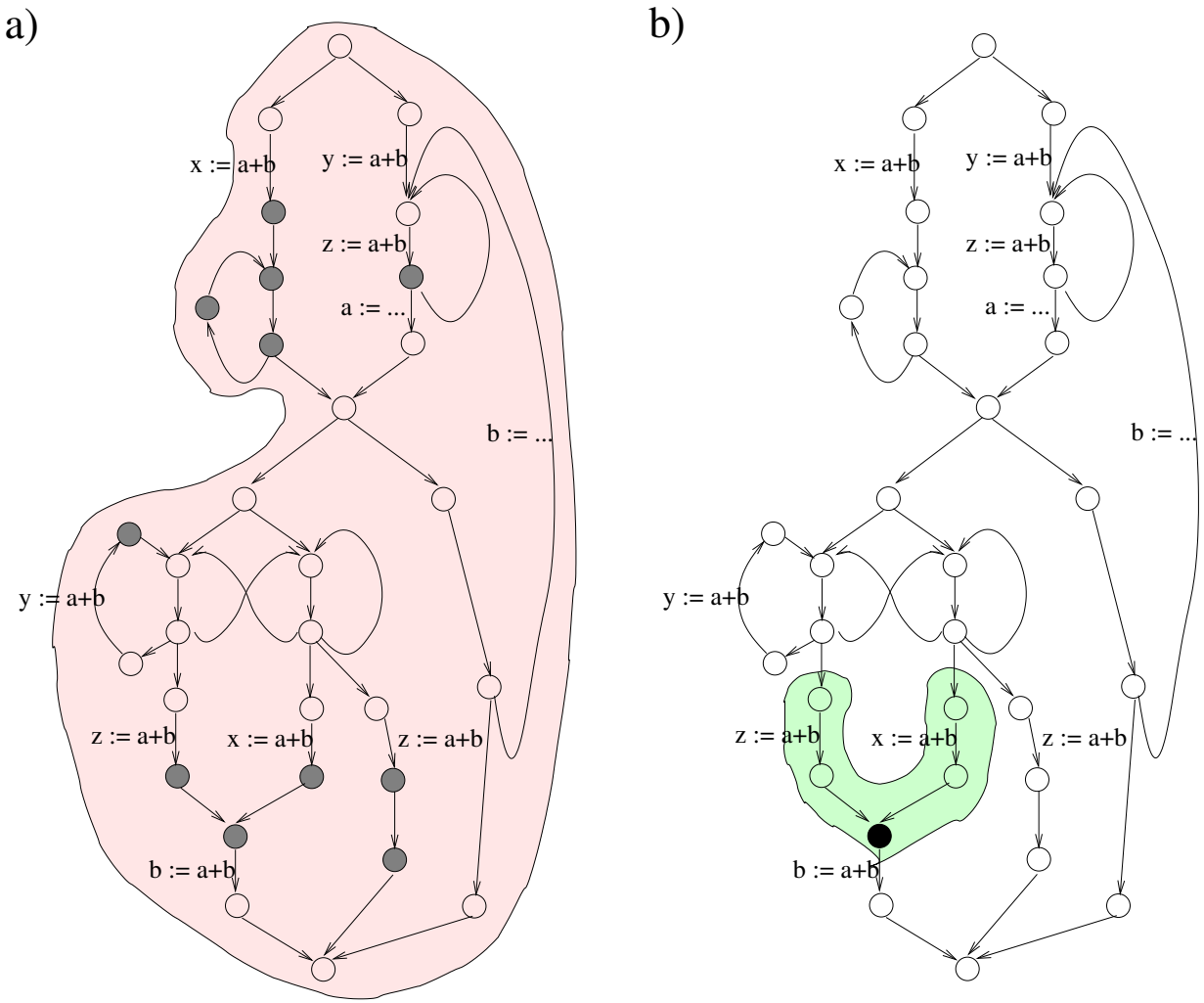
Erschöpfende (xDFA) vs.

anforderungsgetriebene DFA (ddDFA):



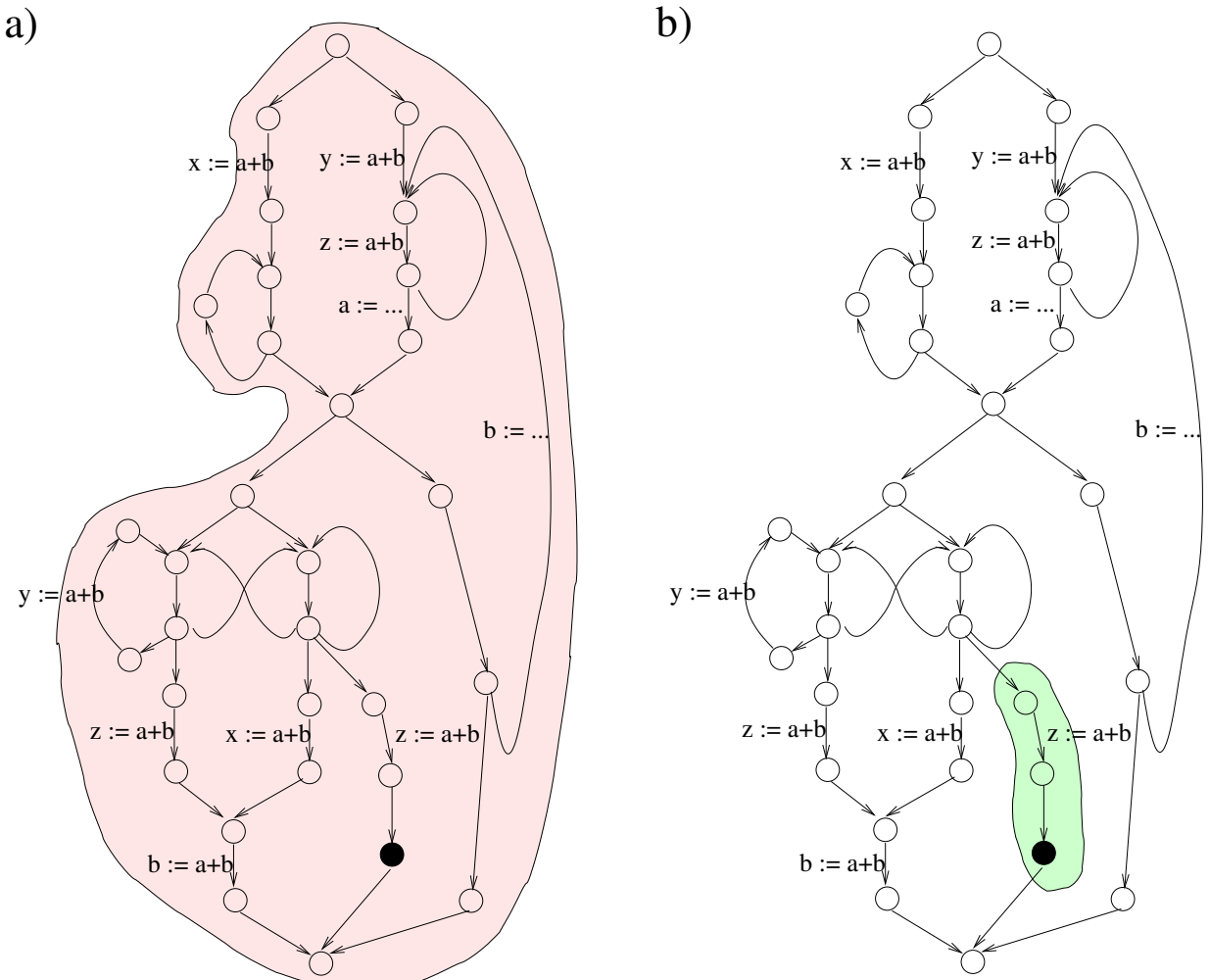
Aufwand xDFA und ddDFA im Vergleich (1)

Verfügbarkeit an einem Punkt: Informeller, anekdotischer Vergleich von Berechnungsaufwand erschöpfend (rosa), anforderungsgetrieben: (grün)



Aufwand xDFA und ddDFA im Vergleich (2)

Verfügbarkeit an einem Punkt: Informeller, anekdotischer Vergleich von Berechnungsaufwand erschöpfend (rosa), anforderungsgetrieben: (grün)



Beobachtung

...in günstigen Fällen

- kann der Aufwand anforderungsetriebener DFA erheblich niedriger sein als für eine entsprechende erschöpfende Analyse.

...in ungünstigen Fällen

- ergibt sich kein Vorteil.

Kapitel 8.9

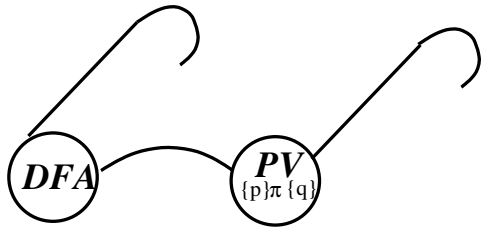
Zum Zusammenhang von DFA und RDFA

Kapitel 8.9.1

Überblick

DFA/reverse DFA vs. Verifikation

...eine Gegenüberstellung:



Data–flow Analysis		**Program Verification**
MaxFP	$[\![e]\!] \in \lceil C \xrightarrow{\text{distributive}} C \rceil$	*Strongest Postcondition View*
Coincidence Theorem $\Vert$		
$MOP_{c_s}(q) \sqsupseteq c_q$	induces	$\{p\}\pi\{?\}$
Link Theorem $\Updownarrow$		
$rJOP_{c_q}(s) \sqsubseteq c_s$	$[\![e]\!]_R(c) =_{df} \bigsqcap \{c' \mid [\![e]\!](c') \sqsupseteq c\}$	$\{?\}\pi\{q\}$
Reverse Coincidence Theorem $\Vert$		
rMinFP		*Weakest Precondition View*

Allgemeiner: Analyse vs. Verifikation

Vergleiche Aufgaben und Vorgehen von:

- Typprüfung (ein Programmierervorschlag für eine gültige Typisierung wird auf Stichhaltigkeit überprüft):
  $\rightsquigarrow$ Verifikation(sproblem)
- Typinferenz (eine stichhaltigeTypisierung wird ohne vorgegebene Typisierungsinfomation generiert bzw. eine Fehlermeldung ausgegeben, wenn dies nicht möglich ist):
  $\rightsquigarrow$ Analyse(problem)

Leisten Typprüfung und Typinferenz nicht im wesentlichen dasselbe? Ist Inferenz (Analyse) nicht sogar schwerer als Prüfung (Verifikation)?

Das Beispiel legt nahe, dass Verifikation nicht notwendig tiefgehender, erschöpfender oder konzeptuell schwerer sein muss als Analyse, vielmehr, dass Verifikation und Analyse zwei Seiten derselben Münze sind oder sogar deren selbe Seite...

Zusammenhang von DFA und reverser DFA

...für distributive DFA-Probleme.

Theorem 8.9.1.1 (Zusammenhangstheorem)

Sei $\mathcal{S}_G$ eine DFA-Spezifikation mit distributiver lokaler DFA-Semantik $[\![\ ]\!]$. Dann gilt für alle Startzusicherungen $c_{\mathbf{s}} \in \mathcal{C}$ und DFA-Anfragen $c_q \in \mathcal{C}$:

$$[\![\ q\ ]\!]^{MaxFP}_{\mathcal{S}_G, c_{\mathbf{s}}} \quad = \quad [\![\ q\ ]\!]^{MOP}_{\mathcal{S}_G, c_{\mathbf{s}}} \quad \sqsupseteq \quad c_q$$

$$\Longleftrightarrow$$

$$[\![\ \mathbf{s}\ ]\!]^{RMinFP}_{\mathcal{S}^R_G, c_q} \quad = \quad [\![\ \mathbf{s}\ ]\!]^{RJOP}_{\mathcal{S}^R_G, c_q} \quad \sqsubseteq \quad c_{\mathbf{s}}$$

Informell: Wir wissen am Programmpunkt q bezüglich der Startzusicherung $c_{\mathbf{s}}$ mindestens Datenflussfakt c_q, wenn wir für die Gültigkeit von Datenflussfakt c_q am Programmpunkt q höchstens die Startzusicherung $c_{\mathbf{s}}$ am Startknoten $\mathbf{s}$ fordern müssen und umgekehrt.

Hausaufgabe

Was gilt für den Zusammenhang

1. von *MinFP-/JOP*-Semantik einer DFA-Spezifikation $\mathcal{S}_G$ mit distributiver lokaler DFA-Semantik $[\![\ ]\!]$ und der von ihr induzierten *RMaxFP-/RMOP*-Semantik des zugehörigen reversen DFA-Problems?
2. aus Theorem 8.9.1.1 und der vorigen Teilaufgabe für DFA-Spezifikationen $\mathcal{S}_G$ mit monotoner, aber nicht distributiver lokaler DFA-Semantik?

Beweisen Sie Ihre Behauptung(en).

Kapitel 8.9.2

Korrektheit, Vollständigkeit

Vorbereitung

Analyse-Szenario: Sei

- ϕ die interessierende Programmeigenschaft (z.B., Verfügbarkeit eines Terms, Lebendigkeit einer Variablen, 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_s eine Startzusicherung.
- c_q eine DFA-Anfrage.

Korrektheit und Vollständigkeit

Definition 8.9.2.1 (Korrektheit)

$\mathcal{S}_G^{\phi\,R}$ ist

1. *RJOP*-korrekt, wenn für alle c_q gilt:
   $[\![\, \mathbf{s} \,]\!]^{RJOP}_{\mathcal{S}_G^R, c_q} = c \;\Rightarrow\; [\![\, q \,]\!]^{MOP}_{\mathcal{S}_G, c} \;\sqsupseteq\; c_q.$
2. *RMOP*-korrekt, wenn für alle c_q gilt:
   $[\![\, \mathbf{s} \,]\!]^{RMOP}_{\mathcal{S}_G^R, c_q} = c \;\Rightarrow\; [\![\, q \,]\!]^{JOP}_{\mathcal{S}_G, c} \;\sqsupseteq\; c_q.$

Definition 8.9.2.2 (Vollständigkeit)

$\mathcal{S}_G^{\phi\,R}$ ist

1. *RJOP*-vollständig, wenn für alle $c_{\mathbf{s}}$ gilt:
   $[\![\, q \,]\!]^{MOP}_{\mathcal{S}_G, c_{\mathbf{s}}} \;\sqsupseteq\; c \Rightarrow [\![\, \mathbf{s} \,]\!]^{RJOP}_{\mathcal{S}_G^R, c} \;\sqsubseteq\; c_{\mathbf{s}}.$
2. *RMOP*-vollständig, wenn für alle $c_{\mathbf{s}}$ gilt:
   $[\![\, q \,]\!]^{JOP}_{\mathcal{S}_G, c_{\mathbf{s}}} \;\sqsupseteq\; c \Rightarrow [\![\, \mathbf{s} \,]\!]^{RMOP}_{\mathcal{S}_G^R, c} \;\sqsubseteq\; c_{\mathbf{s}}.$

Bemerkungen

...das Zusammenhangstheorem 8.9.1.1 liefert, das für distributive DFA-Probleme $\mathcal{S}_G^\phi$ das induzierte reverse DFA-Problem ${\mathcal{S}_G^\phi}^R$ korrekt und vollständig im Sinn von Definition 8.9.2.1 und 8.9.2.2 ist.

...ist $\mathcal{S}_G^\phi$ korrekt und vollständig für ϕ im Sinn von Definition 7.8.1 und 7.8.2 (und somit distributiv), so kann eine erschöpfende *MaxFP*-Analyse durch entsprechende (fortgesetzte) *RMinFP*-Analysen ersetzt werden; zugleich ist die Korrektheit anforderungsgetriebener Datenflussanalyse auf Grundlage reverser Datenflussanalyse gesichert.

Hausaufgabe

Was gilt für Korrektheit und Vollständigkeit im Sinn von Definition 8.9.2.1 und 8.9.2.2 für das induzierte reverse DFA-Problem $\mathcal{S}_G^{\phi^R}$ eines monotonen, aber nicht distributiven DFA-Problems $\mathcal{S}_G^{\phi}$? Was gilt für anforderungsgetriebene Datenflussanalyse auf Grundlage reverser Datenflussanalyse?

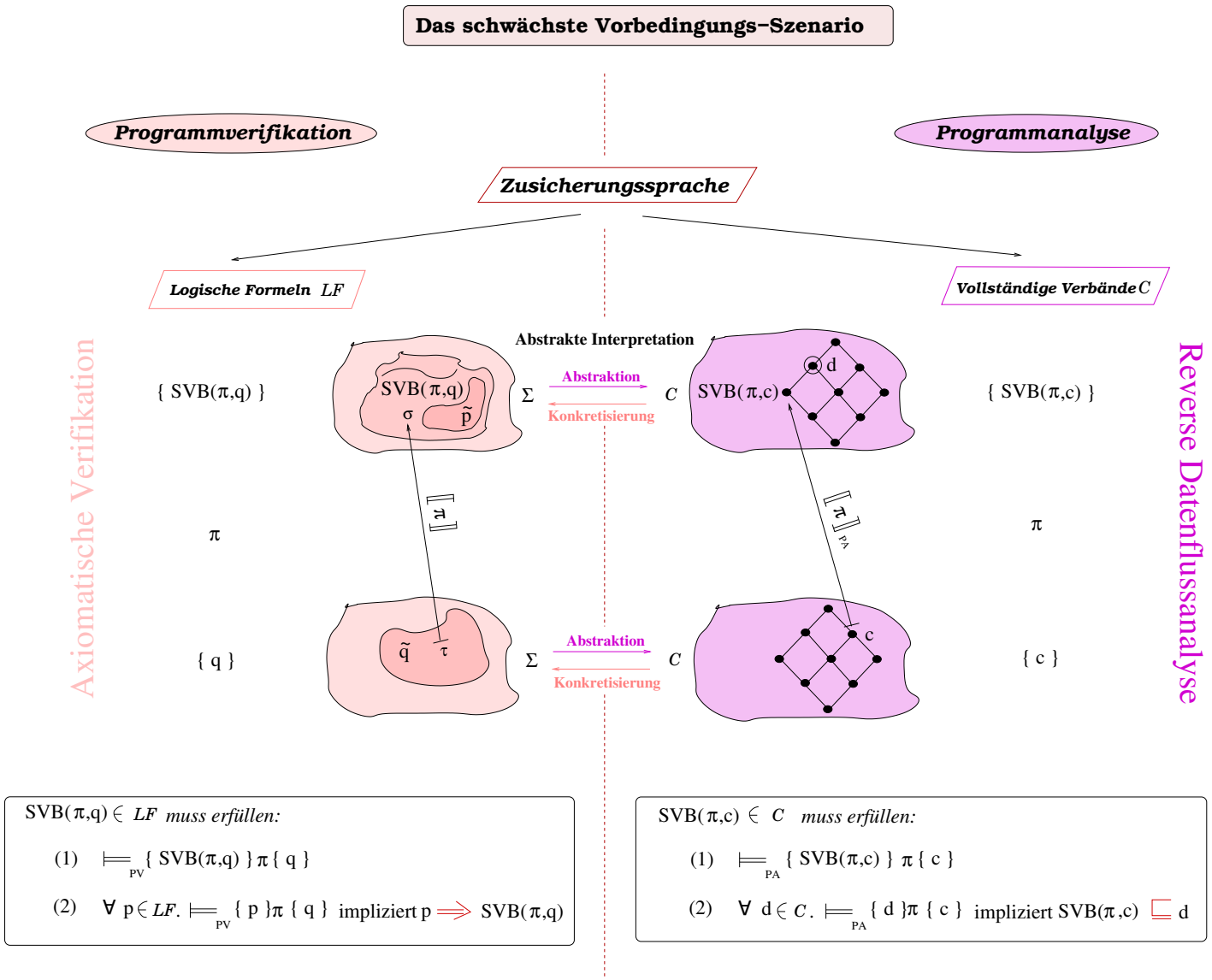
Beweisen Sie Ihre Behauptung.

Kapitel 8.9.3

Problemsichten, Analogien

Analogie von Verifikation und reverser DFA

...unter der Perspektive schwächster Vorbedingungen.



Drei unterschiedliche Problemsichten (1)

The ***specification problem:***

$\{?\}\,\pi\,\{q\}$

... the domain of ***reverse DFA***

The ***verification problem:***

$\{p\}\,\pi\,\{q\}\,?$

... the domain of ***demand–driven DFA***

The ***implementation problem:***

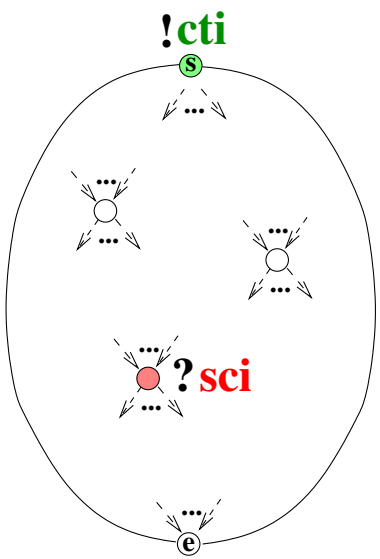
$\{p\}\,\pi\,\{?\}$

... the domain of ***exhaustive DFA***

Drei unterschiedliche Problemsichten (2)

...alternative Darstellung.

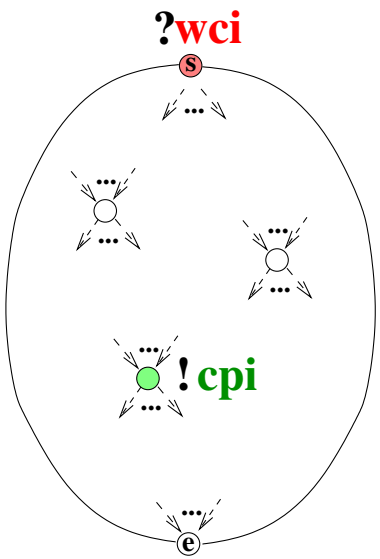
Implementation Problem



! **Given:** Context Information **cti**

? **Sought:** Strongest Component Information **sci**

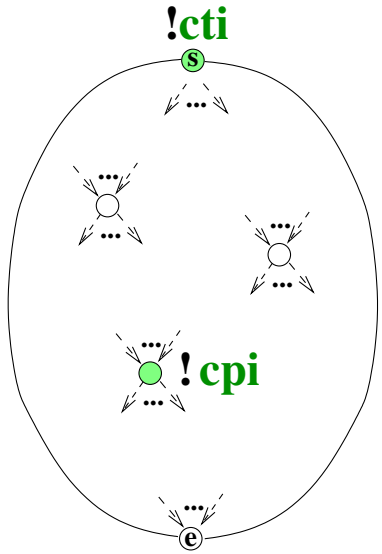
Specification Problem



! **Given:** Component Information **cpi**

? **Sought:** Weakest Context Information **wci**

Verification Problem



! **Given:** Context Information **cti**
Component Information **cpi**

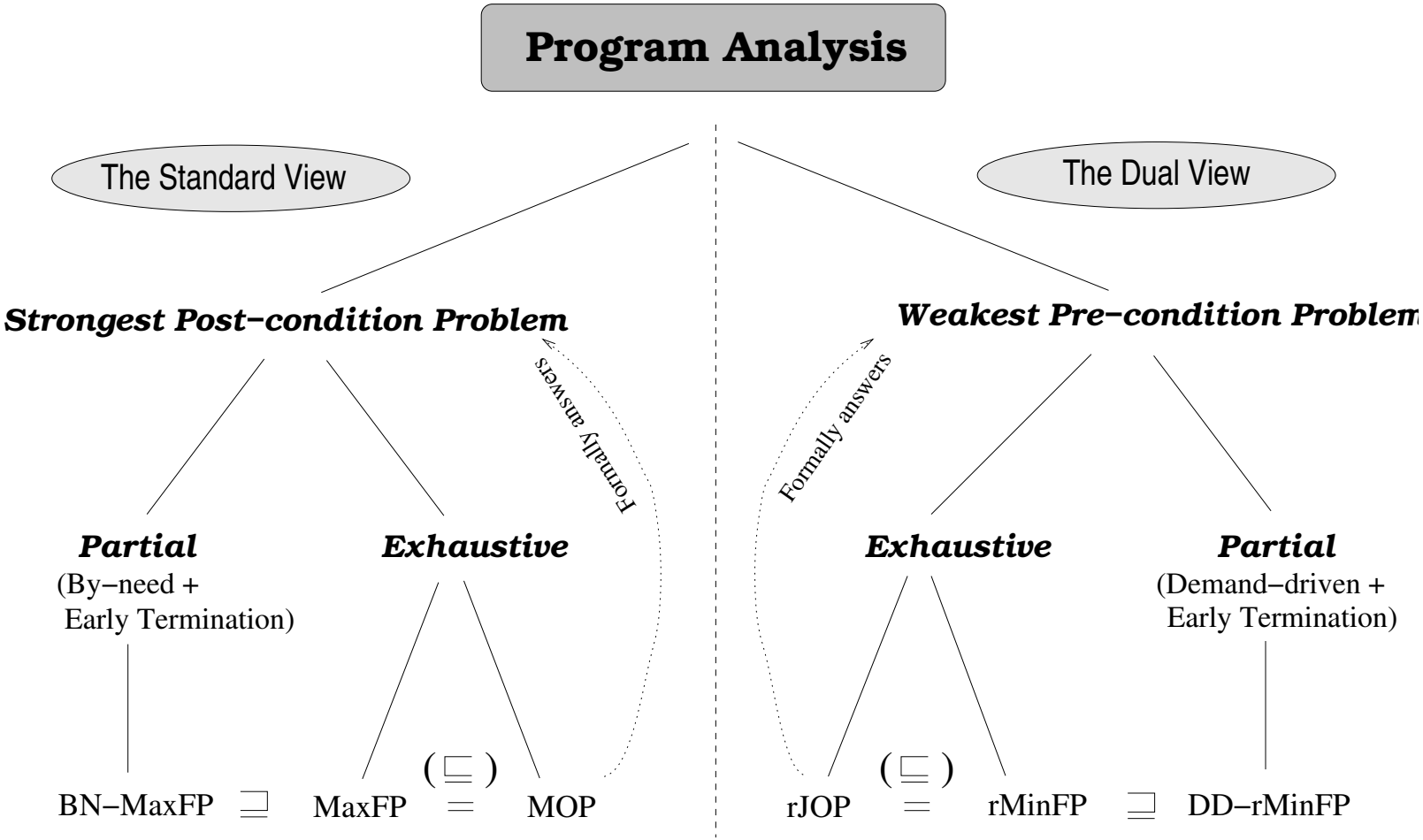
? **Sought:** Validity of **cpi** with respect of **cti**

Kapitel 8.10

Zusammenfassung, Ausblick

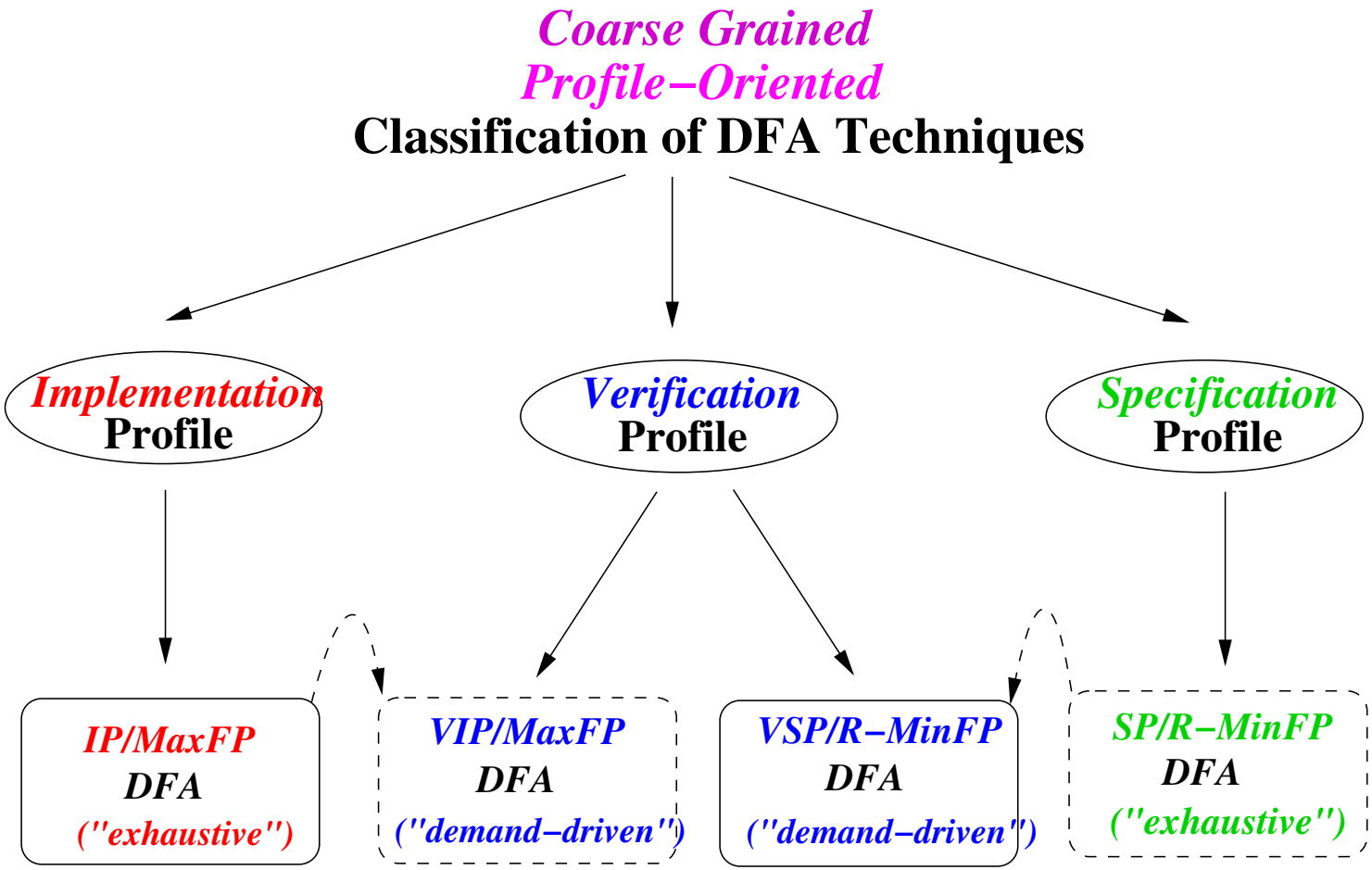
Gegenüberstellung der Problemsichten

...von erschöpfender (xDFA) und anforderungsgetriebener (ddDFA) Datenflussanalyse:



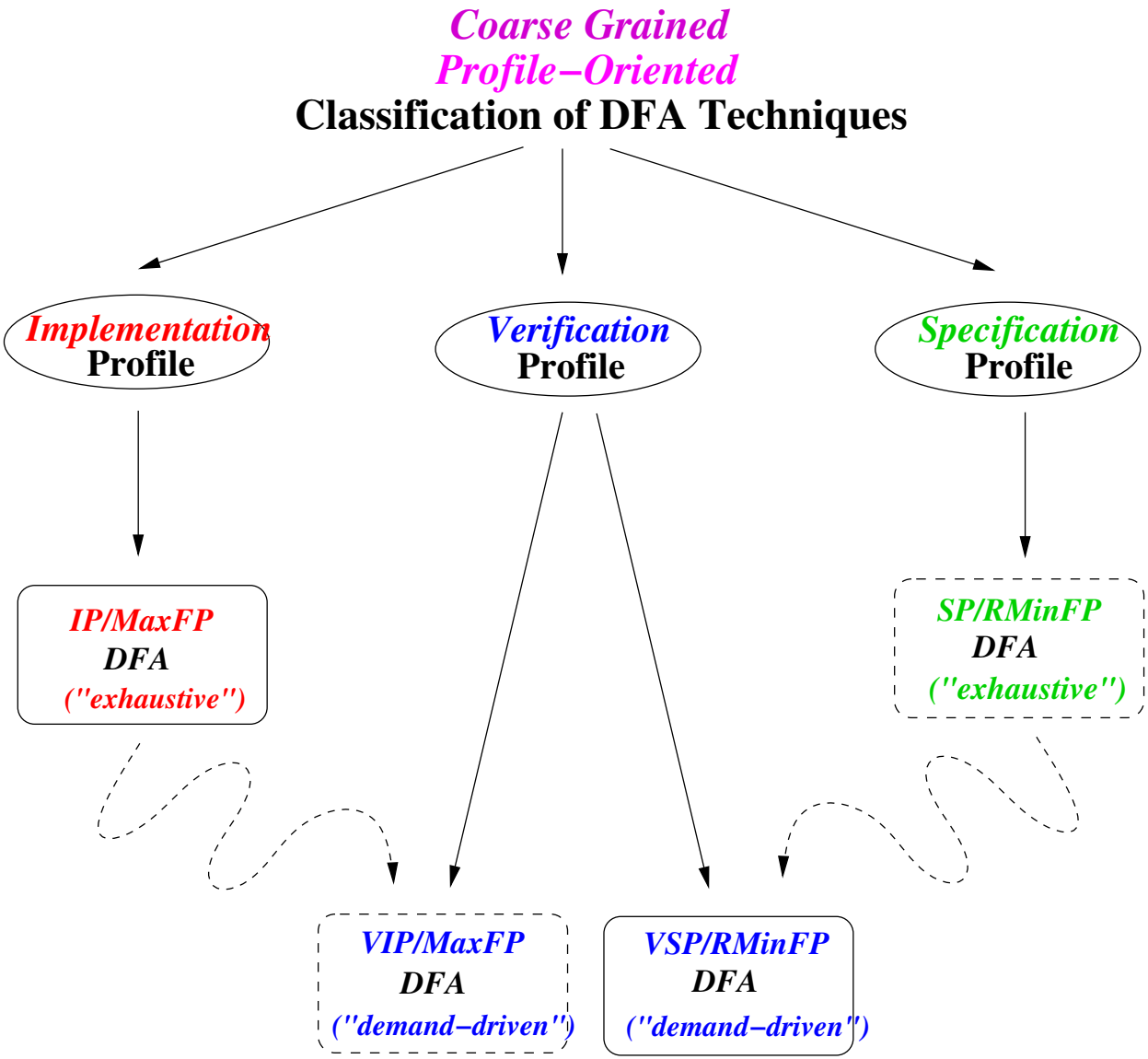
Abgeleitete Klassifikationen (1)

...und Lösungsstrategien aus den Problemsichten.



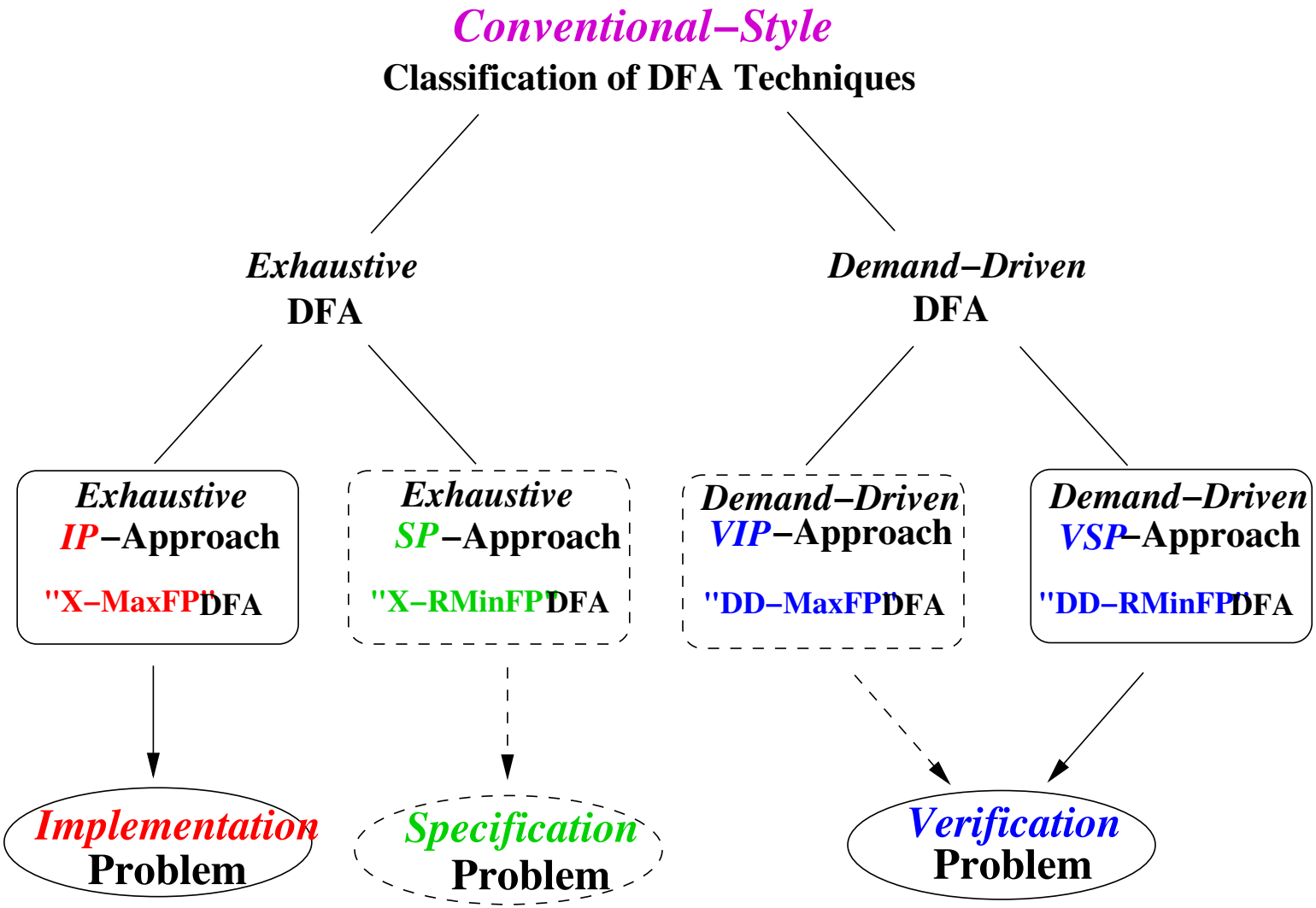
Abgeleitete Klassifikationen (2)

...und Lösungsstrategien aus den Problemsichten.



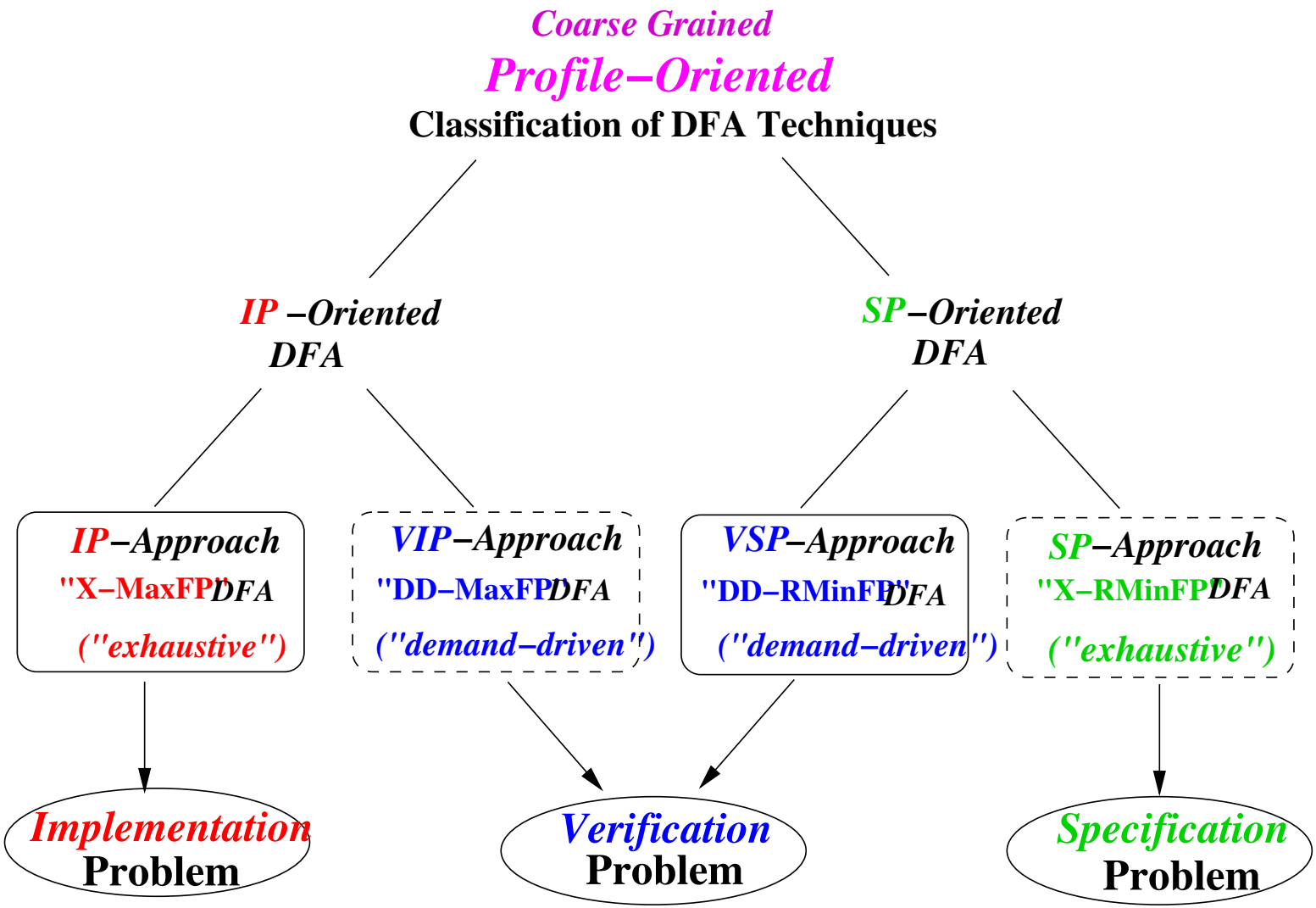
Klassifikation von DFA-Problemen (1)

...aus algorithmenorientierter Sicht.



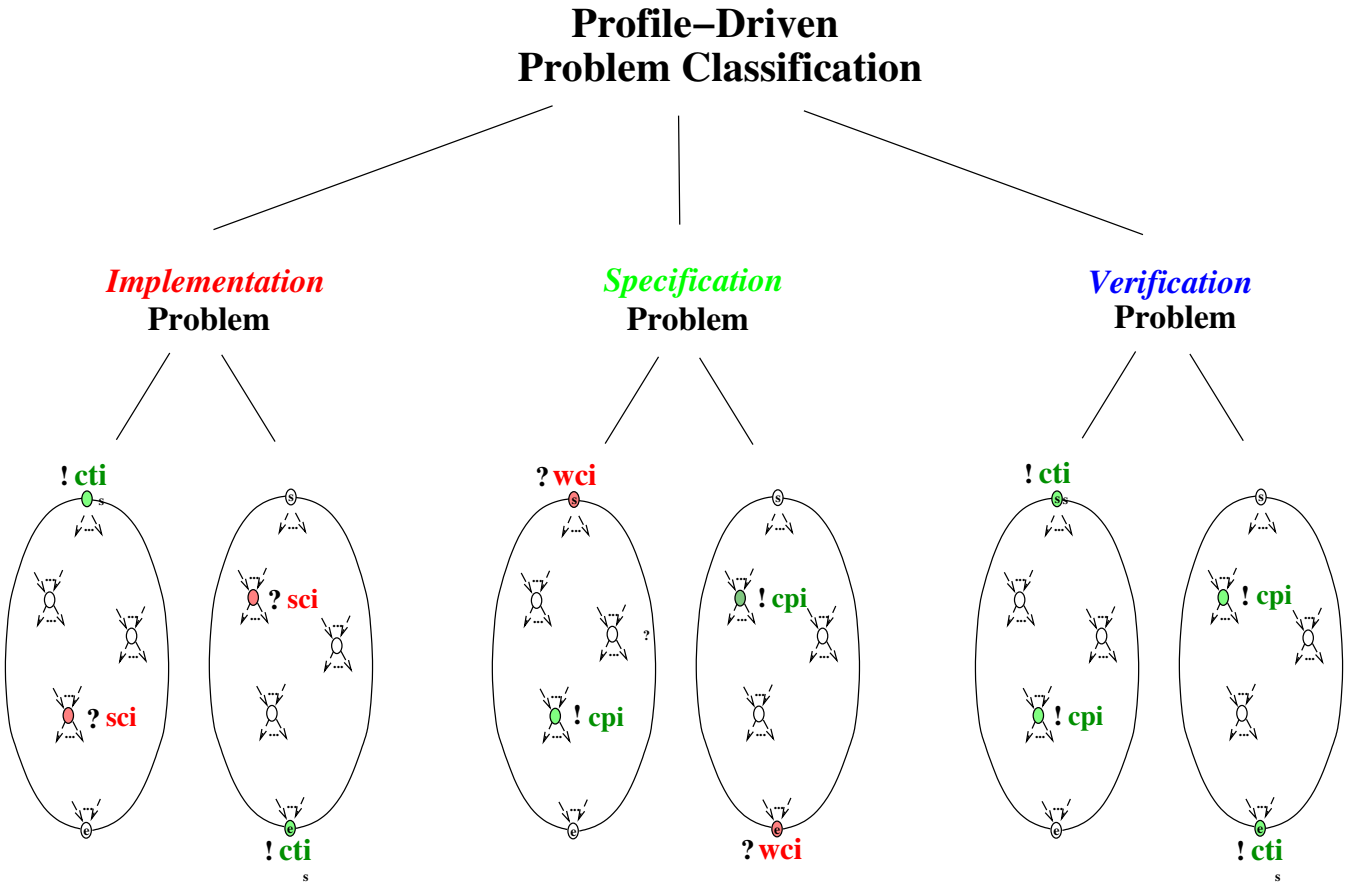
Klassifikation von DFA-Problemen (2)

...aus algorithmenorientierter Sicht.



Klassifikation von DFA-Problemen (3)

...aus Programm- und problemorientierter Sicht.



! cti : Context Information
? sci : Strongest Component Information

? wci : Weakest Context Information
! cpi : Component Information

! cti : Context Information
! cpi : Component Information

... where "!" and "?" means given and sought, respectively.

Kapitel 8.11

Literaturverzeichnis, Leseempfehlungen

Reading for Chapter 8 (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 8 (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 8 (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.


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.

Reading for Chapter 8 (4)


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.


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.

Reading for Chapter 8 (5)

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.

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.

Reading for Chapter 8 (6)


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.


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.

Reading for Chapter 8 (7)


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 9

Parallele Datenflussanalyse

Kapitel 9.1

Motivation

Syntax paralleler Programme

..wir erweitern die Menge der Sprachbefehle von WHILE um eine

- Parallelanweisung $||$

und nennen die erweiterte Sprache $\text{WHILE}_{||}$. Dabei treffen wir folgende Annahmen:

- Alle Komponenten einer Parallelanweisung werden parallel auf einem gemeinsamen Speicher (engl. shared memory) ausgeführt.
- Es gibt weder Sprünge von außen in eine Parallelanweisung hinein noch hinaus noch Sprünge von einer in eine andere Komponente einer Parallelanweisung.
- Eine Parallelanweisung terminiert dann und nur dann, wenn alle ihre Komponenten terminiert sind.
- Anweisungen sind atomar.

Semantik paralleler Programme

...die Bedeutung von $\text{While}_{||}$-Programmen ist durch eine Verschränkungs-Semantik (engl. interleaving semantics) gegeben.

Beachte: Mit diesen Festlegungen sind wesentliche Charakteristika paralleler Programme und paralleler Programmausführungen gegeben:

- Verschränkung der Ausführung von Anweisungen.
- Synchronisation von parallelen Komponenten.

Weitere Erweiterungen (z.B. dynamische Prozesserzeugung) sind möglich, werden in der Folge aber nicht betrachtet.

Herausforderung: Zustandsexplosionsproblem

...parallele Programme können als kompakte Darstellung nichtdeterministischer sequentieller Programme als Ergebnis einer

- Produktkonstruktion aus den parallelen Komponenten

aufgefasst werden.

Diese Transformation erlaubte die Analyse paralleler Programme auf die Analyse sequentieller Programme zurückzuführen, jedoch wächst die Grösse des nichtdeterministischen sequentiellen Produktprogramms

- exponentiell mit der Zahl paralleler Komponenten

so dass diese Rückführung nicht praktikabel ist.

Charakterisierendes Schlagwort:

Zustandsexplosionsproblem!

Hauptergebnis

...unidirektionale Bitvektoranalysen lassen sich für

- parallele Programme aus $\text{While}_{||}$

ebenso einfach und effizient durchführen wie für

- sequentielle Programme aus While.

Aufgrund der Vielzahl und hohen praktischen Relevanz unidirektionaler Programmanalysen für Analyse- und Optimierungszwecke ist dieses Resultat von hoher Wichtigkeit, da es Analyse- und Optimierungstechniken sequentieller Programme ohne Effizienzverlust auf

- parallele Programme

auszudehnen und zu übertragen erlaubt.

Kapitel 9.2

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 paralleler Datenflussanalyse.

Informell: Der funktionale denotationelle Semantik-Ansatz

- hebt das Niveau des *MaxFP*/*MinFP*-Ansatzes von einzelnen Verbandselementen als Startzusicherung auf das Niveau von Funktionen auf dem Gesamtverband.

In der Folge entwickeln wir den funktionalen denotationellen Semantik-Ansatz für die *MaxFP*-Sicht; die Entsprechung für den *MinFP*-Sicht ergibt sich durch Vertauschen von Schnitt- und Vereinigungsoperation des Verbands in trivialer Weise.

Der funktionale denotat. Semantik-Ansatz

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Programm und $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ eine DFA-Spezifikation.

Dann ist durch:

Gleichungssystem 9.2.1 (Funktionales *MaxFP*-GS)

$$f\text{-}inf(n) = \begin{cases} Id_{\mathcal{C}} & \text{falls } n = \mathbf{s} \\ \bigsqcap \{ [\![(n,m)]\!] \circ (f\text{-}inf(m)) \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

die punktweise funktionale Ausdehnung der denotationellen *MaxFP*-Semantik von G gegeben (vgl. Kapitel 7.5.1):

Gleichungssystem 9.2.2 (*MaxFP*-Gleichungssystem)

$$inf(n) = \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcap \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

Äquivalenzresultat

Bezeichnen wir mit

- $f\text{-}inf^{\star}_{\mathcal{C}}(n),\ n \in N$
- $inf^{\star}_{c_s}(n),\ n \in N$

die größten Lösungen der Gleichungssysteme 9.2.1 und 9.2.2, so gilt:

- $\forall\, n \in N.\ inf^{\star}_{c_s}(n) \in \mathcal{C}$
- $\forall\, n \in N.\ f\text{-}inf^{\star}_{\mathcal{C}}(n) \in [\mathcal{C} \to \mathcal{C}]$

und folgendes Äquivalenzresultat:

Theorem 9.2.3 (Äquivalenz)

$$\forall\, n \in N.\ \forall\, c_s \in \mathcal{C}.\ f\text{-}inf^{\star}_{\mathcal{C}}(n)(c_s) = inf^{\star}_{c_s}(n)$$

Hauptergebnis: Korrekth., Sicherh., Koinzidenz

Gemäß Theorem 9.2.3 stimmen die *MaxFP*- und funktionale *MaxFP*-Semantik überein und sind äquivalent.

Mit der weiteren Festlegung $[\![\ ]\!] =_{df} f\text{-}inf^{\star}_{\mathcal{C}}$ erhalten wir die Hauptergebnisse für die funktionale denotationale Semantik als Korollare zu Theorem 9.2.3, Sicherheitstheorem 7.7.1(1) und Koinzidenztheorem 7.7.2(1):

Korollar 9.2.4 (Äquivalenz)

$$\forall\, n \in N.\ \forall\, c_s \in \mathcal{C}.\ [\![\, n \,]\!](c_s) = [\![\, n \,]\!]^{MaxFP}_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)}(n)$$

Korollar 9.2.5 (Sicherheit und Koinzidenz)

1. Sicherheit: $\forall\, n \in N.\ \forall\, c_s \in \mathcal{C}.\ [\![\, n \,]\!](c_s) \sqsubseteq [\![\, n \,]\!]^{MOP}_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)}(n)$, wenn $[\![\ ]\!]$ monoton ist.
2. Koinzidenz: $\forall\, n \in N.\ \forall\, c_s \in \mathcal{C}.\ [\![\, n \,]\!](c_s) = [\![\, n \,]\!]^{MOP}_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)}(n)$, wenn $[\![\ ]\!]$ distributiv ist.

Berechnung der globalen Semantikfunktion $[\![\ ]\!]$

...die Funktion $[\![\ ]\!] : N \to (\mathcal{C} \to \mathcal{C})$ kann offensichtlich

- Argument für Argument mithilfe des Generischen Fixpunktalgorithmus 7.6.1.1 berechnet werden.

In der Folge stellen wir mit Algorithmus 9.2.6 ein weniger naives, direktes Verfahren zur Berechnung von $[\![\ ]\!]$ vor, das sich als Schlüssel zu paral- leler Datenflussanalyse herausstellen wird.

Der generische Fixpunktalgorithmus 9.2.6 (1)

Eingabe: Eine DFA-Spezifikation $\mathcal{S}_G = _{df} (\widehat{\mathcal{C}}, [\![\ ]\!])$ (alle Startzusicherungen werden simultan betrachtet).

Ausgabe: Nach Terminierung von Algorithmus 9.3.6 (s. Terminationstheorem 9.2.7) speichert Variable $f\text{-}inf[n]$ die funktionale *MaxFP*-Semantik bzgl. $\mathcal{S}_G$ am Knoten n.

Zusätzlich gilt (s. Korollar 9.2.5(1) und 9.2.5(2)): Ist $[\![\ ]\!]$

- distributiv: $f\text{-}inf[n]$ speichert
- monoton: $f\text{-}inf[n]$ speichert eine untere Approximation

der (funktionalen) *MOP*-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.

Der generische Fixpunktalgorithmus 9.2.6 (2)

(Prolog: Initialisierung von $f\text{-}inf$ und $workset$)
FORALL $n \in N \backslash \{\mathbf{s}\}$ DO $f\text{-}inf[n] := \lambda c.\ \top$ OD;
$f\text{-}inf[\mathbf{s}] := \lambda c.\ c$;
$workset := N$;
(Hauptprozess: Iterative Fixpunktberechnung)
WHILE $workset \neq \emptyset$ DO
    CHOOSE $m \in workset$;
        $workset := workset \backslash \{ m \}$;
        (Annotationsaktualisierung der Nachfolgerknoten von m)
        FORALL $n \in succ(m)$ DO
            $meet := [\![(m, n)]\!] \circ f\text{-}inf[m] \sqcap f\text{-}inf[n]$;
            IF $f\text{-}inf[n] \sqsupset meet$
                THEN
                    $f\text{-}inf[n] := meet$;
                    $workset := workset \cup \{ n \}$
            FI
        OD ESOOHC OD.

Termination

Theorem 9.2.7 (Termination)

Der Generische Fixpunktalgorithmus 9.2.6 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 9.2.8

Erfüllt $[\mathcal{C} \to \mathcal{C}]$ die absteigende Kettenbedingung, so auch $\mathcal{C}$.

Kapitel 9.3

Parallele Flussgraphen

Parallele Flussgraphen

Wir repräsentieren parallele Programme aus $\text{While}_{||}$ in Form

- knotenbenannter paralleler Flussgraphen

$$G^* = (N^*, E^*, \mathbf{s}^*, \mathbf{e}^*)$$

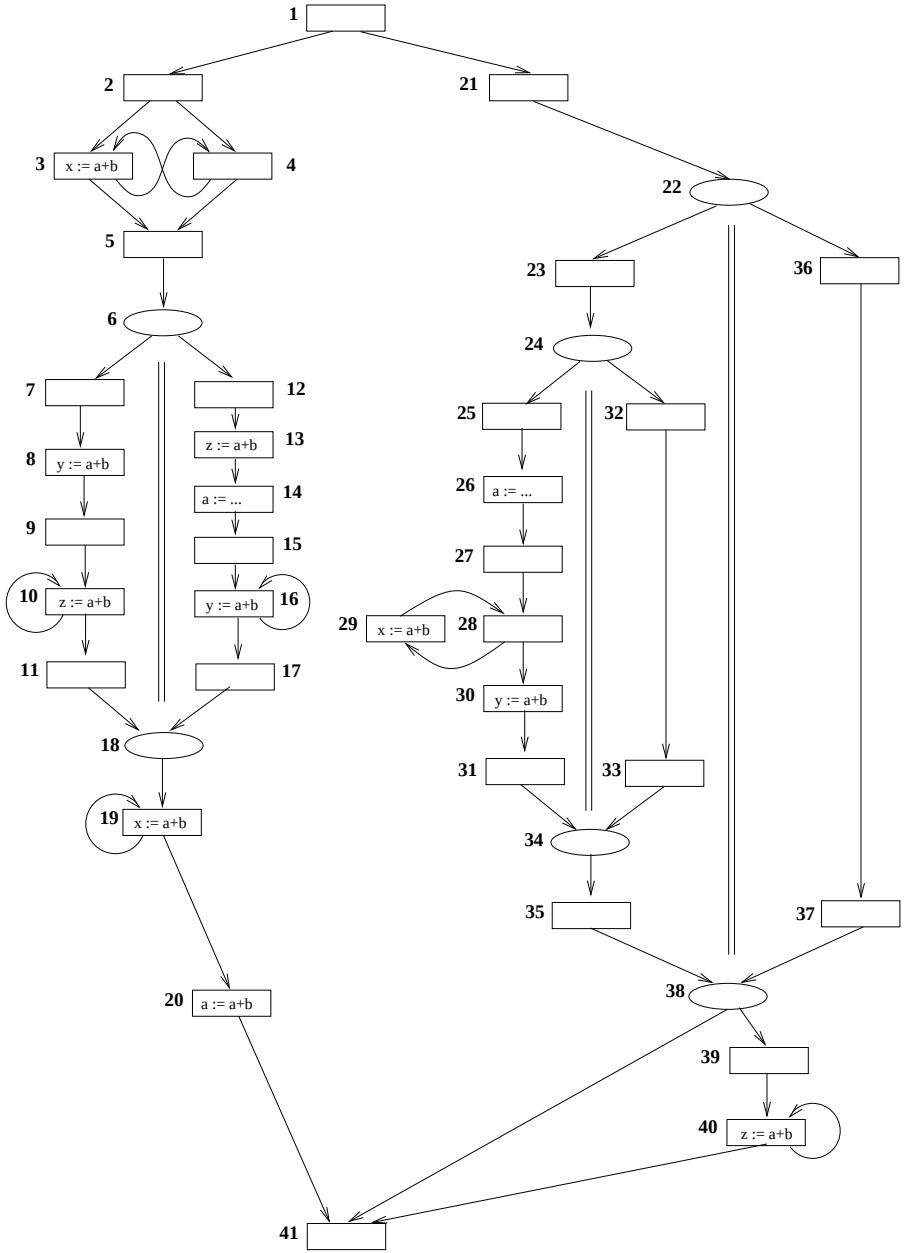
Dabei gilt: Teilgraphen von G^*, die

- 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), wobei die Ein- bzw. Austrittspunkte paralleler Anweisungen 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.

Durchgehendes Beispiel: Paralleler Flussgraph



Kapitel 9.3.1

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 (oder äußerster) und minimaler (oder 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)

Abbildung *pfg* (*cfg*) ordnet jedem Knoten aus einem Flussgraphen $G \in \mathcal{G}_{\mathcal{P}}(G^*)$ ($G \in \mathcal{G}_{\mathcal{C}}(G^*)$) den kleinsten n enthaltenden Flussgraphen aus $\mathcal{G}_{\mathcal{P}}(G^*)$ ($G \in \mathcal{G}_{\mathcal{C}}(G^*)$) 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 9.3.2

Rang paralleler Graphen

Rang paralleler (Teil-) Graphen

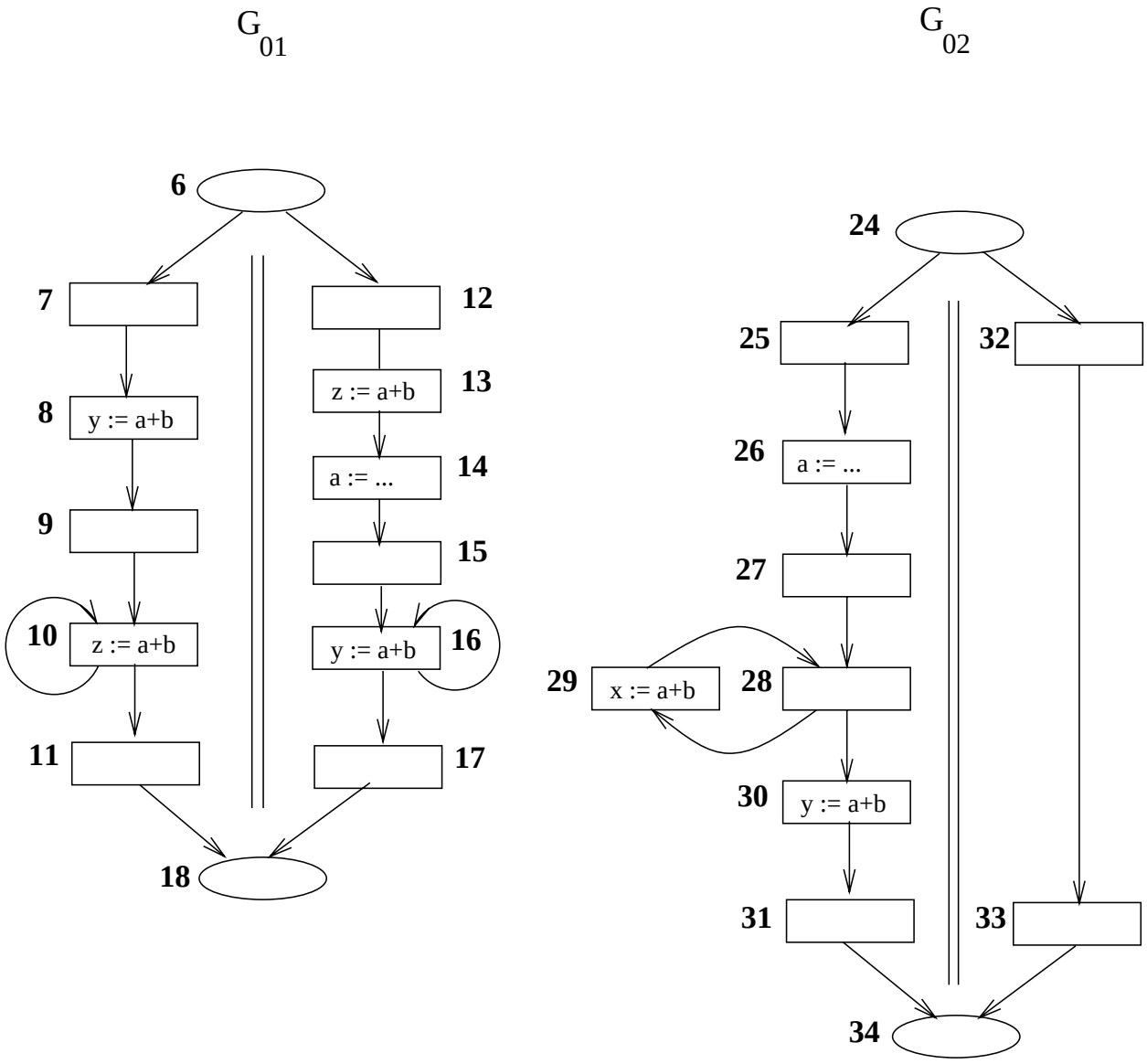
Sei G^* ein paralleles Programm.

Definition 9.3.2.1 (Rang paralleler Graphen)

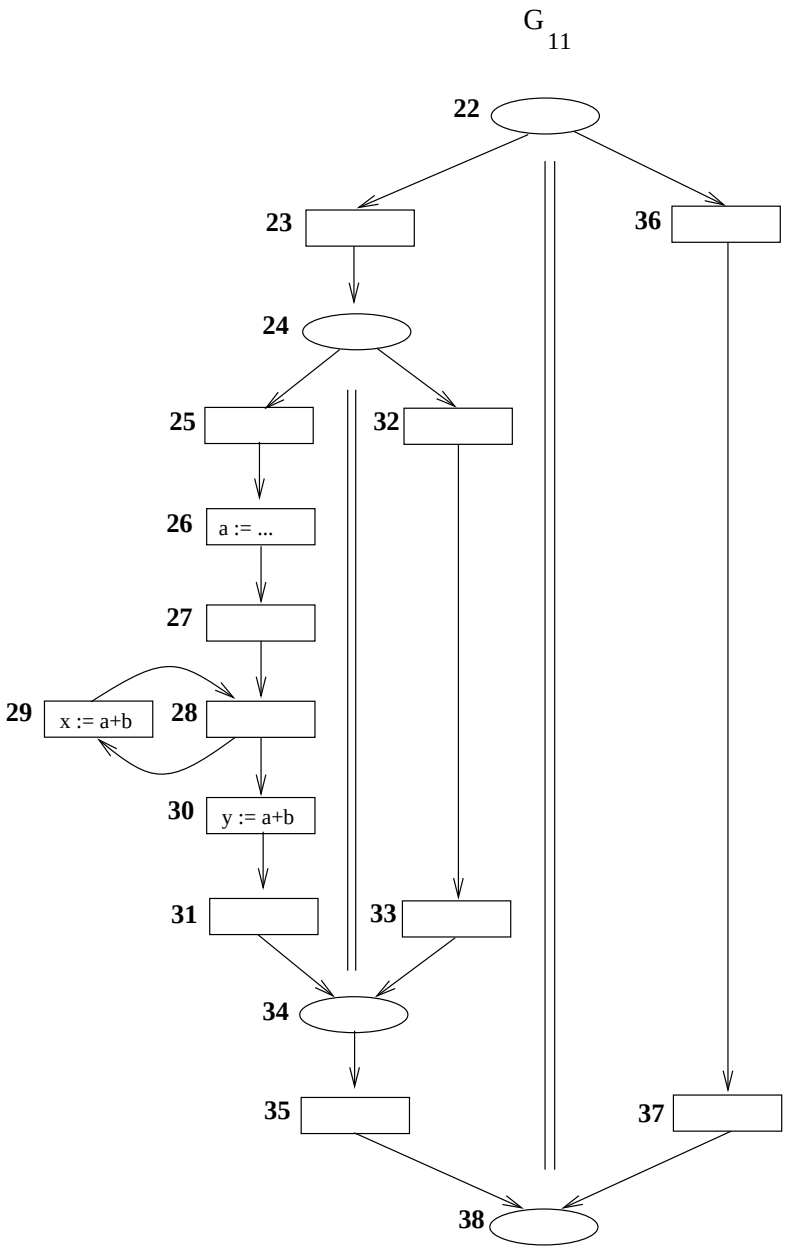
Der Rang (engl. rank) eines parallelen Graphen $G \in \mathcal{G}_{\mathcal{P}}(G^*)$ ist definiert durch:

$$rank(G) =_{df} \begin{cases} 0 & \text{falls } G \in \mathcal{G}^{min}_{\mathcal{P}}(G^*) \\ max\{\, rank(G') \mid G' \in \mathcal{G}_{\mathcal{P}}(G^*) \wedge G' \subset G \,\} + 1 & \text{sonst} \end{cases}$$

Lfd. Bsp.: Zwei parallele Graphen von Rang 0



Lfd. Bsp.: Ein paralleler Graph von Rang 1



Kapitel 9.3.3

Sequentialisierte Graphen

Sequentialisierte Graphen

...eines parallelen Flussgraphen.

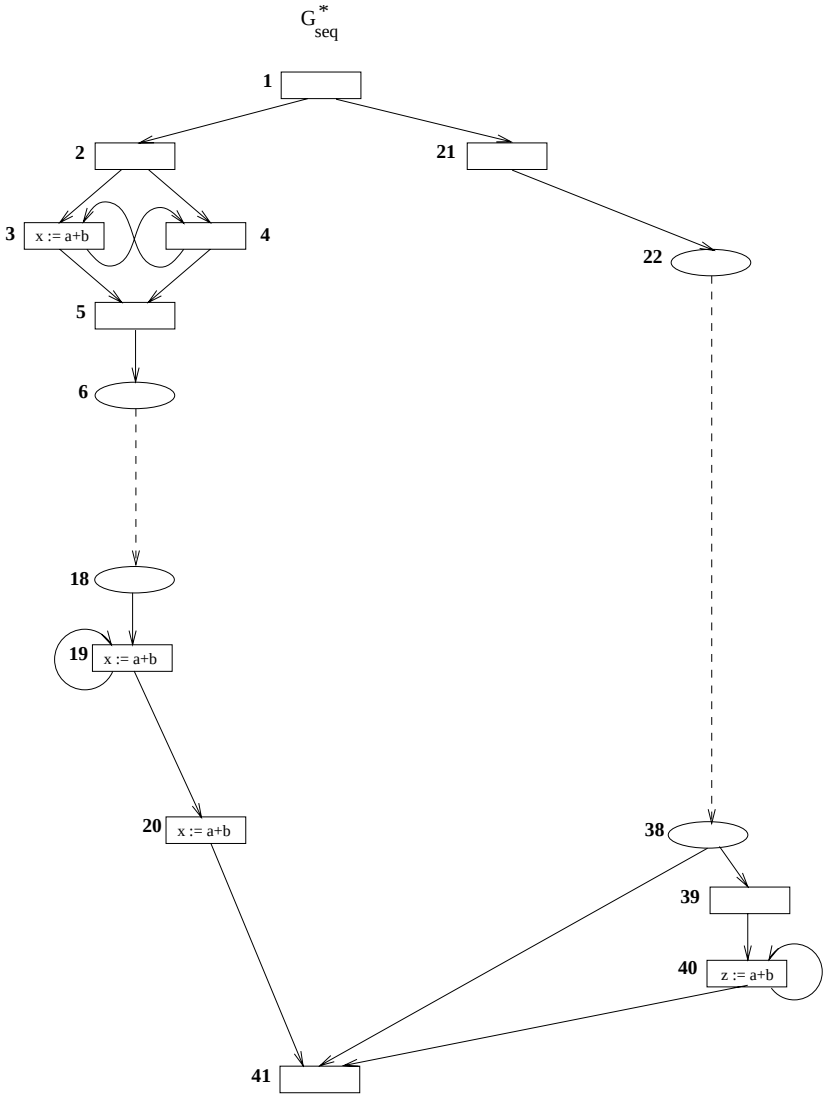
Definition 9.3.3.1 (Sequentialisierter Graph)

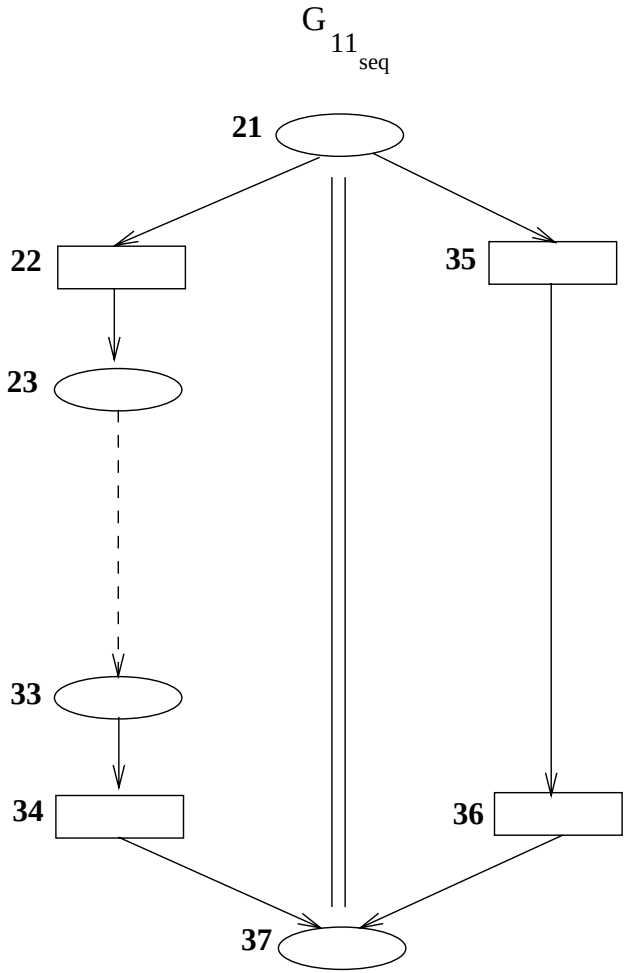
Ist $G \in \mathcal{G}_{\mathcal{P}}(G^*)$, so ist G_{seq} der G zugeordnete (triviale nicht bedeutungsgleiche) sequentialisierte Flussgraph, in dem alle maximalen parallelen Teilgraphen $G' \in \mathcal{G}_{\mathcal{P}}^{max}(G)$ durch eine Kante von $start(G')$ nach $end(G')$ ersetzt sind.

Beachte: Sequentialisierte Graphen enthalten keine parallelen Teilgraphen.

Laufendes Beispiel: Sequentialisierte Graphen

...des durchgehenden Beispiels und des maximalen parallelen Teilgraphen G_{11}:





Kapitel 9.3.4

Verschränkte Vorgänger

Parallele Geschwistergraphen

Sei G^* ein paralleles Programm.

Definition 9.3.4.1 (Parallele Geschwistergraphen)

Die Menge der parallelen Geschwistergraphen (engl. parallel brothers and sisters) eines Komponentengraphen $G \in \mathcal{G}_C(G^*)$ einer Parallelanweisung ist gegeben durch:

$$PG_{BaS}(G) =_{df} \mathcal{G}_C(pfg(G)) \setminus G \cup \begin{cases} \emptyset & \text{falls } pfg(G) \in \mathcal{G}_{\mathcal{P}}^{max}(G^*) \\ PG_{BaS}(cfg(pfg(G))) & \text{sonst} \end{cases}$$

Verschränkte Vorgänger

...die Menge der unmittelbar vor einem Knoten in einer Parallelanweisung dynamisch ausführbaren Knoten ist durch die Vereinigung seiner statischen (Komponenten-) Vorgänger i.S.v. Kapitel 7 und seiner sog. verschränkten Vorgänger gegeben:

Definition 9.3.4.2 (Verschränkte Vorgänger)

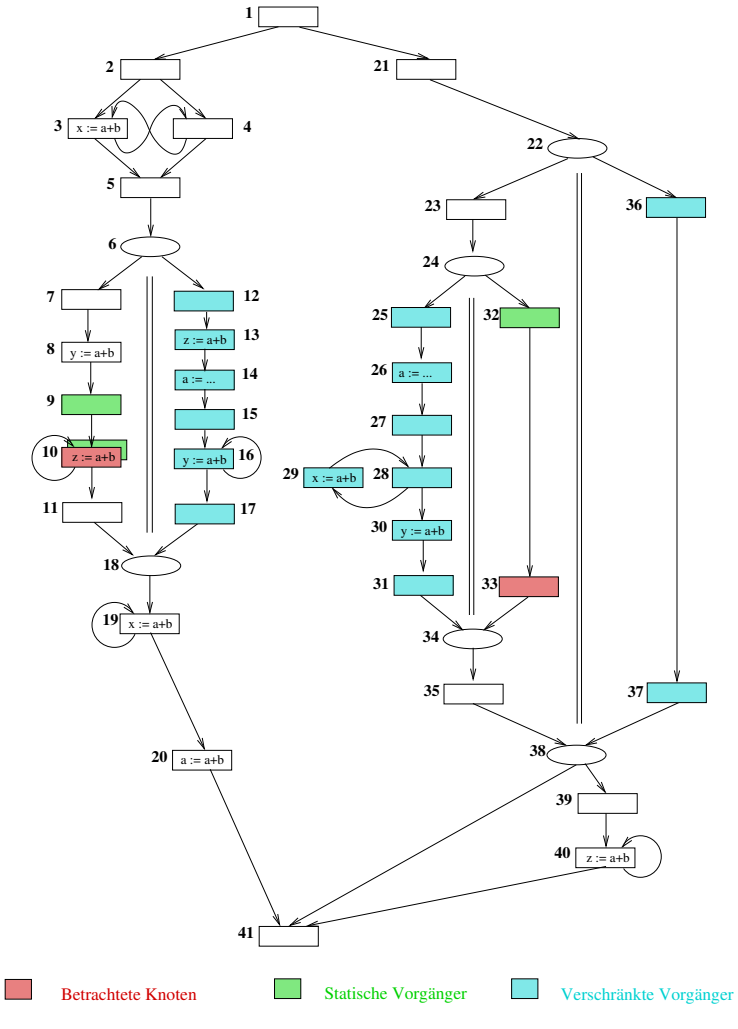
Ist G^* ein paralleles Programm, so ist die Menge der verschränkten Vorgänger (engl. interleaving predecessors) eines Knotens n aus G^* ist gegeben durch:

$$Pred^{Itlvg}_{G^*}(n) =_{df} \begin{cases} Nodes(PG_{BaS}(cfg(n))) & \text{falls } n \in Nodes(\mathcal{G}_{\mathcal{C}}(G^*)) \\ \emptyset & \text{sonst} \end{cases}$$

Lfd. Bsp: Statische, verschränkte Vorgänger

...der Knoten 9 und 33:

- $pred_{G^*}(10) = \{9, 10\}$, $Pred^{ltlvg}_{G^*}(10) = \{12, ..., 17\}$.
- $pred_{G^*}(33) = \{32\}$, $Pred^{ltlvg}_{G^*}(33) = \{25, ..., 31, 36, 37\}$.



Kapitel 9.3.5

Parallele Pfade

G-wohlgeformte Knotenfolgen

Sei G^* ein paralleles Programm

Definition 9.3.5.1 (G-wohlgeformte Knotenfolge)

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.

Beachte: Bed. 2.3 stellt Synchronisation sicher: Am Austrittsknoten einer Parallelanweisung müssen alle ihre parallelen Komponenten vollständig ausgeführt und terminiert sein.

Parallele Pfade

Sei G^* ein paralleles Programm

Definition 9.3.5.2 (Paralleler Pfad)

Eine Knotenfolge $p = (n_1, \ldots, n_q)$ aus G^* heißt paralleler Pfad von

1. $\mathbf{s}^*$ nach $\mathbf{e}^*$ gdw p ist G^*-wohlgeformt.
2. n_1 nach n_q, wenn p Teilpfad eines parallelen Pfades von $\mathbf{s}^*$ nach $\mathbf{e}^*$ ist.

Wir bezeichnen mit:

- $\mathbf{PP}_{G^*}[m, n]$: Menge aller parallelen Pfade von m nach n.
- $\mathbf{PP}_{G^*}[m, n[$: Menge aller parallelen Pfade von m zu einem (statischen oder verschränkten) Vorgänger von n:

$$\mathbf{PP}_{G^*}[m, n[=_{df} \{(n_1, \ldots, n_q) \mid (n_1, \ldots, n_q, n_{q+1}) \in \mathbf{PP}_{G^*}[m, n]\}$$

Kapitel 9.4

Operationelle globale parallele DFA-Semantik

Kapitel 9.4.1

Parallele Aufsammelsemantik

Die parallele Aufsammelsemantik

Sei $\mathcal{S}_{G^*} =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ eine DFA-Spezifikation.

Definition 9.4.1.1 (Parallele DFA-Aufsammelsem.)

Die von $\mathcal{S}_{G^*}$ induzierte (nichtdeterministische) parallele DFA-Aufsammelsemantik (oder globale abstrakte Semantik) an Knoteneingängen ist definiert durch:

$$[\![\ ]\!]^{PCS}_{\mathcal{S}_{G^*}} : N \to \mathcal{P}(\mathcal{C})$$

$$\forall\, n \in N.\ [\![\, n \,]\!]^{PCS}_{\mathcal{S}_{G^*}} =_{df} \{\, [\![\, p \,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{PP}[\mathbf{s}^*, n[\,\}$$

wobei $\mathcal{P}$ den Potenzmengenoperator bezeichnet.

Beachte:

$$[\![\, \mathbf{s}^* \,]\!]^{PCS}_{\mathcal{S}_{G^*}} = \{c_{\mathbf{s}}\}$$

Kapitel 9.4.2

Schnitt-über-alle-parallele-Pfade-Semantik

Die Schnitt-über-alle-par.-Pfade (*SUPP*) Sem.

Sei $\mathcal{S}_{G^*} =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ eine DFA-Spezifikation.

Definition 9.4.2.1 (*SUPP*-Semantik)

Die deterministische *SUPP*-Semantik (engl. *MOPP* semantics) von $\mathcal{S}_{G^*}$ an Knoteneingängen ist definiert durch:

$$[\![\ ]\!]^{SUPP}_{\mathcal{S}_{G^*}} : N^* \to \mathcal{C}$$

$$\begin{aligned} \forall\, n \in N^*.\ [\![\, n \,]\!]^{SUPP}_{\mathcal{S}_{G^*}} &=_{df} \bigsqcap [\![\, n \,]\!]^{PCS}_{\mathcal{S}_{G^*}} \\ &= \bigsqcap \{\, [\![\, p \,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{PP}[\mathbf{s}^*, n[\, \} \end{aligned}$$

Beachte: $\bigsqcap [\![\, n \,]\!]^{PCS}_{\mathcal{S}_{G^*}}$ und folglich $[\![\, n \,]\!]^{SUPP}_{\mathcal{S}_{G^*}}$, $n \in N^*$, existieren, da $\widehat{\mathcal{C}}$ ein vollständiger Verband ist.

Kapitel 9.4.3

Vereinigung-über-alle-parallele-Pfade-Semantik

Die Verein.-über-alle-par.-Pfade (*VUPP*) Sem.

Sei $\mathcal{S}_{G^*} =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ eine DFA-Spezifikation.

Definition 9.4.3.1 (*VUPP*-Semantik)

Die deterministische *VUPP*-Semantik (engl. *JOPP* semantics) von $\mathcal{S}_{G^*}$ an Knoteneingängen ist definiert durch:

$$[\![\ ]\!]^{VUPP}_{\mathcal{S}_{G^*}} : N^* \to \mathcal{C}$$

$$\begin{aligned} \forall\, n \in N^*.\ [\![\, n \,]\!]^{VUPP}_{\mathcal{S}_{G^*}} &=_{df} \bigsqcup [\![\, n \,]\!]^{PCS}_{\mathcal{S}_{G^*}} \\ &= \bigsqcup \{\, [\![\, p \,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{PP}[\mathbf{s}^*, n[\, \} \end{aligned}$$

Beachte: $\bigsqcup [\![\, n \,]\!]^{PCS}_{\mathcal{S}_{G^*}}$ und folglich $[\![\, n \,]\!]^{VUPP}_{\mathcal{S}_{G^*}}$, $n \in N^*$, existieren, da $\widehat{\mathcal{C}}$ ein vollständiger Verband ist.

Kapitel 9.5

Denotationelle globale parallele DFA-Semantik

Kapitel 9.5.1

Unidirektionale Bitvektoranalysen

Lokale DFA-Semantikfunktionen

...unidirektionaler Bitvektoranalysen sind stets Element der Menge

$$\mathcal{F}_{\mathbb{B}} =_{df} \{Cst_{\mathbf{wahr}}, Id_{\mathbb{B}}, Cst_{\mathbf{falsch}}\} \subseteq [\mathbb{B} \to \mathbb{B}]$$

mit

- $Cst_{\mathbf{wahr}} =_{df} \lambda b \in \mathbb{B}.\ \mathbf{wahr}$
- $Id_{\mathbb{B}} =_{df} \lambda b \in \mathbb{B}.\ b$
- $Cst_{\mathbf{falsch}} =_{df} \lambda b \in \mathbb{B}.\ \mathbf{falsch}$

Beachte: 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.

Proposition 9.5.1.1

1. Alle Funktionen aus $\mathcal{F}_{\mathbb{IB}}$ sind distributiv und additiv.
2. $\mathcal{F}_{\mathbb{IB}}$ bildet mit der punktweisen Ordnung auf Funktionen einen vollständigen Verband mit kleinstem Element $Cst_{\mathbf{falsch}}$ und größtem Element $Cst_{\mathbf{wahr}}$, der unter Funktionskomposition abgeschlossen ist.

Hauptlemma 9.5.1.2

Seien $f_i : \mathcal{F}_{\mathbb{IB}} \to \mathcal{F}_{\mathbb{IB}}$, $1 \leq i \leq q$, $q \in \mathbb{N}$, Funktionen von $\mathcal{F}_{\mathbb{IB}}$ nach $\mathcal{F}_{\mathbb{IB}}$. Dann gilt:

$$\exists\, k \in \{1, ..., q\}.\ f_q \circ \ldots \circ f_2 \circ f_1 = f_k \ \wedge\ \forall\, j \in \{k+1, ..., q\}.\ f_j = Id_{\mathbb{IB}}$$

Kapitel 9.5.2

Interferenz und Synchronisation

Interferenz paralleler Komponenten

...sei $\mathcal{S}_{G^*} =_{df} (\widehat{\mathcal{F}_{\mathbb{IB}}}, [\![\ ]\!])$ DFA-Spezifikation eines unidirektionalen Bitvektorproblems, n ein Knoten in einer Parallelanweisung und E die durch $\mathcal{S}_{G^*}$ spezifizierte Eigenschaft.

Definition 9.5.2.1 (Interferenzfest, interferenzlabil)

Eigenschaft E heißt

1. *SUPP*-interferenzfest (engl. interference-proof) am Knoten n (in Zeichen: $\mathit{Interf\text{-}proof}_n^{SUPP}$) gdw E kann nicht durch einen Verschränkungsvorgänger von n zerstört werden.
2. *VUPP*-interferenzfest am Knoten n (in Zeichen: $\mathit{Interf\text{-}proof}_n^{VUPP}$) gdw E kann nicht durch einen Verschränkungsvorgänger von n erzeugt werden.

E heißt interferenzlabil (in Zeichen: $\mathit{Interf\text{-}labile}_n^{SUPP}$) gdw E ist nicht interferenzfest.

Interferenzkorrektheit (1)

...als Folgerung aus Hauptlemma 9.5.1.2, wonach für unidirektionale Bitvektorprobleme der Effekt eines ganzen Pfads vom Effekt einer einzelnen Anweisung abhängt, erhalten wir:

Lemma 9.5.2.2 (Interferenzfest-Lemma)

Für alle $\forall\, n \in N^*$ gilt:

1. $$\mathit{Interf\text{-}proof}_n^{SUPP} \iff \neg\mathit{Interf\text{-}labile}_n^{SUPP} \iff \forall\, m \in \mathit{Pred}_{G^*}^{ltlvg}(n).\ [\![\, m \,]\!] \in \{\mathit{Cst}_{\mathbf{wahr}}, \mathit{Id}_{\mathbb{B}}\}$$
2. $$\mathit{Interf\text{-}proof}_n^{VUPP} \iff \neg\mathit{Interf\text{-}labile}_n^{VUPP} \iff \forall\, m \in \mathit{Pred}_{G^*}^{ltlvg}(n).\ [\![\, m \,]\!] \in \{\mathit{Cst}_{\mathbf{falsch}}, \mathit{Id}_{\mathbb{B}}\}$$

Interferenzkorrektheit (2)

Lemma 9.5.2.3 (Interferenzlabil-Lemma)

Für alle $\forall\, n \in N^*$ gilt:

1. $$Interf\text{-}labile_n^{SUPP} \iff \neg Interf\text{-}proof_n^{SUPP} \iff \exists\, m \in Pred_{G^*}^{ltlvg}(n).\ [\![\, m \,]\!] = Cst_{\textbf{falsch}}$$
2. $$Interf\text{-}labile_n^{VUPP} \iff \neg Interf\text{-}proof_n^{VUPP} \iff \exists\, m \in Pred_{G^*}^{ltlvg}(n).\ [\![\, m \,]\!] = Cst_{\textbf{wahr}}$$

Synchronisation paralleler Komponenten (1)

...ein 4-stufiges 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 $[\![G']\!]^{SUPP}_{\mathcal{S}_{\mathcal{G}^*}}$ bzw. $[\![G']\!]^{VUPP}_{\mathcal{S}_{\mathcal{G}^*}}$ dieser (rein sequentiellen) Graphen mittels Algorithmus 9.2.6.

Synchronisation paralleler Komponenten (2)

3. Synchronisationsschritt: Berechne für alle innersten parallelen Anweisungen $\bar{G}$ von G die *SUPP*- bzw. *VUPP*-Semantik gemäß:

$$[\![\bar{G}]\!]^{*\text{-}SUPP}_{\mathcal{S}_{\mathcal{G}^*}} =$$

$$\begin{cases} Cst_{\mathbf{falsch}} & \text{falls } \exists\, G' \in \mathcal{G}_{\mathcal{C}}(\bar{G}).\ [\![end(G')]\!] = Cst_{\mathbf{falsch}} \\ Id_{\mathbb{B}} & \text{falls } \forall\, G' \in \mathcal{G}_{\mathcal{C}}(\bar{G}).\ [\![end(G')]\!] = Id_{\mathbb{B}} \\ Cst_{\mathbf{wahr}} & \text{sonst} \end{cases}$$

$$[\![\bar{G}]\!]^{*\text{-}VUPP}_{\mathcal{S}_{\mathcal{G}^*}} =$$

$$\begin{cases} Cst_{\mathbf{wahr}} & \text{falls } \exists\, G' \in \mathcal{G}_{\mathcal{C}}(\bar{G}).\ [\![end(G')]\!] = Cst_{\mathbf{wahr}} \\ Id_{\mathbb{B}} & \text{falls } \forall\, G' \in \mathcal{G}_{\mathcal{C}}(\bar{G}).\ [\![end(G')]\!] = Id_{\mathbb{B}} \\ Cst_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

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 trivialen sequentiellen 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:

- $[\![\bar{\mathbf{s}}]\!]^{SUPP}_{\mathcal{S}_{\mathcal{G}^*}} = Id_{\mathbb{B}} \sqcap \bigsqcap \{ [\![n]\!] \mid n \in \bar{N} \}$, $[\![\bar{\mathbf{e}}]\!] = [\![\bar{G}]\!]^{SUPP}_{\mathcal{S}_{\mathcal{G}^*}}$
- $[\![\bar{\mathbf{s}}]\!]^{VUPP}_{\mathcal{S}_{\mathcal{G}^*}} = Id_{\mathbb{B}} \sqcup \bigsqcup \{ [\![n]\!] \mid n \in \bar{N} \}$, $[\![\bar{\mathbf{e}}]\!] = [\![\bar{G}]\!]^{VUPP}_{\mathcal{S}_{\mathcal{G}^*}}$

Fahre mit Schritt 1 fort.

Synchronisationskorrektheit (1)

...Basis für die Korrektheit des Synchronisationsschritts ist Lemma 9.5.2.4 für innerste parallele Anweisungen, das wie die Interferenzkorrektheit eine Konsequenz aus Hauptlemma 9.5.1.2 ist, wonach für unidirektionale Bitvektorprobleme der Effekt eines ganzen Pfads vom Effekt einer einzelnen Anweisung abhängt.

Das heißt, ausgedrückt in Lemma 9.5.2.4:

- 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 9.5.2.4 (Synchronisationskorrektheit: 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:

$$[\![end(G)]\!]^{SUPP}_{\mathcal{S}_{G^*}} = [\![G]\!]^{*\text{-}SUPP}_{\mathcal{S}_{G^*}}$$

$$\begin{cases} Cst_{\mathbf{falsch}} & \text{if } \exists\, 1 \leq i \leq k.\ [\![end(G_i)]\!]^{SUPP}_{\mathcal{S}_{G^*}} = Cst_{\mathbf{falsch}} \\ Id_{\mathbb{B}} & \text{if } \forall\, 1 \leq i \leq k.\ [\![end(G_i)]\!]^{SUPP}_{\mathcal{S}_{G^*}} = Id_{\mathbb{B}} \\ Cst_{\mathbf{wahr}} & \text{otherwise.} \end{cases}$$

2. Die *VUPP*-Semantik von G ist gegeben durch:

$$[\![end(G)]\!]^{VUPP}_{\mathcal{S}_{G^*}} = [\![G]\!]^{*\text{-}VUPP}_{\mathcal{S}_{G^*}}$$

$$\begin{cases} Cst_{\mathbf{wahr}} & \text{if } \exists\, 1 \leq i \leq k.\ [\![end(G_i)]\!]^{VUPP}_{\mathcal{S}_{G^*}} = Cst_{\mathbf{wahr}} \\ Id_{\mathbb{B}} & \text{if } \forall\, 1 \leq i \leq k.\ [\![end(G_i)]\!]^{VUPP}_{\mathcal{S}_{G^*}} = Id_{\mathbb{B}} \\ Cst_{\mathbf{falsch}} & \text{otherwise.} \end{cases}$$

Synchronisationskorrektheit (3)

...als induktive Erweiterung (der funktionalen Variante) des sequentiellen Koinzidenztheorems 7.7.2 für unidirektionale Bitvektorprobleme erhalten wir zusammen mit Lemma 9.5.2.4 die Korrektheit hierarchisch erfolgender Synchronisation(s-schritte):

Hierarchisches Koinzidenztheorem 9.5.2.5

Sei $G \in \mathcal{G}_{\mathcal{P}}(G^*)$ der Graph einer Parallelanweisung und sei $\mathcal{S}_{G^*} = (G^*, [\![\ ]\!])$ eine DFA-Spezifikation mit $[\![\ ]\!] : N^* \to \mathcal{F}_{\mathsf{IB}}$ lokales Semantikfunktional eines unidirektionalen Bitvektor-problems. Dann gilt:

1. $[\![\, end(G) \,]\!]^{SUPP}_{\mathcal{S}_{G^*}} = [\![\, G \,]\!]^{*\text{-}SUPP}_{\mathcal{S}_{G^*}}$
2. $[\![\, end(G) \,]\!]^{VUPP}_{\mathcal{S}_{G^*}} = [\![\, G \,]\!]^{*\text{-}VUPP}_{\mathcal{S}_{G^*}}$

Kapitel 9.5.3

Maximale parallele Fixpunktsemantik

Das $PMaxFP_{UBV}$-Gleichungssystem

...sei $\mathcal{S}_{G^*} =_{df} (\widehat{\mathcal{F}_{\mathbb{B}}}, [\![\ ]\!])$ die DFA-Spezifikation eines unidirektionaen Bitvektorproblems.

Gleichungssystem 9.5.3.1 ($PMaxFP_{UBV}$-GS)

$$[\![n]\!]^S_{\mathcal{S}_{G^*}} =$$

$$\begin{cases} Id_{\mathbb{B}} & \text{falls } n = \mathbf{s}^* \\ [\![pfg(n)]\!]^{*-SUPP}_{\mathcal{S}_{G^*}} \circ [\![start(pfg(n))]\!]^S_{\mathcal{S}_{G^*}} \sqcap Cst_{Interf\text{-}proof_n^{SUPP}} & \text{falls } n \in N^*_X \\ \bigsqcap \{ [\![m]\!] \circ [\![m]\!]^S_{\mathcal{S}_{G^*}} \mid m \in pred_{G^*}(n) \} \sqcap Cst_{Interf\text{-}proof_n^{SUPP}} & \text{sonst} \end{cases}$$

Beachte: Gleichungssystem 9.5.3.1 ist die natürliche Erweiterung von Gleichungssystems 9.2.1 auf parallele Programme.

Die $PMaxFP_{UBV}$-Semantik

Sei $[\![\ ]\!]^{SUPP}_{\mathcal{S}_{G^*}} : N^* \to \mathcal{F}_{\mathbb{B}}$ die eindeutig bestimmte größte Lösung von Gleichungssystem 9.5.3.1. Dann gilt:

Definition 9.5.3.2 ($PMaxFP_{UBV}$ -Semantik)

Die $PMaxFP_{UBV}$-Semantik für das von $\mathcal{S}_{G^*}$ spezifizierte unidirektionale Bitvektorproblem ist definiert durch:

$$[\![\ ]\!]^{PMaxFP_{UBV}}_{\mathcal{S}_{G^*}} : N^* \to \mathcal{F}_{\mathbb{B}}$$

$$\forall\, n \in N^*.\ [\![\, n \,]\!]^{PMaxFP_{UBV}}_{\mathcal{S}_{G^*}} = [\![\, n \,]\!]^{SUPP}_{\mathcal{S}_{G^*}}$$

Kapitel 9.5.4

Minimale parallele Fixpunktsemantik

Das $PMinFP_{UBV}$-Gleichungssystem

...sei $\mathcal{S}_{G^*} =_{df} (\widehat{\mathcal{F}_{\mathbb{B}}}, [\![\ ]\!])$ die DFA-Spezifikation eines unidirektionalen Bitvektorproblems.

Gleichungssystem 9.5.4.1 ($PMinFP_{UBV}$-GS)

$$[\![n]\!]^V_{\mathcal{S}_{G^*}} =$$

$$\begin{cases} Id_{\mathbb{B}} & \text{falls } n = \mathbf{s}^* \\ [\![pfg(n)]\!]^{*\text{-}VUPP}_{\mathcal{S}_{G^*}} \circ [\![start(pfg(n))]\!]^V_{\mathcal{S}_{G^*}} \sqcup Cst_{Interf\text{-}proof_n^{VUPP}} & \\ & \text{falls } n \in N^*_X \\ \bigsqcup\{ [\![m]\!] \circ [\![m]\!]^S_{\mathcal{S}_{G^*}} \mid m \in pred_{G^*}(n)\} \sqcup Cst_{Interf\text{-}proof_n^{VUPP}} & \text{sonst} \end{cases}$$

Die $PMinFP_{UBV}$-Semantik

Sei $[\![\ ]\!]^{VUPP}_{\mathcal{S}_{G^*}} : N^* \to \mathcal{F}_{\mathbb{B}}$ die eindeutig bestimmte kleinste Lösung von Gleichungssystem 9.5.4.1. Dann gilt:

Definition 9.5.2.2 ($PMinFP_{UBV}$ -Semantik)

Die $PMinFP_{UBV}$-Semantik für das von $\mathcal{S}_{G^*}$ spezifizierte unidirektionale Bitvektorproblem ist definiert durch:

$$[\![\ ]\!]^{PMinFP_{UBV}}_{\mathcal{S}_{G^*}} : N^* \to \mathcal{F}_{\mathbb{B}}$$

$$\forall\, n \in N^*.\ [\![\, n \,]\!]^{PMinFP_{UBV}}_{\mathcal{S}_{G^*}} \ = \ [\![\, n \,]\!]^{VUPP}_{\mathcal{S}_{G^*}}$$

Kapitel 9.6

Unidirektionale Bitvektoranalysen: Koinzidenz

Hauptergebnis: Korrektheit und Vollständigkeit

...für unidirektionale Bitvektorprobleme stimmen die

- parallelen operationellen Schnitt- und Vereinigung-über-alle-Pfade-Semantiken

mit ihren

- parallelen denotationellen Fixpunktgegenstücken

überein: Paralleles Bitvektorkoinzidenztheorem 9.6.1.

Paralleles Bitvektorkoinzidenztheorem

...sei $G^* = (N^*, E^*, \mathbf{s}^*, \mathbf{e}^*)$ ein paralleles Programm und $\mathcal{S}_{G^*} =_{df} (\widehat{\mathcal{F}_{\mathbb{B}}}, [\![\ ]\!])$ die DFA-Spezifikation eines unidirektionalen Bitvektorproblems.

Paralleles Bitvektorkoinzidenztheorem 9.6.1

1. Die *SUPP*-Semantik (engl. *MOPP* semantics) und die $PMaxFP_{UBV}$-Semantik stimmen überein, d.h.:

$$\forall\, n \in N^*.\ [\![\, n \,]\!]^{SUPP}_{\mathcal{S}_{G^*}} = [\![\, n \,]\!]^{PMaxFP_{UBV}}_{\mathcal{S}_{G^*}}$$

2. Die *VUPP*-Semantik (engl. *JOPP* semantics) und die $PMinFP_{UBV}$-Semantik stimmen überein, d.h.:

$$\forall\, n \in N^*.\ [\![\, n \,]\!]^{VUPP}_{\mathcal{S}_{G^*}} = [\![\, n \,]\!]^{PMinFP_{UBV}}_{\mathcal{S}_{G^*}}$$

Kapitel 9.7

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. 7.10.1):

$$\forall\, n \in N^*.\ [\![\, n \,]\!]^t_{av} \quad =_{df} \quad \begin{cases} Const_{\mathbf{wahr}} & \text{falls } Transp^t_n \wedge Comp^t_n \\ Id_{\mathbb{B}} & \text{falls } Transp^t_n \wedge \neg Comp^t_n \\ Const_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

Tote Zuweisungen an Variable x (vgl. Kap. 12.3):

$$\forall\, n \in N^*.\ [\![\, n \,]\!]^x_{dead} \quad =_{df} \quad \begin{cases} Const_{\mathbf{wahr}} & \text{falls } \neg Used^x_n \wedge Mod^x_n \\ Id_{\mathbb{B}} & \text{falls } \neg(Used^x_n \vee Mod^x_n \\ Const_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

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

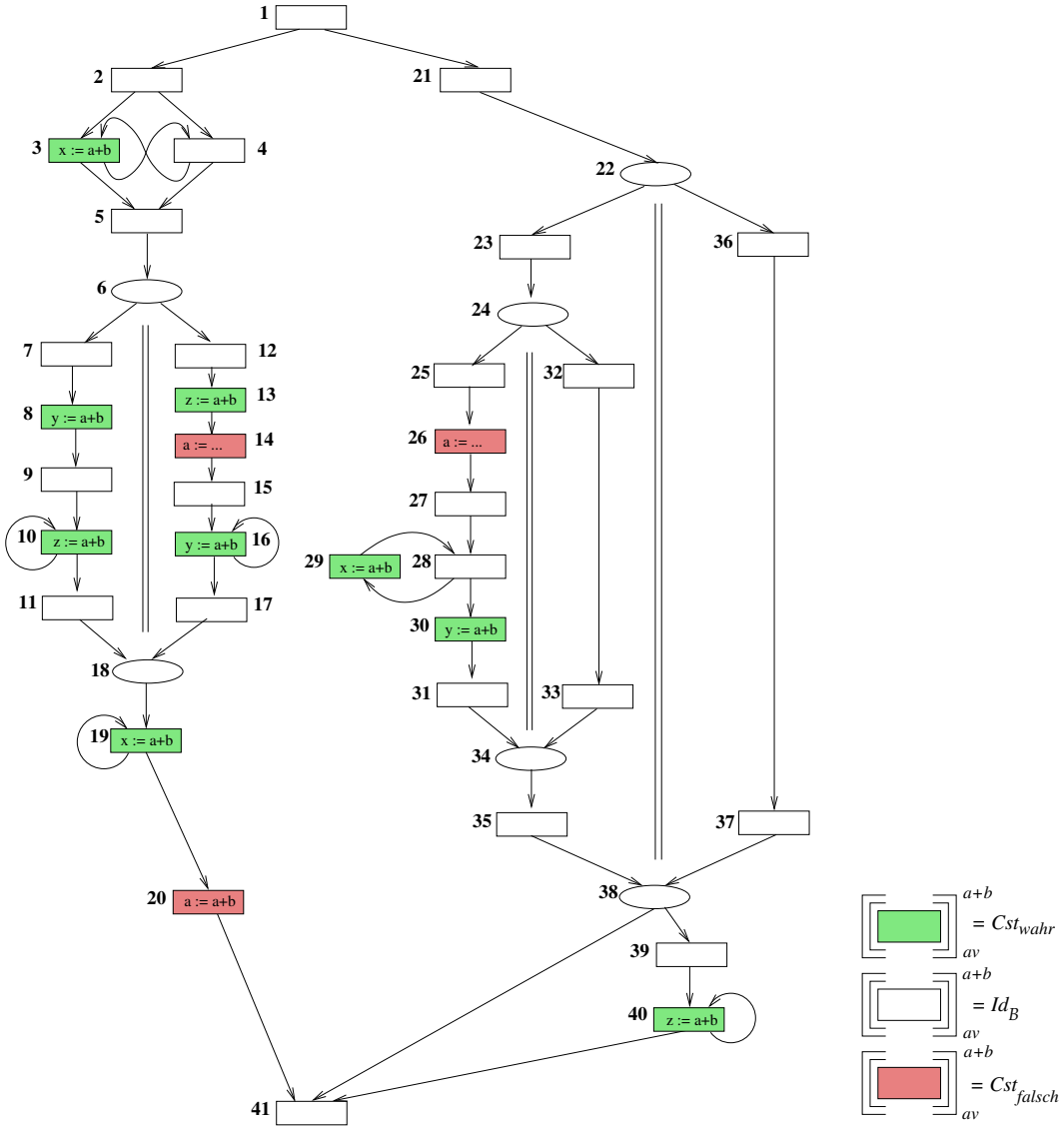
- *SUPP* - bzw. VUPP-Semantik

globalisiert werden, was zur Berechnung folgender Eigenschaften führt:

- *SUPP* -Globalisierung: (Total) verfügbare Ausdrücke bzw. (total) tote Zuweisungen.
- *VUPP* -Globalisierung: Partiell verfügbare Ausdrücke bzw. partiell tote Zuweisungen.

Lokale Semantik: Verfügbarkeitsanalyse

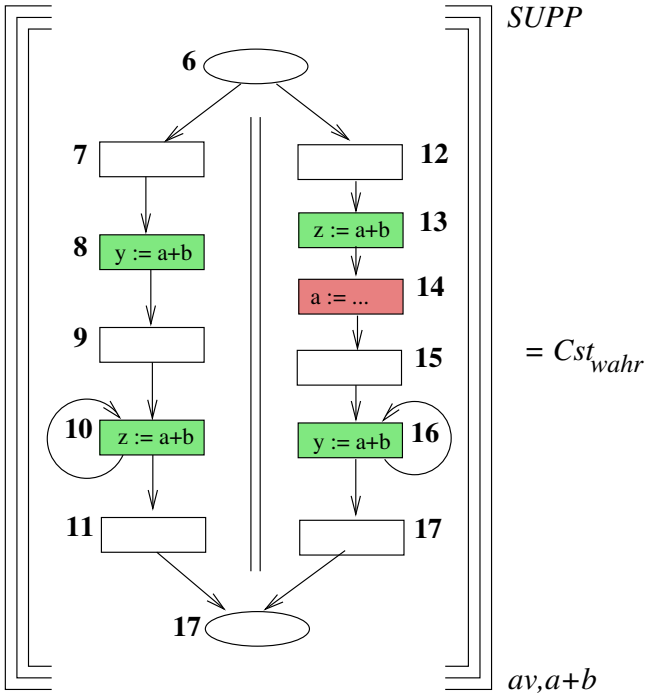
...für Ausdruck a+b:



SUPP -Semantikglobalisierung: 1. Iteration

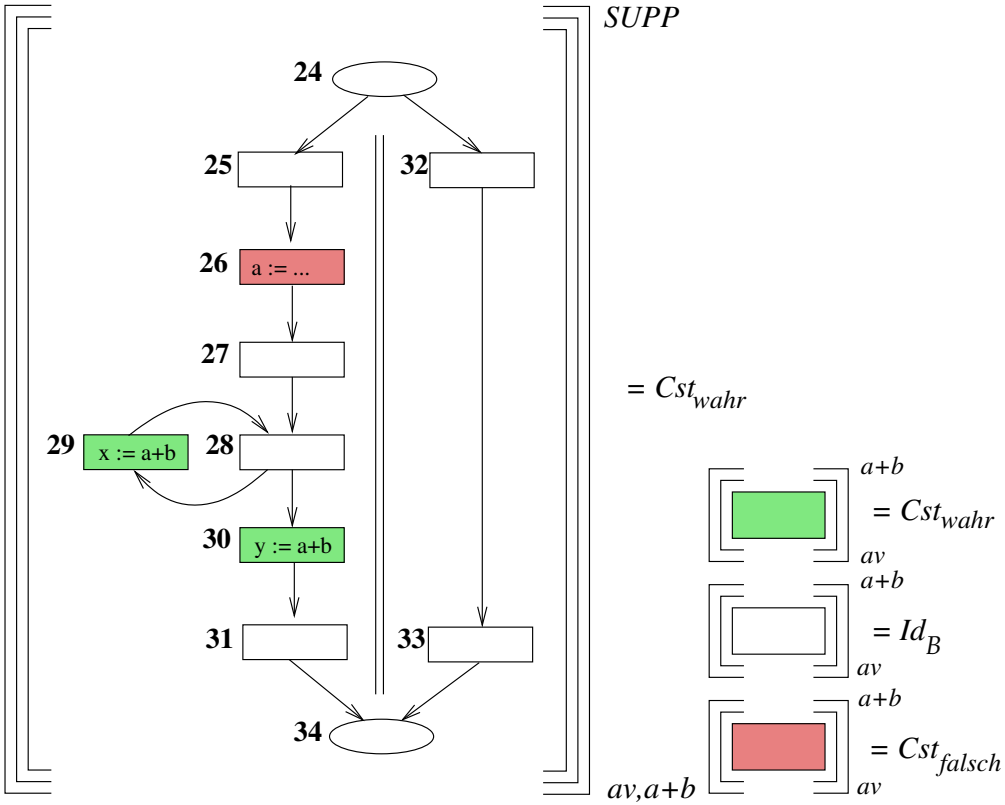
...entsprechend des 4-stufigen Verfahrens aus Kapitel 9.5.2:

$G_{01} = G_{01_{seq}}$



destructive(G_{01}) = wahr

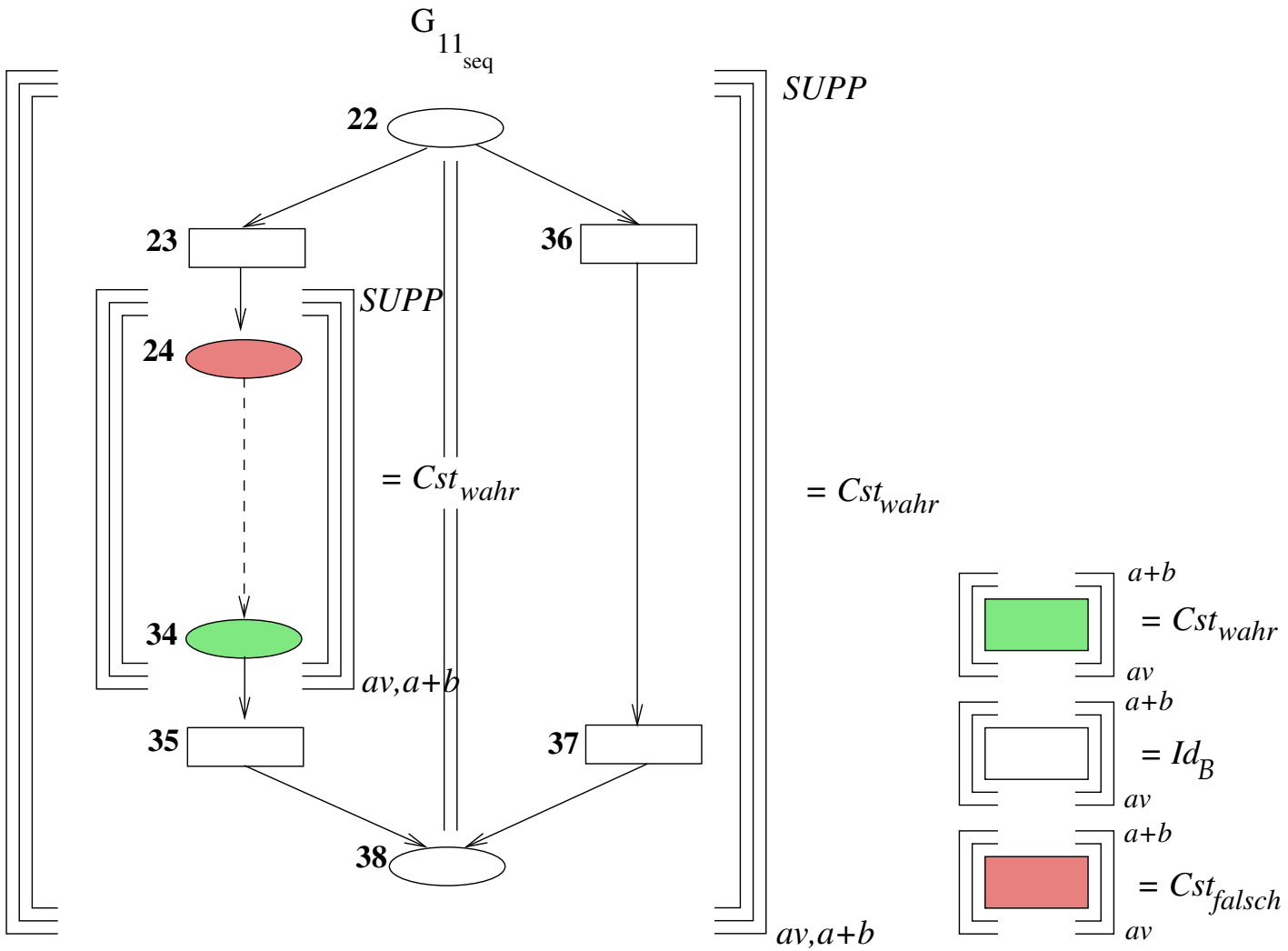
$G_{02} = G_{02_{seq}}$



destructive(G_{02}) = wahr

SUPP -Semantikglobalisierung: 2. Iteration

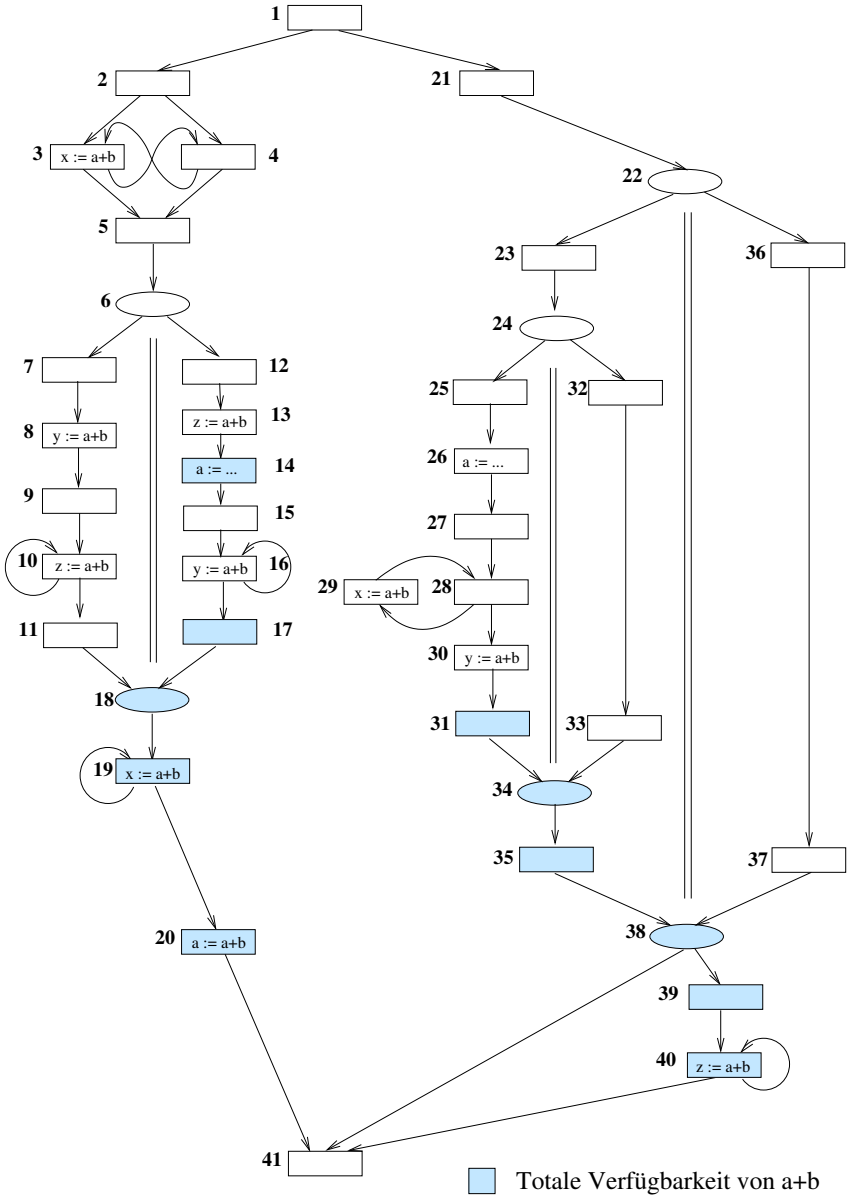
...entsprechend des 4-stufigen Verfahrens aus Kapitel 9.5.2:



destructive(G $_{11_{seq}}$) = wahr

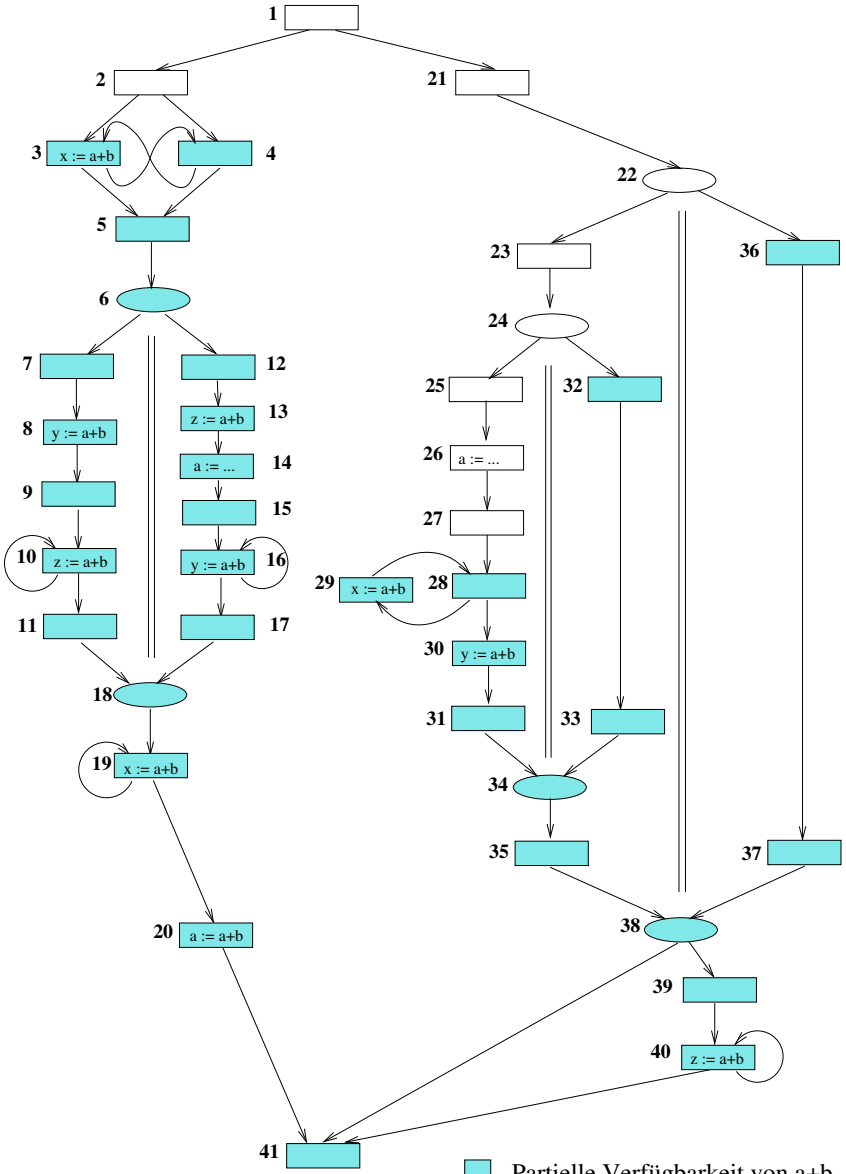
Ergebnis der totalen Verfügbarkeitsanalyse

...für a+b an Knoteneingängen:



Ergebnis der partiellen Verfügbarkeitsanalyse

...für a+b an Knoteneingängen:



Übungsaufgabe

Partielle Verfügbarkeitsanalyse für a+b:

- Gib die *VUPP*-Semantikglobalisierung für das durchgehende Beispiel nach dem 1. und 2. Iterationsschritt des 4-stufigen Verfahrens aus Kapitel 9.5.2 an.

Tote-Zuweisungenanalyse für a:

- Gib die *SUPP*- und *VUPP*-Semantikglobalisierungen für das durchgehende Beispiel nach dem 1. und 2. Iterationsschritt des 4-stufigen Verfahrens aus Kapitel 9.5.2 an.
- 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 9.8

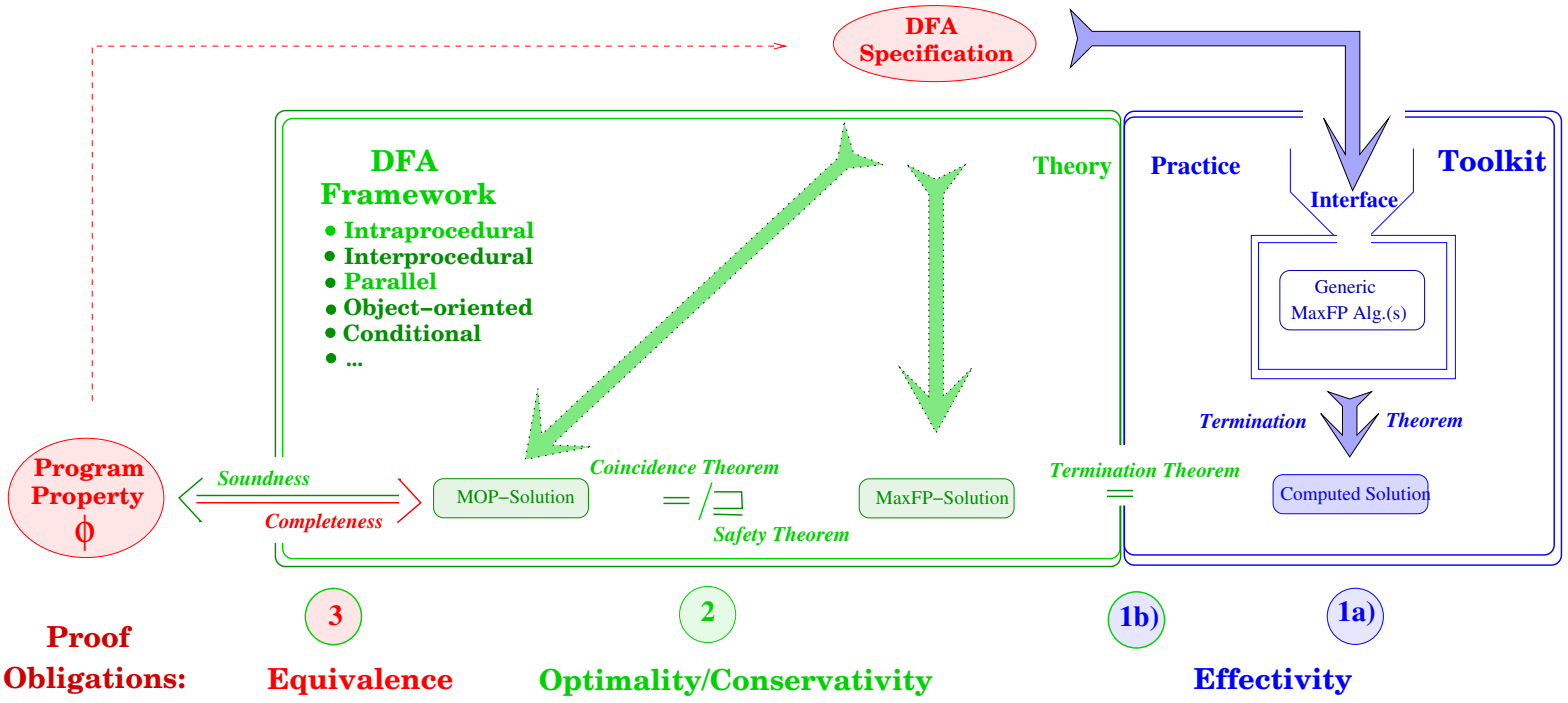
Zusammenfassung

Zusammenfassung

...mit dem Parallelen Koinzidenztheorem 9.6.1 für unidirektionale Bitvektorverfahren ist das Versprechen aus Kapitel 7.11 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 die Existenz lediglich von 3 Funktionen in $\mathcal{F}_{\mathbb{IB}}$, deren äußere Semantik oft wie folgt beschrieben werden kann:

- Mod: Modifikation
- Use: Benutzung (engl. Use)
- $Transp$: Transparenz

Deshalb: Nur MUT! Nur MUT zur DFA paralleler Programme!

Kapitel 9.9

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 9 (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 9 (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 9 (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 9 (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 9 (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 9 (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.


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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 9 (7)

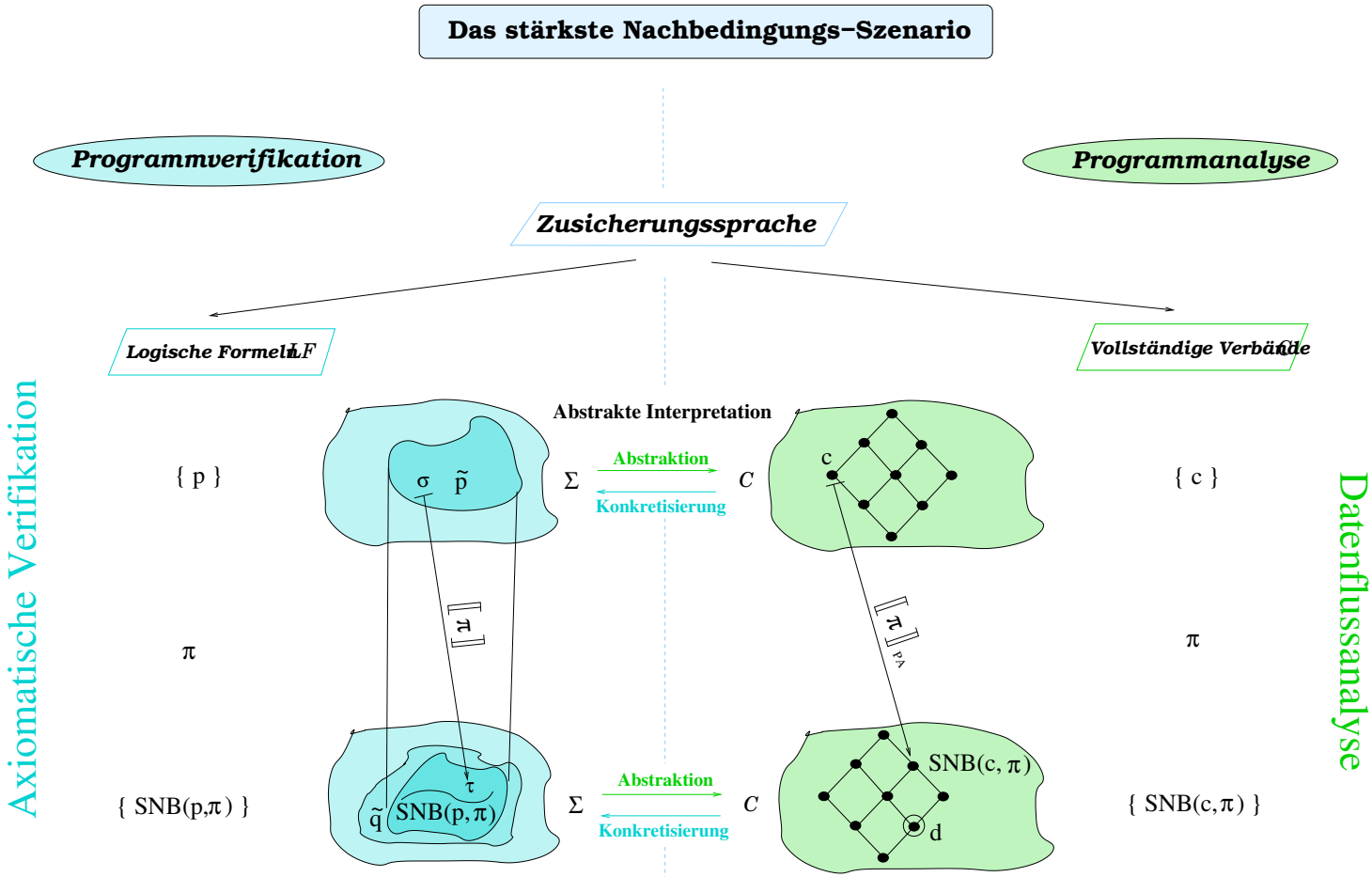
Jürgen Vollmer. *Data Flow Analysis of Parallel Programs*. In Proceedings of the IFIP WG 10.3 Working Conference on Parallel Architectures and Compilation Techniques (PACT'95), 168-177, 1995.

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 10

Programmverifikation vs. Programmanalyse: Axiomatische Verifikation, Datenfluss-analyse im Vergleich

Stärkste Nachbedingungssichten im Vergleich



SNB(p, π) ∈ *LF* *muss erfüllen:*

(1) $\models_{PV} \{ p \}\, \pi\, \{ SNB(p,\pi) \}$

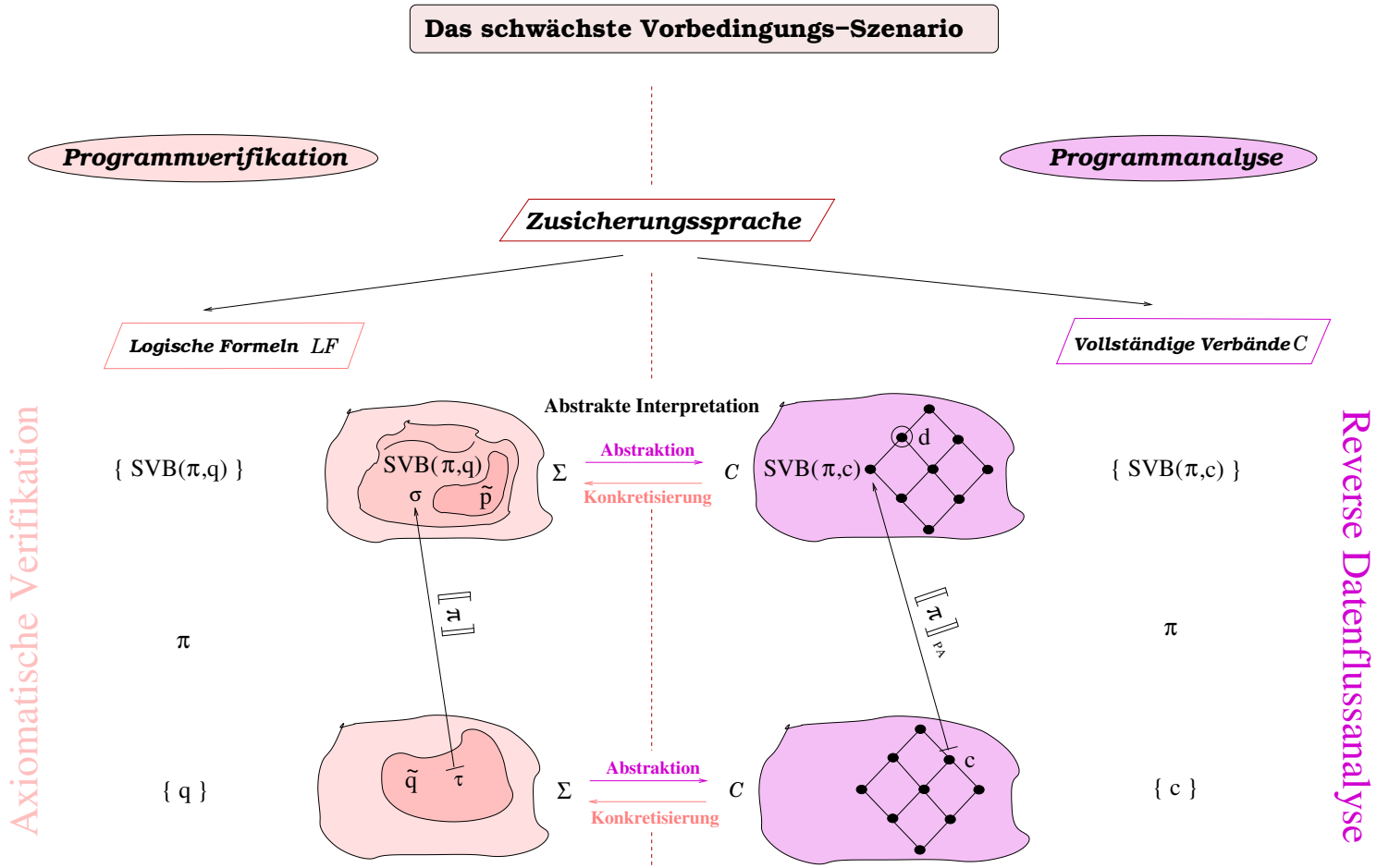
(2) $\forall\, q \in LF.\ \models_{PV} \{ p \}\, \pi\, \{ q \}$ impliziert $SNB(p,\pi) \Longrightarrow q$

SNB(c, π) ∈ *C* *muss erfüllen:*

(1) $\models_{PA} \{ c \}\, \pi\, \{ SNB(c,\pi) \}$

(2) $\forall\, d \in C.\ \models_{PA} \{ c \}\, \pi\, \{ d \}$ impliziert $SNB(c,\pi) \sqsupseteq d$

Schwächste Vorbedingungssichten im Vergleich



SVB(π,q) ∈ *LF* *muss erfüllen:*

(1) $\models_{PV} \{ SVB(\pi,q) \}\, \pi\, \{ q \}$

(2) $\forall\, p \in LF.\ \models_{PV} \{ p \}\pi \{ q \}$ impliziert $p \Longrightarrow SVB(\pi,q)$

SVB(π,c) ∈ *C* *muss erfüllen:*

(1) $\models_{PA} \{ SVB(\pi,c) \}\, \pi\, \{ c \}$

(2) $\forall\, d \in C.\ \models_{PA} \{ d \}\pi \{ c \}$ impliziert $SVB(\pi,c) \sqsubseteq d$

Teil IV

Fixpunkte, Transformation und Optimalität

Informell

$$\text{Optimalität} = \text{Korrektheit} + \text{Vollständigkeit}$$

Kapitel 11

Chaotische Fixpunktiteration

Kapitel 11.1

Motivation

Motivation

...viele praktisch relevante Probleme in der Informatik lassen sich durch die

- kleinste (größte) Lösung

eines Systems rekursiver Gleichungen beschreiben.

Einige Beispiele aus dieser Vorlesung:

- Denotationelle Semantik von WHILE
- Maximale/minimale DFA-Fixpunktsemantik
- Minimale/maximale RDFA-Fixpunktsemantik

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

...Gegenüberstellung des Lösens von Gleichungssystemen und des Berechnens von Fixpunkten von (Familien von) Funktionen:

- 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 $1 \leq k \leq n$

entsprechen einander.

Lösungs- und Fixpunktberechnung

...mittels iterativer Algorithmen und chaotischer Iteration.

Iterative Algorithmen zur Lösungs- bzw. Fixpunktberechnung

- beginnen üblicherweise 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

...möglicherweise das 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, dennoch oft noch zu speziell.

Verallgemeinerungen, Variationen des Tarskischen Iterationsschemas:

- Vektor-Iterationen: Robert [1976]
- Asynchrone 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 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, Liz 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 11.3, 12 und 13.

Weitere Fixpunktresultate:

- Siehe Anhang A5.

Kapitel 11.2

Chaotisches Fixpunktiterationstheorem

Vorbereitungen (1)

Definition 11.2.1 (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 11.2.2 (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 11.2.3 (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 11.2.4 (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 11.2.5 (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 11.2.6 (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 11.2.7

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 11.2.8

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 11.2.9 (Chaotisches Fixpunkiterationsth.)

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 11.2.10

$d := \bot$;
while $\exists\, k \in \mathbb{N}.\ d \neq f_k(d)$ **do**
  choose $k \in \mathbb{N}$ **with** $d \sqsubset f_k(d)$
    $d := f_k(d)$
od

Anmerkung: Wegen f_k, $k \in \mathbb{N}$, inflationär, folgt aus $d \neq f_k(d)$ unmittelbar $d \sqsubset f_k(d)$.

Kapitel 11.3

Anwendungen

Kapitel 11.3.1

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 11.3.1.1 (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 11.2.4

...zur Modellierung der Vektor-Iteration.

Erforderlich: Verallgemeinerung des Strategiebegriffs auf einen Strategiebegriff für Mengen.

Definition 11.3.1.2 (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 11.3.1.3 (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 11.3.1.1 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 11.3.1.4 (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 11.3.1.1 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 11.3.1.4

- ist ein Spezialfall des Chaotischen Fixpunktiterationstheorems 11.2.9 für Vektor-Iterationen und folgt zusammen mit Lemma 11.3.1.3.

Für $|\mathcal{F}| = 1$ reduziert sich

- die Aussage von Theorem 11.3.1.4 auf Tarskis Fixpunkttheorem für den Fall wohlfundierter partieller Ordnungen.

Kapitel 11.3.2

Datenflussanalyse

Anwendung von Fixpunkttheorem 11.2.4

...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 die 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:

$$inf(n) = \begin{cases} c_{\mathbf{s}} & \text{falls } n = \mathbf{s} \\ \bigsqcup \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{sonst} \end{cases}$$

Die *MinFP*-Semantik:

...definiert als die kleinste Lösung des *MinFP*-Gleichungssystems, bezeichnet mit:

$$\mu\text{-}inf^*_{c_{\mathbf{s}}} : N \to \mathcal{C}$$

Generischer *MinFP*-Fixpunktalgorithmus

Generischer *MinFP*-Fixpunktalgorithmus 11.3.2.1

$inf[\mathbf{s}] := c_{\mathbf{s}}$;
forall $n \in N \backslash \{\mathbf{s}\}$ **do** $inf[n] := \bot$ **od**;
$workset := N$;
while $workset \neq \emptyset$ **do**
  choose $k \in workset$
    $workset := workset \backslash \{ k \}$;
    $new := inf[k] \sqcup \bigsqcup \{ [\![(m,k)]\!](inf[m]) \mid m \in pred_G(k) \}$;
    if $new \sqsupset inf[k]$ **then**
      $inf[k] := new$;
      $workset := workset \cup succ_G(k)$
    fi
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

$$d'^k = d^k \sqcup \bigsqcup \{ [\![(m,k)]\!](d^m) \mid m \in pred_G(k) \}$$

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 11.4.2.1:

$$new := inf[k] \sqcup \bigsqcup \{ [\![(m,k)]\!](inf[m]) \mid m \in pred_G(k) \}$$

entspricht:

$$d'^k = d^k \sqcup \bigsqcup \{ [\![(m,k)]\!](d^m) \mid m \in pred_G(k) \}$$

Charakterisierungsresultate

Lemma 11.3.2.2 (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 11.3.2.3 (Korrektheit und Terminierung)

Jeder Lauf des generischen *MinFP*-Fixpunktalgorithmus 11.3.2.1 terminiert mit der *MinFP*-Semantik von G für die lokale DFA-Semantik $[\![\ ]\!]$ und die Startzusicherung c_s.

Anmerkungen

...zum Beweis von Theorem 11.3.2.3:

- Der *MinFP*-Fixpunktalgorithmus 11.3.2.1 folgt dem Muster von Rumpfalgorithmus 11.2.10 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 11.3.1.4 erfüllt, womit Theorem 11.3.2.4 folgt.

Ausblick

...weitere Anwendungen des ‘monotoniefreien’ chaotischen Fixpunktiterationstheorems 11.2.9 in Kapitel 12 und 13 zum Beweis der Optimalität der Transformation für die

- Elimination partiell toten Codes
- Elimination partiell redundanter Anweisungen

Klassische Fixpunkttheoreme sind dafür nicht anwendbar.

Kapitel 11.4

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 11 (1)

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, Liz Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 11 (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)

Kapitel 12

Unnötige Anweisungen

Kapitel 12.1

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 12.2)
- Partiell tote/geisterhafte Anweisungen (Kapitel 12.3)
- Partiell redundante Anweisungen (Kapitel 12.4)
- ...

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 12 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 12.2

Unerreichbare Anweisungen

Zusatzvereinbarungen für Flussgraphen

...für Kapitel 12.2.

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 12.2.1

Statisch unerreichbare Anweisungen

Statisch unerreichbare Knoten und Kanten

...sei $G = (N, E, \mathbf{s})$ ein (knotenbenannter) Flussgraph.

Definition 12.2.1.1 (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 12.2.1.2 (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 12.2.1.3 (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 12.2.1.4 (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 12.2.1.5 (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 12.2.1.6 (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 12.2.1.7 (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 12.2.1.6.
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 12.2.1.8 ($\mathcal{K}$-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 12.2.1.9 (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 12.2.1.10 (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 12.2.1.11 ($\mathcal{K}$-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 12.2.1.12 ($\mathcal{K}$-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 12.2.1.13 (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 12.2.1.14 (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 12.2.1.15 ($\mathcal{K}$-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 12.2.1.16 ($\mathcal{K}$-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:

- Statische Erreichbarkeitsanalysen induzieren Eliminationstransformationen.
- Korrekte und vollständige Erreichbarkeitsanalysen induzieren $\mathcal{K}$-optimale Eliminationstransformationen.

Charakterisierung stat. unerreichbarer Knoten

Proposition 12.2.1.17 (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 12.2.1.18 (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 12.2.1.19 (Speziell)

s ist statisch erreichbar.

Charakterisierung stat. unerreichbarer Kanten

Proposition 12.2.1.20 (Äquivalenz)

Sei $e \in E$ eine Kante. Dann sind äquivalent:

1. e ist statisch unerreichbar.
2. Der Anfangsknoten von e ist statisch unerreichbar.

Proposition 12.2.1.21 (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 12.2.1.22 (Speziell)

Gilt $src(e) = \mathbf{s}$, so ist e statisch erreichbar.

Charakterisierung von Unnötigkeit

...durch statische Unerreichbarkeit.

Lemma 12.2.1.23 (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 12.2.1.24 (Unnötige Knoten, Kanten)

1. Ein statisch unerreichbarer Knoten ist unnötig.
2. Eine statisch unerreichbare Kante ist unnötig.

Korrekte, vollständige Erreichbarkeitsanalysen

Definition 12.2.1.25 (Erreichbarkeitsanalyse)

Eine Analyse, die angewendet auf ein Programm $G = (N, E, \mathbf{s})$ einige Knoten und Kanten als erreichbar markiert, heißt Erreichbarkeitsanalyse.

Definition 12.2.1.26 (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 12.2.1.27 (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 12.2.1.28 (Optimale Erreichb.-Analyse)

Eine Erreichbarkeitsanalyse *EA* heißt optimal gdw *EA* korrekt und vollständig ist.

Korollar 12.2.1.29 (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 12.2.1.30 (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 12.2.1.31 (Optimalität von O_{EA})

O_{EA} ist optimal, d.h. O_{EA} ist korrekt und vollständig.

Optimierungstransformation OT

...zur Elimination unnötiger Anweisungen.

Definition 12.2.1.32 (Optimierung)

Das Optimierungsverfahren zur Elimination $\mathcal{K}$-unnötiger Anweisungen besteht aus zwei Stufen:

- Analysestufe: Erreichbarkeitsanalyse in Graphen mittels einer Erreichbarkeitsanalyse O_{EA}, O_{EA} optimal.
- 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 mit

$$OT =_{df} ET_{O_{EA}}$$

bezeichnet.

Korrektheit, Vollständigkeit, Optimalität

...von OT zur Elimination $\mathcal{K}$-unnötiger Anweisungen.

Lemma 12.2.1.33 (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 12.2.1.34 (Optimalität)

OT ist $\mathcal{K}$-optimal ($OT \in \mathcal{ET}_{\mathcal{K}\text{-}opt}$ $(= \mathcal{ET}_{best})$).

Übungsaufgabe

...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.

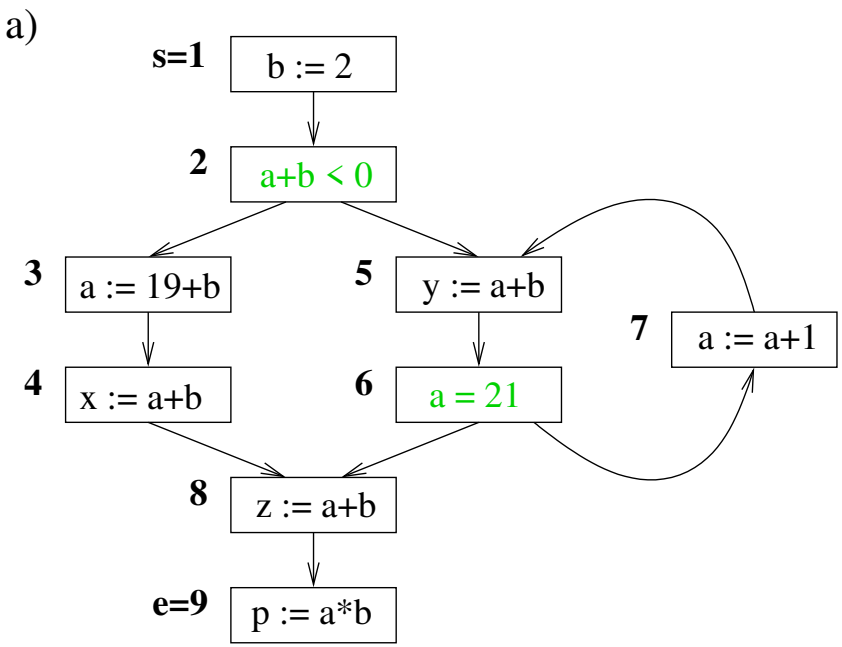
Kapitel 12.2.2

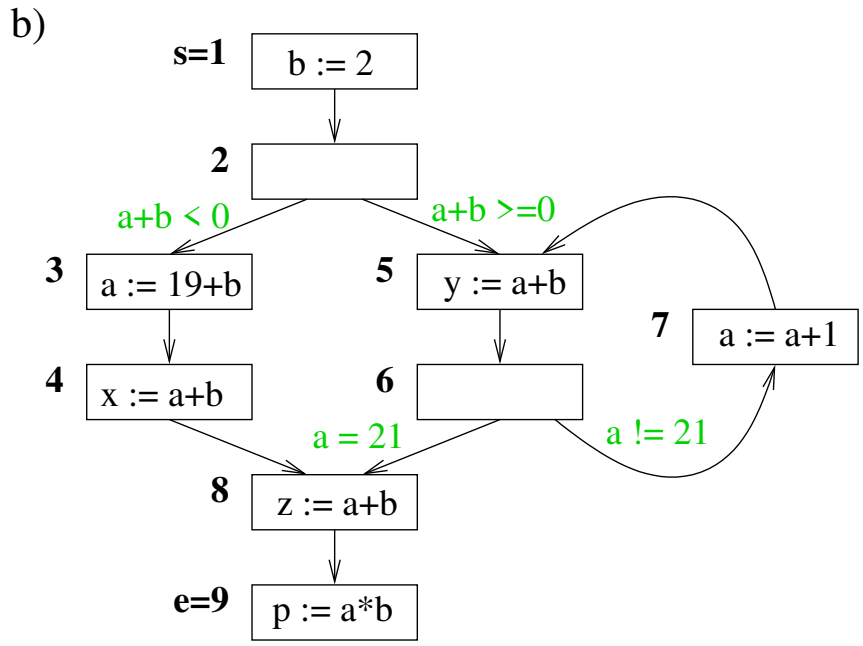
Dynamisch unerreichbare Anweisungen

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.:

$$\forall \sigma \in \Sigma.\ [\![b]\!]_B([\![src(e)]\!]_{\text{WHILE}}(\sigma)) = \mathbf{falsch}$$





Dynamisch unerreichbare Anweisungen

...eine 'semantische' Eigenschaft.

Definition 12.2.2.1 (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 12.2.2.2 (Unentscheidbarkeit)

Dynamische Unerreichbarkeit von Anweisungen ist unentscheidbar.

Entscheidbare Teilklassen dyn. Unerreichbark.

...Lemma 12.2.2.2 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
- ...

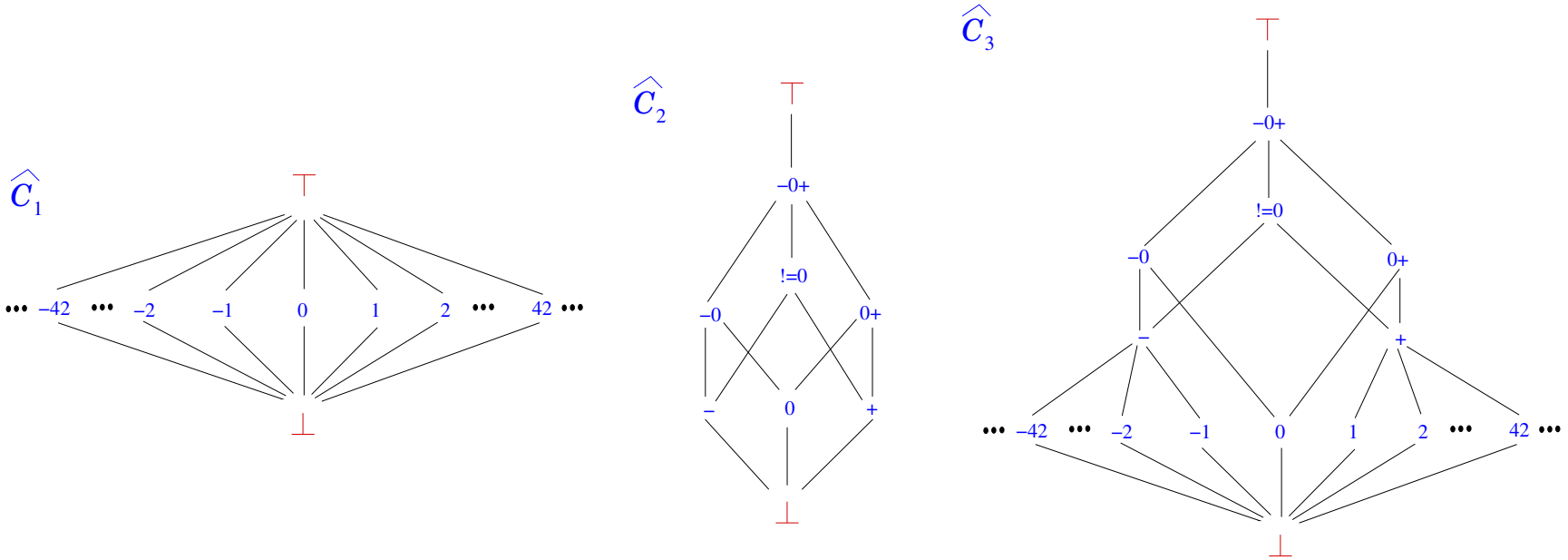
vom Wert **wahr** ist.

Übungsaufgabe

Führe die Idee der $\mathcal{K}$-Unerreichbarkeit von Anweisungen nach dem Vorbild statisch unerreichbarer Anweisungen aus Kapitel 12.2.1 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 12.2.3

Senken, Sackgassen und schwarze Löcher

Senken, Sackgassen, schwarze Löcher (1)

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Programm.

Definition 12.2.3.1 (Statisch liegen in)

Ein Knoten $n \in N$ liegt statisch in

- einer Senke gdw $\mathbf{e}$ ist auf keinem Pfad von n aus erreichbar, d.h. wenn: $\mathbf{P}_G[n, \mathbf{e}] = \emptyset$.
- einem schwarzen Loch gdw n liegt statisch in einer Senke und es gilt: $\bigcup_{m \in N} \{\mathbf{P}_G[n, m]\}$ ist unendlich.
- 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 12.2.3.2 (Statische Senken, etc.)

- Eine statische Senke ist eine Menge von Knoten, die statisch in einer Senke liegen.
- Ein statisches schwarzes Loch ist eine Menge von Knoten, die statisch in einem schwarzen Loch liegen.
- 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 12.2.3.3

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 12.2.3.4

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 12.2.3.5

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 12.2.3.6

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 12.2.1.
- der Knoten statischer Senken: Anwendung des Konzepts statischer Unerreichbarkeit aus Kapitel 12.2.1 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

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.

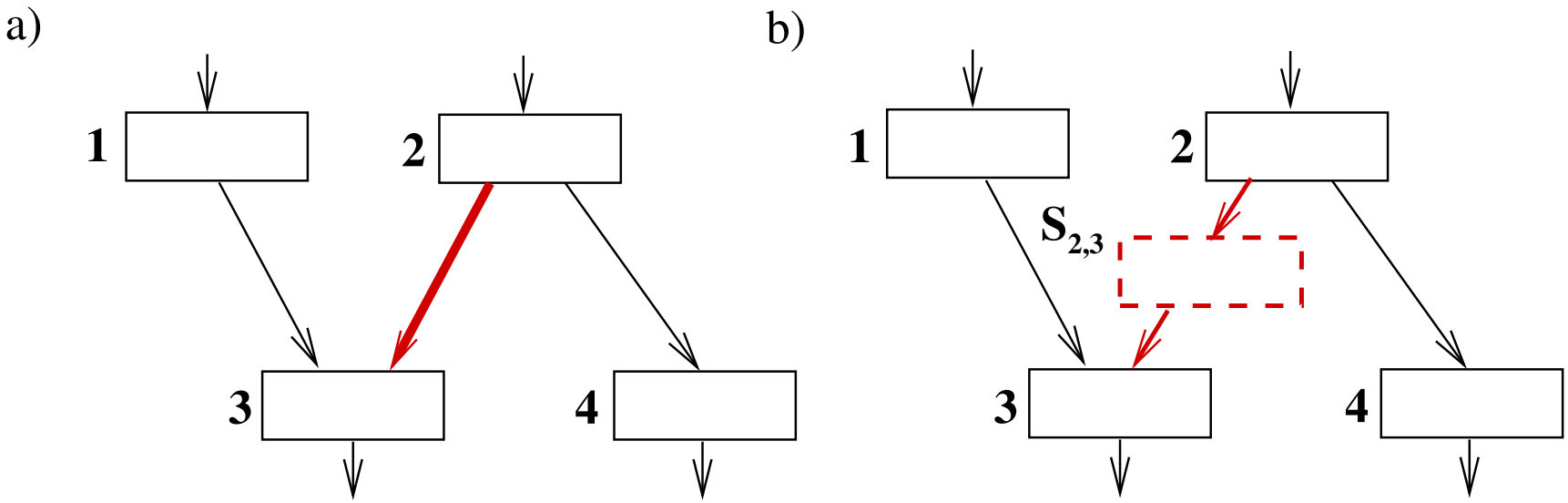
Kapitel 12.3

Partiell tote und geisterhafte Anweisungen

Kritische Kanten

Definition 12.3.1 (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.

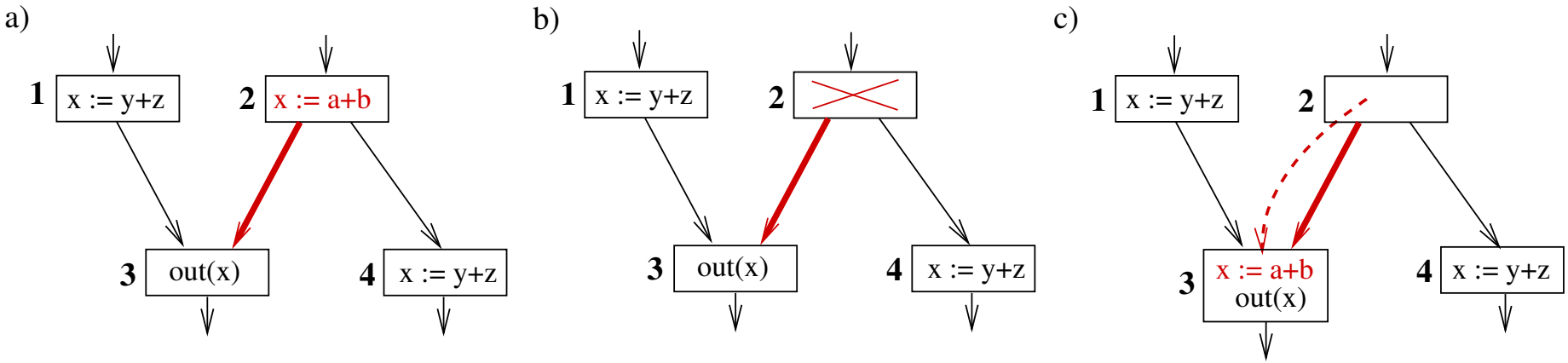
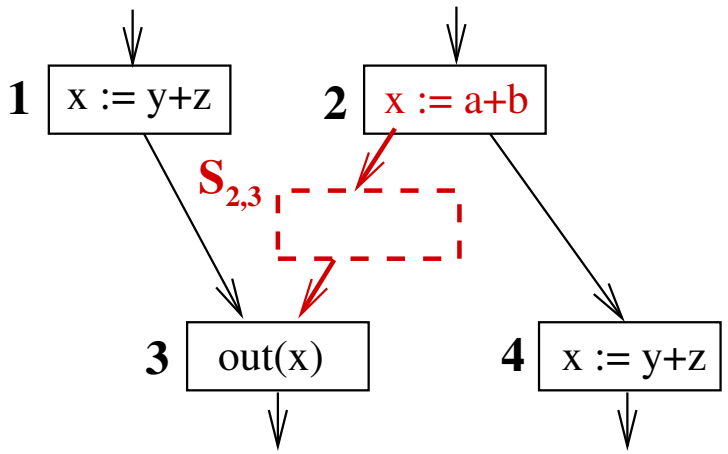


Illustration: Kritische Kanten (2)

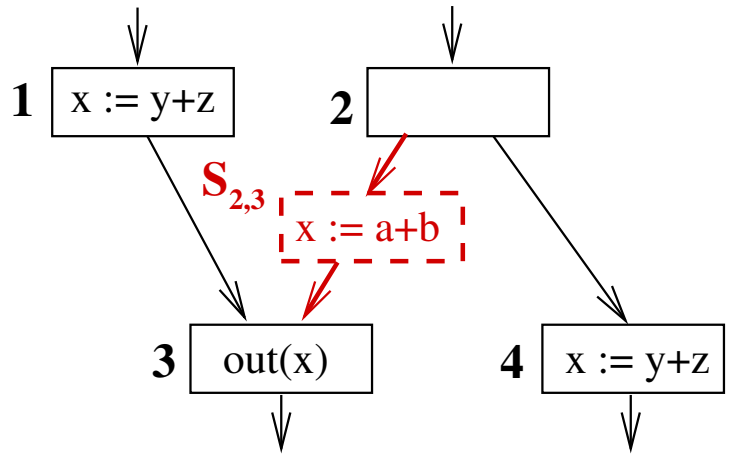
Spalten kritischer Kanten:

- 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)

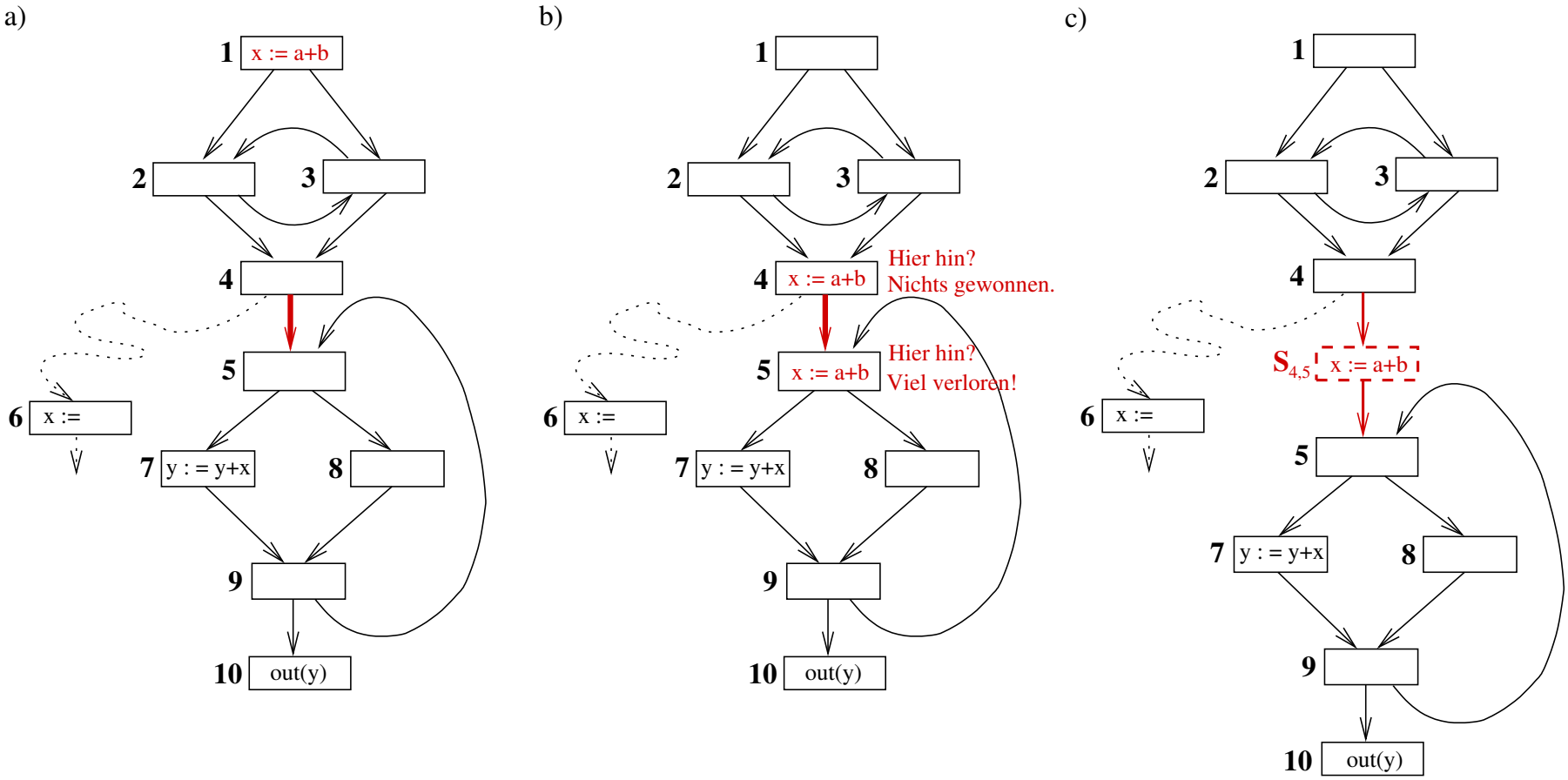


b)



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 12.3 (und 12.4).

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 $\mathbf{s}$ nach $\mathbf{e}$.
  - ist mit einer Instruktion (Zuweisung, Schreibanweisung, Bedingung) benannt (keine Basisblöcke).
- Keine Kante ist kritisch.

$\mathbf{s}$ und $\mathbf{e}$ bezeichnen als Start- und Endknoten ausgezeichnete Knoten in G.

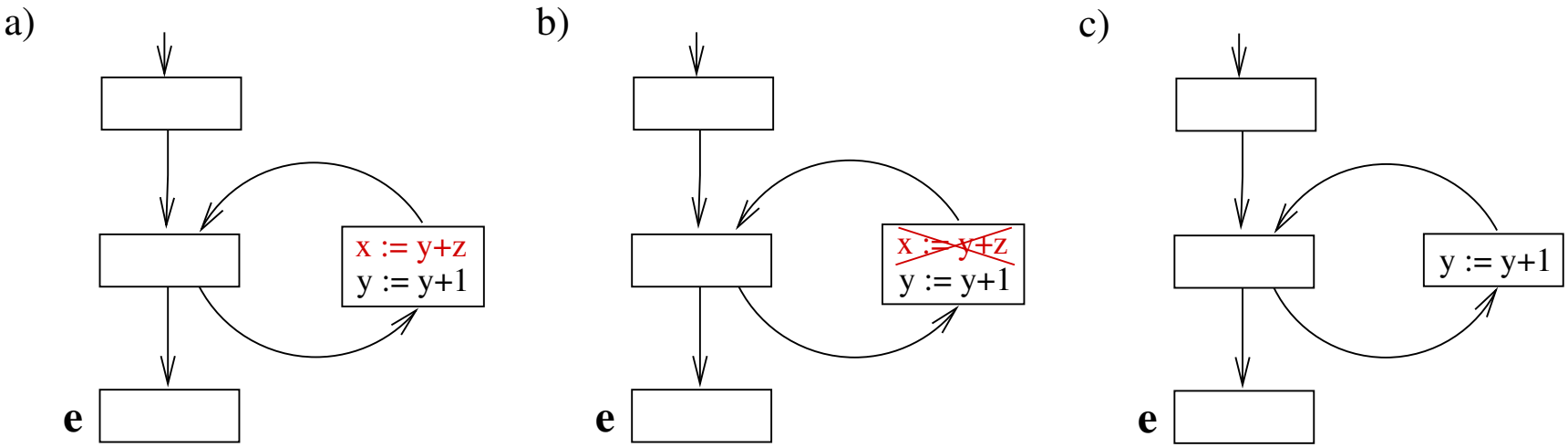
Kapitel 12.3.1

Motivation

Tote Anweisungen

...und ihre Elimination.

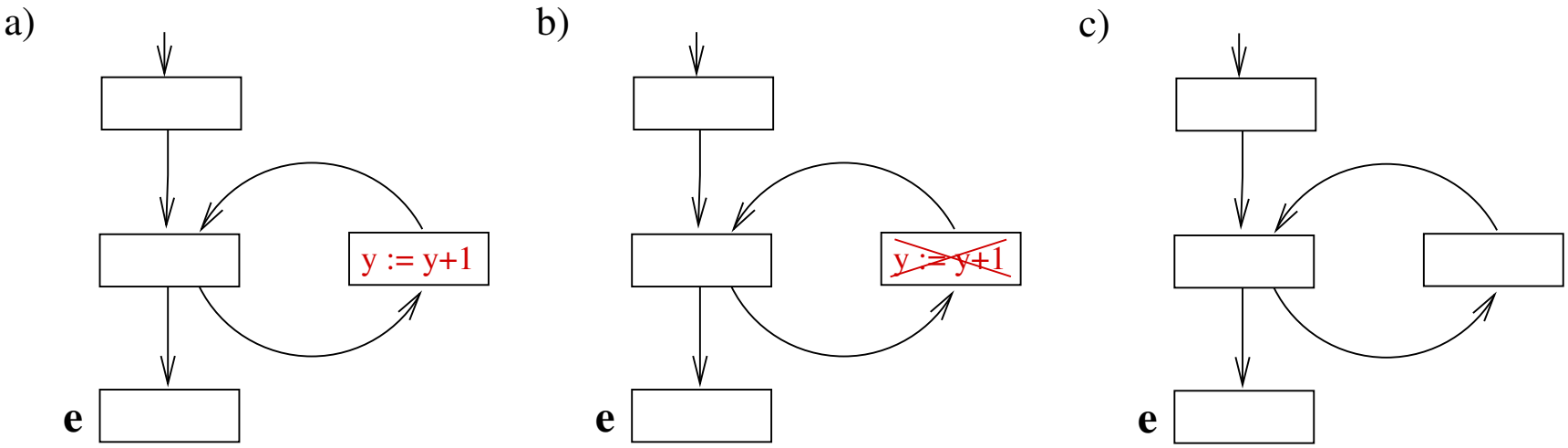
Das Grundmuster: Die Anweisung `x := y+z` ist tot (oder total tot) (engl. (totally) dead): `x` zugewiesene Werte werden an keiner Stelle im Programm gelesen.



Geisterhafte Anweisungen

.....und ihre Elimination.

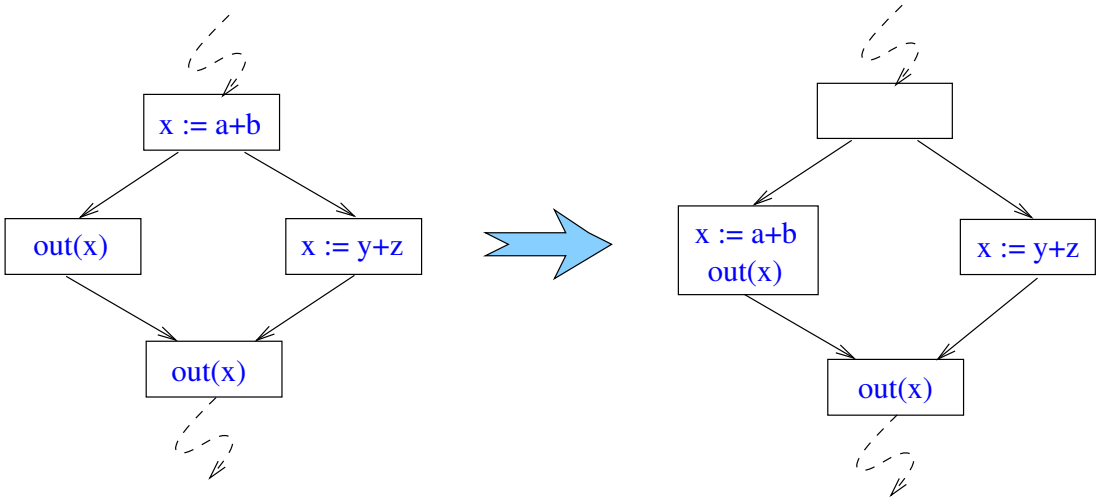
Das Grundmuster: Die Anweisung y := y+1 ist nicht tot, sondern lebendig (engl. live), aber geisterhaft (engl. faint): y zugewiesene Werte beeinflussen weder direkt noch indirekt Ausgabe- oder Bedingungswerte (und damit das beobachtbare Programmverhalten).



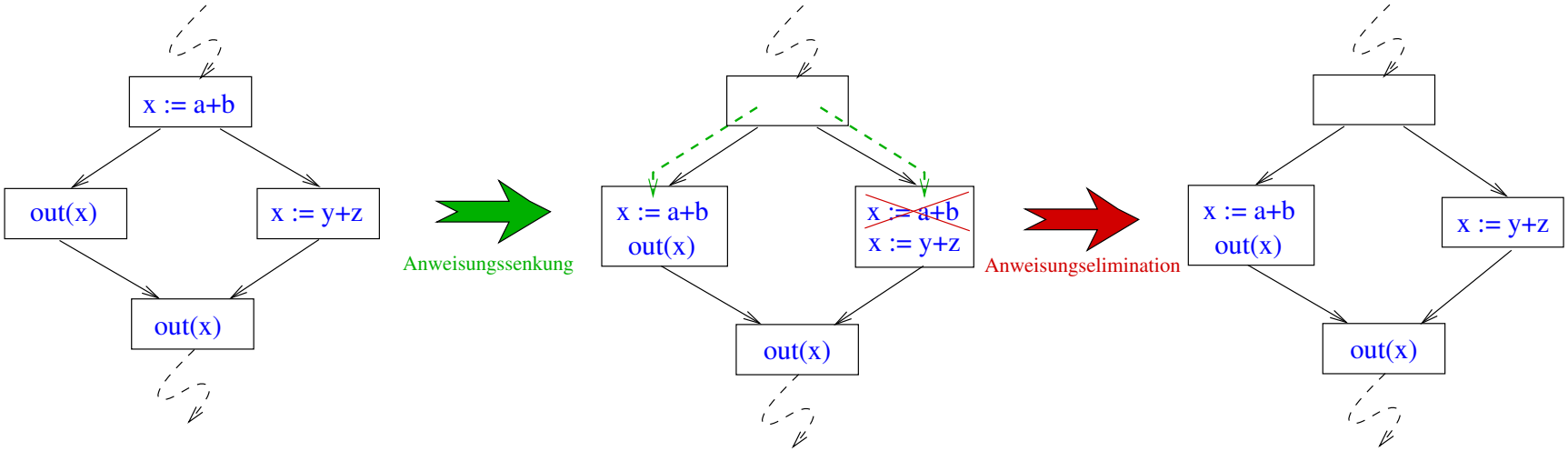
Partiell tote Anweisungen

...und ihre Elimination.

Das Grundmuster:



Die konzeptuelle Verfahrensidee:

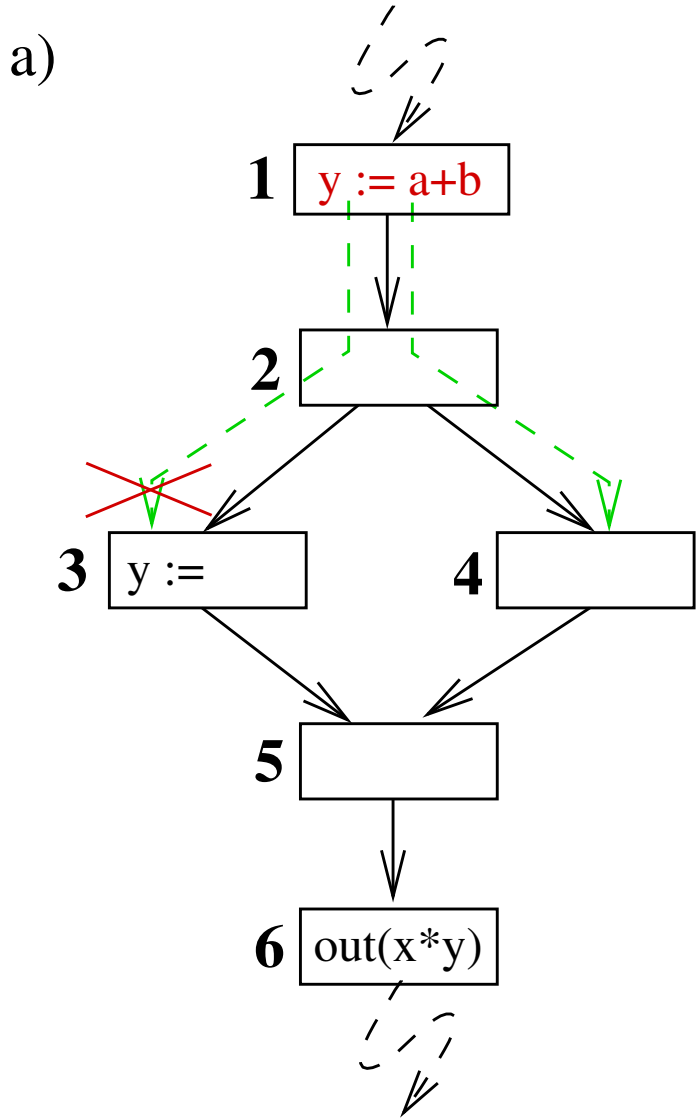


Kapitel 12.3.2

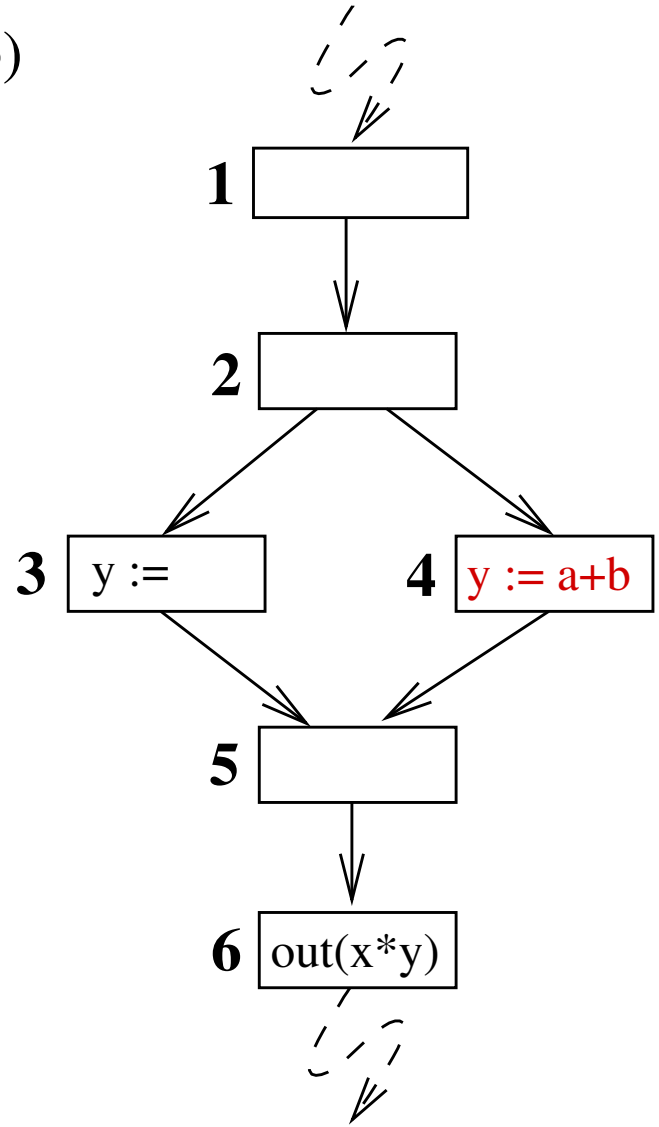
Beispiele

Bsp. 1: Elimination partiell toter Anweisungen

Ausgangsprogramm:

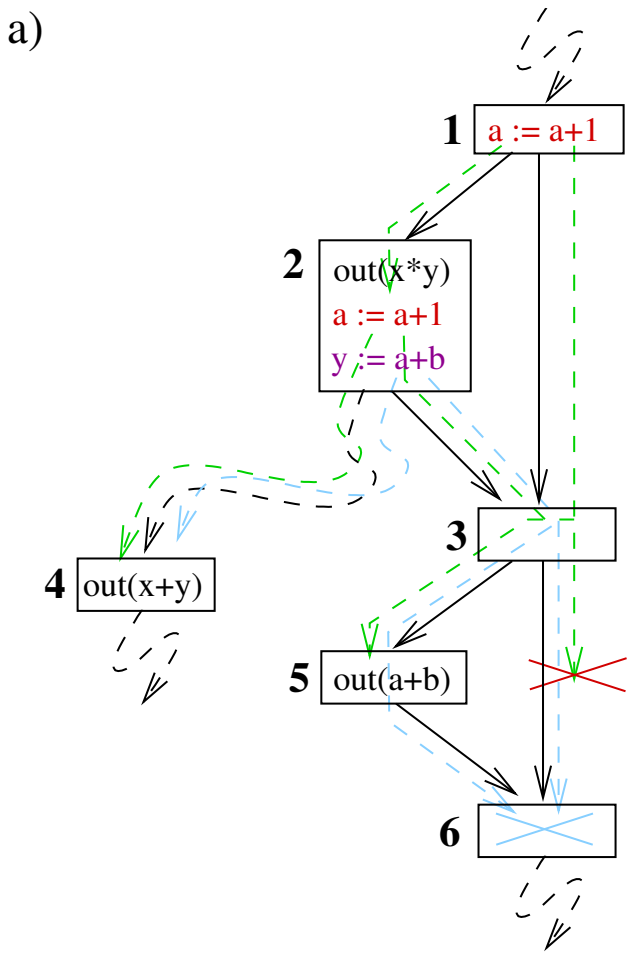


Optimiertes Programm:

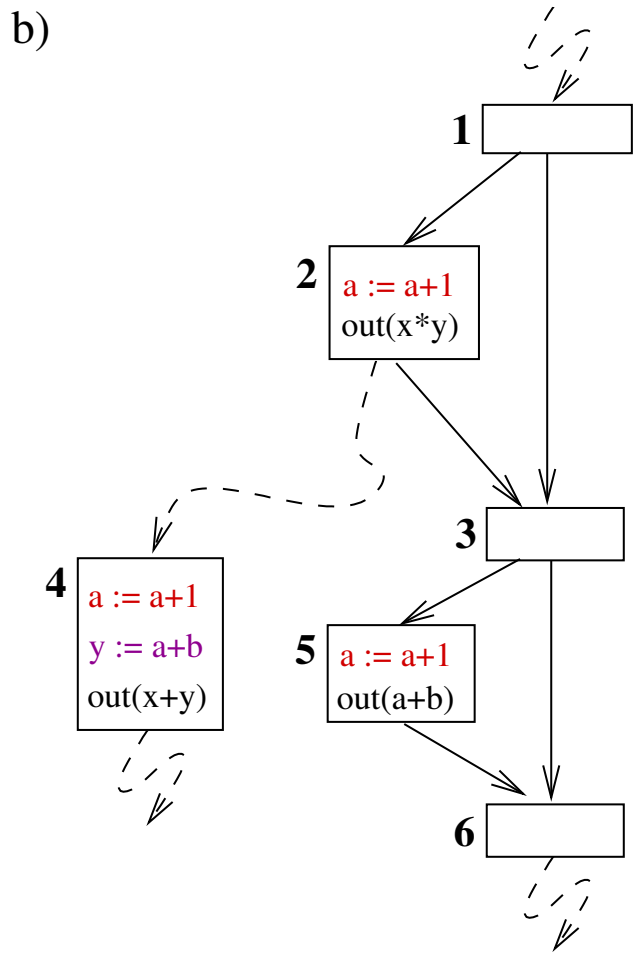


Bsp. 2: Elimination partiell toter Anweisungen

Ausgangsprogramm:



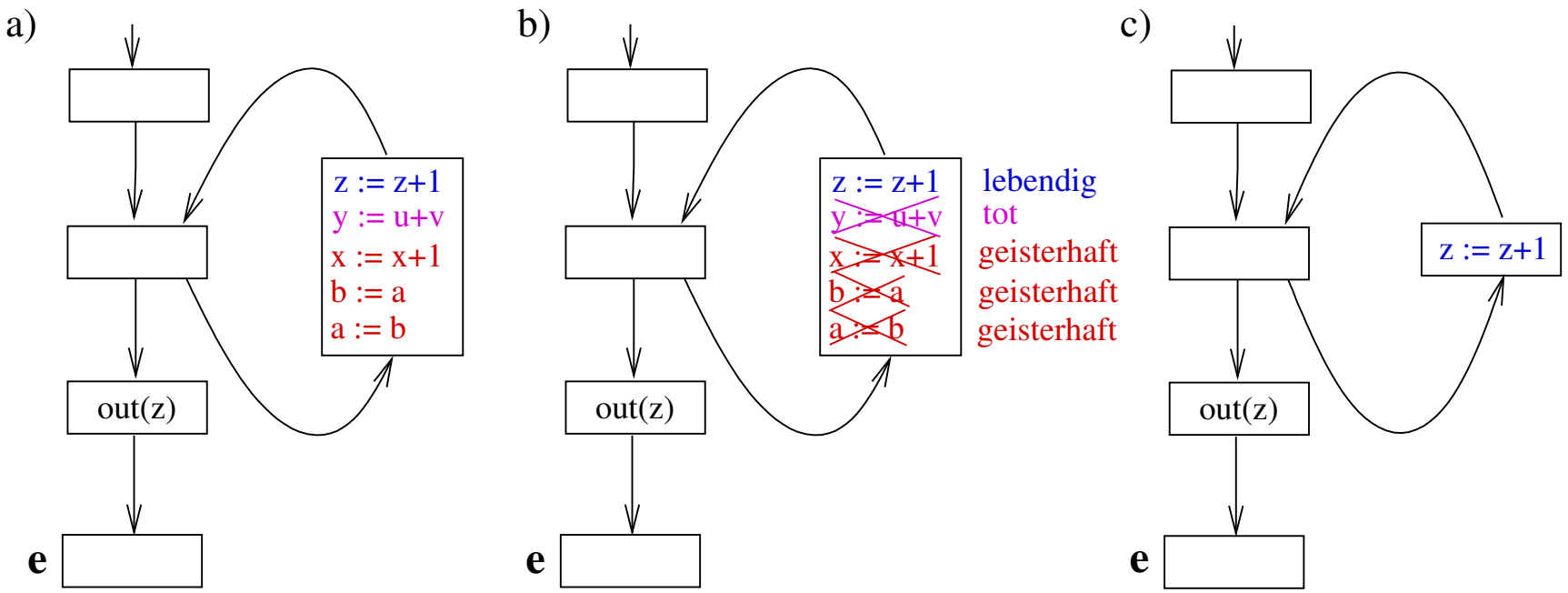
Optimiertes Programm:



...ist i.a. eine 'm2n'-Transformation (hier für $\alpha \equiv a := a + 1$).

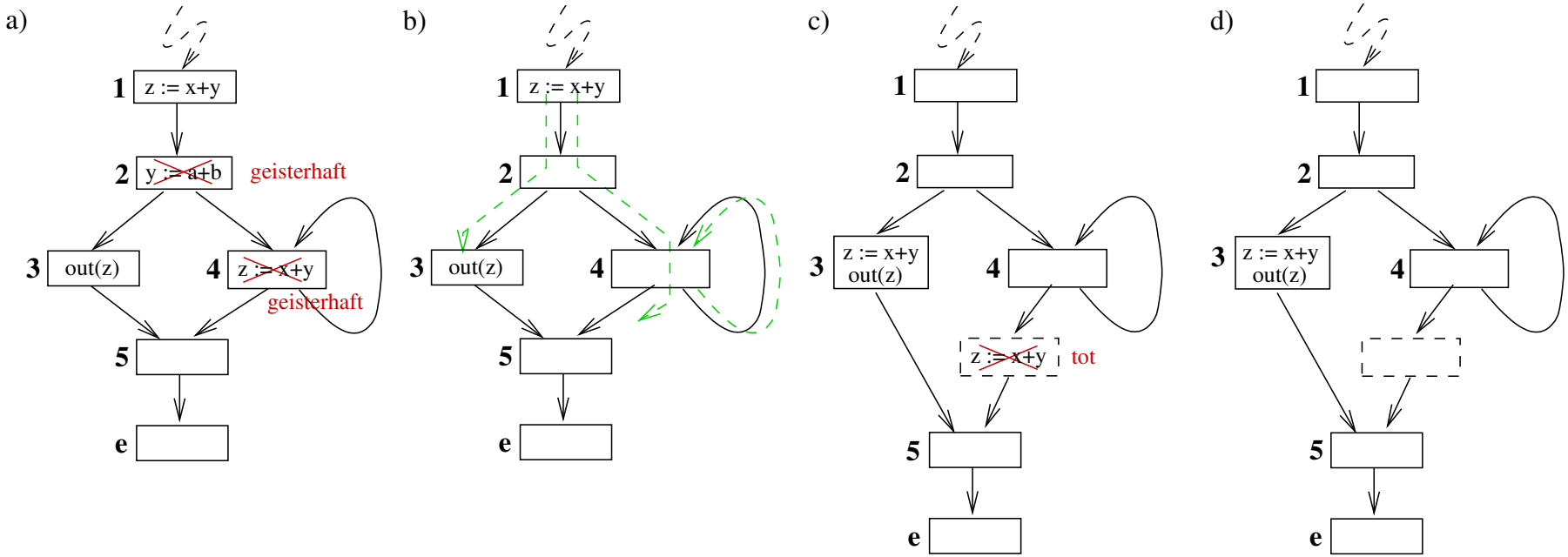
Bsp. 3: Elimination geisterhafter Anweisungen

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 12.3.3

Elementartransformationen

Tote und geisterhafte Anweisungen

Definition 12.3.3.1 (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 12.3.3.2 (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 12.3.3.3

Tote Anweisungen sind geisterhaft.

Beachte: Die Umkehrung von Proposition 12.3.3.3 gilt i.a. nicht.

Anweisungseliminationen

Definition 12.3.3.4 (Anweisungselimination)

Eine Programmtransformation, die einige Anweisungen aus dem Programm streicht, heißt Anweisungselimination (engl. code elimination).

Definition 12.3.3.5 (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 12.3.3.6 (Korrekte Anweisungselim.)

Eine Anweisungselimination ist korrekt, wenn sie eine t- oder g-Anweisungselimination ist.

Anweisungssenkungen

Definition 12.3.3.7 (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 12.3.3.8 (Korrekte Anweisungssenkung)

Eine α-Anweisungssenkung, $\alpha \equiv x := t$, ist korrekt, wenn während des Schiebens zu jedem Zeitpunkt gilt:

- Kein α-Vorkommen wird über eine Anweisung hinweggeschoben, die x liest oder modifiziert oder einen Operanden von t modifiziert (und α dadurch blockiert).
- Kein α-Vorkommen wird in einen Zusammenflussknoten geschoben, wenn dies nicht von jedem Vorgänger des Zusammenflussknotens aus geschieht.

Sprechweisen und Bezeichnungen

Definition 12.3.3.9 (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 12.3.4

Effekte zweiter Ordnung

Effekte zweiter Ordnung

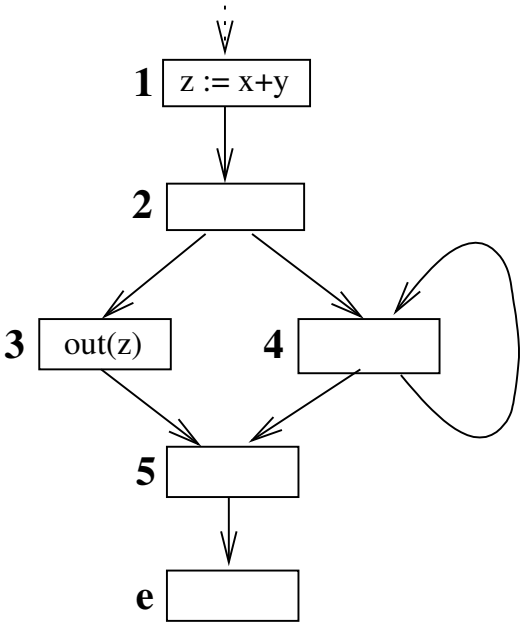
...(engl. second-order effects) treten in 4 Formen auf und sind essentiell für die kombinierte Wirkung der Elementartransformationen von EPTA und EPGA:

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

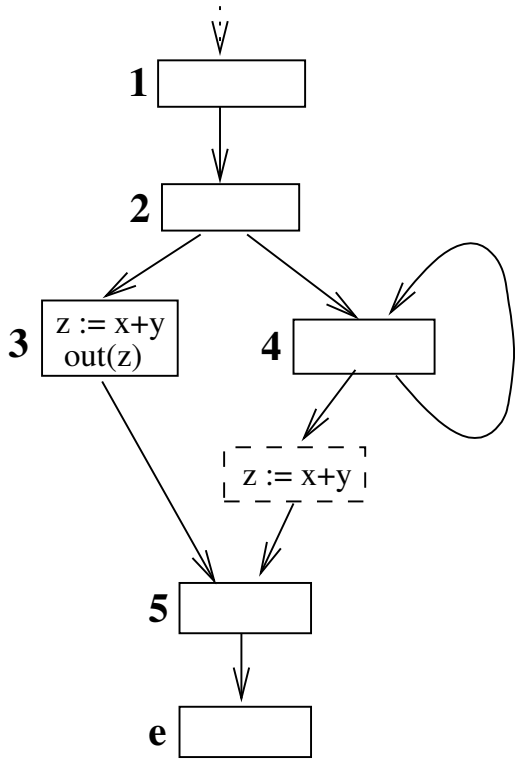
1) Senkungs-Eliminations-Effekt

...Zieleffekt: Eine Elementartransformation (hier: Senkung) ermöglicht anschließend eine Elimination (die erneut Senkungs- oder Eliminationspotential schaffen kann).

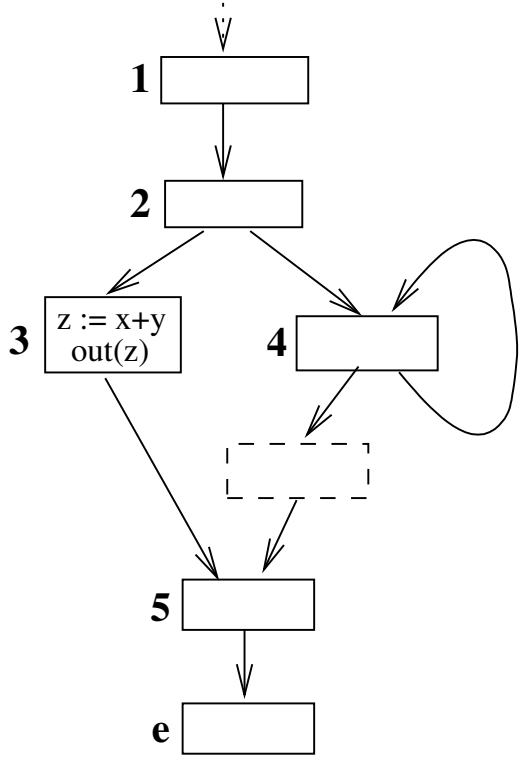
Ausgangsprogramm



Senkung 1. Ord.



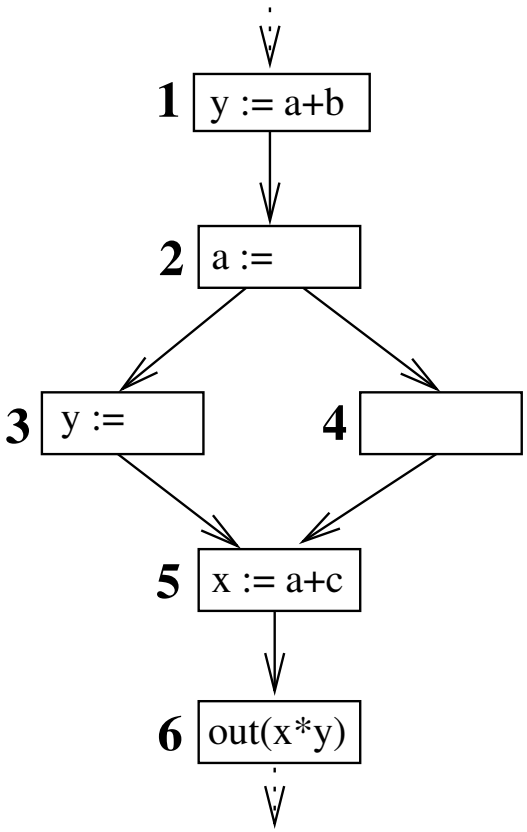
Elimination 2. Ord.



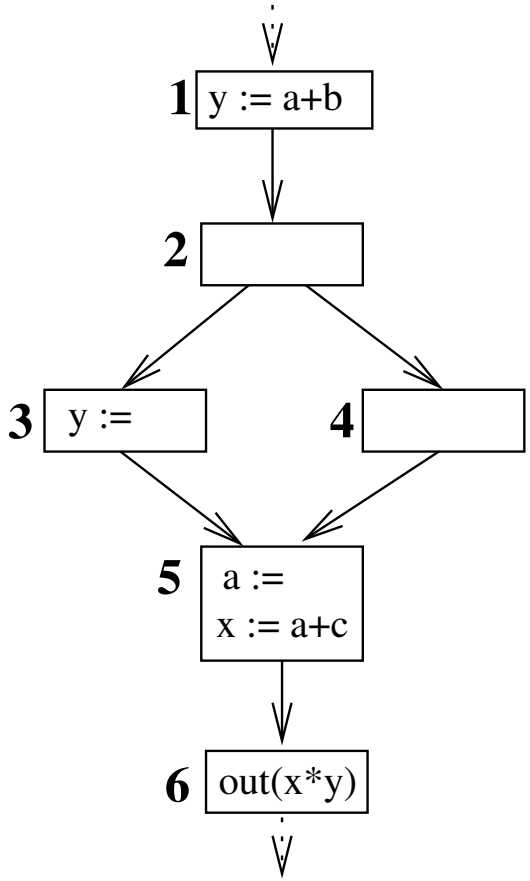
2) Senkungs-Senkungs-Effekt

...Potentialeffekt: Eine Elementartransformation (hier: Senkung) ermöglicht anschließend eine (weitere) Senkung, die erneut Senkungs- oder Eliminationspotential schaffen kann.

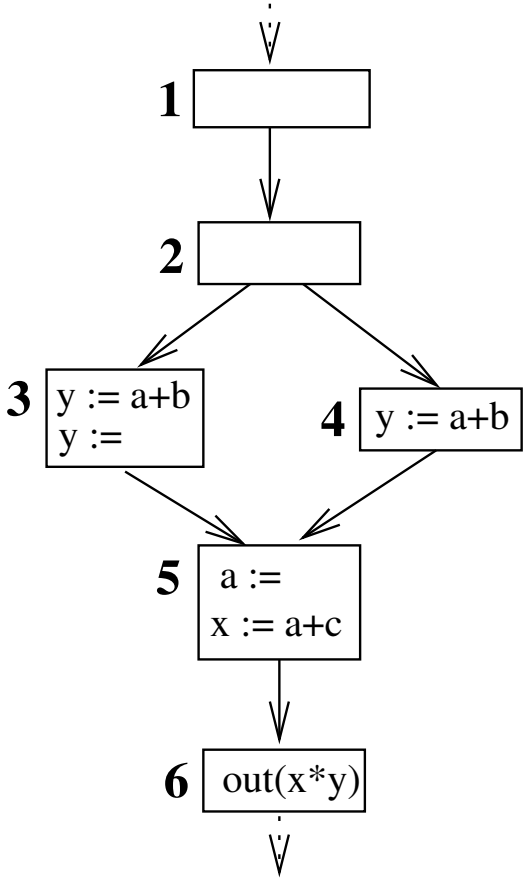
Ausgangsprogramm



Senkung 1. Ord.



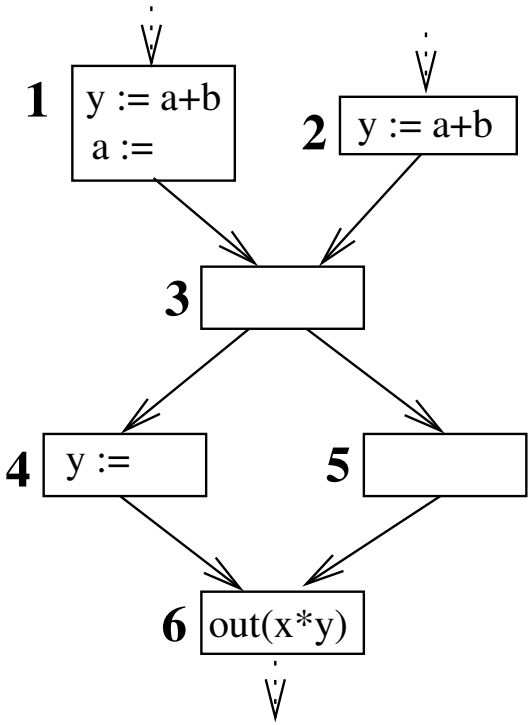
Senkung 2. Ord.



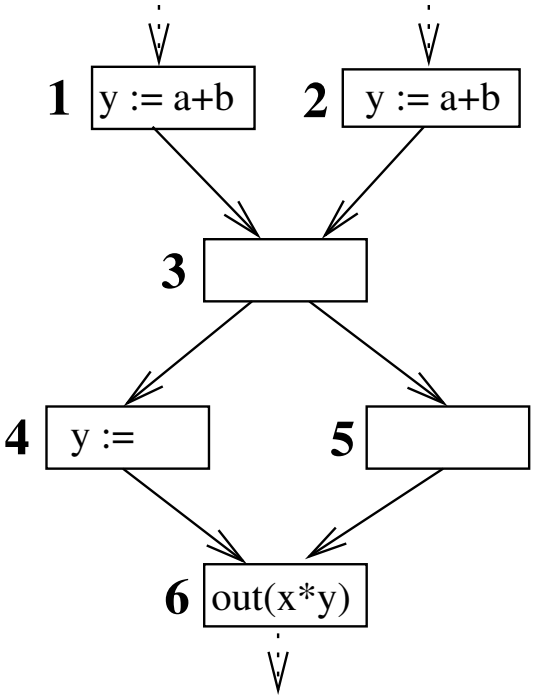
3) Eliminations-Senkungs-Effekt

...Potentialeffekt: Eine Elementartransformation (hier: Elimination) ermöglicht anschließend eine Senkung, die erneut Senkungs- oder Eliminationspotential schaffen kann.

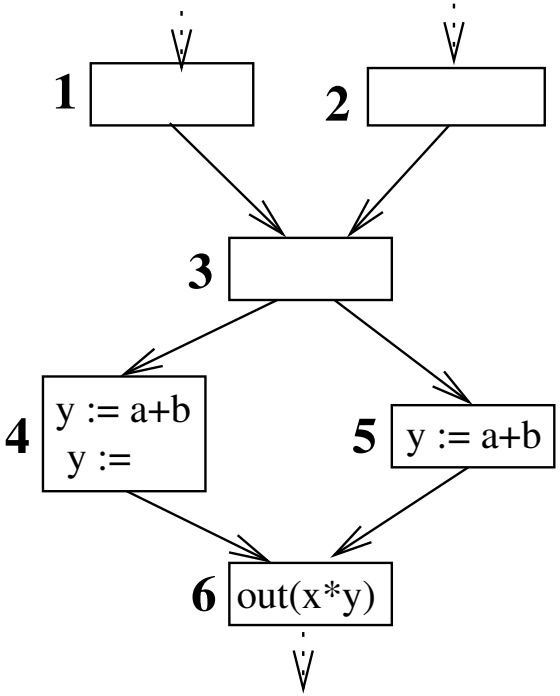
Ausgangsprogramm



Elimination 1. Ord.



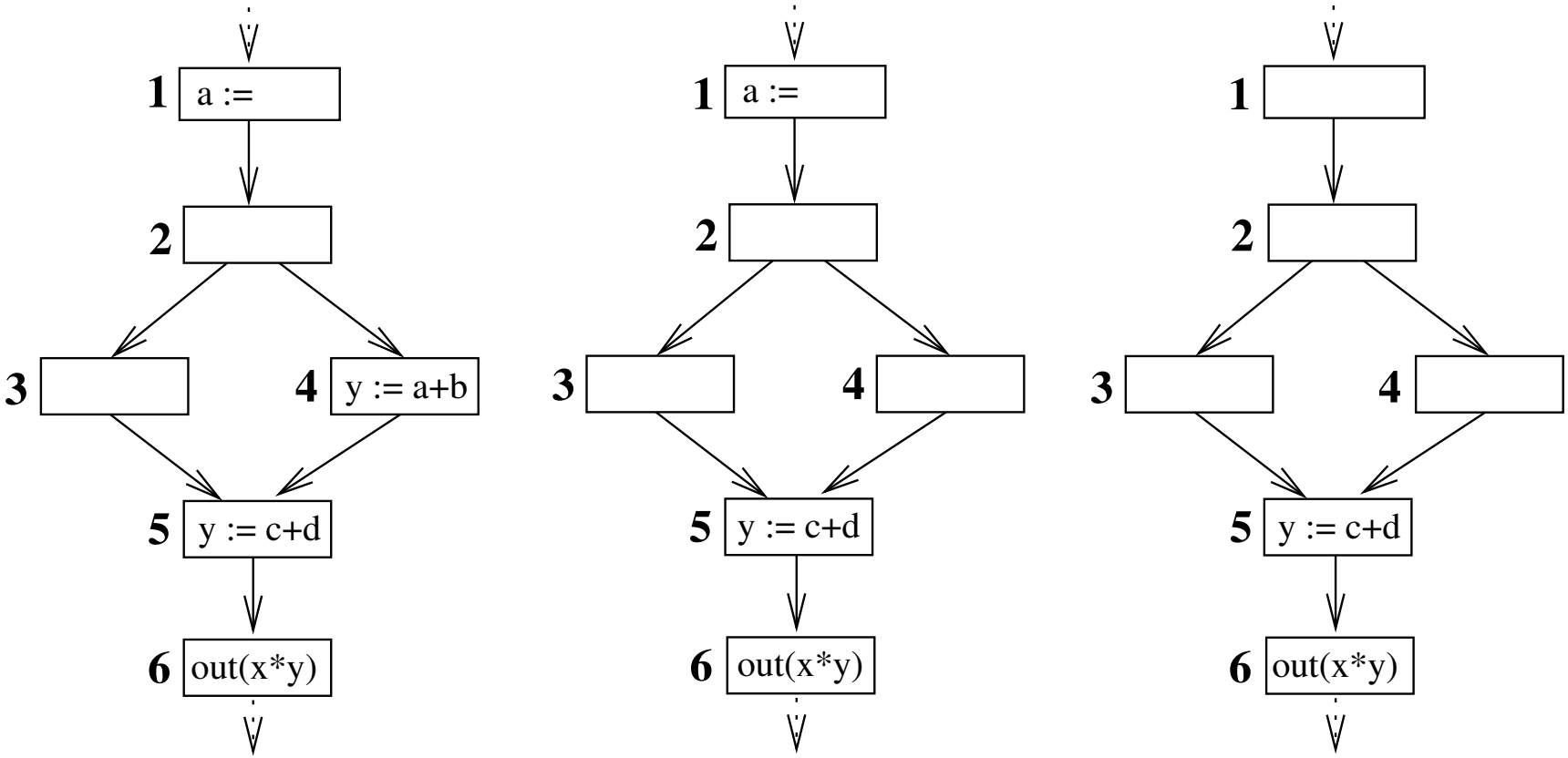
Senkung 2. Ord.



4) Eliminations-Eliminations-Effekt

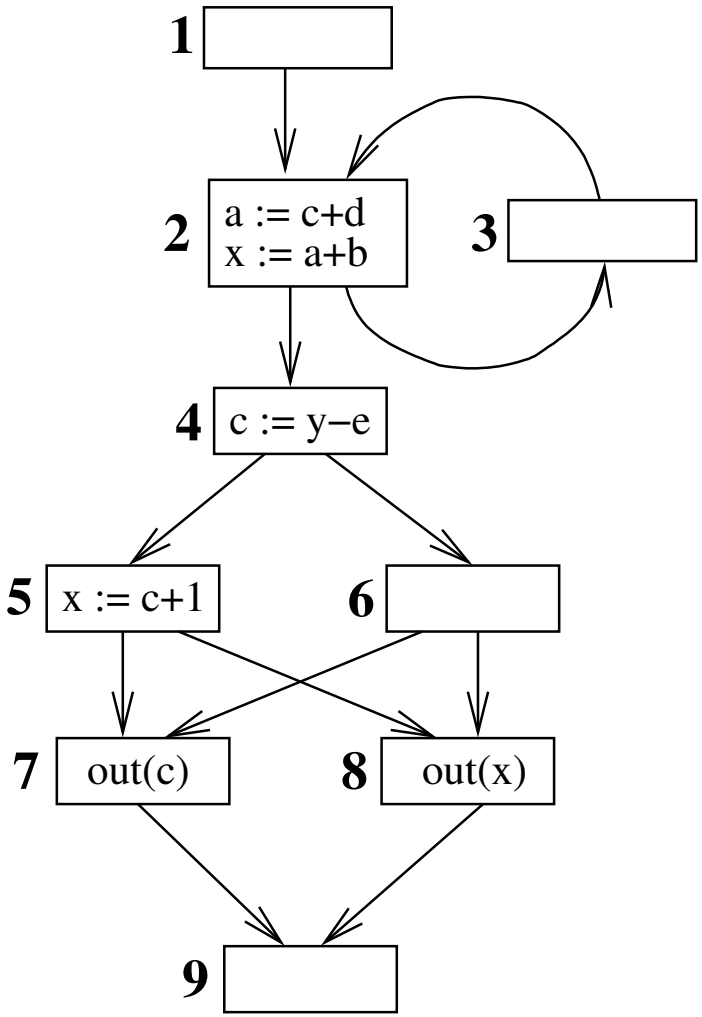
...Zieleffekt: Eine Elementartransformation (hier: Elimination) ermöglicht anschließend eine (weitere) Elimination (die erneut Senkungs- oder Eliminationspotential schaffen kann).

Ausgangsprogramm Elimination 1. Ord. Elimination 2. Ord.

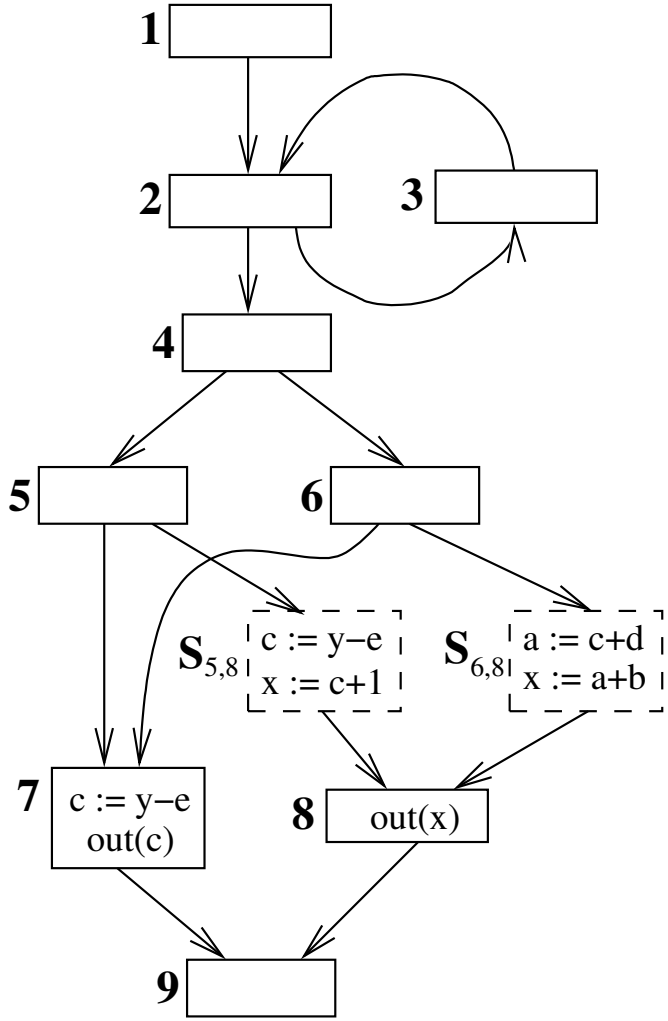


Sequenzwirkung von Effekten 2. Ordnung

Ausgangsprogramm



Optimiertes Programm



Kapitel 12.3.5

EPTA/EPGA: Transformationen

Unnötige Anweisungen

Sei G ein Programm.

Definition 12.3.5.1 (t-unnötige Anweisung)

Eine Anweisung α am Knoten n in G heißt t-unnötig gdw α ist tot am Knoten n.

Definition 12.3.5.2 (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 12.3.5.3 (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.

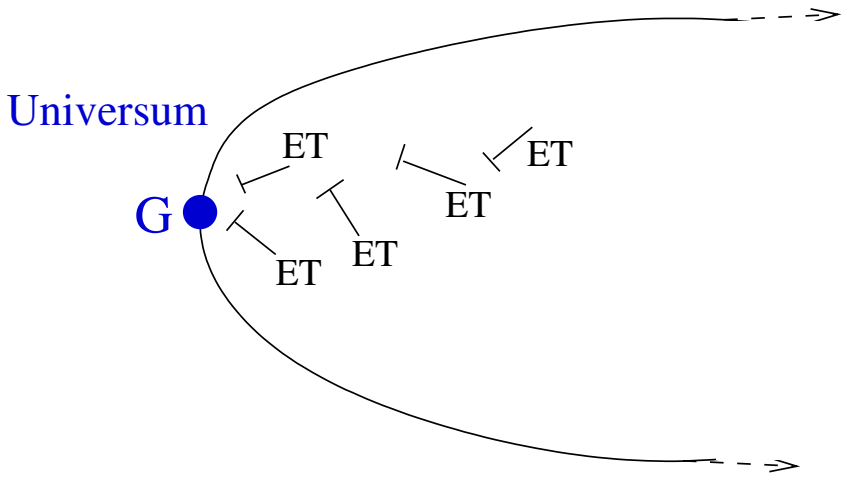
Transformationsrelation, Programmuniversum

- 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:

  $\mathcal{U}_T^G =_{df} \{G' \mid G \vdash_{(\tau_i)_{i \le k}} G',\ \tau = (\tau_i)_{i \in \mathbb{N}} \in T, k \in \mathbb{N}\}$, $T = T_{EPTA}^G$ oder $T = T_{EPGA}^G$: Das von G durch die Präfixe der Transformationsfolgen $\tau \in T$ aufgespannte Universum.



Kapitel 12.3.6

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 12.3.6.1 (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 12.3.6.2 (Quasiordnung)

Die Programmvergleichsrelation besser $\sqsubseteq$ ist eine Quasiordnung (d.h. reflexiv und transitiv, aber nicht antisymmetrisch).

Lemma 12.3.6.3 (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. $G \overset{\sqsubset}{\not\sim} G'$, 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 12.3.6.4 (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 $\mathcal{U}^G_T$:
$$\forall\, G' \in \mathcal{U}^G_T.\ G' \sqsubseteq_{\sim} G^*$$
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 12.3.6.5 (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 12.3.6.6 (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 12.3.6.7 (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 12.3.6.8 (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 12.3.6.9 (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 12.3.6.10

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 12.3.6.11 (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 12.3.6.12 (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 12.3.7

EPTA/EPGA: Optimalität

Maximale Transformationsfolgen

Sei G ein Programm und $T = T^G_{EPGA}$ oder $T = T^G_{EPGA}$.

Definition 12.3.7.1 (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 12.3.7.2 (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 12.3.7.3 (Maximale Transf.-Folge fair)

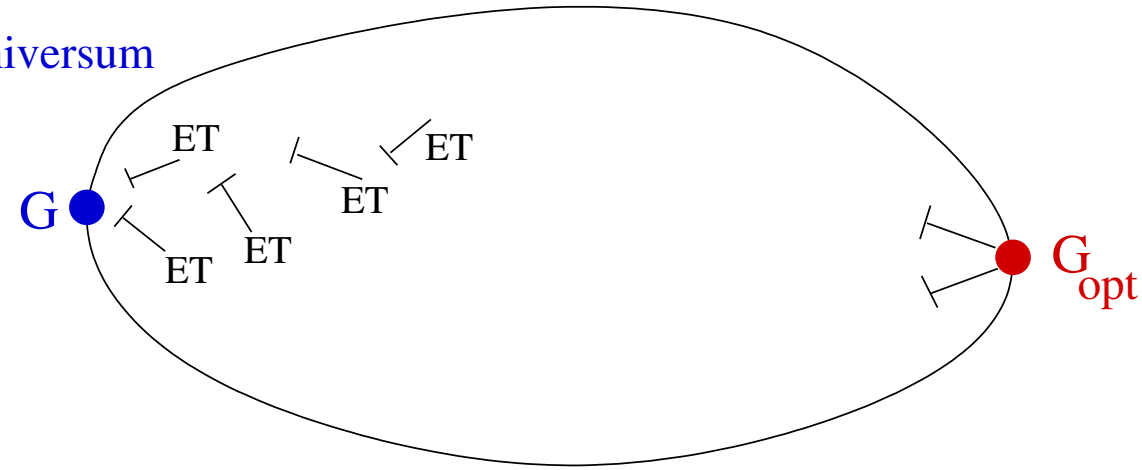
Maximale Transformationsfolgen für G sind fair.

Globale EPTA/EPGA-Optimalität

Sei G ein Programm und $T = T^G_{EPTA}$ oder $T = T^G_{EPGA}$.

Theorem 12.3.7.4 (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 12.3.7.4

...für zwei Beweisvarianten: Über

- Monotonie, Dominanz und Fixpunkttheorem 11.2.9 (Variante 1).
- Konfluenz und Termination der Transformationsrelation, s. Theorem 12.3.7.5 (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 11.2.9 folgt daraus die

- Korrektheit und Optimalität

fairer EPTA- und EPGA-Transformationsfolgen.

Variante 2: Konfluenz, Terminierung

Zeige: Die ETPA- und EPGA-Transformationsrelationen

- $\cdot \vdash_\tau \cdot, \tau \in AS \cup ETA$ (EPTA)
- $\cdot \vdash_\tau \cdot, \tau \in AS \cup EGA$ (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 12.3.7.5 (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 12.3.7.6 (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 12.3.7.7 (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 12.3.7.8 (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 12.3.7.9 (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 12.3.7.10 (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 12.3.7.11 (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 12.3.7.12

Theorem 12.3.7.8 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 12.3.6.1) als G^{opt}_{EPTA}:

Lemma 12.3.7.13 (EPGA besser als EPTA)

$$\forall\, G \in \mathcal{G}.\ G^{opt}_{EPTA} \sqsubseteq_{\approx} G^{opt}_{EPGA}$$

Übungsaufgabe

Gemäß Lemma 12.3.7.13 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:

- $T_1 =_{df} T^G_{AS,EGA} = \{\tau \,|\, \tau = (\tau_i)_{i\in\mathbb{N}} \in \mathcal{L}((AS + EGA)^*)\}$
- $T_2 =_{df} T^G_{EGA\circ(AS,ETA)} = \{\tau \,|\, \tau = (\tau_i)_{i\in\mathbb{N}} \in \mathcal{L}(EGA_{\mathcal{AM}} * (AS + ETA)^*)\}$
- $T_3 =_{df} T^G_{(AS,ETA)\circ EGA\circ(AS,ETA)} = \{\tau \,|\, \tau = (\tau_i)_{i\in\mathbb{N}} \in \mathcal{L}((AS+ETA)^* * EGA_{\mathcal{AM}} * (AS+ETA)^*)\}$

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 (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 12.3.8

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 12.3.8.1 (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 12.3.8.2 (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:

$$\text{N-Insert}_n^\alpha \quad =_{df} \quad \nu\text{-N-SINK}_n^\alpha * \text{Blocked}_n^\alpha$$

$$\text{X-Insert}_n^\alpha \quad =_{df} \quad \nu\text{-X-SINK}_n^\alpha * \sum_{m \in succ(n)} \overline{\nu\text{-N-SINK}_m^\alpha}$$

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

Wie lauten die Spezifikationen der Analysen zur

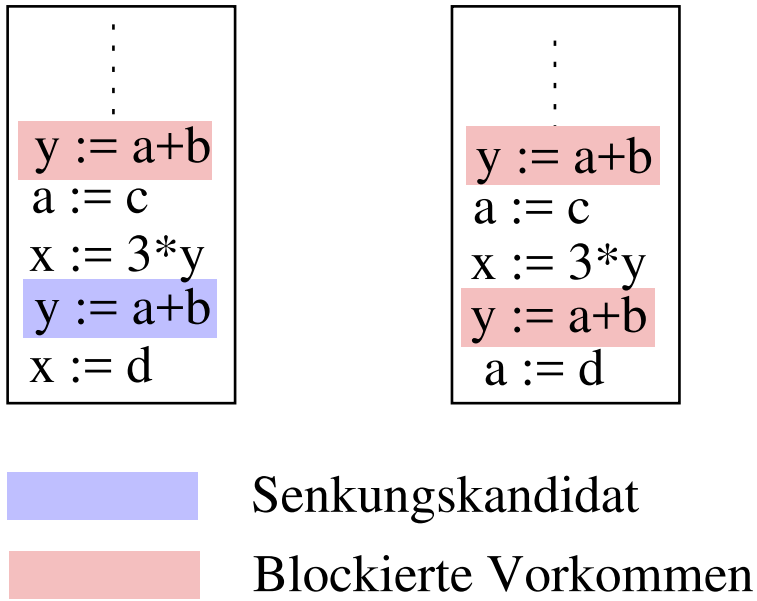
- Erkennung toter Anweisungen
- Erkennung geisterhafter Anweisungen
- Senkung von Anweisungen

im Stil von Kapitel 7?

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 12.4

Partiell redundante Anweisungen

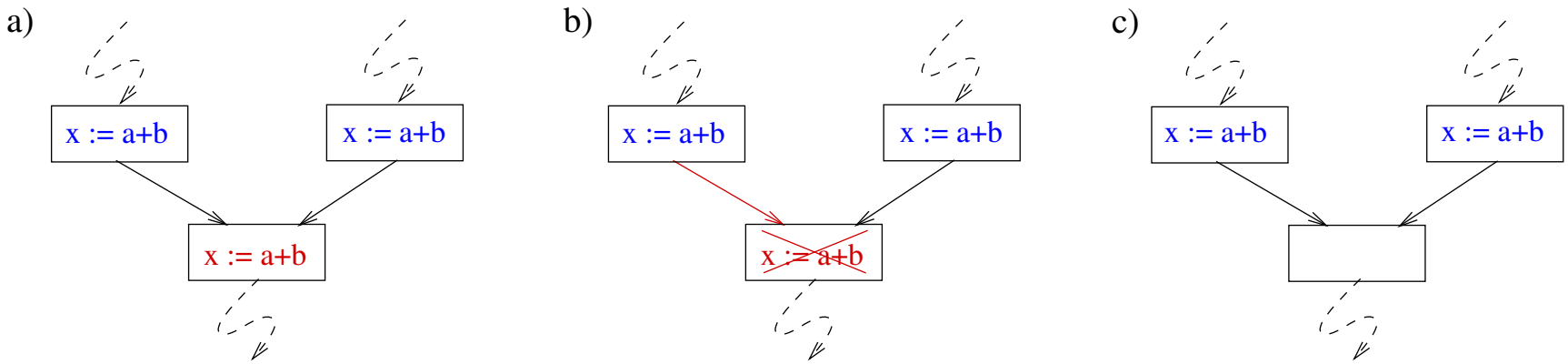
Kapitel 12.4.1

Motivation

Redundante Anweisungen

...und ihre Elimination.

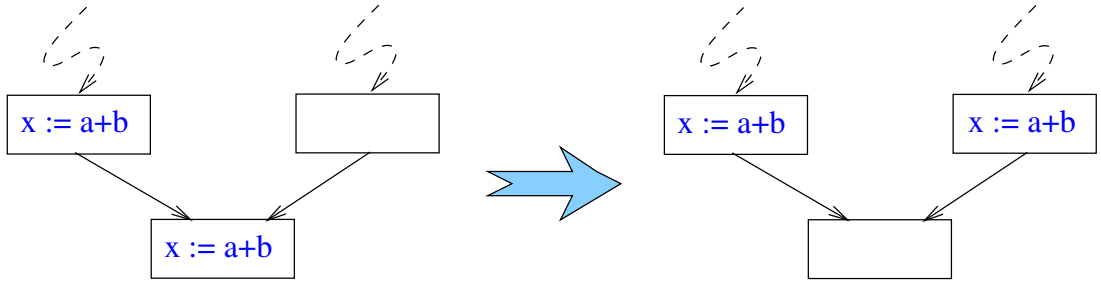
Das Grundmuster: Das rote Vorkommen der Anweisung `x := a+b` ist redundant (oder total redundant) (engl. (totally) redundant) gegenüber den beiden blauen, da weder `x` noch `a` oder `b` zwischen den Vorkommen geschrieben werden.



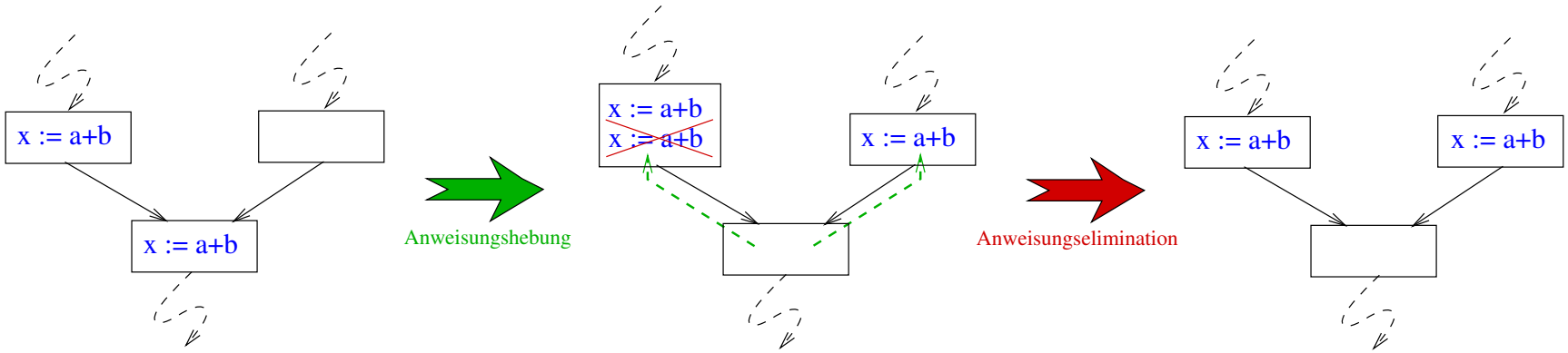
Partiell redundante Anweisungen

...und ihre Elimination.

Das Grundmuster:



Die konzeptuelle Verfahrensidee:



Kapitel 12.4.2

Elementartransformationen

Redundante Anweisungen

Definition 12.4.2.1 (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 12.4.2.2 (Elimination redundanter Anw.)

Eine Anweisungselimination, die einige redundante Anweisungen aus dem Programm streicht, heißt Elimination redundanter Anweisungen (engl. redundant assignment elimination).

Definition 12.4.2.3 (Korrekte Anweisungselim.)

Eine Anweisungselimination ist korrekt, wenn sie eine Elimination redundanter Anweisungen ist.

Anweisungshebungen

Definition 12.4.2.4 (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 12.4.2.5 (Korrekte Anweisungshebung)

Eine α-Anweisungshebung, $\alpha \equiv x := t$, ist korrekt, wenn während des Schiebens zu jedem Zeitpunkt gilt:

- Kein α-Vorkommen wird über eine Anweisung hinweggeschoben, die x liest oder modifiziert oder einen Operanden von t modifiziert (und α dadurch blockiert).
- Kein α-Vorkommen wird (rückwärts) in einen Verzweigungsknoten geschoben, wenn dies nicht von jedem Nachfolger des Verzweigungsknotens aus geschieht.

Sprechweisen und Bezeichnungen

Definition 12.4.2.6 (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 12.4.2.7

Eine Anweisung blockiert die Hebung einer Anweisung gdw sie deren Senkung blockiert.

Kapitel 12.4.3

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

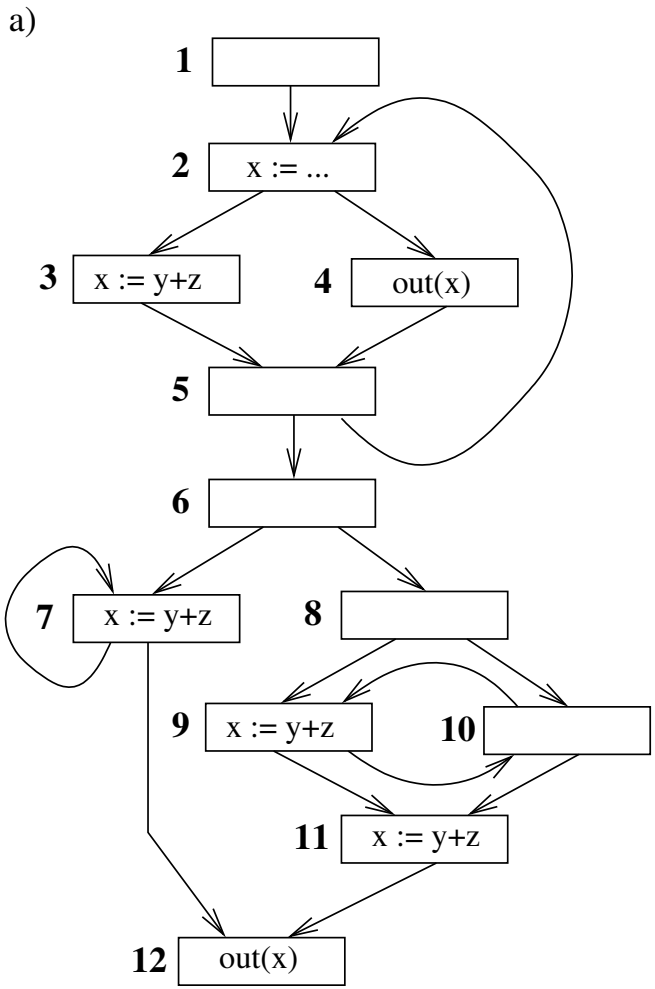
Übungsaufgabe

Gib für jeden der vier Effekte zweiter Ordnung von EPRA ein Beispiel an:

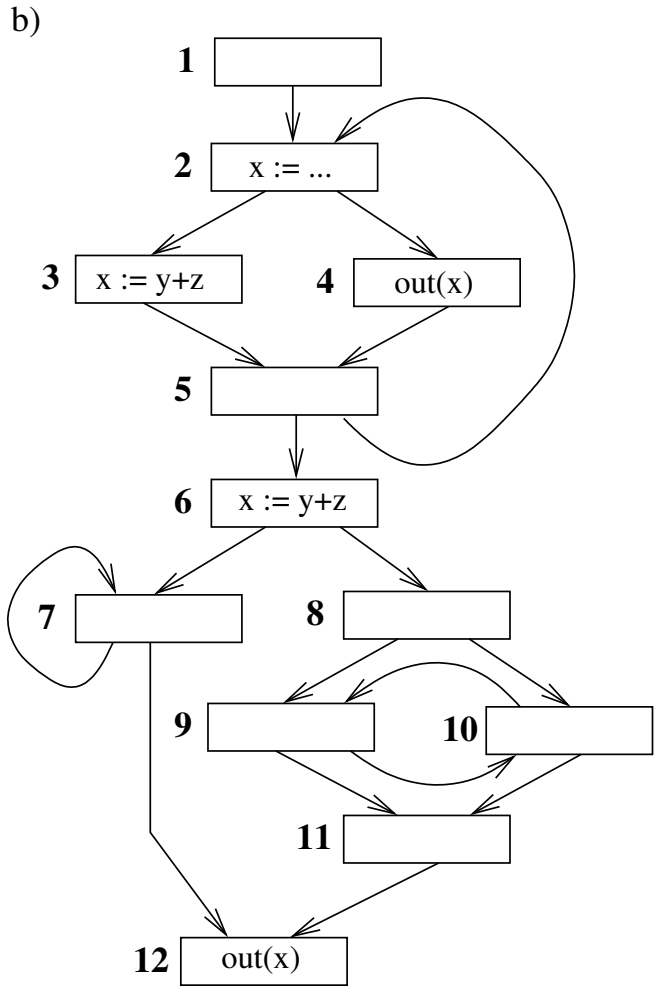
1. Hebungs-Eliminations-Effekt
2. Hebungs-Hebungs-Effekt
3. Eliminations-Hebungs-Effekt
4. Eliminations-Eliminations-Effekt

Sequenzwirkung von Effekten 2. Ordnung

Ausgangsprogramm



Optimiertes Programm



Beachte: Kein Schieben von Anweisungen in Schleifen!

Kapitel 12.4.4

EPRA: Transformation

Unnötige Anweisungen

Sei G ein Programm.

Definition 12.4.4.1 (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 12.4.4.2 (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.

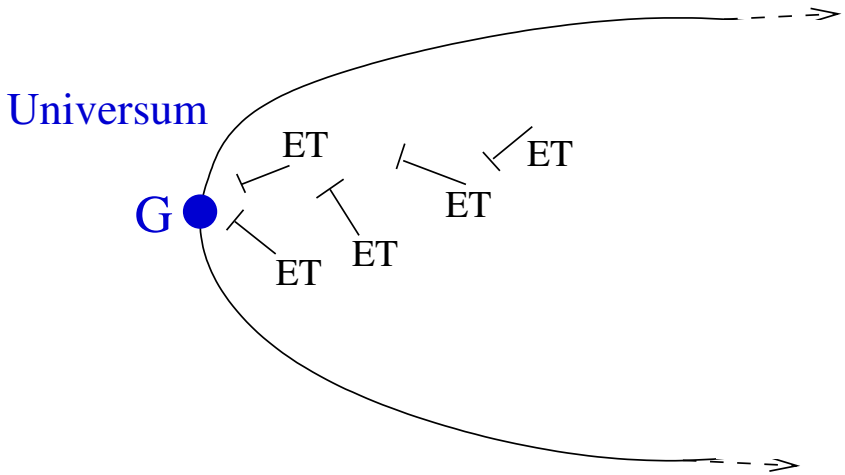
Transformationsrelation, Programmuniversum

- 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:

  $\mathcal{U}^G_{T_{EPRA}} =_{df} \{G' \mid G \vdash_{(\tau_i)_{i \leq k}} G', \tau = (\tau_i)_{i \in \mathbb{N}} \in T^G_{EPRA}, k \in \mathbb{N}\}$: Das von G durch die Präfixe der Transformationsfolgen $\tau \in T^G_{EPRA}$ aufgespannte Universum.



Kapitel 12.4.5

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 12.4.5.1 (Besser)

G' heißt besser als G'' (in Zeichen: $G'' \sqsubseteq G'$) gdw. G' ist besser als G'' i.S.v. Definition 12.3.6.1, 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 12.4.5.2 (Quasiordnung)

Die Programmvergleichsrelation besser $\sqsubseteq_{\sim}$ ist eine Quasiordnung (d.h. reflexiv und transitiv, aber nicht antisymmetrisch).

Lemma 12.4.5.3 (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 12.4.5.4 (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 12.4.5.5 (Lokal beste (optimale) Prg.)

1. Ein Programm $G^* \in \mathcal{U}^G_{T_{EPRA}}$ heißt lokal EPRA-best gdw:

$$\forall\, G' \in \mathcal{U}^{G^*}_{T_{EPRA}}.\ G^* \sim G'$$

2. Bezeichne $\mathcal{G}^{lokopt}_{T_{EPRA}}(G)$ die Menge der lokal EPRA-besten (oder lokal EPRA-optimalen) Programme in $\mathcal{U}^G_{T_{EPRA}}$.

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 12.4.5.6 (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 12.4.5.7 (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 12.4.5.8 (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 12.4.5.9 (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 12.4.5.10

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 12.4.5.11 (Universumskorrekt)

Ist $\tau \in T^G_{EPRA}$, so ist $G_\tau \in \mathcal{U}^G_{T_{EPRA}}$.

Lemma 12.4.5.12 (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 12.4.6

EPRA: Optimalität

Maximale Transformationsfolgen

Sei G ein Programm.

Definition 12.4.6.1 (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 12.4.6.2 (Faire Transf.-Folge)

Eine Transformationsfolge $\tau = (\tau_i)_{i \in \mathbb{N}}$, $\tau \in T^G_{EPRA}$, für G heißt fair, wenn

$$\forall\, k \in \mathbb{N}.\ (\tau_i)_{i \leq k} \text{ nicht maximal } \Rightarrow\ \exists k' > k.\ G_{(\tau_i)_{i \leq k}} \overset{\sqsubset}{\not\sim} G_{(\tau_i)_{i \leq k'}}$$

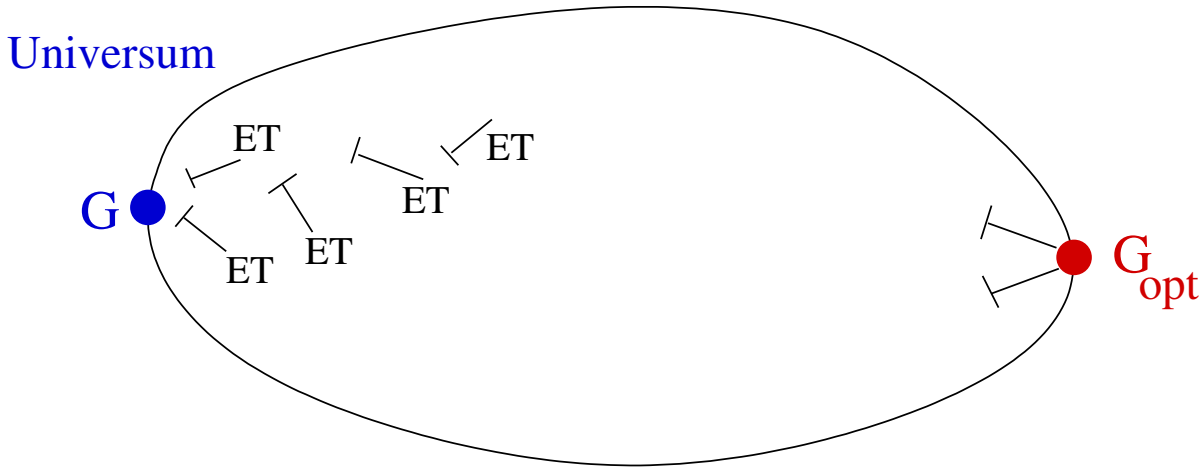
Lemma 12.4.6.3 (Maximale Transf.-Folge fair)

Maximale Transformationsfolgen für G sind fair.

Globale EPRA-Optimalität

Theorem 12.4.6.4 (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 12.4.6.4

...analog zum Beweis von Theorem 12.3.7.4 in zwei Varianten: Über

- Monotonie, Dominanz und Fixpunkttheorem 11.2.9 (Variante 1).
- Konfluenz und Termination der Transformationsrelation, s. Theorem 12.4.6.5 (Variante 2).

EPTA/EPGA-T.-Relation: Konfluenz, Term.

Theorem 12.4.6.5 (Konfluenz, Terminierung)

Die EPRA-Transformationsrelation

- $\cdot \vdash_\tau \cdot, \tau \in \mathit{AH} \cup \mathit{ERA}$

ist (bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken) konfluent und terminierend.

Determiniertheit maximaler Transformationen

Korollar 12.4.6.6 (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 12.4.6.7 (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 12.4.6.8 (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 12.4.6.9 (Existenz global opt. Transf.)

$$\forall\, G \in \mathcal{G}.\ T^{opt}_{EPRA}(G) \neq \emptyset$$

Existenz und Konstruktion

...global optimaler Programme und Transformationen.

Korollar 12.4.6.10 (Existenz global opt. Programme)

$$\forall\, G \in \mathcal{G}.\ \mathcal{G}^{opt}_{T_{EPRA}}(G) \neq \emptyset$$

Korollar 12.4.6.11 (Determiniertheit)

Bis auf irrelevante Umsortierungen von Anweisungen gilt:

$$|\mathcal{G}^{opt}_{T_{EPRA}}(G)| = 1$$

Korollar 12.4.6.12

Theorem 12.4.6.7 beschreibt konstruktiv und effektiv die Bildung endlicher maximaler (und damit fairer und optimaler) EPRA-Transformationsfolgen.

Kapitel 12.4.7

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: Red. Anweisungsanalyse

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: Anweisungshebungsanalyse

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

Wie lauten die Spezifikationen der Analysen zur

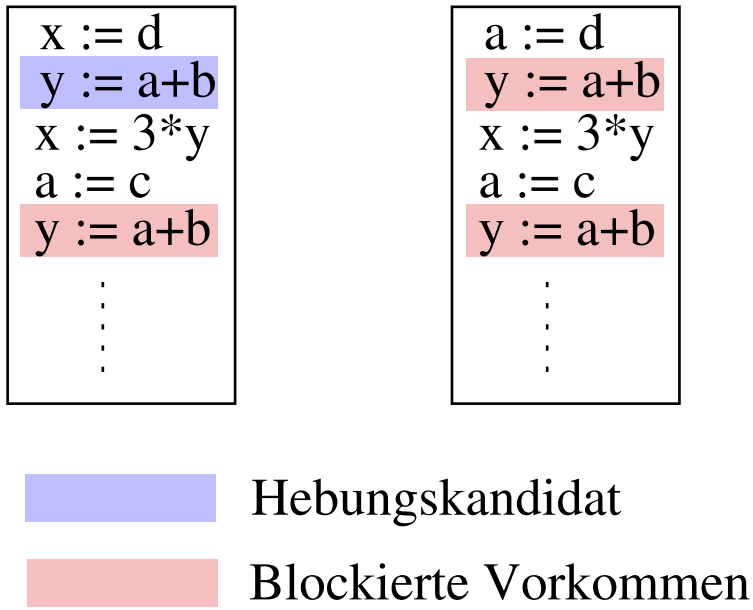
- Erkennung redundanter Anweisungen
- Hebung von Anweisungen

im Stil von Kapitel 7?

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 12.5

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (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 12 (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 13

Transformationskombinationen

Kapitel 13.1

EPTRA: EPTA/EPRA-Kombination

Kapitel 13.1.1

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 12.3 und 12.4):

Theorem 13.1.1.1 (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)

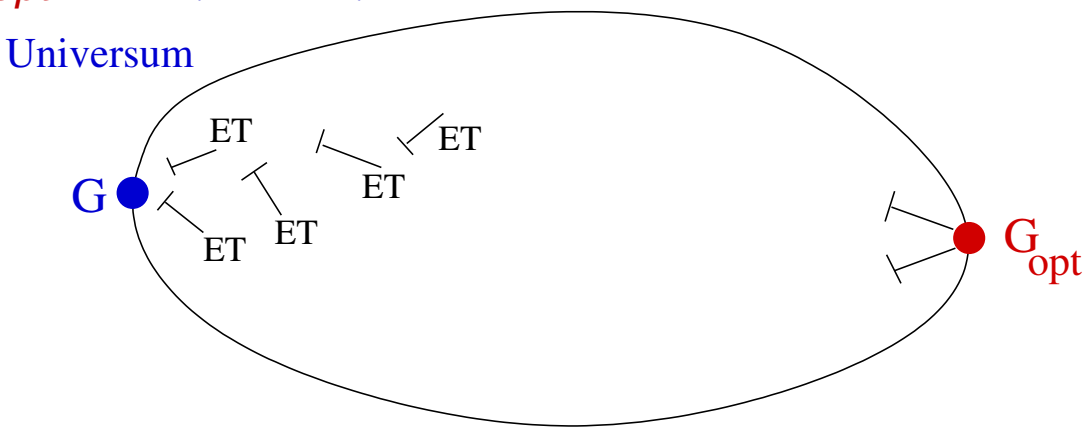
sind (bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken) konfluent und terminierend.

EPTA/EPRA: Globale Optimalität

...für faire EPTA/EPRA-Transformationsfolgen gilt (s. Kapitel 12.3 und 12.4):

Theorem 13.1.1.2 (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 13.1.2

EPTRA: Transformation

EPTRA-Transformationsfolgen

Sei G ein Programm.

Definition 13.1.2.1 (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.

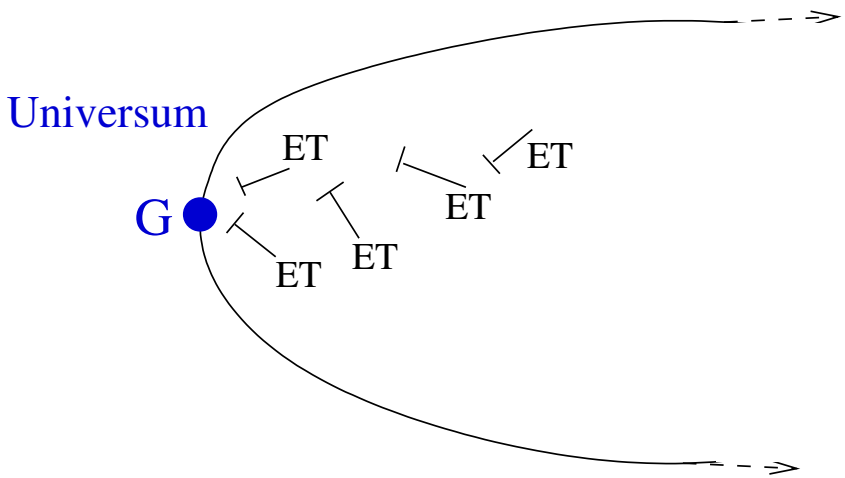
Transformationsrelation, Programmuniversum

- 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:

  $\mathcal{U}^G_{T_{EPRTA}} =_{df} \{G' \mid G \vdash_{(\tau_i)_{i \le k}} G', \tau = (\tau_i)_{i \in \mathbb{N}} \in T^G_{EPTRA}, k \in \mathbb{N}\}$: Das von G durch die Präfixe der Transformationsfolgen $\tau \in T^G_{EPTRA}$ aufgespannte Universum.



Kapitel 13.1.3

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 13.1.3.1 (Besser)

G' heißt besser als G'' (in Zeichen: $G'' \sqsubseteq G'$) gdw G' ist besser als G'' i.S.v. Definition 12.3.6.1, 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 13.1.3.2 (Quasiordnung)

Die Programmvergleichsrelation besser $\sqsubseteq_{\approx}$ ist eine Quasiordnung (d.h. reflexiv und transitiv, aber nicht antisymmetrisch).

Lemma 13.1.3.3 (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. $G \sqsubseteq_{\not\approx T} G'$, 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 13.1.3.4 (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 (oder optimale) Programme

Sei G ein Programm.

Definition 13.1.3.5 (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 13.1.3.6 (EPRTA-bessere Transf.)

Eine Transformationsfolge (oder Transformation) $\tau \in \mathrm{T}^G_{EPRTA}$ heißt EPTRA-besser für G als $\tau' \in T_{EPTRA}$ gdw: $G_{\tau'} \sqsubseteq\!\!\!\sim G_\tau$

Definition 13.1.3.7 (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 13.1.4

EPTRA: Optimalität

EPTRA: Nicht-Konfluenz, Termination

...für die EPTRA-Transformationsrelation gilt:

Theorem 13.1.4.1 (Nicht-Konfluenz, Terminierung)

Die EPTRA-Transformationsrelation

- $\cdot \vdash_\tau \cdot, \tau \in AS \cup ETA \cup AH \cup ERA$ (EPTRA)

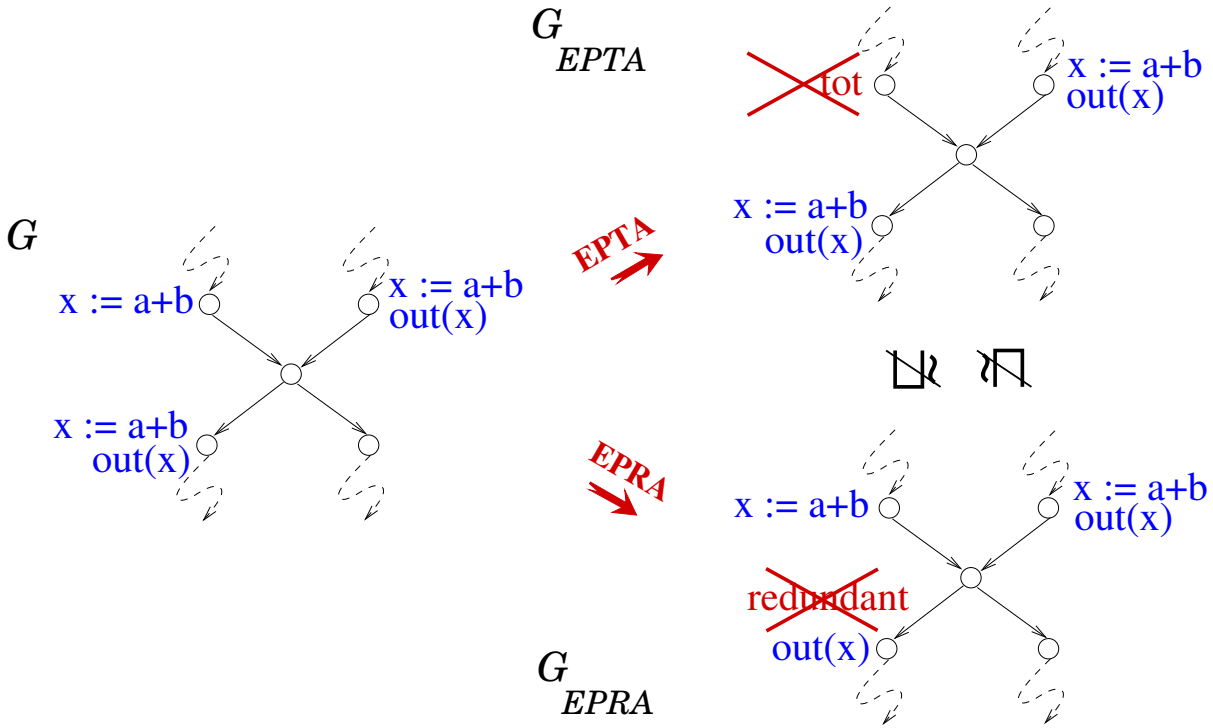
ist (bis auf irrelevante Umsortierungen von Anweisungen im Programm) terminierend, aber (i.a.) nicht konfluent.

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 13.1.4.1

...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

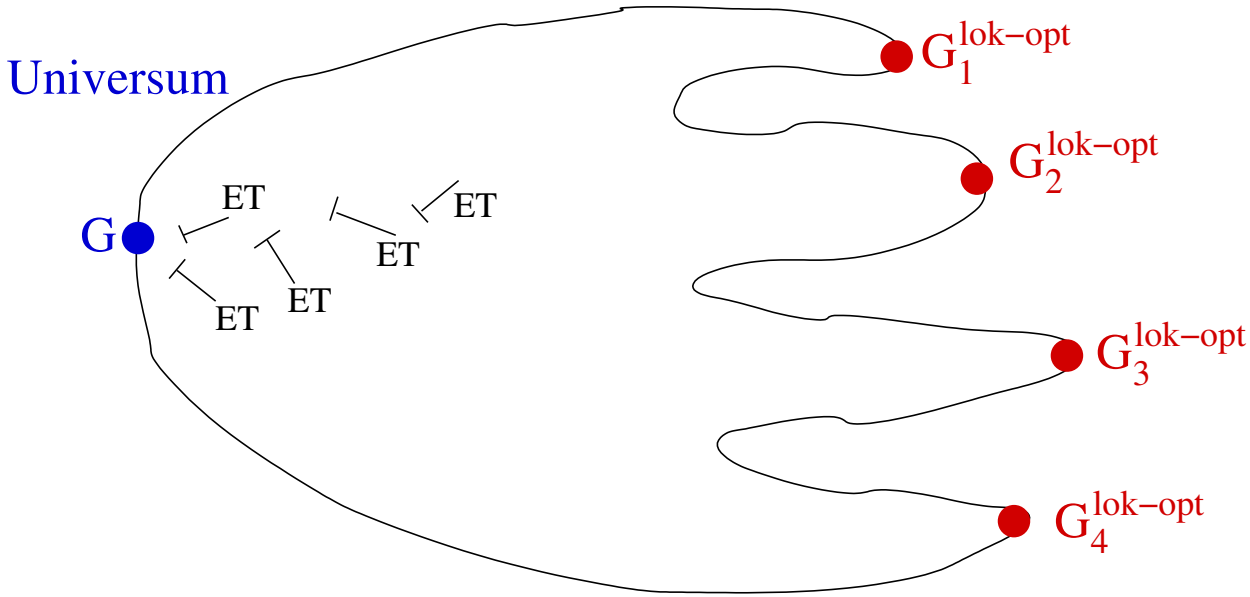
...für faire EPTRA-Transformationsfolgen gilt:

Theorem 13.1.4.2 (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.

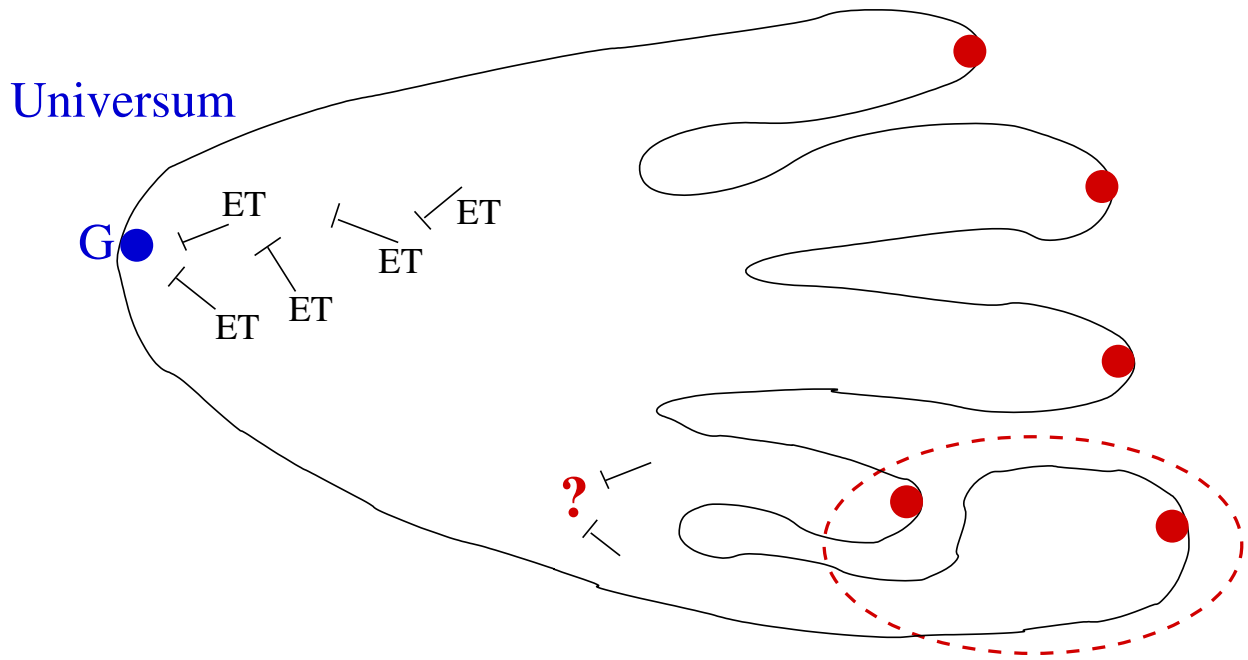
...Konfluenz und globale Optimalität sind für EPTRA verloren!



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 13.1.5

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 12.3.6.1) als G^{opt}_{EPRA} und G^{opt}_{EPTA}:

Lemma 13.1.5.1 (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

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 13.2

EPRAA: EPRA/EPRAd-Kombination

Kapitel 13.2.1

EPRA, EPRAd: Grundtransformationen

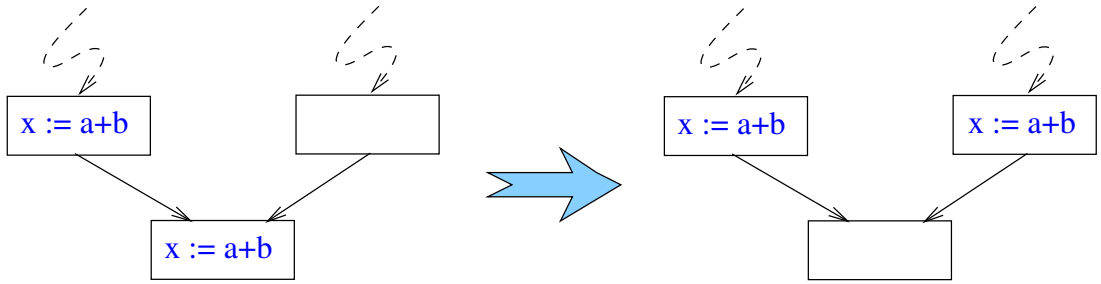
EPRA und EPRAd

...zwei Grundtransformationen zur Elimination redundanten Berechnungsaufwands:

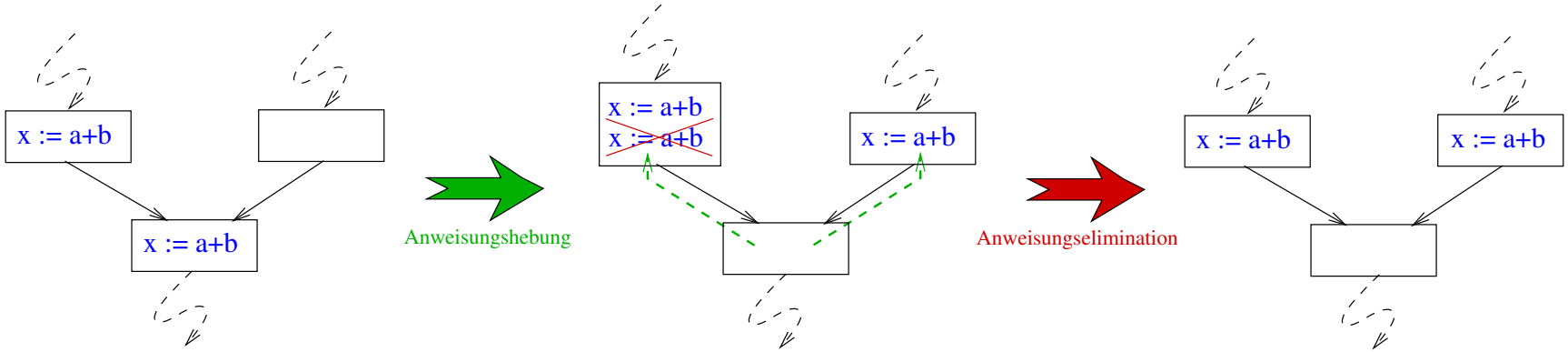
- Elimination partiell redundanter Ausdrücke (EPRAd)
  - $\rightsquigarrow$ Partially Redundant Expression Elimination (PREE)
  - $\rightsquigarrow$ Expression Motion (EM)
- Elimination partiell redundanter Anweisungen (EPRA)
  - $\rightsquigarrow$ Partially Redundant Assignment Elimination (PRAE)
  - $\rightsquigarrow$ Assignment Motion (AM)

Elimination partiell redundanter Anweisungen

Das EPRA-Grundmuster:

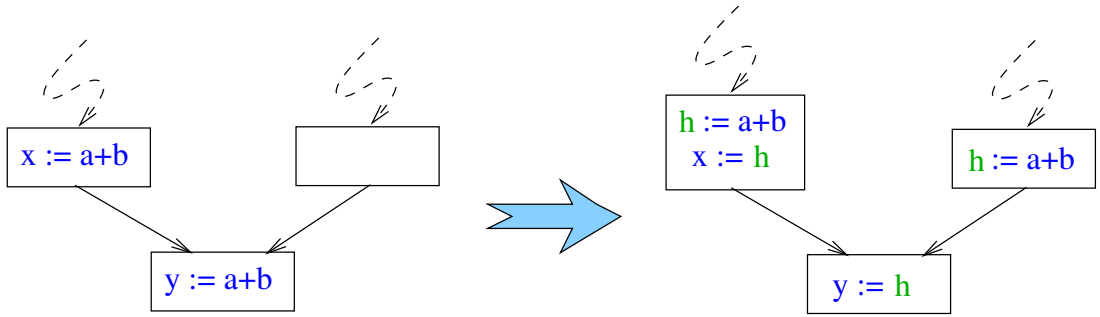


Die konzeptuelle EPRA-Verfahrensidee:

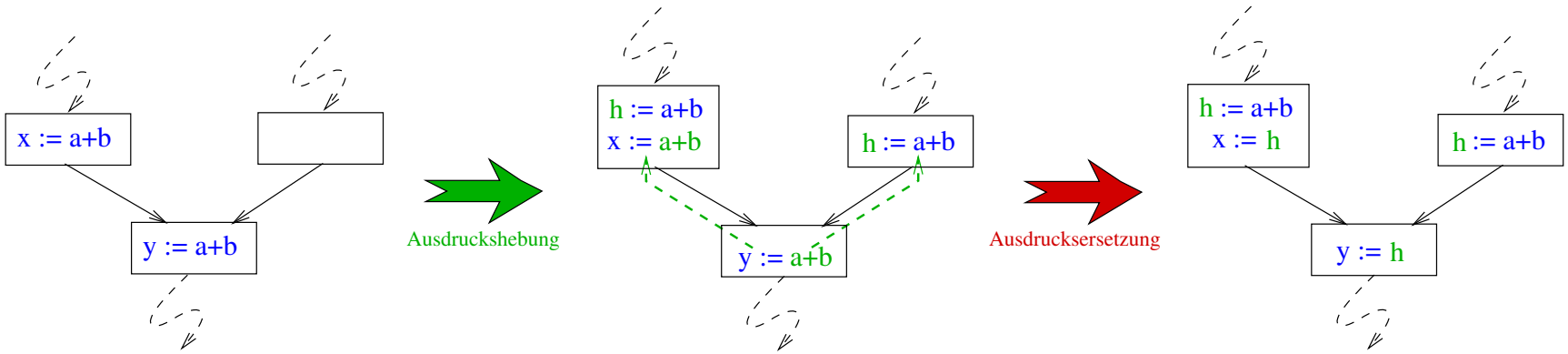


Elimination partiell redundanter Ausdrücke

Das EPRAd-Grundmuster:

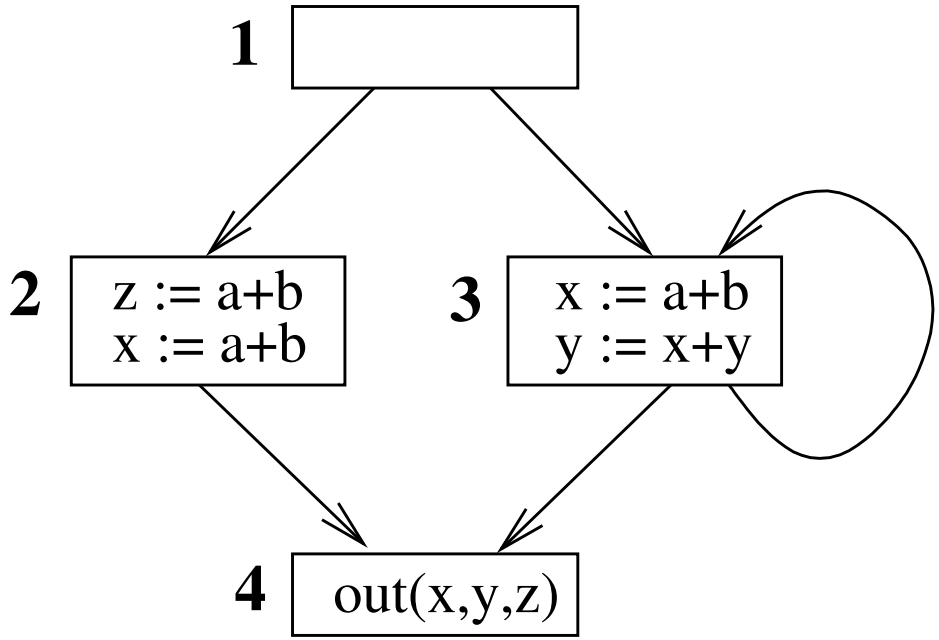


Die konzeptuelle EPRAd-Verfahrensidee:



Ein gemeinsames Beispiel

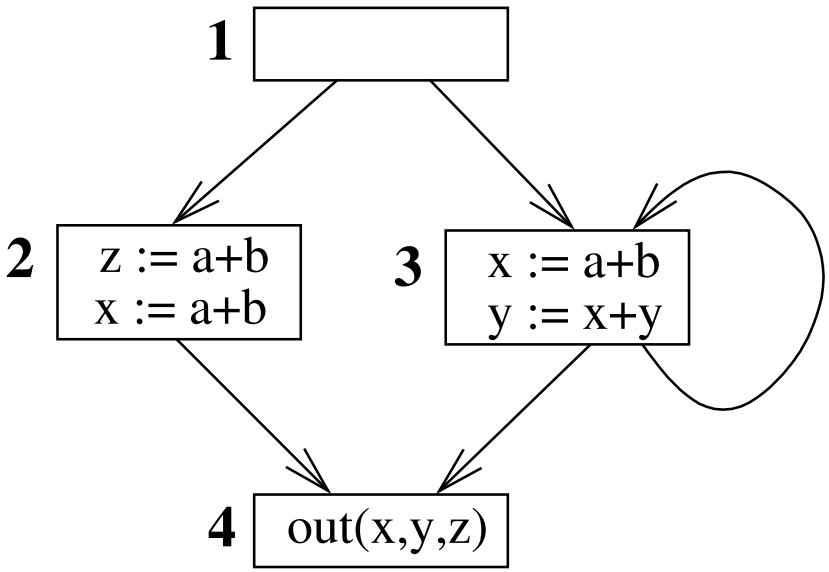
...für die Illustration der verschiedenen Transformationseffekte:



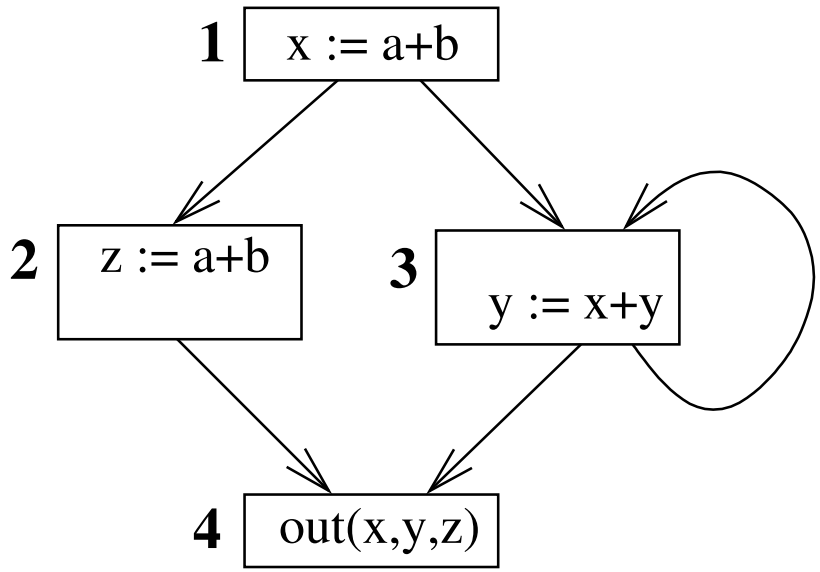
Elimination partiell redundanter Anweisungen

...der EPRA-Effekt auf das gemeinsame Beispiel:

Ausgangsprogramm



Optimiertes Programm

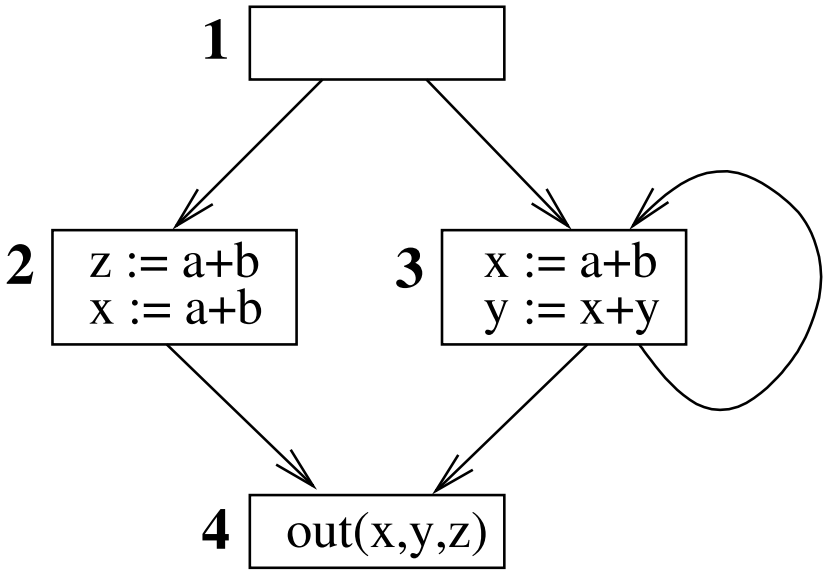


Beachte: Wären die kritischen Kanten $(1,3)$ und $(3,3)$ gespalten, würden maximale EPRA-Transformationen die Anweisung `y := x+y` vom Knoten 3 in die synthetischen Knoten $S_{1,3}$ und $S_{3,3}$ schieben, was bezüglich der Relation 'besser' keinen Unterschied zum obigen optimierten Programm machte.

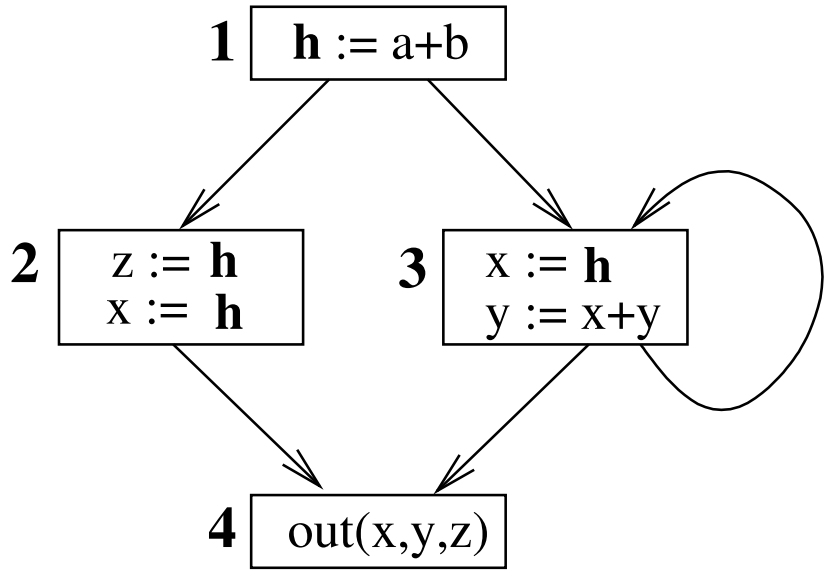
Elimination partiell redundanter Ausdrücke

...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 $= \mathrm{AdH}_{max} * \mathrm{ETRAd}_{max} * (\mathrm{AdS}_{max})^k,\ k \in \{0, 1\}.$

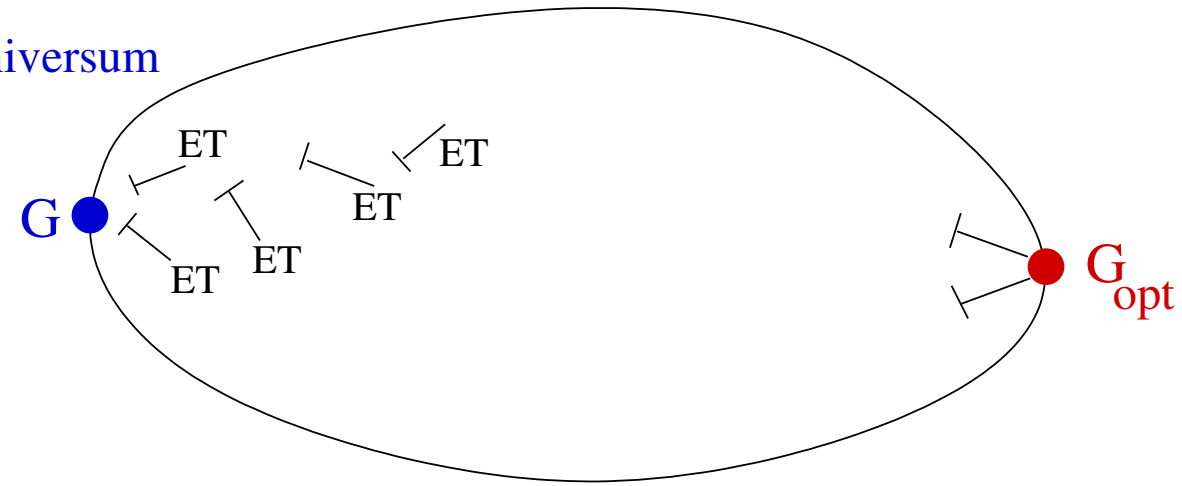
Beachte: EPRAd ist anders als EPRA frei von Effekten zweiter Ordnung: Eine maximale Ausdruckshebung (oder Vorziehen) gefolgt von einer anschließenden einmaligen Elimination total redundanter Ausdrücke und einer optionalen berechnungsneutralen maximalen Zurückschiebung liefert deshalb bereits die bestmögliche Verbesserung und optional geringste Zahl an Hilfsvariableninitialisierungen und -lebenszeiten.

Globale Optimalität von EPRA

...für EPRA gilt (s. Kapitel 12.4):

Theorem 13.2.1.1 (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 13.2.1.2 (Globale Berechnungsoptimalität)

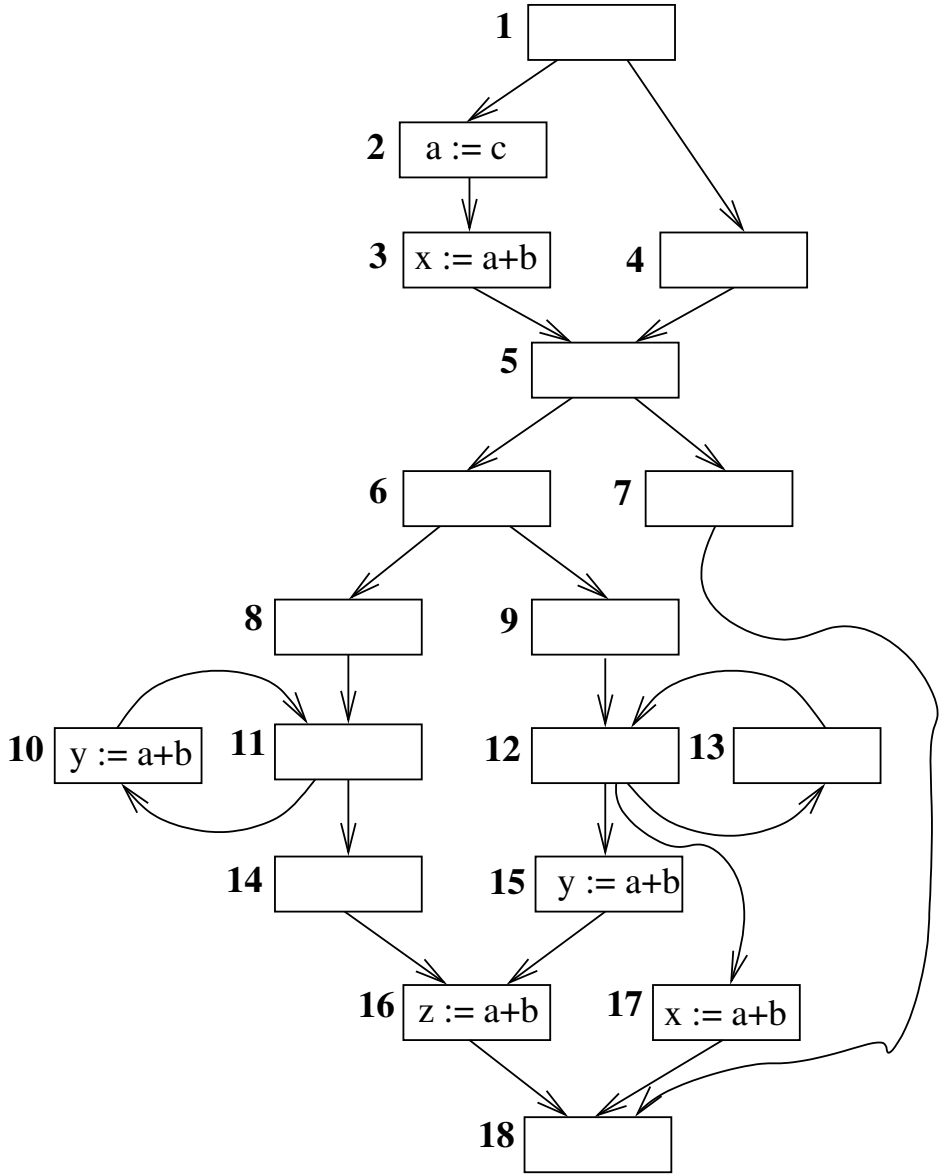
Initialisieren der Hilfsvariablen an den frühest möglichen sicheren Programmpunkten führt zu global besten berechnungsoptimalen Programmen.

Theorem 13.2.1.3 (Globale Lebenszeitoptimalität)

Initialisieren der Hilfsvariablen an den spätest möglichen berechnungsoptimalen Programmpunkten führt zu global besten berechnungs- und hilfsvariablenlebenszeitoptimalen Programmen.

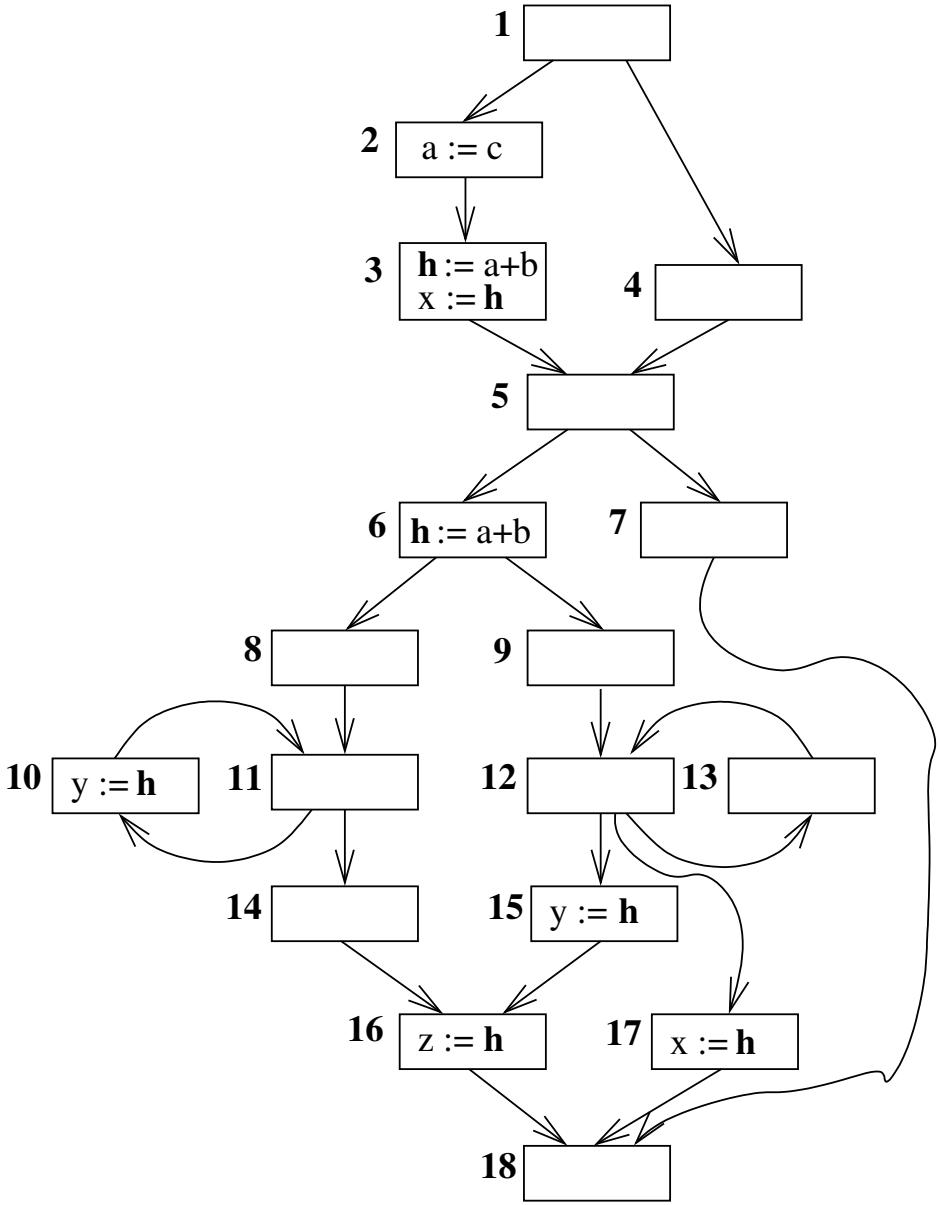
Veranschaul. v. Theorem 13.2.1.2 u. 13.2.1.3

...anhand eines Beispiels:



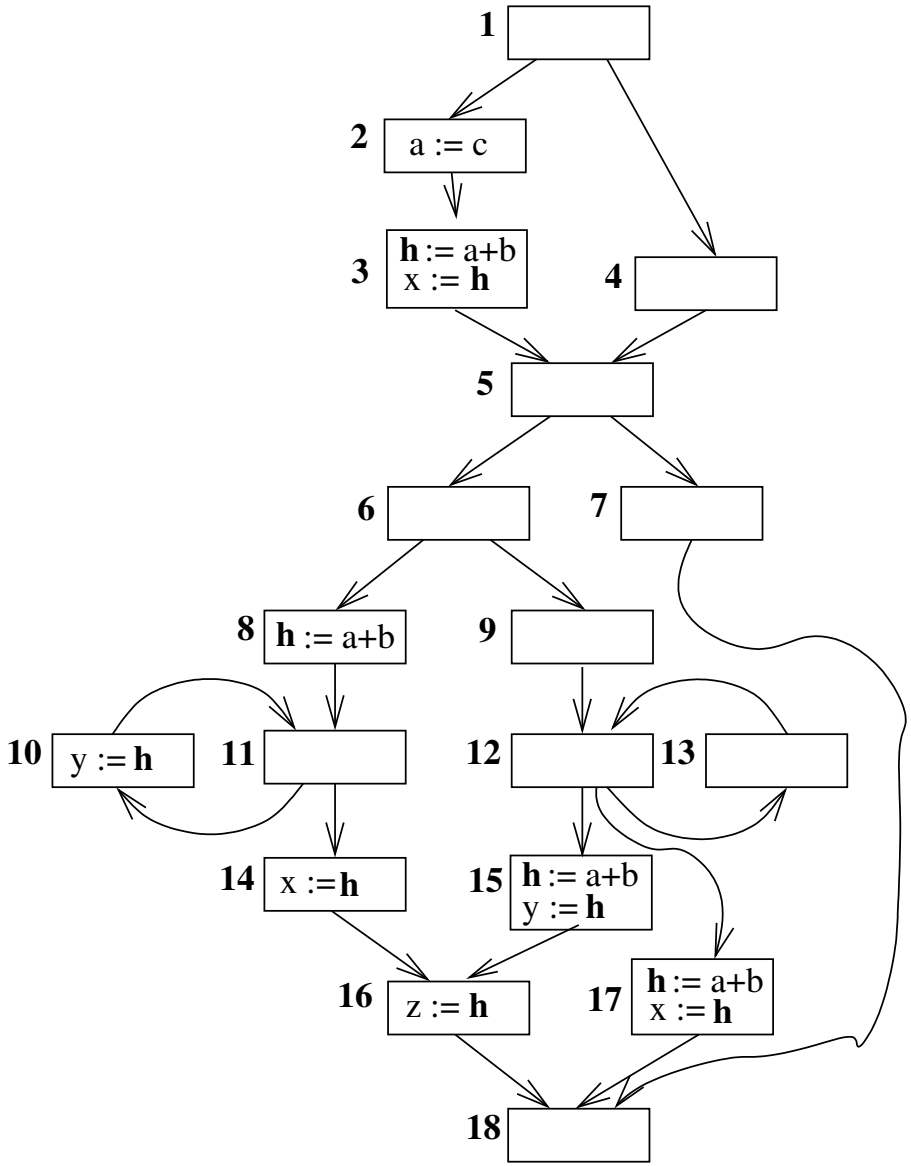
Th. 13.2.1.2: Frühestmögl. sichere Platzierung

...ist global berechnungsoptimal:



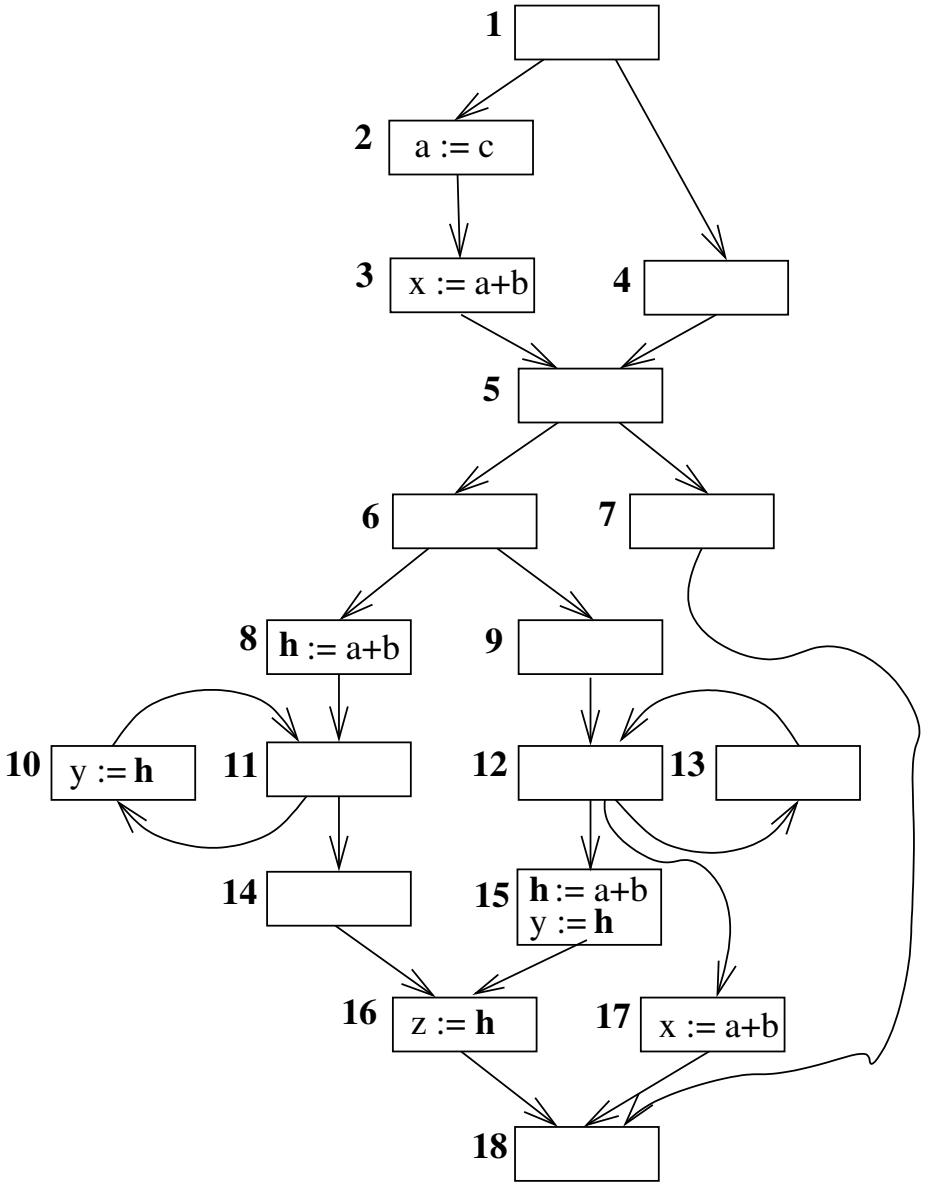
Th. 13.2.1.3: Spätestm. berechn.-opt. Platz.

...ist global berechnungs- und hilfsvariablenlebenszeitoptimal:



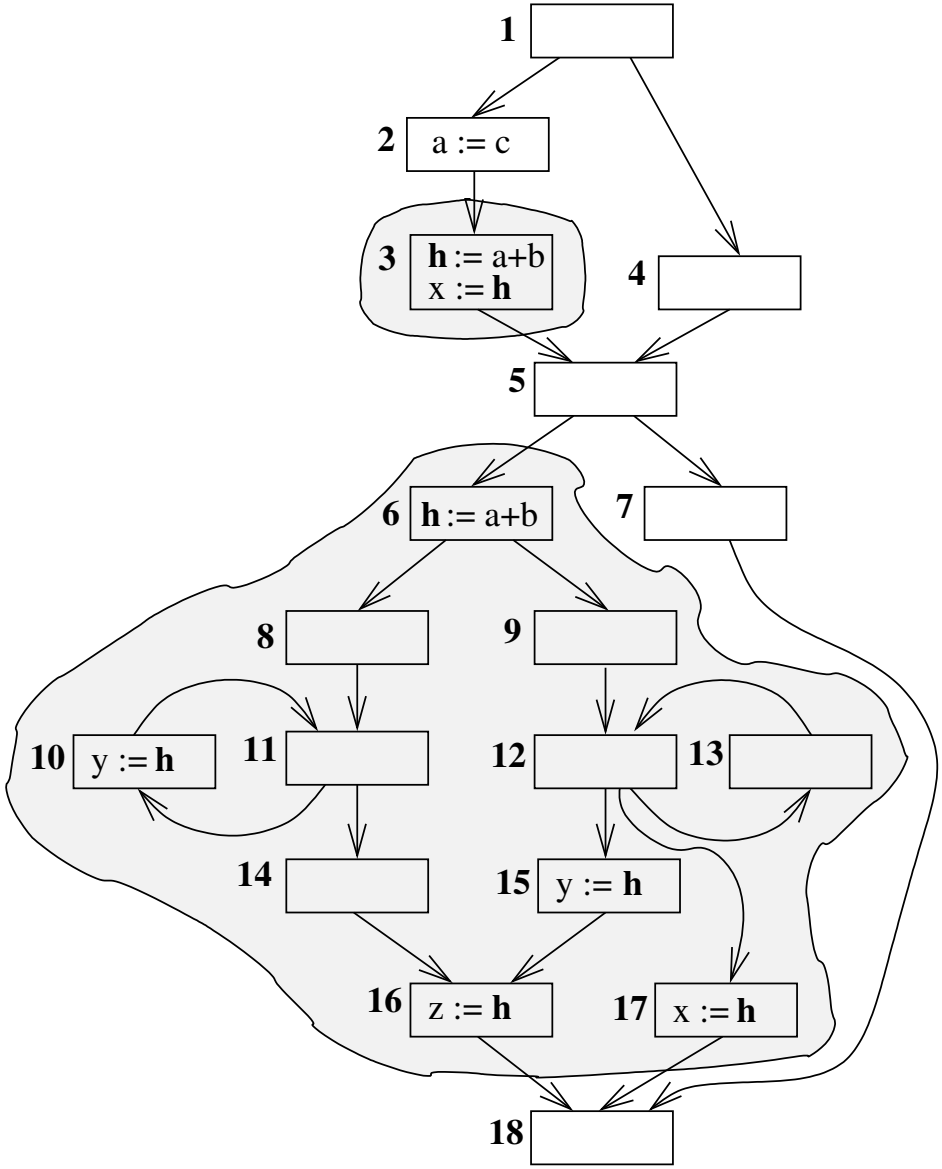
Th. 13.2.1.3: Spätestm. berechn.-opt. Platz.

...nach abschließendem 'Aufräumen':



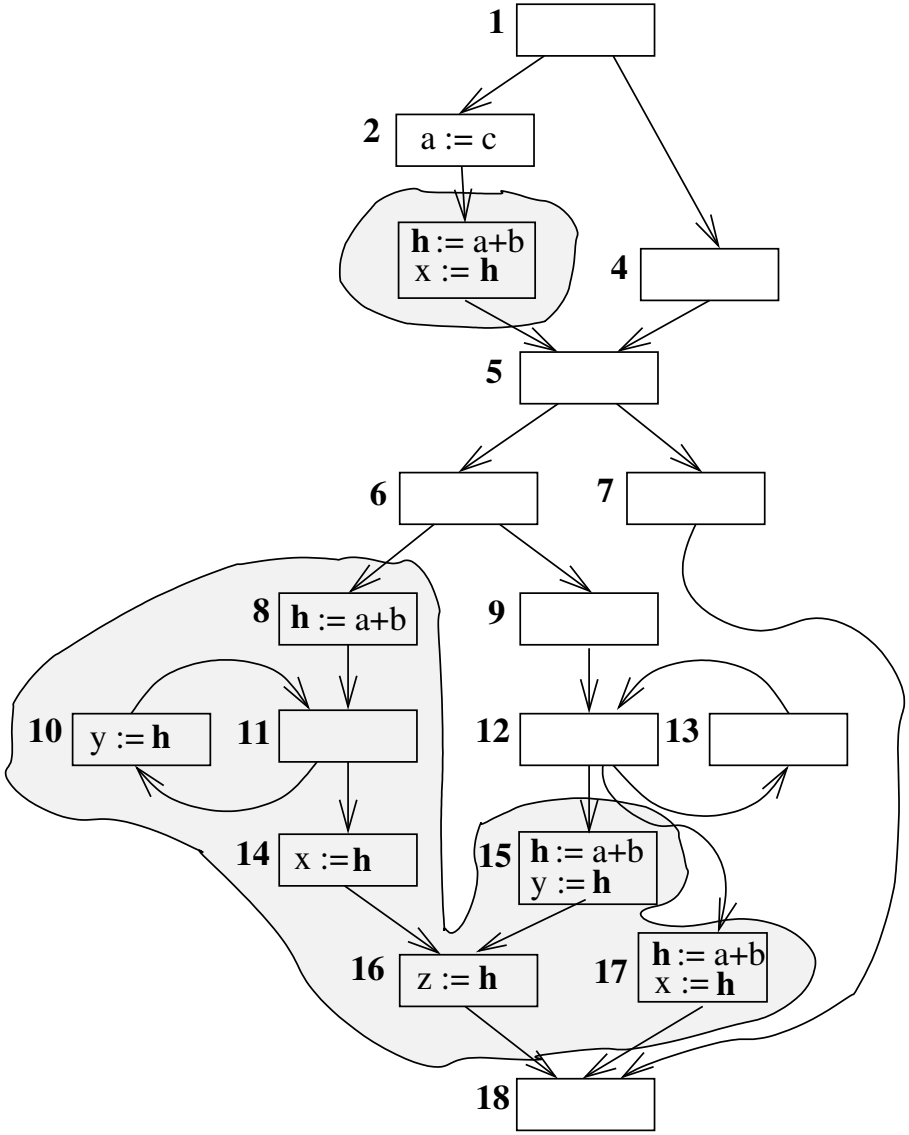
Frühestmögliche sichere Platzierung

...berechnungsoptimal, aber max. Hilfsvariablenlebenszeiten.



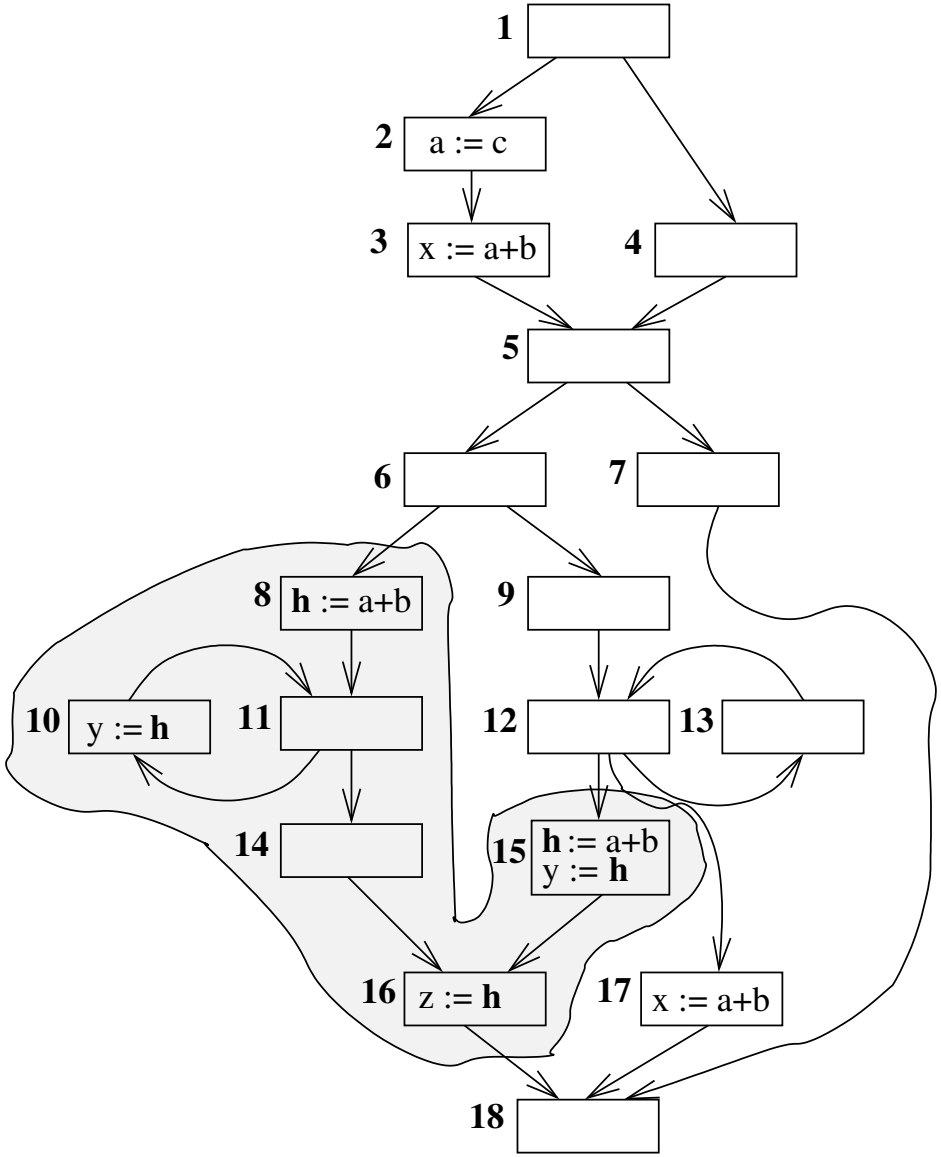
Spätestmögliche berechn.-optimale Platzierung

...berechnungsoptimal, aber min. Hilfsvariablenlebenszeiten.



Spätestmögliche berechn.-optimale Platzierung

...nach ‘Aufräumen’: min. Hilfsvariableninitialisierungen und -lebenszeiten.



Kapitel 13.2.2

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:

- Präprozess
  Ersetze jedes Vorkommen einer Anweisung `x := t` durch die Anweisungssequenz $\texttt{h}_t$ `:= t;` `x :=` $\texttt{h}_t$.
- Hauptprozess
  Wende die Transformationen
  - Heben von Anweisungen (AH)
    $\rightsquigarrow$ Assignment Hoisting
  - Eliminieren (total) redundanter Anweisungen (ERA)
    $\rightsquigarrow$ (Totally) Redundant Assignment Elimination

  wiederholt so lange an bis Stabilität eintritt.
- 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:

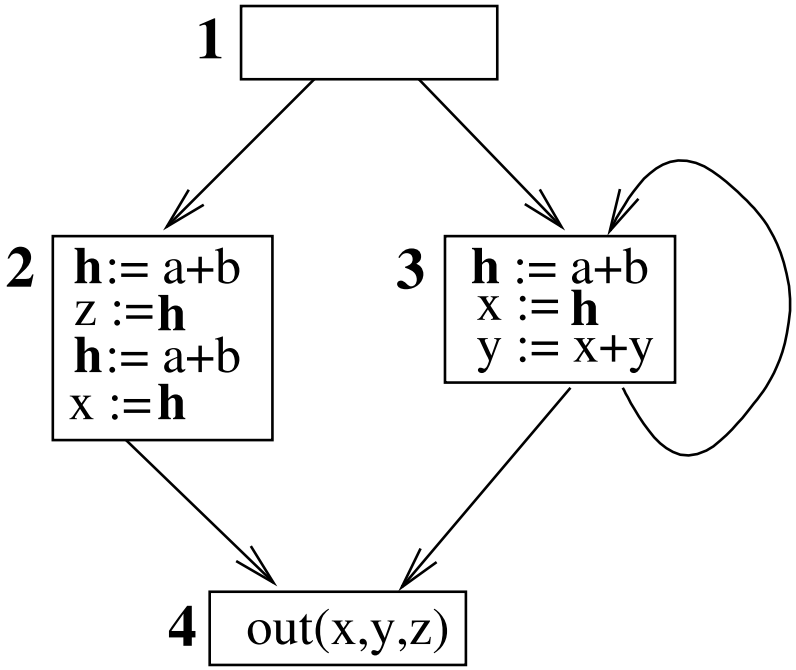
- 'Das Ganze ist mehr als die Summe seiner Teile':

$$\text{EPRAA} > \text{EPRA} + \text{EPRAd}$$

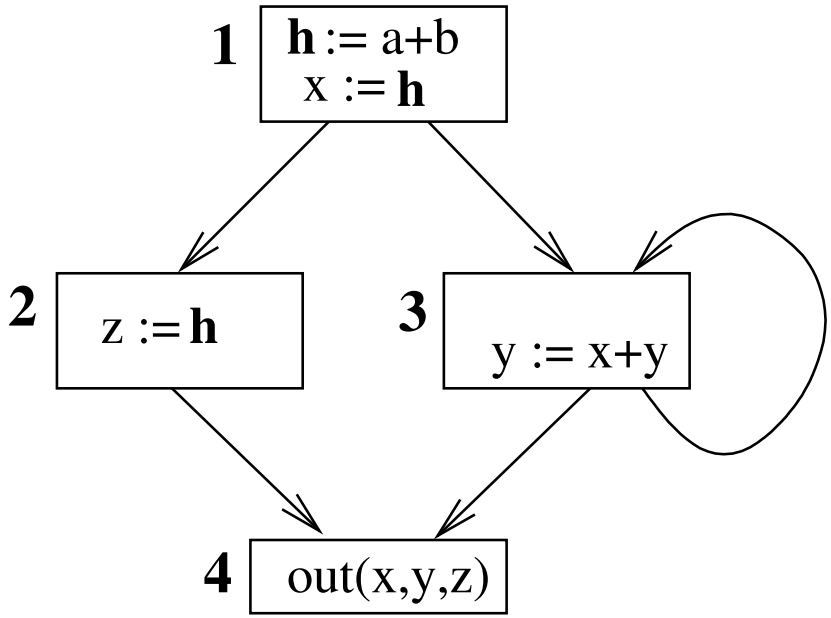
Einheitl. Elimination part. red. Ausd. u. Anw.

...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)

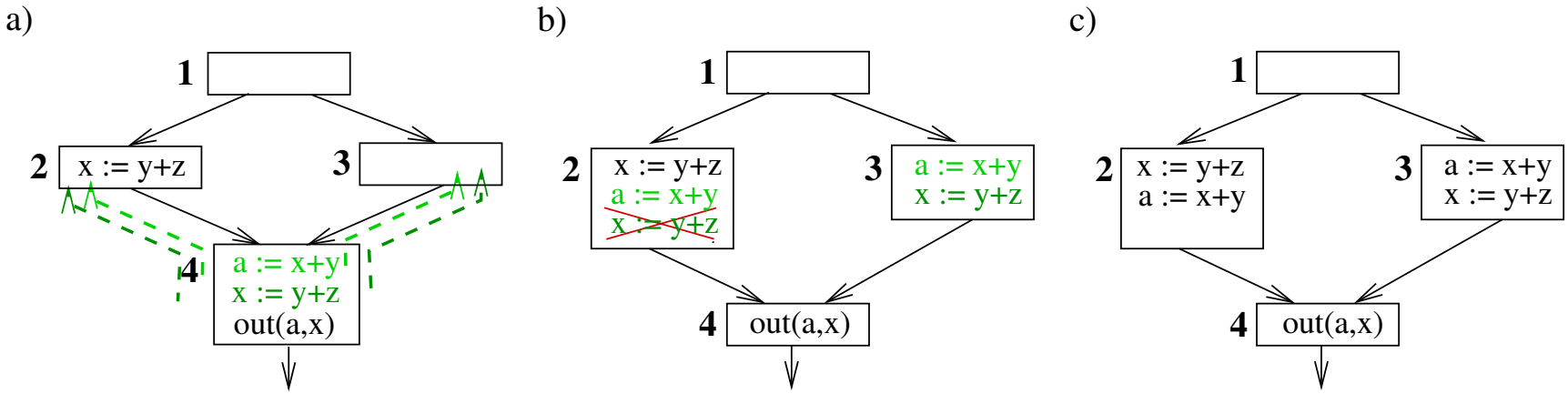
  $\rightsquigarrow$ Elimination-Elimination effects (EE)

Beispiel für einen Hebungs-Eliminations-Effekt

Ausgangsprogramm

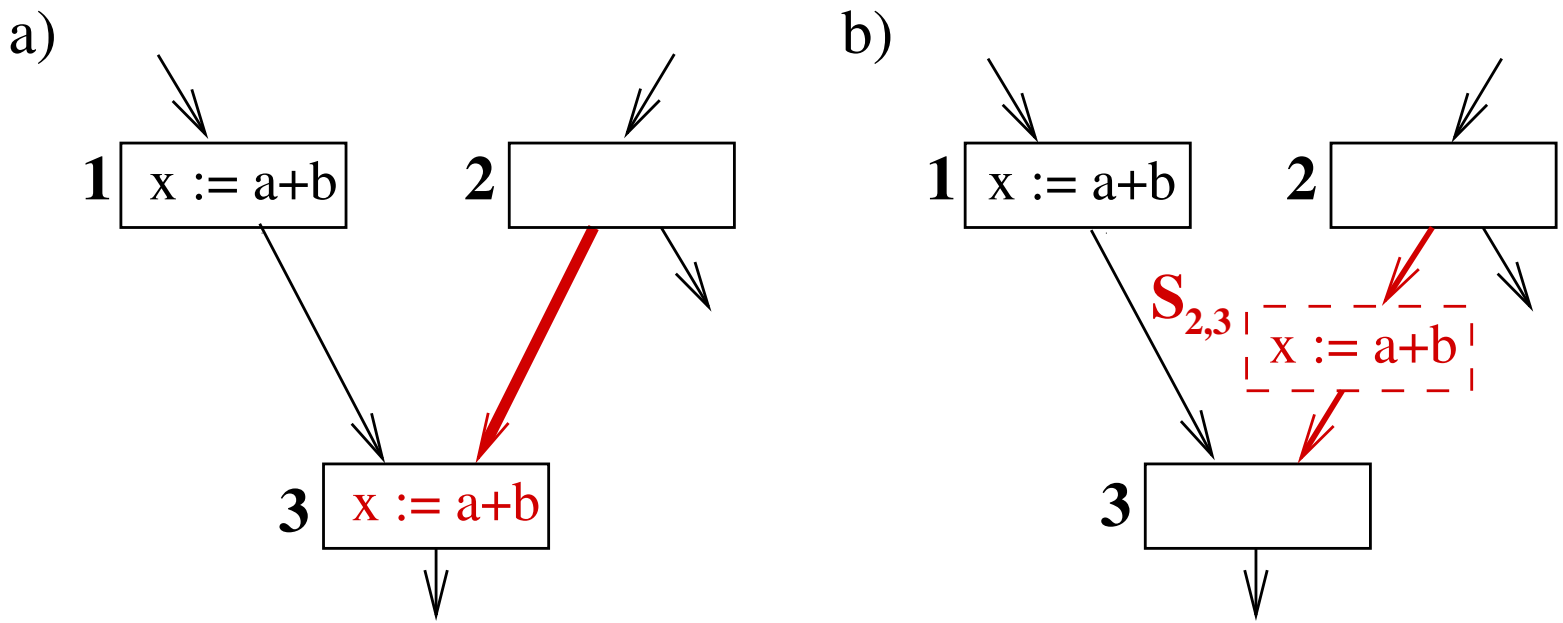
Anweisungshebung (Effekt 1. Ord.)

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:



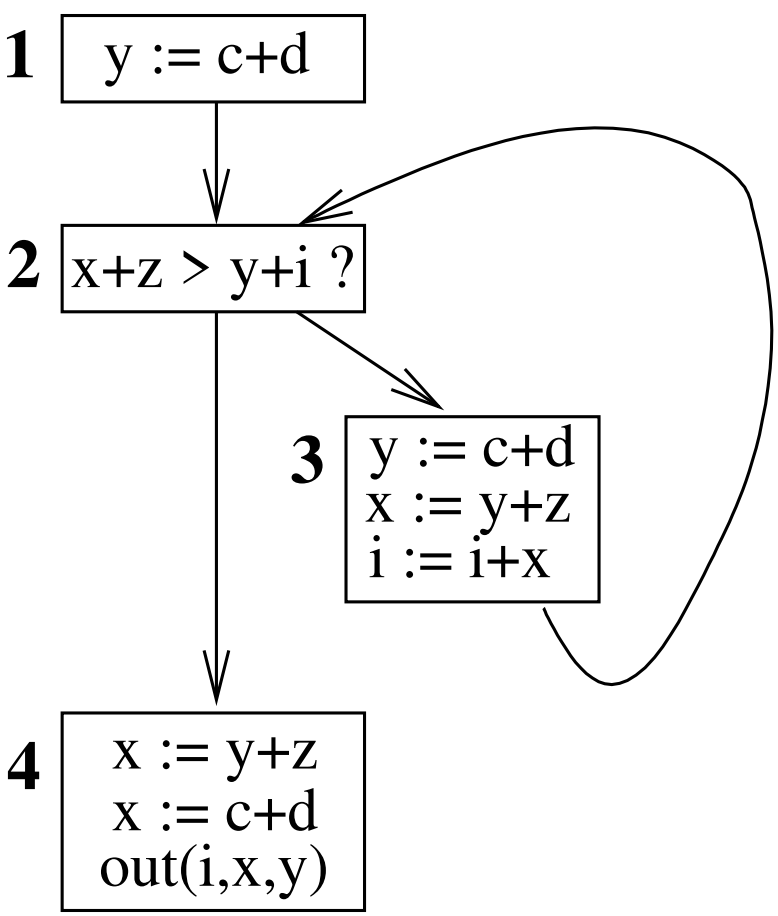
Kapitel 13.2.3

EPRAA: Beispiel

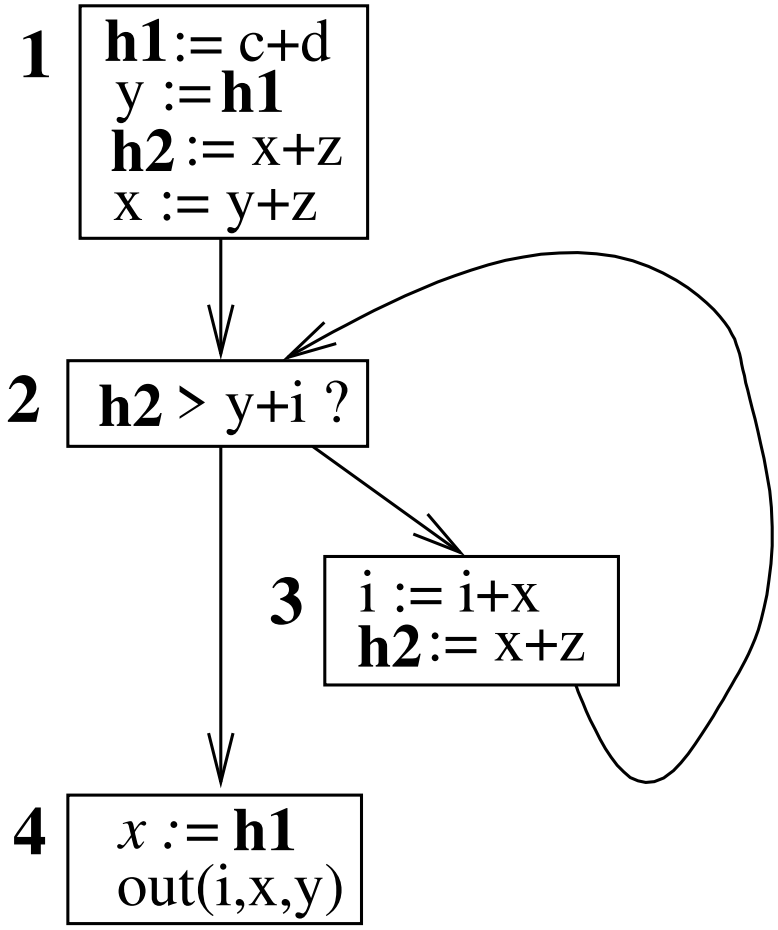
Die EPRAA-Transformation

...illustriert anhand eines größeren Beispiels.

Ausgangsprogramm

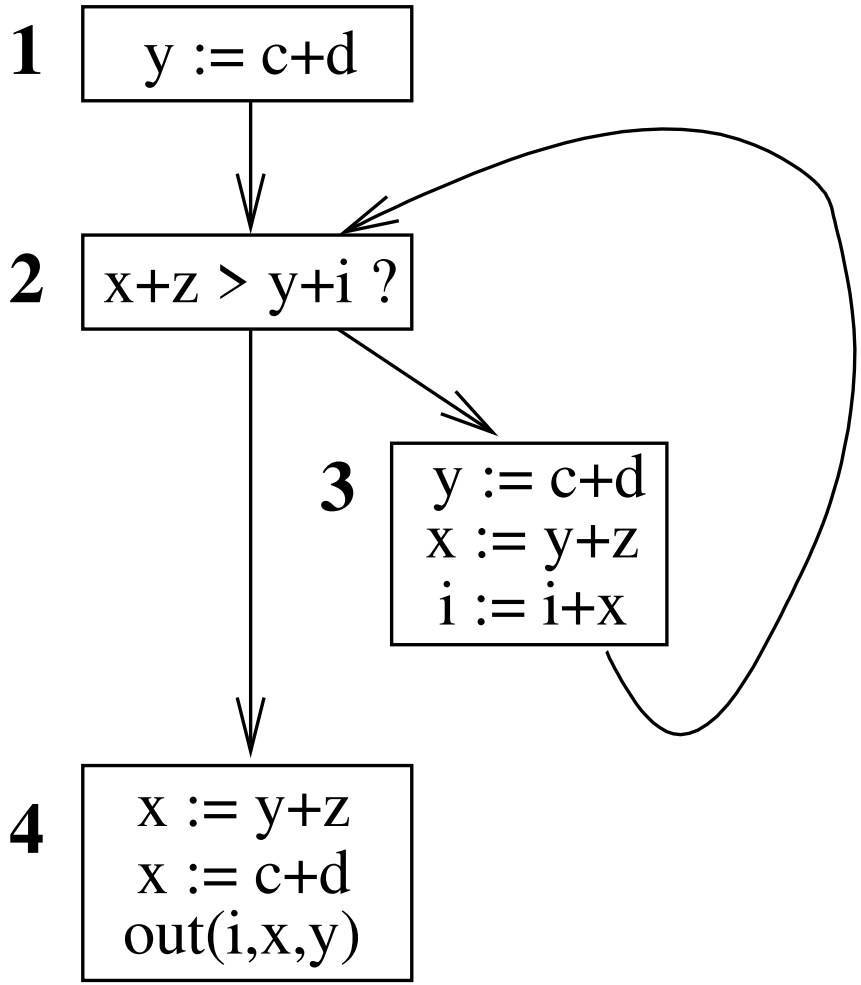


Optimiertes Programm



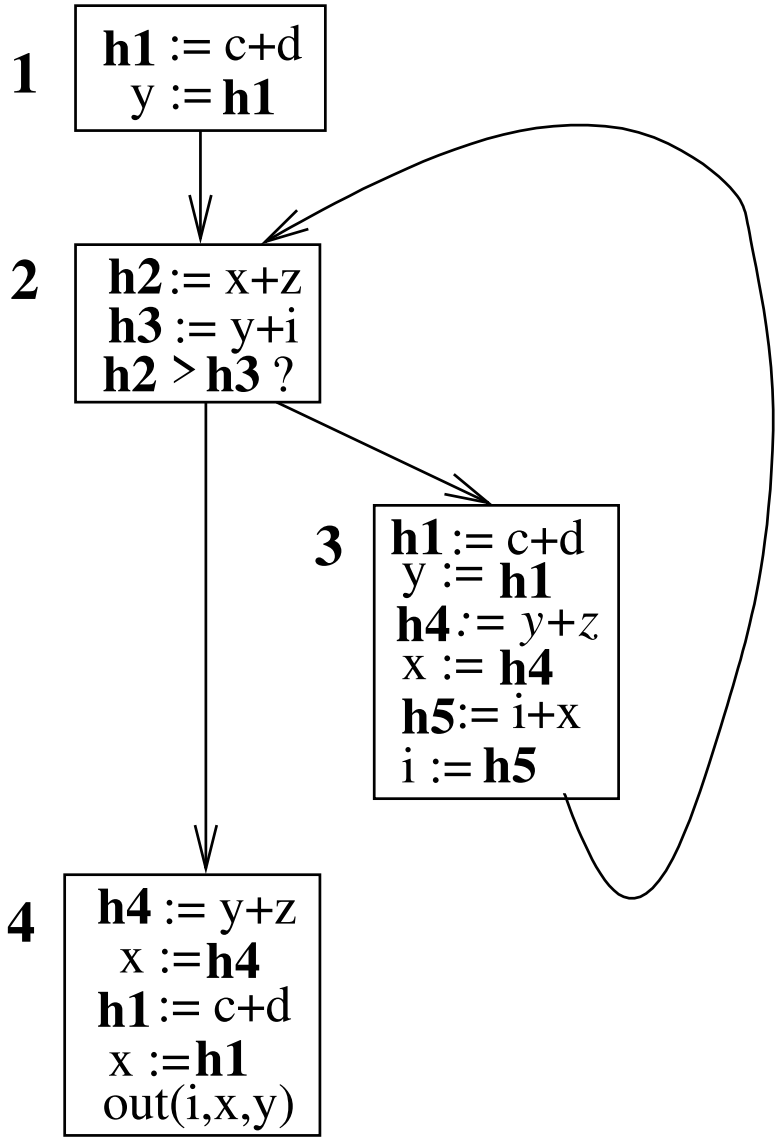
Die EPRAA-Transformation im Detail (1)

Ausgangsprogramm



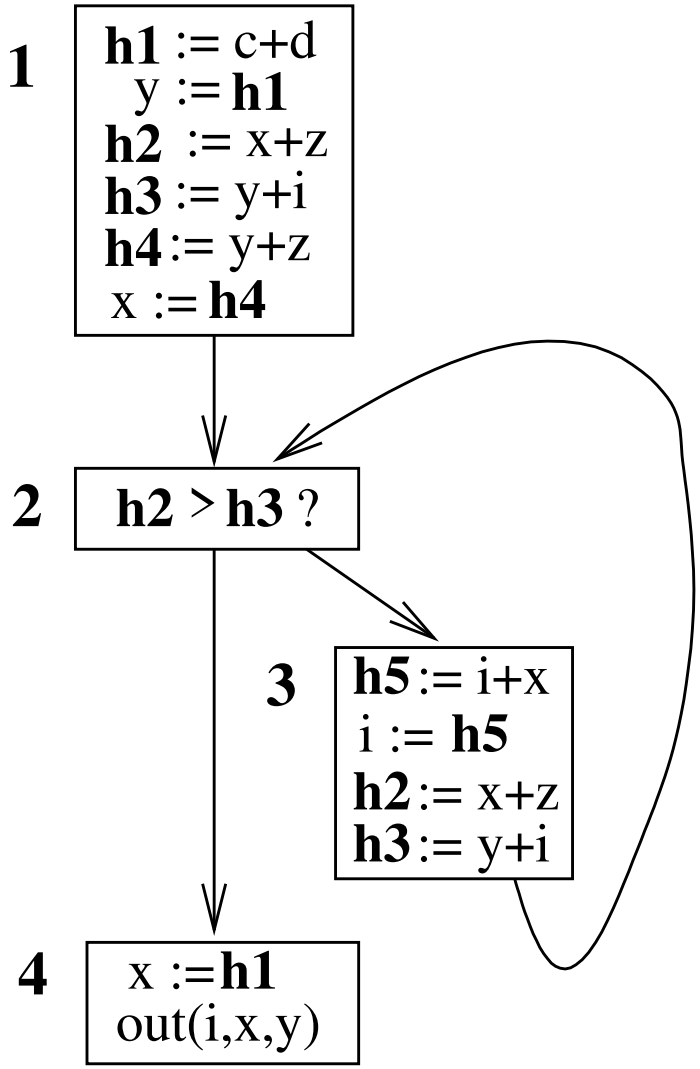
Die EPRAA-Transformation im Detail (2)

Programm nach Präprozess



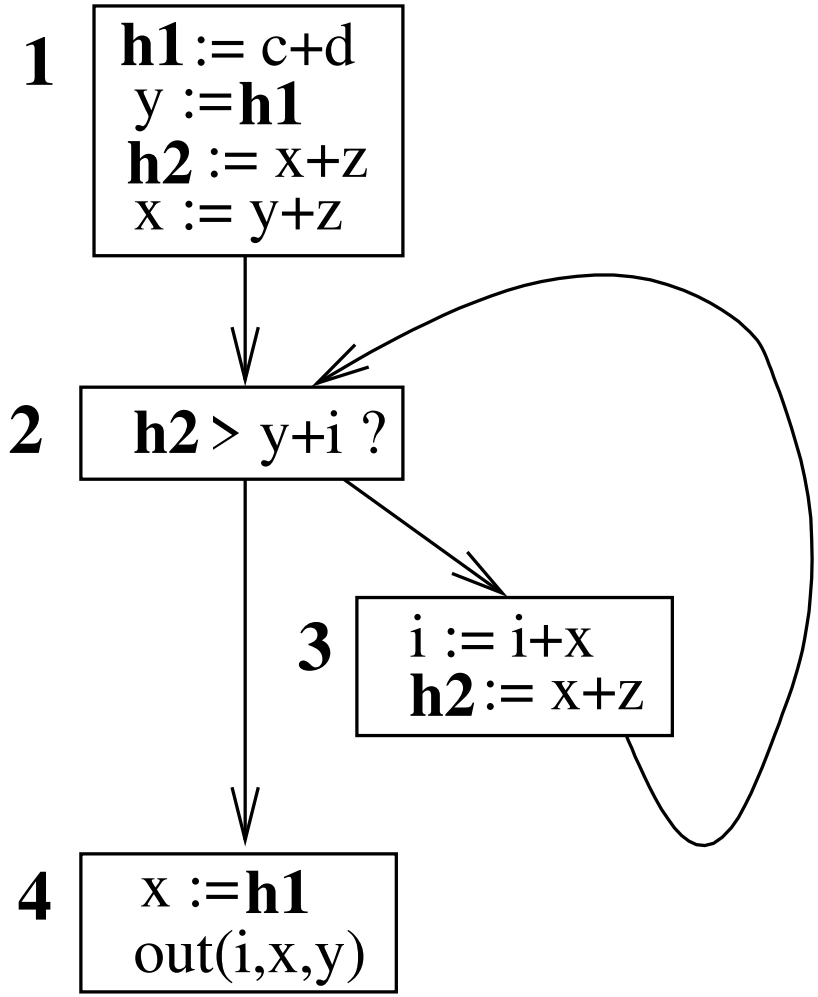
Die EPRAA-Transformation im Detail (3)

Programm nach Hauptprozess



Die EPRAA-Transformation im Detail (4)

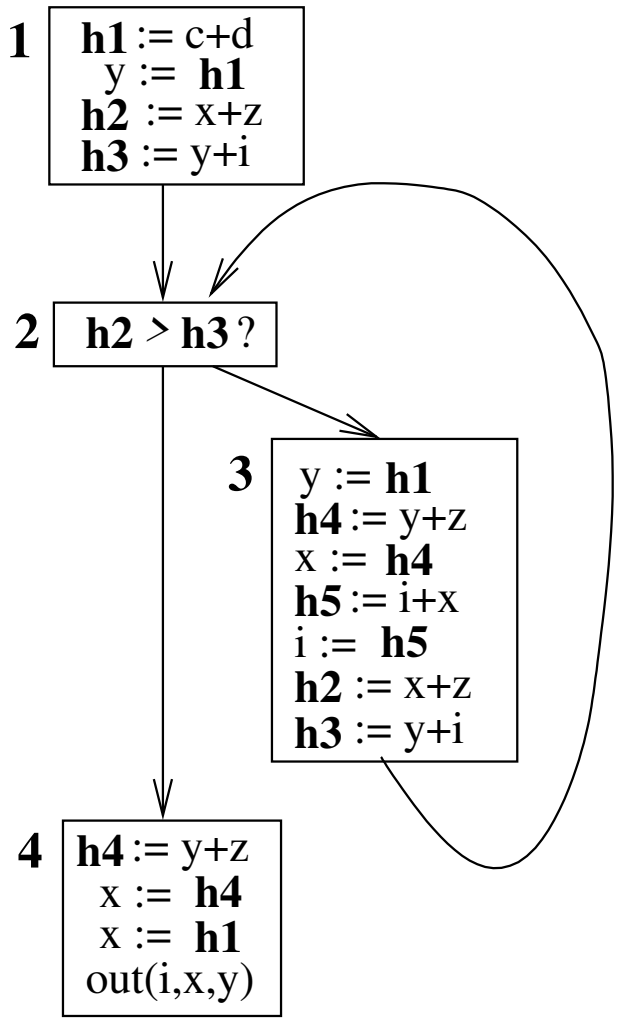
Programm nach Postprozess - das EPRAA-optimierte Prg.



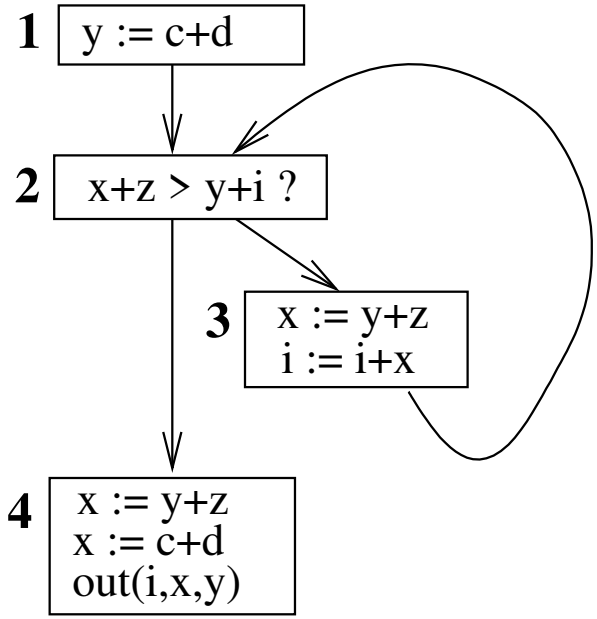
Zum Vergleich: Die schwächeren Effekte

...der EPRAd- und EPRA-Transformationen:

EPRAd-Effekt



EPRA-Effekt



Kapitel 13.2.4

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 $\tau \in T^G_{EPRAA}$ auf G entstehende Programm.

EPRAA: Globale, lokale Optimalität

Sei $\tau_{max} \in \mathsf{T}^{G}_{EPRAA}$ eine maximale EPRAA-Transformation.

Theorem 13.2.4.2 (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 13.2.4.3 (Lokale Anweisungsoptimalität)

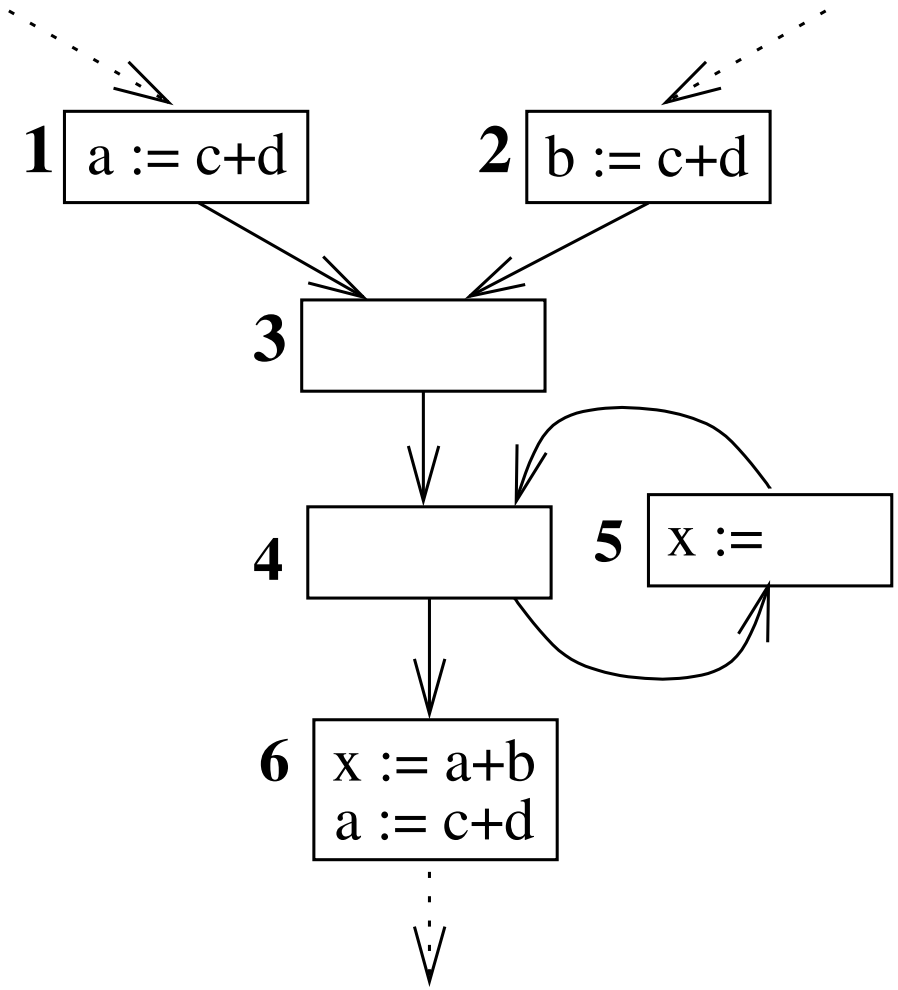
$G_{\tau_{max}}$ ist lokal anweisungsoptimal in $\mathcal{U}^{G}_{EPRAA}$, d.h. es ist nicht möglich, die Zahl der von $G_{\tau_{max}}$ zur Laufzeit ausgeführten Anweisungen durch weitere EPRAA-Elementartransformationen zu verringern.

Theorem 13.2.4.4 (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?

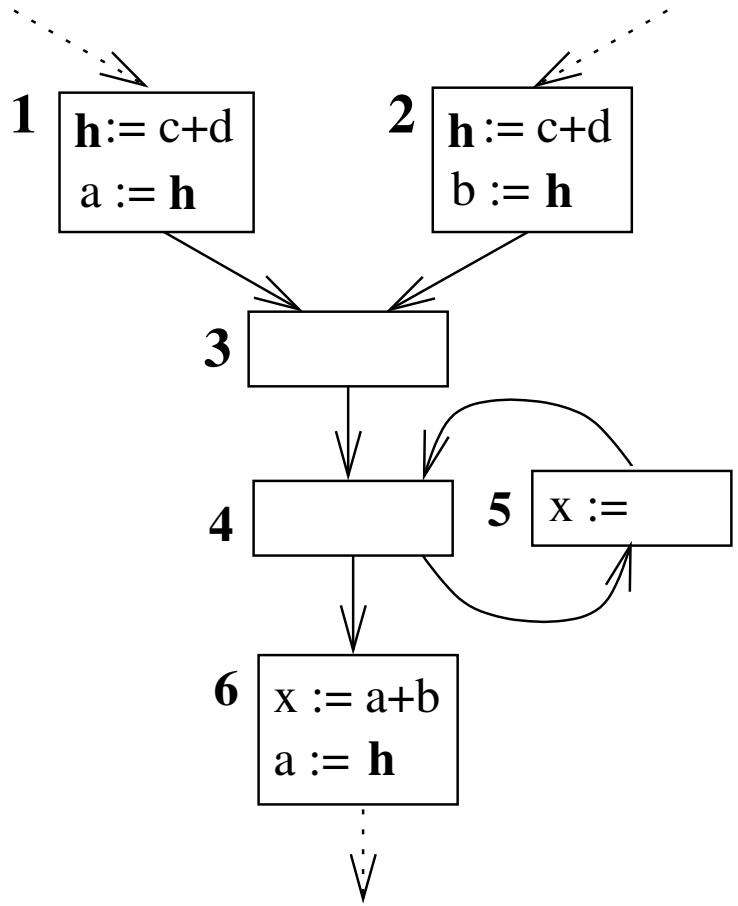
...betrachte folgendes Programm:



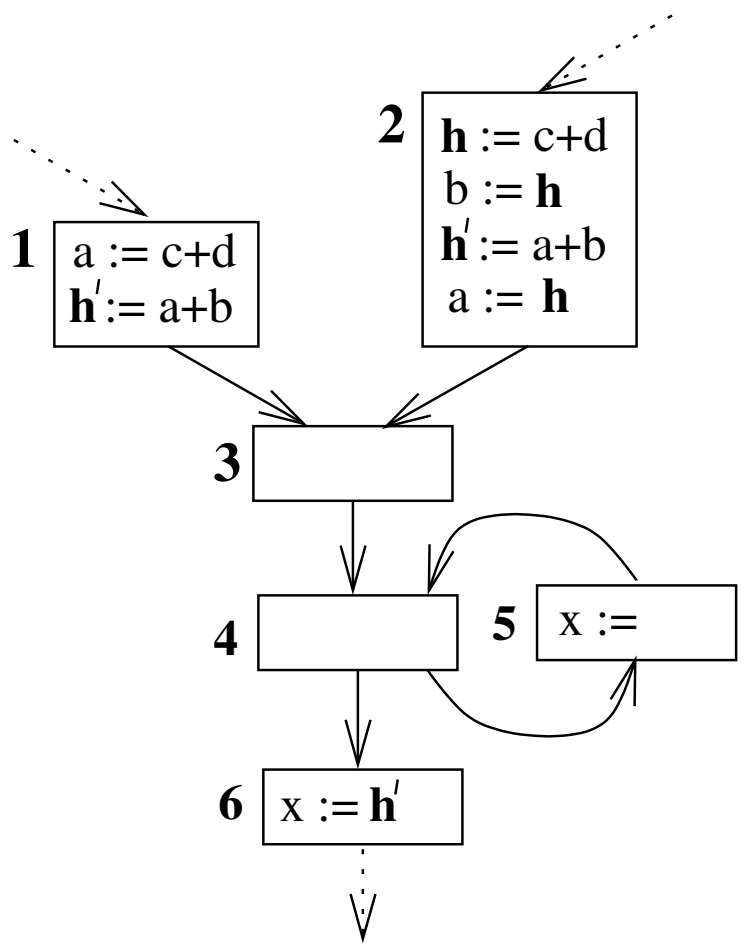
Darum nur lokale Anw./Hilfsv.-Optimalität!

...und folgende zwei unvergleichbare Transformationsresultate:

a)



b)



⇒ Lokale Anw./Hilfsv.-Optimalität ist das bestmögliche!

Kapitel 13.3

Ohne Beschränkung der Allgemeinheit

Kapitel 13.3.1

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. 12.2.3).
- Gibt es mehrere Endknoten, kann durch Einfügen eines künstlichen Endknotens und entsprechender Kanten dies erreicht werden (s. Kap. 12.2.3, indirekt behandelt).
- Anweisungen liegen im Format von Drei-Adress-Code vor (s. Kap. 13.3.2).
- Flussgraphen sind wahlfrei knoten- oder kantenbenannt, mit einzelnen Instruktionen oder mit Sequenzen von Instruktionen (sog. Basisblöcken) (s. Kap. 13.3.3 und Anhang B).
- ...

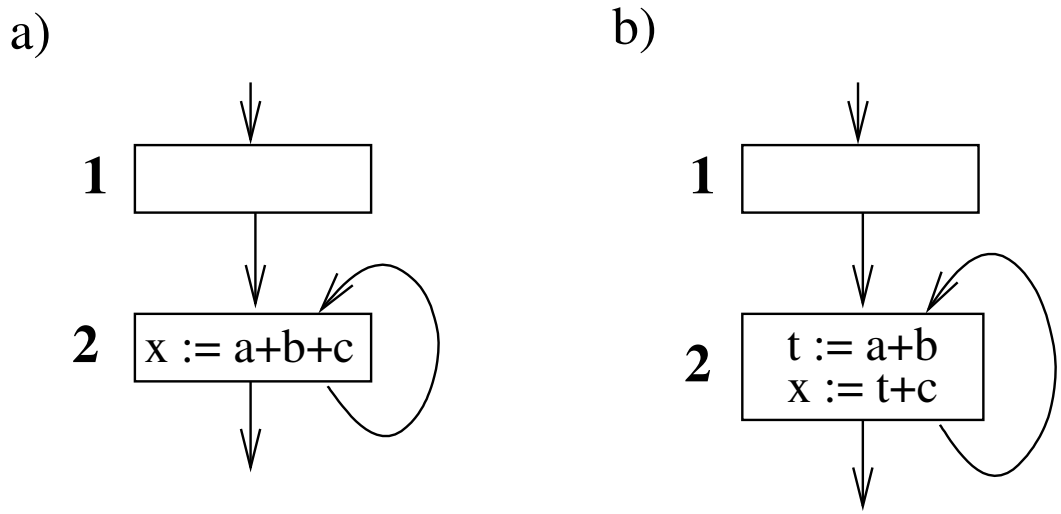
Nicht immer sind solche Annahmen uneingeschränkt unbeschränkend und frei von Nebenwirkungen...

Kapitel 13.3.2

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):

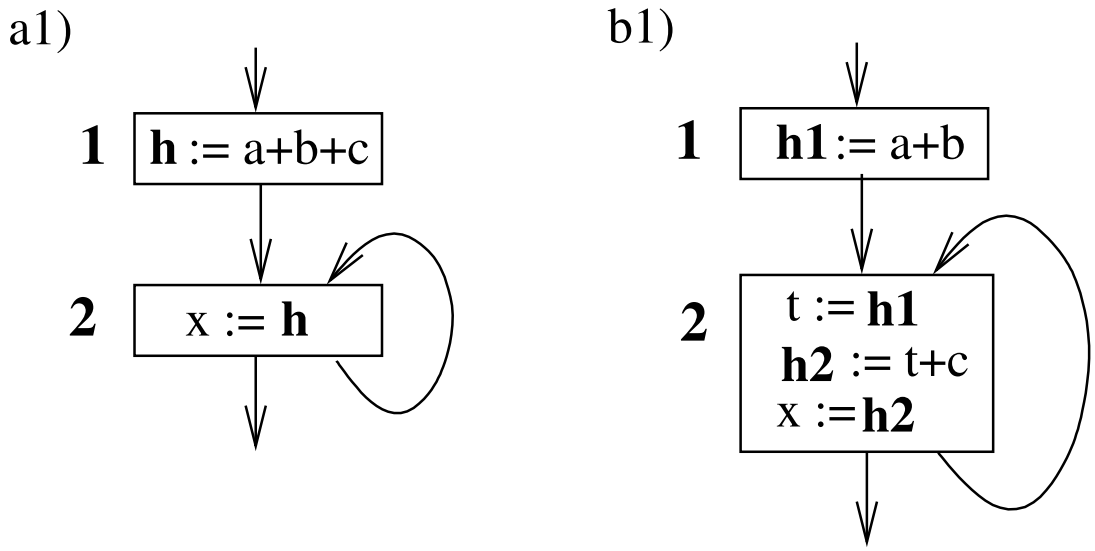


...offenbar ist eine solche Aufspaltung immer möglich;

- 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)

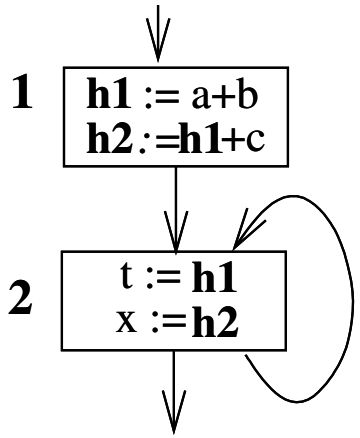
'aufgeräumt' werden kann, woduch Variablen zusammengefasst und in ihrer Zahl reduziert werden.

Frei von Nebenwirkungen oBdA?

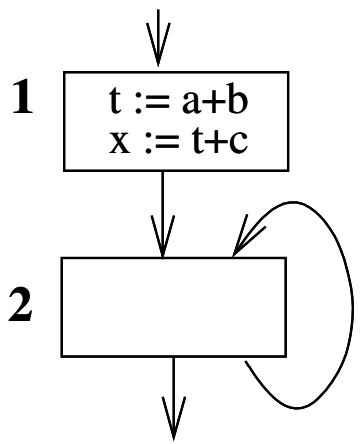
...vergleiche die Wirkungen von

- 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).

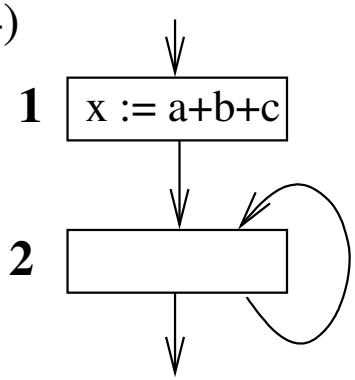
b2)



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 13.3.3

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 16 für Details.

Von Instruktions- zu Basisblockgraphen

...die Analysen der Elementartransformationen für die Berechnung

- toter und redundanter Anweisungen
- der Endpunkte von Anweisungssenkungen und -hebungen

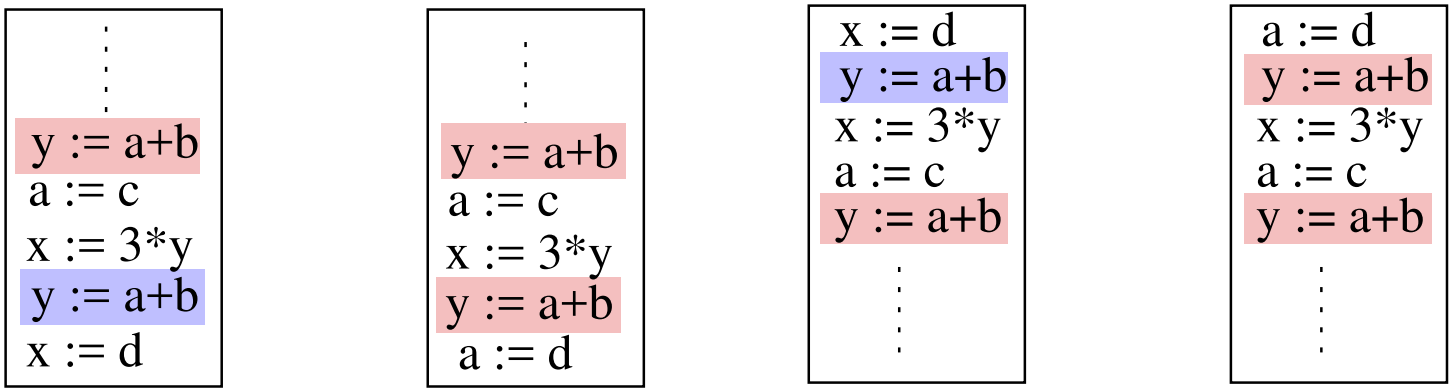
lassen sich in natürlicher Weise von Instruktions- auf Basisblockgraphen ausdehnen; für die Analyse zur Berechnung

- geisterhafter Anweisungen

als sog. nicht-separables Problem gilt das nicht, eine Ausdehnung auf Basisblockgraphen ist nicht möglich (s. Kapitel 13.3 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 12.3.8 bzw. 12.4.7 unverändert für Basisblockgraphen übernommen werden.

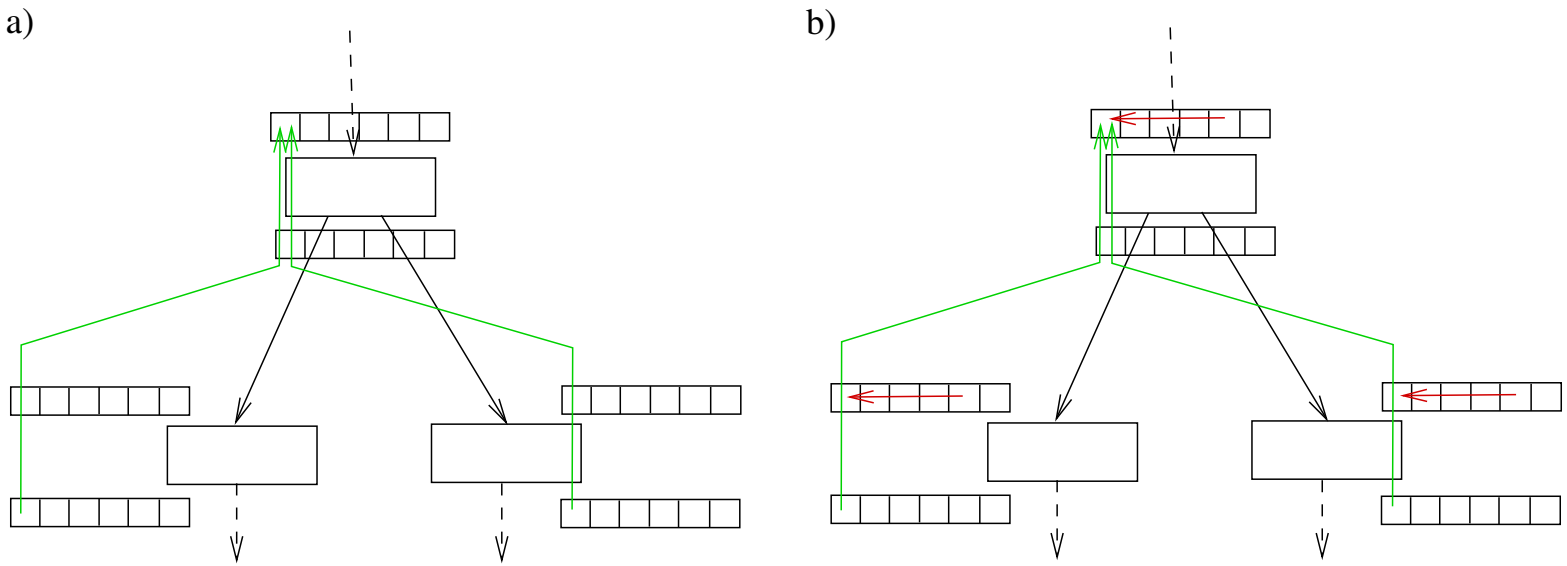
Separable vs. nicht-separable DFA-Probleme

...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 13.4

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 13 (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 13 (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 13 (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 13 (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 14

Konstantenanalyse auf nicht-klassischen 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

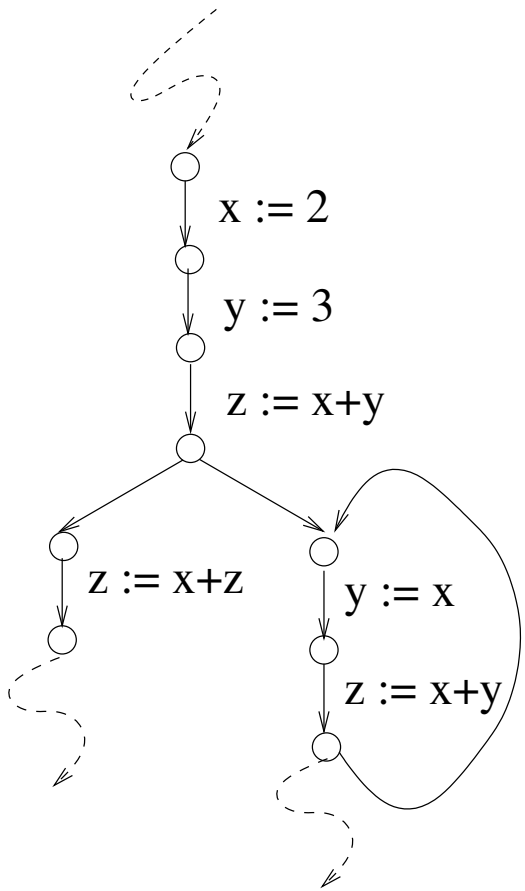
Kapitel 14.1

Motivation

Konstantenanalyse

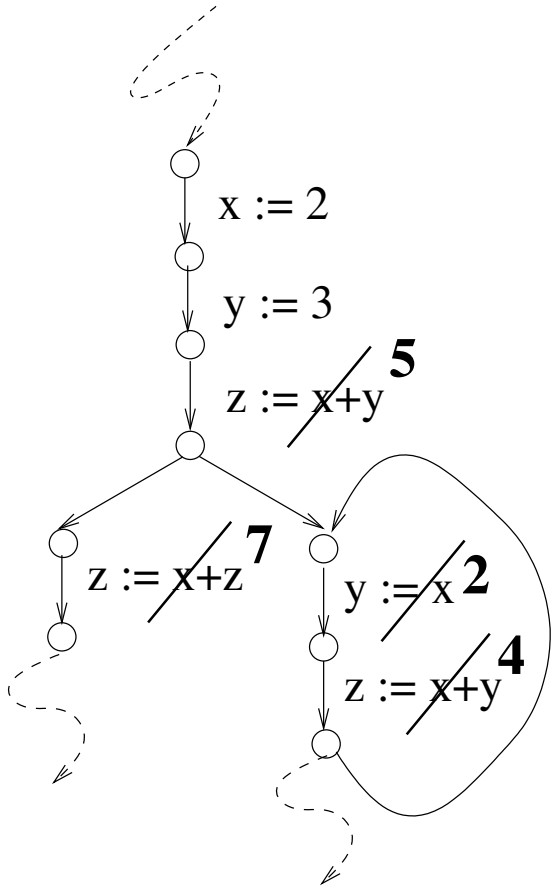
...anhand eines Beispiels für einfache Konstanten:

a)



Original program

b)



After simple constant propagation

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 7.10.2 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)

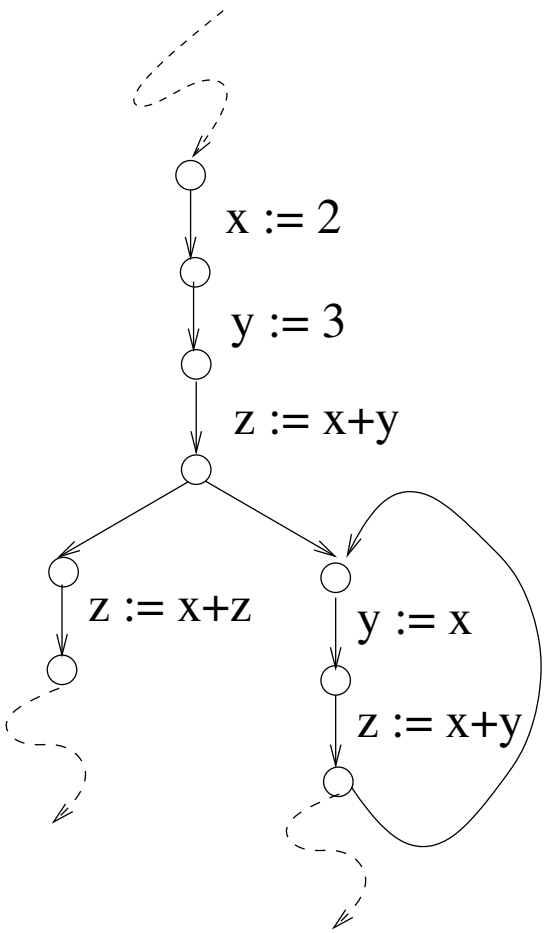
- Anwendungsreichweite
  - Interprozedural
    - Callahan, Cooper, Kennedy, Torczon (SCC'86)
    - Grove, Torczon (PLDI'93)
    - Metzer, Stroud (LOPLAS, 1993)
    - Sagiv, Reps, Horwitz (TAPSOFT'95)
    - Duesterwald, Gupta, Soffa (TOPLAS, 1997)
    - ...
  - Explizit parallel
    - Lee, Midkiff, Padua (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.

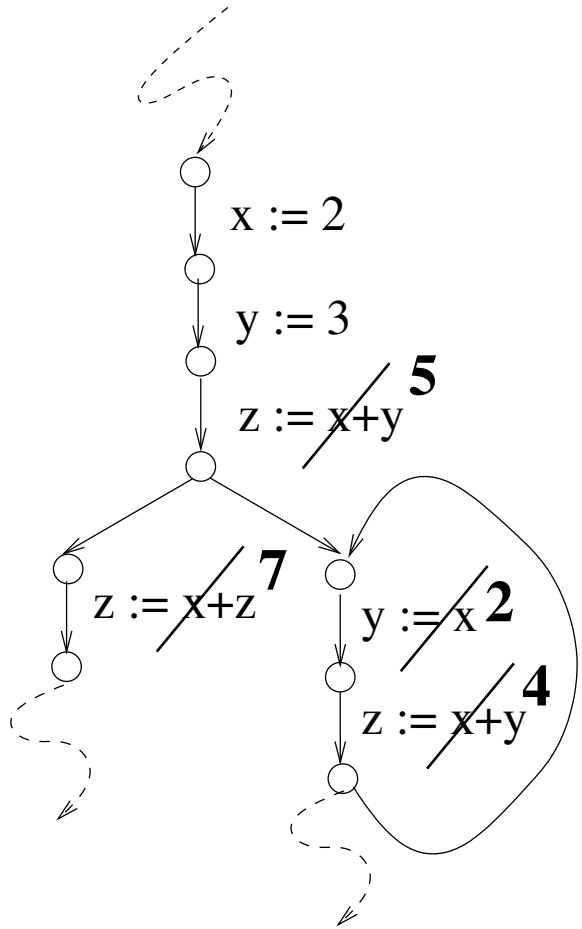
Warum streben nach größerer Ausdruckskraft?

a)



Original program

b)



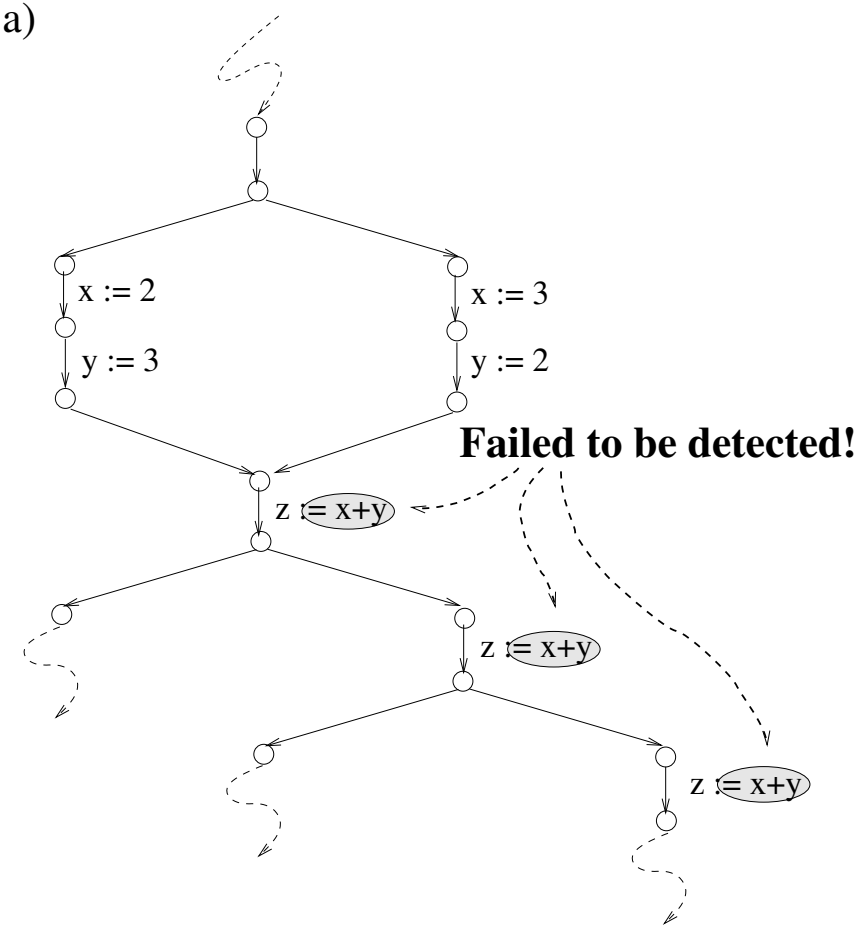
After simple constant propagation

...das Ergebnis der SC-Analyse scheint hier überzeugend.

Der Schein trügt

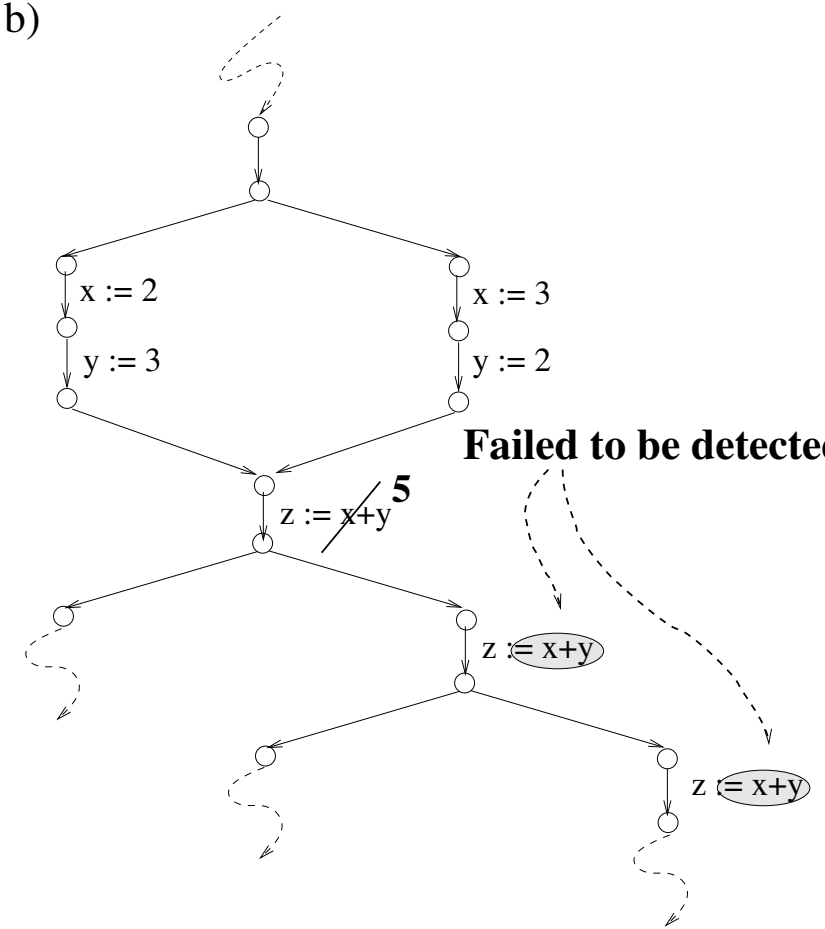
...das Konzept einfacher Konstanten ist tatsächlich schwach:

a)



After simple constant propagation
(Note: No effect at all!)

b)



After simple constant propagation
enriched by the "look–ahead–of–one" heuristics
of Kam and Ullman

Entscheidbarkeitsfragen für Konstantenanalyse

Einerseits: Konstantenanalyse ist

- unentscheidbar: Reif, Lewis (POPL 1977, vgl. Kapitel 7.10.2)

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:

$$\Sigma = \{\sigma \mid \sigma : \mathbf{V} \to \mathbb{Z}_\perp^\top\},\ [\![e]\!]_{sc} : \Sigma \to \Sigma$$

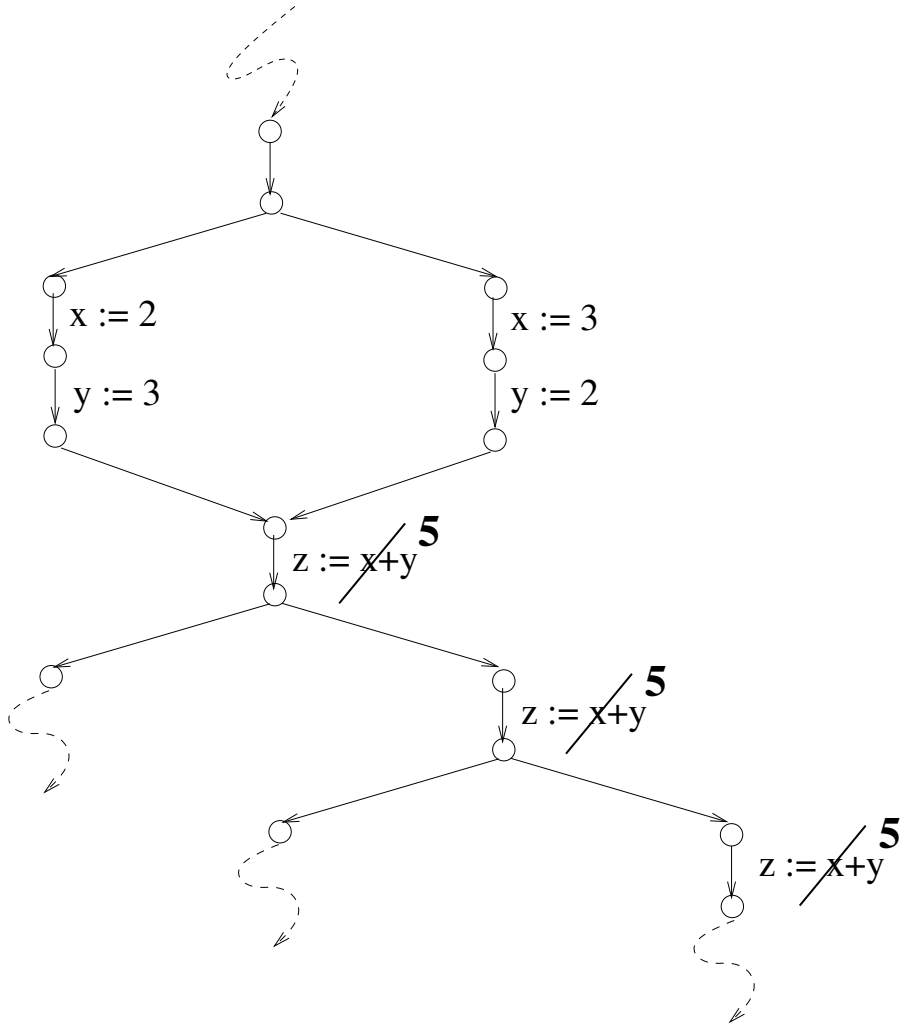
durch jene endlicher Konstantenanalyse:

$$\Sigma_N = \{\sigma_n \mid \sigma_n : \mathbf{T}_n \to \mathbb{Z}_\perp^\top,\ n \in N\},\ [\![e]\!]_{fc} : \Sigma_{\mathbf{T}_{src(e)}} \to \Sigma_{\mathbf{T}dst(e)}$$

zu ersetzen.

Endliche Konstanten

...überkommen die konzeptuelle Schwäche einfacher Konstanten:



The effect of the new algorithm

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 14.1.1

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).

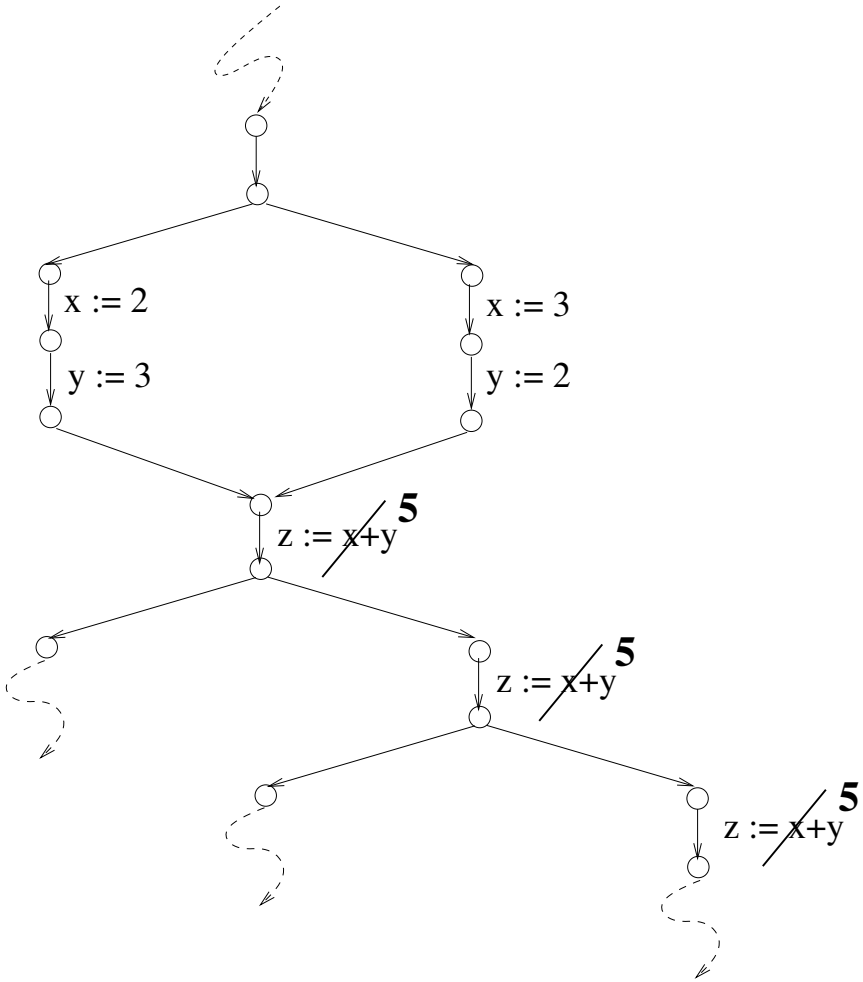
Der Wertegraph stützt sich seinerseits auf die

- Einmal-Zuweisungs-Repräsentation (engl. static single assignment (SSA) form) von Programmen

von Cytron et al. (POPL'89)).

Zurück zum laufenden Beispiel

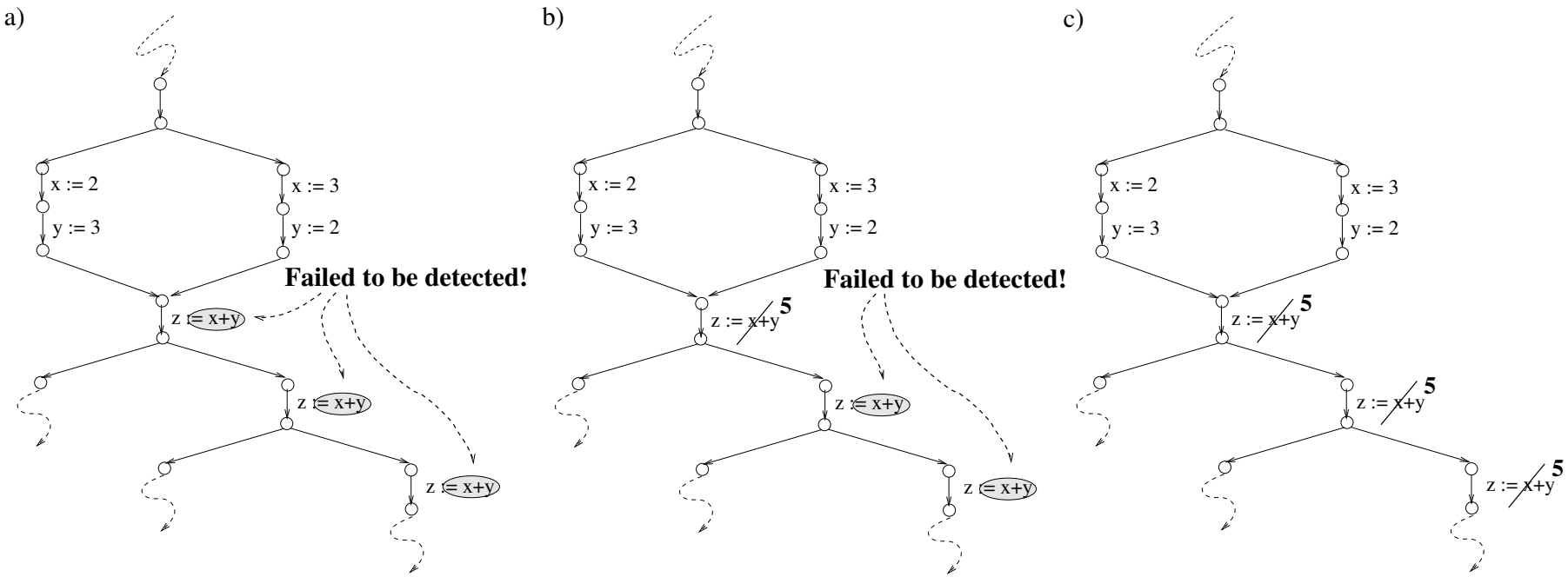
...wie endliche Konstantenanalyse erkennt auch VG-Konstantenanalyse alle Terme als Konstanten:



The effect of the new algorithm

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:



After simple constant propagation
(Note: No effect at all!)

After simple constant propagation
enriched by the "look–ahead–of–one" heuristics
of Kam and Ullman

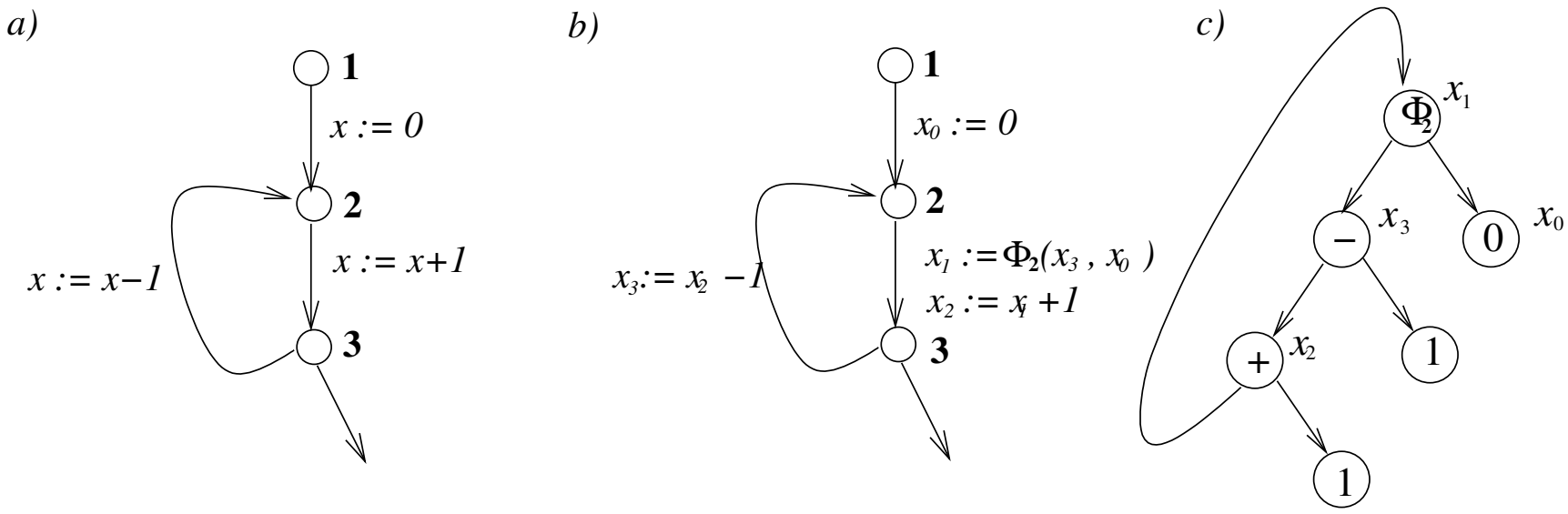
The effect of the new algorithm

Kapitel 14.2

Konstantenanalyse auf dem Wertegraph

Der Wertegraph

...eines Programms nach Alpern, Wegman und Zadec (POPL'88):



Flussgraph SSA-Graph Wertegraph

Konstantenanalyse auf dem Wertegraphen

...in zwei unterschiedlich mächtigen Varianten:

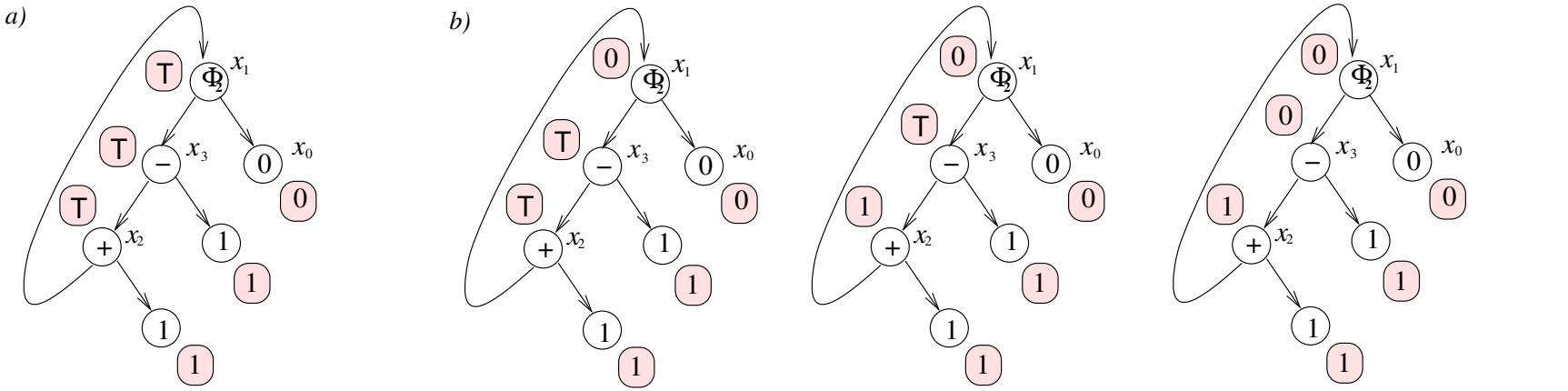
- VG-Basiskonstantenanalyse
  ...erkennt die Klasse einfache Konstanten.
- Volle VG-Konstantenanalyse
  ...erkennt eine Klasse von Konstanten, die weit über die Klassen der einfachen und SC+-Konstanten hinausgeht.

Kapitel 14.2.1

VG-Basiskonstantenanalyse

Konstantenanalyse auf dem Wertegraphen

...illustriert anhand eines Beispiels:



After the initialization step | After the 1st iteration step | After the 2nd iteration step | After the 3rd iteration step: Stable!

Analyseresultat: x_2 und x_3 sind (einfache) Konstanten!

Hauptresultat

...für die VG-Basiskonstantenanalyse.

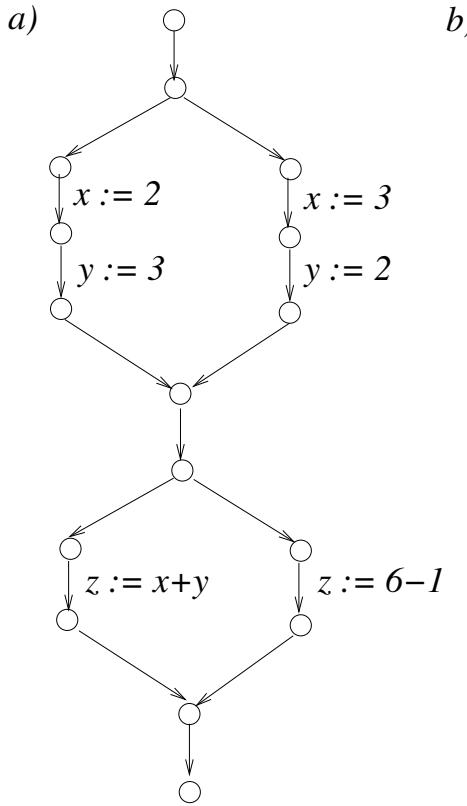
Lemma 14.2.1.1

Die VG-Basiskonstantenanalyse ist korrekt und erkennt die Klasse der einfachen Konstanten.

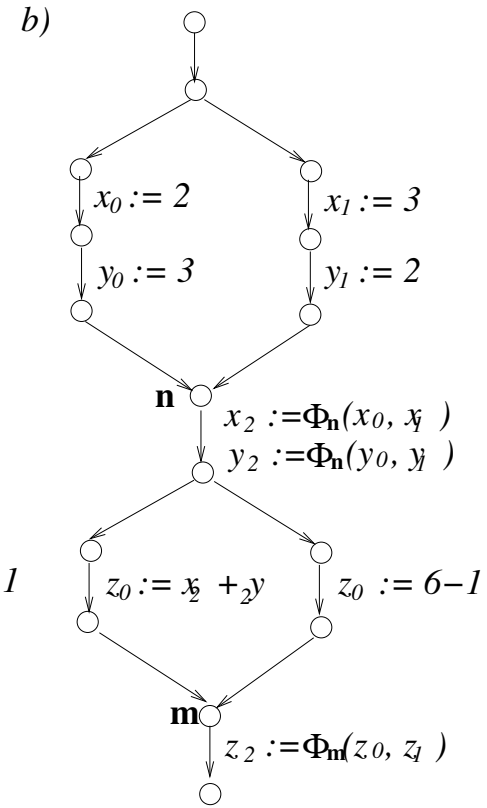
Kapitel 14.2.2

Volle VG-Konstantenanalyse

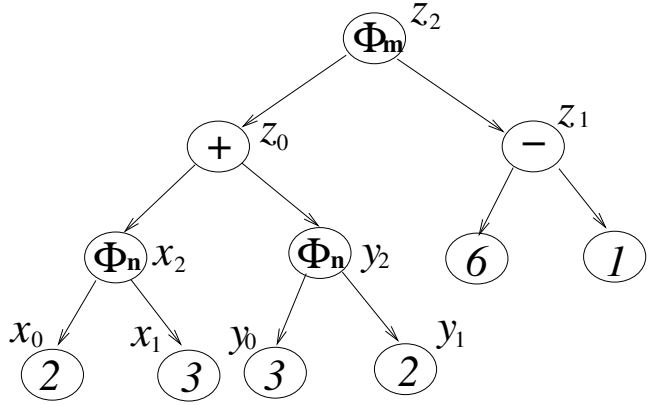
Illustrierendes Beispiel



Flussgraph



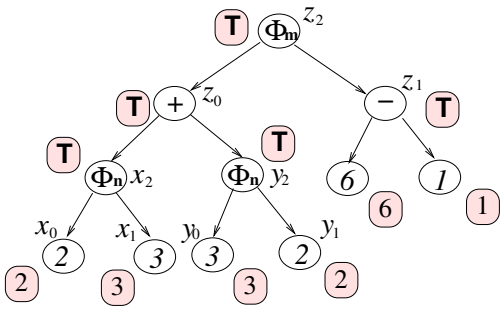
SSA-Graph



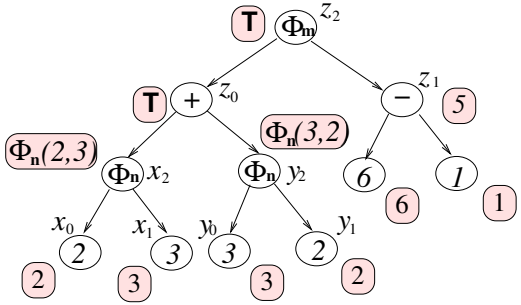
Wertegraph

Volle VG-Konstantenanalyse

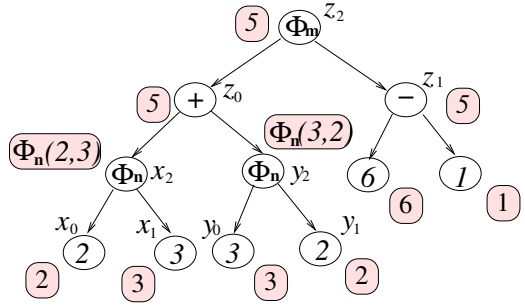
...illustriert anhand des laufenden Beispiels:



The start annotation



After the first iteration



After the second iteration. Stable!

Der technische Clou: Die

- Einführung von Φ-Konstanten
- Anpassung der Evaluationsfunktion auf Wertegraphen!

Hauptresultate

Lemma 14.2.2.1

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.

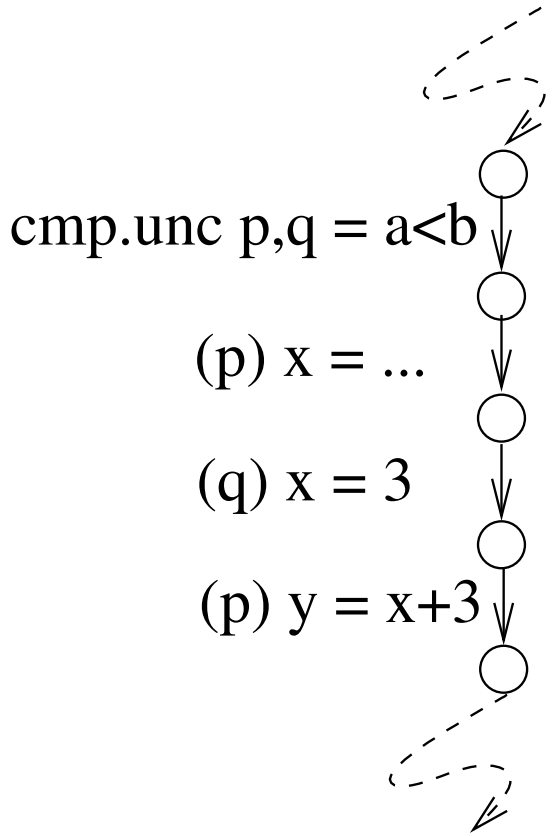
Essentiell dafür ist die Abstützung auf den Wertegraph (und damit indirekt auf den SSA-Graphen eines Programms).

Kapitel 14.3

Konstantenanalyse auf dem prädikatierten Wertegraph

Prädikatierter Code

...als Resultat sog. if-Konversion:

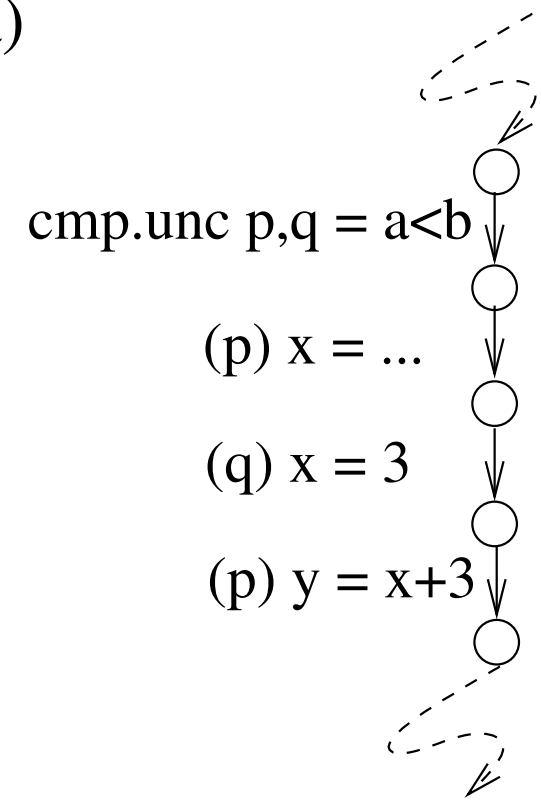


...mit dem Ziel besserer Parallelisierbarkeit durch erhöhte Parallelität auf Instruktionsebene (engl. instruction-level parallelism).

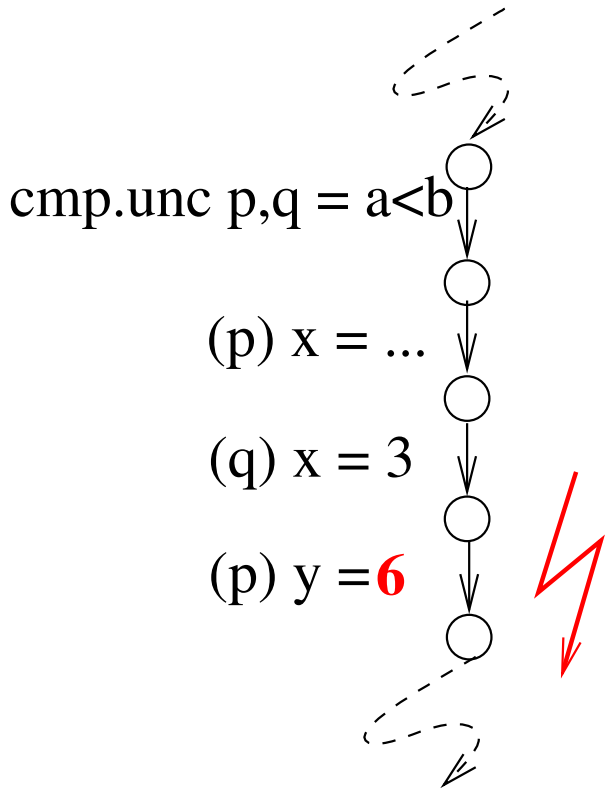
Naive Übertragung

...von Konstantenanalysetechniken von unprädikatiertem auf prädikatierten Code schlägt fehl:

a)



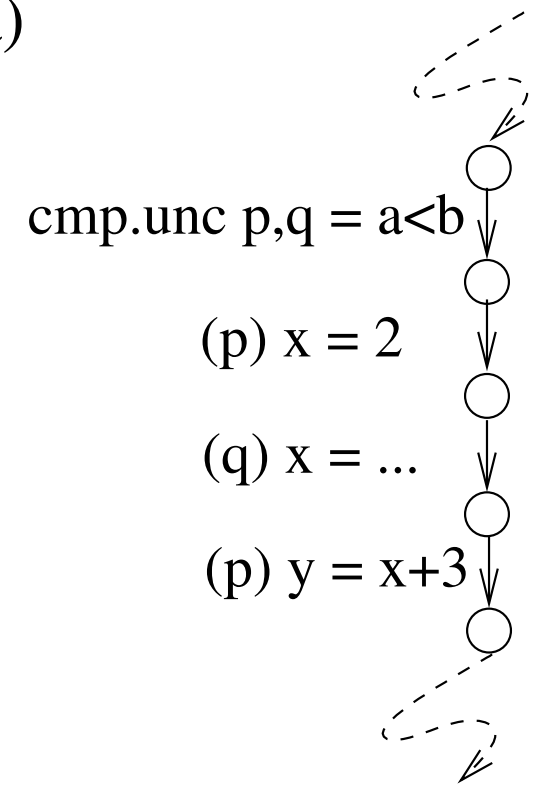
b)



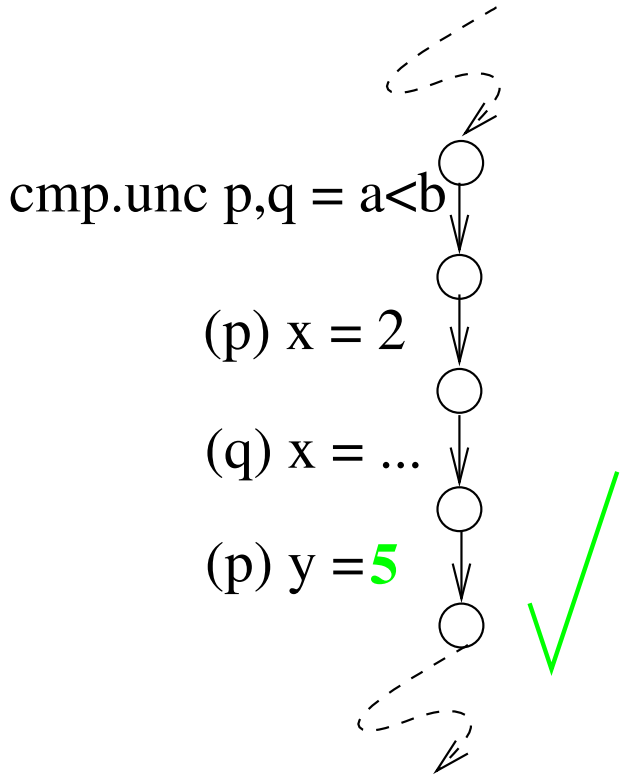
Naive korrekte Übertragung

...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.

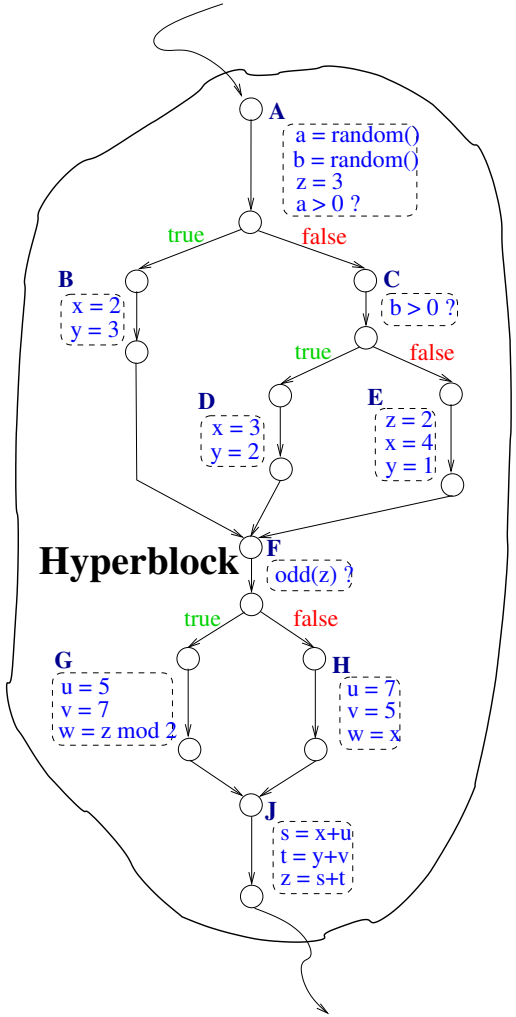
Kapitel 14.3.1

Hyperblöcke, Hyperblockgraphen

Hyperblöcke

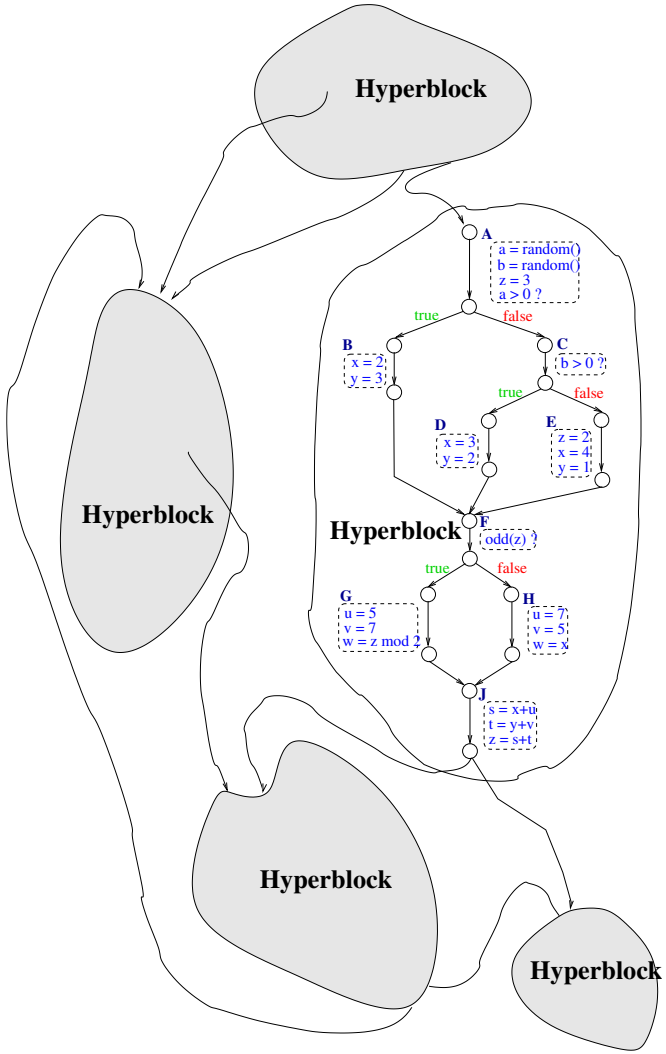
...sind Schlüsselbausteine prädikatierten Codes gekennzeichnet durch:

- ein Eintrittspunkt, mehrere Austrittspunkte.



Hyperblockgraphen

...sind die Hyperblockzerlegung eines Flussgraphen, hier anhand des durchgehenden Beispiels illustriert:



Aufbau der PVG-Konstantenanalyse

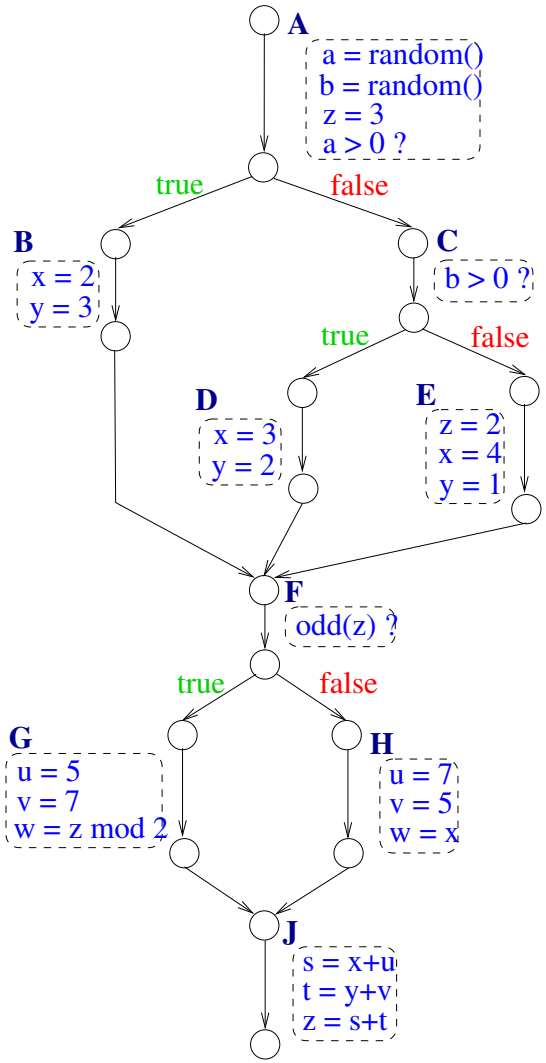
1. Stufe: Lokale Analyse
   - Separate und unabhängige Analyse aller Hyperblöcke eines Programms.
2. Stufe: Globale Analyse
   - Globalisierung der Ergebnisse der lokalen Analysestufe.

Kapitel 14.3.2

Lokale Hyperblock-Konstantenanalyse

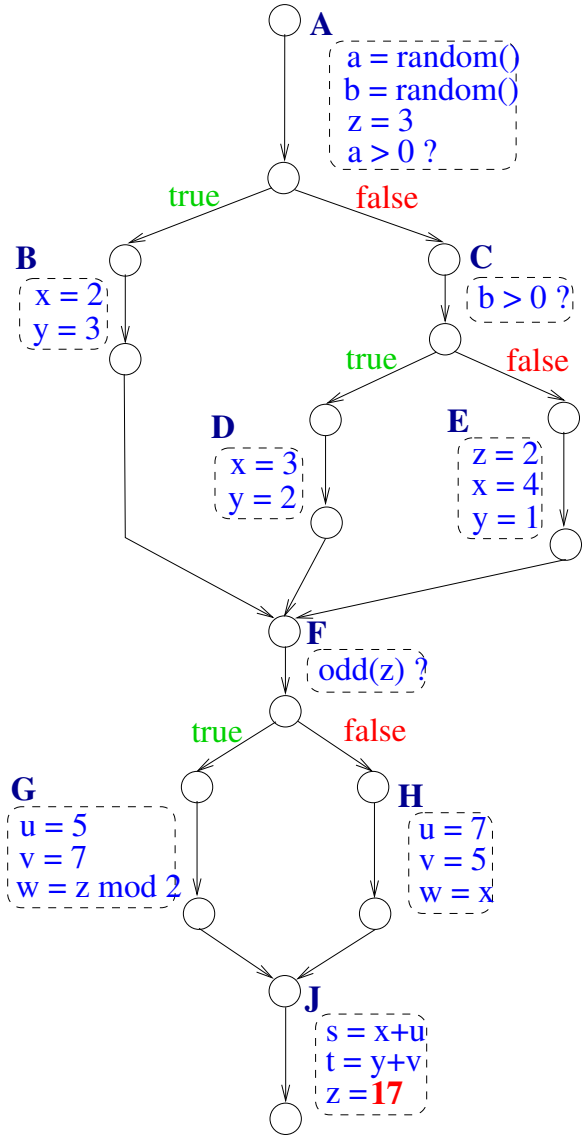
Diskussion der lokalen Analysestufe

....anhand des Hyperblocks unseres durchgehenden Beispiels:



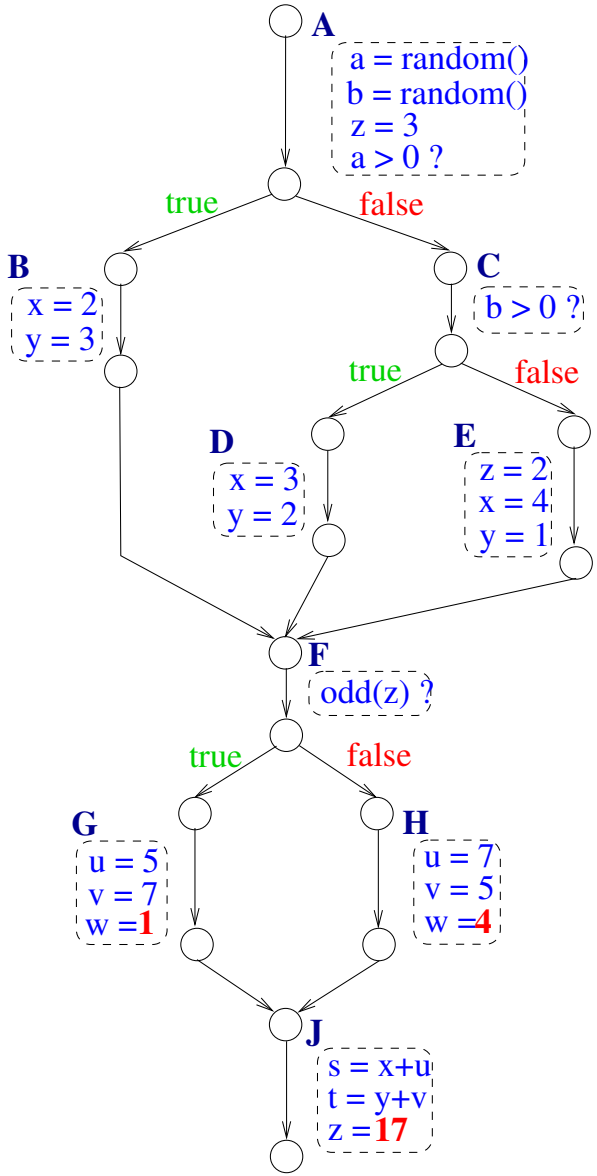
Original Hyperblock

Die PVG-Grundalgorithmustransformation



The Non–Deterministic Path–Precise Basic Optimization

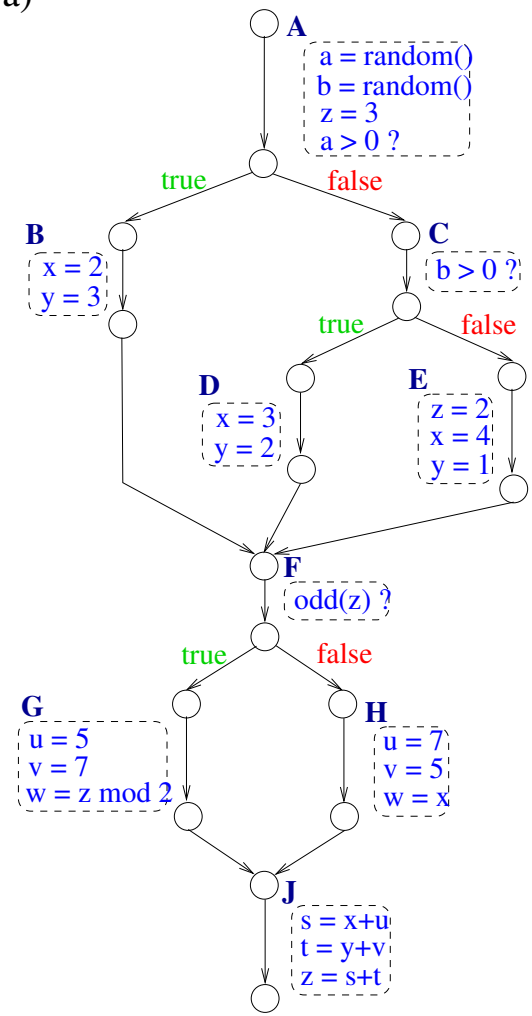
Die volle PVG-Algorithmustransformation



The Deterministic Path–Precise Full Optimization

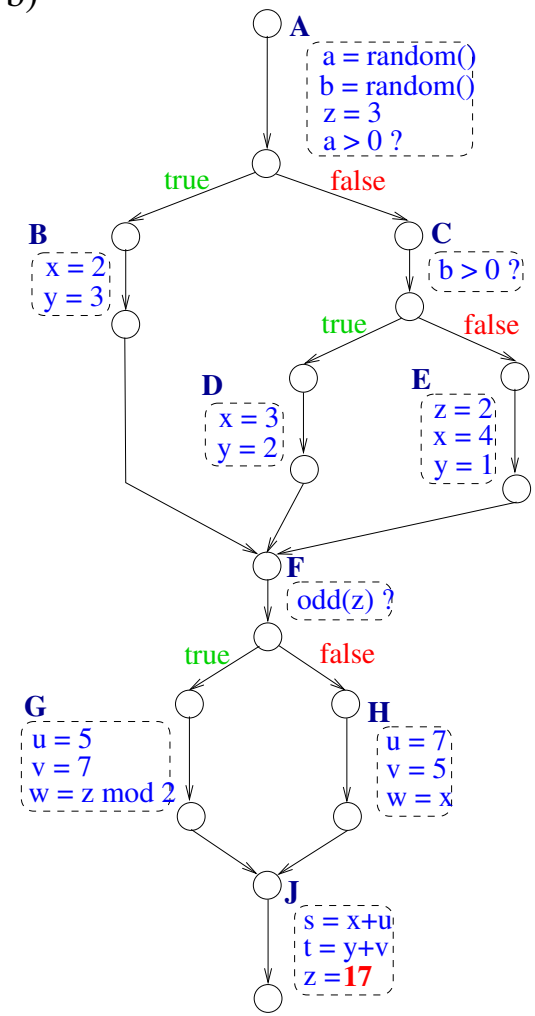
Ausgangsprogramm & beide Transformationen

a)



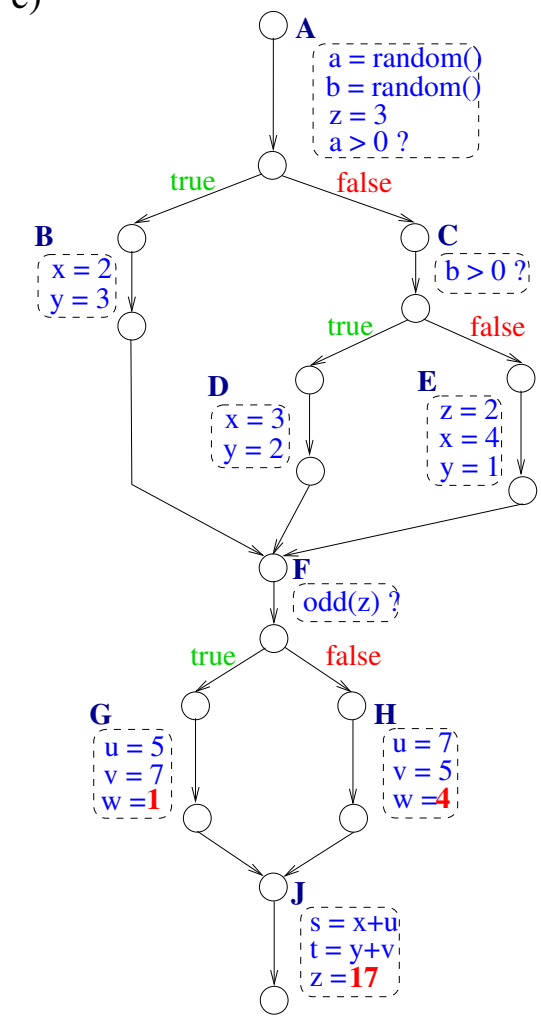
Original Hyperblock

b)



The Non–Deterministic Path–Precise Basic Optimization

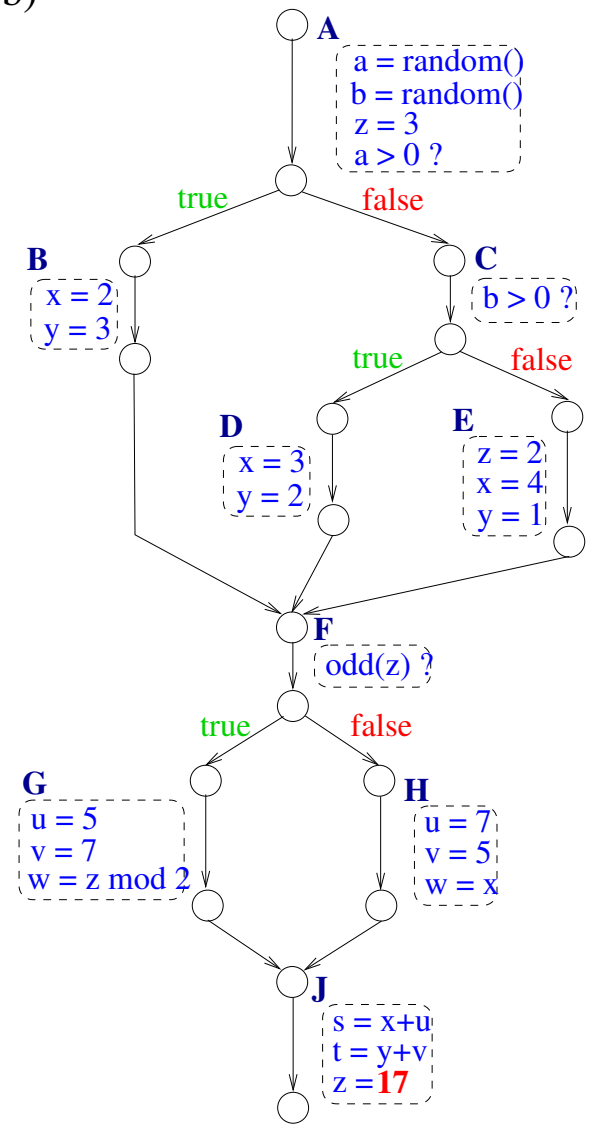
c)



The Deterministic Path–Precise Full Optimization

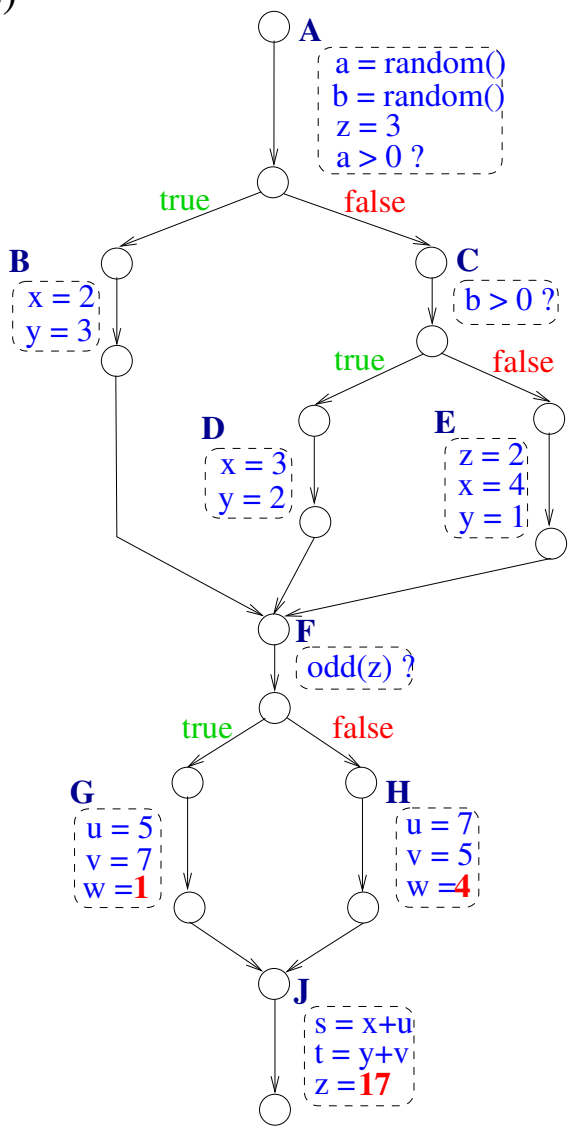
Beide Transformationen auf einen Blick

b)



The Non–Deterministic Path–Precise Basic Optimization

c)



The Deterministic Path–Precise Full Optimization

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):

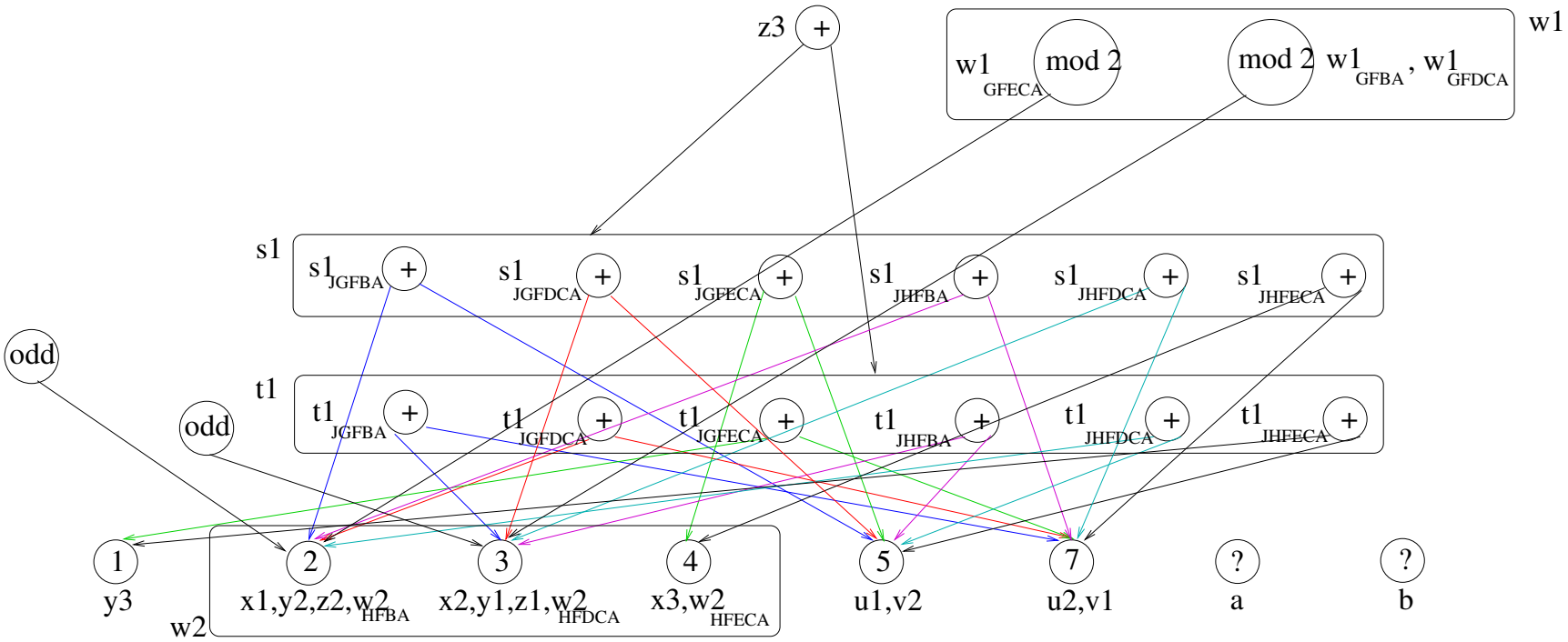
```
begin (p0)     A = OR(TRUE);                    | [*] (HFBA)   w2 = x1;
      (A)      (a1,b1) = (random(),random());   | [*] (HFDCA)  w2 = x2;                 |
      (A)      z1 = 3;                          |     (HFECA)  w2 = x3;
      (A)      cmp.unc BA,CA (a1>0);            |     (H)      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 14.3.3

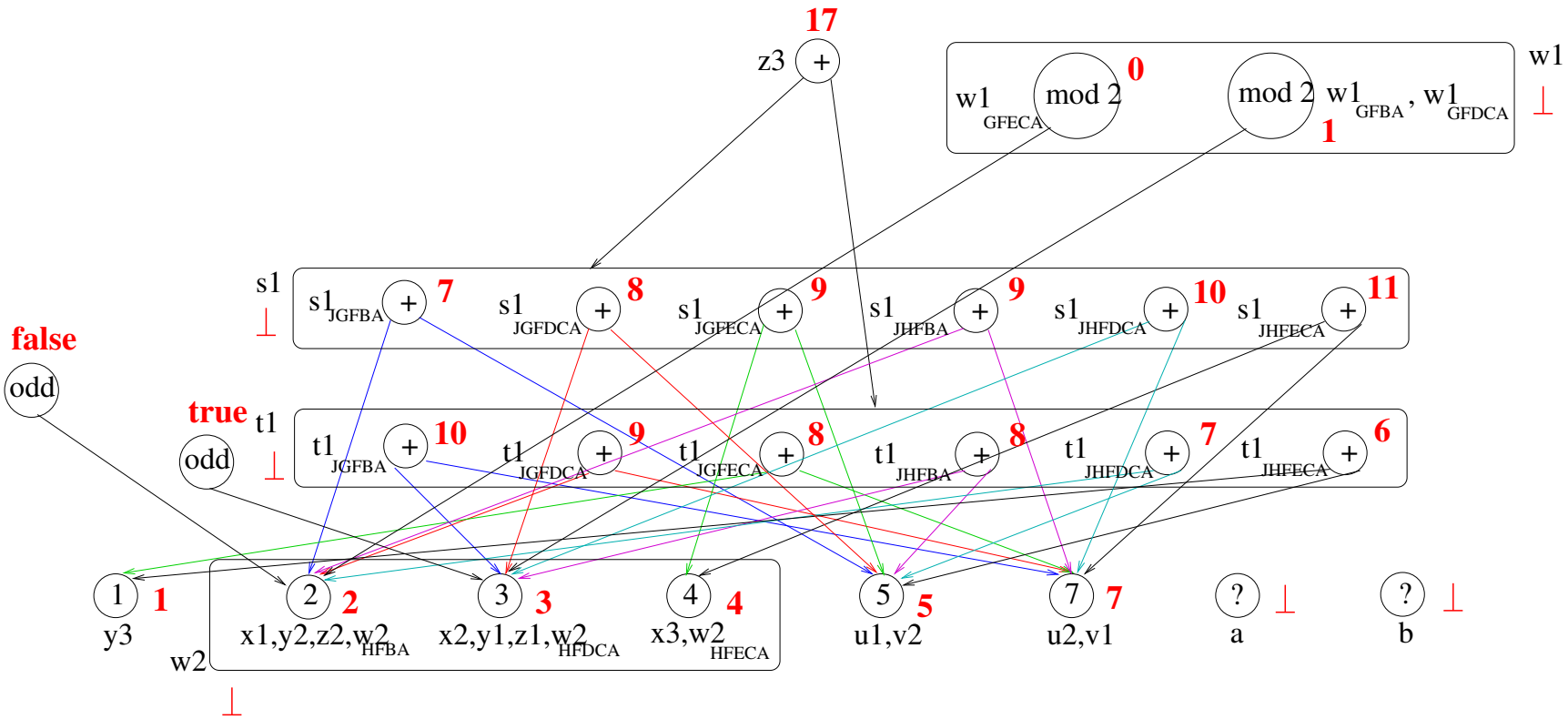
PVG-Basiskonstantenanalyse

Die PVG-Basiskonstantenanalyse

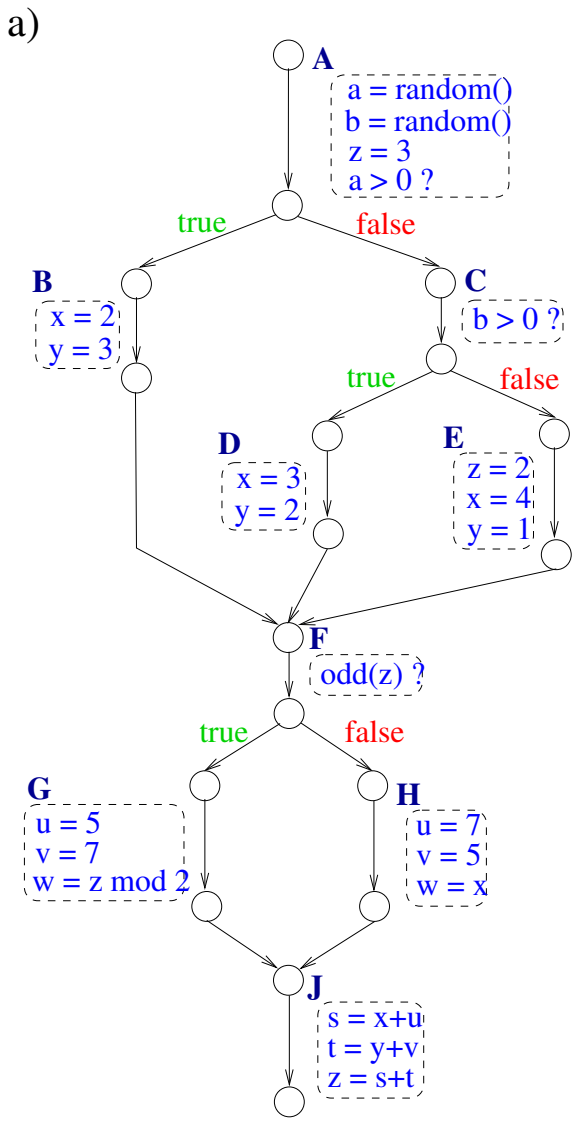
... auf PSSA-basiertem PVG ohne Wächterprädikatausnutzung (engl. guarding predicates):



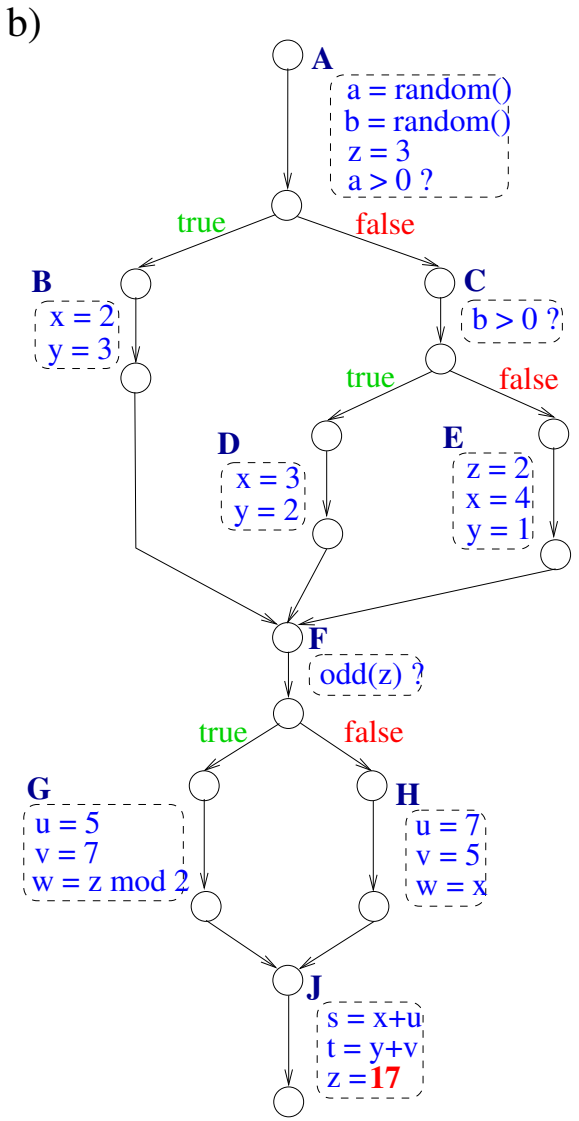
Resultat der PVG-Basiskonstantenanalyse



PVG-Basiskonstantenanalysetransformation



Original Hyperblock



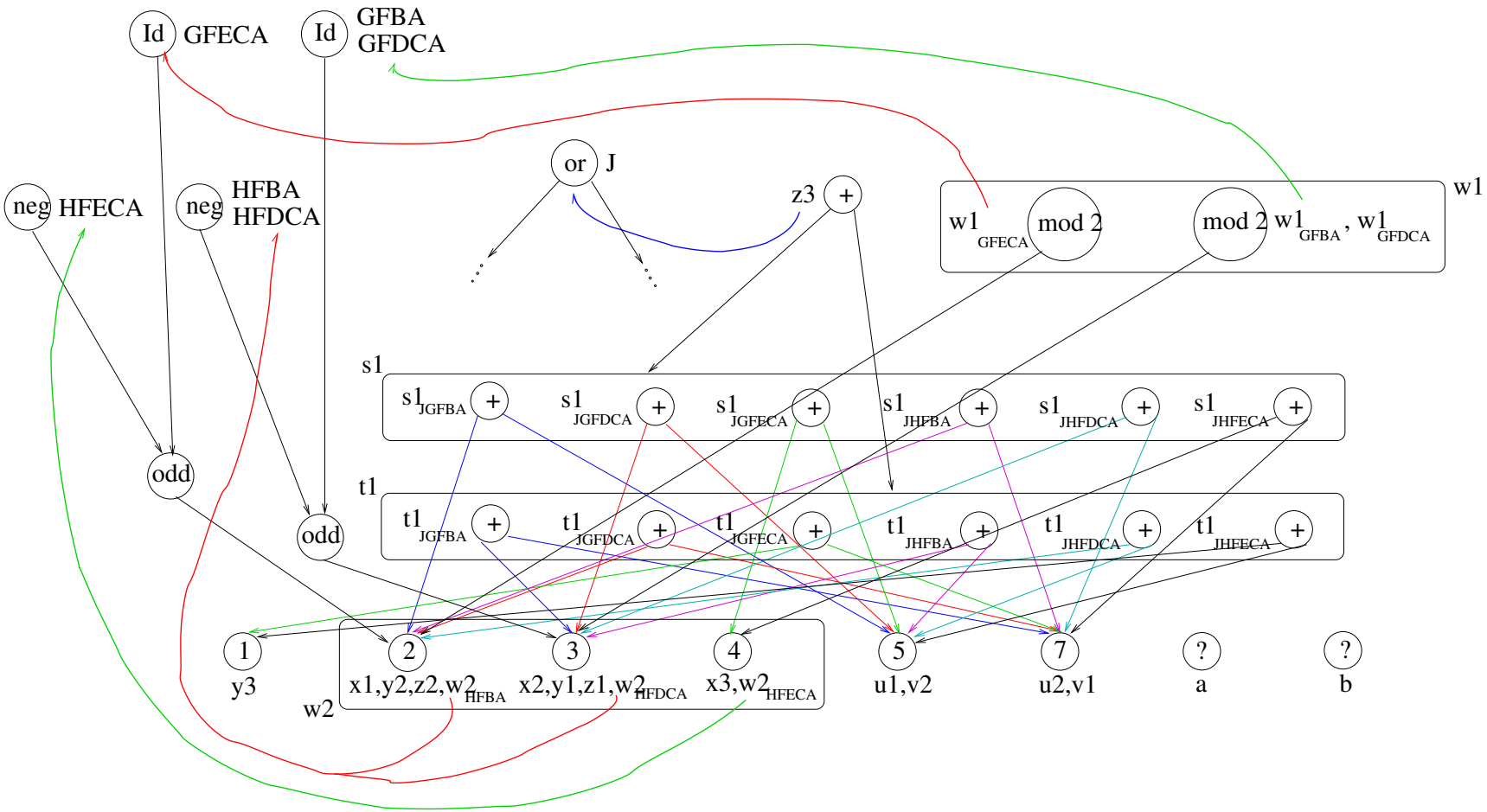
The Non–Deterministic Path–Precise Basic Optimization

Kapitel 14.3.4

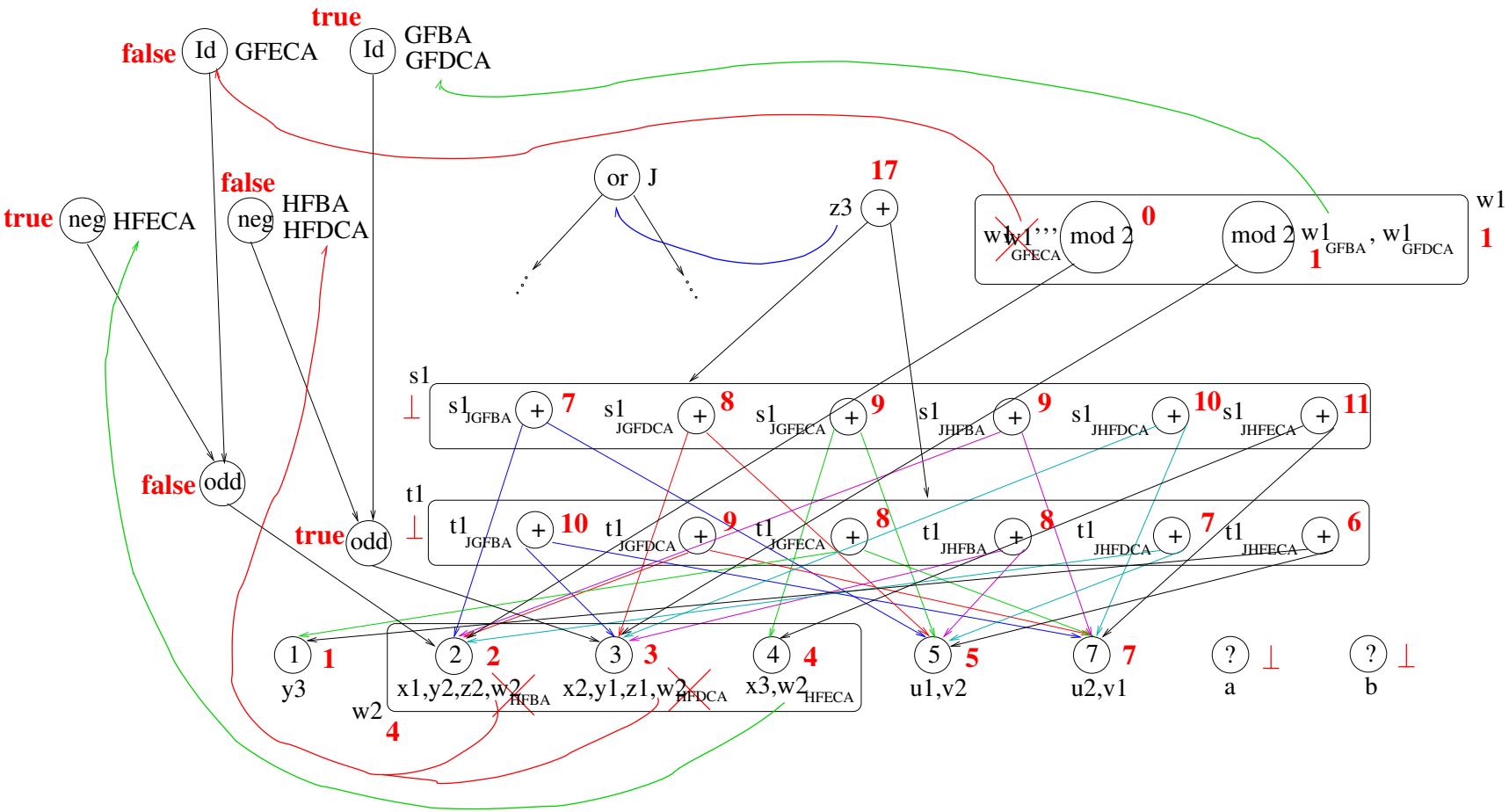
Volle PVG-Konstantenanalyse

Der prädikatierte Wertegraph

...unter Ausnutzung der Wächterprädikate (engl. guarding predicates):

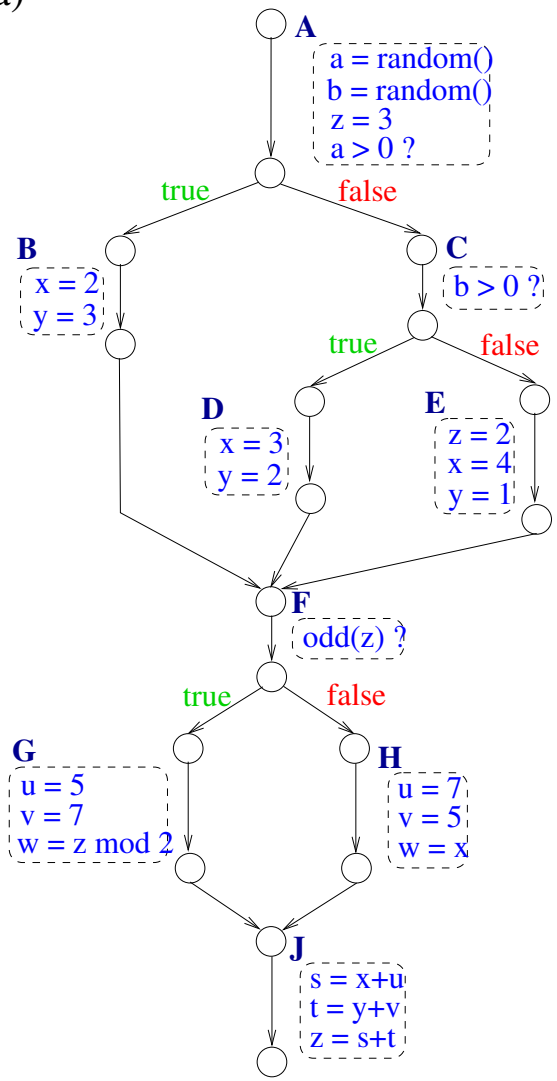


Resultat der vollen PVG-Konstantenanalyse



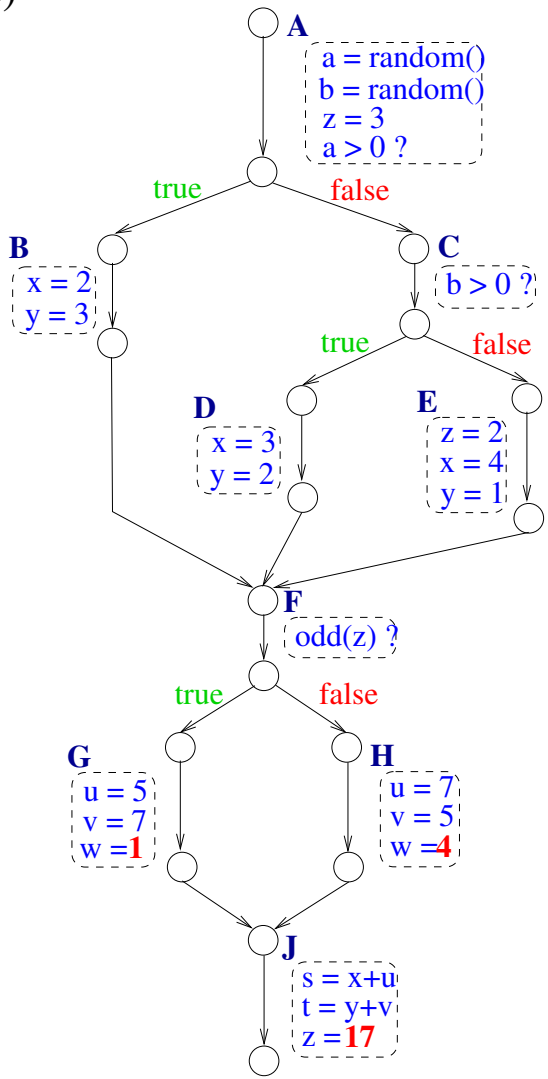
Volle PVG-Konstantenanalysetransformation

a)



Original Hyperblock

c)



The Deterministic Path–Precise Full Optimization

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 14.3.4.1 (Korrektheit)

Die PVG-Basis- und die volle PVG-Konstantenanalyse sind korrekt.

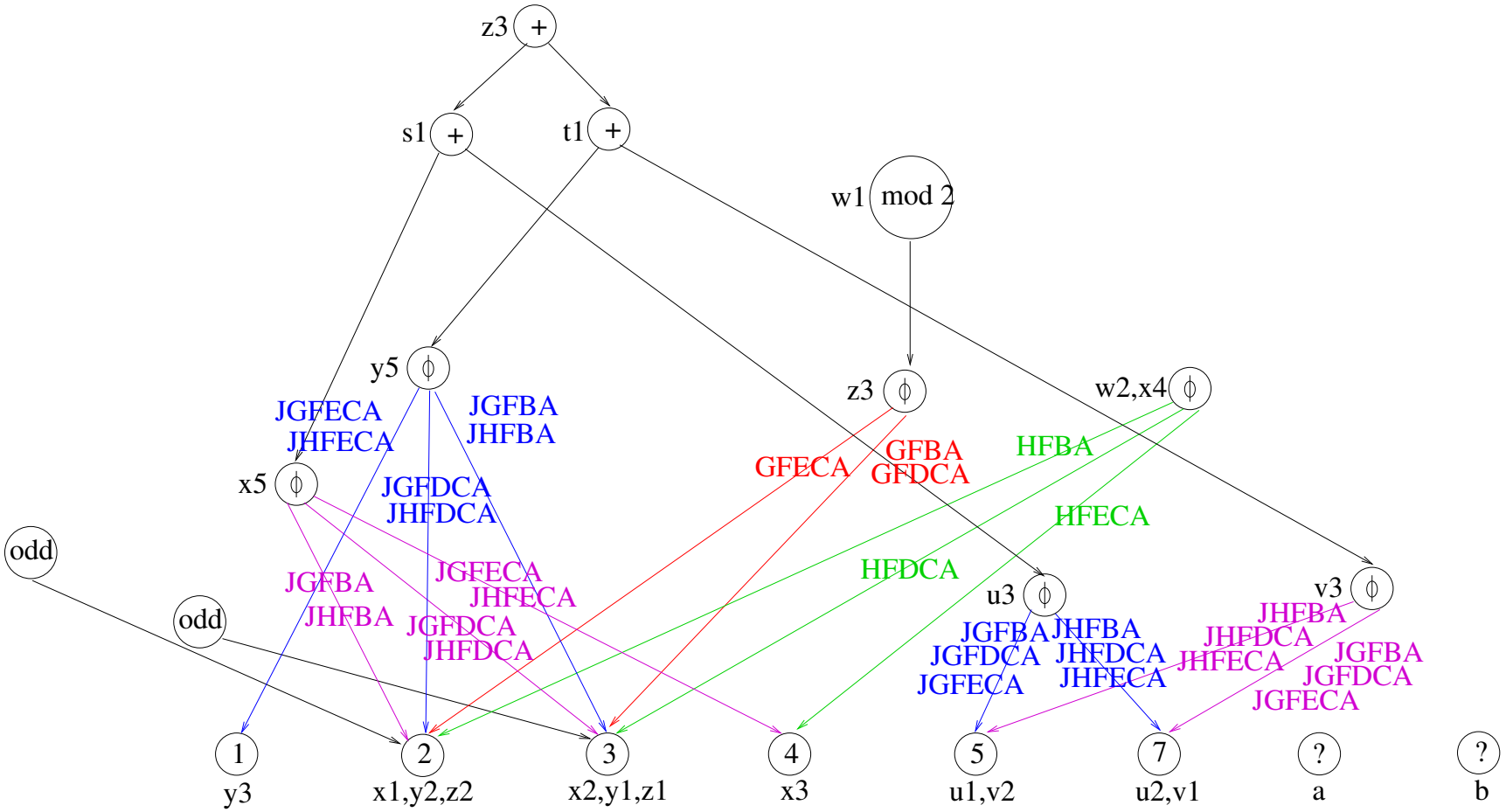
Lemma 14.3.4.2 (Vollständigkeit/Optimalität)

- Die PVG-Basiskonstantenanalyse ist pfad-präzise für nicht-deterministische Interpretation von Verzweigungsbedingungen.
- Die volle PVG-Konstantenanalyse ist prädikatsensitiv pfad-präzise.

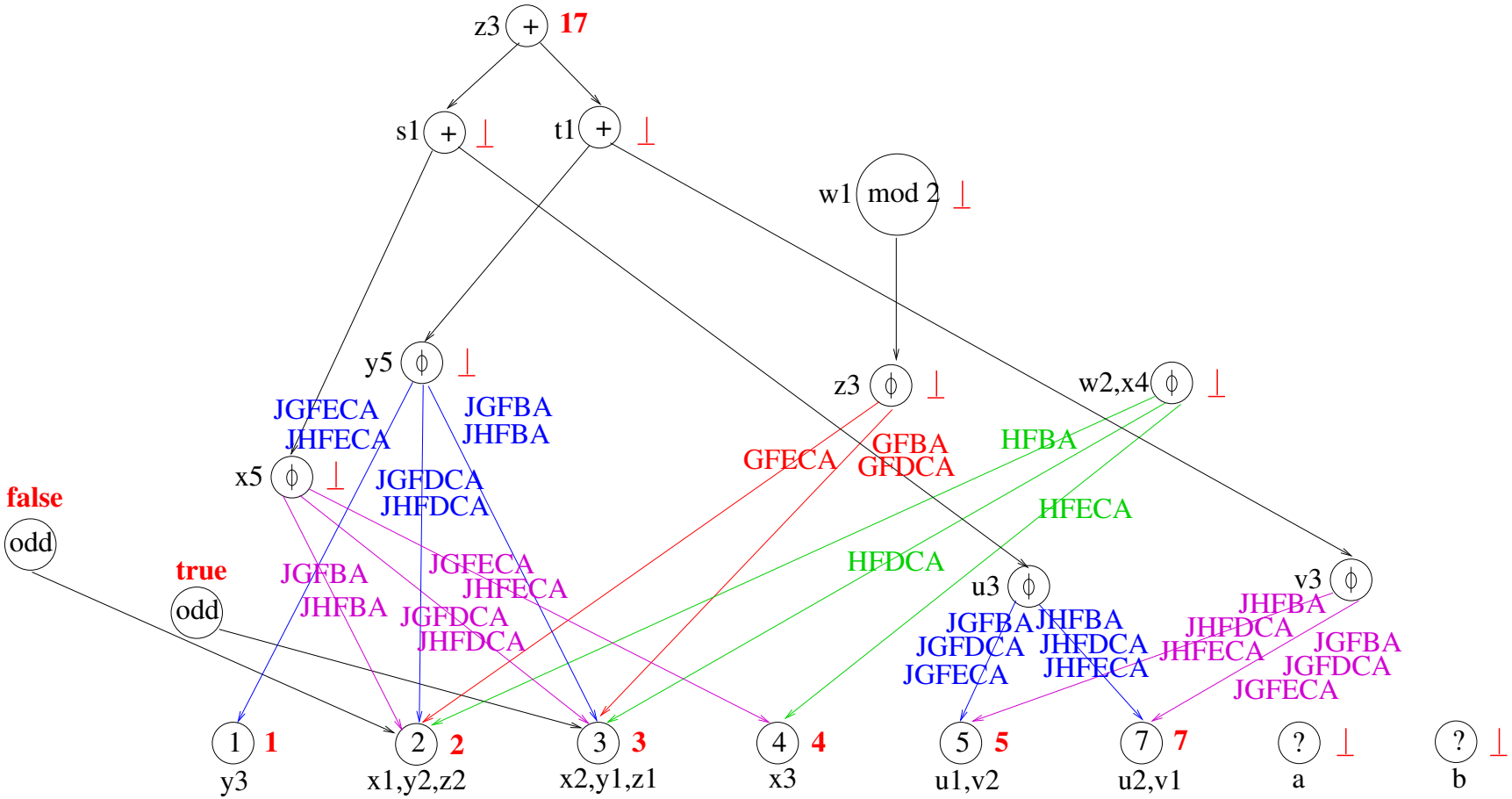
Kapitel 14.3.5

Variationen zur Performanzverbesserung

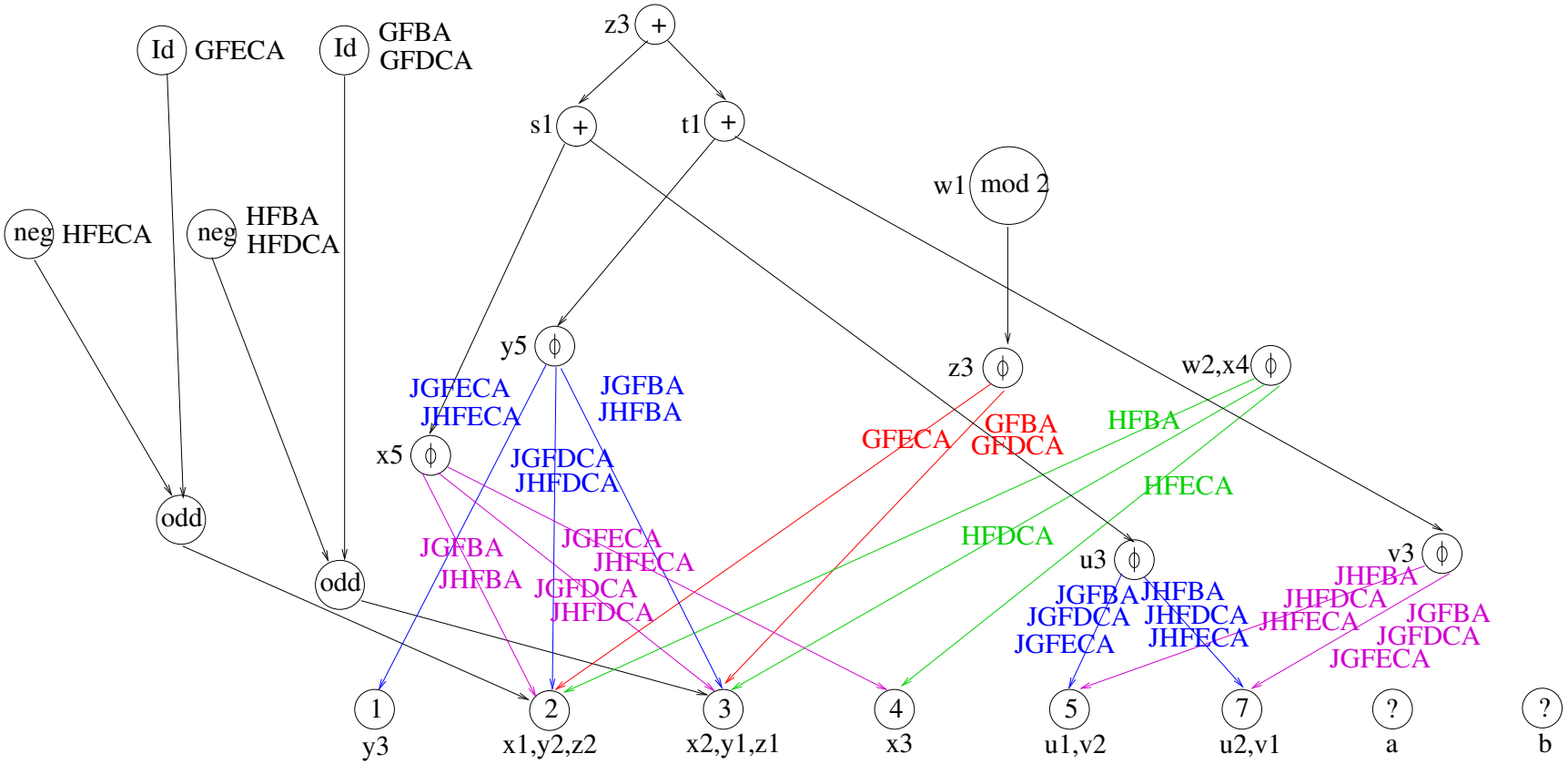
Performanzverbesserung der PVG-Basisanalyse



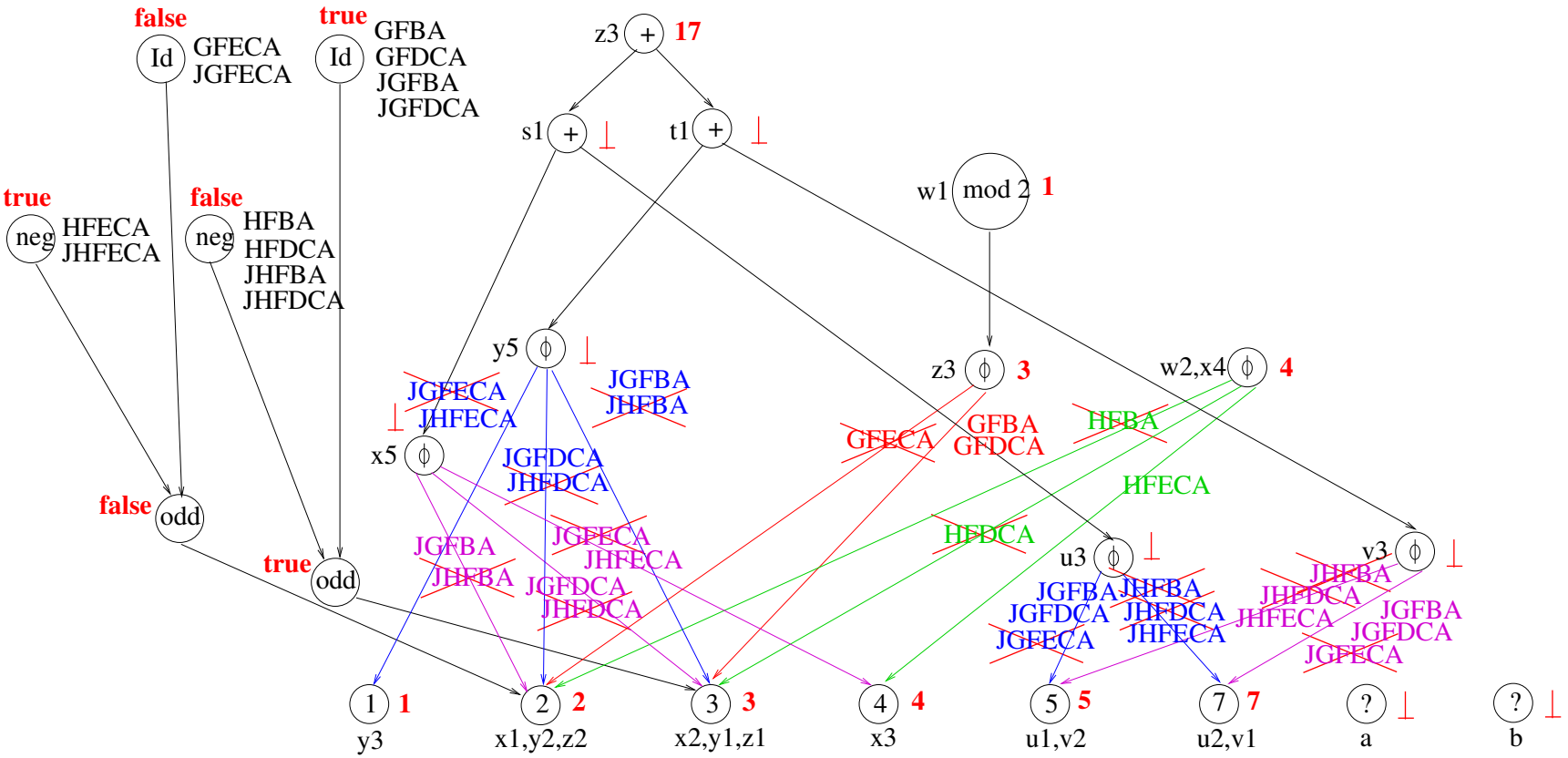
Wirkung d. Performanzverb. f. d. Basisanalyse



Performanzverbesserung d. vollen PVG-Analyse



Wirkung d. Performanzverb. f. d. volle Analyse



Kapitel 14.4

Zusammenfassung

Zusammenfassung

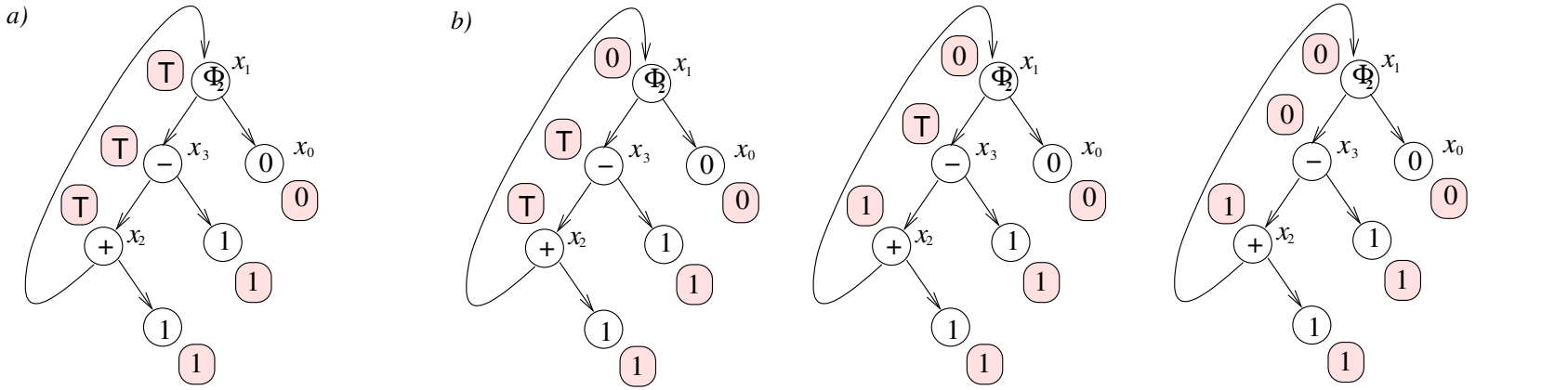
Konstantenanalyse und SSA/PSSA-basierter (prädikatierter) Wertegraph sind

- 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**easy!



After the initialization step | After the 1st iteration step | After the 2nd iteration step | After the 3rd iteration step: Stable!

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 14.5

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (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 14 (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 14 (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 14 (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 14 (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 14 (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 V

Abstrakte Interpretation und Modellprüfung

Kapitel 15

Abstrakte Interpretation und Datenfluss-analyse

Kapitel 15.1

Motivation

Motivation

...abstrakte Interpretation – ein 'mondäner' Programmanalyseansatz (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*

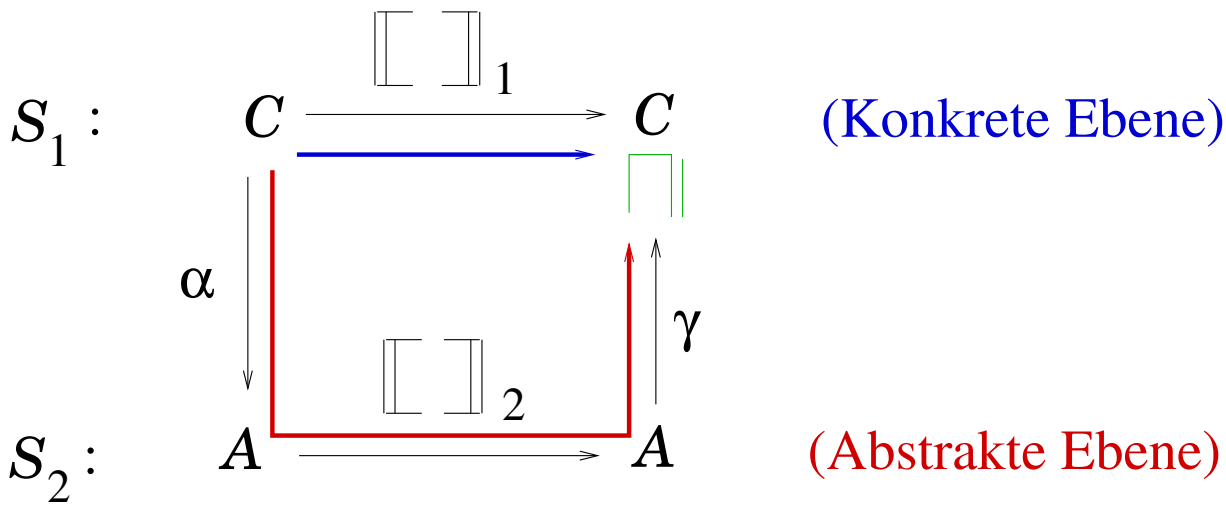
bewiesen wird.

Kapitel 15.2

Theorie abstrakter Interpretation

Theorie abstrakter Interpretation

...ein Ansatz mit zwei (oder mehr) Beobachtungsniveaus:



zusammengehalten durch Wohlzusammenhangsforderungen:

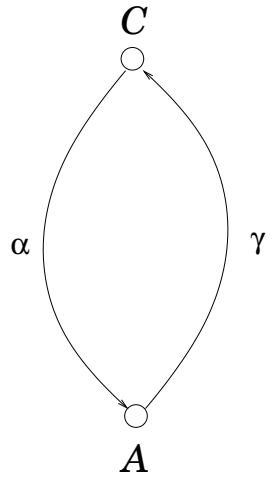
1. $\alpha : \mathcal{C} \to \mathcal{A}$ und $\gamma : \mathcal{A} \to \mathcal{C}$ als sog. Abstraktions- und Konkretisierungsfunktionen bilden eine Galois-Verbindung.
2. das Diagramm (schwach) kommutativ ist.

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.
- $Id_C : C \to C$, $Id_A : A \to A$, d.h.: Identitäten auf C und A: $Id_C = \lambda\, c.\ c$ und $Id_A(a) = \lambda\, a.\ a$.

...so dass das Tupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ folgende Situation beschreibt:



Generalvereinbarung:

- Verbandmäßig kleiner heißt bessere, präzisere Information!

Kapitel 15.2.1

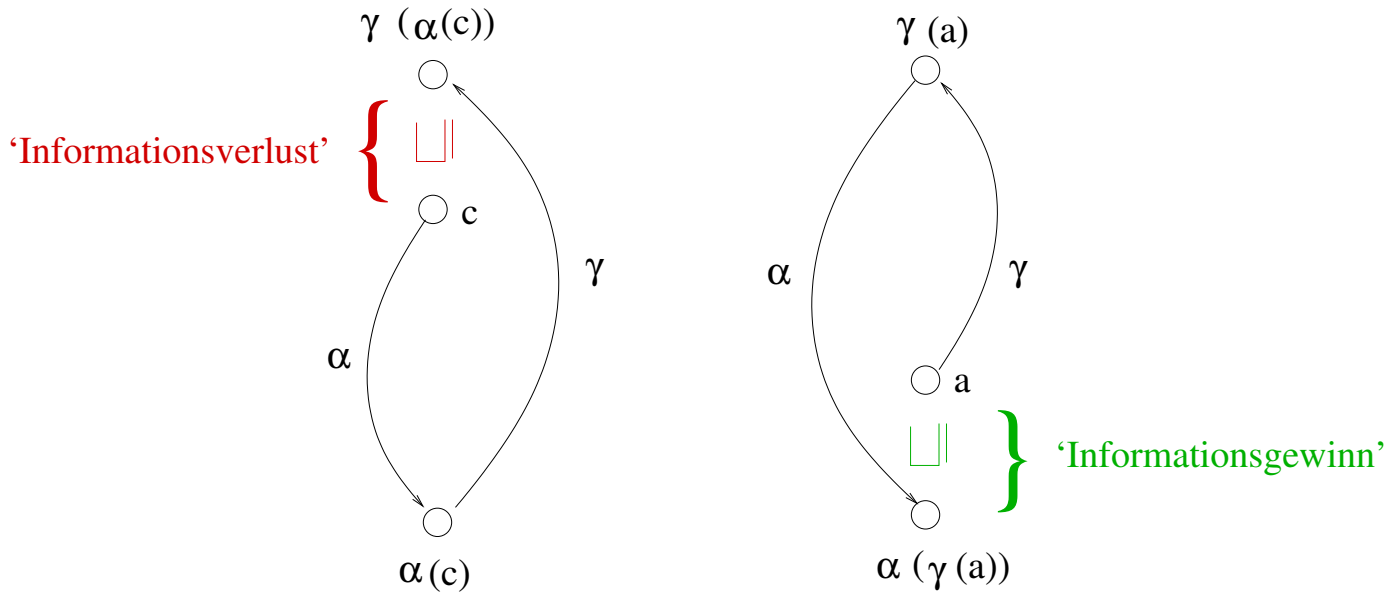
Galois-Verbindungen

Galois-Verbindungen

Definition 15.2.1.1 (Galois-Verbindung)

Ein Quadrupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ heißt Galois-Verbindung (engl. Galois connection) gdw:

1. α und γ sind monoton
2. $\gamma \circ \alpha \sqsupseteq_{\mathcal{C}} Id_C$ ($\sqsupseteq_{\mathcal{C}}$: linksseitig 'Informationsverlust')
3. $\alpha \circ \gamma \sqsubseteq_{\mathcal{A}} Id_A$ ($\sqsubseteq_{\mathcal{A}}$: linksseitig 'Informationsgewinn')



Informell

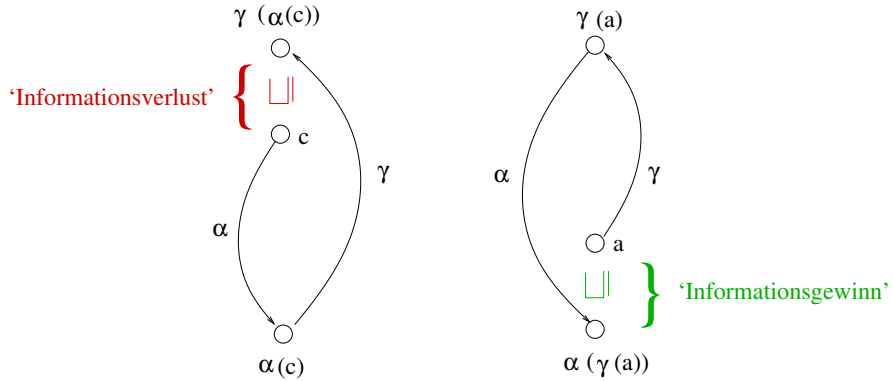
Abstraktion

- kann schaden: Ist die Inklusion $\gamma \circ \alpha \sqsupseteq_C Id_C$ in Definition 15.2.1.1(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 15.2.1.1(2) echt, so bedeutet das (sogar) eine Besserstellung, einen Informationsgewinn auf der abstrakten Ebene.

...entsprechend der Generalvereinbarung: 'kleiner' ist 'besser'.



Maximaler Informationsverlust, -gewinn

Proposition 15.2.1.2 (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:

- $\forall\, c \in C.\ \gamma(c) = \bot_{\mathcal{A}}$
- $\forall\, a \in A.\ \alpha(a) = \top_{\mathcal{C}}$

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:

$$\forall\, [m,n], [p,q] \in IV.$$

$$[\,] \sqsubseteq [\,]$$

$$[\,] \sqsubseteq [p,q]$$

$$[m,n] \sqsubseteq [-\infty, \infty]$$

$$[m,n] \sqsubseteq [p,q] \iff_{df} [m,n] \subseteq [p,q]$$

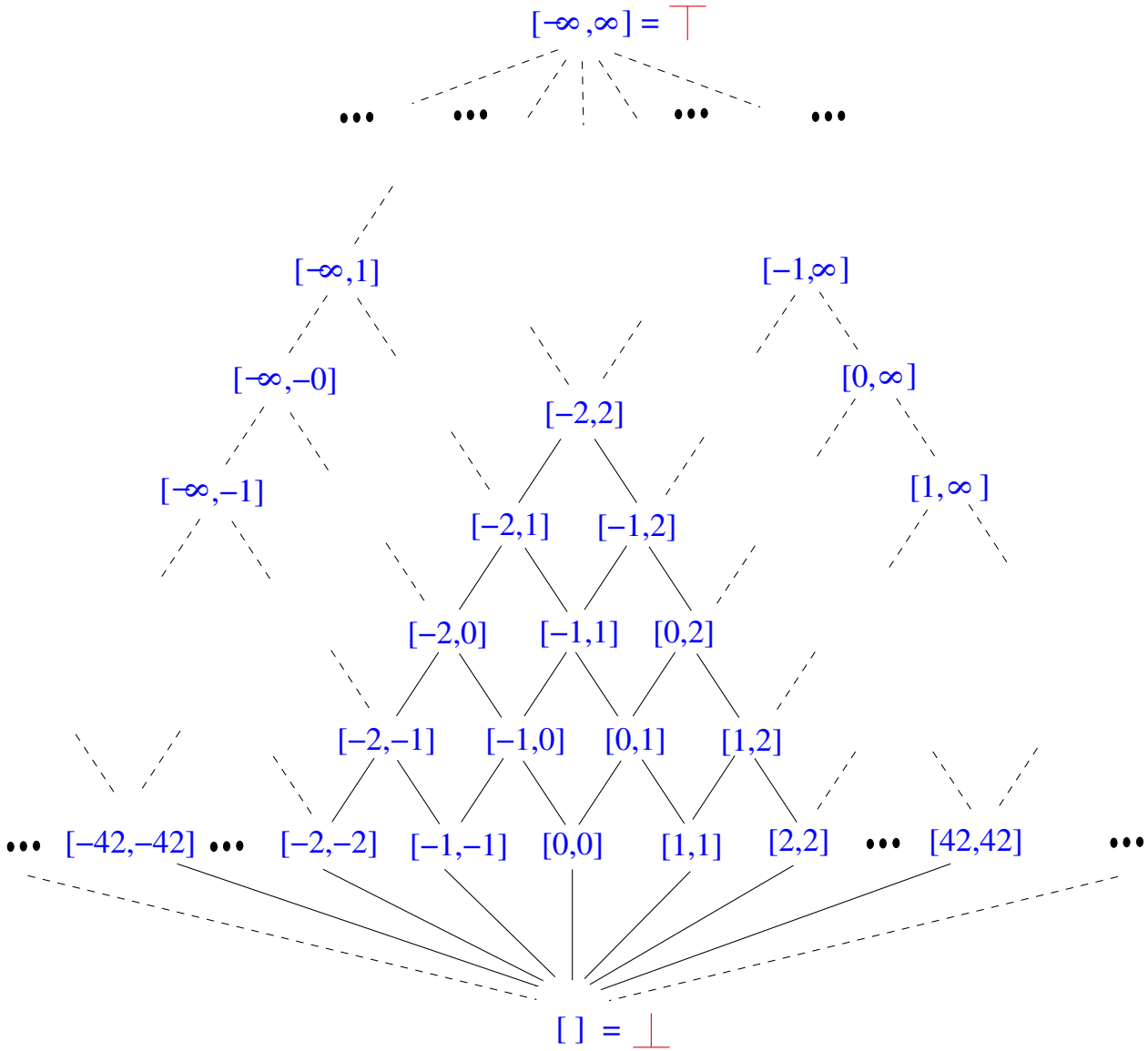
$$= \quad p \leq m \leq n \leq q$$

$$[m,n] \sqcup [p,q] =_{df} [\bigsqcap_{\widehat{\mathbf{Z}}_{-\infty}^{\infty}}([m,n] \cup [p,q]),$$

$$\bigsqcup_{\widehat{\mathbf{Z}}_{-\infty}^{\infty}}([m,n] \cup [p,q])\,]$$

Beispiel: Galois-Verbindung (3)

Der Intervallverband $\mathcal{A}$:



Beispiel: Galois-Verbindung (4)

Proposition 15.2.1.3

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 -gewin- ne durch fortgesetzte Abstraktionen und Konkretisierungen, Neutralität.

Lemma 15.2.1.4 (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 15.2.1.5 (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 15.2.1.5(1)/(2) mit der Festlegung reverser lokaler Semantikfunktionen in Definition 8.2.1.

Eigenschaften von Galois-Verbindungen (3)

...Additivität bzw. Distributivität von Abstraktions- und Konkretisierungsfunktion garantieren einander wechselweise Existenz und eindeutige Bestimmtheit.

Lemma 15.2.1.6 (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 15.2.1.7

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 15.2.1.8

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 15.2.2.4.

Adjunktionscharakterisierung v. Galois-Verbind.

Definition 15.2.1.9 (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 15.2.1.10 (Charakterisierung v. Galois-Verb.)

$(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ ist eine Adjunktion gdw $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ ist eine Galois-Verbindung.

...mit der Adjunktionsbedingung ist oft einfacher zu arbeiten als mit der Verbindungsbedingung.

Kapitel 15.2.2

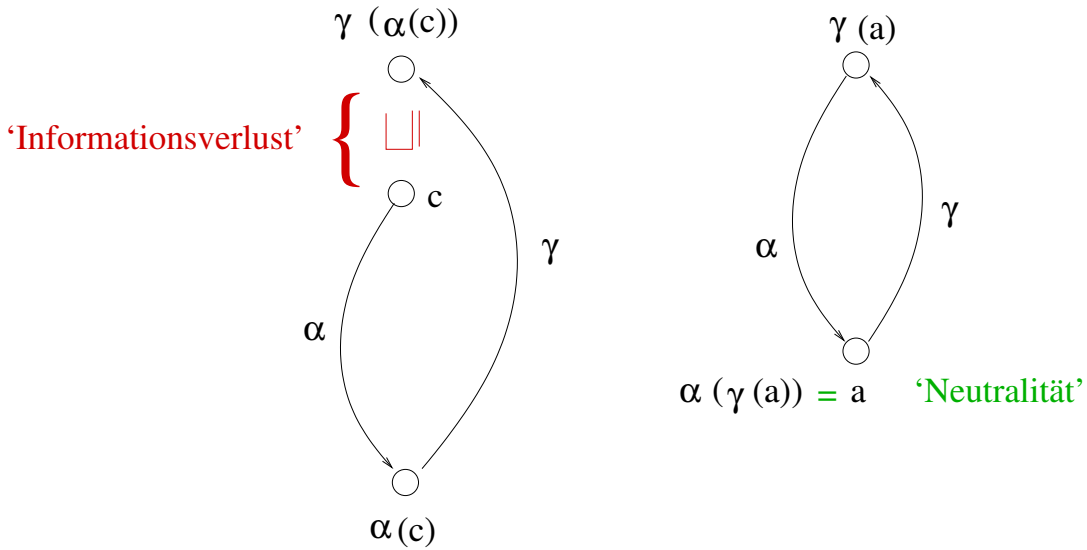
Galois-Passungen

Galois-Passungen

Definition 15.2.2.1 (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')



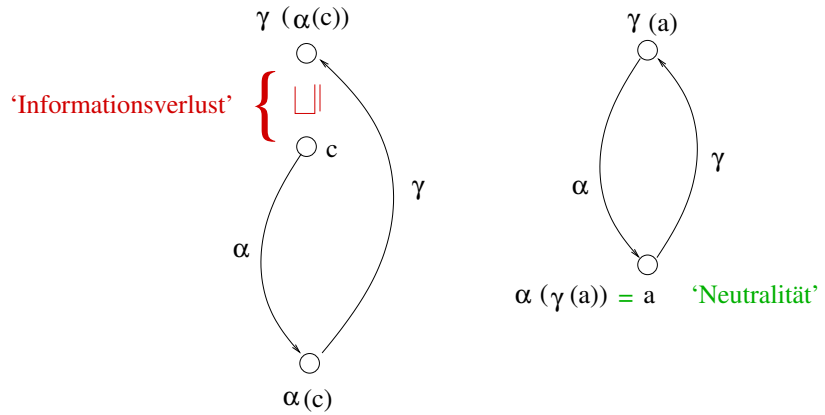
Informell

Wie für Galois-Verbindungen:

- Abstraktion kann schaden: Ist die Inklusion $\gamma \circ \alpha \sqsupseteq_{\mathcal{C}} Id_C$ in Definition 15.2.2.1(1) echt, so bedeutet das eine Schlechterstellung, einen Informationsverlust auf der konkreten Ebene.

Anders als für Galois-Verbindungen:

- Konkretisierung darf zwar nicht schaden, aber auch nicht zu einer Besserstellung, einem Informationsgewinn auf der abstrakten Ebene führen: $\alpha \circ \gamma =_{\mathcal{A}} Id_A$.

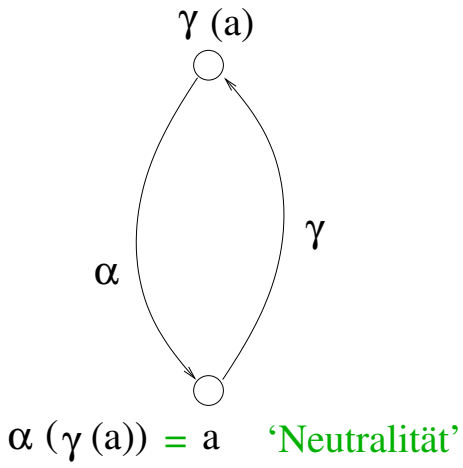


Galois-Passungen: Spezielle Galois-Verbind.

Proposition 15.2.2.2

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 15.2.1.2 ist keine Galois-Passung.

Charakterisierung von Galois-Passungen

Lemma 15.2.2.3 (Äquivalenzaussagen)

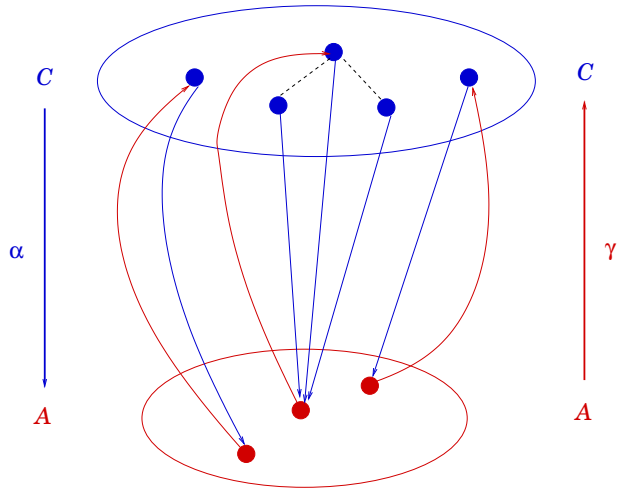
Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung. Dann sind folgende Aussagen äquivalent:

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 ordnungserhaltend, d.h.:
   $\forall\, a_1, a_2 \in A.\ \gamma(a_1) \sqsubseteq_C \gamma(a_2) \ \Leftrightarrow\ a_1 \sqsubseteq_A a_2$.

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 15.2.2.4 (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 15.2.1.6 und 15.2.1.7 und Lemma 15.2.1.9 und 15.2.2.3.

Übungsaufgabe

Ist die Galois-Verbindung $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ aus Proposition 15.2.1.2 zur Intervallanalyse eine Galois-Passung? Beweis oder Gegenbeispiel.

Kapitel 15.3

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 15.3.1

Erschaffende Methoden

1) Extraktionsmethode

Lemma 15.3.1.1 (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 15.3.1.2 (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.

Kapitel 15.3.2

Kombinierende Methoden

1) Unabhängige Attributemethode

Lemma 15.3.2.1 (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 15.3.2.2 (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 15.3.2.3 (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

$$\begin{aligned} \forall f \in [M \overset{tot}{\to} C].\ \alpha'(f) &=_{df} \alpha \circ f \\ \forall g \in [M \overset{tot}{\to} A].\ \gamma'(g) &=_{df} \gamma \circ g \end{aligned}$$

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 15.3.2.4 (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

$$\begin{aligned} \forall\, f \in [C_1 \overset{mon}{\to} C_2].\ \alpha(f) &=_{df} \alpha_2 \circ f \circ \gamma_1 \\ \forall\, g \in [A_1 \overset{mon}{\to} A_2].\ \gamma(g) &=_{df} \gamma_2 \circ g \circ \alpha_1 \end{aligned}$$

eine Galois-Verbindung.

5) Direkte Produktmethode

Lemma 15.3.2.5 (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.

6) Direkte Tensorproduktmethode

...bezeichne $\mathcal{P}$ den Potenzmengenoperator.

Lemma 15.3.2.6 (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.

Kapitel 15.4

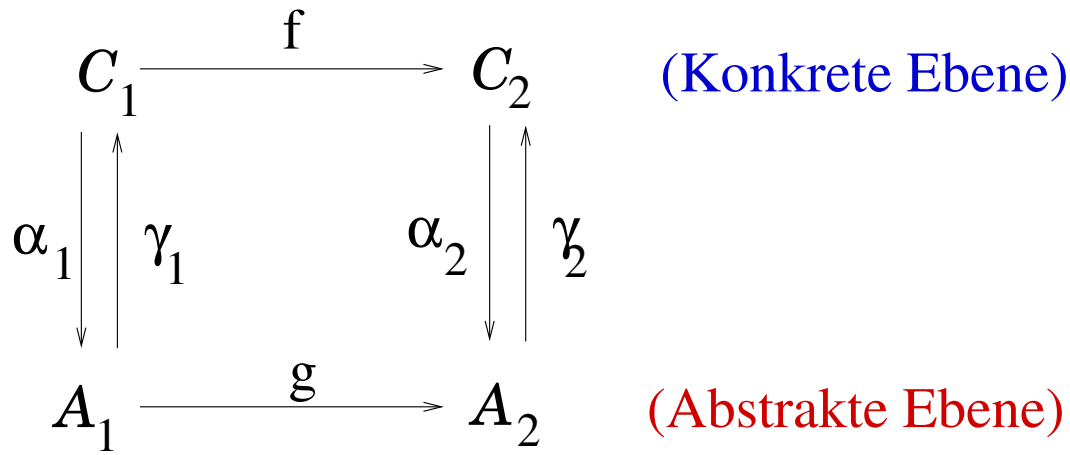
Galois-Systeme

Galois-System

Definition 15.4.1 (Galois-System)

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 15.4.2 (Charakterisierung)

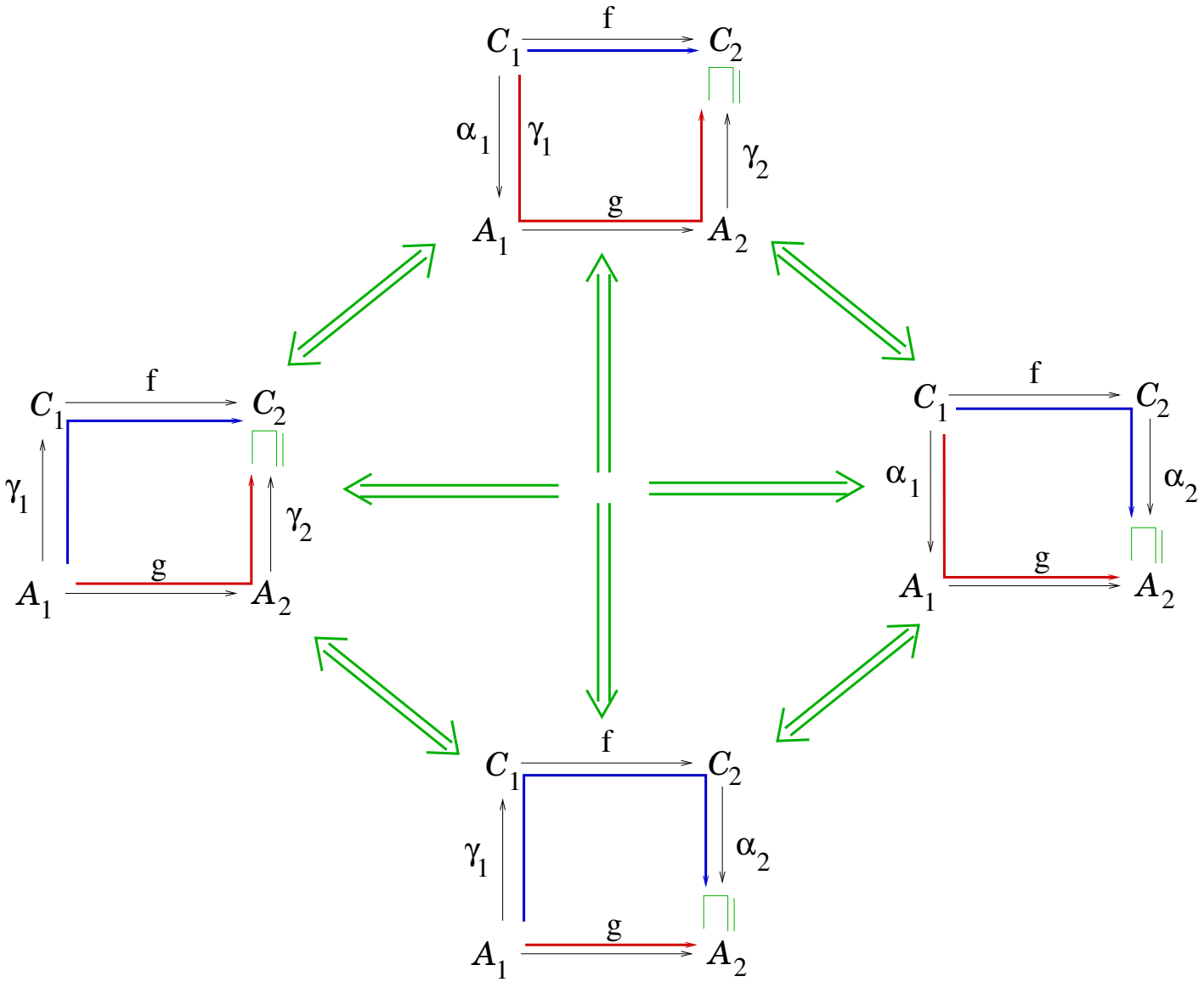
Sei $(f, g, (\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1), (\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2))$ ein Galois-System. Dann sind folgende Aussagen äquivalent:

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 15.4.3 (Charakterisierung)

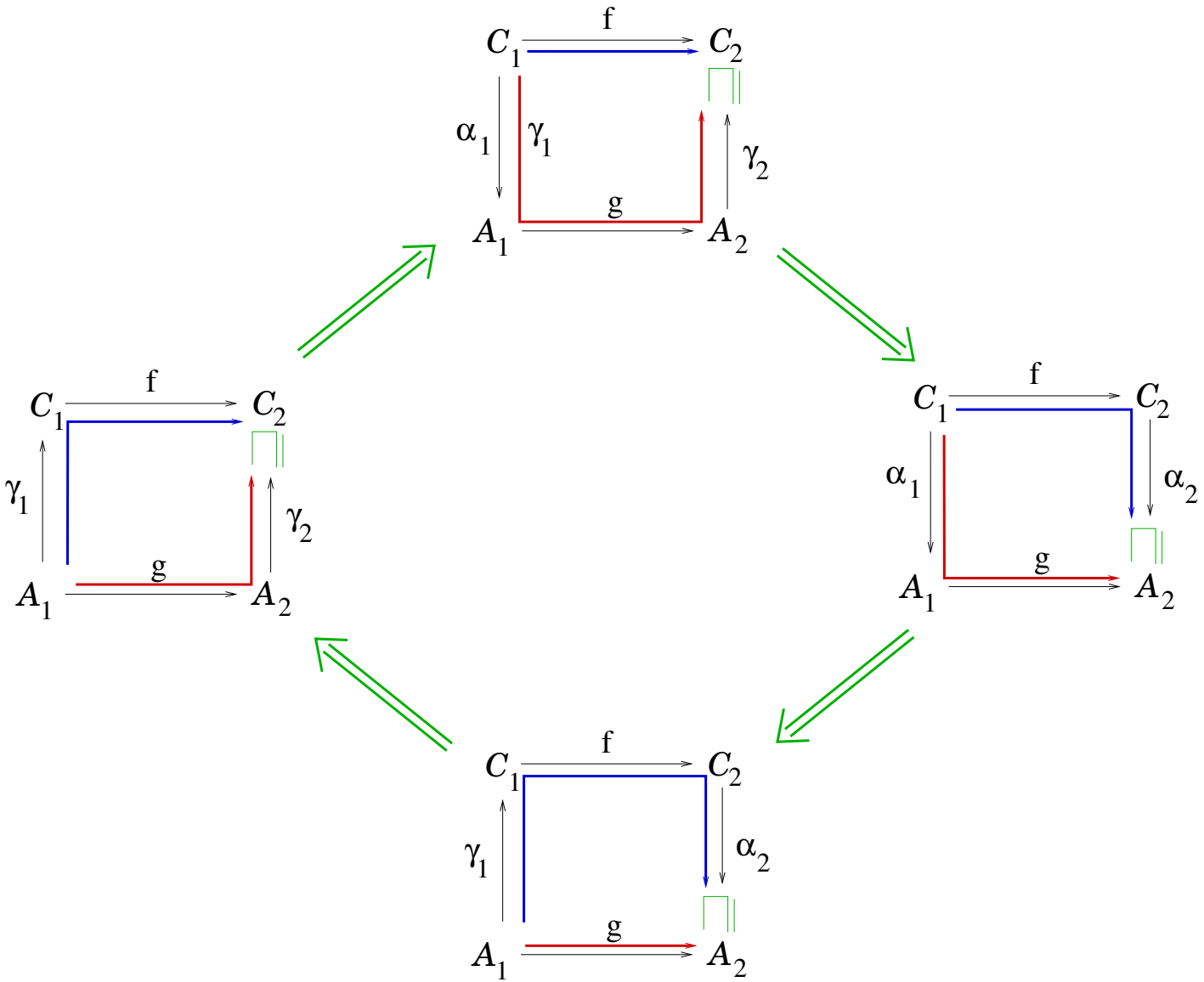
Lemma 15.4.2 gilt analog, wenn überall $\sqsubseteq$ durch $=$ ersetzt wird.

Illustration der Aussage von Lemma 15.4.2



Übungsaufgabe

Beweise Lemma 15.4.2 und 15.4.3 unter Ausnutzung folgender Idee:



Kapitel 15.5

Systeme abstrakter Interpretationen

Idee

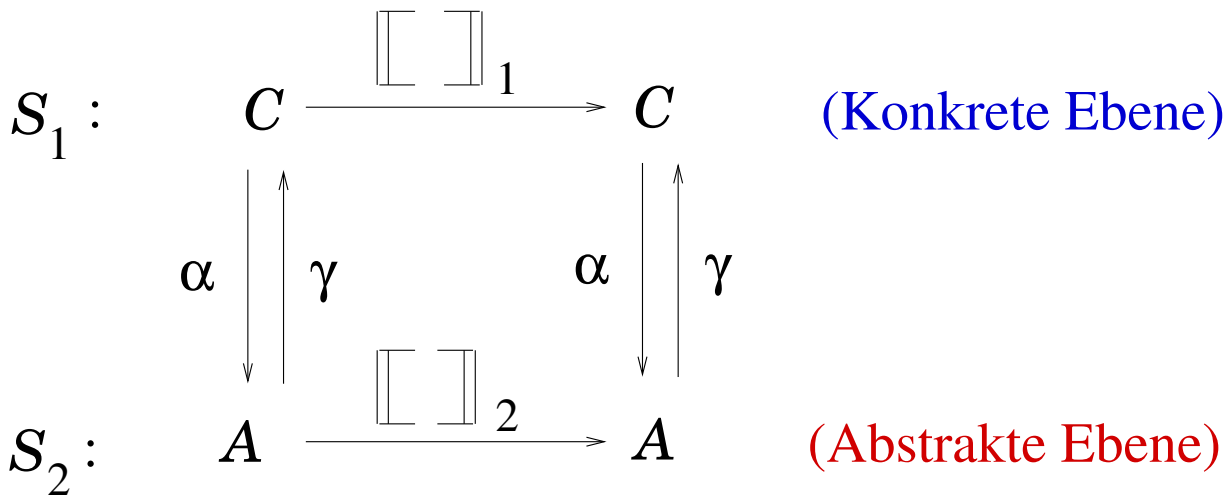
...ersetzen wir in einem Galois-System

$$(f, g, (\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1), (\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2))$$

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:

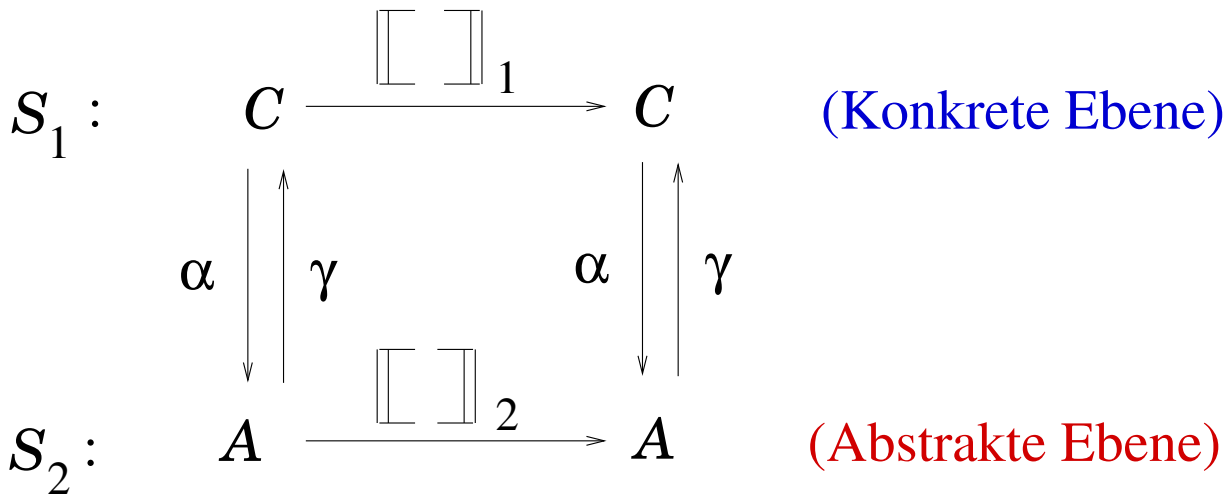


Systeme abstrakter Interpretationen

Definition 15.5.1 (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 7).

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 7) gilt:

$$[\![G]\!]_{\mathcal{S}}^{VUP/SUP} = [\![\mathbf{e}]\!]_{\mathcal{S}}^{MinFP/MaxFP}$$

Diese Beobachtung verknüpft die Theorie (und Praxis) der Datenflussanalyse (s. Kapitel 7) mit der Theorie (und Praxis) der abstrakten Interpretationen (s. Kapitel 15).

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

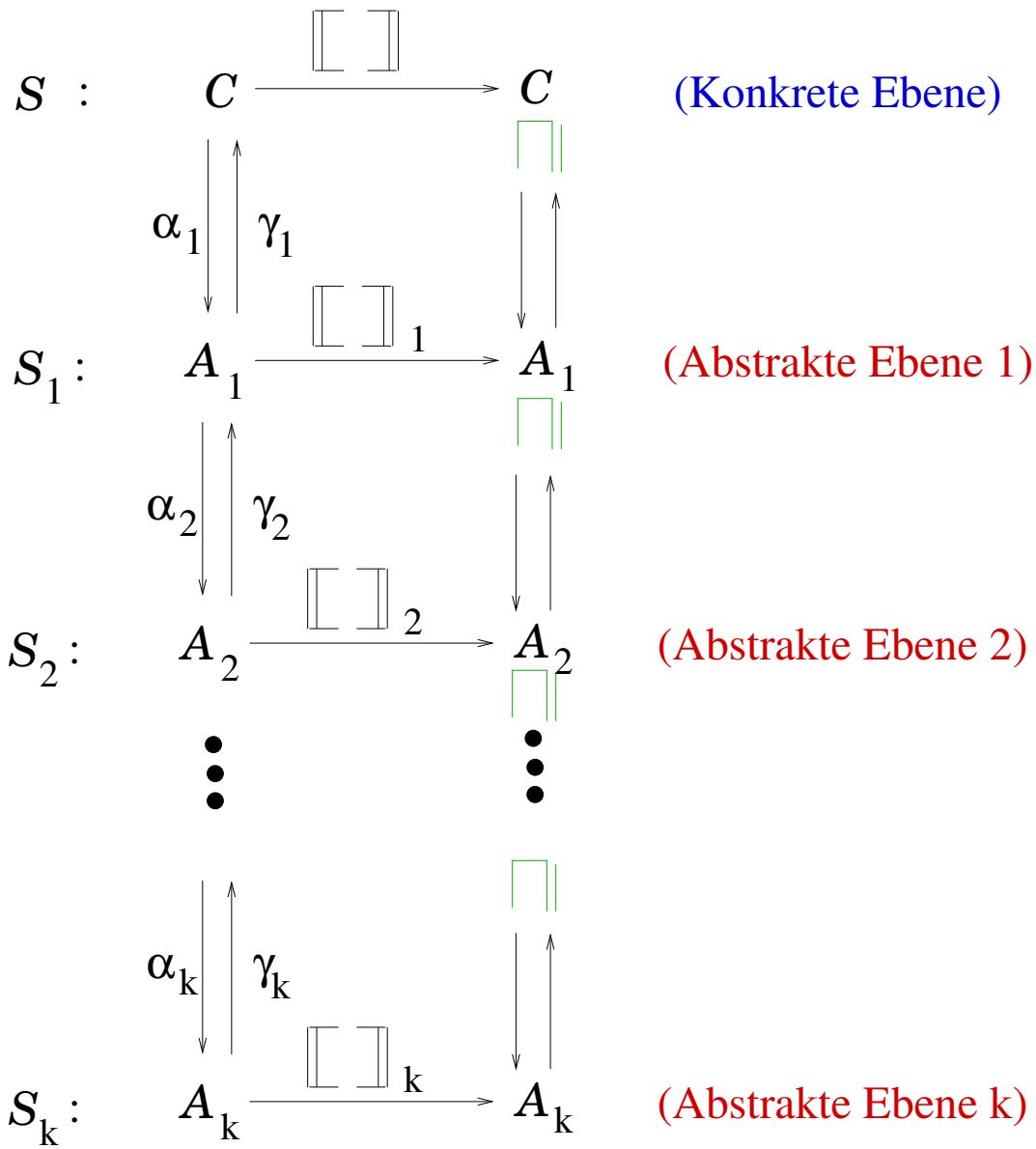
- Galois-Verbindungen oder (sogar) Galois-Passungen

verknüpft sind, so dass abstrakte Interpretationen niedrigerer Ebenen

- korrekt und (idealerweise) vollständig und optimal

bezüglich abstrakter Interpretationen höherer Ebenen sind (s. Kapitel 15.6 und 15.7).

Turm von Galois-Verbindungen/-Passungen (2)



Grundlage für Galois-Verb./-Passungstürme

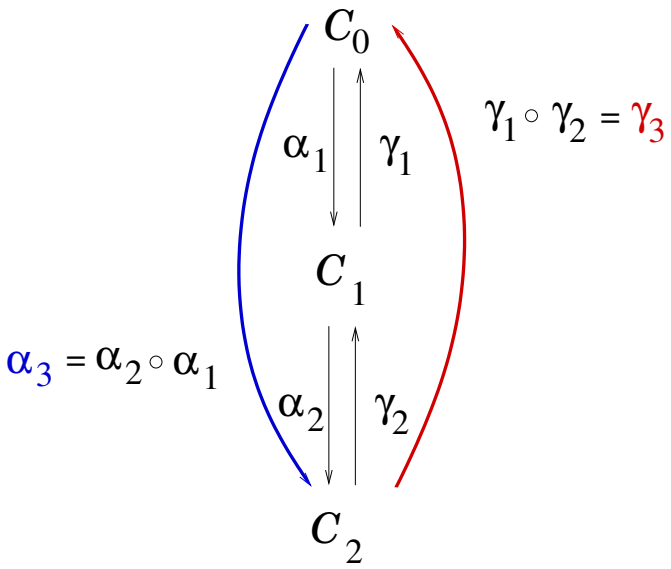
...ist die induktive Ausdehnung von:

Lemma 15.5.2 (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

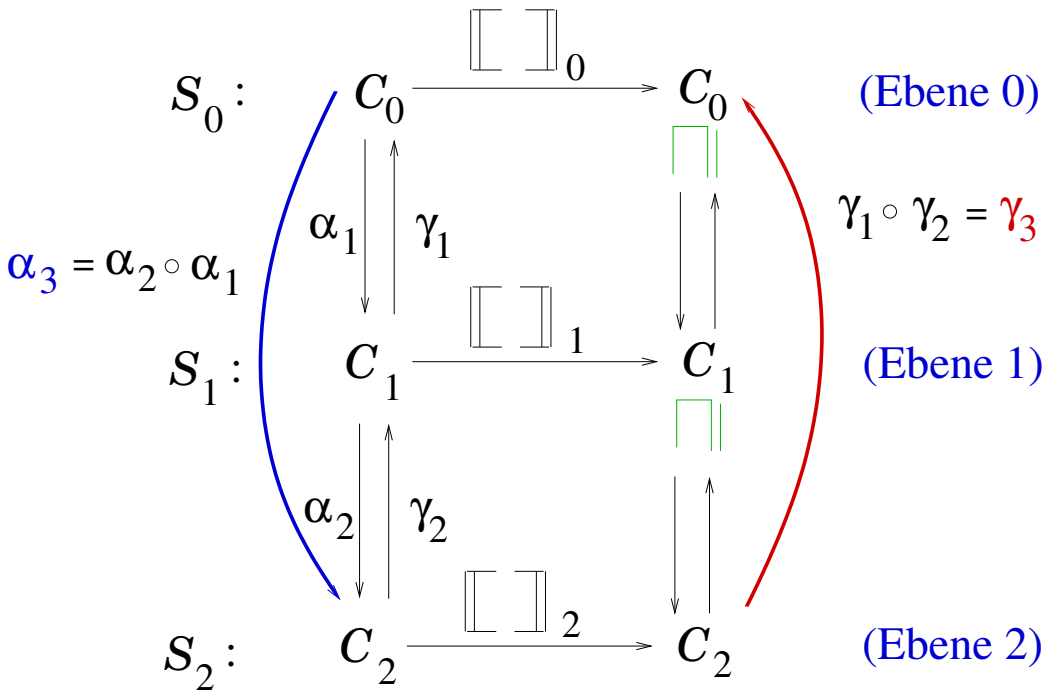
$$(\mathcal{C}_0, \alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2, \mathcal{C}_2)$$

eine Galois-Verbindung (Galois-Passung).



Anwendung

...von Lemma 15.5.2 auf Systeme abstrakter Interpretationen:



Kapitel 15.6

Korrektheit und Vollständigkeit abstrakter Interpretationen

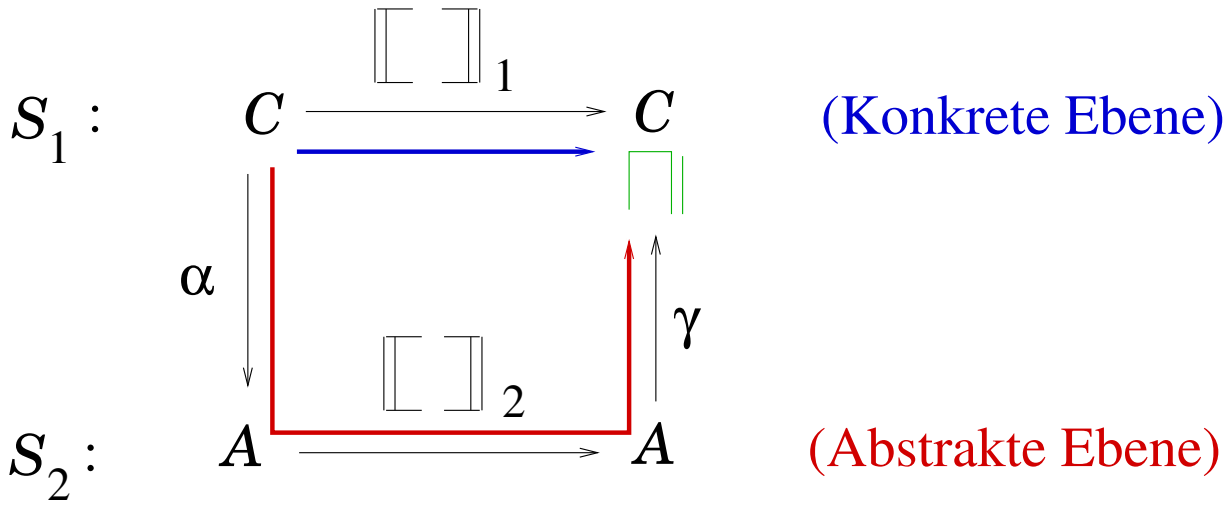
Korrektheits- und Vollständigkeitsbegriff

...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 15.6.1 (Korrektheit, Vollständigkeit)

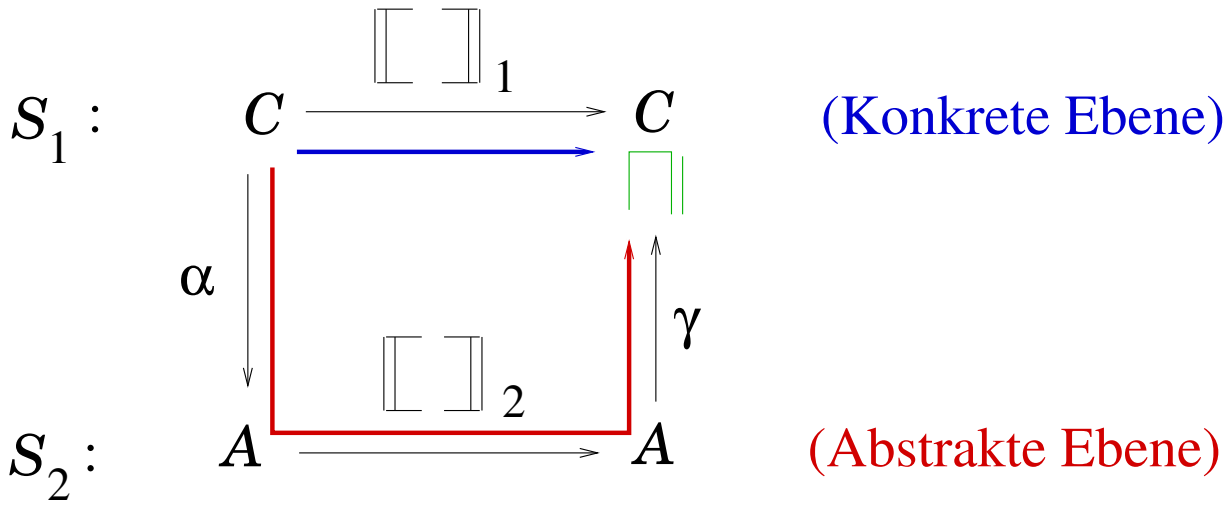
Sei $(\llbracket\ \rrbracket_1, \llbracket\ \rrbracket_2, (\mathcal{C}, \alpha, \gamma, \mathcal{A}))$ ein System abstrakter Interpretationen, wobei $\mathcal{S}_1 = (\mathcal{C}, \llbracket\ \rrbracket_1)$ und $\mathcal{S}_2 = (\mathcal{A}, \llbracket\ \rrbracket_2)$ zwei abstrakte Semantiken auf den vollständigen (Vereinigungs-) Halbverbänden $\mathcal{C}$ bzw. $\mathcal{A}$ sind.

Dann heißt $\mathcal{S}_2 = (\mathcal{A}, \llbracket\ \rrbracket_2)$

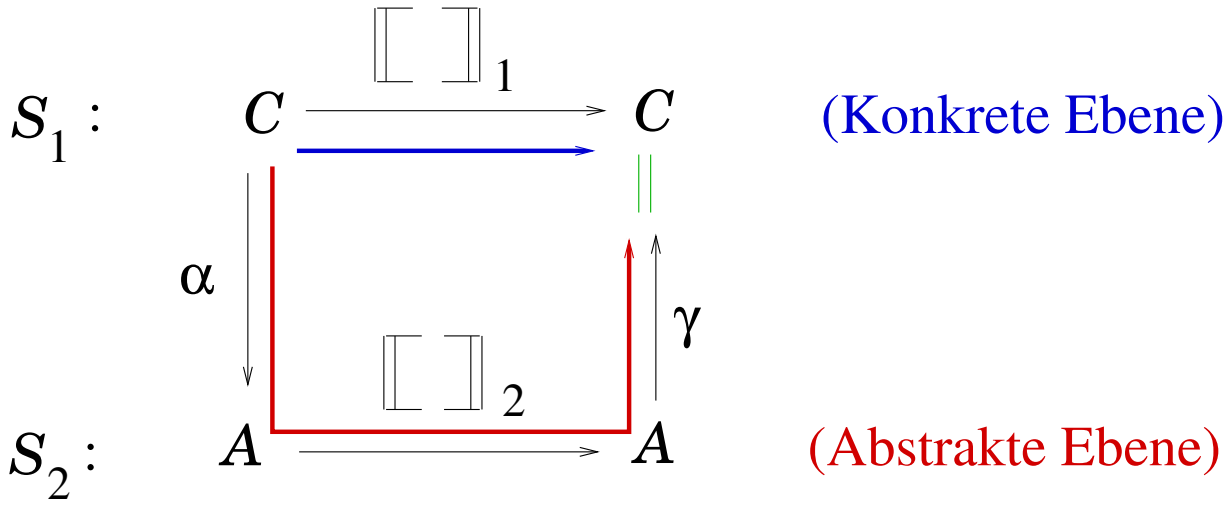
1. korrekt bzgl. $\mathcal{S}_1 = (\mathcal{C}, \llbracket\ \rrbracket_1)$, wenn das nachstehende Diagramm schwach kommutativ ist, d.h., für (mindestens) einige Elemente aus $\mathcal{C}$ die Inklusion echt ist.
2. vollständig bzgl. $\mathcal{S}_1 = (\mathcal{C}, \llbracket\ \rrbracket_1)$, wenn das nachstehende Diagramm (stark) kommutativ ist, d.h. die Inklusion für alle Elemente aus $\mathcal{C}$ unecht ist (also Gleichheit gilt).

Diagramme zu Definition 15.4.1

Korrektheitheit von $\mathcal{S}_2 = (\mathcal{A}, [\![\ ]\!]_2)$ bzgl. $\mathcal{S}_1 = (\mathcal{C}, [\![\ ]\!]_1)$:



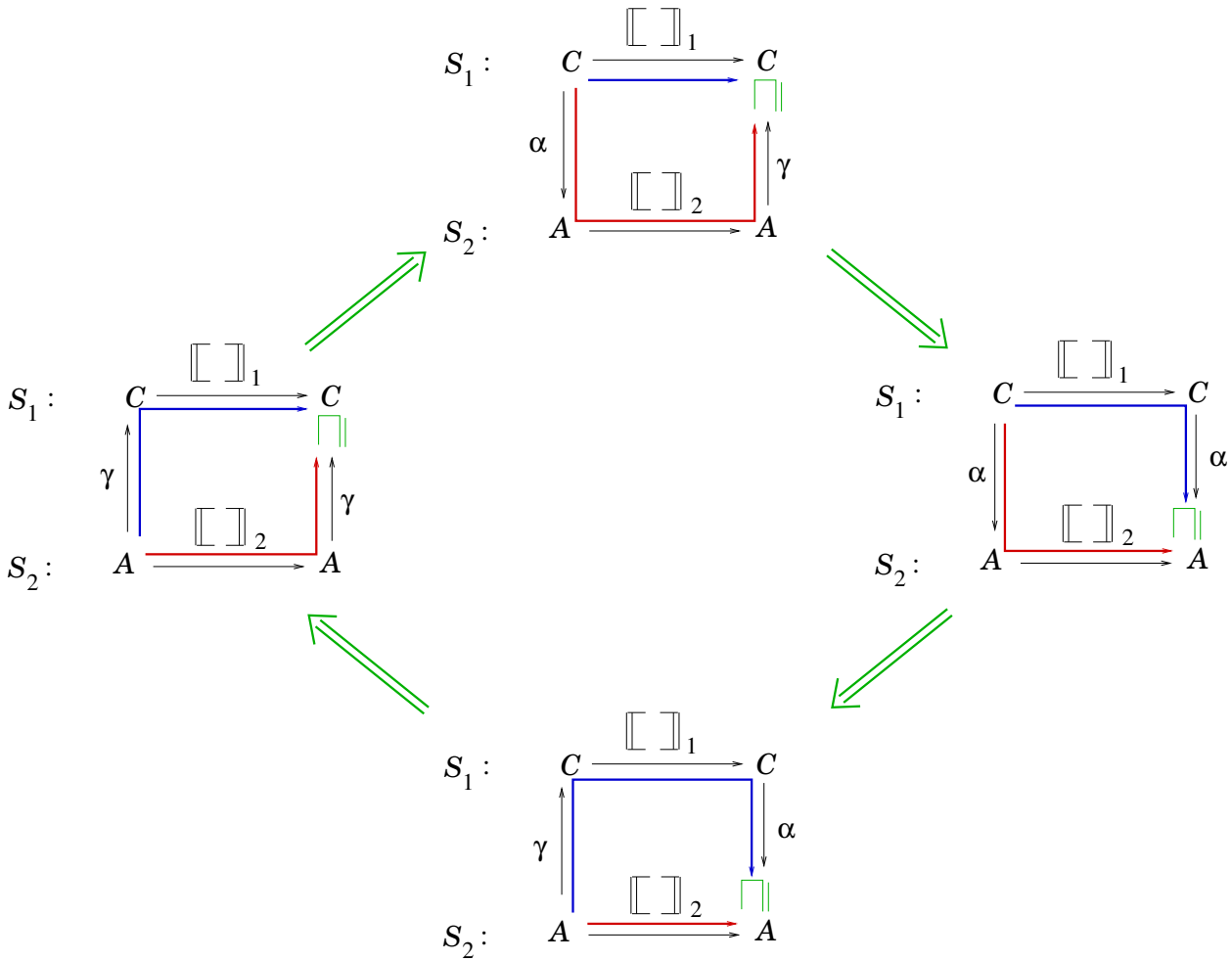
Vollständigkeit von $\mathcal{S}_2 = (\mathcal{A}, [\![\ ]\!]_2)$ bzgl. $\mathcal{S}_1 = (\mathcal{C}, [\![\ ]\!]_1)$:



Als unmittelbare Folgerung (1)

...aus Lemma 15.4.2 für allgemeine Galois-Systeme erhalten wir:

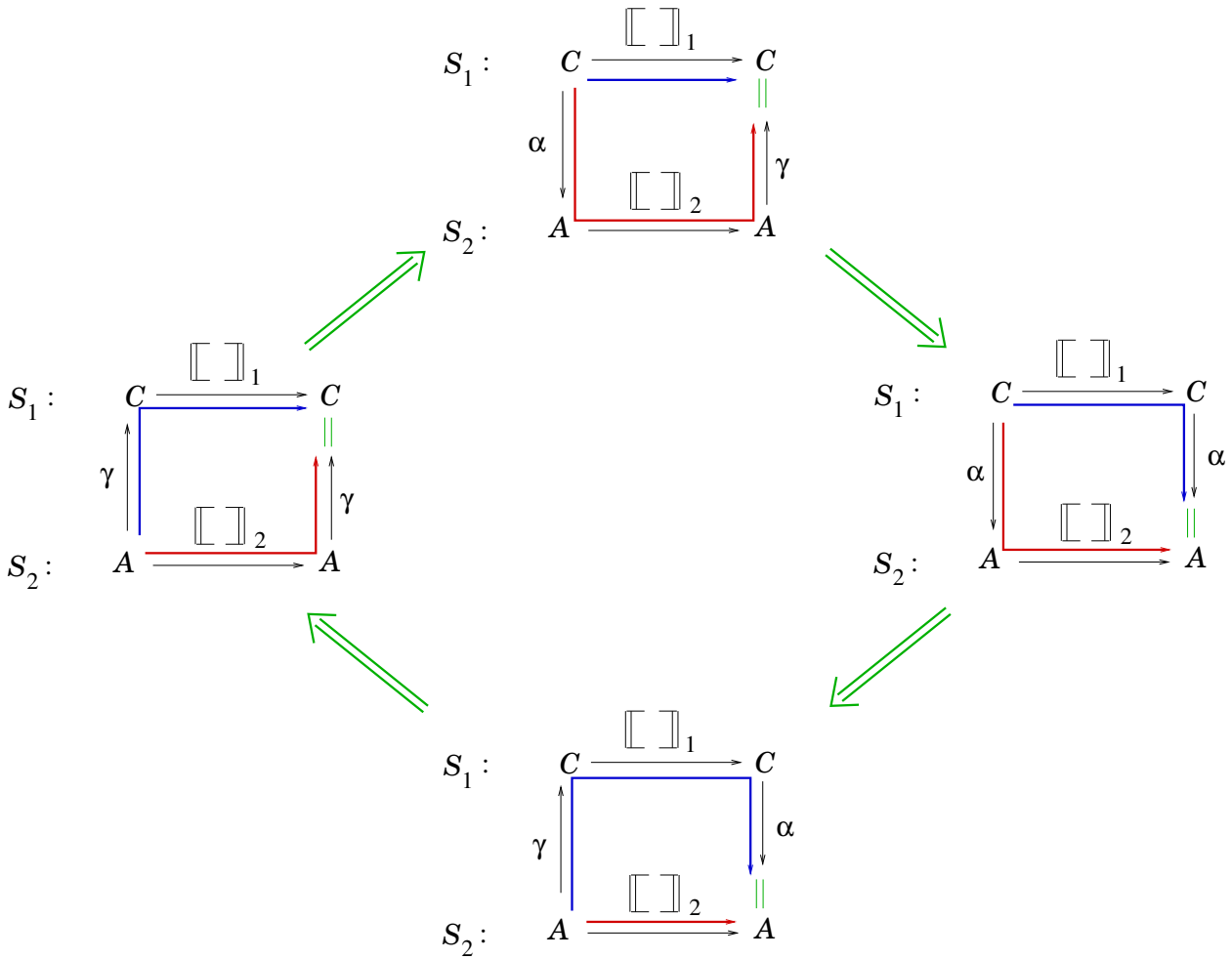
Korollar 15.6.2 (Korrektheit)



Als unmittelbare Folgerung (2)

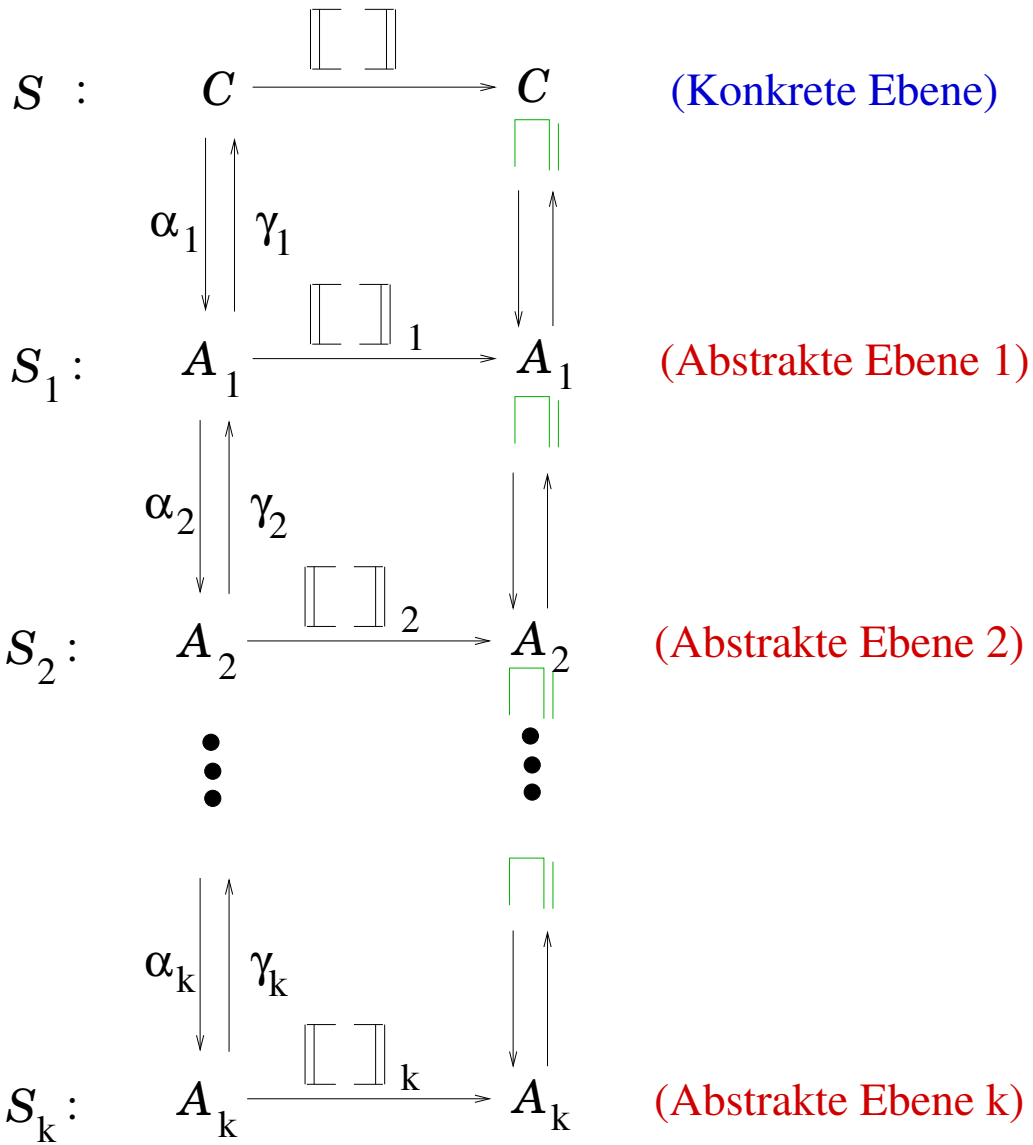
...aus Lemma 15.4.3 für allgemeine Galois-Systeme erhalten wir:

Korollar 15.6.3 (Vollständigkeit)



Verallg. von Korrektheit und Vollständigkeit

...auf Türme von Systemen abstrakter Interpretationen:



Kapitel 15.7

Optimalität abstrakter Interpretationen

Optimalität

...fragt intuitiv nach der

- 'einfachsten', 'abstraktesten', 'niedrigstebenigen' abstrakten Semantik

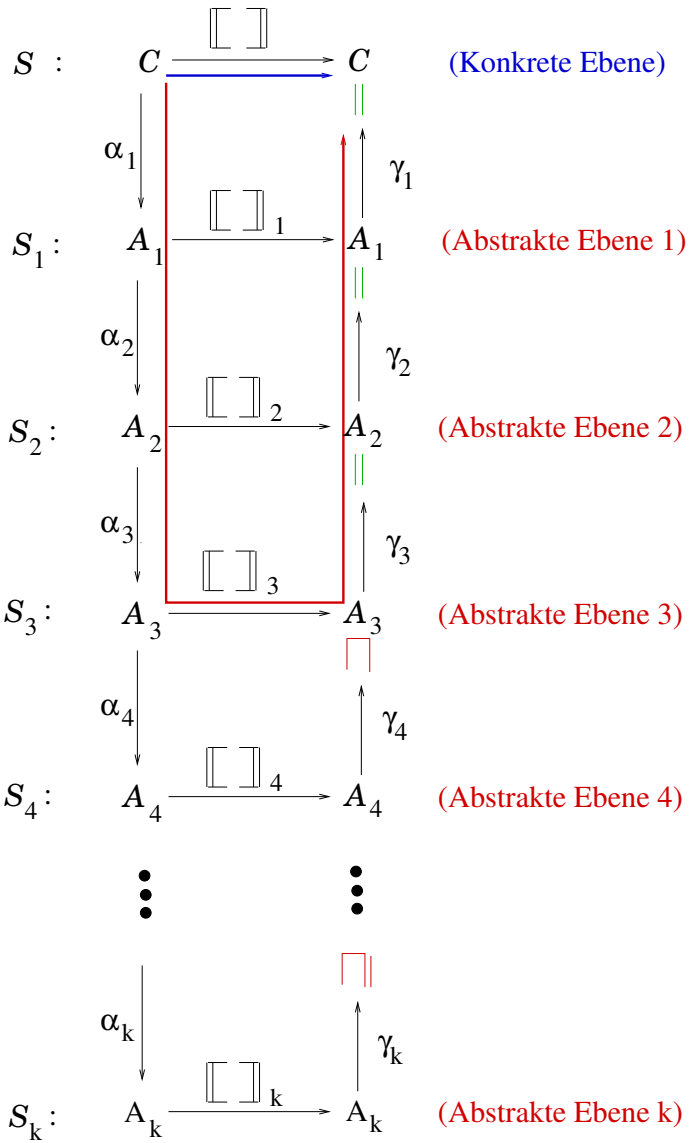
die eine Klasse $\mathcal{K}$ von Analysefrage zu beantworten erlaubt.

Dabei können intuitiv zwei Sichten unterschieden werden:

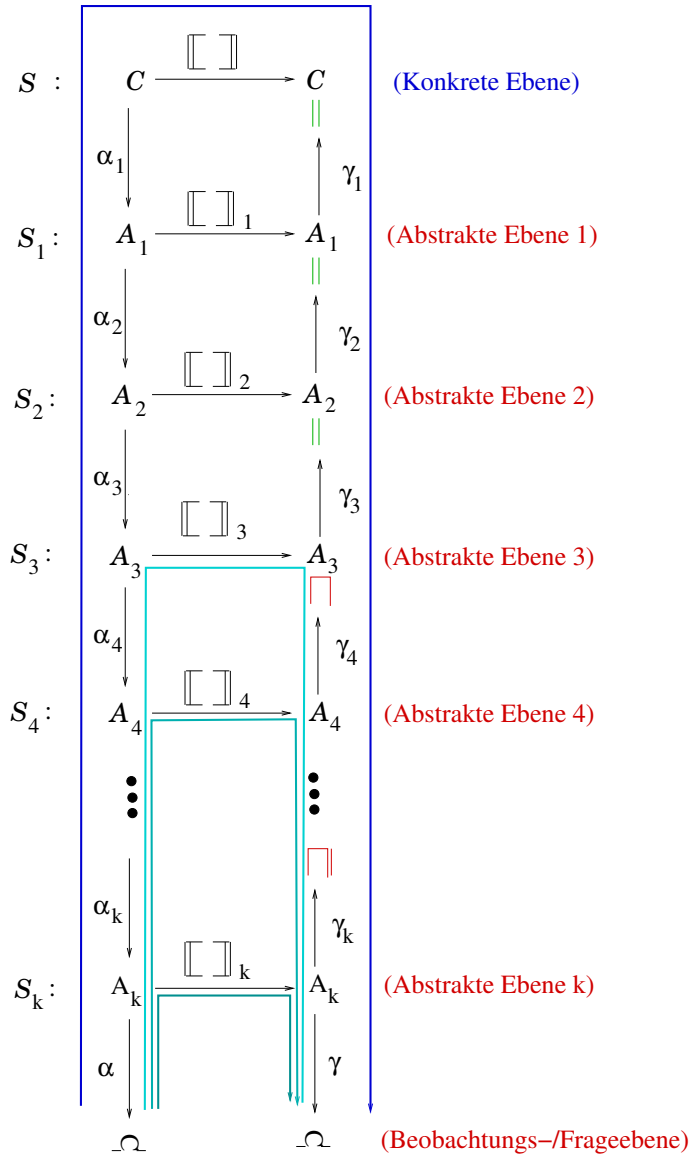
- Von oben nach unten: Wie weit kann abgestiegen werden, ohne Ausdruckskraft für $\mathcal{K}$ zu verlieren?
- Von unten nach oben: Wie weit muss aufgestiegen werden, um eine genügend ausdrucksstarke Analyse zu erreichen, um Analysefragen aus $\mathcal{K}$ zu beantworten?

Illustration

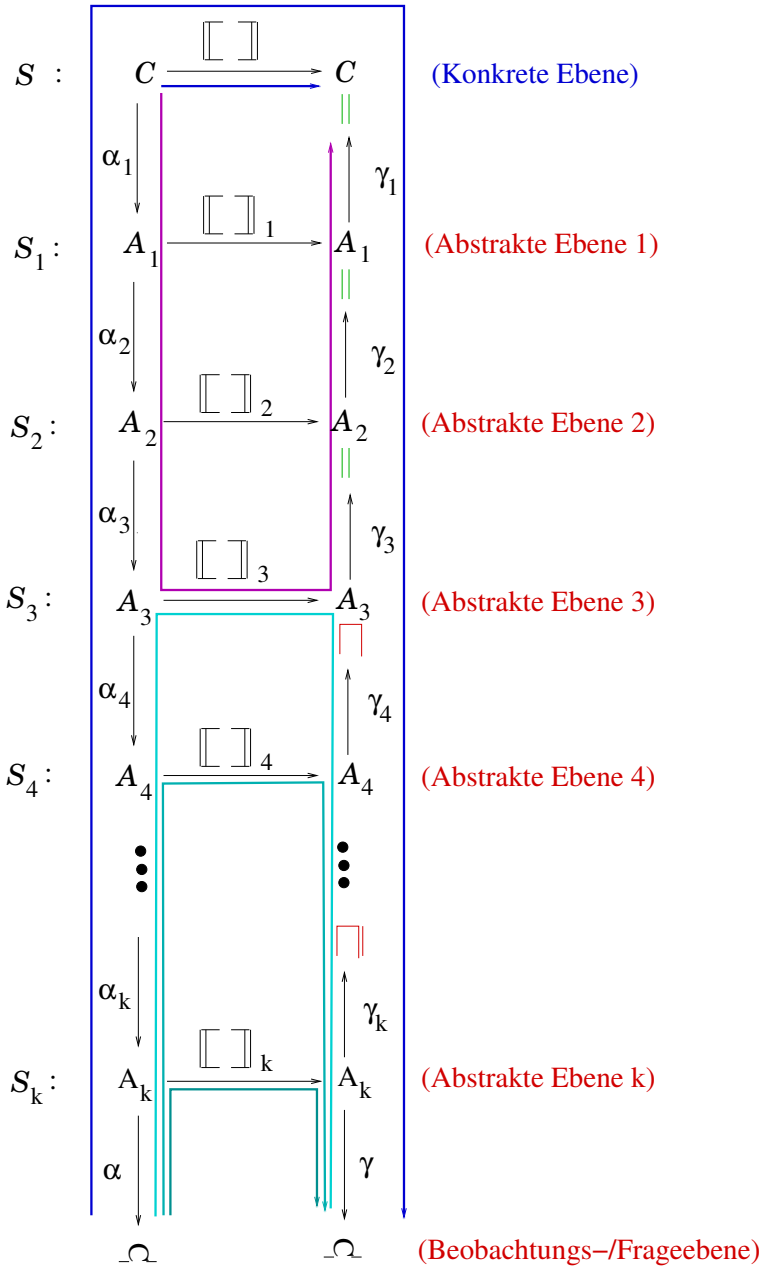
Von oben nach unten:



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 15.7.1 (onu-Optimalität)

Sei $\mathcal{K}$ eine Klasse von Analysefragen und $\mathcal{S}_1 = (\mathcal{C}, [\![\ ]\!]_1)$ ausdruckskräftig für $\mathcal{K}$.

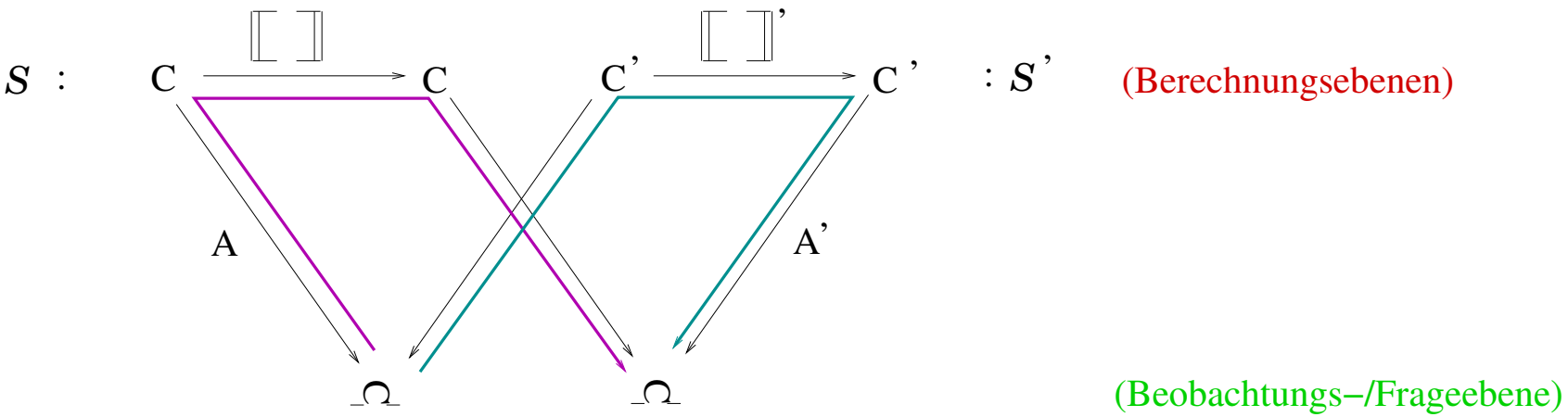
Dann ist $\mathcal{S}_1$ onu-optimal für $\mathcal{K}$, wenn alle für ‘$\mathcal{K}$ ausdruckskräftigen Abstraktionen von $\mathcal{S}_1$ zu $\mathcal{S}_1$ isomorph’ sind.

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.



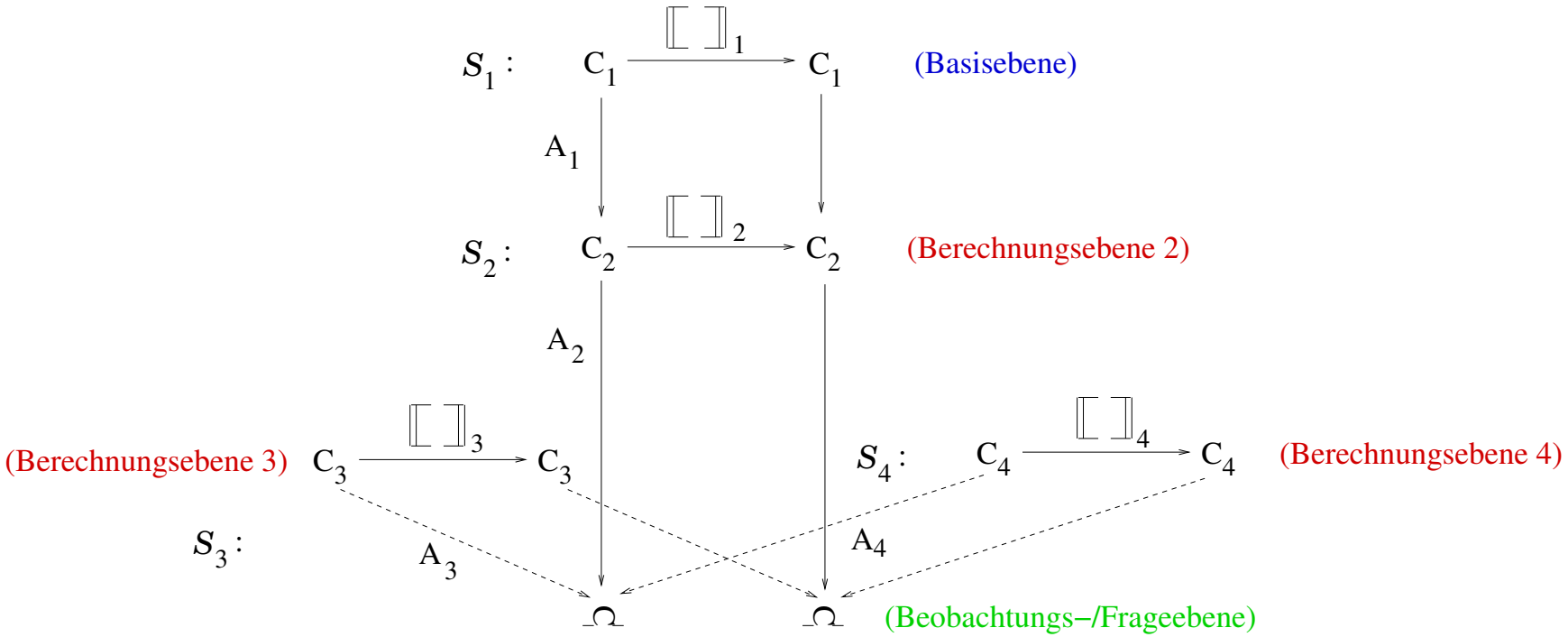
(Berechnungsebenen)

(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 15.7.2 (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 15.7.3 (Globale abstrakte Semantik)

Sei $(\llbracket\ \rrbracket_I, \mathcal{C})$ abstrakte Interpretation und $\llbracket\ \rrbracket : \mathbf{FG} \to (\mathcal{C} \to \mathcal{C})$ die Globalisierung von $\llbracket\ \rrbracket_I$, d.h. $\forall\, G \in \mathbf{FG}\ \forall\, c \in \mathcal{C}$ gilt:

$$\llbracket\, G\, \rrbracket(c) =_{df}$$

$$\begin{cases} \llbracket\, n_k\, \rrbracket_I \circ \ldots \circ \llbracket\, n_1\, \rrbracket_I(c) & \text{falls } G = (n_1, \ldots, n_k) \in \mathbf{LFG} \\ \bigsqcup\{\llbracket\, P\, \rrbracket(c) \mid P \in \mathbf{P}(G)\} & \text{sonst} \end{cases}$$

Dann heißt das Paar $(\llbracket\ \rrbracket, \mathcal{C})$ die von $(\llbracket\ \rrbracket_I, \mathcal{C})$ induzierte (globale) abstrakte Semantik.

Abstraktion und Konkretisierung

Definition 15.7.4 (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.

- 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.
- 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 8 die Paare aus Datenfluss- und reversen Datenflussanalysefunktionen).
- Die Konkretisierungsfunktion A^a ist monoton, i.a. aber nicht additiv.
- Mit den Bezeichnungen aus Kap. 15.2 entsprechen sich α und A und γ und A^a.

Isomorphie abstrakter Semantiken

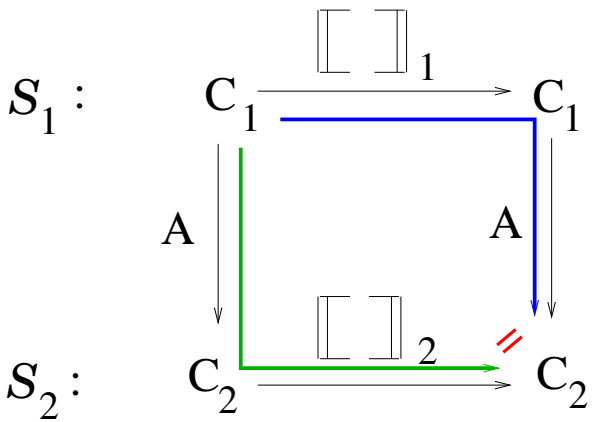
Definition 15.7.5 (Isomorphie)

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 15.7.6 (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.

Modell

Definition 15.7.7 (Modell)

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.

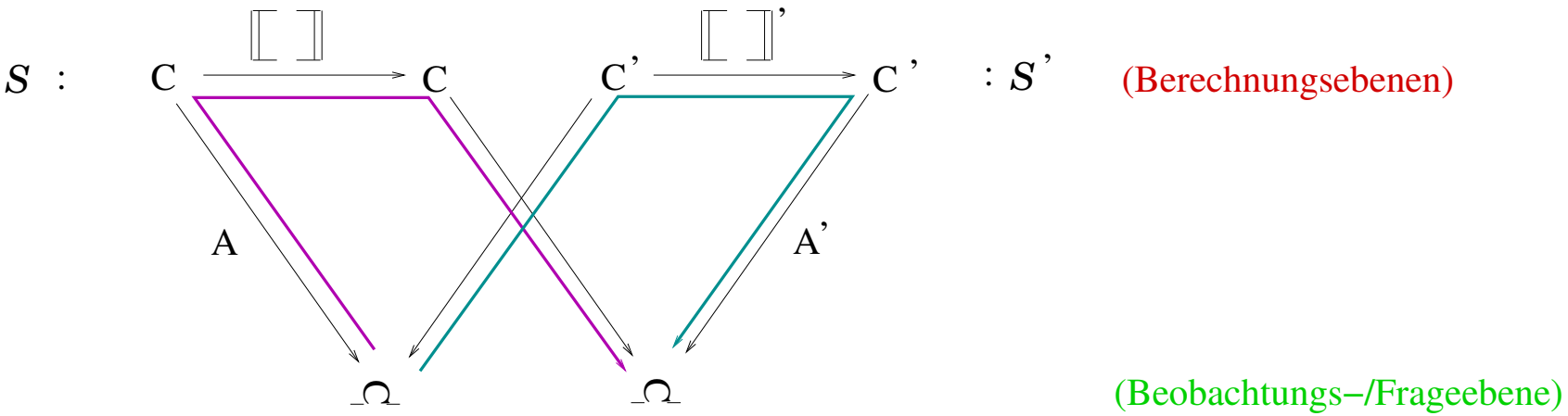
Beobachtungsäquivalenz

Definition 15.7.8 (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 15.7.9 (Äquivalenzrelation)

Die Relation $\approx_\Omega$ ist eine Äquivalenzrelation auf der Menge aller Modelle von Ω.

Verbandshüllen

Definition 15.7.10 (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 15.7.11 (Lokale Optimalität)

Sei $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$ und $\mathcal{S}_2 \rightarrow_{A_2} \Omega$. 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 15.7.12 (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 15.7.13 (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 15.7.14 (Abschneidetheorem)

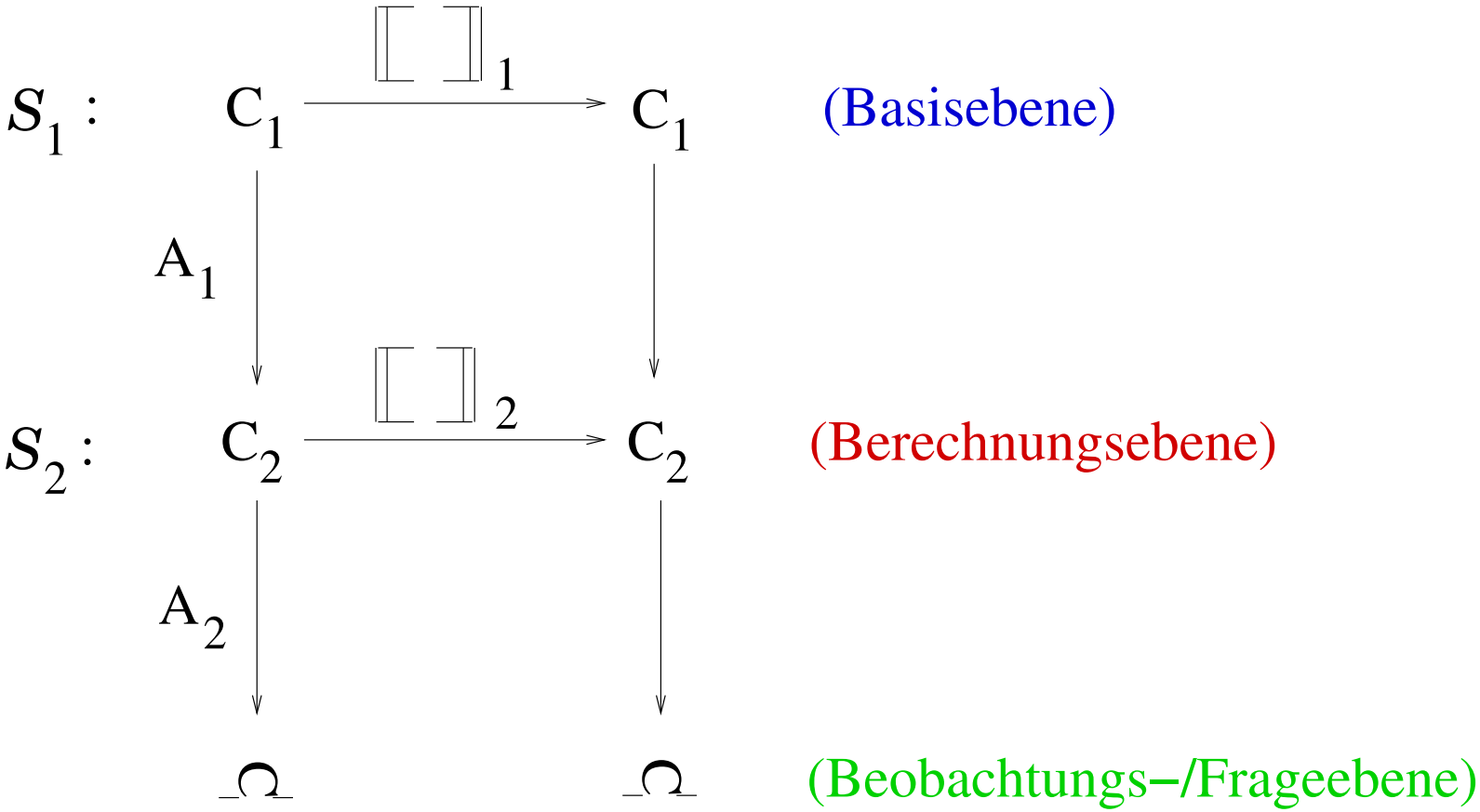
Sei $(\mathcal{S}_2, A_2) \in \Phi(\mathcal{S}_1, A_2 \circ A_1, \Omega)$ mit $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$. Dann gilt:

$$\forall\, G \in \mathbf{FG}.\ A_1 \circ [\![\ G\ ]\!]_1 \circ A_1^a = [\![\ G\ ]\!]_2$$

Insbesondere gilt weiters:

$$(\mathcal{S}_1, A_2 \circ A_1) \approx_\Omega (\mathcal{S}_2, A_2)$$

Veranschaulichung



Äquivalenz

Theorem 15.7.15 (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 dem Existenz- und Eindeutigkeitstheorem 15.5.12 und dem Abschneidetheorem 15.5.13 liefert das Äquivalenztheorem 15.5.14, 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

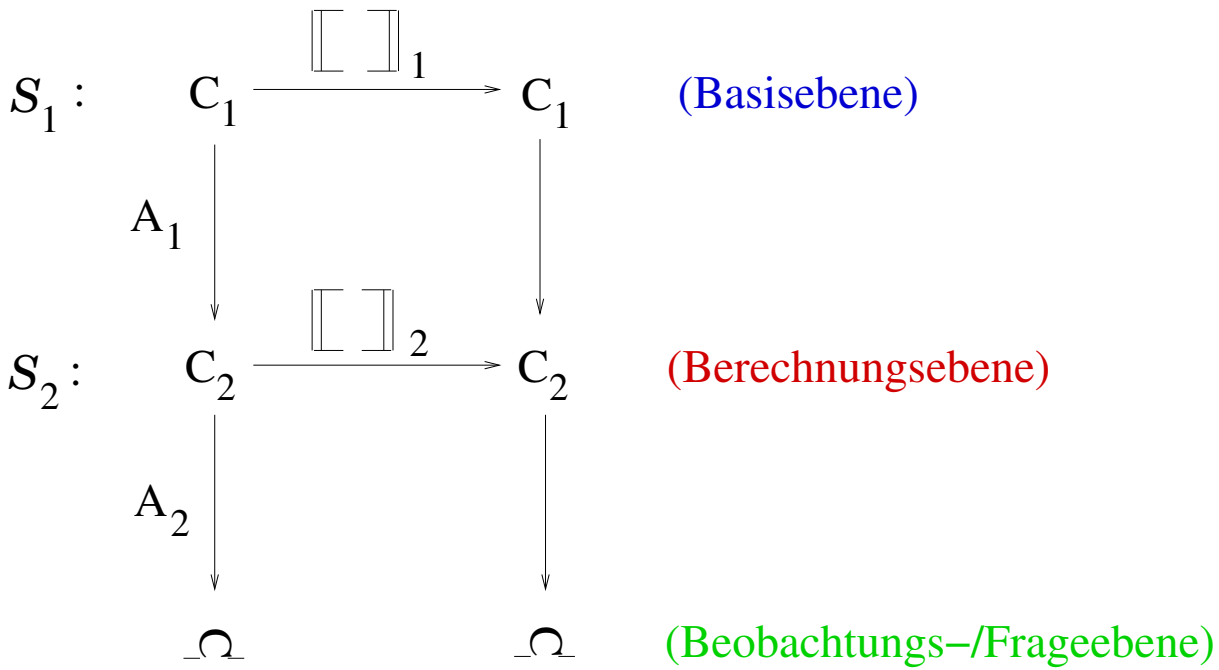
- korrekte, vollständige und optimale Modell.

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 15.8

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (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 15 (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 15 (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 15 (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 15 (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 15 (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 15 (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 15 (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 16

Modellprüfung und Datenflussanalyse

Kapitel 16.1

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 16.2

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 16.3

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

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 16.4

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):

$$NEC =_{df} (\neg(Mod \vee end))\ \mathbf{U}\ Used$$

Frühestheit (Wert kann nicht früher bereitgestellt werden):

$$EAR =_{df} start \vee \neg(\overline{[.]}((\neg(Mod \vee start))\ \overline{\mathbf{U}}\ (NEC \wedge \neg Mod)))$$

Berechnungspunkte: $OCP =_{df} EAR \wedge NEC$

PREE-Optimierungstransformation OT:

1. Deklariere eine frische Hilfsvariable h_t für t.
2. Initialisiere h_t an allen Programmpunkten in OCP mit $h_t := t$.
3. Ersetze alle Vorkommen von t im Programm durch h_t.

Korrektheit und Optimalität

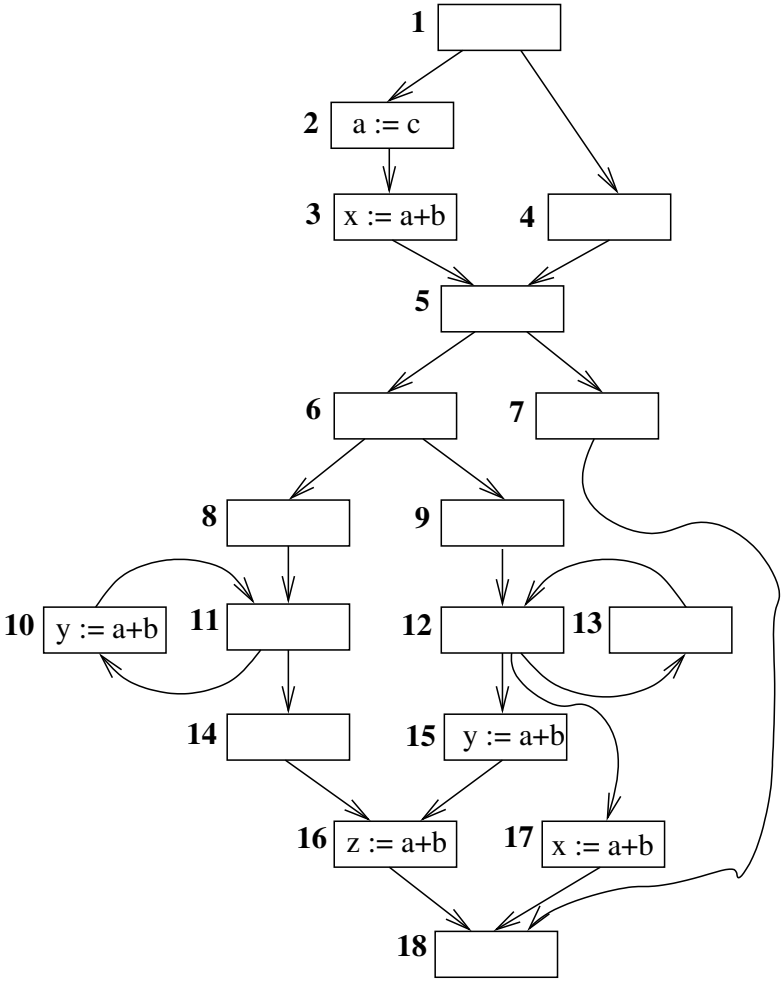
Theorem 16.4.1 (Korrektheit und Optimalität)

OT ist korrekt (d.h. semantikerhaltend) und optimal (d.h. mindestens so gut wie jede andere korrekte Platzierung der Berechnungen von t).

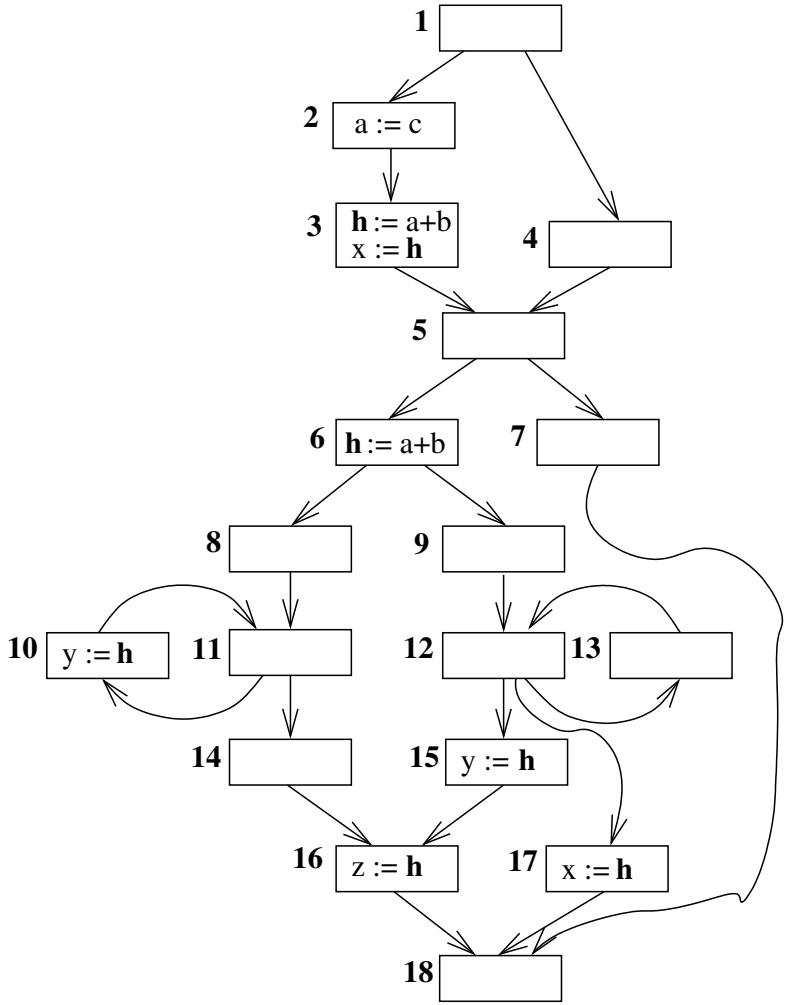
Beachte: OT entspricht der 'busy code motion'-Transformation für die Elimination partiell redundanter Berechnungen (für Details s. LVA 185.A04 Optimierende Compiler, Kapitel 7; auch zur formalen Definition von 'korrekt' und 'optimal').

Beispiel: Anwendung von OT

Ausgangsprogramm



OT-optimiertes Programm



Bem.: OT nimmt kanten-, nicht knotenbenannte Graphen an.

Kapitel 16.5

Zusammenfassung

Zusammenfassung (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 (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 16.6

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (1)

- 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.
- 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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (2)


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.


E. Allen Emerson. *Temporal and Modal Logic*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Hrsg.), Elsevier, 995-1072, 1990.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (3)


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 16 (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 16 (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.
- 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 16 (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 16 (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 17

Modellprüfung und Abstrakte Interpretation

Kapitel 17.1

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 17.2

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (1)


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.


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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (2)

- 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.
- 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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (3)

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.

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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (4)


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.


Kenneth L. MacMillan. *Symbolic Model Checking*. Kluwer, 1993.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (5)


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 17 (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 VI

Abschluss und Ausblick

Kapitel 18

Resümee, Perspektiven

Kapitel 18.1

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

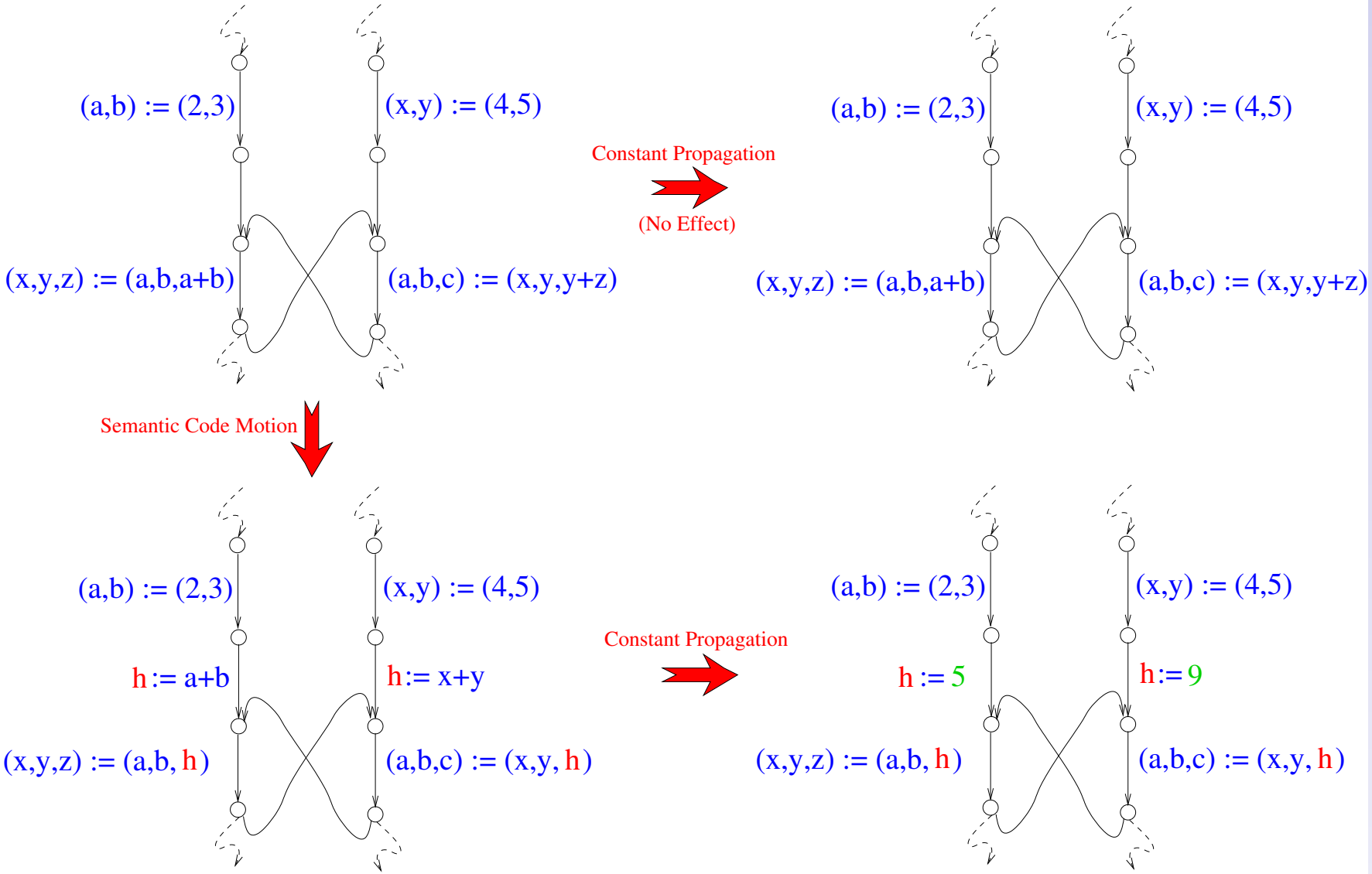
von Transformationen einschließlich Theorien und Techniken zur Evaluation mit Anwendungen insbesondere im

- Übersetzerbau (Optimierung, Parallelisierung, Portabilität, Mehrfachziele (Performanz, Speicher, Energie), Sicherheit, Schutz, Privatsphäre,...)
- Software-Technik (Modellbasierte Entwicklung und Code-Erzeugung, Refaktorisierung,...)

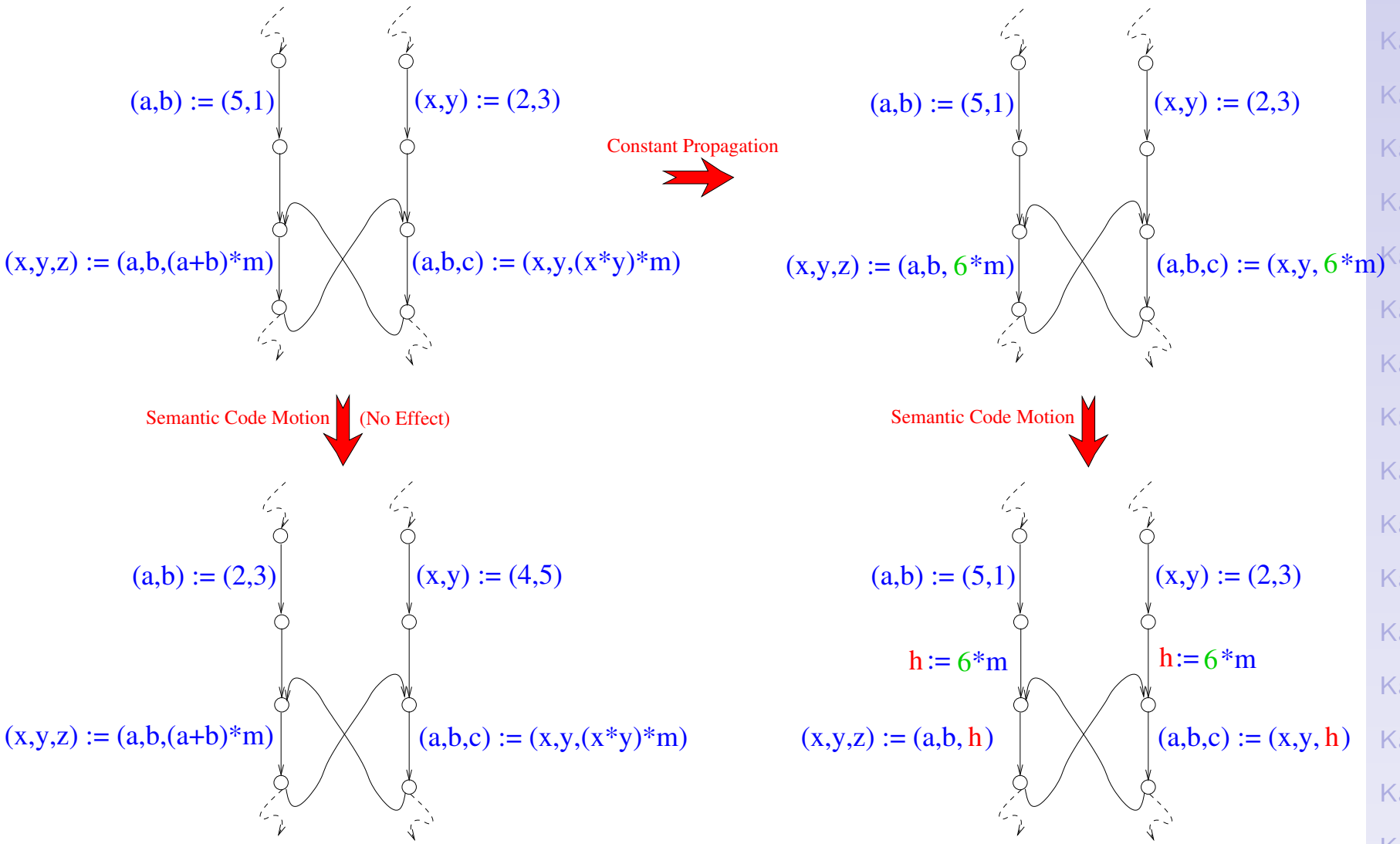
...und darüber hinaus.

Illustriert anhand v. Programmoptimierung (1)

...am Beispiel des Zusammenspiels von Optimierungen:

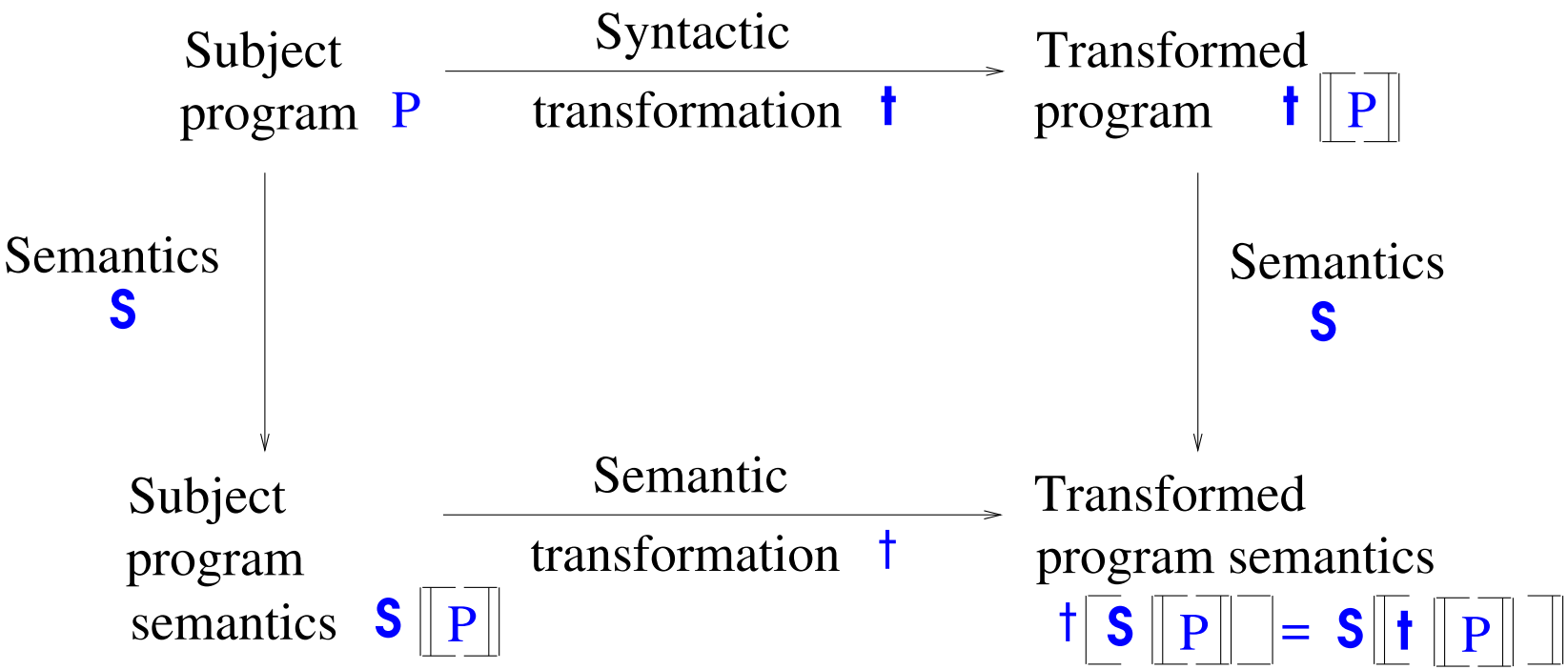


Illustriert anhand v. Programmoptimierung (2)



Analyse, Verifikation und Transformation

...bewiesen (beweisbar) korrekt und vollständig/optimal in einem einheitlichen Rahmen:



(Patrick und Radhia Cousot, POPL 2002)

Wichtig für verschiedenste Gebiete (1)

...und Felder im Bereich und Umfeld von Hard- und Software–Entwicklung, -Analyse, -Verifikation und -Anwendung.

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,...)

Wichtig für verschiedenste Gebiete (1)

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,...)

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.

Chapter 18.2

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (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 18 (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 18 (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.

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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (4)


Flemming Nielson. *Program Transformations in a Denotational Setting*. ACM Transactions on Programming Languages and Systems 7:359-379, 1985.


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 18 (5)

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)


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.


Garret Birkhoff. *Lattice Theory*. American Mathematical Society, 3. Auflage, 1967.

I Lehrbücher (3)


Edmund M. Clarke, Orna Grumberg, Doron Peled. *Model Checking*. MIT Press, 2001.


Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004.


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.


Martin Davis. *Computability and Unsolvability*. Dover Publications, 1982.


Gilles Dowek. *Principles of Programming Languages*. Springer-V, 2009.

I Lehrbücher (4)


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, 2. Auflage, 2003.


Paul R. Halmos. *Naive Set Theory*. Springer-V., Wiederauflage, 2001.


Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.

I Lehrbücher (5)


Hans Hermes. *Einführung in die Verbandstheorie*. Springer-V., 2. Auflage, 1967.


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.


Richard Johnsonbaugh. *Discrete Mathematics*. Pearson, 7. Auflage, 2009.


Uday P. Khedker, Amitabha Sanyal, Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press, 2009.

I Lehrbücher (6)


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.


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.

I Lehrbücher (7)


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.


Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997.


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.

I Lehrbücher (8)


Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. Springer-V., 2. Auflage, 2005.


Doron A. Peled. *Software Reliability Methods*. Springer-V., 2001.


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 (9)

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.


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.


Orna Grumberg, Helmut Veith (Hrsg.). *25 Years of Model Checking: History, Achievements, Perspectives*. Springer-V., LNCS 5000, 2008.

III Sammelbände (2)

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.


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.


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.

V Artikel (5)

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.

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.

V Artikel (6)

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.

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 (7)


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 (8)


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.


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 (9)


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.


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.

V Artikel (10)


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.


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.

V Artikel (11)

- 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.
- 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.

V Artikel (12)


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.


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.

V Artikel (13)


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.


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.

V Artikel (14)


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.


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.

V Artikel (15)


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.


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.

V Artikel (16)


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.


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.

V Artikel (17)


Patrick Cousot, Michael Monerau. *Probabilistic Abstract Interpretation*. In Proceedings of the 21st Symposium on Programming (ESOP 2012), Springer-V., LNCS 7211, 169-193, 2012.


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.


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.

V Artikel (18)


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.


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.

V Artikel (19)


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.


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.

V Artikel (20)


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 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.

V Artikel (21)


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.


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.


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.

V Artikel (22)


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.


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.

V Artikel (23)


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.


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.

V Artikel (24)


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.


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.

V Artikel (25)


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, 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 (26)


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 (27)


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 Ideal of Program Correctness*. The Computer Journal 50(3):254-260, 2007.


Susan Horwitz, Alan J. Demers, Tim Teitelbaum. *An Efficient General Iterative Algorithm for Dataflow Analysis*. Acta Informatica 24(6):679-694, 1987.

V Artikel (28)


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.


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.

V Artikel (29)


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`


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.

V Artikel (30)


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 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.

V Artikel (31)


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. *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.

V Artikel (32)


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.


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.

V Artikel (33)


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. *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.

V Artikel (34)


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Optimal Code Motion: Theory and Practice*. ACM Transactions on Programming Languages and Systems 16(4):1117-1155, 1994.


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.

V Artikel (35)


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. (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.

V Artikel (36)


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.


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.

V Artikel (37)


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.


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`

V Artikel (38)


Dexter Kozen, Jerzy Tiuryn. *On the Completeness of Propositional Hoare Logic*. Information Sciences 139(3-4):187-195, 2001.


Jean-Louis Lassez, V.L. Nguyen, Liz Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.


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.

V Artikel (39)


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.


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.

V Artikel (40)


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.


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.

V Artikel (41)


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.


Yuri V. Matijasevic. *Enumerable Sets are Diophantine (In Russian)*. Dodl. Akad. Nauk SSSR 191, 279-282, 1970.


Yuri V. Matijasevic. *What Should We Do Having Proved a Decision Problem to be Unsolvable?* Algorithms in Modern Mathematics and Computer Science 1979:441-448, 1979.

V Artikel (42)


Yuri V. Matijasevic. *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 (43)


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 (44)


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 (45)


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 (46)


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 (47)


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, 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, 1982.


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

V Artikel (48)


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.


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.

V Artikel (49)


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.


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.

V Artikel (50)


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.


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.

V Artikel (51)


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.


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.

V Artikel (52)


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.


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.

V Artikel (53)


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.


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.

V Artikel (54)


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.


Bernhard Steffen, Jens Knoop. *Finite Constants: Characterizations of a New Decidable Set of Constants*. Theoretical Computer Science 80(2):303-318, 1991.

V Artikel (55)


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.


Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5(2):285-309, 1955.

V Artikel (56)


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.


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.

V Artikel (57)


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.


Mark N. Wegman, F. Kenneth Zadeck. *Constant Propagation With Conditional Branches*. ACM Transactions on Programming Languages and Systems 13(2):181-201, 1991.

V Artikel (58)


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, 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 (59)


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

Relations

Let M_i, $1 \leq i \leq k$, be sets.

Definition A.1.1 (k-ary Relation)

A (k-ary) relation is a set R of ordered tuples of elements of $M_1, \ldots, M_k$, i.e., $R \subseteq M_1 \times \ldots \times M_k$ is a subset of the cartesian product of the sets M_i, $1 \leq i \leq k$.

Examples

- $\emptyset$ is the smallest relation on $M_1 \times \ldots \times M_k$.
- $M_1 \times \ldots \times M_k$ is the biggest relation on $M_1 \times \ldots \times M_k$.

Binary Relations

Let M, N be sets.

Definition A.1.2 (Binary Relation)

A (binary) relation is a set R of ordered pairs of elements of M and N, i.e., R is a subset of the cartesian product of M and N, $R \subseteq M \times N$, called a relation from M to N.

Examples

- $\emptyset$ is the smallest relation from M to N.
- $M \times N$ is the biggest relation from M to N.

Note

- If R is a relation from M to N, it is common to write $m\,R\,n$, $R(m, n)$, or $R\,m\,n$ instead of $(m, n) \in R$.

Between, On

Definition A.1.3 (Between, On)

A relation R from M to N is called a relation between M and N or, synonymously, a relation on $M \times N$.

If M equals N, then R is called a relation on M, in symbols: (M, R).

Domain and Range of a Binary Relation

Definition A.1.4 (Domain and Range)

Let R be a relation from M to N.

The sets

- $dom(R) =_{df} \{m \mid \exists\, n \in N.\ (m, n) \in R\}$
- $ran(R) =_{df} \{n \mid \exists\, m \in M.\ (m, n) \in R\}$

are called the domain and the range of R, respectively.

Properties of Relations on a Set M

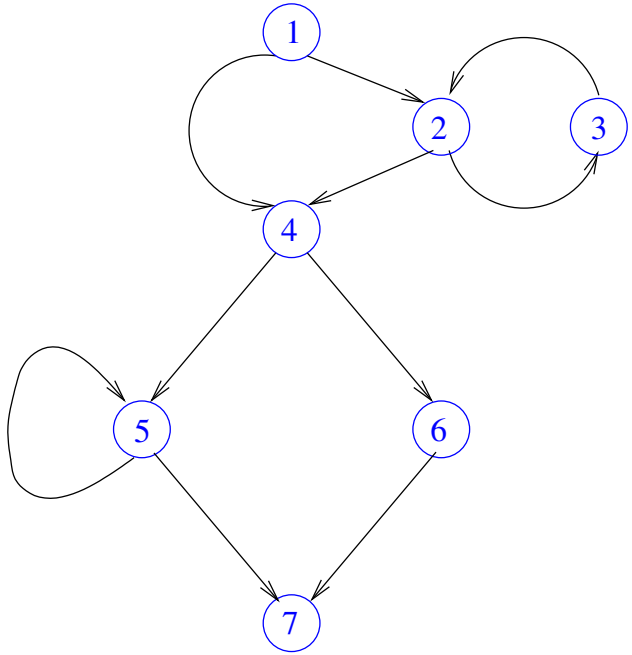
Definition A.1.5 (Properties of Relations on M)

A relation R on a set M is called

- reflexive iff $\forall\, m \in M.\ m\, R\, m$
- irreflexive iff $\forall\, m \in M.\ \neg m\, R\, m$
- transitive iff $\forall\, m, n, p \in M.\ m\, R\, n\ \wedge\ n\, R\, p\ \Rightarrow\ m\, R\, p$
- intransitive iff $\forall\, m, n, p \in M.\ m\, R\, n \wedge n\, R\, p \Rightarrow \neg m\, R\, p$
- symmetric iff $\forall\, m, n \in M.\ m\, R\, n\ \Longleftrightarrow\ n\, R\, m$
- antisymmetric iff $\forall\, m, n \in M.\ m\, R\, n \wedge n\, R\, m \Rightarrow m = n$
- asymmetric iff $\forall\, m, n \in M.\ m\, R\, n \Rightarrow \neg n\, R\, m$
- linear iff $\forall\, m, n \in M.\ m\, R\, n\ \vee\ n\, R\, m\ \vee\ m = n$
- total iff $\forall\, m, n \in M.\ m\, R\, n\ \vee\ n\, R\, m$

(Anti-) Example

Let $G = (N, E, \mathbf{s} \equiv 1, \mathbf{e} \equiv 7)$ be the below (flow) graph, and let R be the relation '$\cdot$ is linked to $\cdot$ via a (directed) edge' on N of G (e.g., node 4 is linked to node 6 but not vice versa).



The relation R is not reflexive, not irreflexive, not transitive, not intransive, not symmetric, not antisymmetric, not asymmetric, not linear, and not total.

Equivalence Relation

Let R be a relation on M.

Definition A.1.6 (Equivalence Relation)

R is an equivalence relation (or equivalence) iff R is reflexive, transitive, and symmetric.

A.2

Geordnete Mengen, Ordnungen

A.2.1

Halbordnungen, partielle Ordnungen

Ordered Sets

Let R be a relation on M.

Definition A.2.1.1 (Pre-Order)

R is a pre-order (or quasi-order) iff R is reflexive and transitive.

Definition A.2.1.2 (Partial Order)

R is a partial order (or poset or order) iff R is reflexive, transitive, and antisymmetric.

Definition A.2.1.3 (Strict Partial Order)

R is a strict partial order iff R is asymmetric and transitive.

Examples of Ordered Sets

Pre-order (reflexive, transitive)

- The relation $\Rightarrow$ on logical formulas.

Partial order (reflexive, transitive, antisymmetric)

- The relations $=$, $\leq$ and $\geq$ on $\mathbb{N}$.
- The relation $m \mid n$ (m is a divisor of n) on $\mathbb{N}$.

Strict partial order (asymmetric, transitive)

- The relations $<$ and $>$ on $\mathbb{N}$.
- The relations $\subset$ and $\supset$ on sets.

Equivalence relation (reflexive, transitive, symmetric)

- The relation $\Longleftrightarrow$ on logical formulas.
- The relation 'have the same prime number divisors' on $\mathbb{N}$.
- The relation 'are citizens of the same country' on people.

Note

- An antisymmetric pre-order is a partial order; a symmetric pre-order is an equivalence relation.
- For convenience, also the pair (M, R) is called a pre-order, partial order, and strict partial order, respectively.
- More accurately, we could speak of the pair (M, R) as of a set M which is pre-ordered, partially ordered, and strictly partially ordered by R, respectively.
- Synonymously, we also speak of M as a pre-ordered, partially ordered, and a strictly partially ordered set, respectively, or of M as a set which is equipped with a pre-order, partial order and strict partial order, respectively.
- On any set, the equality relation $=$ is a partial order, called the discrete (partial) order.

The Strict Part of an Ordering

Let $\sqsubseteq$ be a pre-order (reflexive, transitive) on P.

Definition A.2.1.4 (Strict Part of $\sqsubseteq$)

The relation $\sqsubset$ on P defined by

$$\forall\, p, q \in P.\ p \sqsubset q \iff_{df} p \sqsubseteq q \ \wedge\ p \neq q$$

is called the strict part of $\sqsubseteq$.

Corollary A.2.1.5 (Strict Partial Order)

Let $(P, \sqsubseteq)$ be a partial order, let $\sqsubset$ be the strict part of $\sqsubseteq$.

Then: $(P, \sqsubset)$ is a strict partial order.

Useful Results

Let $\sqsubset$ be a strict partial order (asymmetric, transitive) on P.

Lemma A.2.1.6

The relation $\sqsubset$ is irreflexive.

Lemma A.2.1.7

The pair $(P, \sqsubseteq)$, where $\sqsubseteq$ is defined by

$$\forall\, p, q \in P.\ p \sqsubseteq q \iff_{df} p \sqsubset q \ \vee \ p = q$$

is a partial order.

Induced (or Inherited) Partial Order

Definition A.2.1.8 (Induced Partial Order)

Let $(P, \sqsubseteq_P)$ be a partially ordered set, let $Q \subseteq P$ be a subset of P, and let $\sqsubseteq_Q$ be the relation on Q defined by

$$\forall\, q, r \in Q.\ q \sqsubseteq_Q r \iff_{df} q \sqsubseteq_P r$$

Then: $\sqsubseteq_Q$ is called the induced partial order on Q (or the inherited order from P on Q).

Exercise

Which of the below diagrams are Hasse diagrams (cf. Chapter A.2.8) of partial orders?

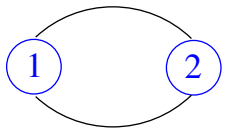
a)

{ }

b)

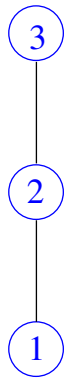
c)

d)

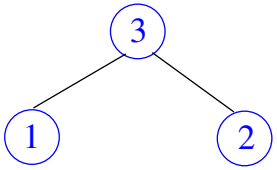


e)

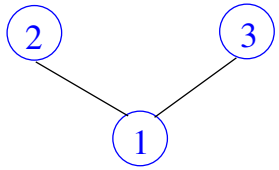
f)



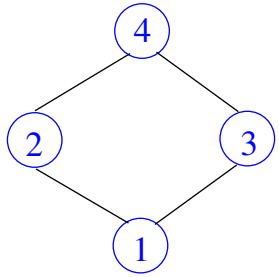
g)



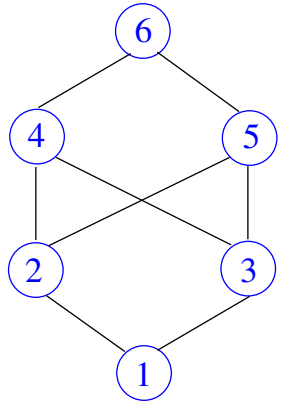
h)



i)



j)



A.2.2

Schranken und extreme Elemente

Bounds in Pre-Orders

Definition A.2.2.1 (Bounds in Pre-Orders)

Let $(Q, \sqsubseteq)$ be a pre-order, let $q \in Q$ and $Q' \subseteq Q$.

q is called a

- lower bound of Q', in signs: $q \sqsubseteq Q'$, if $\forall\, q' \in Q'.\ q \sqsubseteq q'$
- upper bound of Q', in signs: $Q' \sqsubseteq q$, if $\forall\, q' \in Q'.\ q' \sqsubseteq q$
- greatest lower bound (glb) (or infimum) of Q', in signs: $\bigsqcap Q'$, if q is a lower bound of Q' and for every other lower bound $\hat{q}$ of Q' holds: $\hat{q} \sqsubseteq q$.
- least upper bound (lub) (or supremum) of Q', in signs: $\bigsqcup Q'$, if q is an upper bound of Q' and for every other upper bound $\hat{q}$ of Q' holds: $q \sqsubseteq \hat{q}$.

Extremal Elements in Pre-Orders

Definition A.2.2.2 (Extremal Elements in Pre-Ord's)

Let $(Q, \sqsubseteq)$ be a pre-order, let $\sqsubset$ be the strict part of $\sqsubseteq$, and let $Q' \subseteq Q$ and $q \in Q'$.

q is called a

- minimal element of Q', if there is no $q' \in Q'$ with $q' \sqsubset q$.
- maximal element of Q', if there is no $q' \in Q'$ with $q \sqsubset q'$.
- least (or minimum) element of Q', if $q \sqsubseteq Q'$.
- greatest (or maximum) element of Q', if $Q' \sqsubseteq q$.

Note: The least element and the greatest element of Q itself are usually denoted by $\bot$ and $\top$, respectively, if they exist. A least (greatest) element is also a minimal (maximal) element.

Existence and Uniqueness

...of bounds and extremal elements in partially ordered sets.

Let $(P, \sqsubseteq)$ be a partial order, and let $Q \subseteq P$ be a subset of P.

Lemma A.2.2.3 (lub/glb: Unique if Existent)

Least upper bounds, greatest lower bounds, least elements, and greatest elements in Q are unique, if they exist.

Lemma A.2.2.4 (Minimal/Maximal El.: Not Unique)

Minimal and maximal elements in Q are usually not unique.

Note: Lemma A.2.2.3 suggests considering $\bigsqcup$ and $\bigsqcap$ partial maps $\bigsqcup, \bigsqcap : \mathcal{P}(P) \to P$ from the powerset $\mathcal{P}(P)$ of P to P. Lemma A.2.2.3 does not hold for pre-orders.

Characterization of Least, Greatest Elements

...in terms of infima and suprema of sets.

Let $(P, \sqsubseteq)$ be a partial order.

Lemma A.2.2.5 (Characterization of $\bot$ and $\top$)

The least element $\bot$ and the greatest element $\top$ of P are given by the supremum and the infimum of the empty set, and the infimum and the supremum of P, respectively, i.e.,

$$\bot = \bigsqcup \emptyset = \bigsqcap P \quad \text{and} \quad \top = \bigsqcap \emptyset = \bigsqcup P$$

if they exist.

Lower and Upper Bound Sets

Considering $\bigsqcup$ and $\bigsqcap$ partial functions $\bigsqcup, \bigsqcap : \mathcal{P}(P) \to P$ on the powerset of a partial order $(P, \sqsubseteq)$ suggests introducing two further maps $LB, UB : \mathcal{P}(P) \to \mathcal{P}(P)$ on $\mathcal{P}(P)$:

Definition A.2.2.6 (Lower and Upper Bound Sets)

Let $(P, \sqsubseteq)$ be a partial order. Then:

$LB, UB : \mathcal{P}(P) \to \mathcal{P}(P)$ denote two maps, which map a subset $Q \subseteq P$ to the set of its lower bounds and upper bounds, respectively:

1. $\forall\, Q \subseteq P.\ LB(Q) =_{df} \{lb \in P \mid\ lb \sqsubseteq Q\}$
2. $\forall\, Q \subseteq P.\ UB(Q) =_{df} \{ub \in P \mid Q \sqsubseteq ub\}$

Properties of Lower and Upper Bound Sets

Lemma A.2.2.7

Let $(P, \sqsubseteq)$ be a partial order, and let $Q \subseteq P$. Then:

$$\bigsqcup Q = \bigsqcap UB(Q) \quad \text{and} \quad \bigsqcap Q = \bigsqcup LB(Q)$$

if the supremum and the infimum of Q exist.

Lemma A.2.2.8

Let $(P, \sqsubseteq)$ be a partial order, and let $Q, Q_1, Q_2 \subseteq P$. Then:

1. $Q_1 \subseteq Q_2 \Rightarrow LB(Q_1) \supseteq LB(Q_2) \wedge UB(Q_1) \supseteq UB(Q_2)$
2. $UB(LB(UB(Q))) = UB(Q)$
3. $LB(UB(LB(Q))) = LB(Q)$

Note: Lemma A.2.2.8(1) shows that LB and UB are antitonic maps (cf. Chapter A.2.5).

Exercise

Which of the elements of the below diagrams are minimal, maximal, least or greatest?

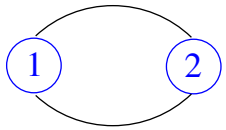
a)

{ }

b)

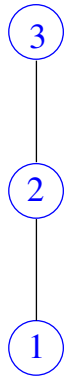
c)

d)

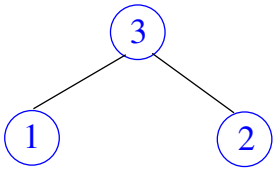


e)

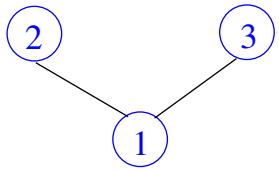
f)



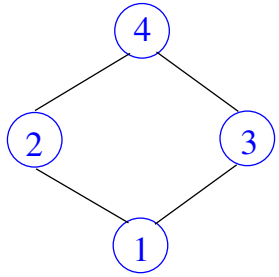
g)



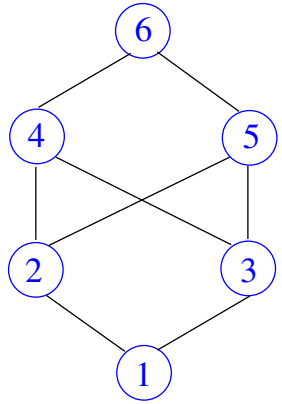
h)



i)



j)



A.2.3

Noethersche Ordnungen, Artinsche Ordnungen

Noetherian Orders and Artinian Orders

Let $(P, \sqsubseteq)$ be a partial order.

Definition A.2.3.1 (Noetherian Order)

$(P, \sqsubseteq)$ is called a Noetherian order, if every non-empty subset $\emptyset \neq Q \subseteq P$ contains a minimal element.

Definition A.2.3.2 (Artinian Order)

$(P, \sqsubseteq)$ is called an Artinian order, if the dual order $(P, \sqsupseteq)$ of $(P, \sqsubseteq)$ is a Noetherian order.

Lemma A.2.3.3

$(P, \sqsubseteq)$ is an Artinian order iff every non-empty subset $\emptyset \neq Q \subseteq P$ contains a maximal element.

Well-founded Orders

Let $(P, \sqsubseteq)$ be a partial order.

Definition A.2.3.4 (Well-founded Order)

$(P, \sqsubseteq)$ is called a well-founded order, if $(P, \sqsubseteq)$ is a Noetherian order and totally ordered.

Lemma A.2.3.5

$(P, \sqsubseteq)$ is a well-founded order iff every non-empty subset $\emptyset \neq Q \subseteq P$ contains a least element.

Noetherian Induction

Theorem A.2.3.6 (Noetherian Induction)

Let $(N, \sqsubseteq)$ be a Noetherian order, let $N_{min} \subseteq N$ be the set of minimal elements of N, and let $\phi : N \to \mathbb{B}$ be a predicate on N. Then:

If

1. $\forall\, n \in N_{min}.\ \phi(n)$ (Induction base)
2. $\forall\, n \in N \backslash N_{min}.\ (\forall\, m \sqsubset n.\ \phi(m)) \Rightarrow \phi(n)$ (Induction step)

then:

$$\forall\, n \in N.\ \phi(n)$$

A.2.4

Ketten

Chains, Antichains

Let $(P, \sqsubseteq)$ be a partial order.

Definition A.2.4.1 (Chain)

A set $C \subseteq P$ is called a chain, if the elements of C are totally ordered, i.e., $\forall\, c_1, c_2 \in C.\ c_1 \sqsubseteq c_2 \vee c_2 \sqsubseteq c_1$.

Definition A.2.4.2 (Antichain)

A set $C \subseteq P$ is called an antichain, if
$\forall\, c_1, c_2 \in C.\ c_1 \sqsubseteq c_2 \Rightarrow c_1 = c_2$.

Definition A.2.4.3 (Finite, Infinite (Anti-) Chain)

Let $C \subseteq P$ be a chain or an antichain. C is called finite, if the number of its elements is finite; C is called infinite otherwise.

Note: Any set P may be converted into an antichain by giving it the discrete order: $(P, =)$.

Ascending Chains, Descending Chains

Definition A.2.4.4 (Ascending, Descending Chain)

Let $C \subseteq P$ be a chain. C given in the form of

- $C = \{c_0 \sqsubseteq c_1 \sqsubseteq c_2 \sqsubseteq \ldots\}$
- $C = \{c_0 \sqsupseteq c_1 \sqsupseteq c_2 \sqsupseteq \ldots\}$

is called an ascending chain and descending chain, respectively.

Eventually Stationary Sequences

Definition A.2.4.5 (Stationary Sequence)

1. An ascending sequence of the form

$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots$$

is called to get stationary, if $\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$.

2. A descending sequence of the form

$$p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$

is called to get stationary, if $\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$.

Chains and Sequences

Lemma A.2.4.6

An ascending or descending sequence of the form

$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots \quad \text{or} \quad p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$

1. is a finite chain iff it gets stationary.
2. is an infinite chain iff it does not get stationary.

Note the subtle difference between the notion of chains in terms of sets

$$\{p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots\} \quad \text{or} \quad \{p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots\}$$

and in terms of sequences

$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots \quad \text{or} \quad p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$

Sequences may contain duplicates, which would correspond to a definition of chains in terms of multisets.

Examples of Chains

- The set $S =_{df} \{n \in \mathbb{N} \mid n \text{ even}\}$ is a chain in $\mathbb{N}$.
- The set $S =_{df} \{z \in \mathbb{Z} \mid z \text{ odd}\}$ is a chain in $\mathbb{Z}$.
- The set $S =_{df} \{\{k \in \mathbb{N} \mid k < n\} \mid n \in \mathbb{N}\}$ is a chain in the powerset $\mathcal{P}(\mathbb{N})$ of $\mathbb{N}$.

Note: A chain can always be given in the form of an ascending or descending chain.

- $\{0 \leq 2 \leq 4 \leq 6 \leq \ldots\}$: $\mathbb{N}$ as ascending chain.
- $\{\ldots \geq 6 \geq 4 \geq 2 \geq 0\}$: $\mathbb{N}$ as descending chain.
- $\{\ldots \leq -3 \leq -1 \leq 1 \leq 3 \leq \ldots\}$: $\mathbb{Z}$ as ascending chain.
- $\{\ldots \geq 3 \geq 1 \geq -1 \geq -3 \geq \ldots\}$: $\mathbb{Z}$ as descending chain.
- ...

Chains and Noetherian Orders

Let $(P, \sqsubseteq)$ be a partial order.

Lemma A.2.4.7 (Noetherian Order)

The following statements are equivalent:

1. $(P, \sqsubseteq)$ is a Noetherian order
2. Every chain of the form
$$p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \dots$$
gets stationary, i.e.: $\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$.
3. Every chain of the form
$$p_0 \sqsupset p_1 \sqsupset p_2 \sqsupset \dots$$
is finite.

Chains and Artinian Orders

Let $(P, \sqsubseteq)$ be a partial order.

Lemma A.2.4.8 (Artinian Order)

The following statements are equivalent:

1. $(P, \sqsubseteq)$ is an Artinian order
2. Every chain of the form

   $$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \dots$$

   gets stationary, i.e.: $\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$.
3. Every chain of the form

   $$p_0 \sqsubset p_1 \sqsubset p_2 \sqsubset \dots$$

   is finite.

Chains and Noetherian, Artinian Orders

Let $(P, \sqsubseteq)$ be a partial order.

Lemma A.2.4.9 (Noetherian and Artinian Order)

$(P, \sqsubseteq)$ is a Noetherian order and an Artinian order iff every chain $C \subseteq P$ is finite.

A.2.5

Gerichtete Mengen

Directed Sets

Let $(P, \sqsubseteq)$ be a partial order, and let $\emptyset \neq D \subseteq P$.

Definition A.2.5.1 (Directed Set)

$D \neq \emptyset$ is called a directed set (in German: gerichtete Menge), if

$$\forall\, d, e \in D.\ \exists\, f \in D.\ f \in UB(\{d, e\}),\ \text{i.e.,}$$

for any two elements d and e there is a common upper bound of d and e in D, i.e., $UB(\{d, e\}) \cap D \neq \emptyset$.

Properties of Directed Sets

Let $(P, \sqsubseteq)$ be a partial order, and let $D \subseteq P$.

Lemma A.2.5.2

D is a directed set iff any finite subset $D' \subseteq D$ has an upper bound in D, i.e., $\exists\, d \in D.\ d \in UB(D')$, i.e., $UB(D') \cap D \neq \emptyset$.

Lemma A.2.5.3

If D has a greatest element, then D is a directed set.

Properties of Finite Directed Sets

Let $(P, \sqsubseteq)$ be a partial order, and let $D \subseteq P$.

Corollary A.2.5.4

Let D be a directed set. If D is finite, then $\bigsqcup D$ *exists* $\in D$ and is the greatest element of D.

Proof. Choosing D a directed set, we have:

$$\exists\, d \in D.\ d \in UB(D),\ \text{i.e.,}\ UB(D) \cap D \neq \emptyset.$$

This means $D \sqsubseteq d$. The antisymmetry of $\sqsubseteq$ yields that d is unique enjoying this property. Thus, d is the (unique) greatest element of D given by $\bigsqcup D$, i.e., $d = \bigsqcup D$.

Note: If D is infinite, the statement of Corollary A.2.5.4 does usually not hold.

Strongly Directed Sets

Let $(P, \sqsubseteq)$ be a partial order, and let $D \subseteq P$.

Definition A.2.5.5 (Strongly Directed Set)

D is called a strongly directed set (in German: stark gerichtete Menge), if any finite subset $D' \subseteq D$ of D has a supremum in D, i.e.,

$$\forall\, D' \subseteq D.\ \exists\, d \in D.\ d = \bigsqcup D'$$

Properties of Strongly Directed Sets (1)

Let $(P, \sqsubseteq)$ be a partial order, and let $D \subseteq P$.

Lemma A.2.5.6

Let D be a strongly directed set. Then: P has a least element $\bot$ with $\bot = \bigsqcup \emptyset$ and $\bot \in D$.

Proof. Let D be a strongly directed set. Since $\emptyset$ *finite* $\subseteq D$, $\bigsqcup \emptyset$ *exists* $\in D$ by Definition A.2.5.5, and thus also the unique least element $\bot = \bigsqcup \emptyset$ of P.

Lemma A.2.5.7

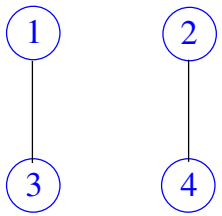
Let D be a strongly directed set. If D is finite, then $\bigsqcup D$ *exists* $\in D$ and is the greatest element of D.

Note: The reasoning of Lemma A.2.5.6 does not hold, if D is a directed set. The statement of Lemma A.2.5.7 does usually not hold, if If D is infinite.

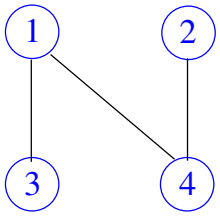
Exercise (1)

Which of the below partial orders are (strongly) directed sets?
Which of their subsets are (strongly) directed sets?

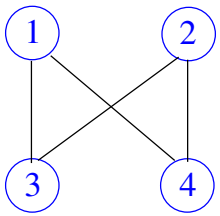
a)



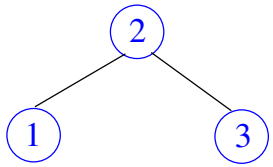
b)



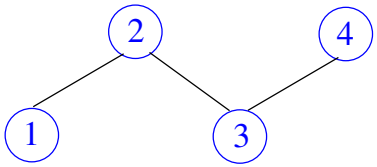
c)



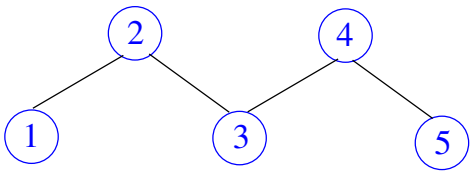
d)



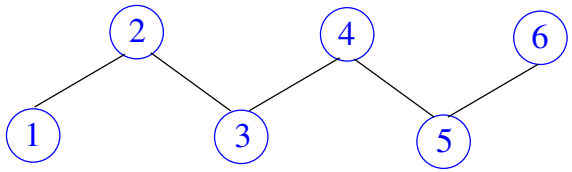
e)



f)



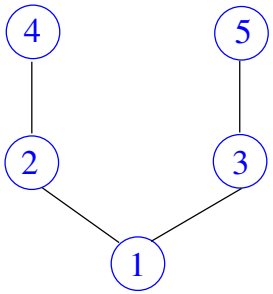
g)



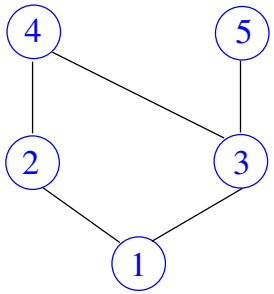
Exercise (2)

Which of the below partial orders are (strongly) directed sets?
Which of their subsets are (strongly) directed sets?

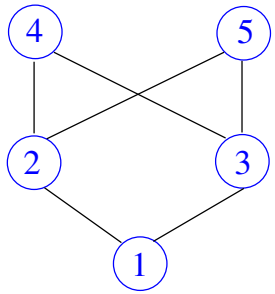
a)



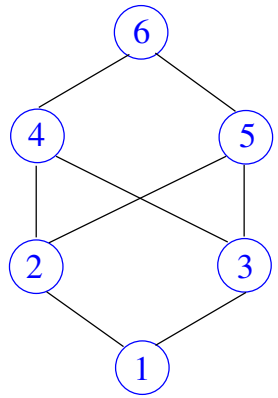
b)



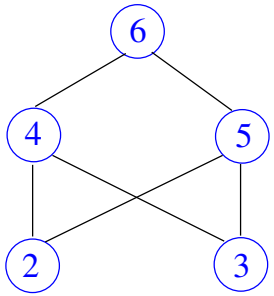
c)



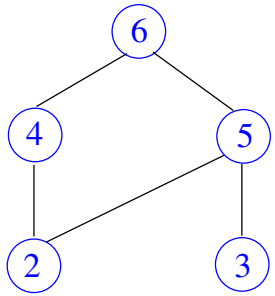
d)



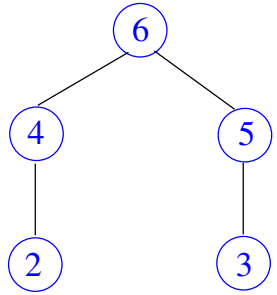
e)



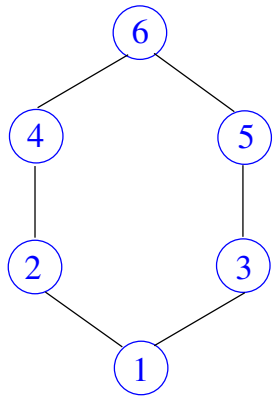
f)



g)



h)



A.2.6

Abbildungen auf partiellen Ordnungen

Monotonic and Antitonic Maps on POs

Let $(C, \sqsubseteq_C)$ and $(D, \sqsubseteq_D)$ be partial orders, and let $f \in [C \to D]$ be a map from C to D.

Definition A.2.6.1 (Monotonic Maps on POs)

f is called monotonic (or order preserving) iff

$$\forall\, c, c' \in C.\ c \sqsubseteq_C c' \Rightarrow f(c) \sqsubseteq_D f(c')$$

(Preservation of the ordering of elements)

Definition A.2.6.2 (Antitonic Maps on POs)

f is called antitonic (or order inversing) iff

$$\forall\, c, c' \in C.\ c \sqsubseteq_C c' \Rightarrow f(c') \sqsubseteq_D f(c)$$

(Inversion of the ordering of elements)

Expanding and Contracting Maps on POs

Let $(C, \sqsubseteq_C)$ be a partial orders (PO), let $f \in [C \to C]$ be a map on C, and let $\hat{c} \in C$ be an element of C.

Definition A.2.6.3 (Expanding Maps on POs)

f is called

- expanding (or inflationary) for $\hat{c}$ iff $\hat{c} \sqsubseteq f(\hat{c})$
- expanding (or inflationary) iff $\forall\, c \in C.\ c \sqsubseteq f(c)$

Definition A.2.6.4 (Contracting Maps on POs)

f is called

- contracting (or deflationary) for $\hat{c}$ iff $f(\hat{c}) \sqsubseteq \hat{c}$
- contracting (or deflationary) iff $\forall\, c \in C.\ f(c) \sqsubseteq c$

A.2.7

Ordnungshomomorphismen, Ordnungsisomorphismen zwischen partiellen Ordnungen

PO Homomorphisms, PO Isomorphisms

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be partial orders, and let $f \in [P \to R]$ be a map from P to R.

Definition A.2.7.1 (PO Hom. & Isomorphism)

f is called an

1. order homomorphism between P and R, if f is monotonic (or order preserving), i.e.,
$$\forall\, p, q \in P.\ p \sqsubseteq_P q \Rightarrow f(p) \sqsubseteq_R f(q)$$
2. order isomorphism between P and R, if f is a bijective order homomorphism between P and R and the inverse f^{-1} of f is an order homomorphism between R and P.

Definition A.2.7.2 (Order Isomorphic)

$(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ are called order isomorphic, if there is an order isomorphism between P and R.

PO Embeddings

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be partial orders, and let $f \in [P \to R]$ be a map from P to R.

Definition A.2.7.3 (PO Embedding)

f is called an order embedding of P in R iff

$$\forall\, p, q \in P.\ p \sqsubseteq_P q \iff f(p) \sqsubseteq_R f(q)$$

Lemma A.2.7.4 (PO Embeddings and Isomorphisms)

f is an order isomorphism between P and R iff f is an order embedding of P in R and f is surjective.

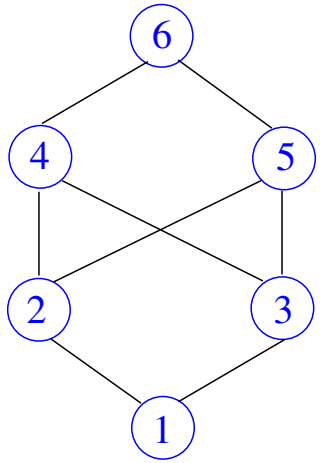
Intuitively: Partial orders, which are order isomorphic, are "essentially the same."

A.2.8

Hasse-Diagramme

Hasse Diagrams

...are an economic graphical representation of partial orders.



The links of a Hasse diagram

- are read from below to above (lower means smaller).
- represent the relation R of '$\cdot$ is an immediate predecessor of $\cdot$' defined by
  $p\,R\,q \iff_{df} p \sqsubset q \wedge \nexists\, r \in P.\ p \sqsubset r \sqsubset q$
  of a partial order $(P, \sqsubseteq)$, where $\sqsubset$ is the strict part of $\sqsubseteq$.

Reading Hasse Diagrams

The Hasse diagram representation of a partial order

- omits links which express reflexive and transitive relations explicitly
- focuses on the 'immediate predecessor' relation.

This focused representation of a Hasse diagram

- is economical (in the number of links)
- while preserving all relevant information of the represented partial order:
  - $p \sqsubseteq q \wedge p = q$ (reflexivity): trivially represented (just without an explicit link)
  - $p \sqsubseteq q \wedge p \neq q$ (transitivity): represented by ascending paths (with at least one link) from p to q.

A.3

Vollständige partielle Ordnungen

A.3.1

Kettenvollständige, gerichtete vollständige partielle Ordnungen

Complete Partially Ordered Sets

...or Complete Partial Orders:

- a slightly weaker ordering notion than that of a lattice (cf. Appendix A.4), which is often more adequate for the modelling of problems in computer science, where full lattice properties are often not required.
- come in two different flavours as so-called
  - Chain Complete Partial Orders (CCPOs)
  - Directed Complete Partial Orders (DCPOs)

  based on the notions of chains and directed sets, respectively, which turn out to be equvialent (cf. Theorem 3.1.7)

Complete Partial Orders: CCPOs

Let $(P, \sqsubseteq)$ be a partial order.

Definition A.3.1.1 (Chain Complete Partial Order)

$(P, \sqsubseteq)$ is a

1. chain complete partial order (pre-CCPO), if every non-empty (ascending) chain $\emptyset \neq C \subseteq P$ has a least upper bound $\bigsqcup C$ in P, i.e., $\bigsqcup C$ *exists* $\in P$.
2. pointed chain complete partial order (CCPO), if every (ascending) chain $C \subseteq P$ has a least upper bound $\bigsqcup C$ in P, i.e., $\bigsqcup C$ *exists* $\in P$.

Complete Partial Orders: DCPOs

Definition A.3.1.2 (Directedly Complete Partial Ord.)

A partial order $(P, \sqsubseteq)$ is a

1. directedly complete partial order (pre-DCPO), if every directed subset $D \subseteq P$ has a least upper bound $\bigsqcup D$ in P, i.e., $\bigsqcup D$ *exists* $\in P$.
2. pointed directedly complete partial order (DCPO), if it is a pre-DCPO and has a least element $\bot$.

Remarks about CCPOs and DCPOs

About CCPOs

- A CCPO is often called a domain.
- 'Ascending chain' and 'chain' can equivalently be used in Definition A.3.1.1, since a chain can always be given in ascending order. 'Ascending chain' is just more intuitive.

About DCPOs

- A directed set S, in which by definition every finite subset has an upper bound in S, does not need to have a supremum in S, if S is infinite. Therefore, the DCPO property does not trivially follow from the directed set property (cf. Corollary A.2.5.5).

Existence of Least Elements in CCPOs

Lemma A.3.1.3 (Least Elem. Existence in CCPOs)

Let $(C, \sqsubseteq)$ be a CCPO. Then there is a unique least element in C, denoted by $\bot$, which is given by the supremum of the empty chain, i.e.: $\bot = \bigsqcup \emptyset$.

Corollary A.3.1.4 (Non-Emptyness of CCPOs)

Let $(C, \sqsubseteq)$ be a CCPO. Then: $C \neq \emptyset$.

Note: Lemma A.3.1.3 does not hold for pre-DCPOs, i.e., if $(D, \sqsubseteq)$ is a pre-DCPO, there does not need to be a least element in D.

Relating Finite POs, DCPOs and CCPOs

Let P be a finite set, and let $\sqsubseteq$ be a relation on P.

Lemma A.3.1.5 (Finite POs, DCPOs and CCPOs)

The following statements are equivalent:

- $(P, \sqsubseteq)$ is a partial order.
- $(P, \sqsubseteq)$ is a pre-CCPO.
- $(P, \sqsubseteq)$ is a pre-DCPO.

Lemma A.3.1.6 (Finite POs, DCPOs and CCPOs)

Let $p \in P$ with $p \sqsubseteq P$. Then the following statements are equivalent.

- $(P, \sqsubseteq)$ is a partial order.
- $(P, \sqsubseteq)$ is a CCPO.
- $(P, \sqsubseteq)$ is a DCPO.

Equivalence of CCPOs and DCPOs

Theorem A.3.1.7 (Equivalence)

Let $(P, \sqsubseteq)$ be a partial order. Then the following statements are equivalent:

- $(P, \sqsubseteq)$ is a CCPO.
- $(P, \sqsubseteq)$ is a DCPO.

SDCPOs: A DCPO Variant

About DCPOs based on Strongly Directed Sets

- Replacing directed sets by strongly directed sets in Definition A.3.1.2 leads to SDCPOs.
- Recalling that strongly directed sets are not empty (cf. Lemma A.2.5.9), there is no analogue of pre-DCPOs for strongly directed sets.
- A strongly directed set S, in which by definition every finite subset has a supremum in S, does not need to have a supremum itself in S, if S is infinite. Therefore, the SDCPO property does not trivially follow from the strongly directed set property (cf. Corollary A.2.5.3).

Examples of CCPOs and DCPOs (1)

- $(\mathcal{P}(\mathbb{N}), \subseteq)$ is a CCPO and a DCPO.
  - Least element: $\emptyset$
  - Least upper bound $\bigsqcup C$ of C *chain* $\subseteq \mathcal{P}(\mathbb{N})$: $\bigcup_{C' \in C} C'$
- The set of finite and infinite strings S partially ordered by the prefix relation $\sqsubseteq_{pfx}$ defined by

$$\forall\, s, s'' \in S.\ s \sqsubseteq_{pfx} s'' \iff_{df}$$
$$s = s'' \vee (s \textit{ finite } \wedge \exists\, s' \in S.\ s +\!\!+ s' = s'')$$

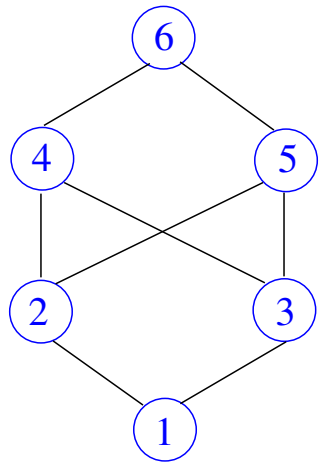
  is a CCPO and a DCPO.
- $(\{-n \mid n \in \mathbb{N}\}, \leq)$ is a pre-CCPO and a pre-DCPO but not a CCPO and not a DCPO.

Examples of CCPOs and DCPOs (2)

- $(\emptyset, \emptyset)$ is a pre-CCPO and a pre-DCPO but not a CCPO and not a DCPO.

  (Both the pre-CCPO (absence of non-empty chains in $\emptyset$) and the pre-DCPO ($\emptyset$ is the only subset of $\emptyset$ and is not directed by definition) property holds trivially. Note also that $P = \emptyset$ implies $\sqsubseteq = \emptyset \subseteq P \times P$).

- The partial order P given by the below Hasse diagram is a CCPO and a DCPO.



Examples of CCPOs and DCPOs (3)

- The set of finite and infinite strings S partially ordered by the lexicographical order $\sqsubseteq_{lex}$ defined by

  $$\forall\, s, t \in S.\ s \sqsubseteq_{lex} t \iff_{df}$$
  $$s = t \vee (\exists\, p\ \mathit{finite}, s', t' \in S.\ s = p +\!\!+ s' \ \wedge\ t = p +\!\!+ t' \ \wedge\ (s' = \varepsilon \ \vee\ s'_1 < t'_1))$$

  where ε denotes the empty string, w_1 denotes the first character of a string w, and $<$ the lexicographical ordering on characters, is a CCPO and a DCPO.

(Anti-) Examples of CCPOs and DCPOs

- $(\mathbb{N}, \leq)$ is not a CCPO and not a DCPO.
- The set of finite strings S_{fin} partially ordered by the
  - prefix relation $\sqsubseteq_{pfx}$ defined by

    $$\forall\, s, s' \in S_{fin}.\ s \sqsubseteq_{pfx} s' \iff_{df} \exists\, s'' \in S_{fin}.\ s +\!\!+ s'' = s'$$

    is not a CCPO and not a DCPO.
  - lexicographical order $\sqsubseteq_{lex}$ defined by

    $$\forall\, s, t \in S_{fin}.\ s \sqsubseteq_{lex} t \iff_{df}$$
    $$\exists\, p, s', t' \in S_{fin}.\ s = p +\!\!+ s' \ \wedge\ t = p +\!\!+ t' \ \wedge$$
    $$(s' = \varepsilon \ \vee\ s'\!\downarrow_1 \ < t'\!\downarrow_1)$$

    where ε denotes the empty string, $w\!\downarrow_1$ denotes the first character of a string w, and $<$ the lexicographical ordering on characters, is not a CCPO and not a DCPO.
- $(\mathcal{P}_{fin}(\mathbb{N}), \subseteq)$ is not a CCPO and not a DCPO.

Exercise

Which of the partial orders given by the below Hasse diagrams are (pre-) CCPOs? Which ones are (pre-) DCPOs?

a)

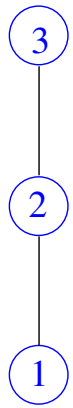
{ }

b)

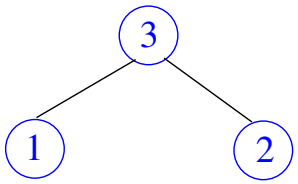
c)

d)

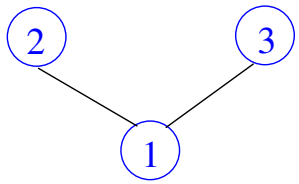
e)



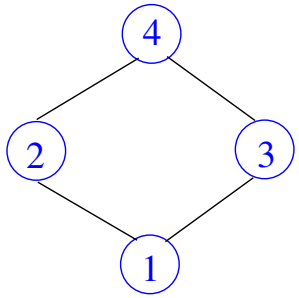
f)



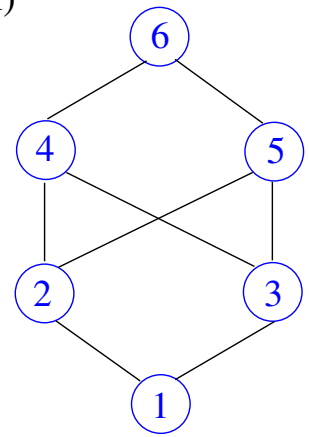
g)



h)



i)



Continuous Maps on CCPOs

Let $(C, \sqsubseteq_C)$ and $(D, \sqsubseteq_D)$ be CCPOs, and let $f \in [C \to D]$ be a map from C to D.

Definition A.3.1.8 (Continuous Maps on CCPOs)

f is called continuous iff f is monotonic and

$$\forall\, C' \neq \emptyset\ \mathit{chain} \subseteq C.\ f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$$

(Preservation of least upper bounds)

Note: $\forall\, S \subseteq C.\ f(S) =_{df} \{ f(s) \mid s \in S \}$

Continuous Maps on DCPOs

Let $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$ be DCPOs, and let $f \in [D \to E]$ be a map from D to E.

Definition A.3.1.9 (Continuous Maps on DCPOs)

f is called continuous iff

$$\forall\, D' \neq \emptyset \textit{ directed set } \subseteq D.\ f(D') \text{ directed set } \subseteq E \wedge f(\bigsqcup_D D') =_E \bigsqcup_E f(D')$$

(Preservation of least upper bounds)

Note: $\forall\, S \subseteq D.\ f(S) =_{df} \{ f(s) \mid s \in S \}$

Characterizing Monotonicity

Let $(C, \sqsubseteq_C), (D, \sqsubseteq_D)$ be CCPOs, let $(E, \sqsubseteq_E), (F, \sqsubseteq_F)$ be DCPOs.

Lemma A.3.1.10 (Characterizing Monotonicity)

1. $f : C \to D$ is monotonic

   iff $\forall\, C' \neq \emptyset$ *chain* $\subseteq C$.

   $$f(C') \text{ chain} \subseteq D \wedge f(\bigsqcup_C C') \sqsupseteq_D \bigsqcup_D f(C')$$

2. $g : E \to F$ is monotonic

   if $\forall\, E' \neq \emptyset$ *directed set* $\subseteq E$.

   $$g(E') \text{ directed set} \subseteq F \wedge g(\bigsqcup_E E') \sqsupseteq_F \bigsqcup_F g(E')$$

Strict Functions on CCPOs and DCPOs

Let $(C, \sqsubseteq_C), (D, \sqsubseteq_D)$ be CCPOs with least elements $\bot_C$ and $\bot_D$, respectively, let $(E, \sqsubseteq_E), (F, \sqsubseteq_F)$ be DCPOs with least elements $\bot_E$ and $\bot_F$, respectively, and let $f \in [C \overset{con}{\to} D]$ and $g \in [E \overset{con}{\to} F]$ be continuous functions.

Definition A.3.1.11 (Strict Functions on CPOs)

f and g are called strict, if the equalities

- $f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$, $g(\bigsqcup_E E') =_F \bigsqcup_F g(E')$

also hold for $C' = \emptyset$ and $E' = \emptyset$, i.e., if the equalities

- $f(\bigsqcup_C \emptyset) =_C f(\bot_C) =_D \bot_D =_D \bigsqcup \emptyset$
- $f(\bigsqcup_E \emptyset) =_E g(\bot_E) =_F \bot_F =_F \bigsqcup \emptyset$

are valid.

A.3.2

Konstruktion vollständiger partieller Ordnungen

Common CCPO and DCPO Constructions

The following construction principles hold for

- CCPOs
- DCPOs

Therefore, we simply write CPO.

Common CPO Constructions: Flat CPOs

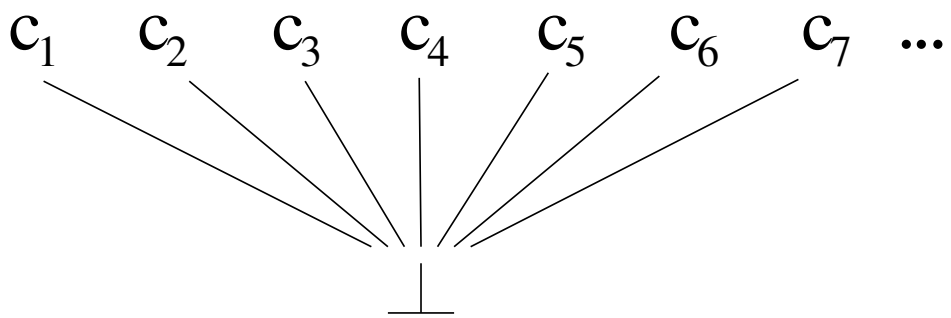
Lemma A.3.2.1 (Flat CPO Construction)

Let C be a set. Then:

$(C \dot\cup \{\bot\}, \sqsubseteq_{flat})$ with $\sqsubseteq_{flat}$ defined by

$$\forall\, c, d \in C \dot\cup \{\bot\}.\ c \sqsubseteq_{flat} d \Leftrightarrow c = \bot \ \vee \ c = d$$

is a CPO, a so-called flat CPO.



Common CPO Constructions: Flat pre-CPOs

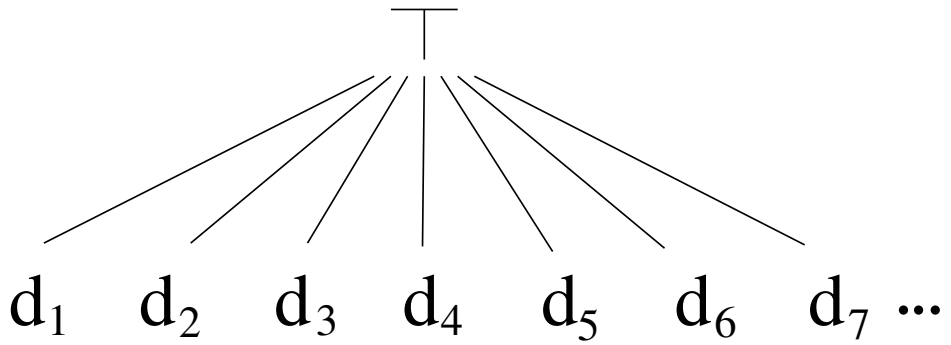
Lemma A.3.2.2 (Flat Pre-CPO Construction)

Let D be a set. Then:

$(D \dot\cup \{\top\}, \sqsubseteq_{flat})$ with $\sqsubseteq_{flat}$ defined by

$$\forall\, d, e \in D \dot\cup \{\top\}.\ d \sqsubseteq_{flat} e \Leftrightarrow e = \top \ \vee \ d = e$$

is a pre-CPO, a so-called flat pre-CPO.



Common CPO Constructions: Products (1)

Lemma A.3.2.3 (Non-strict Product Construction)

Let $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ be CPOs. Then:

The non-strict product $(\times P_i, \sqsubseteq_\times)$, where

- $\times P_i =_{df} P_1 \times P_2 \times \ldots \times P_n$ is the cartesian product of all P_i, $1 \leq i \leq n$
- $\sqsubseteq_\times$ is defined pointwise by

$$\forall\, (p_1, \ldots, p_n), (q_1, \ldots, q_n) \in \times P_i.$$
$$(p_1, \ldots, p_n) \sqsubseteq_\times (q_1, \ldots, q_n) \iff_{df} \forall\, i \in \{1, \ldots, n\}.\ p_i \sqsubseteq_i q_i$$

is a CPO.

Common CPO Constructions: Products (2)

Lemma A.3.2.4 (Strict Product Construction)

Let $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ be CPOs. Then:

The strict (or smash) product $(\bigotimes P_i, \sqsubseteq_\otimes)$, where

- $\bigotimes P_i =_{df} \times P_i$ is the the cartesian product of all P_i
- $\sqsubseteq_\otimes =_{df} \sqsubseteq_\times$ defined pointwise with the additional setting

$$(p_1, \ldots, p_n) = \bot \Leftrightarrow \exists\, i \in \{1, \ldots, n\}.\ p_i = \bot_i$$

is a CPO.

Common CPO Constructions: Sums (1)

Lemma A.3.2.5 (Separated Sum Construction)

Let $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ be CPOs. Then:

The separated (or direct) sum $(\bigoplus_\bot P_i, \sqsubseteq_{\oplus_\bot})$, where

- $\bigoplus_\bot P_i =_{df} P_1 \dot\cup P_2 \dot\cup \ldots \dot\cup P_n \dot\cup \{\bot\}$ is the disjoint union of all P_i, $1 \leq i \leq n$, and a fresh bottom element $\bot$
- $\sqsubseteq_{\oplus_\bot}$ is defined by

  $\forall\, p, q \in \bigoplus_\bot P_i.\ p \sqsubseteq_{\oplus_\bot} q \iff_{df}$

  $$p = \bot \ \vee\ (\exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge\ p \sqsubseteq_i q)$$

is a CPO.

Common CPO Constructions: Sums (2)

Lemma A.3.2.6 (Coalesced Sum Construction)

Let $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ be CPOs. Then:

The coalesced sum $(\bigoplus_\vee P_i, \sqsubseteq_{\oplus_\vee})$, where

- $\bigoplus_\vee P_i =_{df} P_1 \backslash \{\bot_1\} \dot\cup P_2 \backslash \{\bot_2\} \dot\cup \ldots \dot\cup P_n \backslash \{\bot_n\} \dot\cup \{\bot\}$ is the disjoint union of all P_i, $1 \leq i \leq n$, and a fresh bottom element $\bot$, which is identified with and replaces the least elements $\bot_i$ of the sets P_i, i.e., $\bot =_{df} \bot_i$, $i \in \{1, \ldots, n\}$
- $\sqsubseteq_{\oplus_\vee}$ is defined by

  $\forall\, p, q \in \bigoplus_\vee P_i.\ p \sqsubseteq_{\oplus_\vee} q \iff_{df}$

  $$p = \bot \ \vee\ (\exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge\ p \sqsubseteq_i q)$$

is a CPO.

Common CPO Constructions: Function Space

Lemma A.3.2.7 (Continuous Function Space Con.)

Let $(C, \sqsubseteq_C)$ and $(D, \sqsubseteq_D)$ be pre-CPOs. Then:

The continuous function space $([C \overset{con}{\to} D], \sqsubseteq_{cfs})$, where

- $[C \overset{con}{\to} D]$ is the set of continuous maps from C to D
- $\sqsubseteq_{cfs}$ is defined pointwise by
  $\forall f, g \in [C \overset{con}{\to} D].\ f \sqsubseteq_{cfs} g \iff_{df} \forall c \in C.\ f(c) \sqsubseteq_D g(c)$

is a pre-CPO. It is a CPO, if $(D, \sqsubseteq_D)$ is a CPO.

Note: The definition of $\sqsubseteq_{cfs}$ does not require C to be a pre-CPO. This requirement is only to ensure continuous maps.

A.4

Verbände

A.4.1

Verbände, vollständige Verbände

Lattices and Complete Lattices

Let $P \neq \emptyset$ be a non-empty set, and let $(P, \sqsubseteq)$ be a partial order on P.

Definition A.4.1.1 (Lattice)

$(P, \sqsubseteq)$ is a lattice, if every non-empty finite subset P' of P has a least upper bound and a greatest lower bound in P.

Definition A.4.1.2 (Complete Lattice)

$(P, \sqsubseteq)$ is a complete lattice, if every subset P' of P has a least upper bound and a greatest lower bound in P.

Note: Lattices and complete lattices are special partial orders.

Properties of Complete Lattices

Lemma A.4.1.3 (Existence of Extremal Elements)

Let $(P, \sqsubseteq)$ be a complete lattice. Then there is

1. a least element in P, denoted by $\bot$, satisfying: $\bot = \bigsqcup \emptyset = \bigsqcap P$.
2. a greatest element in P, denoted by $\top$, satisfying: $\top = \bigsqcap \emptyset = \bigsqcup P$.

Lemma A.4.1.4 (Characterization Lemma)

Let $(P, \sqsubseteq)$ be a partial order. Then the following statements are equivalent:

1. $(P, \sqsubseteq)$ is a complete lattice.
2. Every subset of P has a least upper bound.
3. Every subset of P has a greatest lower bound.

Properties of Finite Lattices

Lemma A.4.1.5 (Finite Lattices, Complete Lattices)

If $(P, \sqsubseteq)$ is a finite lattice, then $(P, \sqsubseteq)$ is a complete lattice.

Corollary A.4.1.6 (Finite Lattices, $\bot$, and $\top$)

If $(P, \sqsubseteq)$ is a finite lattice, then $(P, \sqsubseteq)$ has a least element and a greatest element.

Complete Semi-Lattices

Let $(P, \sqsubseteq)$ be a partial order, $P \neq \emptyset$.

Definition A.4.1.7 (Complete Semi-Lattices)

$(P, \sqsubseteq)$ is a complete

1. join semi-lattice iff $\forall\, \emptyset \neq S \subseteq P.\ \bigsqcup S$ *exists* $\in P$.
2. meet semi-lattice iff $\forall\, \emptyset \neq S \subseteq P.\ \bigsqcap S$ *exists* $\in P$.

Proposition A.4.1.8 (Spec. Bounds in Com. Semi-L.)

If $(P, \sqsubseteq)$ is a complete

1. join semi-lattice, then $\bigsqcup P$ *exists* $\in P$, while $\bigsqcup \emptyset$ does usually not exist in P.
2. meet semi-lattice, then $\bigsqcap P$ *exists* $\in P$, while $\bigsqcap \emptyset$ does usually not exist in P.

Properties of Complete Semi-Lattices (1)

Lemma A.4.1.9 (Greatest Elem. in a C. Join Semi-L.)

Let $(P, \sqsubseteq)$ be a complete join semi-lattice. Then there is a greatest element in P, denoted by $\top$, which is given by the supremum of P, i.e., $\top = \bigsqcup P$.

Lemma A.4.1.10 (Least Elem. in a C. Meet Semi-L.)

Let $(P, \sqsubseteq)$ be a complete meet semi-lattice. Then there is a least element in P, denoted by $\bot$, which is given by the infimum of P, i.e., $\bot = \bigsqcap P$.

Properties of Complete Semi-Lattices (2)

Lemma A.4.1.11 (Extremal Elements in C. Semi-L.)

If $(P, \sqsubseteq)$ is a complete

1. join semi-lattice where $\bigsqcup \emptyset$ *exists* $\in P$, then $\bigsqcup \emptyset$ is the least element in P, denoted by $\bot$, i.e., $\bot = \bigsqcup \emptyset$.
2. meet semi-lattice where $\bigsqcap \emptyset$ *exists* $\in P$, then $\bigsqcap \emptyset$ is the greatest element in P, denoted by $\top$, i.e., $\top = \bigsqcap \emptyset$.

Characterizing Upper and Lower Bounds

...in complete semi-lattices.

Lemma A.4.1.12 (Ex. & Char. of Bounds in C. S.-L.)

1. Let $(P, \sqsubseteq)$ be a complete join semi-lattice, and let $S \subseteq P$ be a subset of P.

   If there is a lower bound for S in P, i.e, if $\{p \in P \mid p \sqsubseteq S\} \neq \emptyset$, then $\bigsqcap S$ *exists* $\in P$ and $\bigsqcap S = \bigsqcup \{p \in P \mid p \sqsubseteq S\}$.
2. Let $(P, \sqsubseteq)$ be a complete meet semi-lattice, and let $S \subseteq P$ be a subset of P.

   If there is an upper bound for S in P, i.e, if $\{p \in P \mid S \sqsubseteq p\} \neq \emptyset$, then $\bigsqcup S$ *exists* $\in P$ and $\bigsqcup S = \bigsqcap \{p \in P \mid S \sqsubseteq p\}$.

Relating Semi-Lattices and Complete Lattices

Lemma A.4.1.13 (Semi-Lattices, Complete Lattices)

If $(P, \sqsubseteq)$ is a complete

1. join semi-lattice with $\bigsqcup \emptyset$ *exists* $\in P$
2. meet semi-lattice with $\bigsqcap \emptyset$ *exists* $\in P$

then $(P, \sqsubseteq)$ is a complete lattice.

Lattices and Complete Partial Orders

Lemma A.4.1.14 (Lattices and CCPOs, DCPOs)

If $(P, \sqsubseteq)$ is a complete lattice, then $(P, \sqsubseteq)$ is a CCPO and a DCPO.

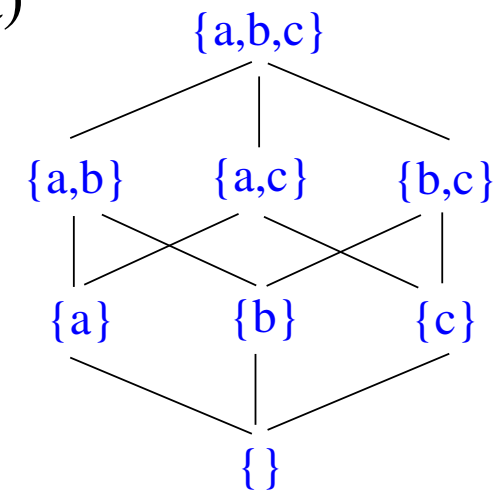
Corollary A.4.1.15 (Finite Lattices, CCPOs, DCPOs)

If $(P, \sqsubseteq)$ is a finite lattice, then $(P, \sqsubseteq)$ is a CCPO and a DCPO.

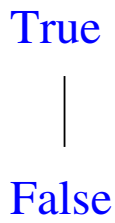
Note: Lemma A.4.1.14 does not hold for lattices.

Examples of Complete Lattices

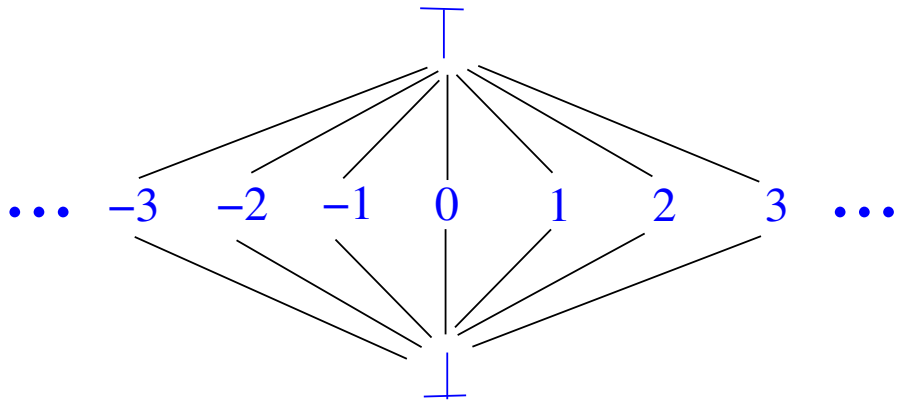
a)



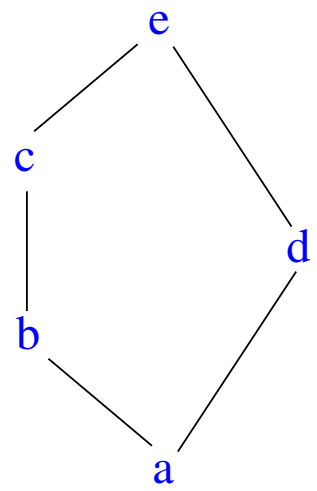
b)



c)

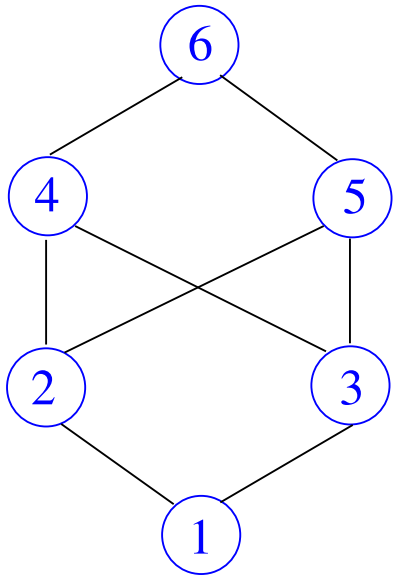


d)



(Anti-) Examples

- The partial order $(P, \sqsubseteq)$ given by the below Hasse diagram is not a lattice (while it is a CCPO and a DCPO).



- $(\mathcal{P}_{fin}(\mathbb{N}), \subseteq)$ is not a complete lattice (and not a CCPO and not a DCPO).

Exercise

Which of the partial orders given by the below Hasse diagrams are lattices? Which ones are complete lattices?

a)

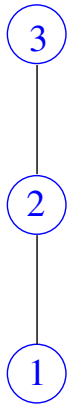
{ }

b)

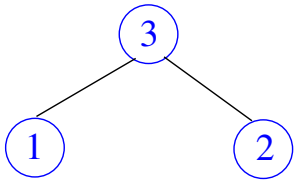
c)

d)

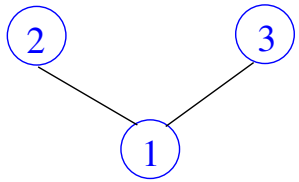
e)



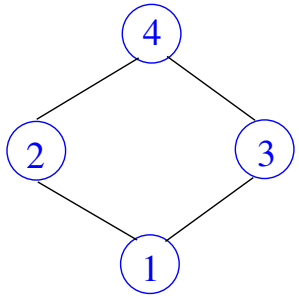
f)



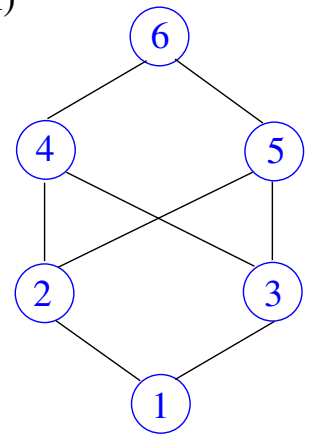
g)



h)



i)



Descending, Ascending Chain Condition

Let $(P, \sqsubseteq)$ be a lattice.

Definition A.4.1.14 (Chain Condition)

P satisfies the

1. descending chain condition, if every descending chain gets stationary, i.e., for every chain $p_1 \sqsupseteq p_2 \sqsupseteq \ldots \sqsupseteq p_n \sqsupseteq \ldots$ there is an index $m \geq 1$ with $p_m = p_{m+j}$ for all $j \in \mathbb{N}$.
2. ascending chain condition, if every ascending chain gets stationary, i.e., for every chain $p_1 \sqsubseteq p_2 \sqsubseteq \ldots \sqsubseteq p_n \sqsubseteq \ldots$ there is an index $m \geq 1$ with $p_m = p_{m+j}$ for all $j \in \mathbb{N}$.

Distributive and Additive Functions on Lattices

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \to P]$ be a function on P.

Definition A.4.1.15 (Distributive, Additive Function)

f is called

- distributive (or $\sqcap$-continuous) iff f is monotonic and
$$\forall P' \subseteq P.\ f(\textstyle\bigsqcap P') = \bigsqcap f(P')$$
(Preservation of greatest lower bounds)
- additive (or $\sqcup$-continuous) iff f is monotonic and
$$\forall P' \subseteq P.\ f(\textstyle\bigsqcup P') = \bigsqcup f(P')$$
(Preservation of least upper bounds)

Note: $\forall S \subseteq P.\ f(S) =_{df} \{ f(s) \mid s \in S \}$

Characterizing Monotonicity

...in terms of the preservation of greatest lower and least upper bounds:

Lemma A.4.1.16 (Characterizing Monotonicity)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \to P]$ be a function on P. Then:

$$f \text{ is monotonic} \iff \forall\, P' \subseteq P.\ f(\bigsqcap P') \sqsubseteq \bigsqcap f(P')$$
$$\iff \forall\, P' \subseteq P.\ f(\bigsqcup P') \sqsupseteq \bigsqcup f(P')$$

Note: $\forall\, S \subseteq P.\ f(S) =_{df} \{\, f(s) \mid s \in S \}$

Useful Results on Mon., Distr., and Additivity

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \to P]$ be a function on P.

Lemma A.4.1.17

f is distributive iff f is additive.

Lemma A.4.1.18

f is monotonic, if f is distributive (or additive).

(i.e., distributivity (or additivity) implies monotonicity.)

A.4.2

Verbandshomomorphismen, Verbands-isomorphismen

Lattice Homomorphisms, Lattice Isomorphisms

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be two lattices, and let $f \in [P \to R]$ be a function from P to R.

Definition A.4.2.1 (Lattice Homorphism)

f is called a lattice homomorphism, if

$$\forall\, p, q \in P.\ f(p \sqcup_P q) = f(p) \sqcup_Q f(q) \wedge f(p \sqcap_P q) = f(p) \sqcap_Q f(q)$$

Definition A.4.2.2 (Lattice Isomorphism)

1. f is called a lattice isomorphism, if f is a lattice homomorphism and bijective.
2. $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ are called isomorphic, if there is lattice isomorphism between P and R.

Useful Results (1)

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be two lattices, and let $f \in [P \to R]$ be a function from P to R.

Lemma A.4.2.3

$$f \in [P \overset{hom}{\to} R] \Rightarrow f \in [P \overset{mon}{\to} R]$$

The reverse implication of Lemma A.4.2.3 does not hold, however, the following weaker relation holds:

Lemma A.4.2.4

$$f \in [P \overset{mon}{\to} R] \Rightarrow$$
$$\forall\, p, q \in P.\ f(p \sqcup_P q) \sqsupseteq_Q f(p) \sqcup_Q f(q) \wedge$$
$$f(p \sqcap_P q) \sqsubseteq_Q f(p) \sqcap_Q f(q)$$

Useful Results (2)

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be two lattices, and let $f \in [P \to R]$ be a function from P to R.

Lemma A.4.2.5

$$f \in [P \xrightarrow{iso} R] \;\Rightarrow\; f^{-1} \in [R \xrightarrow{iso} P]$$

Lemma A.4.2.6

$$f \in [P \xrightarrow{iso} R] \iff f \in [P \xrightarrow{po-hom} R] \text{ wrt } \sqsubseteq_P \text{ and } \sqsubseteq_Q$$

A.4.3

Modulare, distributive und Boolesche Verbände

Modular Lattices

Let $(P, \sqsubseteq)$ be a lattice with meet operation $\sqcap$ and join operation $\sqcup$.

Lemma A.4.3.1

$$\forall\, p, q, r \in P.\ p \sqsubseteq r \Rightarrow p \sqcup (q \sqcap r) \sqsubseteq (p \sqcup q) \sqcap r$$

Definition A.4.3.2 (Modular Lattice)

$(P, \sqsubseteq)$ is called modular, if

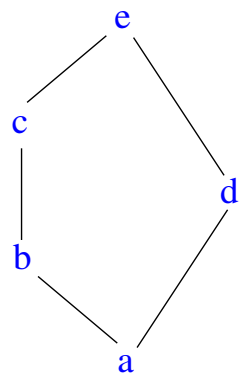
$$\forall\, p, q, r \in P.\ p \sqsubseteq r \Rightarrow p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap r$$

Characterizing Modular Lattices

Let $(P, \sqsubseteq)$ be a lattice.

Theorem A.4.3.3 (Characterizing Modular Lat. I)

$(P, \sqsubseteq)$ is not modular iff $(P, \sqsubseteq)$ contains a sublattice, which is isomorphic to the below lattice:



Theorem A.4.3.4 (Characterizing Modular Lat. II)

$(P, \sqsubseteq)$ is modular iff

$$\forall\, p, q, r \in P.\ p \sqsubseteq q,\ p \sqcap r = q \sqcap r,\ p \sqcup r = q \sqcup r \Rightarrow p = q$$

Distributive Lattices

Let $(P, \sqsubseteq)$ be a lattice with meet operation $\sqcap$ and join operation $\sqcup$.

Lemma A.4.4.5

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.4.3.6 (Distributive Lattice)

$(P, \sqsubseteq)$ is called distributive, if

1. $\forall\, p, q, r \in P.\ p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap (p \sqcup r)$
2. $\forall\, p, q, r \in P.\ p \sqcap (q \sqcup r) = (p \sqcap q) \sqcup (p \sqcap r)$

Towards Characterizing Distributive Lattices

Lemma A.4.3.7

The following two statements are equivalent:

1. $\forall\, p, q, r \in P.\ p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap (p \sqcup r)$
2. $\forall\, p, q, r \in P.\ p \sqcap (q \sqcup r) = (p \sqcap q) \sqcup (p \sqcap r)$

Hence, it is sufficient to require the validity of property (1) or of property (2) in Definition A.4.3.6.

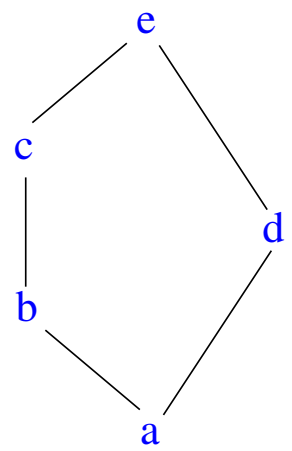
Characterizing Distributive Lattices

Let $(P, \sqsubseteq)$ be a lattice.

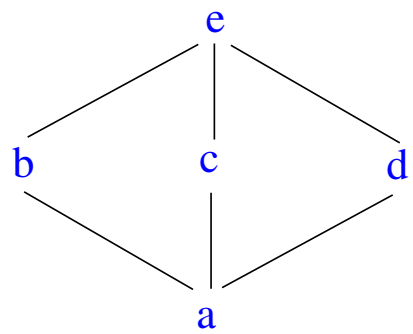
Theorem A.4.3.8 (Characterizing Distributive Lat.)

$(P, \sqsubseteq)$ is not distributive iff $(P, \sqsubseteq)$ contains a sublattice, which is isomorphic to one of the below two lattices:

a)



b)



Corollary A.4.3.9

If $(P, \sqsubseteq)$ is distributive, then $(P, \sqsubseteq)$ is modular.

Boolean Lattices

Let $(P, \sqsubseteq)$ be a lattice with meet operation $\sqcap$, join operation $\sqcup$, least element $\bot$, and greatest element $\top$.

Definition A.4.3.10 (Complement)

Let $p, q \in P$. Then:

1. q is called a complement of p, if $p \sqcup q = \top$ and $p \sqcap q = \bot$.
2. P is called complementary, if every element in P has a complement.

Definition A.4.3.11 (Boolean Lattice)

$(P, \sqsubseteq)$ is called Boolean, if it is complementary, distributive, and $\bot \neq \top$.

Note: If $(P, \sqsubseteq)$ is Boolean, then every element $p \in P$ has an unambiguous unique complement in P, which is denoted by $\bar{p}$.

Useful Result

Lemma A.4.3.12

Let $(P, \sqsubseteq)$ be a Boolean lattice, and let $p, q, r \in P$. Then:

1. $\bar{\bar{p}} = p$ (Involution)
2. $\overline{p \sqcup q} = \bar{p} \sqcap \bar{q}, \quad \overline{p \sqcap q} = \bar{p} \sqcup \bar{q}$ (De Morgan)
3. $p \sqsubseteq q \iff \bar{p} \sqcup q = \top \iff p \sqcap \bar{q} = \bot$
4. $p \sqsubseteq q \sqcup r \iff p \sqcap \bar{q} \sqsubseteq r \iff \bar{q} \sqsubseteq \bar{p} \sqcup r$

Boolean L. Homomorphisms, L. Isomorphisms

Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be two Boolean lattices, and let $f \in [P \to Q]$ be a function from P to Q.

Definition A.4.3.13 (Boolean Lattice Homorphism)

f is called a Boolean lattice homomorphism, if f is a lattice homomorphism and

$$\forall\, p \in P.\ f(\bar{p}) = \overline{f(p)}$$

Definition A.4.3.14 (Boolean Lattice Isomorphism)

f is called a Boolean lattice isomorphism, if f is a Boolean lattice homomorphism and bijective.

Useful Results

Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be two Boolean lattices, and let $f \in [P \overset{bhom}{\to} Q]$ be a Boolean lattice homomorphism from P to Q.

Lemma A.4.3.14

$$f(\bot) = \bot \wedge f(\top) = \top$$

Lemma A.4.3.15

f is a Boolean lattice isomorphism iff $f(\bot) = \bot \wedge f(\top) = \top$

A.4.4

Verbandskonstruktionen

Lattice Constructions: Flat Lattices

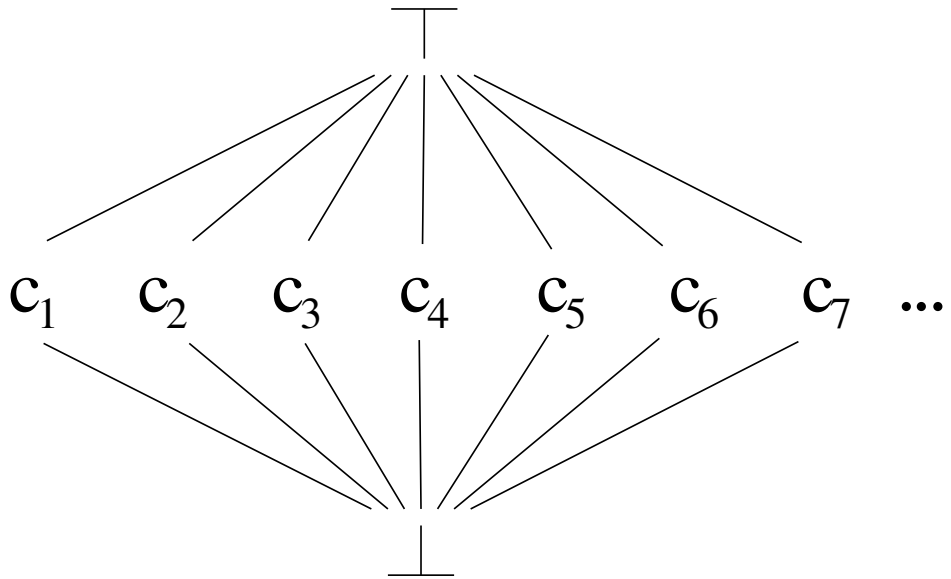
Lemma A.4.4.1 (Flat Construction)

Let C be a set. Then:

$(C \dot\cup \{\bot, \top\}, \sqsubseteq_{flat})$ with $\sqsubseteq_{flat}$ defined by

$$\forall\, c, d \in C \dot\cup \{\bot, \top\}.\ c \sqsubseteq_{flat} d \Leftrightarrow c = \bot \ \vee\ c = d \ \vee\ d = \top$$

is a complete lattice, a so-called flat lattice (or diamond lattice).



Lattice Constructions: Products, Sums,...

Like the principle underlying the construction of flat CPOs and flat lattices, also CPO construction principles for

- non-strict products
- strict products
- separate sums
- coalesced sums
- continuous (here: additive, distributive) function spaces

carry over to lattices and complete lattices (cf. Appendix A.3.2).

A.4.5

Algebraische und ordnungstheoretische Verbandssicht

Motivation

In Definition A.4.1.1, we introduced lattices in terms of

- ordered sets $(P, \sqsubseteq)$, which induces an order-theoretic view of lattices.

Alternatively, lattices can be introduced in terms of

- algebraic structures $(P, \sqcap, \sqcup)$, which induces an algebraic view of lattices.

Next, we will show that both views are equivalent in the sense that a lattice defined order-theoretically can be considered algebraically and vice versa.

Lattices as Algebraic Structures

Definition A.4.5.1 (Algebraic Lattice)

An algebraic lattice is an algebraic structure $(P, \sqcap, \sqcup)$, where

- $P \neq \emptyset$ is a non-empty set
- $\sqcap, \sqcup : P \times P \to P$ are two maps such that for all $p, q, r \in P$ the following laws hold (infix notation):
  - Commutative Laws: $p \sqcap q = q \sqcap p$, $p \sqcup q = q \sqcup p$
  - Associative Laws: $(p \sqcap q) \sqcap r = p \sqcap (q \sqcap r)$, $(p \sqcup q) \sqcup r = p \sqcup (q \sqcup r)$
  - Absorption Laws: $(p \sqcap q) \sqcup p = p$, $(p \sqcup q) \sqcap p = p$

Properties of Algebraic Lattices

Let $(P, \sqcap, \sqcup)$ be an algebraic lattice.

Lemma A.4.5.2 (Idempotency Laws)

For all $p \in P$, the maps $\sqcap, \sqcup : P \times P \to P$ satisfy the following law:

- Idempotency Laws: $p \sqcap p = p$
  $p \sqcup p = p$

Lemma A.4.5.3

For all $p, q \in P$, the maps $\sqcap, \sqcup : P \times P \to P$ satisfy:

1. $p \sqcap q = p \iff p \sqcup q = q$
2. $p \sqcap q = p \sqcup q \iff p = q$

Induced (Partial) Order

Let $(P, \sqcap, \sqcup)$ be an algebraic lattice.

Lemma A.4.5.4

The relation $\sqsubseteq \subseteq P \times P$ on P defined by

$$\forall\, p, q \in P.\ p \sqsubseteq q \iff_{df} p \sqcap q = p$$

is a partial order relation on P, i.e., $\sqsubseteq$ is reflexive, transitive, and antisymmetric.

Definition A.4.5.5 (Induced Partial Order)

The relation $\sqsubseteq$ defined in Lemma A.4.5.4 is called the induced partial order of $(P, \sqcap, \sqcup)$.

Properties of the Induced Partial Order

Let $(P, \sqcap, \sqcup)$ be an algebraic lattice, and let $\sqsubseteq$ be the induced partial order of $(P, \sqcap, \sqcup)$.

Lemma A.4.5.6

For all $p, q \in P$, the infimum ($\widehat{=}$ greatest lower bound) and the supremum ($\widehat{=}$ least upper bound) of the set $\{p, q\}$ exists and is given by the image of $\sqcap$ and $\sqcup$ applied to p and q, respectively, i.e.,

$$\forall\, p, q \in P.\ \bigsqcap\{p, q\} = p \sqcap q \ \wedge\ \bigsqcup\{p, q\} = p \sqcup q$$

Lemma A.4.5.6 can inductively be extended yielding:

Lemma A.4.5.7

Let $\emptyset \neq Q \subseteq P$ be a finite non-empty subset of P. Then:

$$\exists\, glb, lub \in P.\ glb = \bigsqcap Q \ \wedge\ glb = \bigsqcup Q$$

Algebraic Lattices Order-theoretically

Corollary A.4.5.8 (From $(P, \sqcap, \sqcup)$ to $(P, \sqsubseteq)$)

Let $(P, \sqcap, \sqcup)$ be an algebraic lattice. Then:

$(P, \sqsubseteq)$, where $\sqsubseteq$ is the induced partial order of $(P, \sqcap, \sqcup)$, is an order-theoretic lattice in the sense of Definition A.4.1.1.

Induced Algebraic Maps

Let $(P, \sqsubseteq)$ be an order-theoretic lattice.

Definition A.4.5.9 (Induced Algebraic Maps)

The partial order $\sqsubseteq$ of $(P, \sqsubseteq)$ induces two maps $\sqcap$ and $\sqcup$ from $P \times P$ to P defined by

1. $\forall\, p, q \in P.\ p \sqcap q =_{df} \bigsqcap \{p, q\}$
2. $\forall\, p, q \in P.\ p \sqcup q =_{df} \bigsqcup \{p, q\}$

Properties of the Induced Algebraic Maps (1)

Let $(P, \sqsubseteq)$ be an order-theoretic lattice, and let $\sqcap$ and $\sqcup$ be the induced maps of $(P, \sqsubseteq)$.

Lemma A.4.5.10

Let $p, q \in P$. Then the following statements are equivalent:

1. $p \sqsubseteq q$
2. $p \sqcap q = p$
3. $p \sqcup q = q$

Properties of the Induced Algebraic Maps (2)

Let $(P, \sqsubseteq)$ be an order-theoretic lattice, and let $\sqcap$ and $\sqcup$ be the induced maps of $(P, \sqsubseteq)$.

Lemma A.4.5.11

The induced maps $\sqcap$ and $\sqcup$ satisfy, for all $p, q, r \in P$,

- Commutative Laws: $p \sqcap q = q \sqcap p$
  $p \sqcup q = q \sqcup p$
- Associative Laws: $(p \sqcap q) \sqcap r = p \sqcap (q \sqcap r)$
  $(p \sqcup q) \sqcup r = p \sqcup (q \sqcup r)$
- Absorption Laws: $(p \sqcap q) \sqcup p = p$
  $(p \sqcup q) \sqcap p = p$
- Idempotency Laws: $p \sqcap p = p$
  $p \sqcup p = p$

Order-theoretic Lattices Algebraically

Corollary A.4.5.12 (From $(P, \sqsubseteq)$ to $(P, \sqcap, \sqcup)$)

Let $(P, \sqsubseteq)$ be an order-theoretic lattice. Then:

$(P, \sqcap, \sqcup)$, where $\sqcap$ and $\sqcup$ are the induced maps of $(P, \sqcap, \sqcup)$, is an algebraic lattice in the sense of Definition A.4.5.1.

Equivalence of Order-theoretic and Algebraic View of a Lattice (1)

From order-theoretic to algebraic lattices:

- An order-theoretic lattice $(P, \sqsubseteq)$ can be considered algebraically by switching from $(P, \sqsubseteq)$ to $(P, \sqcap, \sqcup)$, where $\sqcap$ and $\sqcup$ are the induced maps of $(P, \sqsubseteq)$.

From algebraic to order-theoretic lattices:

- An algebraic lattice $(P, \sqcap, \sqcup)$ can be considered order-theoretically by switching from $(P, \sqcap, \sqcup)$ to $(P, \sqsubseteq)$, where $\sqsubseteq$ is the induced partial order of $(P, \sqcap, \sqcup)$.

Equivalence of Order-theoretic and Algebraic View of a Lattice (2)

Together, this allows us to simply speak of a lattice P, and to speak only more precisely of P as an

- order-theoretic lattice $(P, \sqsubseteq)$
- algebraic lattice $(P, \sqcap, \sqcup)$

if we want to emphasize that we think of P as a special ordered set or as a special algebraic structure.

Bottom and Top vs. Zero and One (1)

Let P be a lattice with a least and a greatest element.

Considering P

- order-theoretically as $(P, \sqsubseteq)$, it is appropriate to think of its least and greatest element in terms of bottom $\bot$ and top $\top$ with
  - $\bot = \bigsqcup \emptyset$
  - $\top = \bigsqcap \emptyset$
- algebraically as $(P, \sqcap, \sqcup)$, it is appropriate to think of its least and greatest element in terms of zero $\mathbf{0}$ and one $\mathbf{1}$, where $(P, \sqcap, \sqcup)$ is said to have a
  - zero element, if $\exists\, \mathbf{0} \in P.\ \forall\, p \in P.\ p \sqcup \mathbf{0} = p$
  - one element, if $\exists\, \mathbf{1} \in P.\ \forall\, p \in P.\ p \sqcap \mathbf{1} = p$

Bottom and Top vs. Zero and One (2)

Lemma A.4.5.13

Let P be a lattice. Then:

- $(P, \sqsubseteq)$ has a top element $\top$ iff $(P, \sqcap, \sqcup)$ has a one element $\mathbf{1}$, and in that case $\bigsqcap \emptyset = \top = \mathbf{1}$.
- $(P, \sqsubseteq)$ has a bottom element $\bot$ iff $(P, \sqcap, \sqcup)$ has a zero element $\mathbf{0}$, and in that case $\bigsqcup \emptyset = \bot = \mathbf{0}$.

On the Adequacy of the Order-theoretic and the Algebraic View of a Lattice

In mathematics, usually the

- algebraic view of a lattice is more appropriate as it is in line with other algebraic structures ("a set together with some maps satisfying a number of laws"), e.g., groups, rings, fields, vector spaces, categories, etc., which are investigated and dealt with in mathematics.

In computer science, usually the

- order-theoretic view of a lattice is more appropriate, since the order relation can often be interpreted and understood as "$\cdot$ carries more/less information than $\cdot$," "$\cdot$ is more/less defined than $\cdot$," "$\cdot$ is stronger/weaker than $\cdot$," etc., which often fits naturally to problems investigated and dealt with in computer science.

A.5

Fixpunkttheoreme

Fixed Points of Functions

Definition A.5.1 (Fixed Point)

Let M be a set, let $f \in [M \to M]$ be a function on M, and let $m \in M$ be an element of M. Then:

m is called a fixed point of f iff $f(m) = m$.

Least, Greatest Fixed Points in Partial Orders

Definition A.5.2 (Least, Greatest Fixed Point)

Let $(P, \sqsubseteq)$ be a partial order, let $f \in [P \to P]$ be a function on P, and let p be a fixed point of f, i.e., $f(p) = p$. Then:

p is called the

- least fixed point of f, denoted by μf,
  iff $\forall\, q \in P.\ f(q) = q \Rightarrow p \sqsubseteq q$
- greatest fixed point of f, denoted by νf,
  iff $\forall\, q \in P.\ f(q) = q \Rightarrow q \sqsubseteq p$

Towers in Chain Complete Partial Orders

Definition A.5.3 (f-Tower in C)

Let $(C, \sqsubseteq)$ be a CCPO, let $f \in [C \to C]$ be a function on C, and let $T \subseteq C$ be a subset of C. Then:

T is called an f-tower in C iff

1. $\bot \in T$.
2. If $t \in T$, then also $f(t) \in T$.
3. If $T' \subseteq T$ is a chain in C, then $\bigsqcup T' \in T$.

Least Towers in Chain Complete Partial Orders

Lemma A.5.4 (The Least f-Tower in C)

The intersection

$$I =_{df} \bigcap \{ T \mid T \text{ } f\text{-tower in } C \}$$

of all f-towers in C is the least f-tower in C, i.e.,

1. I is an f-tower in C.
2. $\forall\, T$ f-tower in C. $I \subseteq T$.

Lemma A.5.5 (Least f-Towers and Chains)

The least f-tower in C is a chain in C, if f is expanding.

A.5.1

Fixpunkttheoreme für vollständige partielle Ordnungen

Fixed Points of Exp./Monotonic Functions

Fixed Point Theorem A.5.1.1 (Expanding Function)

Let $(C, \sqsubseteq)$ be a CCPO, and let $f \in [C \overset{exp}{\to} C]$ be an expanding function on C. Then:

The supremum of the least f-tower in C is a fixed point of f.

Fixed Point Theorem A.5.1.2 (Monotonic Function)

Let $(C, \sqsubseteq)$ be a CCPO, and let $f \in [C \overset{mon}{\to} C]$ be a monotonic function on C. Then:

f has a unique least fixed point μf, which is given by the supremum of the least f-tower in C.

Note

- Theorem A.5.1.1 and Theorem A.5.1.2 ensure the existence of a fixed point for expanding functions and of a unique least fixed point for monotonic functions, respectively, but do not provide constructive procedures for computing or approximating them.
- This is in contrast to Theorem A.5.1.3, which does so for continuous functions. In practice, continuous functions are thus more important and considered where possible.

Least Fixed Points of Continuous Functions

Fixed Point Theorem A.5.1.3 (Knaster,Tarski,Kleene)

Let $(C, \sqsubseteq)$ be a CCPO, and let $f \in [C \overset{con}{\to} C]$ be a continuous function on C. Then:

f has a unique least fixed point $\mu f \in C$, which is given by the supremum of the (so-called) Kleene chain $\{\bot, f(\bot), f^2(\bot), \ldots\}$, i.e.

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

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

Proof of Fixed Point Theorem A.5.1.3 (1)

We have to prove:

$$\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) = \bigsqcup \{ f^i(\bot) \mid i \geq 0 \}$$

1. exists,
2. is a fixed point of f,
3. is the least fixed point of f.

Proof of Fixed Point Theorem A.5.1.3 (2)

1. Existence

- By definition of $\bot$ as the least element of C and of f^0 as the identity on C we have: $\bot = f^0(\bot) \sqsubseteq f^1(\bot) = f(\bot)$.
- Since f is continuous and hence monotonic, we obtain by means of (natural) induction:
  $\forall\, i, j \in \mathbb{N}_0.\ i < j \Rightarrow f^i(\bot) \sqsubseteq f^{i+1}(\bot) \sqsubseteq f^j(\bot)$.
- Hence, the set $\{f^i(\bot) \mid i \geq 0\}$ is a (possibly infinite) chain in C.
- Since $(C, \sqsubseteq)$ is a CCPO and $\{f^i(\bot) \mid i \geq 0\}$ a chain in C, this implies by definition of a CPO that the least upper bound of the chain $\{f^i(\bot) \mid i \geq 0\}$

$$\bigsqcup \{f^i(\bot) \mid i \geq 0\} = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \text{ exists.}$$

Proof of Fixed Point Theorem A.5.1.3 (3)

2. Fixed point property

$$
\begin{aligned}
& & & f(\bigsqcup_{i\in\mathbb{N}_0} f^i(\bot)) \\
(f \text{ continuous}) & = & & \bigsqcup_{i\in\mathbb{N}_0} f(f^i(\bot)) \\
& = & & \bigsqcup_{i\in\mathbb{N}_1} f^i(\bot) \\
(C' =_{df} \{f^i\bot \mid i \geq 1\} \text{ is a chain} \Rightarrow & & & \\
\bigsqcup C' \text{ exists } = \bot \sqcup \bigsqcup C') & = & & \bot \sqcup \bigsqcup_{i\in\mathbb{N}_1} f^i(\bot) \\
(f^0(\bot) =_{df} \bot) & = & & \bigsqcup_{i\in\mathbb{N}_0} f^i(\bot)
\end{aligned}
$$

Proof of Fixed Point Theorem A.5.1.3 (4)

3. Least fixed point property

- Let c be an arbitrary fixed point of f. Then: $\bot \sqsubseteq c$.
- Since f is continuous and hence monotonic, we obtain by means of (natural) induction:
  $\forall\, i \in \mathbb{N}_0.\ f^i(\bot) \sqsubseteq f^i(c)\ (= c)$.
- Since c is a fixed point of f, this implies:
  $\forall\, i \in \mathbb{N}_0.\ f^i(\bot) \sqsubseteq c\ (= f^i(c))$.
- Thus, c is an upper bound of the set $\{f^i(\bot) \mid i \in \mathbb{N}_0\}$.
- Since $\{f^i(\bot) \mid i \in \mathbb{N}_0\}$ is a chain, and $\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)$ is by definition the least upper bound of this chain, we obtain the desired inclusion

$$\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \sqsubseteq c.$$

□

Least Conditional Fixed Points

Let $(C, \sqsubseteq)$ be a CCPO, let $f \in [C \to C]$ be a function on C, and let $d, c_d \in C$ be elements of C.

Definition A.5.1.4 (Least Conditional Fixed Point)

c_d is called the

- least conditional fixed point of f wrt d (in German: kleinster bedingter Fixpunkt) iff c_d is the least fixed point of C with $d \sqsubseteq c_d$, i.e.,
  $\forall x \in C.\ f(x) = x \ \wedge \ d \sqsubseteq x \Rightarrow c_d \sqsubseteq x$.

Least Cond. Fixed Points of Cont. Functions

Theorem A.5.1.5 (Conditional Fixed Point Theorem)

Let $(C, \sqsubseteq)$ be a CCPO, let $d \in C$, and let $f \in [C \stackrel{con}{\to} C]$ be a continuous function on C which is expanding for d, i.e., $d \sqsubseteq f(d)$. Then:

f has a least conditional fixed point $\mu f_d \in C$, which is given by the supremum of the (generalized) Kleene chain $\{d, f(d), f^2(d), \ldots\}$, i.e.

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

Finite Fixed Points

Let $(C, \sqsubseteq)$ be a CCPO, let $d \in C$, and let $f \in [C \overset{mon}{\to} C]$ be a monotonic function on C.

Theorem A.5.1.6 (Finite Fixed Point Theorem)

If two succeeding elements in the Kleene chain of f are equal, i.e., if there is some $i \in \mathbb{N}$ with $f^i(\bot) = f^{i+1}(\bot)$, then we have: $\mu f = f^i(\bot)$.

Theorem A.5.1.7 (Finite Conditional FP Theorem)

If f is expanding for d, i.e., $d \sqsubseteq f(d)$, and two succeeding elements in the (generalized) Kleene chain of f wrt d are equal, i.e., if there is some $i \in \mathbb{N}$ with $f^i(d) = f^{i+1}(d)$, then we have: $\mu f_d = f^i(d)$.

Note: Theorems A.5.1.6 and A.5.1.7 do not require continuity of f. Monotonicity (and expandingness) of f suffice(s).

Towards the Existence of Finite Fixed Points

Let $(P, \sqsubseteq)$ be a partial order, and let $p, r \in P$.

Definition A.5.1.8 (Chain-finite Partial Order)

$(P, \sqsubseteq)$ is called

- chain-finite (in German: kettenendlich) iff P does not contain an infinite chain.

Definition A.5.1.9 (Finite Element)

p is called

- finite iff the set $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$ does not contain an infinite chain.
- finite relative to r iff the set $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$ does not contain an infinite chain.

Existence of Finite Fixed Points

There are numerous conditions, which often hold in practice and are sufficient to ensure the existence of a least finite fixed point of a function f (cf. Nielson/Nielson 1992), e.g.

- the domain or the range of f are finite or chain-finite,
- the least fixed point of f is finite,
- f is of the form $f(c) = c \sqcup g(c)$ with g a monotonic function on a chain-finite (data) domain.

Fixed Point Theorems, DCPOs, and Lattices

Note: Complete lattices (cf. Lemma A.4.1.13) and DCPOs with a least element (cf. Lemma A.3.1.5) are CCPOs, too.

Thus, we can conclude:

Corollary A.5.1.10 (Fixed Points, Lattices, DCPOs)

The fixed point theorems of Chapter A.5.1 hold for functions on complete lattices and on DCPOs with a least element, too.

A.5.2

Fixpunkttheoreme für Verbände

Fixed Points of Monotonic Functions

Fixed Point Theorem A.5.2.1 (Knaster, Tarski)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \overset{mon}{\to} P]$ be a monotonic function on P. Then:

1. f has a unique least fixed point $\mu f \in P$, which is given by $\mu f = \bigsqcap \{p \in P \mid f(p) \sqsubseteq p\}$.
2. f has a unique greatest fixed point $\nu f \in P$, which is given by $\nu f = \bigsqcup \{p \in P \mid p \sqsubseteq f(p)\}$.

Characterization Theorem A.5.2.2 (Davis)

Let $(P, \sqsubseteq)$ be a lattice. Then:

$(P, \sqsubseteq)$ is complete iff every $f \in [P \overset{mon}{\to} P]$ has a fixed point.

The Fixed Point Lattice of Mon. Functions

Theorem A.5.2.2 (Lattice of Fixed Points)

Let $(P, \sqsubseteq)$ be a complete lattice, let $f \in [P \stackrel{mon}{\to} P]$ be a monotonic function on P, and let $Fix(f) =_{df} \{p \in P \mid f(p) = p\}$ be the set of all fixed points of f. Then:

Every subset $F \subseteq Fix(f)$ has a supremum and an infimum in $Fix(f)$, i.e., $(Fix(f), \sqsubseteq_{\mid Fix(f)})$ is a complete lattice.

Theorem A.5.2.3 (Ordering of Fixed Points)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \stackrel{mon}{\to} P]$ be a monotonic function on P. Then:

$$\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \sqsubseteq \mu f \sqsubseteq \nu f \sqsubseteq \bigsqcap_{i \in \mathbb{N}_0} f^i(\top)$$

Fixed Points of Add./Distributive Functions

For additive and distributive functions, the leftmost and the rightmost inequality of Theorem A.5.2.3 become equalities:

Fixed Point Theorem A.5.2.4 (Knaster,Tarski,Kleene)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \to P]$ be a function on P. Then:

1. f has a unique least fixed point $\mu f \in P$ given by $\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)$, if f is additive, i.e., $f \in [P \stackrel{add}{\to} P]$.
2. f has a unique greatest fixed point $\nu f \in P$ given by $\nu f = \bigsqcap_{i \in \mathbb{N}_0} f^i(\top)$, if f is distributive, i.e., $f \in [P \stackrel{dis}{\to} P]$.

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

A.6

Fixpunktinduktion

Admissible Predicates

Fixed point induction allows proving properties of fixed points. Essential is the notion of an admissible predicate:

Definition A.6.1 (Admissible Predicate)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $\phi : P \to \mathbb{B}$ be a predicate on P. Then:

ϕ is called admissible (or $\sqcup$-admissible) iff for every chain $C \subseteq P$ holds:

$$(\forall\, c \in C.\ \phi(c)) \Rightarrow \phi(\bigsqcup C)$$

Lemma A.6.2

Let $(P, \sqsubseteq)$ be a complete lattice, and let $\phi : P \to \mathbb{B}$ be an admissible predicate on P. Then: $\phi(\bot) = \mathbf{wahr}$.

Proof. The admissibility of ϕ implies $\phi(\bigsqcup \emptyset) = \mathbf{wahr}$. Moreover, we have $\bot = \bigsqcup \emptyset$, which completes the proof.

Sufficient Conditions for Admissibility

Theorem A.6.3 (Admissibility Condition 1)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $\phi : P \to \mathbb{B}$ be a predicate on P. Then:

ϕ is admissible, if there is a complete lattice $(Q, \sqsubseteq_Q)$ and two additive functions $f, g \in [P \stackrel{add}{\to} Q]$, such that

$$\forall\, p \in P.\ \phi(p) \iff f(p) \sqsubseteq_Q g(p)$$

Theorem A.6.4 (Admissibility Condition 2)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $\phi, \psi : P \to \mathbb{B}$ be two admissible predicates on P. Then:

The conjunction of ϕ and ψ, the predicate $\phi \ \wedge \ \psi$ defined by

$$\forall\, p \in P.\ (\phi \ \wedge \ \psi)(p) =_{df} \phi(p) \wedge \psi(p)$$

is admissible.

Fixed Point Induction on Complete Lattices

Theorem A.6.5 (Fixed Point Induction on C. Lat.)

Let $(P, \sqsubseteq)$ be a complete lattice, let $f \in [P \overset{add}{\to} P]$ be an additive function on P, and let $\phi : P \to \mathbb{B}$ be an admissible predicate on P. Then:

The validity of

- $\forall\, p \in P.\ \phi(p) \Rightarrow \phi(f(p))$ (Induction step)

implies the validity of $\phi(\mu f)$.

Note: The induction base, i.e., the validity of $\phi(\bot)$, is implied by the admissibility of ϕ (cf. Lemma A.6.2) and proved when verifying the admissibility of ϕ.

Fixed Point Induction on CCPOs

The notion of admissibility of a predicate carries over from complete lattices to CCPOs.

Theorem A.6.6 (Fixed Point Induction on CCPOs)

Let $(C, \sqsubseteq)$ be a CCPO, let $f \in [C \overset{mon}{\to} C]$ be a monotonic function on C, and let $\phi : C \to \mathbb{B}$ be an admissible predicate on C. Then:

The validity of

- $\forall\, c \in C.\ \phi(c) \Rightarrow \phi(f(c))$ (Induction step)

implies the validity of $\phi(\mu f)$.

Note: Theorem A.6.6 holds (of course still), if we replace the CCPO $(C, \sqsubseteq)$ by a complete lattice $(P, \sqsubseteq)$.

A.7

Literaturverzeichnis, Leseempfehlungen

Appendix A: Further Reading (1)


André Arnold, Irène Guessarian. *Mathematics for Computer Science*. Prentice Hall, 1996.


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)

Appendix A: Further Reading (2)


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)


Garret Birkhoff. *Lattice Theory*. American Mathematical Society, 3rd edition, 1967.


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)

Appendix A: Further Reading (3)

- 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, 2003. (Chapter 1, First Concepts; Chapter 2, Distributive Lattices; Chapter 3, Congruences and Ideals; Chapter 5, Varieties of Lattices)
- Paul R. Halmos. *Naive Set Theory*. Springer-V., Reprint, 2001. (Chapter 6, Ordered Pairs; Chapter 7, Relations; Chapter 8, Functions)

Appendix A: Further Reading (4)


Hans Hermes. *Einführung in die Verbandstheorie*. Springer-V., 2. Auflage, 1967.


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)

Appendix A: Further Reading (5)

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)

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)

Appendix A: Further Reading (6)


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)


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.

Appendix A: Further Reading (7)

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)

Anhang B

Pragmatik: Flussgraphvarianten

B.1

Motivation

Basisblock- vs. Instruktionsgraphen

...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. Instruktionen benannt sind, und untersuchen ihre jeweiligen

- Vor- und Nachteile für die Programmanalyse

...für eine Antwort auf die Frage:

- Knoten- und kantenbenannte Basisblock- und Instruktionsgraphen: (Nur) eine Ge- schmacksfrage?

En passant werden wir dabei weitere praktisch relevante

- DFA-Probleme und -Analysen

kennenlernen.

Von Basisblockgraphen erhoffte Vorteile

...allgemein wird Basisblockgraphen vor allem folgender Anwendungsvorteil zugeschrieben ('Folklore' (engl. ('folk knowledge')):

Bessere Skalierungseigenschaften und Performanzvorteile, da

- weniger Knoten in die (potentiell) berechnungsaufwändige iterative Fixpunktberechnung involviert sind.
- größere Programme im Hauptspeicher gehalten werden können.

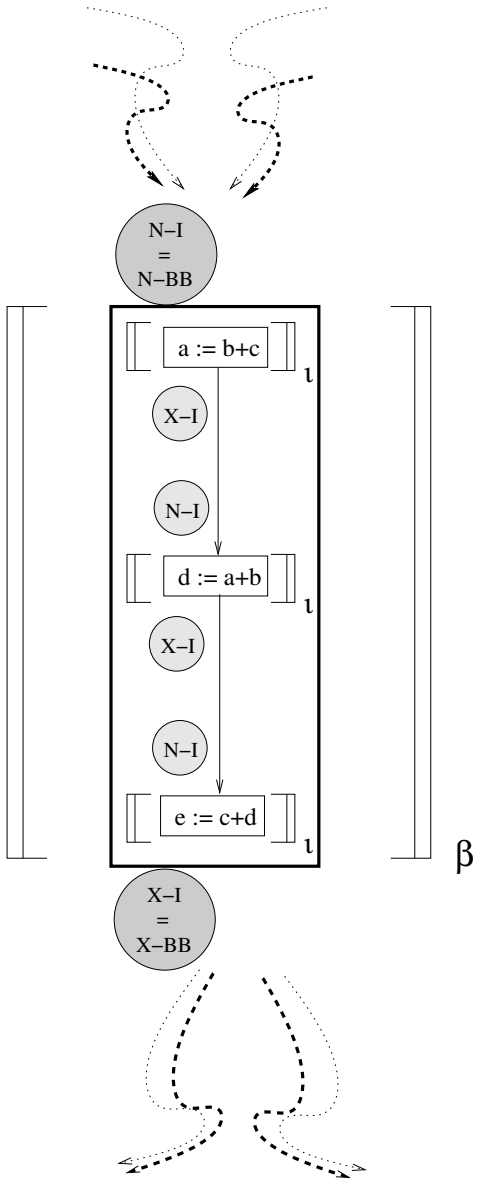
Mit Basisblockgraphen verbundene Nachteile

...sind definitiv auch gegeben, darunter 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- zu Instruktionsgraphen

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 Instruktionsgraphen

SUP_{IG}- und $MaxFP_{IG}$-Ansatz

...für kantenbenannte Instruktionsgraphen.

Die *SUP*-Lösung:

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ SUP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \bigsqcap \{\, \llbracket\, p\,\rrbracket_\iota(c_s) \mid p \in \mathbf{P}_G[s,n] \,\}$$

Die *MaxFP*-Lösung:

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \nu\text{-}inf\,(n)$$

wobei ν-*inf* die größte Lösung des *MaxFP*-Gleichungssystems für Instruktionsgraphen bezeichnet:

$$inf\,(n) = \begin{cases} c_s & \text{falls } n = s \\ \bigsqcap \{\, \llbracket\, (m,n)\,\rrbracket_\iota(inf\,(m)) \mid m \in pred_G(n) \,\} & \text{sonst} \end{cases}$$

B.2.2

Knotenbenannte Basisblockgraphen

Bezeichnungen

...in der Folge bezeichnen wir:

- Basisblockknoten mit fett gesetzten Buchstaben wie **m**, **n**,...
- Instruktionsknoten mit normal gesetzten Buchstaben wie m, n,...

Weiters bezeichnen wir mit

- $[\![\ ]\!]_\beta$
- $[\![\ ]\!]_\iota$

(lokale) abstrakte DFA-Funktionale für Basisblock- bzw. Instruktionsknoten und mit

- *bb*, *start* und *end* drei Abbildungen, die angewendet auf einen Instruktionsknoten *n* bzw. einen Basisblock **n** den Basisblock liefern, zu dem *n* gehört, bzw. den Start- oder Endinstruktionsknoten von **n**.

SUP_{BBG}-Ansatz (1)

...für knotenbenannte Basisblockgraphen.

Die *SUP*-Lösung auf Basisblockebene:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, \mathbf{n} \in \mathbf{N}.\ SUP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\ E\text{-}SUP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ A\text{-}SUP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n})\ )$$

mit

$$E\text{-}SUP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\, \llbracket\ p\ \rrbracket_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s}, \mathbf{n}[\,\}$$

$$A\text{-}SUP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\, \llbracket\ p\ \rrbracket_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s}, \mathbf{n}]\,\}$$

...wobei *E* und *A* für Basisblock-Eingang und -Ausgang stehen.

MOP_{BBG}-Ansatz (2)

...und ihre notwendige Ausdehnung auf die Instruktionsebene:

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ SUP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \\ (\ E\text{-}SUP_{(\llbracket\ \rrbracket_\iota, c_s)}(n),\ A\text{-}SUP_{(\llbracket\ \rrbracket_\iota, c_s)}(n)\ )$$

mit

$$E\text{-}SUP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \begin{cases} E\text{-}SUP_{(\llbracket\ \rrbracket_\beta, c_s)}(bb(n)) \\ \quad \text{falls } n = start(bb(n)) \\ \llbracket\ p\ \rrbracket_\iota(E\text{-}SUP_{(\llbracket\ \rrbracket_\beta, c_s)}(bb(n))) \\ \quad \text{sonst } (p \text{ Präfixpfad von } start(bb(n)) \\ \qquad \text{bis (ausschließlich) } n) \end{cases}$$

$$A\text{-}SUP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \llbracket\ p\ \rrbracket_\iota(E\text{-}SUP_{(\llbracket\ \rrbracket_\beta, c_s)}(bb(n))) \\ (p \text{ Präfixpfad von } start(bb(n)) \text{ bis (ein-} \\ \text{-schließlich) } n)$$

$MaxFP_{BBG}$-Ansatz (1)

...für knotenbenannte Basisblockgraphen:

Die *MaxFP*-Lösung auf Basisblockgraphebene:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\, \forall\, \mathbf{n} \in \mathbf{N}.\ MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\ E\text{-}MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ A\text{-}MaxFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n})\ )$$

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:

$$E\text{-}inf(\mathbf{n}) = \begin{cases} c_{\mathbf{s}} & \text{falls } \mathbf{n} = \mathbf{s} \\ \bigsqcap \{\, A\text{-}inf(\mathbf{m}) \mid \mathbf{m} \in pred_{\mathbf{G}}(\mathbf{n}) \,\} & \text{sonst} \end{cases}$$
$$A\text{-}inf(\mathbf{n}) = \llbracket\, \mathbf{n} \,\rrbracket_\beta(E\text{-}inf(\mathbf{n}))$$

$MaxFP_{BBG}$-Ansatz (2)

...und ihre notwendige Ausdehnung auf die Instruktionsebene:

$$\forall\, c_s \in \mathcal{C}\, \forall\, n \in N.\ MaxFP_{(\![\ ]\!]_\iota, c_s)}(n) =_{df} \\ (\ E\text{-}MaxFP_{(\![\ ]\!]_\iota, c_s)}(n),\ A\text{-}MaxFP_{(\![\ ]\!]_\iota, c_s)}(n)\ )$$

mit

$$E\text{-}MaxFP_{(\![\ ]\!]_\iota, c_s)}(n) =_{df} \nu\text{-}E\text{-}inf(n)$$
$$A\text{-}MaxFP_{(\![\ ]\!]_\iota, c_s)}(n) =_{df} \nu\text{-}A\text{-}inf(n)$$

...wobei ν-E-inf und ν-A-inf die größten Lösungen des $MaxFP$-Gleichungssystems für Instruktionsknoten bezeichnen:

$$E\text{-}inf(n) = \begin{cases} \nu\text{-}E\text{-}inf(bb(n)) & \text{falls } n = start(bb(n)) \\ A\text{-}inf(m) & \text{sonst (wobei } m \text{ der eindeutig bestimmte Vorgänger von } n \text{ in } bb(n) \text{ ist)} \end{cases}$$

$$A\text{-}inf(n) = [\![\ n\ ]\!]_\iota(E\text{-}inf(n))$$

Kapitel B.3

Verfügbare Ausdrücke

B.3.1

Knotenbenannte Basisblockgraphen

Verfügbare Ausdrücke (1)

...für knotenbenannte Basisblockgraphen.

Phase I: Die Basisblockebene

Lokale Prädikate (assoziiert mit Basisblockknoten):

- BB-XComp$_\beta(t)$: t wird von einer Anweisung ι in β berechnet und weder ι noch eine auf ι 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 für Phase 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 (2)

Phase II: Die Instruktionsebene

Lokale Prädikate (assoziiert mit Instruktionsknoten):

- $\text{Comp}_\iota(t)$: ι berechnet t.
- $\text{Transp}_\iota(t)$: ι modifiziert keinen Operanden von t.
- ν-BB-N-Avail, ν-BB-X-Avail: größte Lösungen des BB-Gleichungssystem von Phase I.

Das Instruktions-Gleichungssystem für Phase II:

$$\text{N-Avail}_\iota = \begin{cases} \nu\text{-BB-N-Avail}_{bb(\iota)} & \text{falls } \iota = start(bb(\iota)) \\ \text{X-Avail}_{pred(\iota)} & \text{sonst (da gilt: } |pred(\iota)| = 1) \end{cases}$$

$$\text{X-Avail}_\iota = \begin{cases} \nu\text{-BB-X-Avail}_{bb(\iota)} & \text{falls } \iota = end(bb(\iota)) \\ (\text{N-Avail}_\iota + \text{Comp}_\iota) \cdot \text{Transp}_\iota & \text{sonst} \end{cases}$$

Bezeichnung

...wobei *bb*, *start* und *end* drei Abbildungen sind, die angewendet auf eine Instruktion ι bzw. einen Basisblock β den Basisblock liefern, zu dem ι gehört, bzw. den Start- oder Endinstruktionsknoten von β.

B.3.2

Knotenbenannte Instruktionsgraphen

Verfügbare Ausdrücke

...für knotenbenannte Instruktionsgraphen.

Lokale Prädikate (assoziiert mit Instruktionsknoten):

- $\text{Comp}_\iota(t)$: ι berechnet t.
- $\text{Transp}_\iota(t)$: ι modifiziert keinen Operanden von t.

Gleichungssystem für knotenbenannte Instruktionsgraphen:

$$\text{N-Avail}_\iota = \begin{cases} \textbf{falsch} & \text{falls } \iota = s \\ \prod_{\hat{\iota} \in pred(\iota)} \text{X-Avail}_{\hat{\iota}} & \text{sonst} \end{cases}$$

$$\text{X-Avail}_\iota = (\text{N-Avail}_\iota + \text{Comp}_\iota) \cdot \text{Transp}_\iota$$

B.3.3

Kantenbenannte Instruktionsgraphen

Verfügbare Ausdrücke

...für kantenbenannte Instruktionsgraphen.

Lokale Prädikate (assoziiert mit Instruktionskanten):

- $\text{Comp}_\varepsilon(t)$: Anweisung ι von Kante ε berechnet t.
- $\text{Transp}_\varepsilon(t)$: Anweisung ι von Kante ε modifiziert keinen Operanden von t.

Gleichungssystem für kantenbenannte Instruktionsgraphen:

$$\text{Avail}_n = \begin{cases} \textbf{falsch} & \text{falls } n = s \\ \prod_{m \in pred(n)} (\text{Avail}_m + \text{Comp}_{(m,n)}) \cdot \text{Transp}_{(m,n)} & \text{sonst} \end{cases}$$

B.3.4

Zwischenfazit

Beobachtung

...kantenbenannte Instruktionsgraphen sind konzeptuell und formulierungstechnisch am

- günstigsten.

...knotenbenannte Basisblockgraphen am

- ungünstigsten.

In der Folge

...zwei weitere Beispiele dazu und zur Veranschaulichung des Einflusses von Flussgraphvarianten auf die konzeptuelle und technische Komplexität der Formulierung von Programmanalysen:

- Konstantenanalyse (engl. constant propagation and folding)
- Geistervariablen (engl. faint variables)

Dabei betrachten wir Analyseformulierungen für:

- knotenbenannte Basisblockgraphen
- kantenbenannte Instruktionsgraphen

als die beiden antagonistischen Pole der Graphvarianten.

Kapitel B.4

Konstantenanalyse

Konstantenanalyse

... am Beispiel einfacher Konstanten (engl. simple constants).

Wir benötigen zwei Hilfsfunktionen für Instruktionen:

- Rückwärtssubstitution
- Zustandstransformation

sowie deren Ausdehnungen auf Instruktionssequenzen, speziell Pfadinstruktionssequenzen.

B.4.1

Kantenbenannte Instruktionsgraphen

Rückwärtssubstitution, Zustandstransformation

...für Instruktionen $\iota \equiv (x := t')$.

Wir definieren für ι:

- Rückwärtssubstitution

$$\delta_\iota : \mathbf{T} \to \mathbf{T}$$
$$\forall\, t \in \mathbf{T}.\ \delta_\iota(t) =_{df} t[t'/x]$$

  wobei $t[t/'x]$ die simultane Ersetzung aller Vorkommen von x in t durch t' bezeichnet (syntaktische Substitution).

- Zustandstransformation

$$\theta_\iota : \Sigma \to \Sigma$$
$$\theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

  wobei $\mathcal{E} : \mathbf{T} \to \Sigma \to \mathbb{Z}$ die Auswertung von Termen entsprechend ihrer Semantik (z.B. $\mathcal{E} = [\![\ ]\!]_A$) leistet.

Der Zusammenhang von δ und θ

...ist beschrieben durch das Substitutionslemma B.4.1.1, wobei $\mathcal{I}$ die Menge aller Instruktionen bezeichne.

Lemma B.4.1.1 (Substitutionslemma)

$$\forall\, t \in \mathbf{T}\ \forall\, \sigma \in \Sigma\ \forall\, \iota \in \mathcal{I}.\ \mathcal{E}(\delta_\iota(t))(\sigma) = \mathcal{E}(t)(\theta_\iota(\sigma))$$

Beweis induktiv über den Aufbau von t.

Einfache Konstanten auf Instruktionsgraphen

Seien bzw. bezeichnen:

- $\mathsf{SC}_n \in \Sigma =_{df} \{\sigma \mid \sigma : \mathbf{V} \to \mathbb{Z}_{\perp}^{\top}\}$
- $\sigma_s \in \Sigma \backslash \{\sigma_{\top}\}$ Anfangszustand (oder Startzusicherung)

Das SC-Gleichungssystem für Instruktionsgraphen:

$$\mathsf{SC}_n = \begin{cases} \sigma_{\mathbf{s}} & \text{falls } n = s \\ \lambda\, v.\ \bigsqcap\{\, \mathcal{E}(\delta_{(m,n)}(v))(\mathsf{SC}_m) \mid m \in \mathit{pred}(n) \,\} & \text{sonst} \end{cases}$$

...wobei SC als Bezeichnung von simple constants abgeleitet ist.

B.4.2

Knotenbenannte Basisblockgraphen

Rückwärtssubstitution, Zustandstransformation

...auf Instruktionssequenzen von Pfaden, speziell damit auch auf Instruktionssequenzen von Basisblöcken.

Ausdehnung von δ und θ zur:

- Rückwärtssubstitution auf Pfadinstruktionssequenzen

$$\Delta_p : \mathbf{T} \to \mathbf{T}$$

$$\Delta_p =_{df} \begin{cases} \delta_{n_q} & \text{falls } q = 1 \\ \Delta_{(n_1,\ldots,n_{q-1})} \circ \delta_{n_q} & \text{falls } q > 1 \end{cases}$$

- Zustandstransformation auf Padinstruktionssequenzen

$$\Theta_p : \Sigma \to \Sigma$$

$$\Theta_p =_{df} \begin{cases} \theta_{n_1} & \text{falls } q = 1 \\ \Theta_{(n_2,\ldots,n_q)} \circ \theta_{n_1} & \text{falls } q > 1 \end{cases}$$

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 (1)

Phase I: Basisblockebene

Seien bzw. bezeichnen:

- $\Sigma =_{df} \{\sigma \mid \sigma : \mathbf{V} \to \mathbb{Z}_{\perp}^{\top}\}$
- $\Delta_{\beta}(v) =_{df} \delta_{\iota_1} \circ \ldots \circ \delta_{\iota_q}(v)$, wobei $\beta \equiv \iota_1; \ldots; \iota_q$.
- $\text{BB-N-SC}_{\beta}, \text{BB-X-SC}_{\beta}, \text{N-SC}_{\iota}, \text{X-CP}_{\iota} \in \Sigma$
- $\sigma_s \in \Sigma$ Anfangszustand (oder Startzusicherung)

Das BB-Gleichungssystem für einf. Konstanten von Phase I:

$$\text{BB-N-SC}_{\beta} = \begin{cases} \sigma_s & \text{falls } \beta = \mathbf{s} \\ \bigsqcap\{\text{BB-X-SC}_{\hat{\beta}} \mid \hat{\beta} \in \mathit{pred}(\beta)\} & \text{sonst} \end{cases}$$

$$\text{BB-X-SC}_{\beta} = \lambda\, v.\, \mathcal{E}(\Delta_{\beta}(v))(\text{BB-N-SC}_{\beta})$$

Einfache Konstanten auf BB-Graphen (2)

Phase II: Anweisungsebene

Vorberechnete Resultate (aus Phase I):

- ν-BB-N-SC, ν-BB-X-SC: die größten Lösung des Gleichungssystems von Phase I.

Das Inst.-Gleichungssystem für einf. Konstanten von Phase II:

$$\text{N-SC}_\iota = \begin{cases} \nu\text{-BB-N-SC}_{bb(\iota)} & \text{falls } \iota = start(bb(\iota)) \\ \text{X-SC}_{pred(\iota)} & \text{sonst (da gilt: } |pred(\iota)| = 1) \end{cases}$$

$$\text{X-SC}_\iota = \begin{cases} \nu\text{-BB-X-SC}_{bb(\iota)} & \text{falls } \iota = end(bb(\iota)) \\ \lambda v.\, \mathcal{E}(\delta_\iota(v))(\text{N-SC}_\iota) & \text{sonst} \end{cases}$$

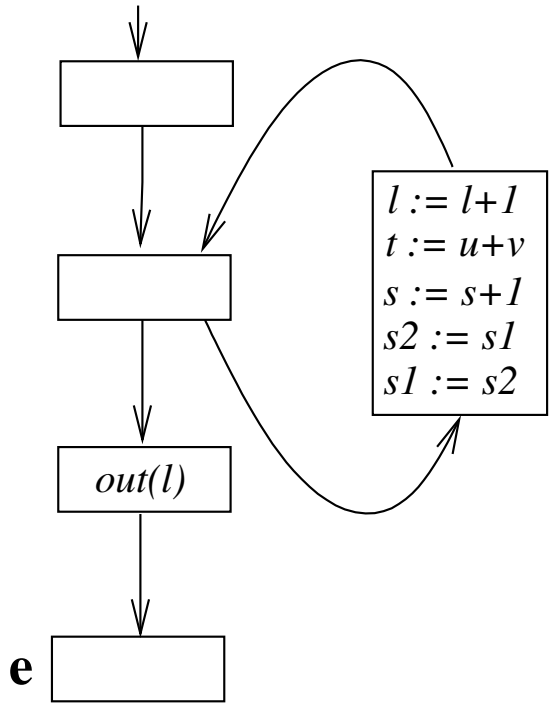
...wobei *bb*, *start* und *end* drei Abbildungen sind, die angewendet auf eine Instruktion ι bzw. einen Basisblock β den Basisblock liefern, zu dem ι gehört, bzw. den Start- oder Endinstruktionsknoten von β.

B.5

Geistervariablen

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 Instruktionsgraphen.

Lokale Prädikate (assoziiert mit Instruktionskanten):

- $\mathrm{LifeEnforcingUse}_{\varepsilon}^{v}$: Variable v kommt in der Anweisung ι von Kante ε vor und wird von ihr 'zu leben gezwungen' (z.B. wenn ι eine Ausgabeanweisung, Verzweigungsbedingung oder Schleifenabbruchbedingung ist).
- $\mathrm{Mod}_{\varepsilon}^{v}$: Anweisung ι von Kante ε modifiziert Variable v.
- $\mathrm{AssUse}_{\varepsilon}^{v}$: Variable v kommt rechtsseitig in der Zuweisung ι von Kante ε vor.
- $\mathit{LhsVar}_{\varepsilon}$: Bezeichnet die linksseitige Variable der Zuweisung ι an Kante ε.

Geistervariablenanalyse (2)

Das GV-Gleichungssystem für Instruktionsgraphen:

$$\begin{aligned} \mathrm{FAINT}_n^v \quad = \quad \prod_{m \in succ(n)} \quad & (\,\overline{\mathrm{LifeEnforcingUse}_{(n,m)}^v} \;*\\ & (\mathrm{FAINT}_m^v \;+\; \mathrm{Mod}_{(n,m)}^v) \;*\\ & (\mathrm{FAINT}_m^{LhsVar_{(n,m)}} + \overline{\mathrm{AssUse}_{(n,m)}^v})\,) \end{aligned}$$

Intuitiv: Eine Variable v ist geisterhaft am Knoten n, wenn v

- von keiner Instruktion 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.

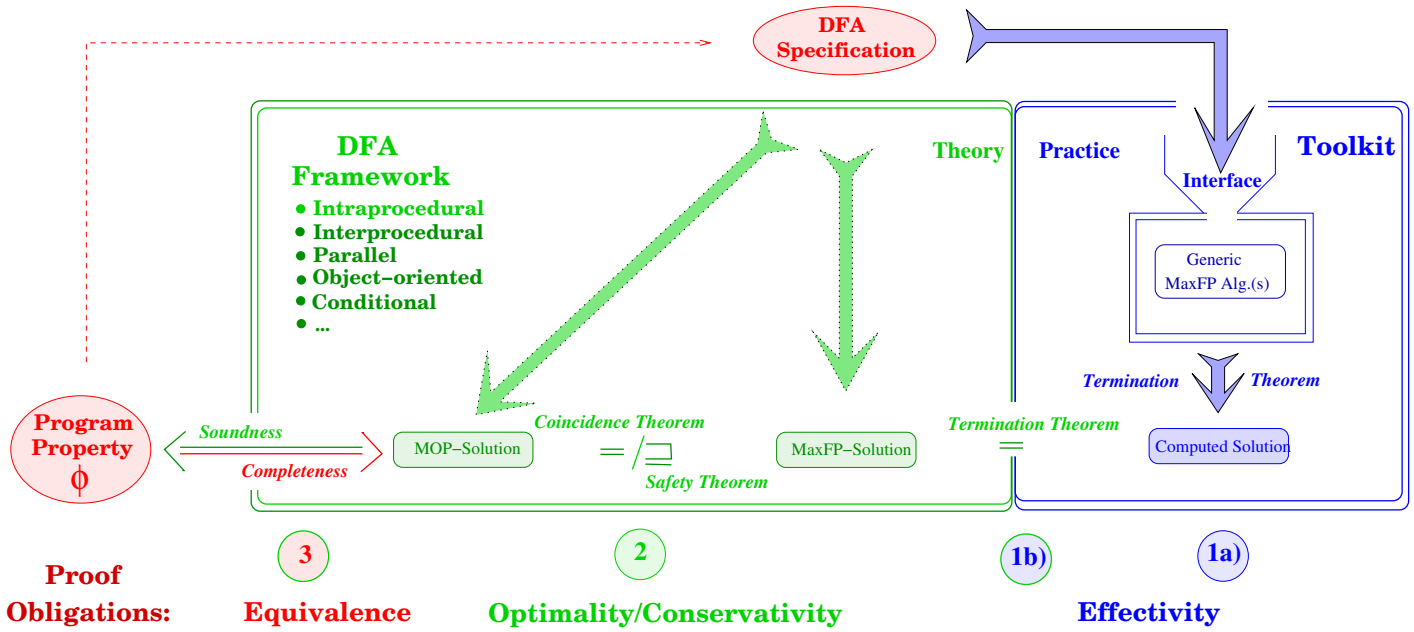
B.6

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

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.