Analyse und Verifikation

LVA 185.276, VU 2.0, ECTS 3.0
SS 2017

(Stand: 24.05.2017)

Jens Knoop

Technische Universität Wien
Institut für Computersprachen

Inhaltsverzeichnis

Inhaltsverzeichnis (1)

Teil I: Motivation

Inhaltsverzeichnis (2)

Inhaltsverzeichnis (3)

Inhaltsverzeichnis (4)

Inhaltsverzeichnis (5)

Inhaltsverzeichnis (6)

Inhaltsverzeichnis (7)

Inhaltsverzeichnis (8)

Inhaltsverzeichnis (9)

Teil V: Abschluss und Ausblick

Teil I

Motivation

Kapitel 1

Grundlagen

Grundlagen

Syntax und Semantik von Programmiersprachen:

- 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 beispielsweise)

Literaturhinweise (1)

Als Textbücher:

- 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 und vertiefend:

- Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. Verification of Sequential and Concurrent Programs. 3. Auflage, Springer-V., 2009.
- 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.
- Krzysztof R. Apt, Ernst-Rüdiger Olderog. Programmverifikation – Sequentielle, parallele und verteilte Programme. Springer-V., 1994.

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.

Unser Beispielszenario

- Modell(programmier)sprache *WHILE*
  - Syntax und Semantik
- Semantikdefinitionsstile
  1. Operationelle Semantik
     - 1.1 Großschritt-Semantik: Natürliche Semantik
     - 1.2 Kleinschritt-Semantik: Strukturell operationelle Semantik
  2. Denotationelle Semantik
  3. Axiomatische Semantik

Intuitiv

- 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 als Zusicherungen ausgedrückt.
  Bestimmte andere Aspekte der Ausführung werden dabei i.a. ignoriert.

Kapitel 1.1

Motivation

Motivation

...formale Semantik von Programmiersprachen einzuführen:

Die (mathematische) Rigorosität formaler Semantik

- erlaubt Mehrdeutigkeiten, Über- und Unterspezifikationen natürlichsprachlicher Beschreibungen aufzudecken und aufzulösen
- bietet die Grundlage für vertrauenswürdige Implementierungen der Programmiersprache, für die Analyse, Verifikation und Transformation von Programmen

Unsere Modellsprache

- Die (Programmier-) Sprache *WHILE*
  - Syntax
  - Semantik
- Semantikdefinitionsstile

  ...wofür sie besonders geeignet sind und wie sie in Beziehung zueinander stehen:
  - Operationelle Semantik ($\rightsquigarrow$ Sprachimplementierung)
    - Natürliche Semantik
    - Strukturell operationelle Semantik
  - Denotationelle Semantik ($\rightsquigarrow$ Sprachdesign)
  - Axiomatische Semantik ($\rightsquigarrow$ Anwend'gsprogrammierung)
    - Beweiskalküle für partielle und totale Korrektheit
    - Korrektheit, Vollständigkeit

Kapitel 1.2

Modellsprache *WHILE*

Modellsprache *WHILE*

WHILE, der sog. "while"-Kern imperativer Programmiersprachen, besitzt:

- Zuweisungen (einschließlich der leeren Anweisung)
- Fallunterscheidungen
- while-Schleifen
- Sequentielle Komposition

Bemerkung: *WHILE* ist "schlank", doch Turing-mächtig!

Syntax und Semantik von *WHILE* im Überblick

- Syntax: Programme der Form:

$$\begin{aligned} \pi \quad ::= \quad & x := a \mid \mathit{skip} \mid \\ & \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi} \mid \\ & \text{while } b \text{ do } \pi_1 \text{ od} \mid \\ & \pi_1 ; \pi_2 \end{aligned}$$

- Semantik: Zustandstransformationen der Form:

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

mit $\Sigma =_{df} \{\sigma \mid \sigma : \mathbf{Var} \to \mathrm{ID}\}$ Menge aller Zustände über der Variablenmenge **Var** und geeignetem Datenbereich ID.

(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".)

Syntax von Zahlwörtern und Ausdrücken

Zahlwörter (Numerale)

$$\begin{aligned} z &::= 0 \mid 1 \mid 2 \mid \ldots \mid 9 \\ n &::= z \mid n\,z \end{aligned}$$

Arithmetische Ausdrücke

$$a ::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \mid a_1/a_2 \mid \ldots$$

Wahrheitswertausdrücke (Boolesche Ausdrücke)

$$\begin{aligned} b ::= {} & \mathit{true} \mid \mathit{false} \mid \\ & a_1 = a_2 \mid a_1 \neq a_2 \mid a_1 < a_2 \mid a_1 \leq a_2 \mid \ldots \mid \\ & b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b_1 \end{aligned}$$

Vereinbarungen

In der Folge bezeichnen wir mit:

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

Ausblick

In der Folge betrachten wir im Detail:

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

...und deren Beziehungen zueinander, d.h. die Beziehungen zwischen

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

Kapitel 1.3

Semantik von Ausdrücken

Motivation

Die Semantik von *WHILE* stützt sich ab auf die Semantiken von

- Zahlwörtern
- arithmetischen Ausdrücken
- Wahrheitswertausdrücken

und den Begriff der

- Speicherzustände

Diese sind daher zunächst festzulegen.

Speicherzustände

Die Menge der (Speicher-) Zustände ist gegeben durch

- $\Sigma =_{df} \{ \sigma \mid \sigma : \mathbf{Var} \to \mathbb{Z} \}$

wobei

- $\mathbb{Z} =_{df} \{\ldots, \mathbf{-1}, \mathbf{-2}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$

die Menge der ganzen Zahlen bezeichnet.

Semantik von Zahlwörtern

$[\![.]\!]_N : \mathbf{Num} \to \mathbb{Z}$ ist induktiv definiert durch

- $[\![0]\!]_N =_{df} \mathbf{0}$, ..., $[\![9]\!]_N =_{df} \mathbf{9}$
- $[\![n\, i]\!]_N =_{df} \mathit{plus}(\mathit{mal}(\mathbf{10}, [\![n]\!]_A), [\![i]\!]_N)$, $i \in \{0, \ldots, 9\}$
- $[\![-\, n]\!]_N =_{df} \mathit{minus}([\![n]\!]_N)$

Bemerkungen:

- 0, 1, 2, ... bezeichnen syntaktische Entitäten, Darstellungen von Zahlen; $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, ... bezeichnen semantische Entitäten, hier ganze Zahlen: $\mathbb{Z} =_{df} \{..., \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$.
- $\mathit{plus}, \mathit{mal} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, $\mathit{minus} : \mathbb{Z} \to \mathbb{Z}$ bezeichnen semantische Operationen, hier die übliche Addition, Multiplikation und Vorzeichenwechsel auf $\mathbb{Z}$.
- Die Semantik von Zahlwörtern ist zustandsunabhängig.

Semantik arithmetischer und Wahrheitswert-ausdrücke

Semantik

- arithmetischer Ausdrücke: $[\![.]\!]_A : \mathbf{Aexpr} \to (\Sigma \hookrightarrow \mathbb{Z})$
- Wahrheitswertausdrücke: $[\![.]\!]_B : \mathbf{Bexpr} \to (\Sigma \hookrightarrow \mathbb{B})$

wobei

- $\mathbb{Z} =_{df} \{..., \mathbf{0}, \mathbf{1}, \mathbf{2}, ...\}$ die Menge ganzer Zahlen
- $\mathbb{B} =_{df} \{\mathbf{wahr}, \mathbf{falsch}\}$ die Menge der Wahrheitswerte

bezeichnen.

Semantik arithmetischer Ausdrücke

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

- $[\![n]\!]_A(\sigma) =_{df} [\![n]\!]_N$
- $[\![x]\!]_A(\sigma) =_{df} \sigma(x)$
- $[\![a_1 + a_2]\!]_A(\sigma) =_{df} \mathit{plus}([\![a_1]\!]_A(\sigma), [\![a_2]\!]_A(\sigma))$
- $[\![a_1 * a_2]\!]_A(\sigma) =_{df} \mathit{mal}([\![a_1]\!]_A(\sigma), [\![a_2]\!]_A(\sigma))$
- $[\![a_1 - a_2]\!]_A(\sigma) =_{df} \mathit{minus}([\![a_1]\!]_A(\sigma), [\![a_2]\!]_A(\sigma))$
- ... (andere Operatoren analog)

wobei

- plus, mal, $\mathit{minus} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ die übliche Addition, Multiplikation und Subtraktion auf $\mathbb{Z}$ bezeichnen.

Bemerkung:

- $+$, $*$, $-$ bezeichnen syntaktische Entitäten (kurz: Operatoren); *plus*, *mal*, *minus* bezeichnen semantische Entitäten (kurz: Operationen)

Semantik Boolescher Ausdrücke (1)

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

- $[\![\, \mathit{true} \,]\!]_B(\sigma) =_{df} \textbf{wahr}$
- $[\![\, \mathit{false} \,]\!]_B(\sigma) =_{df} \textbf{falsch}$
- $[\![\, a_1 = a_2 \,]\!]_B(\sigma) =_{df}$
  $\begin{cases} \textbf{wahr} & \text{falls } \mathit{gleich}([\![\, a_1 \,]\!]_A(\sigma), [\![\, a_2 \,]\!]_A(\sigma)) \\ \textbf{falsch} & \text{sonst} \end{cases}$
- ... (andere Relatoren (z.B. $<$, $\leq$, ...) analog)

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

Semantik Boolescher Ausdrücke (2)

Dabei bezeichnen

- **wahr** und **falsch** die Wahrheitswertkonstanten "wahr" und "falsch"
- *und*, *oder* : $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ und *neg* : $\mathbb{B} \to \mathbb{B}$ die übliche zweistellige logische Konjunktion und Disjunktion und einstellige Negation auf $\mathbb{B}$
- *gleich* : $\mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$ die übliche Gleichheitsrelation auf $\mathbb{Z}$

Semantik Boolescher Ausdrücke (3)

Bemerkung:

- Beachte auch hier den Unterschied zwischen den syntaktischen Entitäten *true* und *false* und ihren semantischen Gegenstücken **wahr** und **falsch**
- Beachte ebenso den Unterschied zwischen den
  - Relatoren $=$, $>$, $<$ (syntaktische Entitäten) und den Relationen *gleich*, *größer*, *kleiner* (semantische Entitäten)
  - Operatoren $\neg$, $\wedge$, $\vee$ (syntaktische Entitäten) und den Operationen *neg*, *und*, *oder* (semantische Entitäten)

Kapitel 1.4

Syntaktische und semantische Substitution

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

Lemma 1.4.1

Seien $a \in \mathbf{AExpr}$ 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 1.4.2

Seien $b \in \mathbf{BExpr}$ und $\sigma, \sigma' \in \Sigma$ mit $\sigma(x) = \sigma'(x)$ für alle $x \in FV(b)$. Dann gilt:

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

Syntaktische und semantische Substitution

Von zentraler Bedeutung:

- Der Substitutionsbegriff!

Substitution tritt in zwei Spielarten in Erscheinung:

- Syntaktische Substitution
- Semantische Substitution

Der Zusammenhang von syntaktischer und semantischer Substitution wird beschrieben durch:

- Das Substitutionslemma 1.4.5

Syntaktische Substitution

Definition 1.4.3 (Syntaktische Substitution)

Die syntaktische Substitution für arithmetische Terme ist eine dreistellige Abbildung

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

die induktiv definiert ist durch:

$$\begin{array}{rcl} n[t/x] & =_{df} & n \quad \text{für } n \in \mathbf{Num} \\ y[t/x] & =_{df} & \begin{cases} t & \text{falls } y = x \\ y & \text{sonst} \end{cases} \\ (t_1\ op\ t_2)[t/x] & =_{df} & (t_1[t/x]\ op\ t_2[t/x]) \text{ für } op \in \{+, *, -, \ldots\} \end{array}$$

Semantische Substitution

Definition 1.4.4 (Semantische Substitution)

Die semantische Substitution ist eine dreistellige Abbildung

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

die definiert ist durch:

$$\sigma[\mathbf{z}/x](y) =_{df} \begin{cases} \mathbf{z} & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

Substitutionslemma

Zusammenhang von syntaktischer und semantischer Substitution:

Lemma 1.4.5 (Substitutionslemma)

$$[\![e[t/x]]\!]_A(\sigma) = [\![e]\!]_A(\sigma[[\![t]\!]_A(\sigma)/x])$$

wobei

- $[t/x]$ die syntaktische Substitution und
- $[[\![t]\!]_A(\sigma)/x]$ die semantische Substitution

bezeichnen.

Analog gilt ein entsprechendes Substitutionslemma für $[\![\ ]\!]_B$.

Kapitel 1.5

Induktive Beweisprinzipien

Induktive Beweisprinzipien (1)

Grundlegend:

- Vollständige Induktion
- Verallgemeinerte Induktion
- Strukturelle Induktion

...zum Beweis einer Aussage (insbesondere der drei Lemmata in Kapitel 1.4).

Induktive Beweisprinzipien (2)

Sei A eine Aussage, S eine (induktiv definierte) Struktur und bezeichne Komp(s) die Menge der Komponenten von $s \in S$.

Die Prinzipien der

- vollständigen Induktion

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

- verallgemeinerten Induktion

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

- strukturellen Induktion

$$(\forall\, s \in S.\, \forall\, s' \in \mathit{Komp}(s).\, A(s')) \Rightarrow A(s)) \quad \Rightarrow \quad \forall\, s \in S.\, A(s)$$

Induktive Beweisprinzipien (3)

Bemerkungen:

- Vollständige, verallgemeinerte und strukturelle Induktion sind gleich mächtig.
- Abhängig vom Anwendungsfall ist oft eines der Induktionsprinzipien zweckmäßiger, d.h. einfacher anzuwenden.

  Zum Beweis von Aussagen über induktiv definierte Datenstrukturen ist i.a. das Prinzip der strukturellen Induktion am zweckmäßigsten.

Beispiel: Beweis von Lemma 1.4.1 (1)

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

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

Induktionsanfang:

Fall 1: Sei $a \equiv n$, $n \in \mathbf{Num}$.
Mit den Definitionen von $[\![\ ]\!]_A$ und $[\![\ ]\!]_N$ erhalten wir unmittelbar wie gewünscht:

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

Beispiel: Beweis von Lemma 1.4.1 (2)

Fall 2: Sei $a \equiv x$, $x \in \mathbf{Var}$.

Mit der Definition von $[\![\ ]\!]_A$ erhalten wir auch hier wie gewünscht:

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

Beispiel: Beweis von Lemma 1.4.1 (3)

Induktionsschluss:

Fall 3: Sei $a \equiv a_1 + a_2$, $a_1, a_2 \in$ **Aexpr**

Wir erhalten wie gewünscht:

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

Übrige Fälle: Analog. q.e.d.

Kapitel 1.6

Semantikdefinitionsstile

Motivation

Die Semantik einer Programmiersprache kann auf verschiedene Weise festgelegt werden. Wir sprechen hier 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
- axiomatische

Definitionsstil.

Intuition und Ausblick

- Sprach- und Anwendungsimplementierersicht
  - Operationelle Semantik
    - Strukturell operationelle Semantik (small-step semantics)
    - Natürliche Semantik (big-step semantics)
- Sprachentwicklersicht
  - Denotationelle Semantik
- (Anwendungs-) Programmierer- und Verifizierersicht
  - Axiomatische Semantik

Kapitel 1.7

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)

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 (small-step semantics)

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

Literaturhinweise

- 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, Denmark, 1981, reprinted 1991.

Strukturell operationelle Semantik (1)

Die strukturell operationelle Semantik (SO-Semantik) von *WHILE* (im Sinne von Gordon D. Plotkin)

- ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation

$$\Sigma \hookrightarrow \Sigma$$

zu.

Das zugehörige

- Zustandstransformationsfunktional $[\![\, . \,]\!]_{sos}$ der strukturell operationellen Semantik

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

definieren wir in der Folge im Detail.

Strukturell operationelle Semantik (2)

Intuitiv:

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

Zentral dafür:

- Der Begriff der Konfiguration!

Konfigurationen

Wir unterscheiden:

- Nichtterminale bzw. (Zwischen-) Konfigurationen γ der Form $\langle \pi, \sigma \rangle$:
  ...das (Rest-) Programm π ist auf den (Zwischen-) Zustand σ anzuwenden.
- Terminale bzw. finale Konfigurationen γ der Form σ:
  ...der Zustand σ ist der Resultatzustand nach Ende der (regulären) Berechnung.

Vereinbarung:

- $\Gamma =_{df} (\mathbf{Prg} \times \Sigma) \cup \Sigma$ bezeichne die Menge aller Konfigurationen, $\gamma \in \Gamma$.

SOS-Regeln von *WHILE* – Axiome (1)

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

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

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

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

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

SOS-Regeln von *WHILE* – Regeln (2)

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

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

Sprechweisen

Wir unterscheiden:

- Prämissenlose Regeln, sog. Axiome, der Form

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

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

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

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

Beobachtung

Im Fall der SO-Semantik von *WHILE* haben wir also:

- 5 Axiome
  ...für die leere Anweisung, Zuweisung, Fallunterscheidung und while-Schleife.
- 2 Regeln
  ...für die sequentielle Komposition.

Berechnungsschritt, Berechnungsfolge

- Ein Berechnungsschritt ist von der Form

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

- Eine 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 $i \in \{0, \ldots, k-1\}$,
  - eine unendliche Folge von Konfigurationen mit $\gamma_0 = \langle \pi, \sigma \rangle$ und $\gamma_i \Longrightarrow \gamma_{i+1}$ für alle $i \in \mathbb{N}$.

Eigenschaften von Berechnungsfolgen

Definition 2.1.1 (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, falls sie unendlich ist
- irregulär terminierend sonst (z.B. Division durch $\mathbf{0}$)

Beispiel 1(4)

Sei

- $\sigma \in \Sigma$ mit $\sigma(x) = \mathbf{3}$
- $\pi \in \mathbf{Prg}$ mit
  $\pi \equiv y := 1;\ \text{while } x <> 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$
  (Bemerkung: $\equiv$ steht für 'syntaktisch identisch')

Bestimme

- die von der Anfangskonfiguration 'π angesetzt auf σ'

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

  induzierte Berechnungsfolge.

Beispiel 2(4)

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

$$
\begin{array}{ll}
\mathbf{(} \hat{=} & \langle x := x - 1; \\
 & \ \text{while } x <> 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{3}/y])\rangle \ \mathbf{)}
\end{array}
$$

Beispiel 3(4)

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

Beispiel 4(4)

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

Detailbetrachtung zum Beispiel 1(3)

$$\begin{array}{l} \langle y := 1;\ \text{while } x <> 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma\rangle \\ ([\text{ass}_{sos}], \\ [\text{comp}^2_{sos}]) \quad \Longrightarrow \langle \text{while } x <> 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{1}/y]\rangle \end{array}$$

steht vereinfachend für:

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

Detailbetrachtung zum Beispiel 2(3)

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

steht vereinfachend für:

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

Detailbetrachtung zum Beispiel 3(3)

$$\begin{array}{rl} & \langle (y := y * x;\ x := x - 1);\\ & \ \text{while } x <> 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, \sigma[\mathbf{1}/y]\rangle \\ ([\mathrm{ass}_{sos}], & \\ [\mathrm{comp}^2_{sos}], & \\ [\mathrm{comp}^1_{sos}]) \quad \Longrightarrow & \quad \langle x := x - 1;\\ & \ \text{while } x <> 1 \text{ do } y := y * x;\ x := x - 1 \text{ od},\\ & \qquad (\sigma[\mathbf{1}/y])[\mathbf{3}/y]\rangle \end{array}$$

steht vereinfachend für:

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

Determinismus der SOS-Regeln

Lemma 2.1.2

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

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

Korollar 2.1.3

Die von den SOS-Regeln für eine Konfiguration induzierte Berechnungsfolge ist eindeutig bestimmt, d.h. deterministisch.

Salopper, wenn auch weniger präzise:

Die SO-Semantik von *WHILE* ist deterministisch!

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

Dank Korollar 2.1.3 ist es sinnvoll festzulegen:

Definition 2.1.4 (SO-Semantik)

Die strukturell operationelle Semantik von *WHILE* ist gegeben durch das Funktional

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

das definiert ist durch:

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

Bemerkung: $\Rrightarrow^*$ bezeichnet die reflexiv-transitive Hülle von $\Rrightarrow$.

Variante induktiver Beweisführung

Induktion über die Länge von Berechnungsfolgen:

- Induktionsanfang
  - Beweise, dass A für Berechnungsfolgen der Länge 0 gilt.
- Induktionsschritt
  - Beweise unter der Annahme, dass A für Berechnungsfolgen der Länge kleiner oder gleich k gilt (Induktionshypothese!), dass A auch für Berechnungsfolgen der Länge $k+1$ gilt.

Anwendung

Induktive Beweisführung über die Länge von Berechnungsfolgen ist typisch zum Nachweis von Aussagen über Eigenschaften strukturell operationeller Semantik.

Ein Beispiel hierfür ist der Beweis des folgenden Lemmas:

Lemma 2.1.5

$$\forall \pi, \pi' \in \mathbf{Prg},\ \sigma, \sigma'' \in \Sigma,\ k \in \mathbb{N}.\ \left(\langle \pi_1; \pi_2, \sigma\rangle \Rightarrow^k \sigma''\right) \Rightarrow$$

$$\exists\, \sigma' \in \Sigma,\ k_1, k_2 \in \mathbb{N}.\ \left(k_1 + k_2 = k \wedge \langle \pi_1, \sigma\rangle \Rightarrow^{k_1} \sigma' \wedge \langle \pi_2, \sigma'\rangle \Rightarrow^{k_2} \sigma''\right)$$

Kapitel 2.2

Natürliche Semantik

Natürliche Semantik (big-step semantics)

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

Natürliche Semantik (1)

Die natürliche Semantik (N-Semantik) von *WHILE*

- ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation

$$\Sigma \hookrightarrow \Sigma$$

zu.

Das zugehörige

- Zustandstransformationsfunktional $[\![\, . \,]\!]_{ns}$ der natürlichen Semantik

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

definieren wir in der Folge im Detail.

Natürliche Semantik (2)

Intuitiv:

- Die N-Semantik ist am Zusammenhang zwischen initialem und finalem Speicherzustand einer Berechnung eines Programms $\pi \in \mathbf{Prg}$ interessiert.

Zentral auch hier:

- Der von der SO-Semantik bekannte Begriff der Konfiguration.

NS-Regeln von *WHILE* (1)

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

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

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

NS-Regeln von WHILE (2)

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

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

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

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

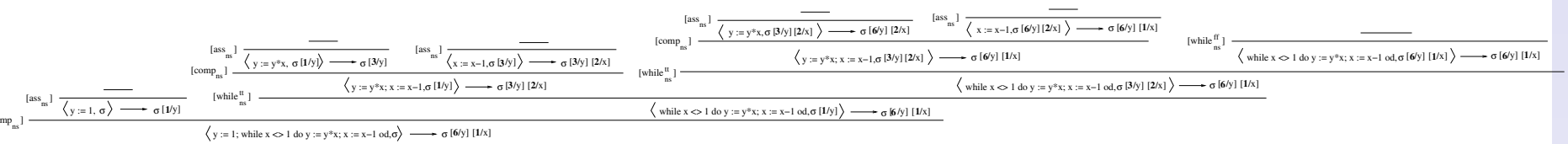
Beispiel 1(2)

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

Dann gilt:

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

$$\longrightarrow \sigma[\mathbf{6}/y][\mathbf{3}/x]$$



Beispiel 2(2)

Das gleiche Beispiel in “etwas” gefälligerer Darstellung:

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

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

Determinismus der NS-Regeln

Lemma 2.2.1

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

Korollar 2.2.2

Die von den NS-Regeln für eine Konfiguration induzierte finale Konfiguration ist (sofern definiert) eindeutig bestimmt, d.h. deterministisch.

Salopper, wenn auch weniger präzise:

Die N-Semantik von *WHILE* ist deterministisch!

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

Dank Korollar 2.2.2 ist es sinnvoll festzulegen:

Definition 2.2.3 (N-Semantik)

Die natürliche Semantik von *WHILE* ist gegeben durch das Funktional

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

das definiert ist durch:

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

Variante induktiver Beweisführung

Induktion über die Form von Ableitungsbäumen:

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

Anwendung

Induktive Beweisführung über die Form von Ableitungsbäumen ist typisch zum Nachweis von Aussagen über Eigenschaften natürlicher Semantik.

Ein Beispiel hierfür ist der Beweis von Lemma 2.2.1!

Kapitel 2.3

Strukturell operationelle und natürliche Semantik im Vergleich

Strukturell operationelle Semantik

Der Fokus liegt auf

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

  Intuitive Bedeutung der Transitionsrelation $\langle \pi, \sigma \rangle \equiv\!> \gamma$ mit γ von der Form $\langle \pi', \sigma' \rangle$ oder σ': Die Transition beschreibt den ersten Schritt d. Berechnungsf. v. 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.
  - γ ist von der Form σ':
    Die Abarbeitung von π ist vollständig; π angesetzt auf σ terminiert in einem Schritt in σ' regulär.

Natürliche Semantik

Der Fokus liegt auf

- Zusammenhang von initialem und finalem Zustand einer Berechnungsfolge

  Intuitive Bedeutung von

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

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

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

Lemma 2.3.1

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

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

Lemma 2.3.2

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

Beweis durch Induktion über die Länge der Ableitungsfolge $\langle \pi, \sigma \rangle \equiv\!\!> ^{k} \sigma'$, d.h. durch Induktion über k.

Äquivalenz von SO- und N-Semantik

Aus Lemma 2.3.1 und Lemma 2.3.2 folgt:

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, Denmark, 1981, reprinted 1991.


Gordon D. Plotkin. *An Operational Semantics for CSP*. In Proceedings of TC-2 Working Conference on Formal Description of Programming Concepts II, Dines Bjørner (Ed.), 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 von *WHILE*

...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.*

Denotationelle Semantik

Die denotationelle Semantik (D-Semantik) von *WHILE*

- ordnet jedem Programm π als Bedeutung eine partiell definierte Zustandstransformation

$$\Sigma \hookrightarrow \Sigma$$

zu.

Das zugehörige

- Zustandstransformationsfunktional $[\![\, . \,]\!]_{ds}$ der denotationellen Semantik

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

definieren wir in der Folge im Detail.

Vergleich operationeller und denotationeller Semantik

- Operationelle Semantik
  ...der Fokus liegt darauf, wie ein Programm ausgeführt wird.
- Denotationelle Semantik
  ...der Fokus liegt auf dem Effekt, den die Ausführung eines Programms hat: Für jedes syntaktische Konstrukt gibt es eine semantische Funktion, die ersterem ein mathematisches Objekt zuweist, d.h. eine Funktion, die den Effekt der Ausführung des Konstrukts beschreibt (jedoch nicht, wie dieser Effekt erreicht wird).

Kompositionalität

Zentral für denotationelle Semantiken: Kompositionalität!

Intuitiv:

- Für jedes Element der elementaren syntaktischen Konstrukte/Kategorien gibt es eine zugehörige semantische Funktion.
- Für jedes Element eines zusammengesetzten syntaktischen Konstrukts/Kategorie gibt es eine semantische Funktion, die über die semantischen Funktionen der Komponenten des zusammengesetzten Konstrukts definiert ist.

Kapitel 3.1

Denotationelle Semantik

Def. Gleichungen der denotationellen Semantik

$$[\![\mathit{skip}]\!]_{ds} = \mathit{Id}$$

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

$$[\![\pi_1;\ \pi_2]\!]_{ds} = [\![\pi_2]\!]_{ds} \circ [\![\pi_1]\!]_{ds}$$

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

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

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

Dabei bezeichnen *FIX* das sog. Zustandstransformationsfunktional und $\mathit{Id} : \Sigma \to \Sigma$ die identische Zustandstransformation definiert durch: $\forall \sigma \in \Sigma.\ \mathit{Id}(\sigma) =_{df} \sigma$

Das Hilfsfunktional cond: Fallunterscheidung

Funktionalität:

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

definiert durch

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

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

- 1. Argument: Prädikat (in unserem Kontext partiell definiert; siehe Kapitel 1.3)
- 2. und 3. Argument: Je eine partiell definierte Zustandstransformation
- Resultat: Ebenfalls eine partiell definierte Zustandstransformation

Daraus ergibt sich

...für die Bedeutung der Fallunterscheidung:

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

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

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

Erinnerung:

- In unserem Szenario sind $[\![.]\!]_A$ und $[\![.]\!]_B$ partiell definiert (z.B. Division durch **0**).

Das Hilfsfunktional FIX

Funktionalität:

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

Im Zshg. mit $[\![\ ]\!]_{ds}$ wird FIX angewendet auf das Funktional

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

das definiert ist durch:

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

mit b und π aus dem Kontext

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

Beachte:

- FIX ist ein Funktional, das sog. Zustandstransformationsfunktional.

Zustandstransformationsfunktional FIX

1. Offenbar müssen if b then $(\pi;$ while b do π od) else skip fi und while b do π od denselben Effekt haben.
2. Somit gilt: $[\![\,\text{while } b \text{ do } \pi \text{ od}\,]\!]_{ds} =$

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

3. Anwenden von F auf $[\![\,\text{while } b \text{ do } \pi \text{ od}\,]\!]_{ds}$ liefert gleichfalls: $F\ [\![\,\text{while } b \text{ do } \pi \text{ od}\,]\!]_{ds} =_{df}$

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

4. Aus 2.) und 3.) folgt damit:

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

   Das heißt: Die denotationelle Semantik der while-Schleife ist ein Fixpunkt des Funktionals F.

In der Folge werden wir sehen: Der kleinste Fixpunkt!

‘Vernünftigkeit’ der definierenden Gleichungen der denotationellen Semantik

Lemma 3.1.1

Für alle $\pi \in \mathbf{Prg}$ ist durch die “Definierenden Gleichungen der denotationellen Semantik” von der gleichnamigen Folie in eindeutiger Weise eine (partielle) Funktion definiert.

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

Dank Lemma 3.1.1 ist es sinnvoll festzulegen:

Definition 3.1.2 (D-Semantik)

Die denotationelle Semantik von *WHILE* ist gegeben durch das Funktional

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

wobei für alle $\pi \in \mathbf{Prg}$ die Funktion $[\![\ \pi\ ]\!]_{ds}$ diejenige gemäß Lemma 3.1.1 gegebene (partielle) Funktion ist.

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

Lemma 3.1.2

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

Beweis durch strukturelle Induktion über den induktiven Aufbau von π.

Lemma 3.1.3

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

Beweis Zeige, dass für alle $\sigma, \sigma' \in \Sigma$ gilt:

$$\langle \pi, \sigma \rangle \Rrightarrow^* \sigma' \Rightarrow [\![\, \pi \,]\!]_{ds}(\sigma) = \sigma'.$$

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

Korollar 3.1.4 (Äquivalenz von $[\![\ ]\!]_{ds}$ und $[\![\ ]\!]_{sos}$)

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

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

Aus Theorem 3.1.4 und Theorem 2.3.3 folgt:

Theorem 3.1.5

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

Fazit

Die Äquivalenz der strukturell operationellen, natürlichen und denotationellen Semantik von *WHILE* legt es nahe, den semantikangebenden Index in der Folge fortzulassen und vereinfachend von $[\![\ ]\!]$ als **der** Semantik der Sprache *WHILE* zu sprechen:

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

definiert durch (z.B.)

$$[\![\ ]\!] =_{df} [\![\ ]\!]_{sos}$$

Kapitel 3.2

Fixpunktfunktional

Zur Bedeutung von FIX F

In der Folge wollen wir die Bedeutung von

- *FIX F*

genauer untersuchen und aufklären.

Zur Bedeutung von FIX F

Erinnerung:

Hilfsfunktional *FIX* mit Funktionalität:

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

Im Zshg. mit $[\![\ ]\!]_{ds}$ wird *FIX* angewendet auf das Funktional

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

das definiert ist durch:

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

mit b und π aus dem Kontext

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

Wir wollen zeigen:

- Die denotationelle Semantik der while-Schleife ist der kleinste Fixpunkt des Funktionals F.

Dazu folgende Erinnerung

Aus der Beobachtung, dass

- while b do π od dieselbe Bedeutung haben muss wie if b then (π; while b do π od) else *skip* fi

folgt die Gleichheit von:

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

Zusammen mit der Definition von F folgt damit:

- $[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds}$ muss Fixpunkt des Funktionals F sein, das definiert ist durch

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

Das heißt, es muss gelten:

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

Dies führt uns zu einer kompositionellen Definition von $[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds}$ und damit insgesamt von $[\![\ \ ]\!]_{ds}$.

Unser Arbeitsplan

Wir benötigen:

- Einige Resultate aus der Fixpunkttheorie.

Diese erlauben uns:

- Nachzuweisen, dass diese Resultate auf unsere Situation anwendbar sind.

Dabei wird vorausgesetzt:

- Mathematische Grundlagen über Ordnungen, CPOs, Stetigkeit von Funktionen und benötigte Resultate darüber, insbesondere das Fixpunkttheorem.

Diese Grundlagen sind im Anhang zusammengefasst.

Folgende drei Eigenschaften

...werdend entscheidend sein:

1. $[\Sigma \hookrightarrow \Sigma]$ kann vollständig partiell geordnet werden.
2. F ist im Anwendungskontext stetig.
3. Fixpunktbildung wird im Anwendungskontext ausschließlich auf stetige Funktionen angewendet.

Insgesamt folgt aus diesen Eigenschaften die Wohldefiniertheit von

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

Ordnung auf Zustandstransformationen

Bezeichne

- $[\Sigma \hookrightarrow \Sigma]$ die Menge der partiell definierten Zustandstransformationen.

Wir definieren folgende Ordnung(srelation) auf $[\Sigma \hookrightarrow \Sigma]$:

$$g_1 \sqsubseteq g_2 \iff \forall\, \sigma \in \Sigma.\ g_1\ \sigma\ \mathit{definiert} = \sigma' \Rightarrow g_2\ \sigma\ \mathit{definiert} = \sigma'$$

mit $g_1, g_2 \in [\Sigma \hookrightarrow \Sigma]$

Lemma 3.2.1

1. $([\Sigma \hookrightarrow \Sigma], \sqsubseteq)$ ist eine partielle Ordnung.
2. Die *total undefinierte* (d.h. nirgends definierte) Funktion $\bot : \Sigma \hookrightarrow \Sigma$ mit $\bot\ \sigma = \mathit{undef}$ für alle $\sigma \in \Sigma$ ist *kleinstes* Element in $([\Sigma \hookrightarrow \Sigma], \sqsubseteq)$.

Ordnung auf Zustandstransformationen

Es gilt sogar:

Lemma 3.2.2

Das Paar $([\Sigma \hookrightarrow \Sigma], \sqsubseteq)$ ist eine vollständige partielle Ordnung (CPO) mit kleinstem Element $\bot$.

Weiters gilt:

Die kleinste obere Schranke $\bigsqcup Y$ einer Kette Y ist gegeben durch

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

Das heißt: $(\bigsqcup Y)\,\sigma = \sigma' \iff \exists\, g \in Y.\ g\ \sigma = \sigma'$

Graph einer totalen Funktion

Definition 3.2.3 (Graph einer totalen Funktion)

Der Graph einer totalen Funktion $f : M \to N$ ist definiert durch

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

Für totale Funktionen gilt:

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

Graph einer partiellen Funktion

Definition 3.2.4 (Graph einer partiellen Funktion)

Der Graph einer partiellen Funktion $f : M \hookrightarrow N$ mit Definitionsbereich $M_f \subseteq M$ ist definiert durch

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

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 $m \notin M_f$

Stetigkeitsresultate (1)

Lemma 3.2.5

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

$$F\ g = cond(p, g, g_0)$$

Dann gilt: F ist stetig.

Vorgriff auf Anhang A:
Seien $(C, \sqsubseteq_C)$ und $(D, \sqsubseteq_D)$ zwei CPOs und sei $f : C \to D$ 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. f monoton und
  $\forall\, C' \subseteq C.\ f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$
  (Erhalt der kleinsten oberen Schranken)

Stetigkeitsresultate (2)

Lemma 3.2.6

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

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

Dann gilt: F ist stetig.

Insgesamt

Zusammen mit

Lemma 3.2.7

Die Gleichungen zur Festlegung der denotationellen Semantik von *WHILE* definieren eine totale Funktion

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

sind wir durch!

Wir können beweisen:

Theorem 3.2.8 (Wohldefiniertheit von $[\![\ ]\!]_{ds}$)

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

ist wohldefiniert!

Somit wie anfangs angedeutet

Aus

1. Die Menge $[\Sigma \hookrightarrow \Sigma]$ der partiell definierten Zustandstransformationen bildet zusammen mit der Ordnung $\sqsubseteq$ eine CPO.
2. Funktional F mit "$F\ g = \mathit{cond}(p, g, g_0)$" und "$g \circ g_0$" sind stetig
3. In der Definition von $[\![\ ]\!]_{ds}$ wird die Fixpunktbildung ausschließlich auf stetige Funktionen angewendet.

...ergibt sich wie gewünscht die Wohldefiniertheit von:

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

Kapitel 3.3

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.

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.

Axiomatische Semantik

Wir betrachten insbesondere: Korrektheit und Vollständigkeit der axiomatischen Semantik

Erinnerung:

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

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

- Gültigkeit einer Korrektheitsformel im Sinne
  - partieller Korrektheit
  - totaler Korrektheit

Zwei wegbereitende klassische Arbeiten

- 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 4.1

Partielle und totale Korrektheit

Partielle Korrektheit

Sei $\pi \in$ **Prg** ein *WHILE*-Programm und seien p und q zwei logische Formeln:

Definition 4.1.1 (Partielle Korrektheit)

Eine Hoaresche Zusicherung $\{p\}\ \pi\ \{q\}$ heißt

- gültig (im Sinne der partiellen Korrektheit) oder kurz (partiell) korrekt (in Zeichen: $\models_{pk} \{p\}\ \pi\ \{q\}$) gdw für jeden Anfangszustand σ 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 auch die Nachbedingung q in σ' erfüllt.

Totale Korrektheit

Sei $\pi \in$ **Prg** ein *WHILE*-Programm und seien p und q zwei logische Formeln:

Definition 4.1.2 (Totale Korrektheit)

Eine Hoaresche Zusicherung $[p]\ \pi\ [q]$ heißt

- gültig (im Sinne der totalen Korrektheit) oder kurz (total) korrekt (in Zeichen: $\models_{tk} [p]\ \pi\ [q]$) gdw für jeden Anfangszustand σ 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.

Informell

"Totale Korrektheit = Partielle Korrektheit + Terminierung"

Eigenschaften von Korrektheitsformeln (1)

Definition 4.1.3 (Charakterisierung)

Die Zustandsmenge

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

heißt Charakterisierung von $p \in \textbf{Bexp}$.

Eigenschaften von Korrektheitsformeln (2)

Lemma 4.1.4 (Eigenschaften von Korrektheitsformeln)

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

- partiell korrekt (in Zeichen: $\models_{pk} \{p\}\ \pi\ \{q\}$), falls $[\![\ \pi\ ]\!](Ch(p)) \subseteq Ch(q)$
- total korrekt (in Zeichen: $\models_{tk} [p]\ \pi\ [q]$), falls $\{p\}\ \pi\ \{q\}$ partiell korrekt ist und $Def([\![\ \pi\ ]\!]) \supseteq Ch(p)$ gilt.

  Dabei bezeichnet $Def([\![\ \pi\ ]\!])$ die Menge aller Zustände, für die π regulär terminiert.

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

Erinnerung

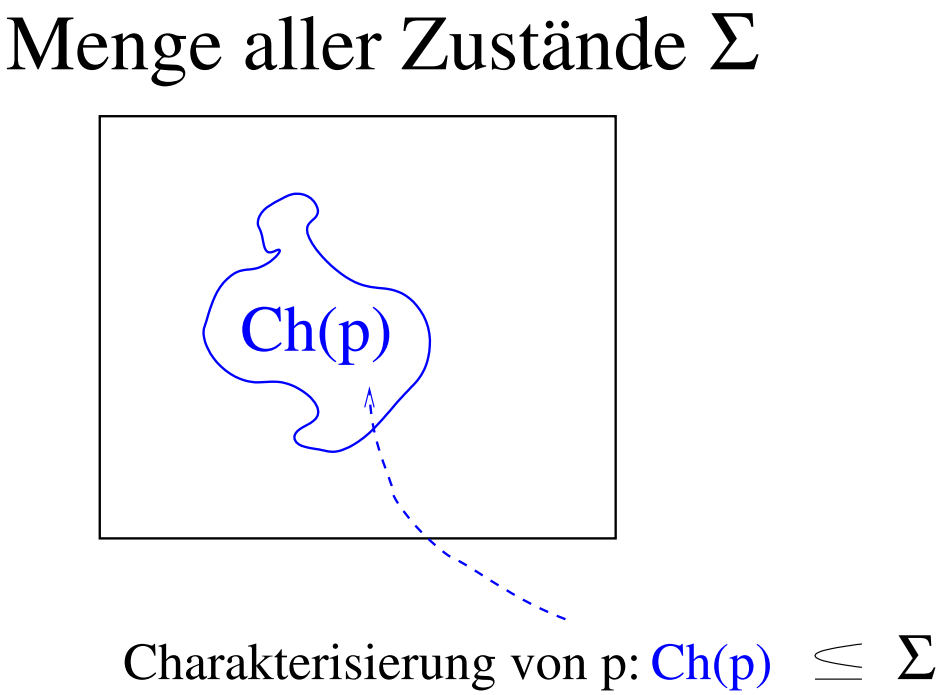
...an einige Sprechweisen:

Ein (deterministisches) Programm π

- angesetzt auf einen Anfangszustand σ terminiert regulär gdw. π nach endlich vielen Schritten in einem Zustand $\sigma' \in \Sigma$ endet.
- angesetzt auf einen Anfangszustand σ terminiert irregulär gdw. π nach endlich vielen Schritten zu einer Konfiguration $\langle \pi', \sigma' \rangle$ führt, für die es keine Folgekonfiguration gibt (z.B. wg. Division durch $\mathbf{0}$).
- Ein Programm π heißt divergent gdw. π terminiert für keinen Anfangszustand (weder regulär noch irregulär).

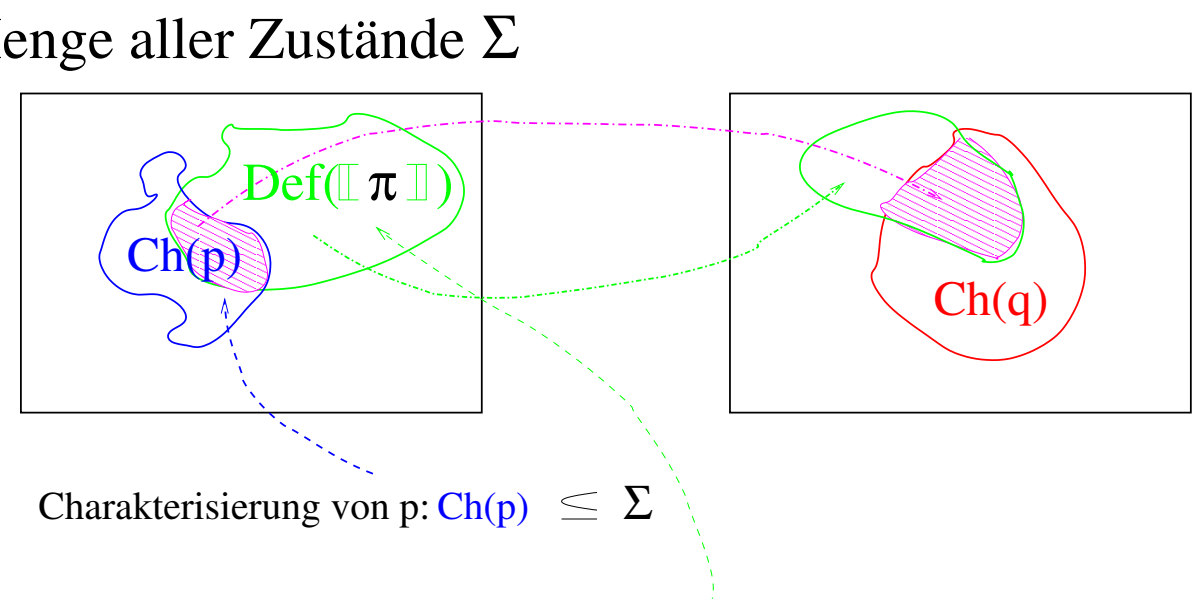
Veranschaulichung (1)

...der Charakterisierung $Ch(p)$ einer logischen Formel p:



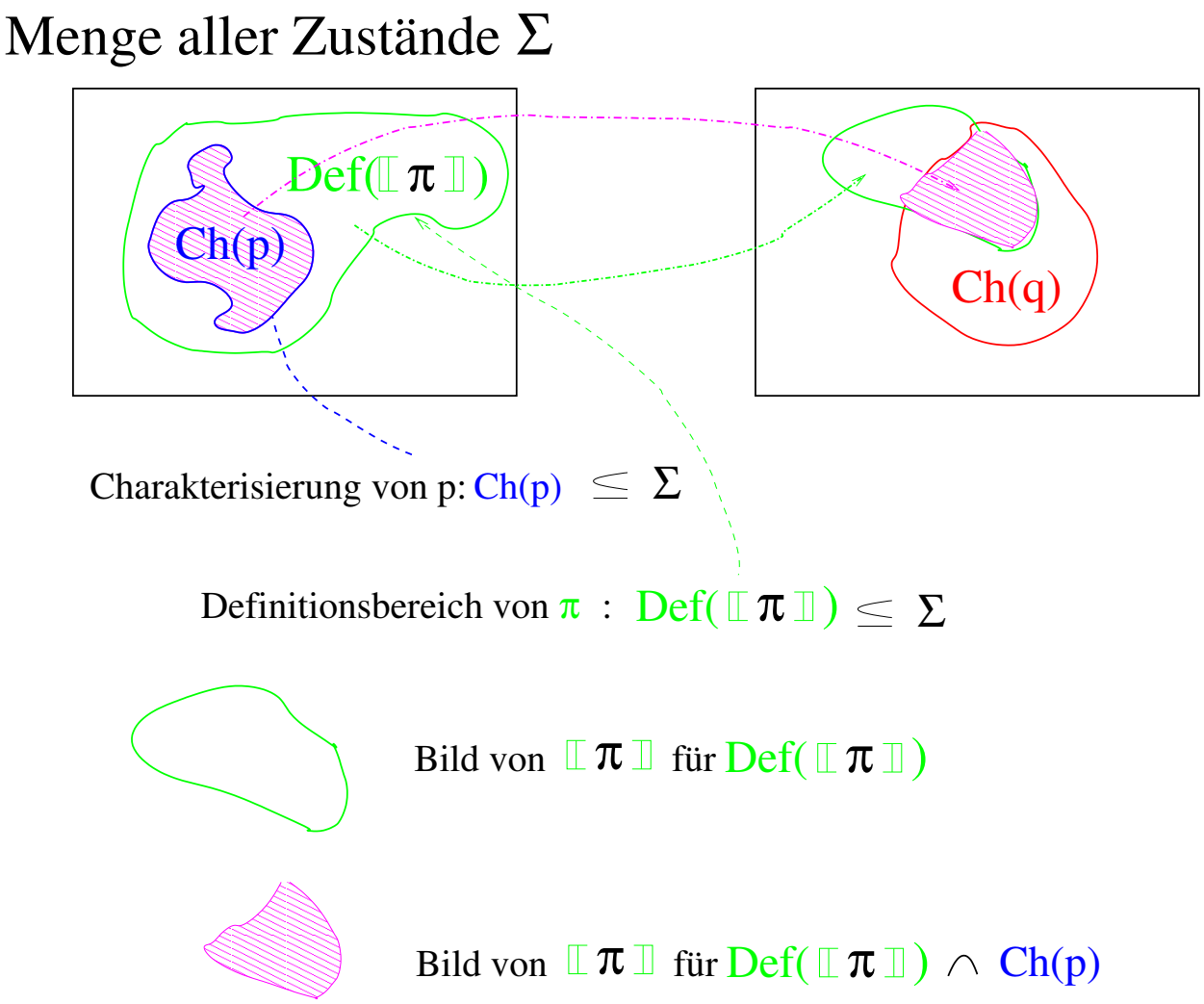
Veranschaulichung (2)

...der Gültigkeit eine Hoareschen Zusicherung $\{p\}\ \pi\ \{q\}$ im Sinne partieller Korrektheit:



Veranschaulichung (3)

...der Gültigkeit eine Hoareschen Zusicherung $[p]\ \pi\ [q]$ im Sinne totaler Korrektheit:



Stärkste Nachbedingungen, schwächste Vorbedingungen

In der Folge:

Präzisierung der Begriffe

- Stärkste Nachbedingung
- Schwächste (liberale) Vorbedingung

Stärkste Nach- und schwächste Vorbedingungen (1)

In der Situation der vorigen Abbildungen gilt:

- $[\![\pi]\!](Ch(p))$ heißt stärkste Nachbedingung von π bezüglich p.
- $[\![\pi]\!]^{-1}(Ch(q))$ heißt schwächste Vorbedingung von π bezüglich q, wobei $[\![\pi]\!]^{-1}(\Sigma') =_{df} \{\sigma \in \Sigma \mid [\![\pi]\!](\sigma) \in \Sigma'\}$
- $[\![\pi]\!]^{-1}(Ch(q)) \cup C(Def([\![\pi]\!]))$ heißt schwächste liberale Vorbedingung von π bezüglich q, wobei C den Mengenkomplementoperator (bzgl. der Grundmenge Σ) bezeichnet.

Stärkste Nach- und schwächste Vorbedingungen (2)

Lemma 4.1.5

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

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

Beweis: Übungsaufgabe

Zusammenhang von partieller und totaler Korrektheit

Lemma 4.1.6

Für deterministische Programme π gilt:

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

d.h. für deterministische Programme impliziert totale Korrektheit bzgl. eines Paars aus Vor- und Nachbedingung auch partielle Korrektheit bzgl. dieses Paars aus Vor- und Nachbedingung.

Schwächste Vor- & stärkste Nachbedingungen

...aus logischer Perspektive:

Definition 4.1.7 (schwächer als)

Seien A, B, A_1, A_2, ... (logische) Formeln

- A heißt schwächer als B, wenn gilt: $B \Rightarrow A$
- A_i heißt schwächste Formel in $\{A_1, A_2, \ldots\}$, wenn gilt: $A_j \Rightarrow A_i$ für alle j.

Defintion 4.1.8 (stärker als)

Seien A, B, A_1, A_2, ... (logische) Formeln

- A heißt stärker als B, wenn gilt: B ist schwächer als A, d.h. wenn gilt: $A \Rightarrow B$
- A_i heißt stärkste Formel in $\{A_1, A_2, \ldots\}$, wenn gilt: $A_i \Rightarrow A_j$ für alle j.

Schwächste Vorbedingungen

Definition 4.1.9 (Schwächste Vorbedingung)

Sei π ein Programm und q eine Formel. Dann heißt

- $wp(\pi, q)$ schwächste Vorbedingung für totale Korrektheit von π bezüglich (der Nachbedingung) q, wenn

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

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

Schwächste liberale Vorbedingungen

Definition 4.1.10 (Schwächste liberale Vorbeding'g)

Sei π ein Programm und q eine Formel. Dann heißt

- $wlp(\pi, q)$ schwächste liberale Vorbedingung für partielle Korrektheit von π bezüglich (der Nachbedingung) q, wenn

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

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

Zusammenfassung von Sprechweisen (1)

Hoaresche Zusicherungen sind von einer der zwei Formen

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

wobei

- p, q logische Formeln sind (meist prädikatenlogische Formeln 1. Stufe)
- π ein Programm ist
- p und q Vor- bzw. Nachbedingung heißen.

Zusammenfassung von Sprechweisen (2)

In einer Hoareschen Zusicherung werden üblicherweise

- geschweifte Klammern wie in
  $$\{p\}\ \pi\ \{q\}$$
  für Tripel im Sinne partieller Korrektheit und
- eckige Klammern wie in
  $$[p]\ \pi\ [q]$$
  für Tripel im Sinne totaler Korrektheit

benutzt.

Zwei Beispiele Hoarescher Zusicherungen

$$\{a > 0\}$$

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

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

...zum Ausdruck partieller Korrektheit von π bzgl. der Vorbedingung $a > 0$ und der Nachbedingung $y = a!$

$$[a > 0]$$

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

$$[y = a!]$$

...zum Ausdruck totaler Korrektheit von π bzgl. der Vorbedingung $a > 0$ und der Nachbedingung $y = a!$

Kapitel 4.2

Beweiskalkül für partielle Korrektheit

Hoare-Kalkül HK_{PK} für partielle Korrektheit

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

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

[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] $\dfrac{\{I \wedge b\}\ \pi\ \{I\}}{\{I\}\ \text{while } b \text{ do } \pi \text{ od}\ \{I \wedge \neg b\}}$

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

Diskussion von Vorwärtszuweisungsregel(n)

- Eine Vorwärtsregel für die Zuweisung wie

$$[\text{ass}_{fwd}] \quad \frac{}{\{p\}\ x{:=}t\ \{\exists z.\ p[z/x] \wedge\ x{=}t[z/x]\}}$$

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

- Beachte: Folgende scheinbar naheliegende quantorfreie Realisierung der Vorwärtszuweisungsregel ist nicht korrekt:

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

Beweis: Übungsaufgabe

Kapitel 4.3

Beweiskalkül für totale Korrektheit

Hoare-Kalküle: Partielle/totale Korrektheit (1)

Für die Sprache *WHILE* sind

- die Hoare-Kalküle für partielle und totale Korrektheit i.w. identisch.

Sie unterscheiden sich

- einzig in der Regel zur Behandlung der while-Schleife.

Hoare-Kalküle: Partielle/totale Korrektheit (2)

Im Kern muss die Regel [while] von HK_{PK} ersetzt werden durch eine

- terminationsordnungssensitive Regel.

Hierfür gibt es unterschiedliche Möglichkeiten, wobei eine Abwägung zu treffen ist zwischen

- Einfachheit der Regel
- Einfachheit der Regelanwendung.

In der Folge werden wir

- zwei Varianten mit unterschiedlicher Abwägung vorstellen.

Hoare-Kalkül HK_{TK} für totale Korrektheit

Variante 1: Regeleinfachheit vor Regelanwendungseinfachheit

...ersetze die Regel [while] aus HK_{PK} durch:

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

wobei

- t arithmetischer Term über ganzen Zahlen,
- w ganzzahlige Variable, die in I, b, π und t nicht frei vorkommt.

$\rightsquigarrow (\mathbb{N}, <)$ als Terminationsordnung!

Hoare-Kalkül HK_{TK} für totale Korrektheit

Variante 2: Regelanwendungseinfachheit vor Regeleinfachheit

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

wobei

- u Boolescher Ausdruck über der Variablen v,
- t Term (sog. Terminierungsterm),
- w Variable, die in I, b, π und t nicht frei vorkommt,
- $M{=}_{df} \{\sigma(v) \mid \sigma \in Ch(u)\}$ bzgl. $\sqsubset$ Noethersch geordnete Menge (sog. noethersche Halbordnung).

$$\rightsquigarrow (M, \sqsubset) \text{ als Terminationsordnung!}$$

Vergleich von [while$_{TK}$] und [while$_{TK}$]

Beachte:

- [while$_{TK}$]: beliebige Noethersche Ordnung als Terminationsordnung zulässig: $(M, \sqsubset)$
- [while$'_{TK}$]: statt beliebiger Noetherscher Terminationsordnung Festlegung auf eine spezielle Noethersche Ordnung als Terminationsordnung, nämlich: $(\mathbb{N}, <)$.

Oft erfordert die Rückspiegelung einer sich 'natürlich' anbietenden Noetherschen Terminationsordnung auf die spezielle Noethersche Ordnung $(\mathbb{N}, <)$ zusätzlichen Modellierungsaufwand.

In diesen Fällen hat [while$_{TK}$] pragmatische Vorteile gegenüber Regel [while$'_{TK}$].

Zur Vollständigkeit

...seien die übrigen Regeln des Hoare-Kalkül HK_{TK} für totale Korrektheit hier ebenfalls angegeben:

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

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

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

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

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

Bemerkung

In den vorigen Regeln verwenden wir (auch) für totale Korrektheit geschweifte statt eckiger Klammern für zugesicherte Eigenschaften, um einen Bezeichnungskonflikt mit der ebenfalls durch eckige Klammern bezeichneten syntaktischen Substitution zu vermeiden.

Wohlfundierte Ordnungen (1)

Definition 4.3.1 (Irreflexive partielle Ordnung)

Sei P eine Menge und sei $<$ eine irreflexive und transitive Relation auf P.

Dann ist das Paar $(P, <)$ eine irreflexive partielle Ordnung.

Beispiele: $(\mathbb{Z}, <)$, $(\mathbb{Z}, >)$, $(\mathbb{N}, <)$, $(\mathbb{N}, >)$

Wohlfundierte Ordnungen (2)

Definition 4.3.2 (Wohlfundierte Ordnung)

Sei $(P, <)$ eine irreflexive partielle Ordnung und sei W eine Teilmenge von P.

Dann heißt die Relation $<$ auf W wohlfundiert, wenn es keine unendlich absteigende Kette

$$\ldots < w_2 < w_1 < w_0$$

von Elementen $w_i \in W$ gibt. Das Paar $(W, <)$ heißt dann eine wohlfundierte Struktur oder auch eine wohlfundierte oder Noethersche Ordnung.

Sprechweise: Gilt $w < w'$ für $w, w' \in W$, sagen wir, w ist kleiner als w' oder w' ist größer als w.

Beispiele: $(\mathbb{N}, <)$, aber nicht $(\mathbb{Z}, <)$, $(\mathbb{Z}, >)$ oder $(\mathbb{N}, >)$

Prinzipien zur Konstruktion wohlfundierter Ordnungen

...aus gegebenen wohlfundierten Ordnungen:

Lemma 4.3.3

Seien $(W_1, <_1)$ und $(W_2, <_2)$ zwei wohlfundierte Ordnungen. Dann sind auch

- $(W_1 \times W_2, <_{com})$ mit komponentenweiser Ordnung definiert durch

$$(m_1, m_2) <_{com} (n_1, n_2) \text{ gdw. } m_1 <_1 n_1 \wedge m_2 <_2 n_2$$

- $(W_1 \times W_2, <_{lex})$ mit lexikographischer Ordnung def. durch

$$(m_1, m_2) <_{lex} (n_1, n_2) \text{ gdw.}$$
$$(m_1 <_1 n_1) \vee (m_1 = n_1 \wedge m_2 <_2 n_2)$$

wohlfundierte Ordnungen.

Anmerkungen zu

...den der

- Konsequenzregel [cons] und der
- Schleifenregeln [while$_{PK}$] und [while$_{TK}$]

von HK_{PK} bzw. HK_{TK} zugrundeliegenden Intuitionen.

Zur Konsequenzregel (1)

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

Intuitiv:

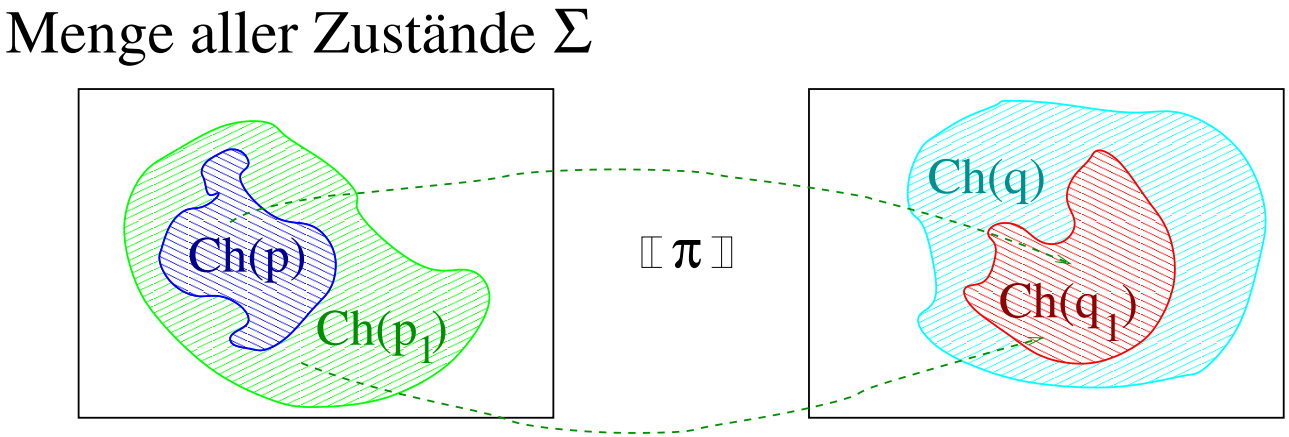
Die Konsequenzregel

- stellt die Schnittstelle zwischen Programmverifikation und den logischen Formeln der Zusicherungssprache dar
- erlaubt es,
  - 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.

Zur Konsequenzregel (2)

Veranschaulichung von Verstärkung und Abschwächung:



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

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

Zur while-Regel in HK_{PK}

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

Intuitiv:

- Das durch I beschriebene Prädikat gilt
  - vor und nach jeder Ausführung des Rumpfes der while-Schleife
  - und wird deswegen als Invariante der while-Schleife bezeichnet.
- Die while-Regel besagt weiter, dass
  - wenn zusätzlich (zur Invarianten) auch b vor jeder Ausführung des Schleifenrumpfs gilt, dass nach Beendigung der while-Schleife $\neg\, b$ wahr ist.

Zur while-Regel in HK_{TK} (1)

Erinnerung:

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

wobei

- u Boolescher Ausdruck über der Variablen v,
- t arithmetischer Term,
- w Variable, die in I, b, π und t nicht frei vorkommt,
- $M{=}_{df} \{\sigma(v) \mid Ch(u)\}$ bzgl. $\sqsubset$ Noethersch geordnete Menge (sog. Noethersche Halbordnung).

$\rightsquigarrow (M, \sqsubset)$ Terminationsordnung!

Zur while-Regel in HK_{TK} (2)

- Prämisse 1: $I \wedge b \Rightarrow u[t/v]$
  Wann immer der Schleifenrumpf noch einmal ausgeführt wird (d.h. $I \wedge b$ ist wahr), gilt, dass $u[t/v]$ wahr ist, woraus aufgrund der Definition von M folgt, dass der Wert von t Element einer Noethersch geordneten Menge ist.

Zur while-Regel in HK_{TK} (3)

- Prämisse 2: $\{I \wedge b \wedge t = w\}\ \pi\ \{I \wedge t < w\}$
  - w speichert den initialen Wert von t (w ist sog. *logische Variable*), d.h. den Wert, den t vor Eintritt in die Schleife hat (gilt, da w als logische Variable insbesondere nicht in π vorkommt)
  - Zusammen damit, dass der Wert von w (als logische Variable) invariant unter der Ausführung des Schleifenrumpfs ist, garantiert $t < w$ in der Nachbedingung von Prämisse 2, dass der Wert von t nach jeder Ausführung des Schleifenrumpfs bzgl. der noetherschen Ordnung abgenommen hat.

Zur while-Regel in HK_{TK} (4)

- Zusammen implizieren die obigen beiden Punkte die Terminierung der while-Schleife, da es in einer Noethersch geordneten Menge keine unendlich absteigenden Ketten gibt. Folglich kann die Bedingung $I \wedge b$ in Prämisse 1 nicht unendlich oft wahr sein, da dies zusammen mit Prämisse 2 ein unendliches Absteigen erforderte.)

Programm- vs. logische Variablen (1)

Wir unterscheiden in Zusicherungen $\{p\}\ \pi\ \{q\}$ zwischen:

- Programmvariablen

  ...Variablen, die in π vorkommen

- logischen Variablen

  ...Variablen, die in π nicht vorkommen

Logische Variablen erlauben

- sich initiale Werte von Programmvariablen zu "merken", um in Nachbedingungen geeignet darauf Bezug zu nehmen.

Programm- vs. logische Variablen (2)

Beispiel:

- $\{x = n\}\ y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x; x := x - 1 \text{ od } \{y = n! \wedge n > 0\}$
  ...die Nachbedingung macht eine Aussage über den Zusammenhang des Anfangswertes von x (gespeichert in n) und des schließlichen Wertes von y.
- $\{x = n\}\ y := 1;\ \text{while } x \neq 1 \text{ do } y := y * x; x := x - 1 \text{ od } \{y = x! \wedge x > 0\}$
  ...die Nachbedingung macht eine Aussage über den Zusammenhang der schließlichen Werte von x und y.
  (Beachte: nur mit Programmvariablen keine Aussage über die Fakultätsberechnung in diesem Bsp.!)

HK_{TK} versus HK_{PK}

Beachte:

HK_{TK} und HK_{PK} sind bis auf die (Prämisse der) Schleifenregel identisch:

- Totale Korrektheit: $[\text{while}_{TK}]$

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

- Partielle Korrektheit: $[\text{while}_{PK}]$

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

Abschließende Beobachtung

Beweistechnische Anmerkung:

"Zerlegt" man $[\text{while}'_{TK}]$ wie folgt:

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

wird deutlich, dass der Nachweis totaler Korrektheit einer Hoareschen Zusicherung besteht aus

- dem Nachweis ihrer partiellen Korrektheit
- dem Nachweis der Termination

Diese Trennung kann im Beweis explizit vollzogen werden ("Totale Korrektheit = Partielle Korrektheit + Terminierung"). Der Gesamtbeweis wird dadurch modular. Der Terminationsnachweis selbst ist oft einfach.

Bemerkung: Die obige Trennung kann für $[\text{while}_{TK}]$ analog vorgenommen werden.

Kapitel 4.4

Korrektheit und Vollständigkeit

Korrektheits- und Vollständigkeitsfrage

Sei K ein Kalkül für partielle bzw. totale Korrektheit

Zentral sind dann:

- Korrektheitsfrage: Ist jede mithilfe von K ableitbare (in Zeichen: $\vdash$) Korrektheitsformel partiell bzw. total korrekt?
- Vollständigkeitsfrage: Ist jede partiell bzw. total korrekte Korrektheitsformel (in Zeichen: $\models$) mithilfe von K ableitbar?

Speziell:

- Sind HK_{PK} und HK_{TK} korrekt und vollständig?

Korrektheits- u. Vollständigkeitsbedeutung

Definition 4.4.1 (Korrektheit und Vollständigkeit)

Sei K ein Hoarescher Beweiskalkül (z.B. HK_{PK} und HK_{TK}).

Dann heißt K

- korrekt (engl. sound), falls gilt: Ist eine Korrektheitsformel mit K herleitbar/beweisbar (in Zeichen: $\vdash \{p\}\ \pi\ \{q\}$), dann ist sie auch semantisch gültig:

$$\vdash \{p\}\ \pi\ \{q\} \Rightarrow \models \{p\}\ \pi\ \{q\}$$

- vollständig (engl. complete), falls gilt: Ist eine Korrektheitsformel semantisch gültig, dann ist sie auch mit K herleitbar/beweisbar:

$$\models \{p\}\ \pi\ \{q\} \Rightarrow \vdash \{p\}\ \pi\ \{q\}$$

Bemerkung

Die Sprechweise

- Hoaresches Tripel oder kurz Hoare-Tripel bzw.
- Hoaresche Zusicherung oder kurz Korrektsheitsformel

betont jeweils die

- syntaktische Sicht ($\vdash_{pk} \{p\}\ \pi\ \{q\}$) bzw.
- semantische Sicht ($\models_{pk} \{p\}\ \pi\ \{q\}$)

auf

- $\{p\}\ \pi\ \{q\}$ (analog für $[p]\ \pi\ [q]$)

Korrektheit von HK_{PK} und HK_{TK}

Theorem 4.4.2 (Korrektheit von HK_{PK}, HK_{TK})

1. HK_{PK} ist korrekt, d.h. jede mit HK_{PK} ableitbare Korrektheitsformel (in Zeichen: $\vdash_{pk} \{p\}\ \pi\ \{q\}$) ist gültig im Sinne partieller Korrektheit:

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

2. HK_{TK} ist korrekt, d.h. jede mit HK_{TK} ableitbare Korrektheitsformel (in Zeichen: $\vdash_{tk} [p]\ \pi\ [q]$) ist gültig im Sinne totaler Korrektheit:

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

Beweis durch Induktion über die Anzahl der Regelanwendungen im Beweisbaum zur Ableitung der Korrektheitsformel.

Zur Vollständigkeit Hoarescher Beweiskalküle

Generell müssen wir unterscheiden zwischen Vollständigkeit

- extensionaler und
- intensionaler

Ansätze.

Extensionale vs. intensionale Ansätze

- Extensional
  $\rightsquigarrow$ Vor- und Nachbedingungen sind durch Prädikate beschrieben.
- Intensional
  $\rightsquigarrow$ Vor- und Nachbedingungen sind durch Formeln einer Zusicherungssprache beschrieben.

Vollständigkeit von HK_{PK} und HK_{TK}

Für den extensionalen Ansatz gilt:

Theorem 4.4.3 (Vollständigkeit von HK_{PK}&HK_{TK})

1. HK_{PK} ist vollständig, d.h. jede im Sinne partieller Korrektheit gültige Korrektheitsformel ist mit HK_{PK} ableitbar:

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

2. HK_{TK} ist vollständig, d.h. jede im Sinne totaler Korrektheit gültige Korrektheitsformel ist mit HK_{TK} ableitbar:

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

Beweis durch strukturelle Induktion über den Aufbau von π.

Zur Vollständigkeit von HK_{PK} und HK_{TK}

Für intensionale Ansätze (durch unterschiedliche Wahlen der Zusicherungssprache) gilt Vollständigkeit i.a. nur relativ zur Entscheidbarkeit und Ausdruckskraft der Zusicherungssprache.

Informell

- Entscheidbarkeit

  ...ist die Gültigkeit von Formeln der Zusicherungssprache algorithmisch verifizierbar bzw. falsifizierbar?

- Ausdruckskraft

  ...lassen sich alle Prädikate, insbesondere schwächste und schwächste liberale Vorbedingungen und Terminationsfunktionen, durch Formeln der Zusicherungssprache beschreiben?

  $\rightsquigarrow$ *tieferliegende Frage*: ...lassen sich schwächste Vorbedingungen, etc. syntaktisch ausdrücken?

Stichwort: Relative Vollständigkeit im Sinne von Cook.

Kapitel 4.5

Beispiele zum Beweis partieller Korrektheit

Die beiden Beispiele im Überblick (1)

...Beweis der partiellen Korrektheit Hoarescher Zusicherungen anhand zweier Programme zur Berechnung

- der Fakultät und
- der ganzzahligen Division mit Rest

Die beiden Beispiele im Überblick (2)

Im Detail:

Beweise, dass die beiden Hoareschen Tripel

$$\{a > 0\}$$

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

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

und

$$\{x \geq 0 \wedge y > 0\}$$

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

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

gültig sind im Sinne partieller Korrektheit.

Die beiden Beispiele im Überblick (3)

In der Folge geben wir die Beweise dafür in zwei Notationsvarianten an:

- baumartig (kanonische Variante)
- lineare Beweisskizze (pragmatische Variante)

Fakultätsbeispiel: Baumartiger Beweis (1)

Erster Beweis

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{(y*x)*(x-1)!=a! \;\&\; (x-1)>0\}\ y:=y*x\ \{y*(x-1)!=a! \;\&\; (x-1)>0\}}\,\textbf{Ass} \quad \dfrac{}{\{y*(x-1)!=a! \;\&\; (x-1)>0\}\ x:=x-1\ \{y*x!=a! \;\&\; x>0\}}\,\textbf{Ass}}{\{(y*x)*(x-1)!=a! \;\&\; (x-1)>0\}\ y:=y*x;\ x:=x-1\ \{y*x!=a! \;\&\; x>0\}}\,\textbf{Comp}}{y*x!=a! \;\&\; x>0 \;\&\; x>1 \Longrightarrow (y*x)*(x-1)!=a! \;\&\; (x-1)>0}\,\textbf{Cons}}{\{y*x!=a! \;\&\; x>0 \;\&\; x>1\}\ y:=y*x;\ x:=x-1\ \{y*x!=a! \;\&\; x>0\}}\,\textbf{While}}{\{y*x!=a! \;\&\; x>0\}\ \text{WHILE } x>1 \text{ DO } y:=y*x;\ x:=x-1 \text{ OD } \{y*x!=a! \;\&\; x>0 \;\&\; \sim(x>1)\}\ y*x!=a! \;\&\; x>0 \;\&\; \sim(x>1) \Longrightarrow y=a!}\,\textbf{Cons}$$

$$\dfrac{\dfrac{\mathbf{T}}{\{a>0\}\ x:=a;\ y:=1\ \{y*x!=a! \;\&\; x>0\}}\,\textbf{Cons} \quad \{y*x!=a! \;\&\; x>0\}\ \text{WHILE } x>1 \text{ DO } y:=y*x;\ x:=x-1 \text{ OD } \{y=a!\}}{\{a>0\}\ x:=a;\ y:=1;\ \text{WHILE } x>1 \text{ DO } y:=y*x;\ x:=x-1 \text{ OD } \{y=a!\}}\,\textbf{Comp}$$

wobei

$$\mathbf{T} \equiv \left\{ \dfrac{\dfrac{\dfrac{}{\{1*a!=a! \;\&\; a>0 \;\&\; a=a\}\ x:=a\ \{1*x!=a! \;\&\; a>0 \;\&\; x=a\}}\,\textbf{Ass} \quad \dfrac{}{\{1*x!=a! \;\&\; a>0 \;\&\; x=a\}\ y:=1\ \{y*x!=a! \;\&\; a>0 \;\&\; x=a\}}\,\textbf{Ass}}{\{1*a!=a! \;\&\; a>0 \;\&\; a=a\}\ x:=a;\ y:=1\ \{y*x!=a! \;\&\; a>0 \;\&\; x=a\}}\,\textbf{Comp}}{a>0 \Longrightarrow 1*a!=a! \;\&\; a>0 \;\&\; a=a \quad \{1*a!=a! \;\&\; a>0 \;\&\; a=a\}\ x:=a;\ y:=1\ \{y*x!=a! \;\&\; a>0 \;\&\; x=a\} \Longrightarrow y*x!=a! \;\&\; x>0}\,\textbf{Cons} \right.$$

$$\{a>0\}\ x:=a;\ y:=1\ \{y*x!=a! \;\&\; x>0\}$$

& : Logisches und

~ : Logisches nicht

Fakultätsbeispiel: Baumartiger Beweis (2)

Zweiter Beweis

$$\dfrac{\dfrac{\mathbf{T}^{\mathrm{I}}}{\{a>0\}\ x:=a;\ y:=1\ \{y*x!=a!\ \&\ x>0\}}\ \textbf{Cons} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{(y*x)*(x-1)!=a!\ \&\ (x-1)>0\}\ y:=y*x\ \{y*(x-1)!=a!\ \&\ (x-1)>0\}}\ \textbf{Ass} \qquad \dfrac{}{\{y*(x-1)!=a!\ \&\ (x-1)>0\}\ x:=x-1\ \{y*x!=a!\ \&\ x>0\}}\ \textbf{Ass}}{\{(y*x)*(x-1)!=a!\ \&\ (x-1)>0\}\ y:=y*x;\ x:=x-1\ \{y*x!=a!\ \&\ x>0\}}\ \textbf{Comp}}{y*x!=a!\ \&\ x>0\ \&\ x>1 \Longrightarrow (y*x)*(x-1)!=a!\ \&\ (x-1)>0}\ \textbf{Cons}}{\{y*x!=a!\ \&\ x>0\ \&\ x>1\}\ y:=y*x;\ x:=x-1\ \{y*x!=a!\ \&\ x>0\}}\ \textbf{While}}{\{y*x!=a!\ \&\ x>0\}\ \text{WHILE } x>1 \text{ DO } y:=y*x;\ x:=x-1 \text{ OD } \{y*x!=a!\ \&\ x>0\ \&\ \sim(x>1)\}}\ \textbf{Cons}}{\{a>0\}\ x:=a;\ y:=1;\ \text{WHILE } x>1 \text{ DO } y:=y*x;\ x:=x-1 \text{ OD } \{y=a!\}}\ \textbf{Comp}$$

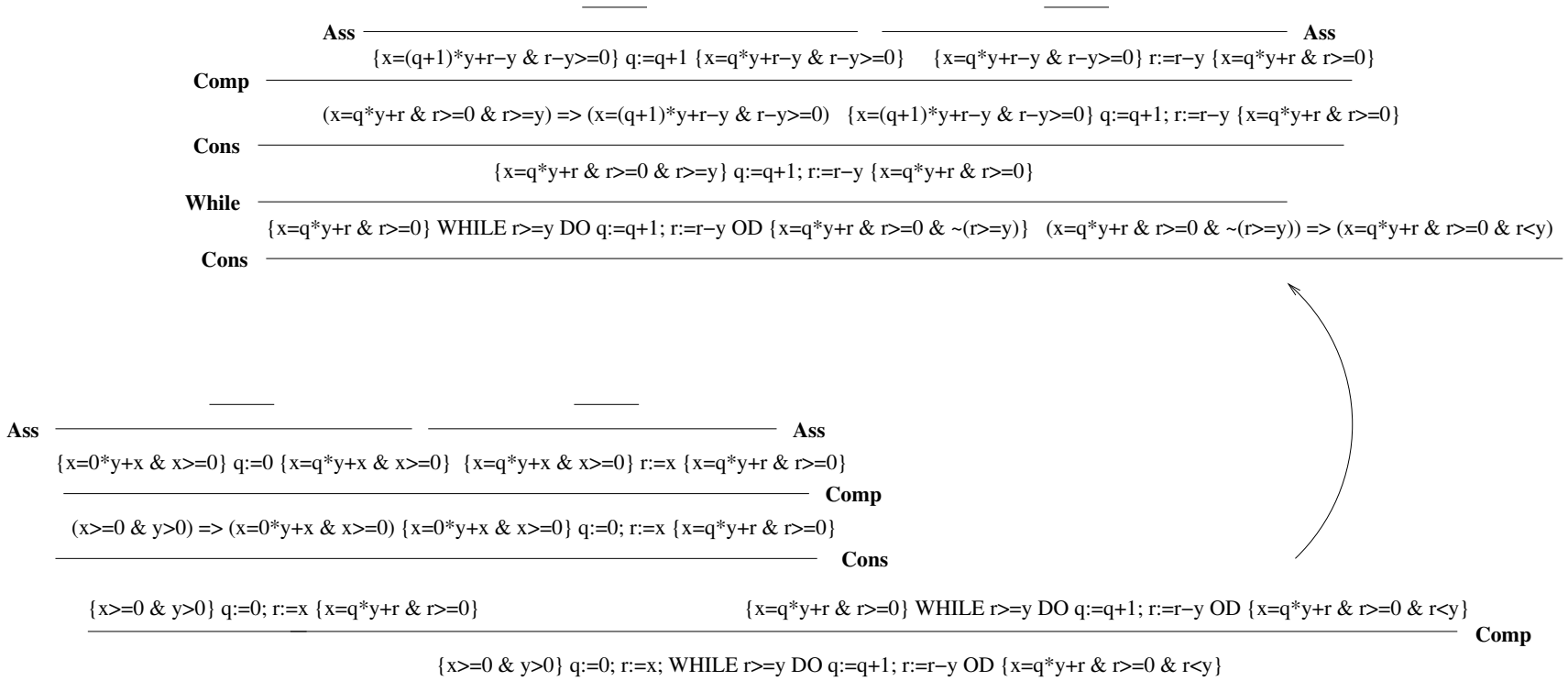
wobei

$$\mathbf{T}^{\mathrm{I}} \equiv \left\{ \dfrac{\dfrac{\dfrac{}{\{1*a!=a!\ \&\ a>0\}\ x:=a\ \{1*x!=a!\ \&\ x>0\}}\ \textbf{Ass} \qquad \dfrac{}{\{1*x!=a!\ \&\ x>0\}\ y:=1\ \{y*x!=a!\ \&\ x>0\}}\ \textbf{Ass}}{a>0 \Longrightarrow 1*a!=a!\ \&\ a>0 \qquad \{1*a!=a!\ \&\ a>0\}\ x:=a;\ y:=1\ \{y*x!=a!\ \&\ x>0\}}\ \textbf{Comp}}{\{a>0\}\ x:=a;\ y:=1\ \{y*x!=a!\ \&\ x>0\}}\ \textbf{Cons} \right.$$

& : Logisches und

~ : Logisches nicht

Divisionsbeispiel: Baumartiger Beweis



& : Logisches und

~ : Logisches nicht

Lineare Beweisskizzen

- Die unmittelbare baumartige Notation von Hoareschen Korrektheitsbeweisen ist i.a. unhandlich.
- Alternativ hat sich deshalb eine Notationsvariante eingebürgert, bei der in den Programmtext Zusicherungen als Annotationen eingestreut werden.
- In der Folge demonstrieren wir diesen Notationsstil am Beispiel des Nachweises der partiellen Korrektheit unseres Fakultätsprogramms bezüglich der angegebenen Vor- und Nachbedingung. Man spricht auch von einem sog. linearen Beweis bzw. einer linearen Beweisskizze.

Fakultätsbeispiel: Lineare Beweisskizze (1)

Beweise, dass das Hoaresche Tripel

$$\{a > 0\}$$

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

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

gültig ist im Sinne partieller Korrektheit.

Wir entwickeln den Beweis in der Folge Schritt für Schritt!

Fakultätsbeispiel: Lineare Beweisskizze (2)

Schritt 1

"Träumen" der Invariante:

- $\{y * x! = a! \wedge x > 0\}$

...um die [while]-Regel anwenden zu können.

Fakultätsbeispiel: Lineare Beweisskizze (3)

Schritt 2 Behandlung des Rumpfs der while-Schleife

Der Nachweis der Gültigkeit von

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$y := y * x;$$

$$x := x - 1;$$

$$\{y * x! = a! \wedge x > 0\}$$

erlaubte mithilfe der [while]-Regel den Übergang zu:

$$\{y * x! = a! \wedge x > 0\}$$

$$\text{while } x > 1 \text{ do}$$

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$y := y * x;$$

$$x := x - 1;$$

$$\{y * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

Fakultätsbeispiel: Lineare Beweisskizze (4)

Behandlung des Rumpfs der while-Schleife im Detail:

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$y := y * x;$$

$$x := x - 1;$$

$$\{y * x! = a! \wedge x > 0\}$$

Fakultätsbeispiel: Lineare Beweisskizze (5)

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

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$y := y * x;$$

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

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

$$\{y * x! = a! \wedge x > 0\}$$

Fakultätsbeispiel: Lineare Beweisskizze (6)

Nach abermaliger Anwendung der [ass]-Regel erhalten wir

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

...wobei noch eine "Beweislücke" verbleibt!

Fakultätsbeispiel: Lineare Beweisskizze (7)

Schluss der "Beweislücke" in der zugrundeliegenden Theorie:

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

Fakultätsbeispiel: Lineare Beweisskizze (8)

Anwendung der [while]-Regel liefert nun wie gewünscht:

$$\{y * x! = a! \wedge x > 0\}$$

$$\text{while } x > 1 \text{ do}$$

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

Fakultätsbeispiel: Lineare Beweisskizze (9)

Schritt 3 Zur gewünschten Nachbedingung verbleibt offenbar ebenfalls eine Beweislücke:

$$\{y * x! = a! \wedge x > 0\}$$

$$\text{while } x > 1 \text{ do}$$

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

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

Fakultätsbeispiel: Lineare Beweisskizze (10)

Schluss der Beweislücke in der zugrundeliegenden Theorie:

$$\{y * x! = a! \wedge x > 0\}$$

while $x > 1$ do

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

od [while]

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

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

$$\{y * x! = a! \wedge x > 0 \wedge x \leq 1\}$$

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

$$\{y * x! = a! \wedge x = 1\}$$

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

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

Fakultätsbeispiel: Lineare Beweisskizze (11)

Aus Platzgründen etwas verkürzt dargestellt:

$$\{y * x! = a! \wedge x > 0\}$$

$$\text{while } x > 1 \text{ do}$$

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

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

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

Fakultätsbeispiel: Lineare Beweisskizze (12)

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

$$\{a > 0\}$$

$$x := a;$$

$$y := 1;$$

$$\{y * x! = a! \wedge x > 0\}$$

$$\text{while } x > 1 \text{ do}$$

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

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

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

Fakultätsbeispiel: Lineare Beweisskizze (13)

Einmalige Anwendung der [ass]-Regel liefert:

$$\{a > 0\}$$

$$x := a;$$

$$\{1 * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0\}$$

$$\text{while } x > 1 \text{ do}$$

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

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

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

Fakultätsbeispiel: Lineare Beweisskizze (14)

Abermalige Anwendung der [ass]-Regel liefert:

$$\{a > 0\}$$

$$\{1 * a! = a! \wedge a > 0\}$$

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

$$\{1 * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0\}$$

$$\text{while } x > 1 \text{ do}$$

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

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

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

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

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

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

$$\{y * x! = a! \wedge x > 0\}$$

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

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

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

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

Fakultätsbeispiel: Lineare Beweisskizze (15)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

$$\{a > 0\}$$
$$\Downarrow \text{[cons]}$$
$$\{1 * a! = a! \wedge a > 0\}$$
$$x := a; \text{ [ass]}$$
$$\{1 * x! = a! \wedge x > 0\}$$
$$y := 1; \text{ [ass]}$$
$$\{y * x! = a! \wedge x > 0\}$$
$$\text{while } x > 1 \text{ do}$$
$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$
$$\Downarrow \text{[cons]}$$
$$\{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\}$$
$$y := y * x; \text{ [ass]}$$
$$\{y * (x - 1)! = a! \wedge x - 1 > 0\}$$
$$x := x - 1; \text{ [ass]}$$
$$\{y * x! = a! \wedge x > 0\}$$
$$\text{od [while]}$$
$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$
$$\Downarrow \text{[cons]}$$
$$\{y = a!\}$$

Insgesamt (16)

$$\{a > 0\}$$
$$\Downarrow \text{ [cons]}$$
$$\{1 * a! = a! \wedge a > 0\}$$
$$x := a; \text{ [ass]}$$
$$\{1 * x! = a! \wedge x > 0\}$$
$$y := 1; \text{ [ass]}$$
$$\{y * x! = a! \wedge x > 0\}$$
$$\text{while } x > 1 \text{ do}$$
$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$
$$\Downarrow \text{ [cons]}$$
$$\{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\}$$
$$y := y * x; \text{ [ass]}$$
$$\{y * (x - 1)! = a! \wedge x - 1 > 0\}$$
$$x := x - 1; \text{ [ass]}$$
$$\{y * x! = a! \wedge x > 0\}$$
$$\text{od [while]}$$
$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$
$$\Downarrow \text{ [cons]}$$
$$\{y * x! = a! \wedge x > 0 \wedge x \leq 1\}$$
$$\Downarrow \text{ [cons]}$$
$$\{y * x! = a! \wedge x = 1\}$$
$$\Downarrow \text{ [cons]}$$
$$\{y = a!\}$$

Fakultätsbeispiel: Lineare Beweisskizze (17)

Damit haben wir insgesamt wie gewünscht gezeigt:

Das Hoaresche Tripel

$$\{a > 0\}$$

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

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

ist gültig im Sinne partieller Korrektheit.

Vergleich linearen und baumartigen Beweisstils

Vorteil linearen gegenüber baumartigen Beweisnotationsstils:

- wenig Redundanz
- deshalb insgesamt knappere Beweise

Kapitel 4.6

Beispiele zum Beweis totaler Korrektheit

Das Beispiel im Überblick

Beweise, dass das Hoaresche Tripel

$$[a > 0]$$

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

$$[y = a!]$$

gültig ist im Sinne totaler Korrektheit.

Wir entwickeln den Beweis in der Folge Schritt für Schritt!

Wahl von Invariante und Terminierungsterm

Schritt 1

"Träumen"

- der Invariante: $y * x! = a! \wedge x > 0$
- des Terminierungsterms: $t \equiv x$
- von u: $u \equiv v \geq 0$

...um die [while]-Regel anwenden zu können.

Beachte:

- Aus der Wahl von $u \equiv v \geq 0$ und von $b \equiv x > 1$ folgt:
  - $M = \{0, 1, 2, 3, 4, \ldots\}$
  - $(v \geq 0)[x/v] \equiv x \geq 0$

  ...und somit insgesamt: $I \ \wedge \ b \Rightarrow \sigma(x) \in M$ mit $(M, <)$ Noethersch geordnet.

Hinweis zur Notation: $\equiv$ steht für *syntaktisch gleich*

Wahl von Invariante und Terminierungsterm

Mit der vorherigen Wahl von I, t und u gilt:

$$\begin{aligned} M &=_{df} \{\sigma(v) \mid \sigma \in Ch(u)\} \\ &= \{\sigma(v) \mid \sigma \in Ch(v \geq 0)\} \\ &= \{\sigma(v) \mid \sigma \in \Sigma \ \wedge \ groessergleich([\![v]\!]_A(\sigma), [\![0]\!]_A(\sigma))\} \\ &= \{\sigma(v) \mid \sigma \in \Sigma \ \wedge \ groessergleich(\sigma(v), \mathbf{0}) = \mathbf{wahr}\} \\ &= \{\sigma(v) \mid \sigma \in \Sigma \ \wedge \ \sigma(v) \geq \mathbf{0}\} \\ &= \mathbb{N} \cup \{\mathbf{0}\} \end{aligned}$$

Damit haben wir insbesondere:

- $(M, <) = (\mathbb{N} \cup \{\mathbf{0}\}, <)$ ist Noethersch geordnet.
- $u[t/x] = (v \geq 0)[x/v] = x \geq 0$

Bemerkung

Der Beweis wird schrittweise in Form einer linearen Beweisskizze präsentiert.

Fakultätsbeispiel: Lineare Beweisskizze (1)

Schritt 2 Behandlung des Rumpfs der while-Schleife...

Der Nachweis der Gültigkeit von

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$$y := y * x;$$

$$x := x - 1;$$

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

erlaubte mithilfe der [while]-Regel den Übergang zu:

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

while $x > 1$ do

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$$y := y * x;$$

$$x := x - 1;$$

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

od [while]

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

Fakultätsbeispiel: Lineare Beweisskizze (2)

Behandlung des Rumpfs der while-Schleife im Detail:

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$$y := y * x;$$

$$x := x - 1;$$

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

Fakultätsbeispiel: Lineare Beweisskizze (3)

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

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$$y := y * x;$$
$$[y * (x-1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$
$$x := x - 1; \text{ [ass]}$$
$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

Fakultätsbeispiel: Lineare Beweisskizze (4)

Nach abermaliger Anwendung der [ass]-Regel erhalten wir...

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

...wobei noch eine "Beweislücke" verbleibt!

Fakultätsbeispiel: Lineare Beweisskizze (5)

Schluss der "Beweislücke" in der zugrundeliegenden Theorie:

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$
$$\Downarrow [\text{cons}]$$
$$[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$
$$y := y * x; [\text{ass}]$$
$$[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$
$$x := x - 1; [\text{ass}]$$
$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

Fakultätsbeispiel: Lineare Beweisskizze (6)

Anwendung der [while]-Regel liefert nun wie gewünscht:

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

$$\text{while } x > 1 \text{ do}$$

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$

$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

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

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

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

Fakultätsbeispiel: Lineare Beweisskizze (7)

Schritt 3: Zur gewünschten Nachbedingung verbleibt offenbar ebenfalls eine Beweislücke:

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

$$\text{while } x > 1 \text{ do}$$

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

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

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

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

$$[y = a!]$$

Fakultätsbeispiel: Lineare Beweisskizze (8)

Schluss der Beweislücke in der zugrundeliegenden Theorie:

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

while $x > 1$ do

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$\Downarrow$ [cons]

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

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

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

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

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

od [while]

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

$\Downarrow$ [cons]

$$[y * x! = a! \wedge x > 0 \wedge x \leq 1]$$

$\Downarrow$ [cons]

$$[y * x! = a! \wedge x = 1]$$

$\Downarrow$ [cons]

$$[y = a!]$$

Fakultätsbeispiel: Lineare Beweisskizze (9)

Aus Platzgründen etwas verkürzt dargestellt:

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

$$\text{while } x > 1 \text{ do}$$

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

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

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

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

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

$$[y = a!]$$

Fakultätsbeispiel: Lineare Beweisskizze (10)

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

$$[a > 0]$$

$$x := a;$$

$$y := 1;$$

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

$$\text{while } x > 1 \text{ do}$$

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

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

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

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

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

$$[y = a!]$$

Fakultätsbeispiel: Lineare Beweisskizze (11)

Einmalige Anwendung der [ass]-Regel liefert:

$$[a > 0]$$

$$x := a;$$

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

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

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

$$\text{while } x > 1 \text{ do}$$

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

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

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

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

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

$$[y = a!]$$

Fakultätsbeispiel: Lineare Beweisskizze (12)

Abermalige Anwendung der [ass]-Regel liefert:

$$[a > 0]$$

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

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

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

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

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

$$\text{while } x > 1 \text{ do}$$

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

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

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

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

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

$$[y = a!]$$

Fakultätsbeispiel: Lineare Beweisskizze (13)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

$$[a > 0]$$

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

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

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

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

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

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

$$\text{while } x > 1 \text{ do}$$

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

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

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

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

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

$$[y = a!]$$

Insgesamt (14)

$$[a > 0]$$

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

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

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

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

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

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

$$\text{while } x > 1 \text{ do}$$

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

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

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

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

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

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

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

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

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

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

$$[y * x! = a! \wedge x > 0 \wedge x \leq 1]$$

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

$$[y * x! = a! \wedge x = 1]$$

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

$$[y = a!]$$

Fakultätsbeispiel: Lineare Beweisskizze (15)

Damit haben wir wie gewünscht insgesamt gezeigt:

Das Hoaresche Tripel

$$[a > 0]$$

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

$$[y = a!]$$

ist gültig im Sinne totaler Korrektheit.

Kapitel 4.7

Automatische Ansätze axiomatischer Programmverifikation

Ansätze zu (semi-) automatischer axiomatischer Programmverifikation

Unter anderem:

- Theorema, RISC, JKU Linz
- KeY-Hoare, KIT Karlsruhe, Chalmers University of Technology, TU Darmstadt
- ...

Theorema-Projekt, www.theorema.org (1)

"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.

Theorema-Projekt, www.theorema.org (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, www.key-project.org (1)

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, www.key-project.org (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 Handreichung:

- 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.
  htpp://i12www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf

Kapitel 4.8

Historische Meilensteine der Programmverifikation

Meilensteine der Programmverifikation (1)

Erste Anfänge

1949 Turings Vision: Korrekte Programme
Beispiel Fakultätsfunktion: Zusicherungen und Terminierungsfunktion

Axiomatische Methode

1967 Floyd: Flussdiagramme
Hoare: while-Programme

Erweiterung dieser 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, objekt-orientierte 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: Model Checking
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 4.9

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., 3rd edition, 2009. (Chapter 3, While Programs; Chap. 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*. 2nd edition, 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 teh 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 pages. `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)

Tudor Jebelean, Laura Kovács, Nikolaj Popov. *Experimental Program Verification in the Theorema System*. In Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004), 92-99, 2004. `www.risc.jku.at/publications/download/risc_2243/KoPoJeb.pdf`

Laura Kovács, Tudor Jebelean. *Practical Aspects of Imperative Program Verification using Theorema*. In Proceedings of the 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2003), 317-320, 2003. `www.risc.jku.at/publications/download/risc_464/synasc03.pdf`

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (8)

Laura Kovács, Tudor Jebelean. *Generation of Invariants in Theorema*. In Proceedings of the 10th International Symposium of Mathematics and its Applications, 407-415, 2003. `www.risc.jku.at/publications/download/risc_2053/2003-11-06-A.pdf`

Dexter Kozen, Jerzy Tiuryn. *On the Completeness of Propositional Hoare Logic*. Information Sciences 139(3-4):187-195, 2001.

Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009. (Chapter 1, Introduction: What do we want to know about the Program?, Chapter 2, How to prove a Program Correct: Programs without Loops; Chapter 3, How to prove a Program Correct: Iterative Programs)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (9)


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


Konstantinos Mamouras. *On the Hoare Theory of Monadic Recursion Schemes*. In Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (CSL-LICS'14), Article 69, 69.1-69.10, 2014. ACM Transactions on Computational Logic 17(2):13.1-13.30, 2016.


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 (10)

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 (11)


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 (12)


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

Worst-Case Execution Time Analyse

Kapitel 5.1

Motivation

Motivation

Von Verifikation zu Analyse:

- Worst-Case Execution Time-Analyse als erstes Beispiel

...nach

- Hanne Riis Nielson, Flemming Nielson. Semantics with Applications – A Formal Introduction. Wiley, 1992.

Worst-Case Execution Time (WCET)-Analyse

Motivation:

- In vielen Anwendungsbereichen sind Aussagen über die Ausführungszeit erforderlich.
- Der Nachweis totaler Korrektheit garantiert zwar Terminierung, sagt aber nichts über den Ressourcen-, speziell den Zeitbedarf aus.

In der Folge:

- Erweiterung und Adaptierung des Beweissystems für totale Korrektheit, um solche Aussagen zu ermöglichen.

Die grundlegende Idee (1)

...zur Zuordnung von Ausführungszeiten:

- Leere Anweisung
  ...Ausführungszeit in $\mathcal{O}(\mathbf{1})$, d.h. Ausführungszeit ist beschränkt durch eine Konstante.
- Zuweisung
  ...Ausführungszeit in $\mathcal{O}(\mathbf{1})$.
- (Sequentielle) Komposition
  ...Ausführungszeit entspricht, bis auf einen konstanten Faktor, der Summe der Ausführungszeiten der Komponenten.

Die grundlegende Idee (2)

- Fallunterscheidung
  ...Ausführungszeit entspricht, bis auf einen konstanten Faktor, der größeren der Ausführungszeiten der beiden Zweige.
- (while)-Schleife
  ...Ausführungszeit der Schleife entspricht, bis auf einen konstanten Faktor, der Summe der wiederholten Ausführungszeiten des Rumpfes der Schleife.

Bemerkung: Verfeinerungen der Granularität sind offenbar möglich.

Formalisierung

...dieser grundlegenden Idee in 3 Schritten:

1. Angabe einer Semantik, die die Auswertungszeit arithmetischer und Boolescher Ausdrücke beschreibt (s. Kapitel 5.2).
2. Erweiterung und Adaption der natürlichen Semantik von *WHILE* zur Bestimmung der Ausführungszeit eines Programms (s. Kapitel 5.2).
3. Erweiterung und Adaption des Beweissystems für totale Korrektheit zum Nachweis über die Größenordnung der Ausführungszeit von Programmen (s. Kapitel 5.3).

Kapitel 5.2

Zeitbewusste natürliche Semantik

Erster Schritt

Festlegung von (abstrakten) Semantikfunktionen

- $[\![.]\!]_{TA} : \mathbf{Aexpr} \to \mathbb{Z}$ und
- $[\![.]\!]_{TB} : \mathbf{Bexpr} \to \mathbb{Z}$

zur Beschreibung der Auswertungszeit arithmetischer und Boolescher Ausdrücke (in Zeiteinheiten einer abstrakten Maschine).

Semantik zur Ausführungszeit der Auswertung arithmetischer Ausdrücke

$[\![\, . \,]\!]_{TA} : \mathbf{Aexpr} \to \mathbb{Z}$ induktiv definiert durch

- $[\![\, n \,]\!]_{TA} =_{df} \mathbf{1}$
- $[\![\, x \,]\!]_{TA} =_{df} \mathbf{1}$
- $[\![\, a_1 + a_2 \,]\!]_{TA} =_{df} [\![\, a_1 \,]\!]_{TA} + [\![\, a_2 \,]\!]_{TA} + \mathbf{1}$
- $[\![\, a_1 * a_2 \,]\!]_{TA} =_{df} [\![\, a_1 \,]\!]_{TA} + [\![\, a_2 \,]\!]_{TA} + \mathbf{1}$
- $[\![\, a_1 - a_2 \,]\!]_{TA} =_{df} [\![\, a_1 \,]\!]_{TA} + [\![\, a_2 \,]\!]_{TA} + \mathbf{1}$
- $[\![\, a_1 / a_2 \,]\!]_{TA} =_{df} [\![\, a_1 \,]\!]_{TA} + [\![\, a_2 \,]\!]_{TA} + \mathbf{1}$
- ... (andere Operatoren analog, ggf. auch mit operationsspezifischen Kosten)

Semantik zur Ausführungszeit der Auswertung Boolescher Ausdrücke

$[\![\, . \,]\!]_{TB} : \mathbf{Bexpr} \to \mathbb{Z}$ induktiv definiert durch

- $[\![\, \mathit{true} \,]\!]_{TB} =_{df} \mathbf{1}$
- $[\![\, \mathit{false} \,]\!]_{TB} =_{df} \mathbf{1}$
- $[\![\, a_1 = a_2 \,]\!]_{TB} =_{df} [\![\, a_1 \,]\!]_{TA} + [\![\, a_2 \,]\!]_{TA} + \mathbf{1}$
- $[\![\, a_1 < a_2 \,]\!]_{TB} =_{df} [\![\, a_1 \,]\!]_{TA} + [\![\, a_2 \,]\!]_{TA} + \mathbf{1}$
- ... (andere Relatoren (z.B. $\leq$, ...) analog)

- $[\![\, \neg b \,]\!]_{TB} =_{df} [\![\, b \,]\!]_{TB} + \mathbf{1}$
- $[\![\, b_1 \wedge b_2 \,]\!]_{TB} =_{df} [\![\, b_1 \,]\!]_{TB} + [\![\, b_2 \,]\!]_{TB} + \mathbf{1}$
- $[\![\, b_1 \vee b_2 \,]\!]_{TB} =_{df} [\![\, b_1 \,]\!]_{TB} + [\![\, b_2 \,]\!]_{TB} + \mathbf{1}$

Anmerkungen zu $[\![.]\!]_{TA}$ und $[\![.]\!]_{TB}$

Die Semantikfunktionen

- $[\![\cdot]\!]_{TA}$
- $[\![\cdot]\!]_{TB}$

beschreiben intuitiv die Anzahl der Zeiteinheiten, die eine (hier nicht spezifizierte) abstrakte Maschine zur Auswertung arithmetischer und Boolescher Ausdrücke benötigt.

Zweiter Schritt

Erweiterung und Anpassung der

- natürlichen Semantik von *WHILE*

zur Bestimmung der Ausführungszeit von Programmen.

Idee

Übergang von Transitionen der Form

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

zu Transitionen der Form

$$\langle \pi, \sigma \rangle \ \rightarrow^t \ \sigma'$$

mit der Bedeutung, dass π angesetzt auf σ nach t Zeiteinheiten in σ' terminiert.

N-Semantik erweitert um den Zeitaspekt (1)

...für das Beispiel von *WHILE*:

$$[\text{skip}_{tns}] \quad \frac{}{\langle skip,\sigma\rangle \rightarrow^1 \sigma}$$

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

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

N-Semantik erweitert um den Zeitaspekt (2)

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

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

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

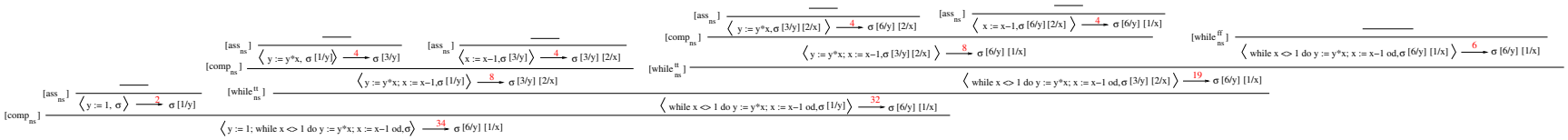
$$[\text{while}^{ff}_{tns}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \to^{[\![b]\!]_{TB}+3} \sigma} \quad [\![b]\!]_B(\sigma) = \textbf{falsch}$$

Beispiel zur nat. "Zeit"-Semantik (1)

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

Dann gilt:

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



Beispiel zur nat. "Zeit"-Semantik (2)

Das gleiche Beispiel in "etwas" gefälligerer Darstellung:

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

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

Kapitel 5.3

Zeitbewusste axiomatische Semantik

Dritter Schritt

Erweiterung und Anpassung der

- des Beweiskalküls für totale Korrektheit

um den Ausführungszeitaspekt von Programmen.

Idee (1)

Übergang zu Korrektheitsformeln der Form

$$\{p\}\ \pi\ \{e \Downarrow q\}$$

wobei

- p und q logische Formeln (wie bisher!) und
- $e \in \mathbf{Aexp}$ ein arithmetischer Ausdruck ist.

Idee (2)

Definition 5.1.1 (Gültigkeit von Korrektheitsformeln)

Die Korrektheitsformel

$$\{p\}\ \pi\ \{e \Downarrow q\}$$

ist gültig gdw. für jeden Anfangszustand σ 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, und die benötigte Ausführungszeit ist in $\mathcal{O}(e)$.

Idee (3)

Anders ausgedrückt:

Die Korrektheitsformel

$$\{p\}\ \pi\ \{e \Downarrow q\}$$

ist gültig (in Zeichen: $\models\ \{p\}\ \pi\ \{e \Downarrow q\}$) gdw.

es existiert eine natürliche Zahl **k**, so dass für alle Zustände σ gilt:

Ist die Vorbedingung p in σ erfüllt, dann gibt es einen Zustand σ' und eine natürliche Zahl t, so dass

die Nachbedingung q in σ' erfüllt ist und weiters gilt:

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

Idee (4)

Beachte:

- Im Ausdruck

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

wird der Ausdruck e im Anfangszustand σ ausgewertet, nicht im terminalen Zustand σ'.

- Diesem Umstand ist geschuldet, dass die (jetzt folgende) Festlegung der Regeln $[comp_e]$ und $[while_e]$ komplizierter ausfällt als möglicherweise zunächst vermutet.

Intuition zu den Kalkülregeln (1)

...für $[comp_e]$:

- Die $[comp_e]$-Regel setzt voraus, dass es Beweise gibt, die zeigen, dass e_1 und e_2 die Größenordnung der Zahl der Schritte angeben zur Ausführung von π_1 und π_2.
- e_1 drückt dies für π_1 relativ zum Anfangszustand von π_1 aus; e_2 drückt dies für π_2 relativ zum Anfangszustand von π_2 aus.
- Aus diesem Grund drückt nicht einfach die Summe $e_1 + e_2$ das Zeitverhalten der sequentiellen Komposition $\pi_1; \pi_2$ aus.
- Vielmehr muss e_2 durch einen Ausdruck e_2' ersetzt werden, so dass e_2' ausgewertet im Anfangszustand von e_1 den Wert von e_2 im Anfangszustand von π_2 beschränkt.
- Dies wird durch die erweiterte Vor- und Nachbedingung von π_1 unter Verwendung der frischen logischen Variable u erreicht.

Intuition zu den Kalkülregeln (2)

...für $[\mathit{while}_e]$:

- Die $[\mathit{while}_e]$-Regel geht davon aus, dass die Ausführungszeit des Schleifenrumpfs durch e_1, die der gesamten Schleife durch e beschränkt ist. Wie für die $[\mathit{comp}_e]$-Regel drückt die Summe $e_1 + e$ nicht das Zeitverhalten der gesamten Schleife aus, da e_1 sich auf den Zustand vor Ausführung des Rumpfs der while-Schleife Bezug nimmt und e auf den Zustand nach einmaliger Ausführung des Schleifenrumpfs.
- Ähnlich wie für die $[\mathit{comp}_e]$-Regel wird ein Ausdruck e' benötigt, der vor Ausführung des Schleifenrumpfs ausgewertet Ausdruck e nach dessen Ausführung beschränkt.
- Dann muss gelten, dass e die Ungleichung $e \geq e_1 + e'$ erfüllt, da e die Ausführung der while-Schranke unabhängig von der Anzahl ihrer Wiederholungen beschränken muss.

Axiomatische Semantik mit Zeitaspekt (1)

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

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

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

wobei u frische logische Variable ist

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

$$[\mathrm{cons}_e] \quad \frac{p\Rightarrow p' \quad \{p'\}\ \pi\ \{e'\Downarrow q'\} \quad q'\Rightarrow q}{\{p\}\ \pi\ \{e\Downarrow q\}}$$

wobei (für eine natürliche Zahl $\mathbf{k}$) gilt: $e' \leq \mathbf{k} * e$

Axiomatische Semantik mit Zeitaspekt (2)

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

wobei gilt:

$p(z+1) \Rightarrow (b \wedge e \geq e_1 + e')$,
$p(0) \Rightarrow (\neg\, b \wedge 1 \leq e)$,
u ist eine frische logische Variable,
z nimmt Werte aus den natürlichen Zahlen an,
d.h. $z \geq 0$.

Beispiele (1)

Die Korrektheitsformel

$$\{x=3\}\ \texttt{y:=1; while x/=1 do y:=y*x; x:=x-1 od}\ \{1 \Downarrow \mathrm{True}\}$$

beschreibt, dass die Ausführungszeit des Fakultätsprogramms angesetzt auf einen Zustand, in dem x den Wert $\mathbf{3}$ hat, von der Größenordnung von $\mathbf{1}$ ist, also durch eine Konstante beschränkt ist.

Beispiele (2)

Die Korrektheitsformel

$$\{x>0\}\ \texttt{y:=1; while x/=1 do y:=y*x; x:=x-1 od}\ \{x\Downarrow True\}$$

beschreibt, dass die Ausführungszeit des Fakultätsprogramms angesetzt auf einen Zustand, in dem x einen Wert größer als $\mathbf{0}$ hat, von der Größenordnung von x ist, also linear beschränkt ist.

Kapitel 5.4

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (1)


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.


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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (2)


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.


Marvin Damschen, Lars Bauer, Jörg Henkel. *Timing Analysis of Tasks on Runtime Reconfigurable Processors*. In IEEE Transactions on Very Large Scale Integration Systems, 2016.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (3)


Stephen A. Edwards, Edward A. Lee. *The Case for the Precision-timed (PRET) Machine*. In Proc. of Design Automat. Conference, ACM, 264-265.


J. 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.


J. Gustafsson, A. Betts. *The Mälardalen WCET Benchmarks: Past, Present, and Future*. In Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2011), 136-146, 2010.


T. Leveque, E. Borde, A. Marref, J. Carlson. *Hierarchical Composition of Parametric WCET in a Component Based Approach*. In Proceedings of the 14th IEEE International

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (4)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 10.2, Assertions for Execution Time)


G. Ottosson, M. Sjodin. *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, 1997.


Peter Puschner, Raimund Kirner, R.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*

Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (5)

- H. Theiling. *ILP-based Interprocedural Path Analysis*. Springer-V., LNCS 2491, 349-363, 2002.
- Lothar Thiele, Reinhard Wilhelm. *Design for Timing Predictability*. Real-Time Syst. 28(2-3):157-177, 2004.
- 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.

Teil II

Analyse

Kapitel 6

Programmanalyse

Programmanalyse

...speziell Datenflussanalyse.

Typische Fragen sind:

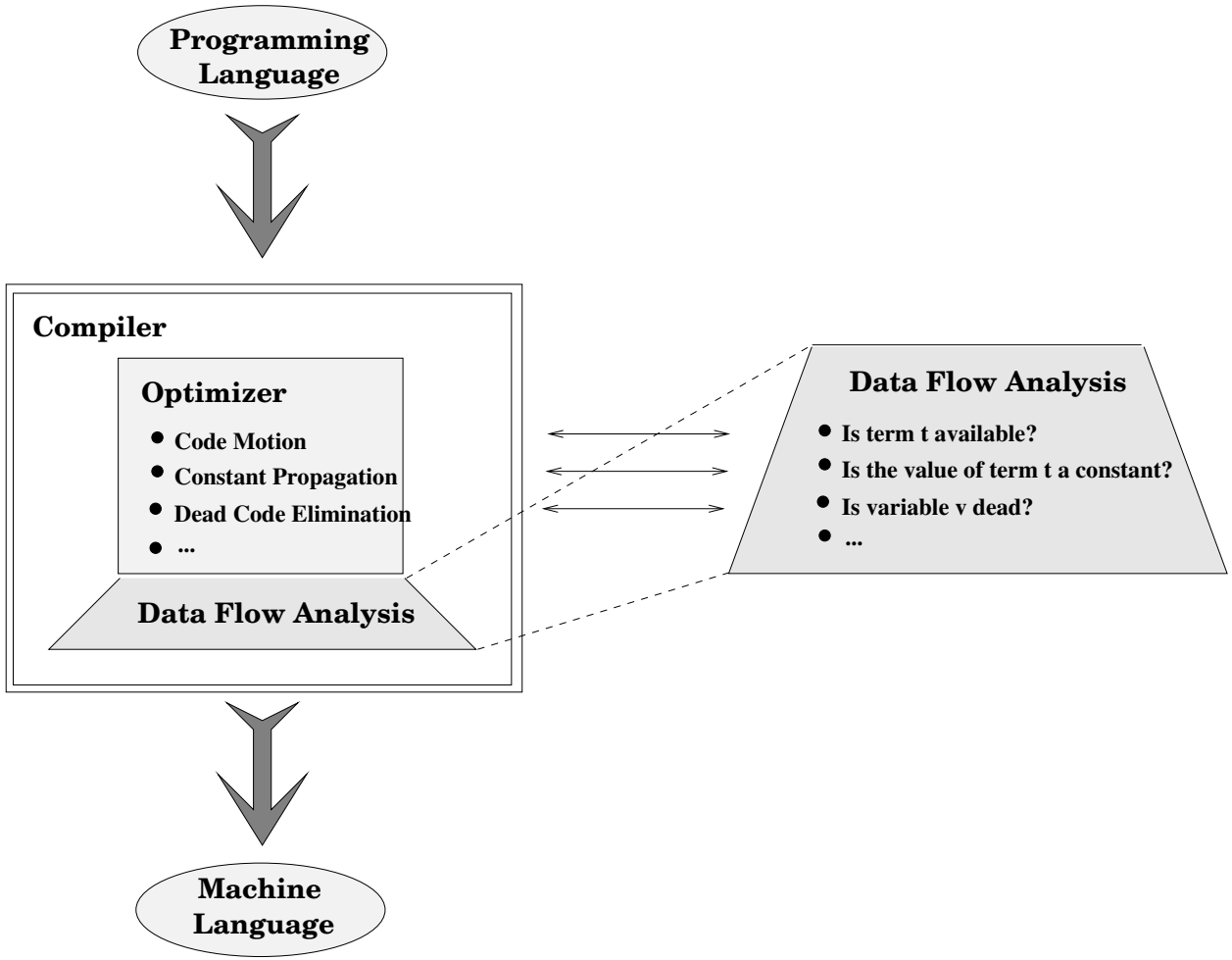
- Welchen Wert hat eine Variable an einer Programmstelle?
  $\rightsquigarrow$ Konstantenausbreitung und Faltung
- Steht der Wert eines Ausdrucks an einer Programmstelle verfügbar?
  $\rightsquigarrow$ Elimination (partiell) redundanten Codes, Code Motion
- Ist eine Variable tot an einer Programmstelle?
  $\rightsquigarrow$ Elimination (partiell) toten Codes

Kapitel 6.1

Motivation

Motivation

...(Programm-) Analyse zur (Programm-) Optimierung



Zentrale Fragen

Grundlegendes:

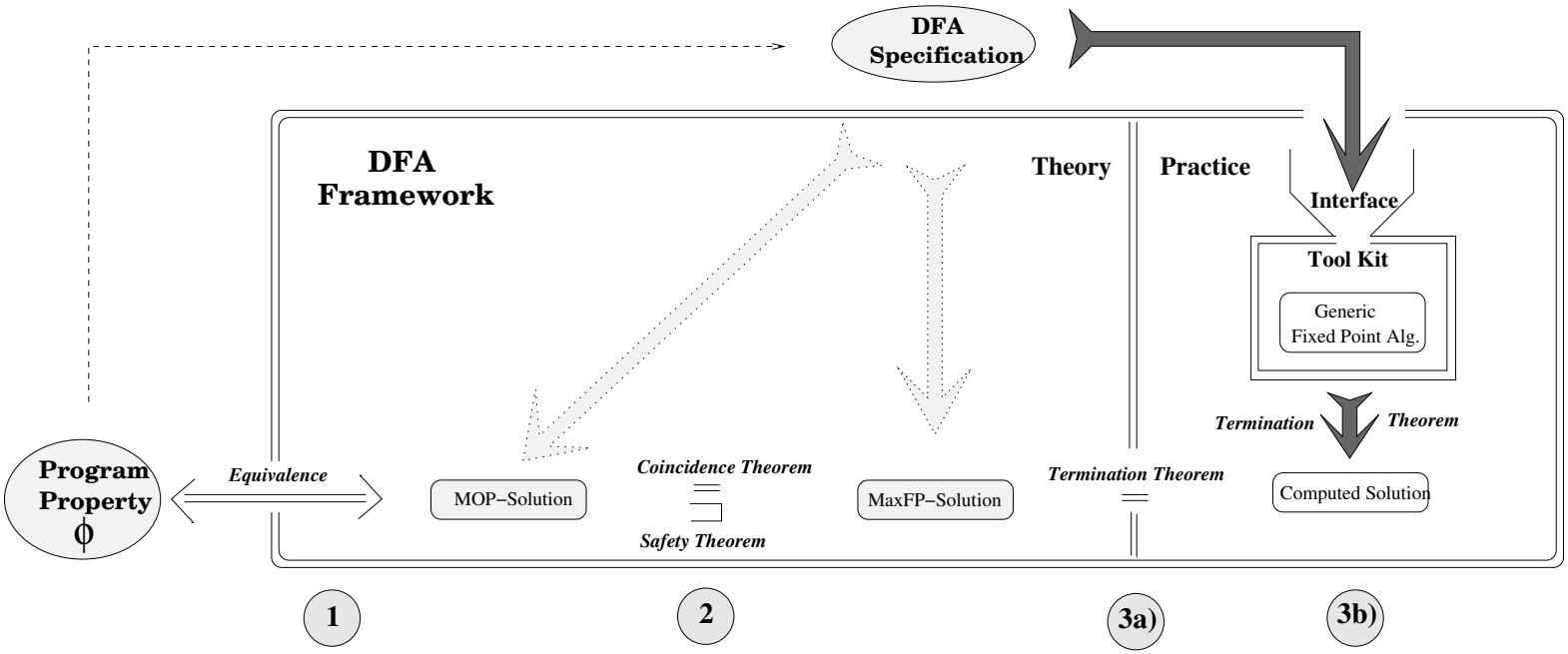
- Was bedeutet Optimalität
  ...in Analyse und in Optimierung?

Und auch (scheinbar) Nebensächliches:

- Was ist eine angemessene Programmrepräsentation?

Überblick – DFA in Theorie und Praxis (1)

DFA-Rahmen / DFA-Werkzeugkistensicht:



Überblick – DFA in Theorie und Praxis (2)

Komponenten einer (intraprozeduralen) DFA-Spezifikation:

- (Lokale) abstrakte Semantik
  1. Ein Datenflussanalyseverband $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$
  2. Ein Datenflussanalysefunktional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$
  3. Anfangsinformation/-zusicherung $c_{\mathbf{s}} \in \mathcal{C}$

Darauf aufbauend:

- Globalisierungsstrategien
  1. "Meet over all Paths"-Ansatz (*MOP*)
  2. Maximaler Fixpunktansatz (*MaxFP*)
- Generischer Fixpunktalgorithmus

Überblick – DFA in Theorie und Praxis (3)

Hauptresultate:

- Korrektheit: Sicherheits-Theorem
- Vollständigkeit: Koinzidenz-Theorem

Sowie:

- Effektivität: Terminierungs-Theorem

Überblick – DFA in Theorie und Praxis (4)

Genauer lassen sich unterscheiden:

- intraprozedurale,
- interprozedurale,
- objektorientierte,
- parallele,
- konditionale,
- ...

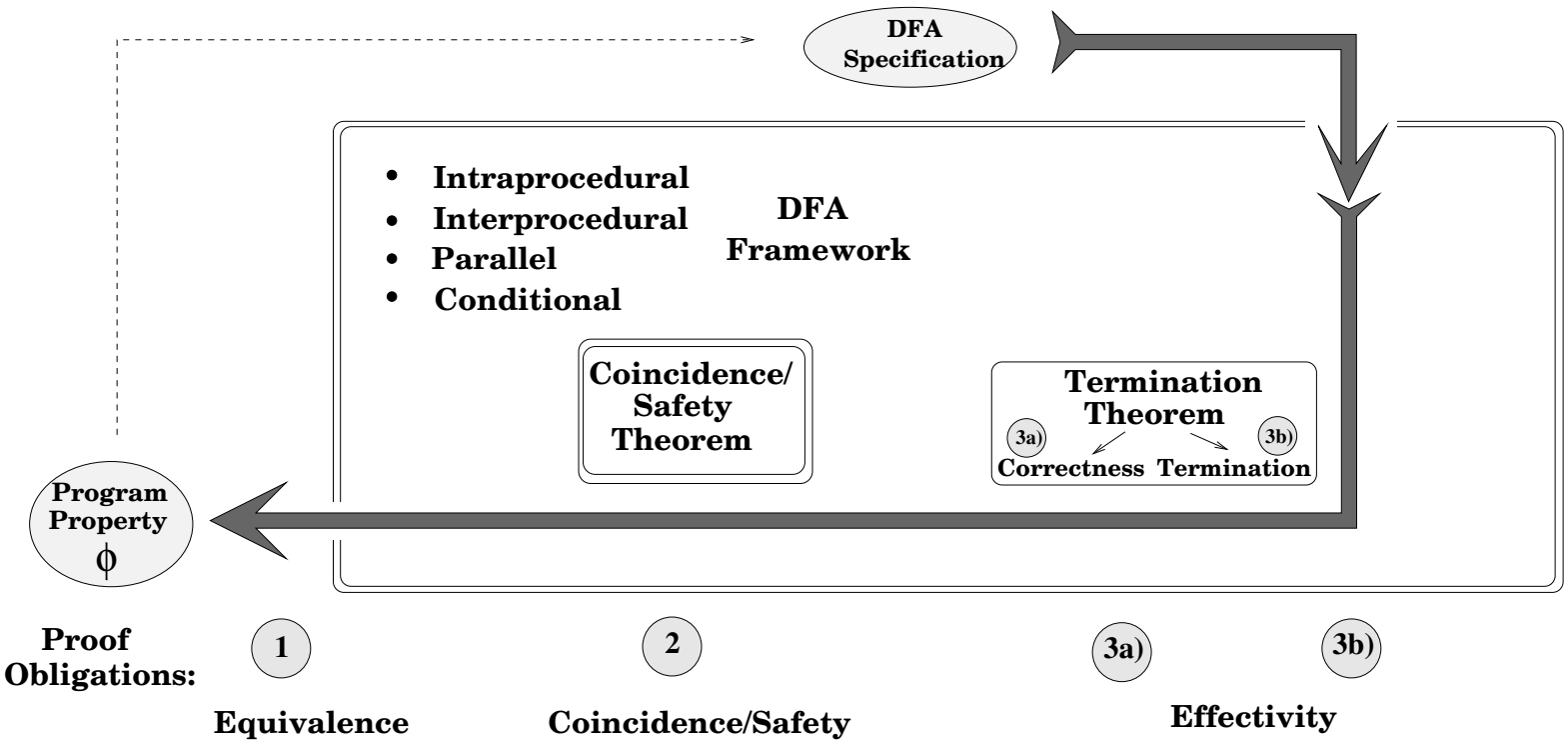
Datenflussanalyse (DFA).

Das grundlegende Prinzip...

Überblick – DFA in Theorie und Praxis (5)

...bleibt stets gleich!

Die allgemeine DFA-Rahmen / DFA-Werkzeugkistensicht:



Ziel

Optimale Programmoptimierung!

...ein weißer Schimmel in der Informatik?

Ohne Fleiß kein Preis!

In der Sprechweise optimierender Übersetzung:

...ohne Analyse keine Optimierung!

Kapitel 6.2

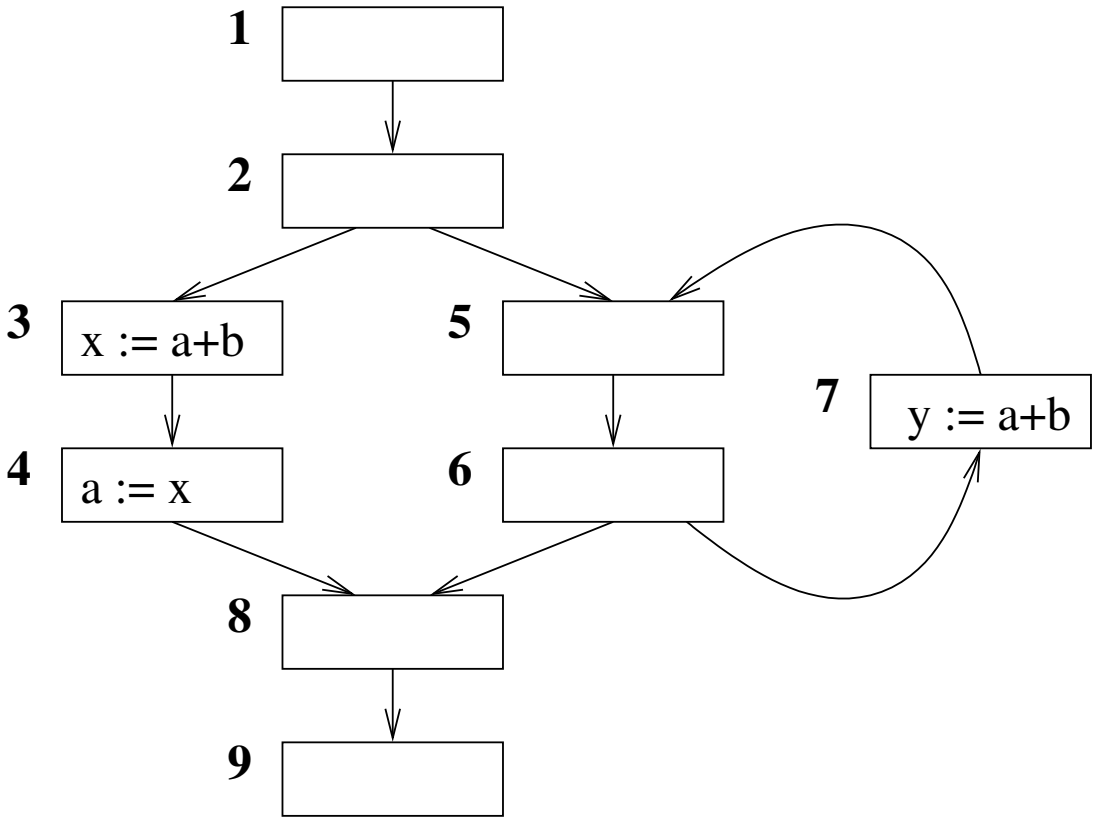
Datenflussanalyse

Programmrepräsentation

Im Bereich der Programmanalyse, speziell Datenflussanalyse, ist üblich:

- die Repräsentation von Programmen in Form (nichtdeterministischer) Flussgraphen

Beispiel



Flussgraphen (1)

Definition 6.2.1 (Flussgraph)

Ein (nichtdeterministischer) Flussgraph ist ein Quadrupel $G = (N, E, s, e)$ mit

- Knotenmenge (engl. Nodes) N
- Kantenmenge (engl. Edges) $E \subseteq N \times N$
- ausgezeichnetem Startknoten s
- ausgezeichnetem Endknoten e

Ohne Beschränkung der Allgemeinheit nehmen wir an, dass

- s keine Vorgänger (d.h. keine eingehenden Kanten) besitzt
- e keine Nachfolger (d.h. keine ausgehenden Kanten) besitzt
- alle Knoten aus N auf einem Pfad von s nach e liegen

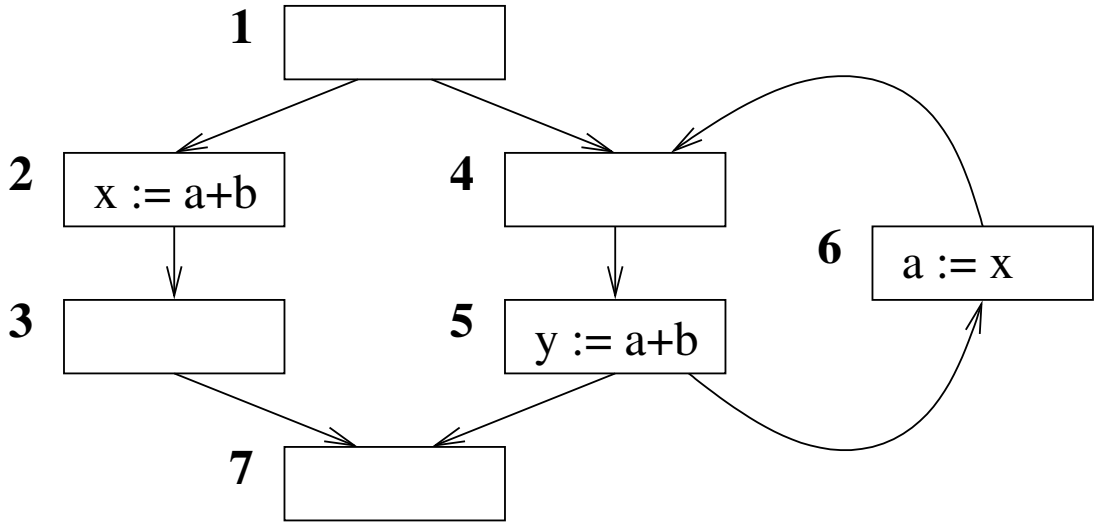
Flussgraphen (2)

Bemerkung:

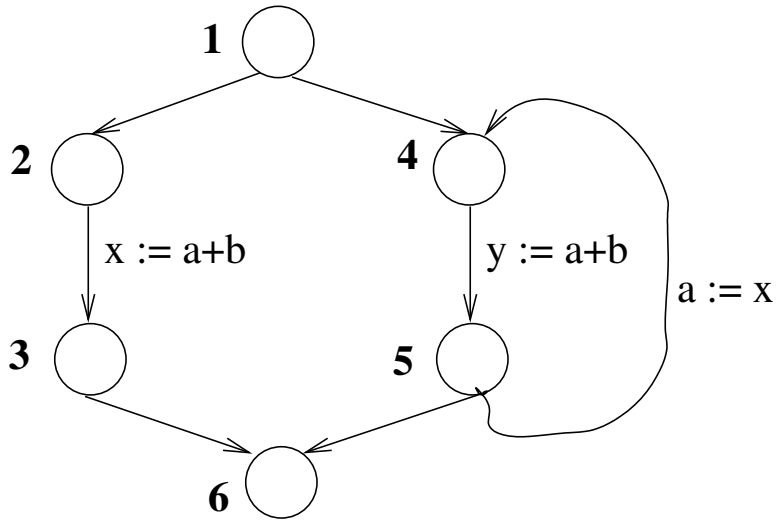
- Knoten repräsentieren Programmpunkte
- Kanten die Verzweigungsstruktur.
- Elementare Programmanweisungen (Zuweisungen, Tests) können wahlweise durch
  - Knoten ($\leadsto$ knotenbenannter Flussgraph)
  - Kanten ($\leadsto$ kantenbenannter Flussgraph)

  repräsentiert werden.

Beispiel: Knotenbenannter Flussgraph



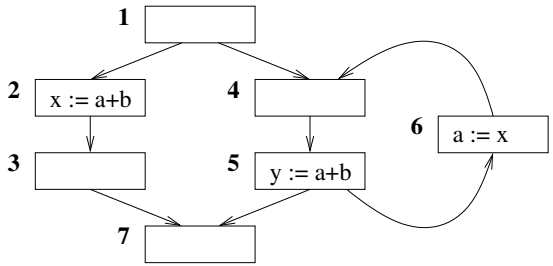
Beispiel: Kantenbenannter Flussgraph



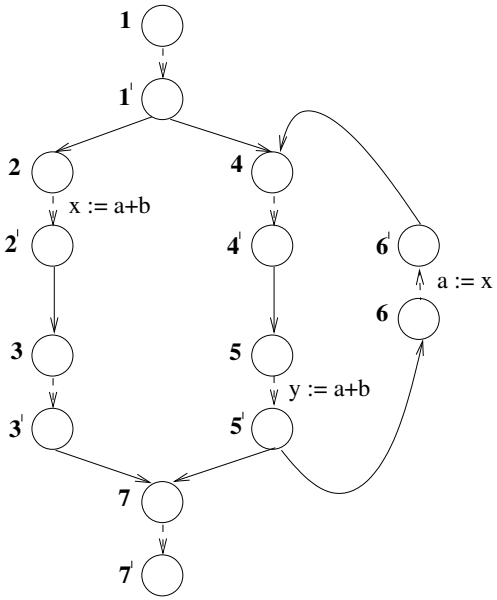
Flussgraphdarstellungsvarianten (1)

Knoten- vs. kantenbenannte Flussgraphen
(hier mit Einzelanweisungsbenennung)

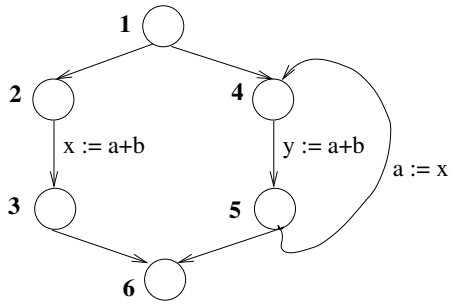
a)



b)



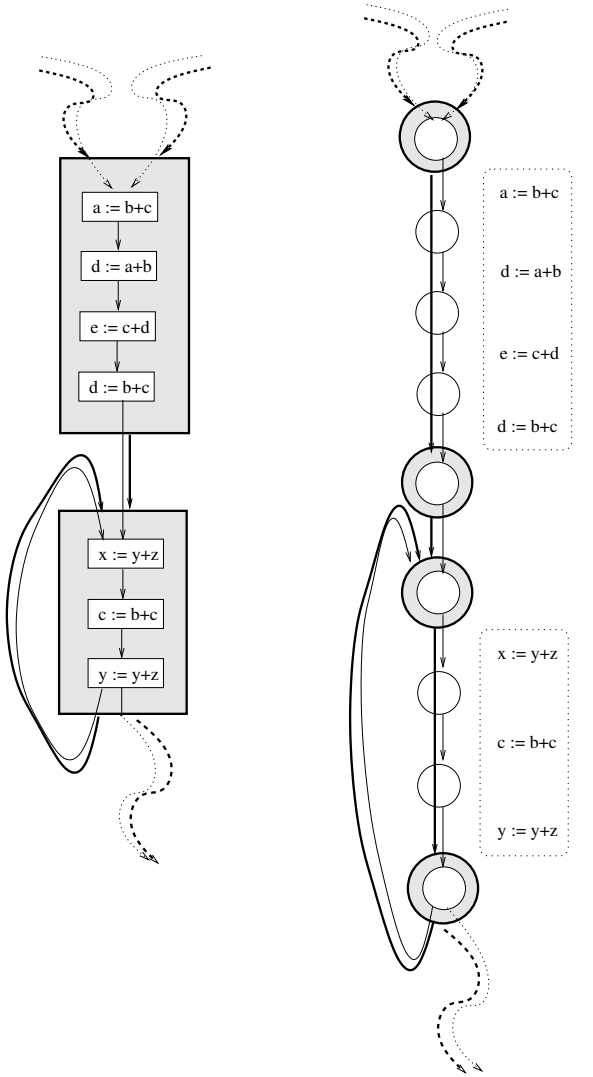
i) Schematisch



ii) "Optimiert"

Flussgraphdarstellungsvarianten (2)

Knoten- vs. kantenbenannte Flussgraphen
(hier mit Basisblockbenennung)



Node–labeled (BB–) Graph Edge–labeled (BB–) Grap

Flussgraphdarstellungsvarianten (3)

Wir unterscheiden:

- Knotenbenannte Graphen
  - Einzelanweisungsgraphen (SI-Graphen)
  - Basisblockgraphen (BB-Graphen)
- Kantenbenannte Graphen
  - Einzelanweisungsgraphen (SI-Graphen)
  - Basisblockgraphen (BB-Graphen)

In der Folge betrachten wir bevorzugt kantenbenannte SI-Graphen.

Bezeichnungen für Pfadmengen

Sei $G = (N, E, s, e)$ ein Flussgraph, seien m, n zwei Knoten aus N. Dann bezeichne:

- $\mathbf{P}_G[m, n]$: Die Menge aller Pfade von m nach n.
- $\mathbf{P}_G[m, n[$: Die Menge aller Pfade von m zu einem Vorgänger von n.
- $\mathbf{P}_G]m, n]$: Die Menge aller Pfade von einem Nachfolger von m nach n.
- $\mathbf{P}_G]m, n[$: Die Menge aller Pfade von einem Nachfolger von m zu einem Vorgänger von n.

Bemerkung: Wenn G aus dem Kontext eindeutig hervorgeht, schreiben wir einfacher auch $\mathbf{P}$ statt $\mathbf{P}_G$.

DFA-Spezifikation, DFA-Problem

Definition 6.2.2 (DFA-Spezifikation)

Eine DFA-Spezifikation ist festgelegt durch

- eine (lokale) abstrakte Semantik bestehend aus
  1. einem DFA-Verband $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$
  2. einem DFA-Funktional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$
- eine Anfangsinformation/-zusicherung: $c_{\mathbf{s}} \in \mathcal{C}$

Defintion 6.2.3 (DFA-Problem)

Eine DFA-Spezifikation legt ein DFA-Problem fest.

Praktisch relevant

...sind sog.

- monotone
- distributive
- additive

DFA-Probleme über DFA-Verbänden, die die

- absteigende
- aufsteigende Kettenbedingung

erfüllen.

Monotonie, Distributivität, Additivität

...von DFA-Funktionalen, DFA-Problemen:

Definition 6.2.4

Ein DFA-Funktional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ heißt monoton/distributiv/additiv gdw für alle $e \in E$ gilt, dass $[\![\, e \,]\!]$ monoton/distributiv/additiv ist.

Definition 6.2.5

Ein DFA-Problem heißt monoton/distributiv/additiv gdw das DFA-Funktional $[\![\ ]\!]$ der zugrundeliegenden DFA-Spezifikation $(\hat{\mathcal{C}}, [\![\ ]\!], c_s)$ ist monoton/distributiv/additiv.

Ausdehnung von Monotonie, Distributivität, Additivität (1)

...von Funktionen auf CPOs auf Funktionen auf (DFA-) Verbänden (vgl. Kapitel 3.3).

Definition 6.2.6 (Monotonie, Distributivität, Additivität)

Sei $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ ein vollständiger (DFA-) Verband und $f : \mathcal{C} \to \mathcal{C}$ eine Funktion auf $\mathcal{C}$. Dann heißt f

1. monoton gdw $\forall\, c, c' \in \mathcal{C}.\ c \sqsubseteq c' \Rightarrow f(c) \sqsubseteq f(c')$
   (Erhalt der Ordnung der Elemente)
2. distributiv gdw $\forall\, C' \subseteq \mathcal{C}.\ \ f(\bigsqcap C') = \bigsqcap \{f(c) \mid c \in C'\}$
   (Erhalt der größten unteren Schranken)
3. additiv gdw $\forall\, C' \subseteq \mathcal{C}.\ \ f(\bigsqcup C') = \bigsqcup \{f(c) \mid c \in C'\}$
   (Erhalt der kleinsten oberen Schranken)

Ausdehnung von Monotonie, Distributivität, Additivität (2)

Analog zum entsprechenden Lemma auf CPOs gilt auch auf (DFA-) Verbänden folgende oft nützliche äquivalente Charakterisierung der Monotonie (vgl. Kapitel 3.3):

Lemma 6.2.7

Sei $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ ein vollständiger (DFA-) Verband und $f : \mathcal{C} \to \mathcal{C}$ eine Funktion auf $\mathcal{C}$. Dann gilt:

$$f \text{ ist monoton} \iff \forall\, C' \subseteq \mathcal{C}.\ \ f(\bigsqcap C') \sqsubseteq \bigsqcap \{f(c) \mid c \in C'\}$$

Ausdehnung von absteigender/aufsteigender Kettenbedingung

...von CPOs auf (DFA-) Verbände (vgl. Kapitel 3.3):

Definition 6.2.8 (Ab-/aufsteigende Kettenbeding'g)

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

1. die absteigende Kettenbedingung, falls jede absteigende Kette stationär wird, d.h. für jede Kette $c_1 \sqsupseteq c_2 \sqsupseteq \ldots \sqsupseteq c_n \sqsupseteq \ldots$ gibt es einen Index $m \geq 1$ so dass $c_m = c_{m+j}$ für alle $j \in \mathbb{N}$ gilt
2. die aufsteigende Kettenbedingung, falls jede aufsteigende Kette stationär wird, d.h. für jede Kette $c_1 \sqsubseteq c_2 \sqsubseteq \ldots \sqsubseteq c_n \sqsubseteq \ldots$ gibt es einen Index $m \geq 1$ so dass $c_m = c_{m+j}$ für alle $j \in \mathbb{N}$ gilt

Nächstes Ziel

...die Globalisierung lokaler abstrakter Semantiken von Anweisungen auf Flussgraphen.

Dafür zwei (Globalisierungs-) Strategien:

- "Meet over all Paths"-Ansatz (*MOP*)
  $\rightsquigarrow$ führt auf spezifizierende Lösung eines DFA-Problems.
- Maximaler Fixpunktansatz (*MaxFP*)
  $\rightsquigarrow$ führt auf berechenbare Lösung eines DFA-Problems.

Kapitel 6.3

MOP-Ansatz

Der *MOP*-Ansatz

Grundlegend für die *MOP*-Strategie:

Definition 6.3.1 (Pfadausdehnung von $[\![\ ]\!]$)

Die Ausdehnung einer lokalen abstrakten Semantik $[\![\ ]\!]$ auf Pfade $p = \langle e_1, e_2 \ldots, e_q \rangle$ ist definiert durch

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

wobei $Id_{\mathcal{C}}$ die Identität auf $\mathcal{C}$ bezeichnet.

Die *MOP*-Lösung

Definition 6.3.2 (*MOP*-Lösung)

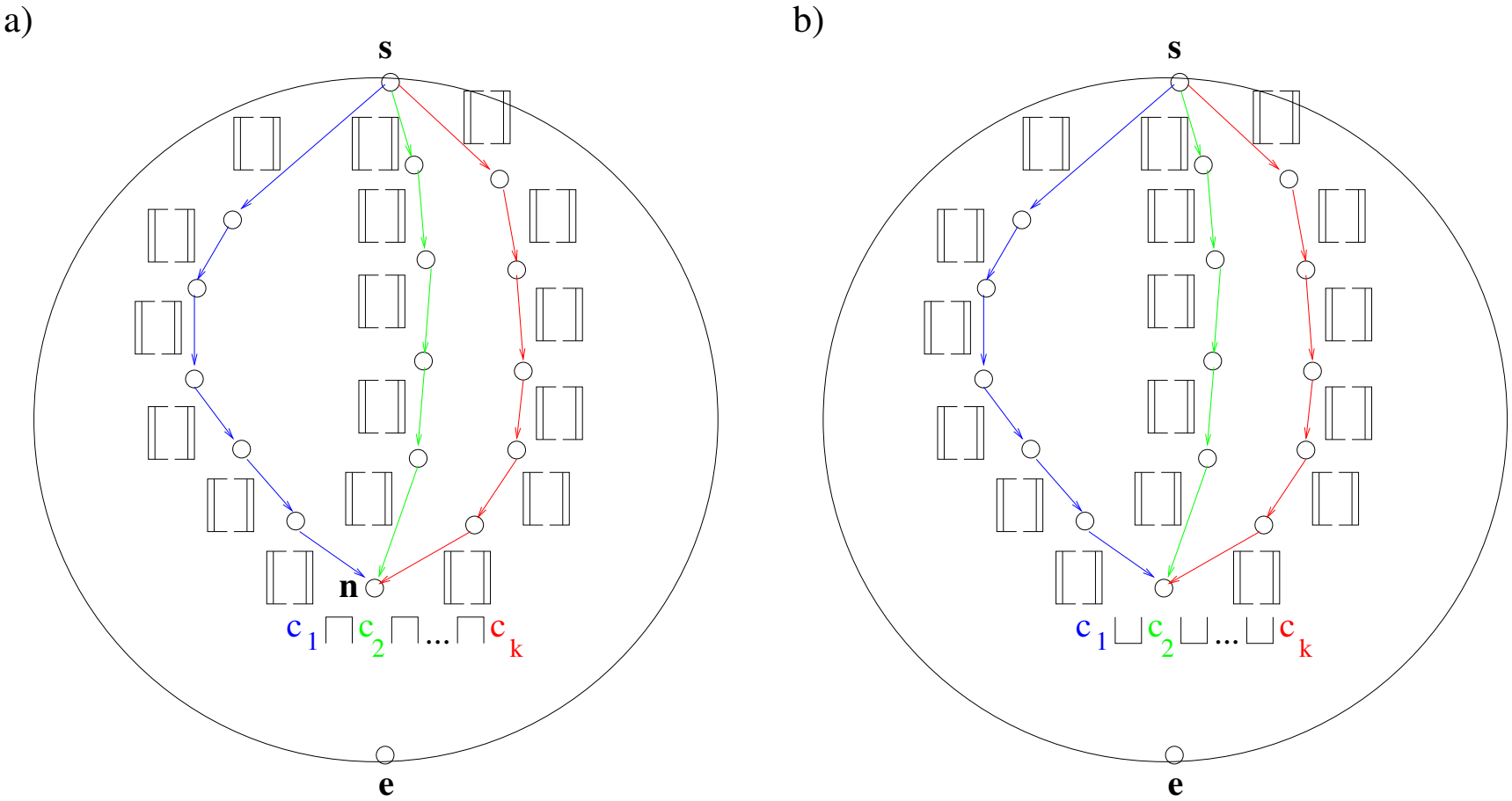
Sei $(\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ die Spezifikation eines DFA-Problems. Dann ist für alle Knoten $n \in N$ die *MOP*-Lösung definiert durch:

$$MOP_{(\widehat{\mathcal{C}},[\![\ ]\!],c_{\mathbf{s}})}(n) = \bigsqcap \{ [\![p]\!](c_{\mathbf{s}}) \mid p \in \mathbf{P}[\mathbf{s}, n] \}$$

Zentral:

- Die *MOP*-Lösung ist das Maß aller DFA-Dinge. Sie ist die spezifizierende Lösung eines durch $(\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ gegebenen intraprozeduralen DFA-Problems.

Veranschaulichung



a) Traditionelle DFA-Sicht:
'Schneiden' v. Infos: *MOP*
(Allquantifizierung)

b) Traditionelle AI-Sicht:
'Vereinigen' v. Infos: *JOP*
(Ex. Quantifizierung)

Wermutstropfen

...ist die Unentscheidbarkeit der *MOP*-Lösung im allgemeinen:

Theorem 6.3.3 (Unentscheidbarkeit)

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

Es gibt keinen Algorithmus A mit folgenden Eigenschaften:

1. Eingabe für A sind
   - 1.1 Algorithmen zur Berechnung von Schnitt, Gleichheitstest und Anwendung von Funktionen auf Verbandselemente eines monotonen Datenflussanalyserahmens
   - 1.2 eine durch $(\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ gegebene Instanz I dieses Rahmens
2. Ausgabe von A ist die *MOP*-Lösung von I.

Deshalb betrachten wir eine zweite Globalisierungsstrategie.

Kapitel 6.4

MaxFP-Ansatz

Der *MaxFP*-Ansatz

Grundlegend für die *MaxFP*-Strategie:

Definition 6.4.1 (*MaxFP*-Gleichungssystem)

Das *MaxFP*-Gleichungssystem ist gegeben durch:

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

Die *MaxFP*-Lösung

Definition 6.4.2 (*MaxFP*-Lösung)

Sei $(\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ die Spezifikation eines DFA-Problems. Dann ist für alle Knoten $n \in N$ die *MaxFP*-Lösung definiert durch:

$$MaxFP_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)}(n) =_{df} inf^*_{c_s}(n)$$

wobei $inf^*_{c_s}$ die größte Lösung des *MaxFP*-Gleichungssystems 6.4.1 bezüglich $(\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ bezeichnet.

Zentral:

- Die *MaxFP*-Lösung ist die (unter geeigneten Voraussetzungen, siehe Terminierungstheorem 6.6.2) effektiv berechenbare Lösung eines durch $(\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ gegebenen intraprozeduralen DFA-Problems.

Kapitel 6.5

Koinzidenz- und Sicherheitstheorem

Hauptresultate: Korrektheit u. Vollständigkeit

Grundsätzlich:

- Zusammenhang von *MOP*-Lösung und *MaxFP*-Lösung

Genauer:

- Korrektheit und Vollständigkeit der *MaxFP*-Lösung bzgl. der *MOP*-Lösung

Im Detail:

- Korrektheitsfrage:
  Gilt stets $\mathit{MaxFP}_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)} \sqsubseteq \mathit{MOP}_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)}$?
- Vollständigkeitsfrage:
  Gilt stets $\mathit{MaxFP}_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)} \sqsupseteq \mathit{MOP}_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)}$?

Korrektheit

Theorem 6.5.1 (Sicherheit)

Die *MaxFP*-Lösung ist eine sichere (konservative), d.h. untere Approximation der *MOP*-Lösung, d.h.,

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{(\widehat{c},[\![\ ]\!],c_s)}(n) \sqsubseteq MOP_{(\widehat{c},[\![\ ]\!],c_s)}(n)$$

falls das DFA-Funktional $[\![\ ]\!]$ monoton ist.

Vollständigkeit (und zugleich Korrektheit)

Theorem 6.5.2 (Koinzidenz)

Die *MaxFP*-Lösung stimmt mit der *MOP*-Lösung überein, d.h.,

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{(\widehat{c},[\![\ ]\!],c_s)}(n) = MOP_{(\widehat{c},[\![\ ]\!],c_s)}(n)$$

falls das DFA-Funktional $[\![\ ]\!]$ distributiv ist.

Bemerkung

Statt von

- Korrektheit
- Vollständigkeit

spricht man im Zusammenhang mit DFA traditionell von

- Sicherheit
- Koinzidenz

Kapitel 6.6

Generischer Fixpunktalgorithmus und Terminierungstheorem

Hauptresultat: Effektivität

Grundsätzlich:

- Zusammenhang von *MaxFP*-Lösung und Generischem Fixpunktalgorithmus 6.6.1

Genauer:

- Effektivität des Generischen Fixpunktalgorithmus 6.6.1 für die *MaxFP*-Lösung

Im Detail:

- Effektivitätsfrage:
  Terminiert der Generische Fixpunktalgorithmus 6.6.1 stets mit der $MaxFP_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)}$-Lösung?

Generischer Fixpunktalgorithmus 6.6.1 (1)

Eingabe: (1) Ein Flussgraph $G = (N, E, \mathbf{s}, \mathbf{e})$, (2) ein DFA-Problem gegeben durch eine DFA-Spezifikation mit (lokaler) abstrakter Semantik bestehend aus einem DFA-Verband $\widehat{\mathcal{C}}$, einem DFA-Funktional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$, und (3) einer Anfangsinformation $c_{\mathbf{s}} \in \mathcal{C}$.

Ausgabe: Unter den Voraussetzungen des Terminierungstheorems 6.6.2 die $MaxFP_{(\widehat{\mathcal{C}},[\![\ ]\!],c_{\mathbf{s}})}$-Lösung. Abhängig von den Eigenschaften des DFA-Funktionals gilt dann:

(i) $[\![\ ]\!]$ ist distributiv: Variable *inf* enthält für jeden Knoten die stärkste Nachbedingung bezüglich der Anfangsinformation $c_{\mathbf{s}}$.

(ii) $[\![\ ]\!]$ ist monoton: Variable *inf* enthält für jeden Knoten eine sichere (d.h. untere) Approximation der stärksten Nachbedingung bezüglich der Anfangsinformation $c_{\mathbf{s}}$.

Generischer Fixpunktalgorithmus 6.6.1 (2)

Bemerkung:

- Die stärkste Nachbedingung ist durch die $MOP_{(\widehat{C},[\![\ ]\!],c_S)}$-Lösung gegeben.
- Die Variable *workset* steuert den iterativen Prozess. Ihre Elemente sind Knoten aus G, deren Annotation jüngst aktualisiert worden ist, was möglicherweise zu einer verbandsmäßig kleineren Annotation an ihren Nachfolgerknoten führt.

Generischer Fixpunktalgorithmus 6.6.1 (3)

(Prolog: Initialisierung von *inf* und *workset*)
FORALL $n \in N \backslash \{\mathbf{s}\}$ DO $inf[n] := \top$ OD;
$inf[\mathbf{s}] := c_{\mathbf{s}}$;
$workset := N$;
(Hauptprozess: Iterative Fixpunktberechnung)
WHILE $workset \neq \emptyset$ DO
    CHOOSE $m \in workset$;
        $workset := workset \backslash \{ m \}$;
        (Aktualisiere die Nachfolgerumgebung von Knoten 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.

Effektivität

Theorem 6.6.2 (Terminierung)

Der Generische Fixpunktalgorithmus 6.6.1 terminiert mit der $MaxFP_{(\widehat{\mathcal{C}},[\![\ ]\!],c_s)}$-Lösung, falls

a) das DFA-Funktional $[\![\ ]\!]$ monoton ist

b) der DFA-Verband $\widehat{\mathcal{C}}$ die absteigende Kettenbedingung erfüllt.

Bemerkung

Der Generische Fixpunktalgorithmus 6.6.1 ist formuliert für

- allquantifizierte ("distributive") Vorwärtsprobleme

Die anderen drei DFA-Problemvarietäten

- existenziell quantifizierter ("additiver") Probleme
- Rückwärtsprobleme

können mithilfe des Generischen Fixpunktalgorithmus 6.6.1 gelöst werden, indem

- der Verband (durch Vertauschen von $\sqsubset$ mit $\sqsupset$)
- der Flussgraph (durch Umdrehen aller Kanten)

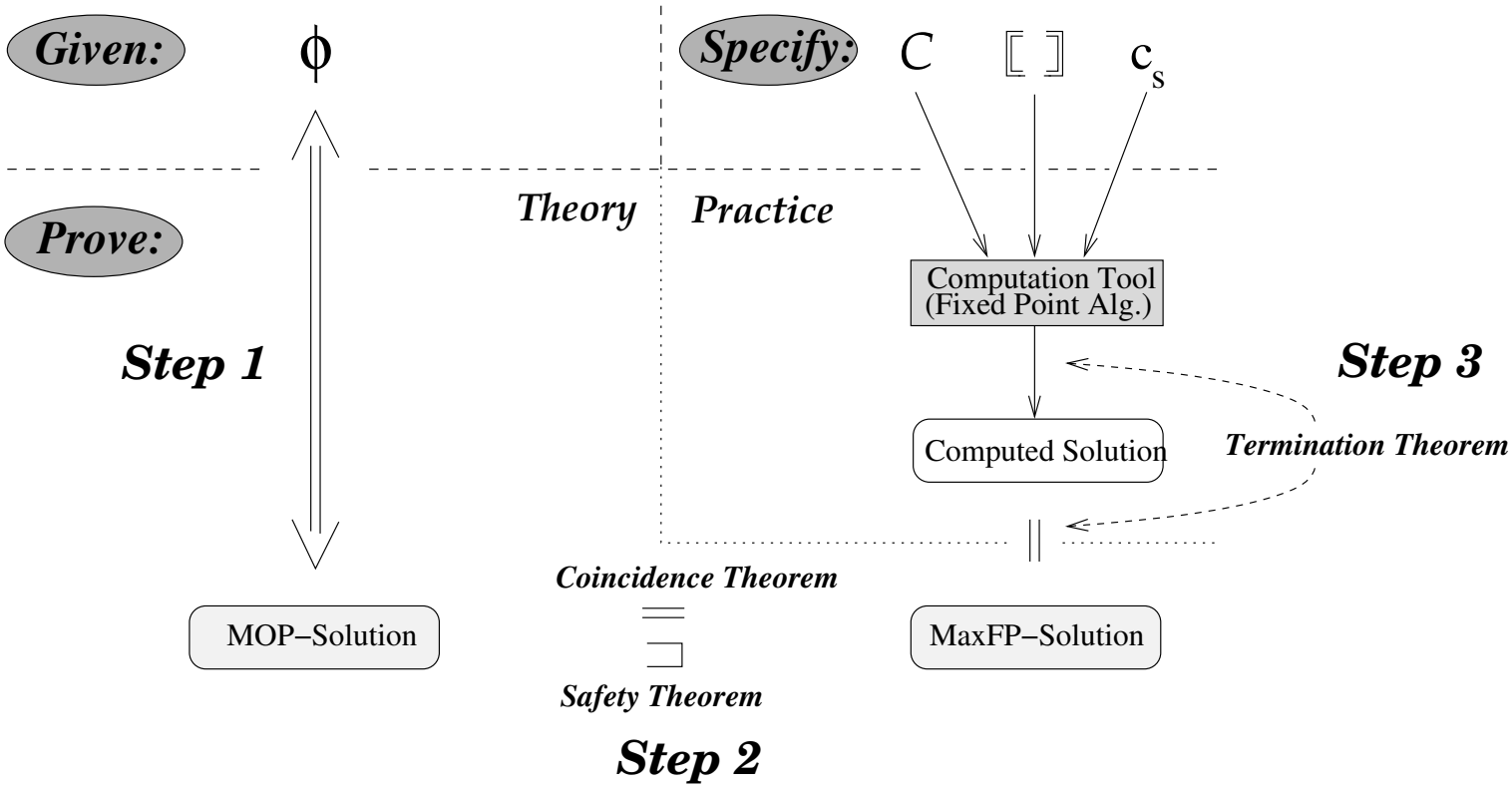
auf "den Kopf gestellt" wird.

Kapitel 6.7

Zusammenfassung und Überblick

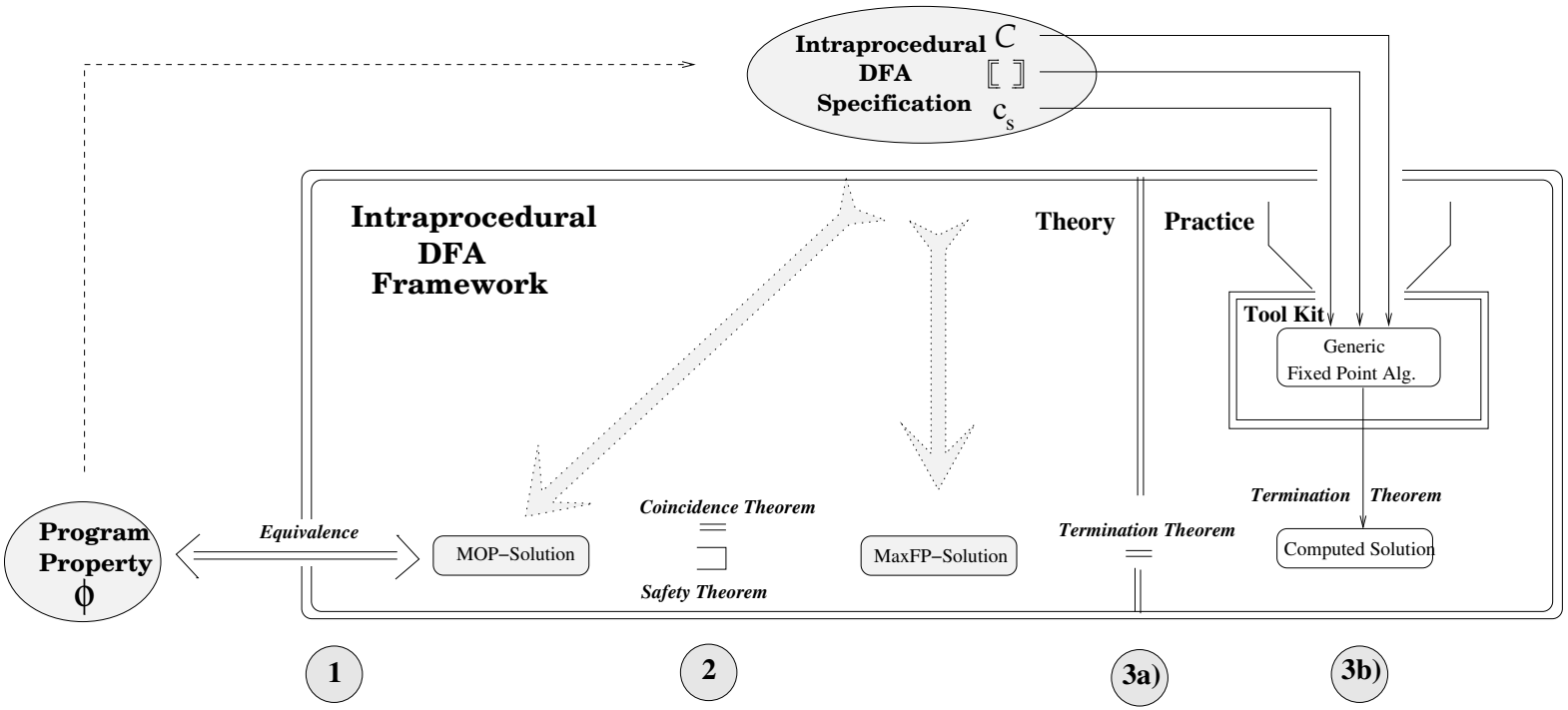
Intraprozedurale DFA im Überblick (1)

Ein Bild sagt mehr als 1000 Worte:



Intraprozedurale DFA im Überblick (2)

Fokussiert auf die Rahmen-/Werkzeugkistensicht:



Kapitel 6.8

Beispiele: Verfügbare Ausdrücke, einfache Konstanten

Zwei prototypische DFA-Probleme

- Verfügbare Ausdrücke
  $\rightsquigarrow$ kanonisches Bsp. eines distributiven DFA-Problems
- Einfache Konstanten
  $\rightsquigarrow$ kanonisches Bsp. eines monotonen DFA-Problems

Kapitel 6.8.1

Verfügbare Ausdrücke

Verfügbare Ausdrücke (1)

...ein typisches distributives DFA-Problem.

Informell:

- Ein Ausdruck t ist verfügbar an einem Knoten n eines Flussgraphen, wenn t auf allen Pfaden p von **s** nach n berechnet wird und nach der letzten Berechnung von t auf p kein Operand von t mehr verändert wird (d.h. nicht mehr linksseitig in einer Zuweisung vorkommt).

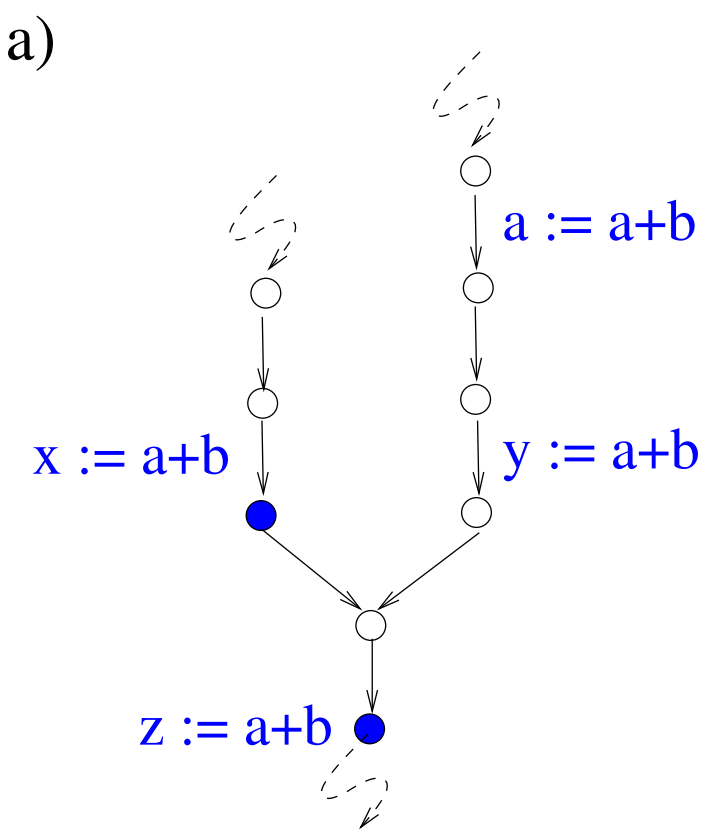
Verfügbare Ausdrücke (2)

Bemerkung:

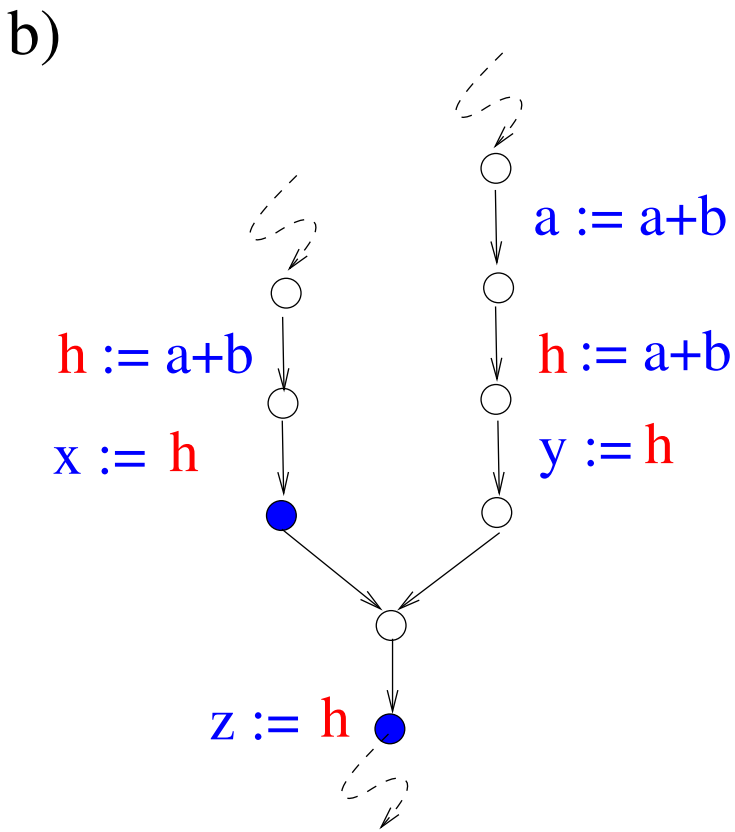
- t wird oft als Kandidatenausdruck bezeichnet.
- Ist t verfügbar an einem Knoten n, so kann eine Wiederberechnung an n durch Rückgriff auf seinen zuvor berechneten (und zu diesem Zweck gespeicherten) Wert vermieden werden; dadurch entsteht bestimmter Berechnungsaufwand zur Laufzeit überhaupt nicht, er wird vermieden: Performanzverbesserung $\leadsto$ Optimierung

Veranschaulichung

a) Vor der Optimierung



b) Nach der Optimierung



Die Optimierung wird bezeichnet als

- Elimination total redundanter Berechnungen

Spezifikation als DFA-Problem

- (Lokale) abstrakte Semantik für verfügbare Ausdrücke:
  1. DFA-Verband:
     $$(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathbb{B}, \wedge, \vee, \leq, \mathbf{falsch}, \mathbf{wahr})$$
  2. DFA-Funktional: $[\![\ ]\!]_{av} : E \to (\mathbb{B} \to \mathbb{B})$ definiert durch
     $$\forall\, e \in E.\ [\![e]\!]_{av} =_{df} \begin{cases} Cst_{\mathbf{wahr}} & \text{falls } Comp_e \wedge Transp_e \\ Id_{\mathbb{B}} & \text{falls } \neg Comp_e \wedge Transp_e \\ Cst_{\mathbf{falsch}} & \text{sonst} \end{cases}$$
- Anfangsinformation: $b_s \in \mathbb{B}$

Bezeichnungen (1)

Dabei bezeichnen:

- $\hat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, <, \mathbf{falsch}, \mathbf{wahr})$: Verband der Wahrheitswerte mit $\bot = \mathbf{falsch} < \mathbf{wahr} = \top$ und dem logischen "und" und "oder" als Schnitt- bzw. Vereinigungsoperation $\sqcap$ und $\sqcup$.
- $Cst_{\mathbf{wahr}}$, $Cst_{\mathbf{falsch}}$: die konstanten Funktionen "wahr" und "falsch" auf $\mathbb{B}$
- $Id_{\mathbb{B}}$: die Identität auf $\mathbb{B}$.

Bezeichnungen (2)

...sowie relativ zu einem fest gewählten Kandidatenausdruck t:

- $Comp_e$: t wird von der Anweisung an Kante e berechnet (d.h. t kommt rechtsseitig als (Teil-) Ausdruck vor)
- $Transp_e$: kein Operand von t erhält durch die Anweisung an Kante e einen neuen Wert (d.h. kein Operand von t kommt linksseitig vor: e ist transparent für t)

Hauptergebnisse

Lemma 6.8.1.1 (Distributivität u. Kettenbedingung)

1. $[\![\ ]\!]_{av}$ ist distributiv.
2. $\hat{\mathbb{B}}$ erfüllt die absteigende Kettenbedingung.

Korollar 6.8.1.2 (Koinzidenz und Terminierung)

Für das DFA-Problem verfügbarer Ausdrücke

1. stimmen die *MOP*-Lösung und die *MaxFP*-Lösung überein, d.h. die *MaxFP*-Lösung ist korrekt und vollständig bezgl. der *MOP*-Lösung.
2. terminiert der Generische Fixpunktalgorithmus 6.6.1 mit der *MaxFP*-Lösung.

Kapitel 6.8.2

Einfache Konstanten

Einfache Konstanten

...ein typisches monotones (nicht distributives) DFA-Problem.

Informell:

- Ein Ausdruck t ist einfache Konstante an einem Knoten n eines Flussgraphen, wenn jeder Operand von t auf allen Pfaden p von **s** nach n vom selbem konstanten Wert ist.

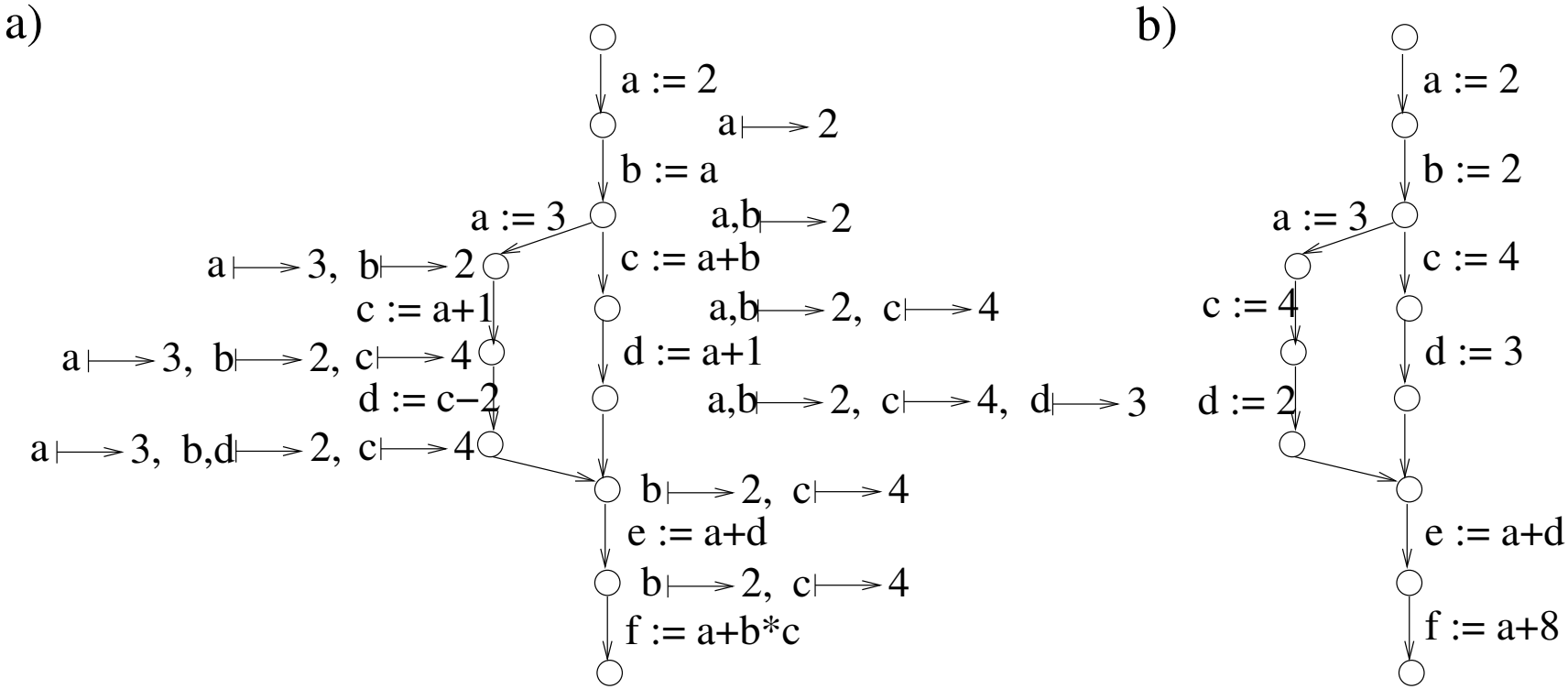
Bemerkung:

- Ist t eine einfache Konstante am Knoten n, so kann der Wert von t schon zur Übersetzungszeit berechnet werden und somit Berechnungsaufwand von der Laufzeit in die Übersetzungszeit eines Programms verschoben werden: Performanzverbesserung $\rightsquigarrow$ Optimierung

Veranschaulichung

a) Vor der Optimierung

b) Nach der Optimierung



Die Optimierung wird bezeichnet als

- Konstantenausbreitung und Faltung

Spezifikation als DFA-Problem

- (Lokale) abstrakte Semantik für einfache Konstanten:
  1. DFA-Verband:

$$(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\Sigma, \sqcap, \sqcup, \sqsubseteq, \sigma_\bot, \sigma_\top)$$

  2. DFA-Funktional:

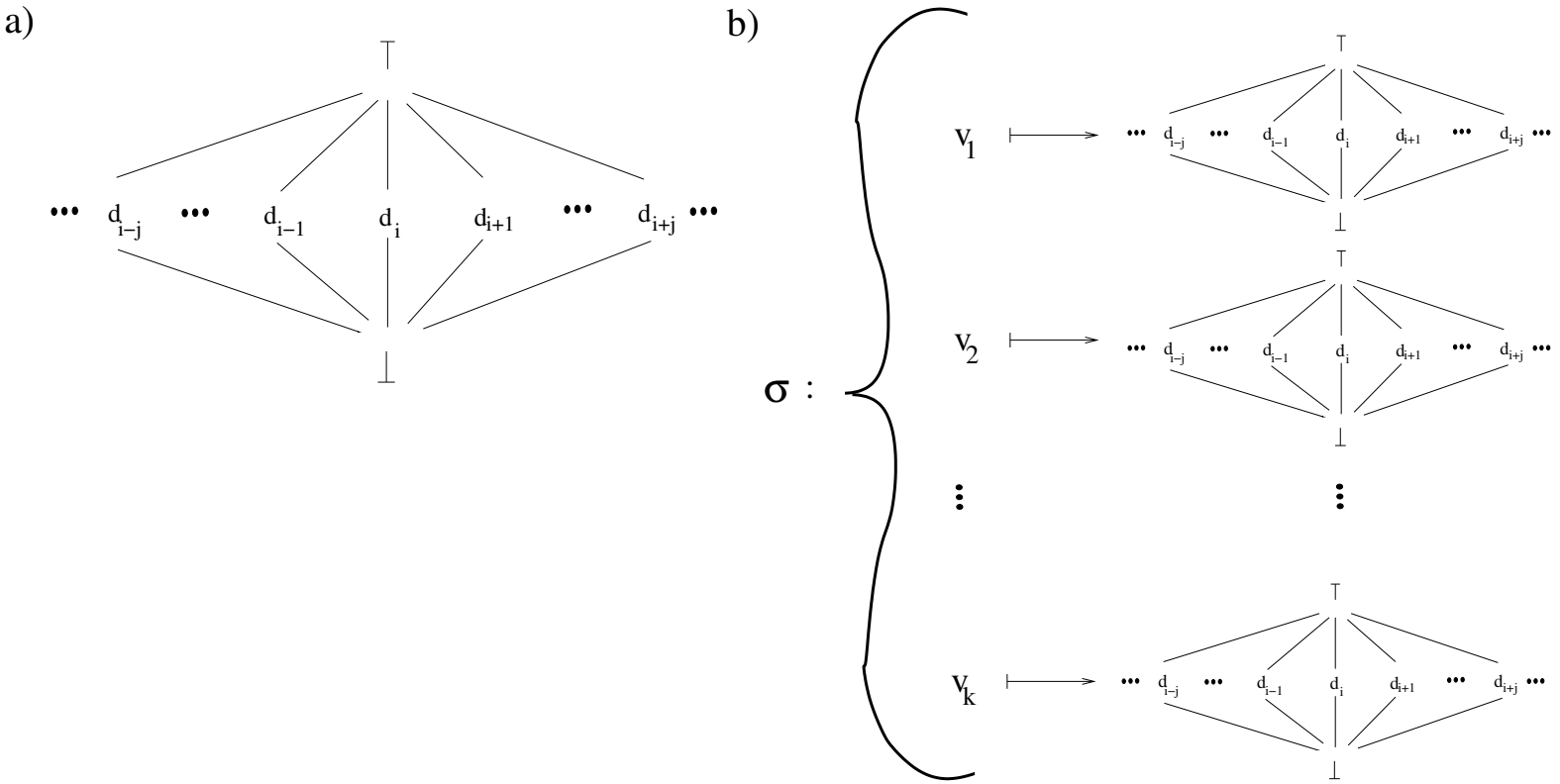
$$[\![\ ]\!]_{sc} : E \to (\Sigma \to \Sigma) \text{ definiert durch}$$

$$\forall e \in E.\ [\![e]\!]_{sc} =_{df} \theta_e$$

- Anfangsinformation: $\sigma_s \in \Sigma$

DFA-Verband für einfache Konstanten

Der “kanonische” Verband f. Konstantenausbreitung/Faltung:



Zusätzlich nötige Begriffe und Definitionen

...um die Definition der lokalen abstrakten Semantik für das DFA-Problem einfacher Konstanten abzuschließen:

- Termsyntax
- Interpretation
- Zustand
- Termsemantik
- Zustandstransformationsfunktion

Termsyntax (1)

Variablen, Konstanten, Operatoren

Sei

- **V** eine Menge von Variablen
- **Op** eine Menge von n-stelligen Operatoren, $n \geq 0$, sowie $\mathbf{C} \subseteq \mathbf{Op}$ die Menge der 0-stelligen Operatoren aus **Op**, die sog. Konstanten.

Termsyntax (2)

Zusammengesetzte Terme

Definition 6.8.2.1 (Termsyntax)

Wir legen fest:

1. Jede Variable $v \in \mathbf{V}$ und jede Konstante $c \in \mathbf{C}$ ist ein Term.
2. Ist $op \in \mathbf{Op}$ ein n-stelliger Operator, $n \geq 1$, und sind $t_1, \ldots, t_n$ Terme, dann ist auch $op(t_1, \ldots, t_n)$ ein Term.
3. Es gibt keine weiteren Terme außer den nach den obigen beiden Regeln konstruierbaren.

Die Menge aller Terme bezeichnen wir mit $\mathbf{T}$.

Interpretation

Definition 6.8.2.2 (Interpretation)

Sei ID' ein geeigneter Datenbereich (z.B. die Menge der ganzen Zahlen), seien $\bot$ und $\top$ zwei ausgezeichnete Elemente mit $\bot, \top \notin \mathrm{ID}'$ und sei $\mathrm{ID} =_{df} \mathrm{ID}' \cup \{\bot, \top\}$.

Eine Interpretation über $\mathbf{T}$ und ID ist ein Paar $I \equiv (\mathrm{ID}, I_0)$, wobei

- I_0 eine Funktion ist, die mit jedem 0-stelligen Operator $c \in \mathbf{Op}$ ein Datum $I_0(c) \in \mathrm{ID}'$ und mit jedem n-stelligen Operator $op \in \mathbf{Op}$, $n \geq 1$, eine totale Funktion $I_0(op) : \mathrm{ID}^n \to \mathrm{ID}$ assoziiert, die als strikt angenommen wird (d.h. $I_0(op)(d_1, \ldots, d_n) = \bot$, wann immer es ein $j \in \{1, \ldots, n\}$ gibt mit $d_j = \bot$)

Zustand, Zustandsmenge, undef. Zustand

Definition 6.8.2.3 (Zustand, Zustandsmenge)

- Ein Zustand $\sigma : \mathbf{V} \to \mathrm{ID}$ ist eine Abbildung von der Menge der Programmvariablen $\mathbf{V}$ auf den Datenbereich ID.
- Mit
$$\Sigma =_{df} \{ \sigma \mid \sigma : \mathbf{V} \to \mathrm{ID} \}$$
bezeichnen wir die Menge aller Zustände.
- Mit $\sigma_\perp$ bezeichnen wir den (total) undefinierten Zustand aus Σ, für den gilt: $\forall\, v \in \mathbf{V}.\ \sigma_\perp(v) = \perp$

Termsemantik

Definition 6.8.2.4 (Termsemantik)

Die Semantik von Termen $t \in \mathbf{T}$ ist durch die induktiv definierte Evaluationsfunktion $\mathcal{E}$ gegeben:

$$\mathcal{E} : \mathbf{T} \to (\Sigma \to \mathsf{ID}) \quad \text{definiert durch}$$

$$\forall t \in \mathbf{T}\, \forall \sigma \in \Sigma.\, \mathcal{E}(t)(\sigma) \models_{df} \begin{cases} \sigma(x) & \text{falls } t \equiv x \in \mathbf{V} \\ I_0(c) & \text{falls } t \equiv c \in \mathbf{C} \\ I_0(op)(\mathcal{E}(t_1)(\sigma), \ldots, \mathcal{E}(t_r)(\sigma)) & \\ \qquad \text{falls } t \equiv op(t_1, \ldots, t_r) & \end{cases}$$

Zustandstransformationsfunktion

Definition 6.8.2.5 (Zustandstransformationsfkt.)

Die Zustandstransformationsfunktion θ_ι

$$\theta_\iota : \Sigma \to \Sigma, \quad \iota \equiv x := t$$

ist definiert durch:

$$\forall\, \sigma \in \Sigma\ \forall\, y \in \mathbf{V}.\ \theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y \equiv x \\ \sigma(y) & \text{sonst} \end{cases}$$

Hauptergebnisse

Lemma 6.8.2.6 (Monotonie und Kettenbedingung)

1. $[\![\ ]\!]_{sc}$ ist monoton.
2. $\hat{\Sigma}$ erfüllt die absteigende Kettenbedingung.

Beachte: Distributivität von $[\![\ ]\!]_{sc}$ ist i.a. verletzt!

Korollar 6.8.2.7 (Sicherheit und Terminierung)

Für das DFA-Problem einfacher Konstanten

1. stimmen die *MOP*-Lösung und die *MaxFP*-Lösung i.a. nicht überein; die *MaxFP*-Lösung ist aber stets eine sichere, d.h. untere Approximation der *MOP*-Lösung.
2. terminiert der Generische Fixpunktalgorithmus 6.6.1 mit der *MaxFP*-Lösung.

Kapitel 6.9

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (1)

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley, 2nd edition, 2007. (Chapter 1, Introduction; Chapter 9.2, Introduction to Data-Flow Analysis; Chapter 9.3, Foundations of Data-Flow Analysis)

Randy Allen, Ken Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufman Publishers, 2002. (Chapter 4.4, Data Flow Analysis)

Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004. (Chapter 1, Overview of Compilation; Chapter 8, Introduction to Code Optimization; Chapter 9, Data Flow Analysis)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (2)


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


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 6 (3)

Jens Knoop. *From DFA-frameworks to DFA-generators: A unifying multiparadigm approach*. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Springer-V., LNCS 1579, 360-374, 1999.

Jens Knoop, 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, 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 6 (4)

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)

Thomas J. Marlowe, Barbara G. Ryder. *Properties of Data Flow Frameworks*. Acta Informatica 20:121-163, 1990.

Robert Morgan. *Building an Optimizing Compiler*. Digital Press, 1998.

Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997. (Chapter 1, Introduction to Advanced Topics; Chapter 4, Intermediate Representations; Chapter 7, Control-Flow Analysis; Chapter 8, Data Flow Analysis; Chapter 11, Introduction to Optimization; Chapter 12, Early Optimizations)

Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (5)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 5, Static Program Analysis)

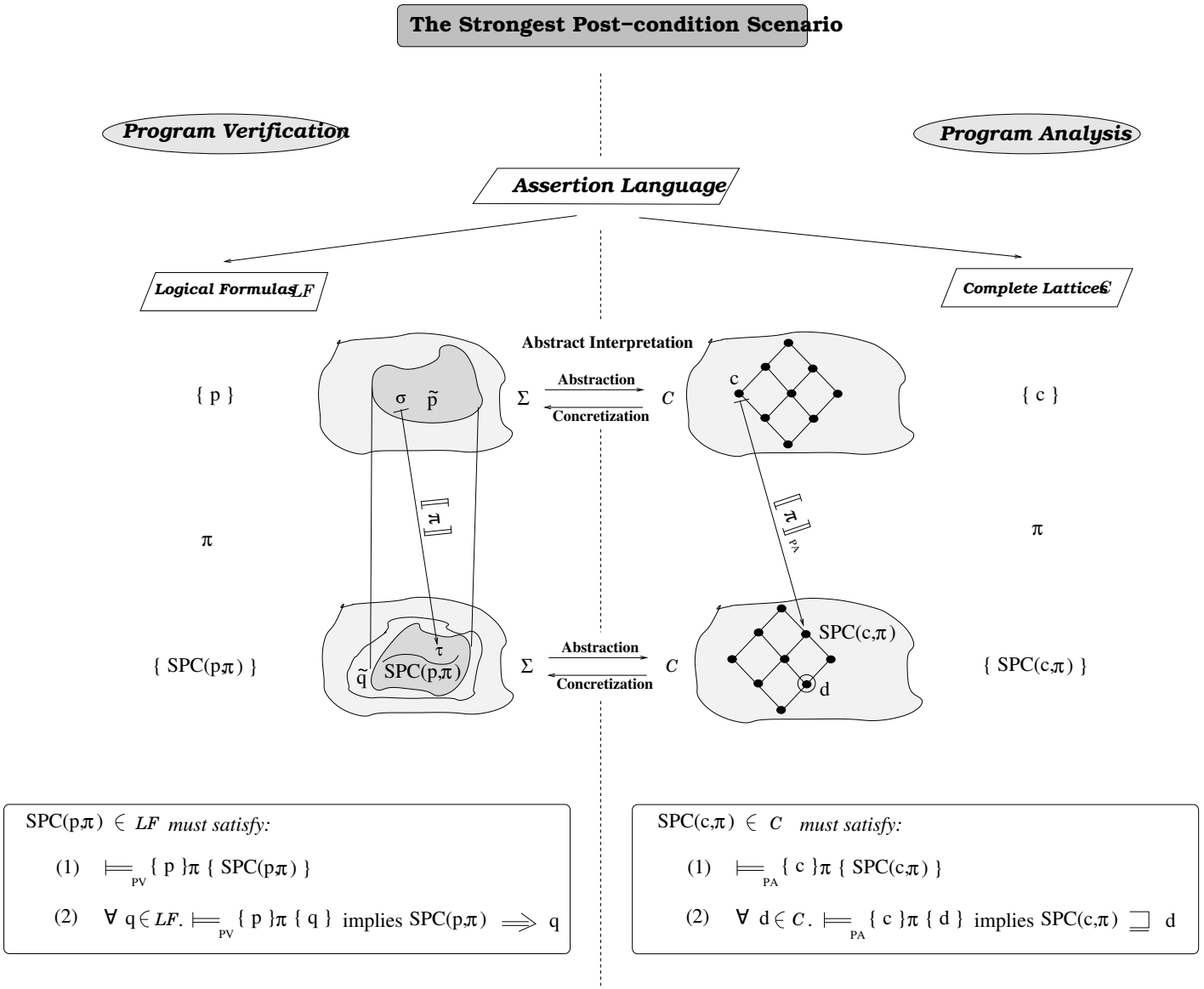

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 7, Program Analysis; Chapter 8, More on Program Analysis; Appendix B, Implementation of Program Analysis)


Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. 2nd edition, Springer-V., 2005. (Chapter 1, Introduction; Chapter 2, Data Flow Analysis; Chapter 6, Algorithms)

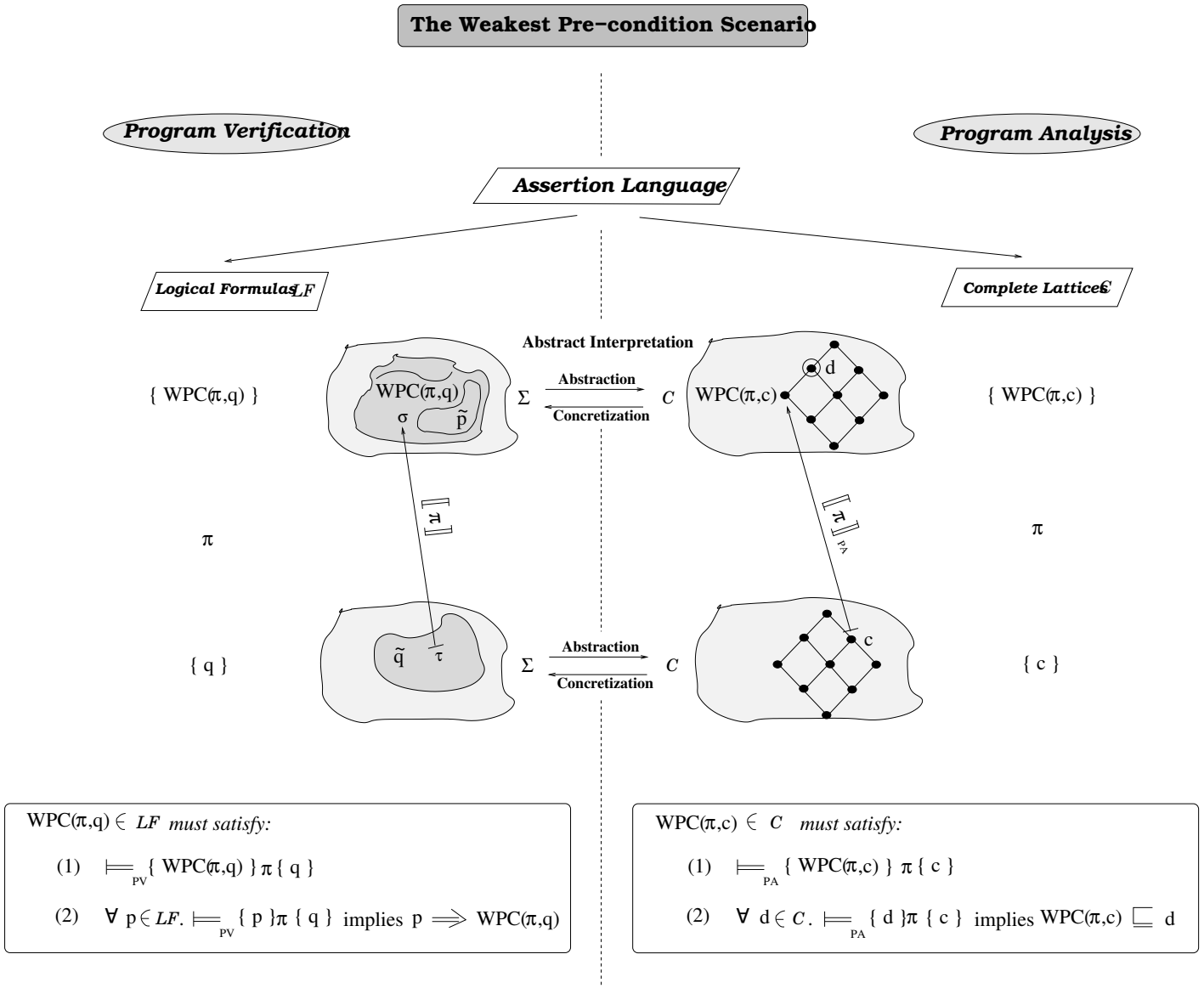
Kapitel 7

Programmverifikation vs. Programmanalyse

Programmverifikation vs. -analyse: SPC-Sicht



Programmverifikation vs. -analyse: WPC-Sicht



Kapitel 8

Reverse Datenflussanalyse

Kapitel 8.1

Grundlagen

Vorbereitung (1)

Erweiterung des DFA-Verbands $\widehat{\mathcal{C}}$ um ein *failure*-Element:

Sei $(\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ eine DFA-Spezifikation, sei *failure* ein neues Element nicht aus $\mathcal{C}$. Wir erweitern $\widehat{\mathcal{C}}$ (bzw. $\mathcal{C}$) um das Element *failure* zum erweiterten Verband

$$\widehat{\mathcal{C}}_f =_{df} (\mathcal{C}_f =_{df} \mathcal{C} \dot{\cup} \{\mathit{failure}\}, \sqsubseteq_f, \sqcap_f, \sqcup_f, \bot, \mathit{failure})$$

so dass *failure* größtes Element in $\widehat{\mathcal{C}}_f$ ist, d.h.

1. $\forall\, c \in \mathcal{C}_f.\ c \sqsubseteq_f \mathit{failure}$
2. $\forall\, c, c' \in \mathcal{C}.\ c \sqsubseteq_f c'$ gdw $c \sqsubseteq c'$

Beachte:

- Durch die Festlegung von $\sqsubseteq_f$ sind auch $\sqcap_f$ und $\sqcup_f$ eindeutig festgelegt.
- Das neue Element *failure* repräsentiert eine unerfüllbare DFA-Information.

Vorbereitung (2)

Ausdehnung des DFA-Funktionals $[\![\ ]\!]$ auf $\mathcal{C}$ zum DFA-Funktional $[\![\ ]\!]_f$ auf $\mathcal{C}_f$:

Hierzu legen wir fest:

$$\forall\, c \in \mathcal{C}_f\ \forall\, e \in E.\ [\![\, e\,]\!]_f(c) =_{df} \begin{cases} [\![\, e\,]\!](c) & \text{falls } c \neq \mathit{failure} \\ \mathit{failure} & \text{sonst} \end{cases}$$

Lemma 8.1.1

- $[\![\ ]\!]_f$ ist monoton gdw $[\![\ ]\!]$ ist monoton.
- $[\![\ ]\!]_f$ ist distributiv gdw $[\![\ ]\!]$ ist distributiv.

Reverse abstrakte Semantik

Sei $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ eine (lokale) abstrakte Semantik auf $\mathcal{C}$. Dann ist die durch die Erweiterung $[\![\ ]\!]_f$ von $[\![\ ]\!]$ definierte reverse (lokale) abstrakte Semantik wie folgt festgelegt:

Definition 8.1.2 (Reverse abstrakte Semantik)

Die reverse abstrakte Semantik zu einer abstrakten Semantik $(\widehat{\mathcal{C}}, [\![\ ]\!])$ ist gegeben durch:

1. (Reverser) DFA-Verband $\widehat{\mathcal{C}}_f =_{df} (\mathcal{C}_f, \sqsubseteq_f, \sqcap_f, \sqcup_f, \bot, \mathit{failure})$
2. Reverses DFA-Funktional
   $[\![\ ]\!]_R : E \to (\mathcal{C}_f \to \mathcal{C}_f)$ definiert durch

$$\forall\, e \in E\ \forall\, c \in \mathcal{C}.\ [\![\, e \,]\!]_R(c) =_{df} \bigsqcap \{\, c' \mid [\![\, e \,]\!]_f(c') \sqsupseteq c \,\}$$

DFA vs. rDFA

Intuitiv:

- DFA zielt für jede Programmstelle auf die Berechnung des stärkst möglichen Datenflussfakts (relativ zu einer gegebenen Startinformation).
- rDFA zielt für jede Programmstelle auf die Berechnung eines schwächst möglichen Datenflussfakts, so dass ein 'gewünschter' Datenflussfakt an einer bestimmten Programmstelle gültig ist.
- Reverses Gegenstück der meet-over-all-paths Globalisierung einer abstrakten Semantik ist daher die reverse join-over-all-paths Globalisierung der zugehörigen reversen abstrakten Semantik.

Grapherweiterung um Anfrageknoten

Der Übergang von DFA zu rDFA ist geradlinig, die Globalisierung der lokalen reversen abstrakten Semantik erfordert aber die folgende Grapherweiterung:

- Sei $G' = (N', E', \mathbf{s}', \mathbf{e}')$ ein Flussgraph und $q \in N'$ der interessierende Programmpunkt, der sog. Anfrageknoten (query node).
- Ist q von $\mathbf{s}'$ verschieden, dann wird G' durch eine Kopie $\mathbf{q}$ von q zu $G = (N, E, \mathbf{s}, \mathbf{e})$ erweitert, wobei der neue Knoten $\mathbf{q}$ dieselben Vorgänger wie q besitzt, aber keine Nachfolger.

Beobachtung

Es gilt:

- Die Hinzunahme von $\mathbf{q}$ hat keinen Einfluss auf die *MOP*-Lösung irgendeines der ursprünglichen Knoten von G.
- Die *MOP*-Lösungen von $\mathbf{q}$ und q stimmen überein.

Die folgenden drei Lemmata fassen diese Beobachtungen zusammen und formalisieren sie.

Zusammenhang von $[\![\ ]\!]_f$ und $[\![\ ]\!]_R$ (1)

Lemma 8.1.3

Sei $[\![\ ]\!]_f$ ein (erweitertes) DFA-Funktional. Dann gilt für jede Kante $e \in E$:

1. $[\![\ e\ ]\!]_R$ ist wohldefiniert und monoton.
2. $[\![\ e\ ]\!]_R$ ist additiv, falls $[\![\ e\ ]\!]_f$ distributiv ist.

Zusammenhang von $[\![\ ]\!]_f$ und $[\![\ ]\!]_R$ (2)

Lemma 8.1.4

Sei $[\![\ ]\!]_f$ ein (erweitertes) DFA-Funktional. Dann gilt für jede Kante $e \in E$:

1. $[\![e]\!]_R \circ [\![e]\!]_f \sqsubseteq Id_{\mathcal{C}_f}$, falls $[\![e]\!]$ monoton ist.
2. $[\![e]\!]_f \circ [\![e]\!]_R \sqsupseteq Id_{\mathcal{C}_f}$, falls $[\![e]\!]$ distributiv ist.

Sprechweise in der Theorie "Abstrakter Interpretation":

- $[\![e]\!]_f$ und $[\![e]\!]_R$ bilden eine Galois-Verbindung.

Zusammenhang von $[\![\ ]\!]_f$ und $[\![\ ]\!]_R$ (3)

Lemma 8.1.5

1. $\forall\, n \in N' \cap N.\ \mathbf{P}_{G'}[\mathbf{s}, n] = \mathbf{P}_G[\mathbf{s}, n]$
2. $\forall\, q \in N' \backslash \{\mathbf{s}\}.\ \mathbf{P}_{G'}[\mathbf{s}, q] = \mathbf{P}_G[\mathbf{s}, \mathbf{q}]$
3. $\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N' \cap N.\ MOP_{(G', c_{\mathbf{s}})}(n) = MOP_{(G, c_{\mathbf{s}})}(n)$
4. $MOP_{(G, c_{\mathbf{s}})}(q) = MOP_{(G, c_{\mathbf{s}})}(\mathbf{q})$

wobei $\mathbf{q}$ ("query node") Kopie von q ist mit $pred(\mathbf{q}) = pred(q)$ und $succ(\mathbf{q}) = \emptyset$.

rDFA-Spezifikation und rDFA-Problem

Mit den vorherigen Festlegungen und Beobachtungen ist es sinnvoll festzulegen:

Definition 8.1.6 (rDFA-Spezifikation)

Eine rDFA-Spezifikation zu einer abstrakten Semantik $(\widehat{\mathcal{C}}, [\![\ ]\!])$ ist ein Tripel $(\widehat{\mathcal{C}}_f, [\![\ ]\!]_R, c_q)$ gegeben durch

- eine reverse (lokale) abstrakte Semantik bestehend aus
  1. einem DFA-Verband $\widehat{\mathcal{C}}_f =_{df} (\mathcal{C}_f, \sqsubseteq_f, \sqcap_f, \sqcup_f, \bot, \mathit{failure})$
  2. einem reversen DFA-Funktional $[\![\ ]\!]_R : E \to (\mathcal{C}_f \to \mathcal{C}_f)$
- einer DFA-Frage: $c_q \in \mathcal{C}$

Definition 8.1.7 (rDFA-Problem)

Eine rDFA-Spezifikation legt ein rDFA-Problem fest.

Kapitel 8.2

R-JOP-Ansatz

Ausdehnung von $[\![\ ]\!]_R$ auf Pfade

Grundlegend für die *R-JOP*-Strategie:

Definition 8.2.1 (Pfadausdehnung von $[\![\ ]\!]_R$)

Die Ausdehnung einer reversen (lokalen) abstrakten Semantik auf Pfade $p = \langle e_1, \ldots, e_{q-1}, e_q \rangle$ ist definiert durch:

$$[\![\ p\ ]\!]_R =_{df} \begin{cases} Id_C & \text{falls } \lambda_p < 1 \\ [\![\ \langle e_1, \ldots, e_{q-1} \rangle\ ]\!]_R \circ [\![\ e_q\ ]\!]_R & \text{sonst} \end{cases}$$

Beachte: Die obige Ausdehnung bedeutet einen Rückwärtsdurchlauf von Pfad p.

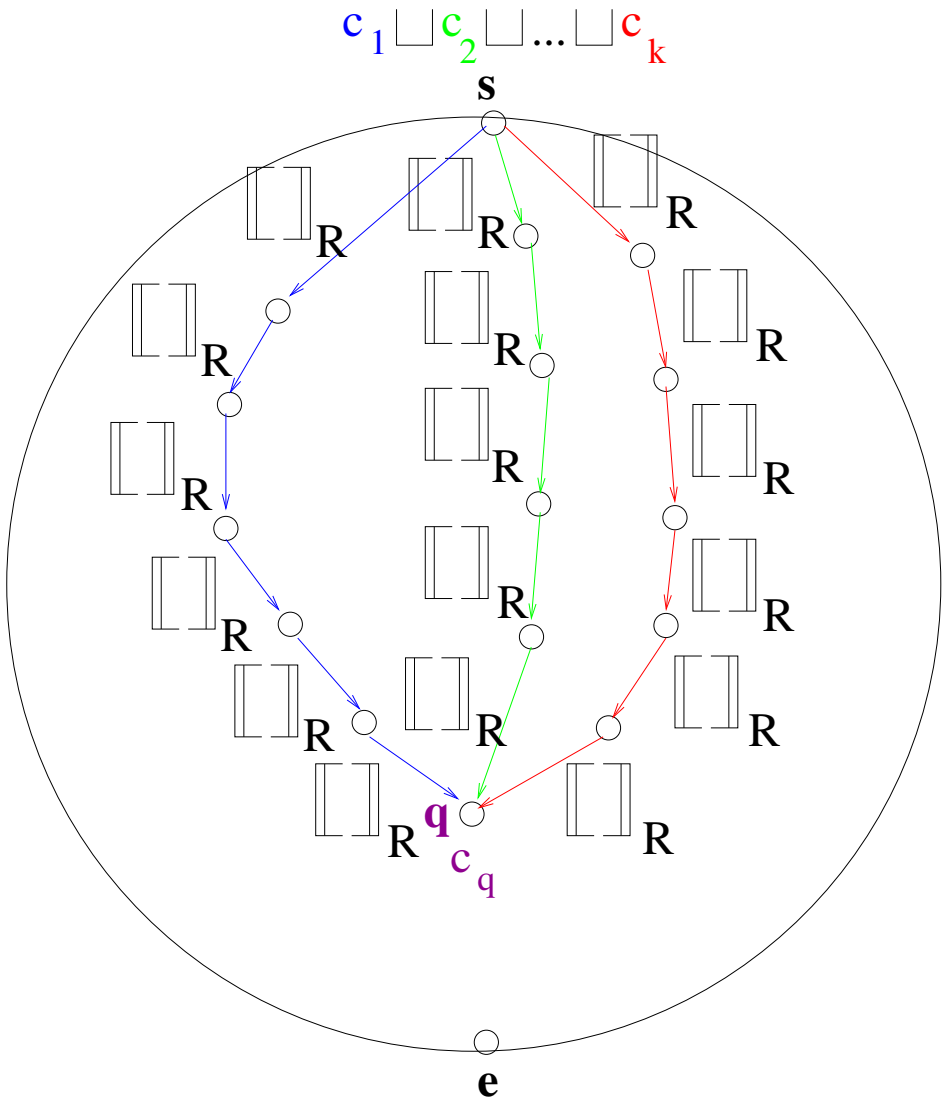
Der *R-JOP*-Ansatz

Definition 8.3.2 (*R-JOP*-Lösung)

Sei $(\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ die Spezifikation eines rDFA-Problems. Dann ist für alle Knoten $n \in N$ die *R-JOP*-Lösung definiert durch:

$$R\text{-}JOP_{(\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)}(n) =_{df} \bigsqcup \{ [\![p]\!]_R(c_q) \mid p \in \mathbf{P}[n, \mathbf{q}] \}$$

Veranschaulichung



Kapitel 8.3

R-MinFP-Ansatz

Der *R-MinFP*-Ansatz

Grundlegend für die *R-MinFP*-Strategie:

Definition 8.3.1 (*R-MinFP*-Gleichungssystem)

Das *R-MinFP*-Gleichungssystem ist gegeben durch:

$$reqInf(n) = \begin{cases} c_q & \text{falls } n = \mathbf{q} \\ \bigsqcup \{ [\![(n,m)]\!]_R(reqInf(m)) \mid m \in succ(n) \} & \text{sonst} \end{cases}$$

Die *R-MinFP*-Lösung

Definition 8.3.2 (*R-MinFP*-Lösung)

Sei $(\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ die Spezifikation eines rDFA-Problems. Dann ist für alle Knoten $n \in N$ die *R-MinFP*-Lösung definiert durch:

$$R\text{-}MinFP_{(\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)}(n) =_{df} reqInf^*_{c_q}(n)$$

wobei $reqInf^*_{c_q}$ die kleinste Lösung des *R-MinFP*-Gleichungssystems 8.3.1 bzgl. $(\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)$ bezeichnet.

Kapitel 8.4

Reverses Koinzidenz- und Sicherheitstheorem

Hauptresultate: Korrektheit u. Vollständigkeit

Grundsätzlich:

- Zusammenhang von *R-JOP*-Lösung und *R-MinFP*-Lösung

Genauer:

- Korrektheit und Vollständigkeit der *R-MinFP*-Lösung bzgl. der *R-JOP*-Lösung

Im Detail:

- Korrektheitsfrage:
  Gilt stets $\mathit{R\text{-}MinFP}_{(\widehat{\mathcal{C}_f},[\![\ ]\!]_R,c_q)} \sqsupseteq \mathit{R\text{-}JOP}_{(\widehat{\mathcal{C}_f},[\![\ ]\!]_R,c_q)}$?
- Vollständigkeitsfrage:
  Gilt stets $\mathit{R\text{-}MinFP}_{(\widehat{\mathcal{C}_f},[\![\ ]\!]_R,c_q)} \sqsubseteq \mathit{R\text{-}JOP}_{(\widehat{\mathcal{C}_f},[\![\ ]\!]_R,c_q)}$?

Korrektheit

Theorem 8.4.1 (Reverse Sicherheit)

Die *R-MinFP*-Lösung ist eine sichere (konservative), d.h. obere Approximation der *R-JOP*-Lösung, d.h.,

$$\forall\, c_q \in \mathcal{C}\ \forall\, n \in N.\ R\text{-}MinFP_{(\widehat{c}_f, [\![\ ]\!]_R, c_q)}(n) \sqsupseteq R\text{-}JOP_{(\widehat{c}_f, [\![\ ]\!]_R, c_q)}(n)$$

Beachte:

- Die reversen Semantikfunktionen sind stets monoton (vgl. Lemma 8.1.3(1)); deshalb braucht diese Voraussetzung in Theorem 8.4.1 nicht explizit genannt zu werden.

Vollständigkeit (und zugleich Korrektheit)

Theorem 8.4.2 (Reverse Koinzidenz)

Die *R-MinFP*-Lösung stimmt mit der *R-JOP*-Lösung überein, d.h.,

$$\forall\, c_q \in \mathcal{C}\ \forall\, n \in N.\ R\text{-}MinFP_{(\widehat{\mathcal{C}_f},[\![\ ]\!]_R,c_q)}(n) = R\text{-}JOP_{(\widehat{\mathcal{C}_f},[\![\ ]\!]_R,c_q)}(n)$$

falls das $[\![\ ]\!]_R$ induzierende DFA-Funktional $[\![\ ]\!]$ distributiv ist.

Beachte:

- Distributivität von $[\![\ ]\!]$ (und deshalb auch $[\![\ ]\!]_f$) impliziert Addivität von $[\![\ ]\!]_R$ (vgl. Lemma 8.1.1 und 8.1.3(2)). Somit ist Addivität von $[\![\ ]\!]_R$ implizit in Theorem 8.4.2 gefordert, wird aber auf die Distributivitätseigenschaft des Ursprungproblems zurückgeführt.

Kapitel 8.5

Generischer Fixpunktalgorithmus und Reverses Terminierungstheorem

Hauptresultat: Effektivität

Grundsätzlich:

- Zusammenhang von *R-MinFP*-Lösung und Generischem Fixpunktalgorithmus 8.5.1

Genauer:

- Effektivität des Generischen Fixpunktalgorithmus 8.5.1 für die *R-MinFP*-Lösung

Im Detail:

- Effektivitätsfrage:
  Terminiert der Generische Fixpunktalgorithmus 8.5.1 stets mit der $R\text{-}MinFP_{(\widehat{\mathcal{C}_f}, [\![\ ]\!]_R, c_q)}$-Lösung?

Der generische *R-MinFP*-Alg. 8.5.1 (1)

Eingabe: (1) Ein Flussgraph $G = (N, E, \mathbf{s}, \mathbf{e})$, (2) ein (Anfrage-) Programmpunkt q, (3) ein reverses DFA-Problem gegeben durch eine reverse (lokale) abstrakte Semantik bestehend aus einem erweiterten DFA-Verband $\widehat{\mathcal{C}_f}$ und einem reversen DFA-Funktional $[\![\ ]\!]_R : E \to (\mathcal{C}_f \to \mathcal{C}_f)$ induziert von einem DFA-Funktional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$, und (4) eine Anfrageinformation $c_q \in \mathcal{C}$ am Anfragepunkt q.

Der generische *R-MinFP*-Alg. 8.5.1 (2)

Ausgabe: Unter den Voraussetzungen des Terminierungstheorems 8.5.2 die $R\text{-}MinFP_{(\widehat{\mathcal{C}}_f, [\![\ ]\!]_R, c_q)}$-Lösung. Abhängig von den Eigenschaften des reversen DFA-Funktionals gilt dann:

(i) $[\![\ ]\!]_R$ ist additiv: Variable $reqInf[\mathbf{s}]$ enthält die schwächste Vorbedingung für c_q an q, d.h. den kleinsten DFA-Fakt, der beim Programmaufruf gelten muss, um die Gültigkeit von c_q an q zu garantieren. Ist dies *failure*, so gilt c_q an q definitiv nicht.

(ii) $[\![\ ]\!]_R$ ist monoton: Variable $reqInf[\mathbf{s}]$ enthält eine obere Approximation der schwächsten Vorbedingung für c_q an q, d.h. eine obere Schranke für den DFA-Fakt, der beim Programmaufruf gelten muss, um die Gültigkeit von c_q an q zu garantieren. Ist dies *failure*, so schließt dies aufgrund der Überapproximation nicht aus, dass c_q an q gilt.

Der generische *R-MinFP*-Alg. 8.5.1 (3)

Bemerkung:

- Die schwächste Vorbedingung ist durch die $R\text{-}JOP_{(\widehat{C}_f, [\![\ ]\!]_R, c_q)}$-Lösung gegeben.
- Die Variable *workset* steuert den iterativen Prozess. Ihre Elemente sind Knoten aus dem erweiterten Flussgraphen G', deren Annotation jüngst aktualisiert worden ist, was möglicherweise zu einer verbandsmäßig größeren Annotation an ihren Vorgängerknoten führt.

Der generische *R-MinFP*-Alg. 8.5.1 (4)

(Prolog: Initialisierung von *reqInf* und *workset*)
FORALL $n \in N \backslash \{\mathbf{q}\}$ DO $reqInf[n] := \bot$ OD;
$reqInf[\mathbf{q}] := c_q$;
$workset := \{\mathbf{q}\}$;

Der generische *R-MinFP*-Alg. 8.5.1 (5)

```
( Hauptprozess: Iterative Fixpunktberechnung )
WHILE workset ≠ ∅ DO
    CHOOSE m ∈ workset;
        workset := workset\{m};
        ( Aktualisierung d. Vorgängerumgebung von Knoten m )
        FORALL n ∈ pred(m) DO
            join := [[ (n, m) ]]_R(reqInf[m]) ⊔_f reqInf[n];
            IF reqInf[n] ⊏_f join
                THEN
                    reqInf[n] := join;
                    workset := workset ∪ {n}
            FI
        OD
    ESOOHC
OD.
```

Effektivität

Aufgrund der stets gegebenen Monotonie der reversen Semantikfunktionen (vgl. Lemma 8.1.2(1)) erhalten wir:

Theorem 8.5.2 (Reverse Terminierung)

Der Generische Fixpunktalgorithmus 8.5.1 terminiert mit der $R\text{-}MinFP_{(\widehat{\mathcal{C}}_f, [\![\ ]\!]_R, c_q)}$-Lösung, falls der DFA-Verband $\widehat{\mathcal{C}}$ die aufstei- gende Kettenbedingung erfüllt.

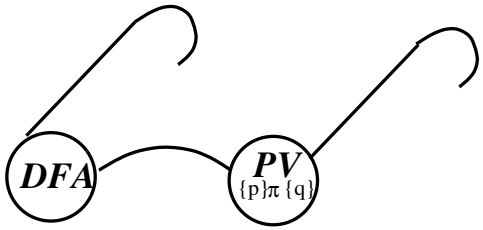
Beachte:

- $\widehat{\mathcal{C}}_f$ erfüllt die aufsteigende Kettenbedingung gdw $\widehat{\mathcal{C}}$ die aufsteigende Kettenbedingung erfüllt. Wie bei Theorem 8.4.2 wird hier die entscheidende Voraussetzung auf die entsprechende Eigenschaft des Ursprungsproblems zurückgeführt.

Kapitel 8.6

Analyse und Verifikation: Analogien und Gemeinsamkeiten

DFA/rDFA vs. Verifikation



Data–flow Analysis | **Program Verification**

MaxFP — $[\![e]\!] \in \sqsubset C \xrightarrow{\text{distributive}} C \sqsupset$ — *Strongest Postcondition View*

Coincidence Theorem ||

$MOP_{c_s}(q) \sqsupseteq c_q$ — $\{p\}\pi\{?\}$

induces

Link Theorem

$rJOP_{c_q}(s) \sqsubseteq c_s$ — $[\![e]\!]_R(c) =_{df} \bigsqcap \{c' \mid [\![e]\!](c') \sqsupseteq c\}$ — $\{?\}\pi\{q\}$

Reverse Coincidence Theorem ||

rMinFP — *Weakest Precondition View*

Zusammenhang DFA und rDFA

...für distributive DFA-Probleme.

Theorem 8.6.1 (Zusammenhangs-Theorem)

Es gilt:

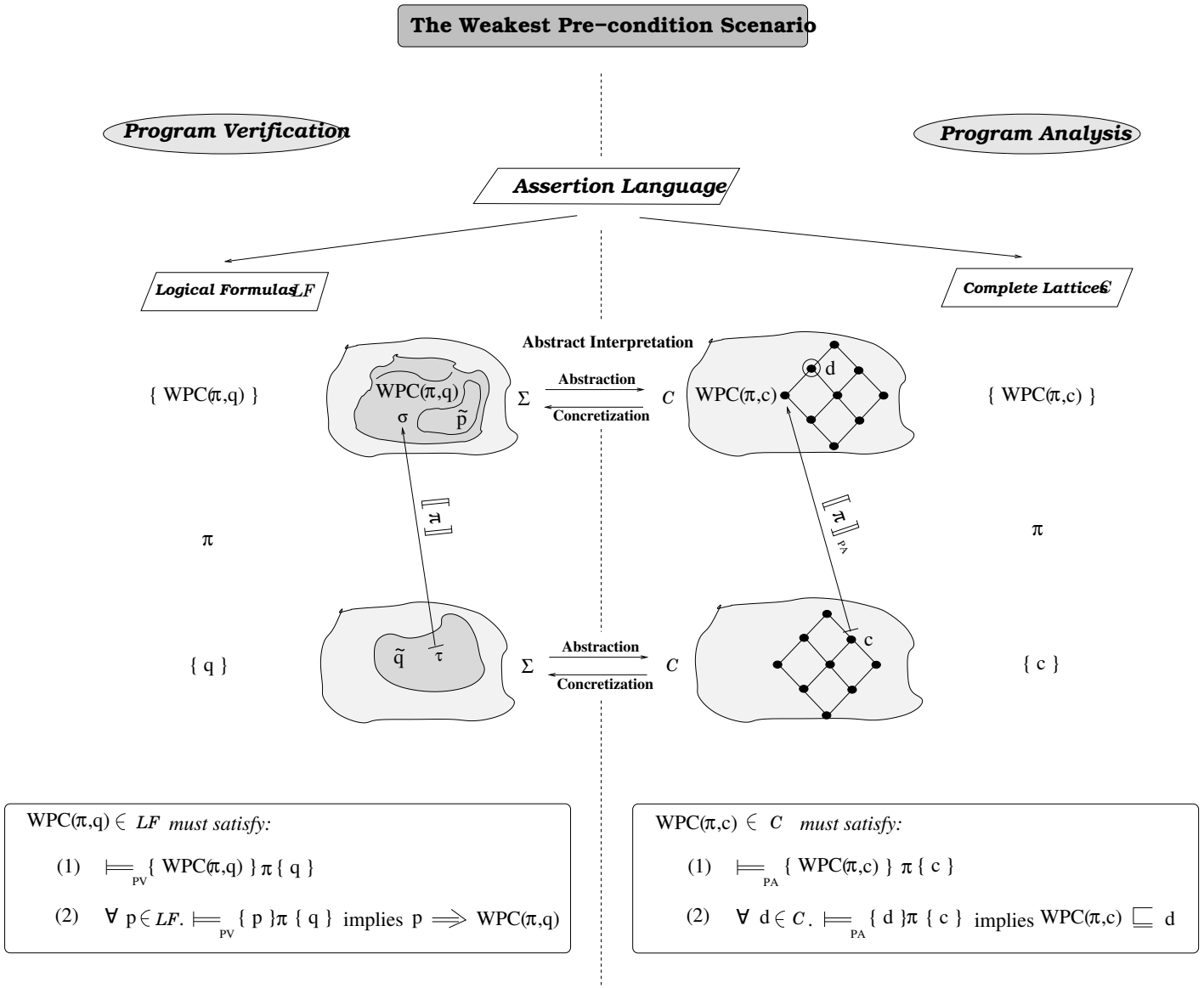
$$MaxFP_{(\widehat{\mathcal{C}},[\![\ ]\!],c_{\mathbf{s}})}(q) \;=\; MOP_{(\widehat{\mathcal{C}},[\![\ ]\!],c_{\mathbf{s}})}(q) \;\sqsupseteq\; c_q$$

$$\Longleftrightarrow$$

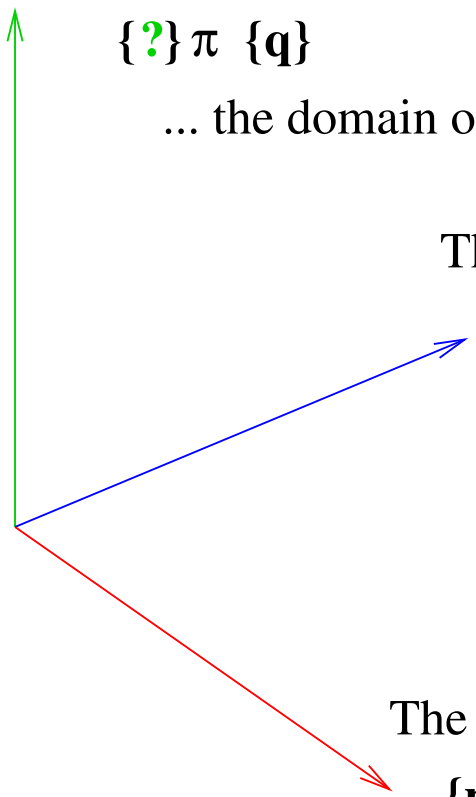
$$R\text{-}MinFP_{(\widehat{\mathcal{C}}_f,[\![\ ]\!]_R,c_q)}(\mathbf{s}) \;=\; R\text{-}JOP_{(\widehat{\mathcal{C}}_f,[\![\ ]\!]_R,c_q)}(\mathbf{s}) \;\sqsubseteq\; c_{\mathbf{s}}$$

falls das DFA-Funktional $[\![\ ]\!]$ distributiv ist.

Insgesamt: WPC-Analogie Verifikation/(r)DFA



Drei unterschiedliche Problemsichten (1)



The ***specification problem:***

{?} π {q}

... the domain of ***reverse DFA***

The ***verification problem:***

{p} π {q} ?

... the domain of ***demand–driven DFA***

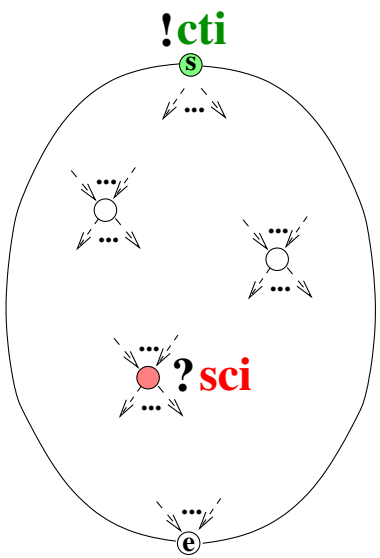
The ***implementation problem:***

{p} π {?}

... the domain of ***exhaustive DFA***

Drei unterschiedliche Problemsichten (2)

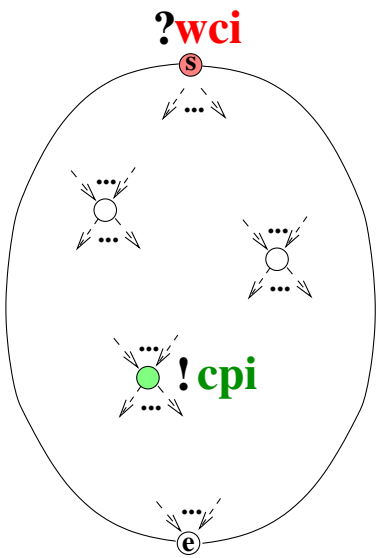
Implementation Problem



! **Given:** Context Information **cti**

? **Sought:** Strongest Component Information **sci**

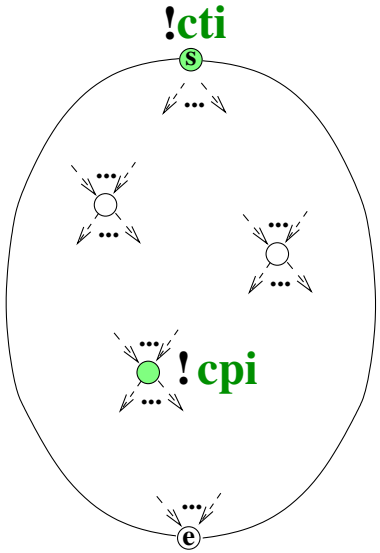
Specification Problem



! **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 8.7

Anwendungen reverser Datenflussanalyse

Beispiele

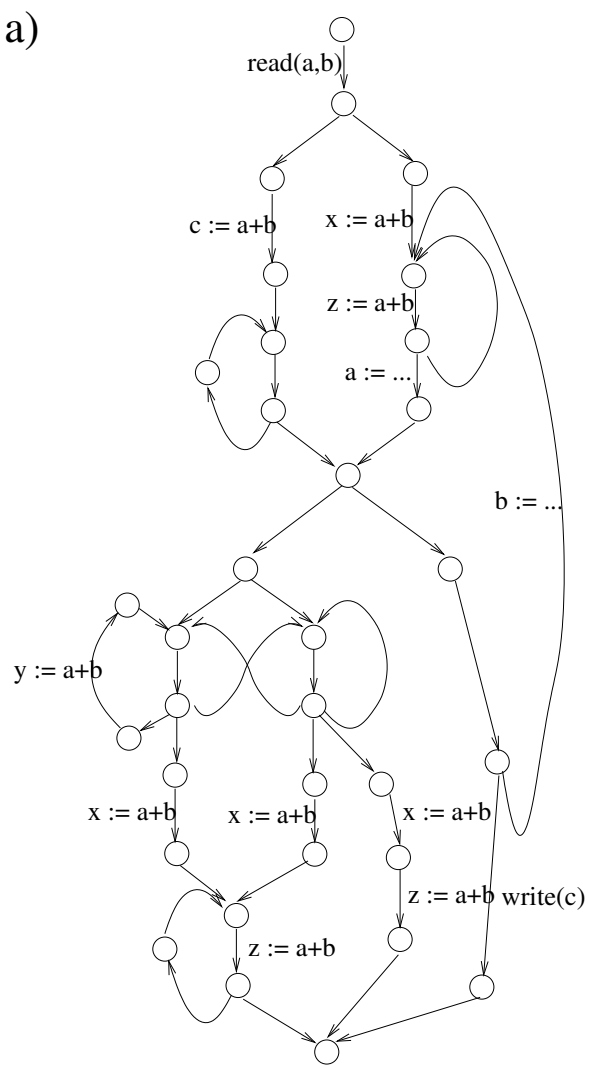
Anwendungen reverser DFA

- Einfache Analysatoren und Optimierer
- "Hot Spot" Programmanalyse und -optimierung
- Debugger
- ...

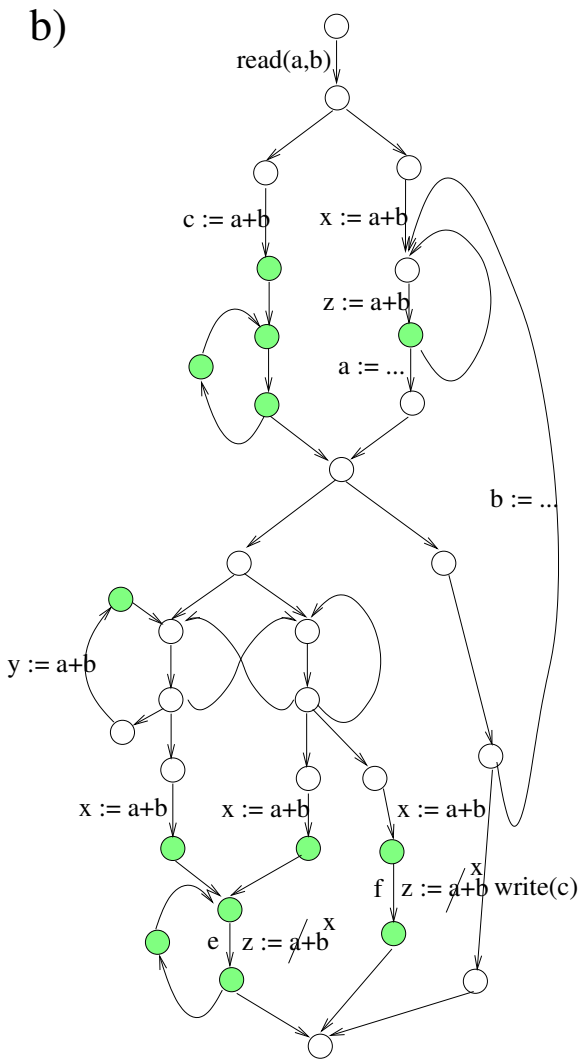
Insbesondere

- Anforderungsgetriebene Datenflussanalyse (demand-driven data-flow analysis)

Einfacher Analysator und Optimierer



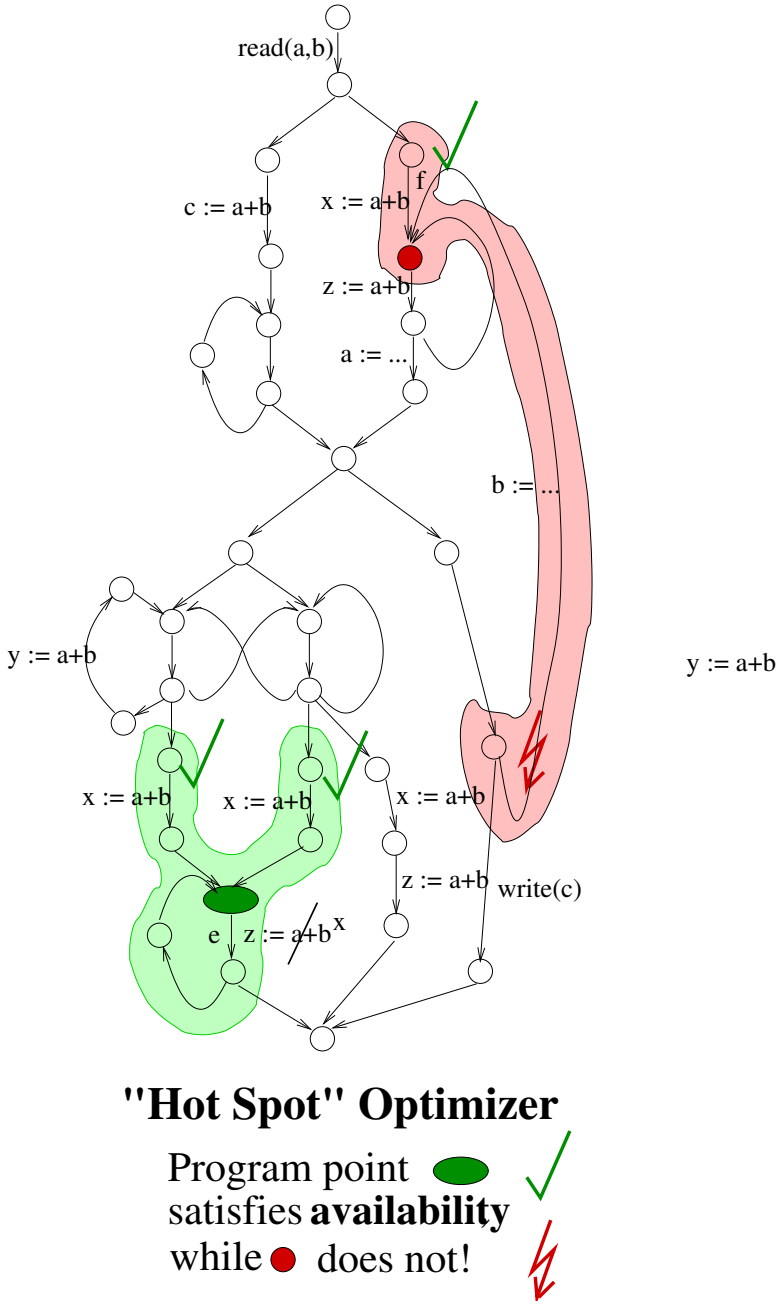
Flow graph



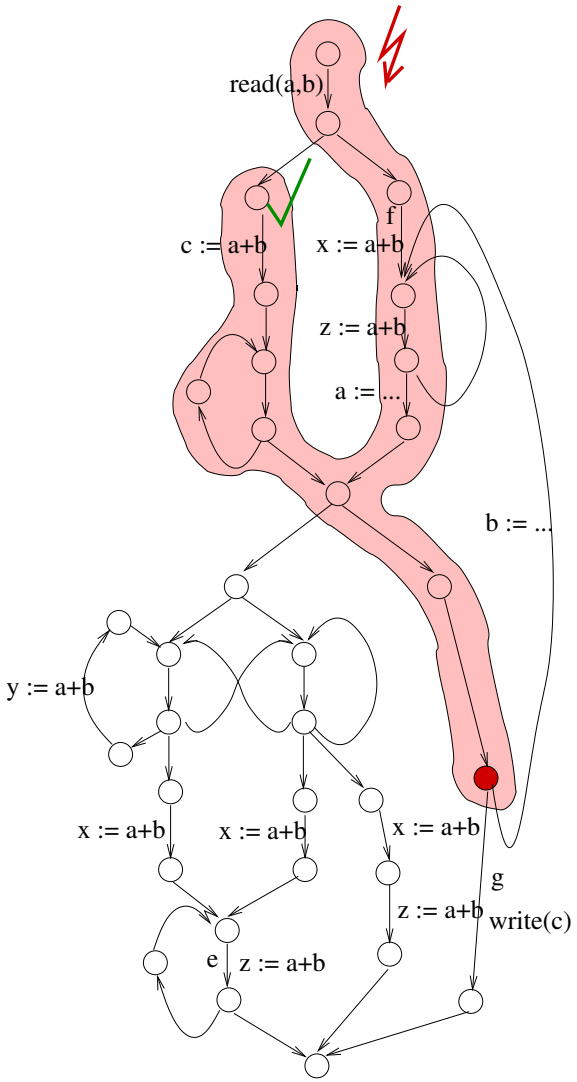
Data−flow information

Program points satisfying **availability** ●

“Hot Spot”-Analysator und Optimierer



Debugger



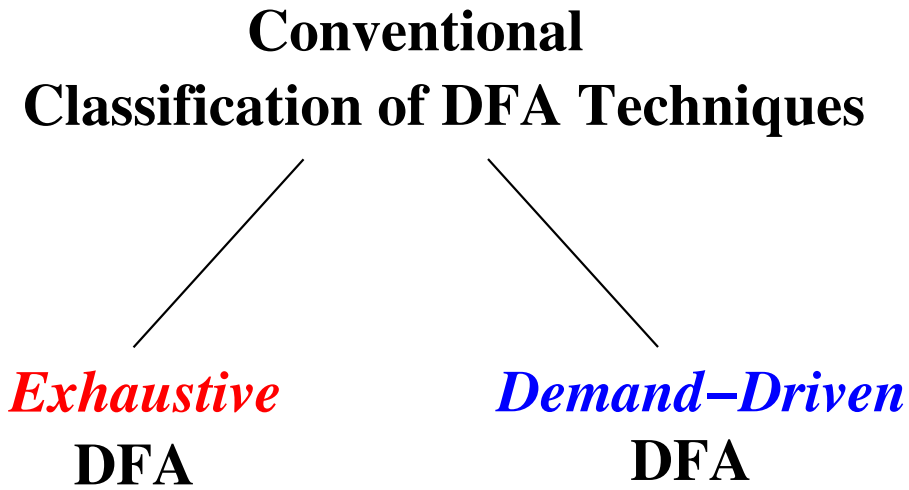
Debugger

Variable c is not initialized along some paths reaching program point ●

Erschöpfende vs. anforderungsgetriebene DFA

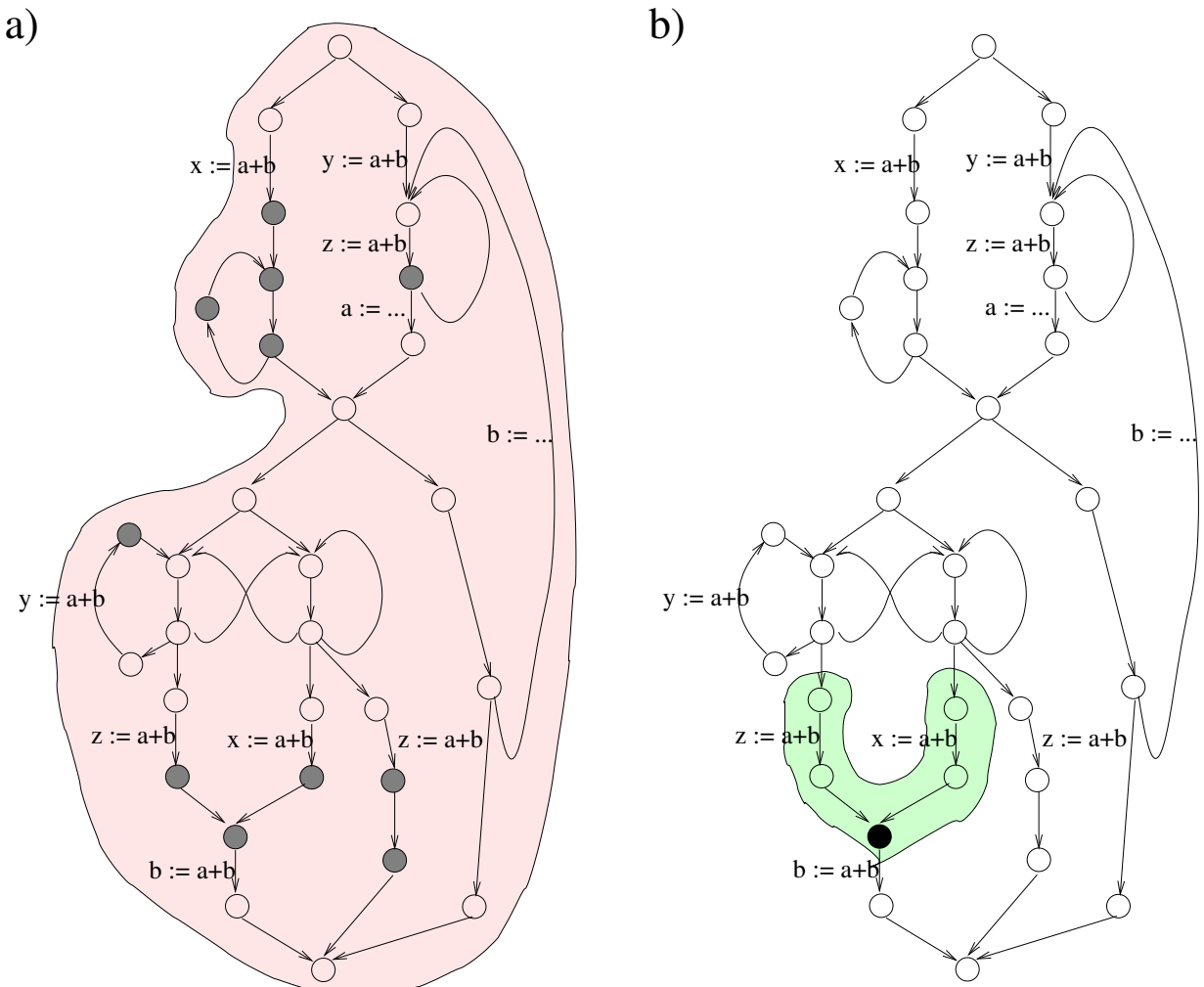
Erschöpfende (xDFA) vs.

anforderungsgetriebene DFA (ddDFA):



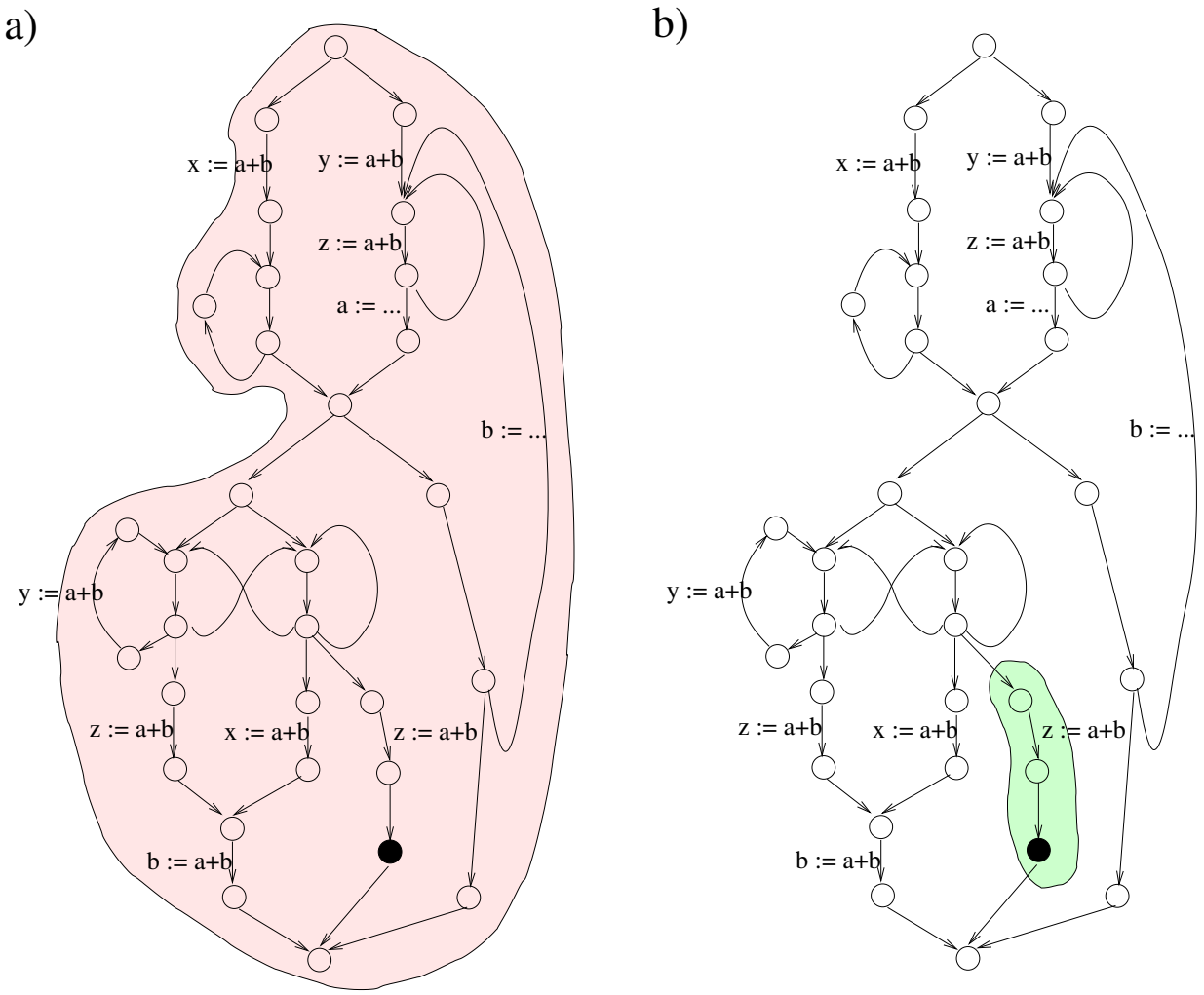
Aufwand xDFA und ddDFA im Vergleich (1)

Verfügbarkeit an einem Punkt: Vergleich Berechnungsaufwand erschöpfend (rosa), anforderungsgetrieben (grün)



Aufwand xDFA und ddDFA im Vergleich (2)

Verfügbarkeit an einem Punkt: Vergleich Berechnungsaufwand erschöpfend (rosa), anforderungsgetrieben (grün)



Reverse Verfügbarkeit (1)

Reverse Verfügbarkeit im Detail

Erforderliche Hilfsfunktionen:

$R\text{-}Cst_{\mathbf{wahr}}$, $R\text{-}Cst_{\mathbf{falsch}}$, und $R\text{-}Id_{\mathcal{B}_X}$:

$$\forall\, b \in \mathcal{B}_X.\ R\text{-}Cst_{\mathbf{wahr}}(b) =_{df} \begin{cases} \mathbf{falsch} & \text{falls } b \in \mathbb{B} \\ \mathit{failure} & \text{sonst (d.h. falls } b = \mathit{failure}) \end{cases}$$

$$\forall\, b \in \mathcal{B}_X.\ R\text{-}Cst_{\mathbf{falsch}}(b) =_{df} \begin{cases} \mathbf{falsch} & \text{falls } b = \mathbf{falsch} \\ \mathit{failure} & \text{sonst} \end{cases}$$

$$R\text{-}Id_{\mathcal{B}_X} =_{df} Id_{\mathcal{B}_X}$$

wobei $\mathcal{B}_X =_{df} \{\mathbf{falsch}, \mathbf{wahr}, \mathit{failure}\}$.

Reverse Verfügbarkeit (2)

Reverse abstrakte Semantik für Verfügbarkeit:

1. DFA-Verband:
   $(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathcal{B}_X, \wedge_f, \vee_f, \leq_f, \textbf{falsch}, \textit{failure})$
   mit **falsch** $\leq_f$ **wahr** $\leq_f$ *failure*

2. Reverses DFA-Funktional:
   $[\![\ ]\!]_{av_R} : E \to (\ \mathcal{B}_X \to \mathcal{B}_X\ )$ definiert durch

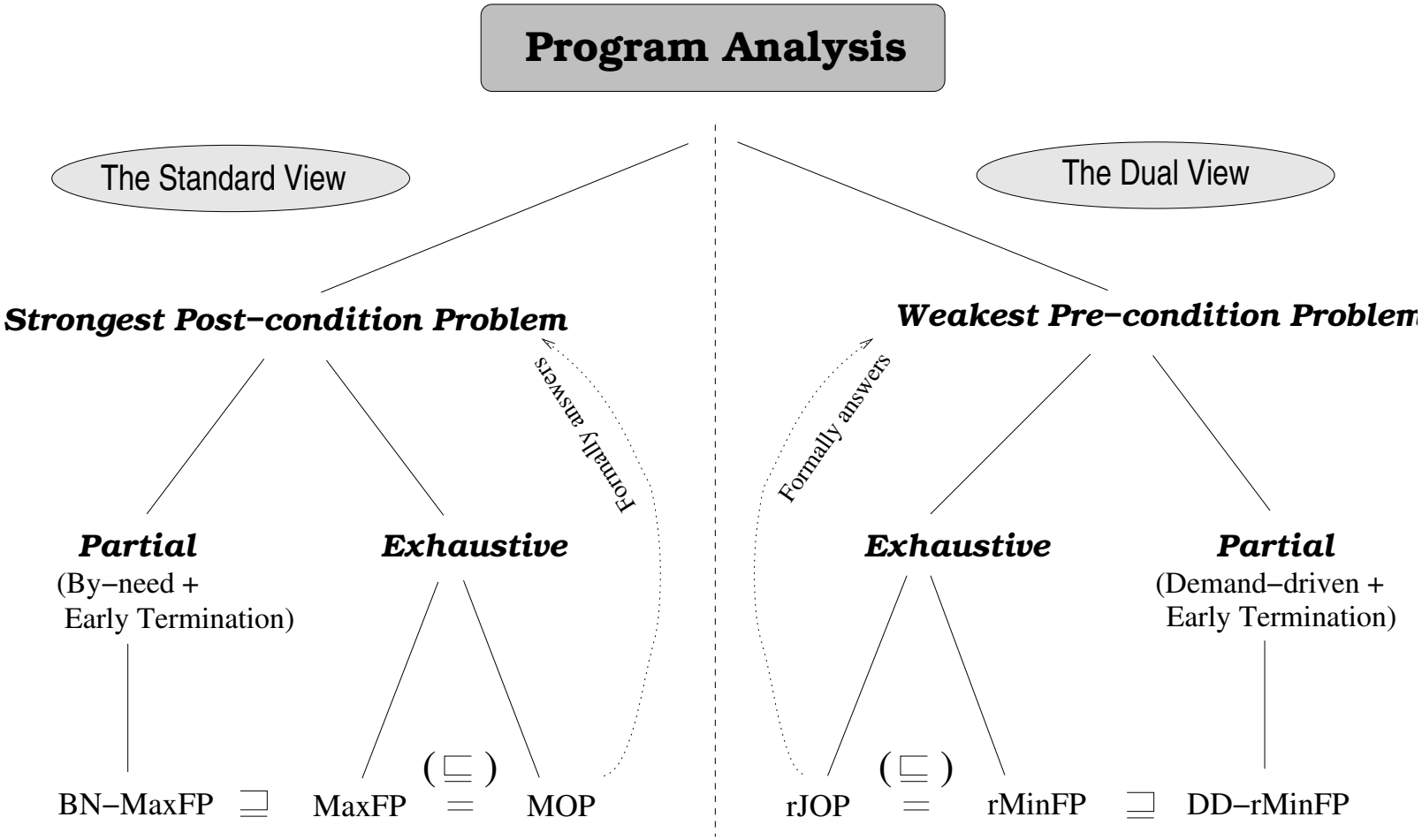
$$\forall\, e \in E.\ [\![\ e\ ]\!]_{av_R} =_{df} \begin{cases} R\text{-}Cst_{\textbf{wahr}} & \text{falls } [\![\ e\ ]\!]_{av} = Cst_{\textbf{wahr}} \\ R\text{-}Id_{\mathcal{B}_X} & \text{falls } [\![\ e\ ]\!]_{av} = Id_{\mathbb{B}} \\ R\text{-}Cst_{\textbf{falsch}} & \text{falls } [\![\ e\ ]\!]_{av} = Cst_{\textbf{falsch}} \end{cases}$$

Hinweis: Siehe auch Ergänzungsfolien zu "Hot Spot"-Programmanalyse und -optimierung auf der Webseite zur LVA.

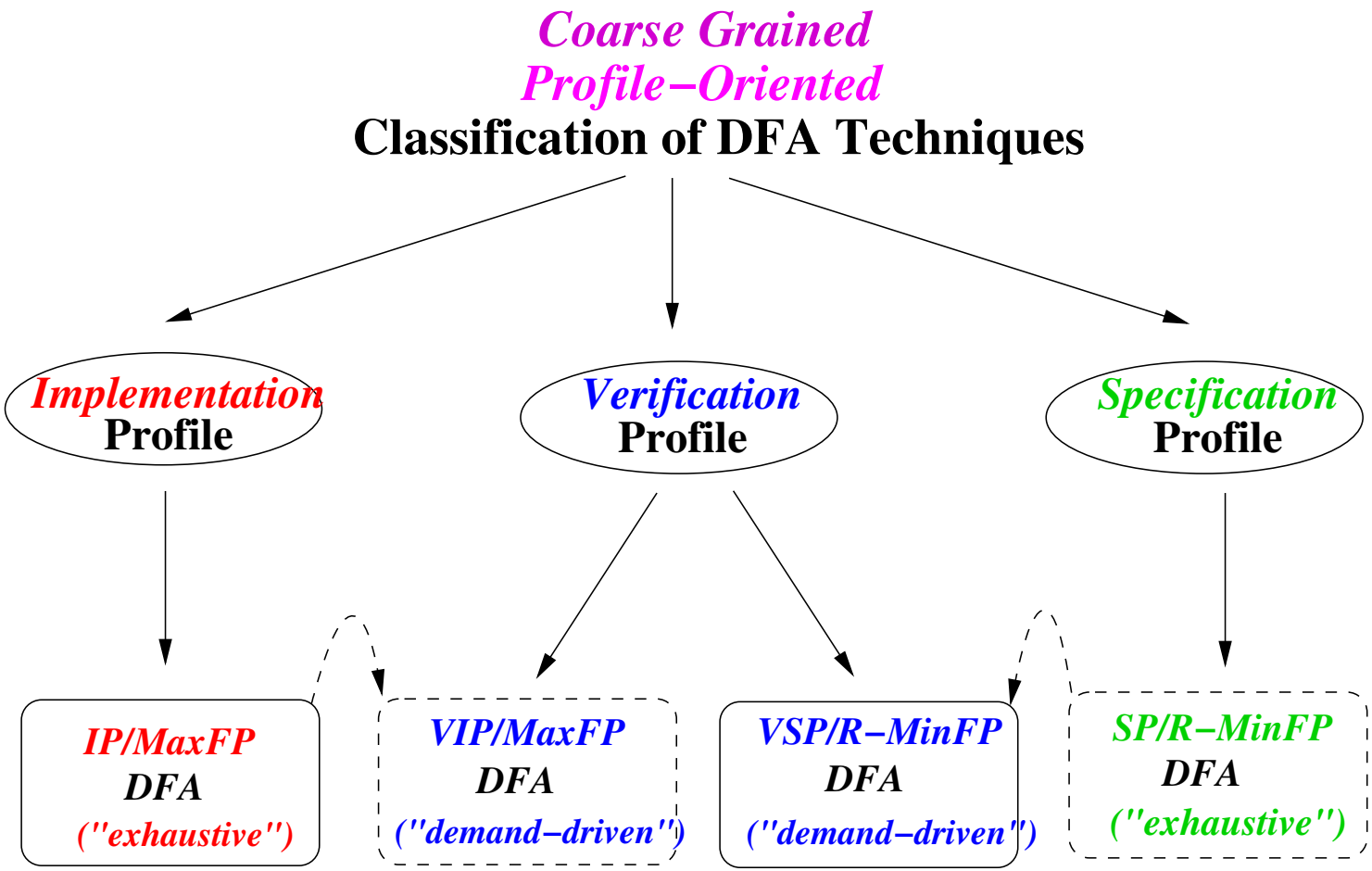
Kapitel 8.8

Zusammenfassung

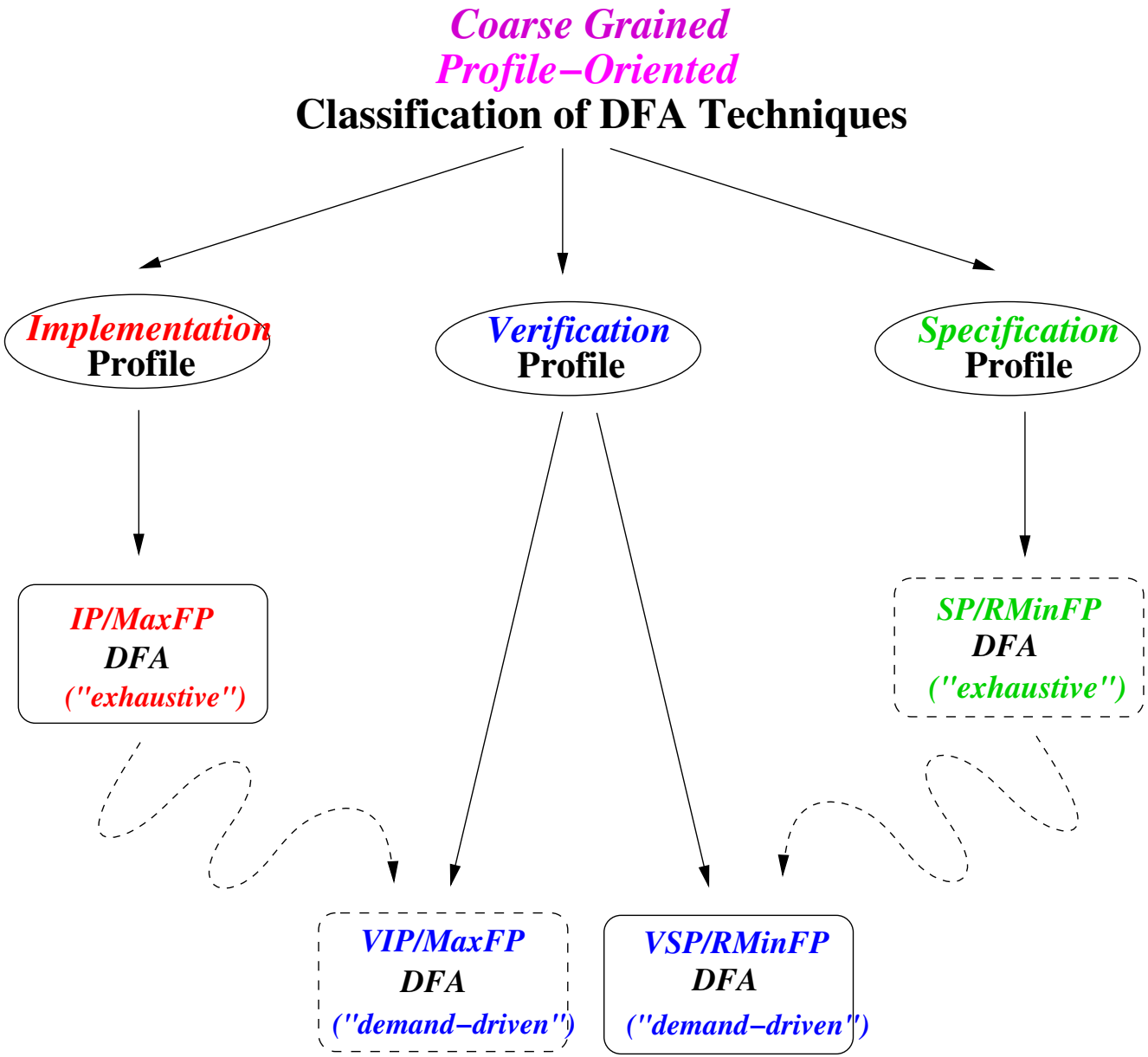
Gegenüberstellung von xDFA und ddDFA



Induzierte Lösungsstrategien (1)

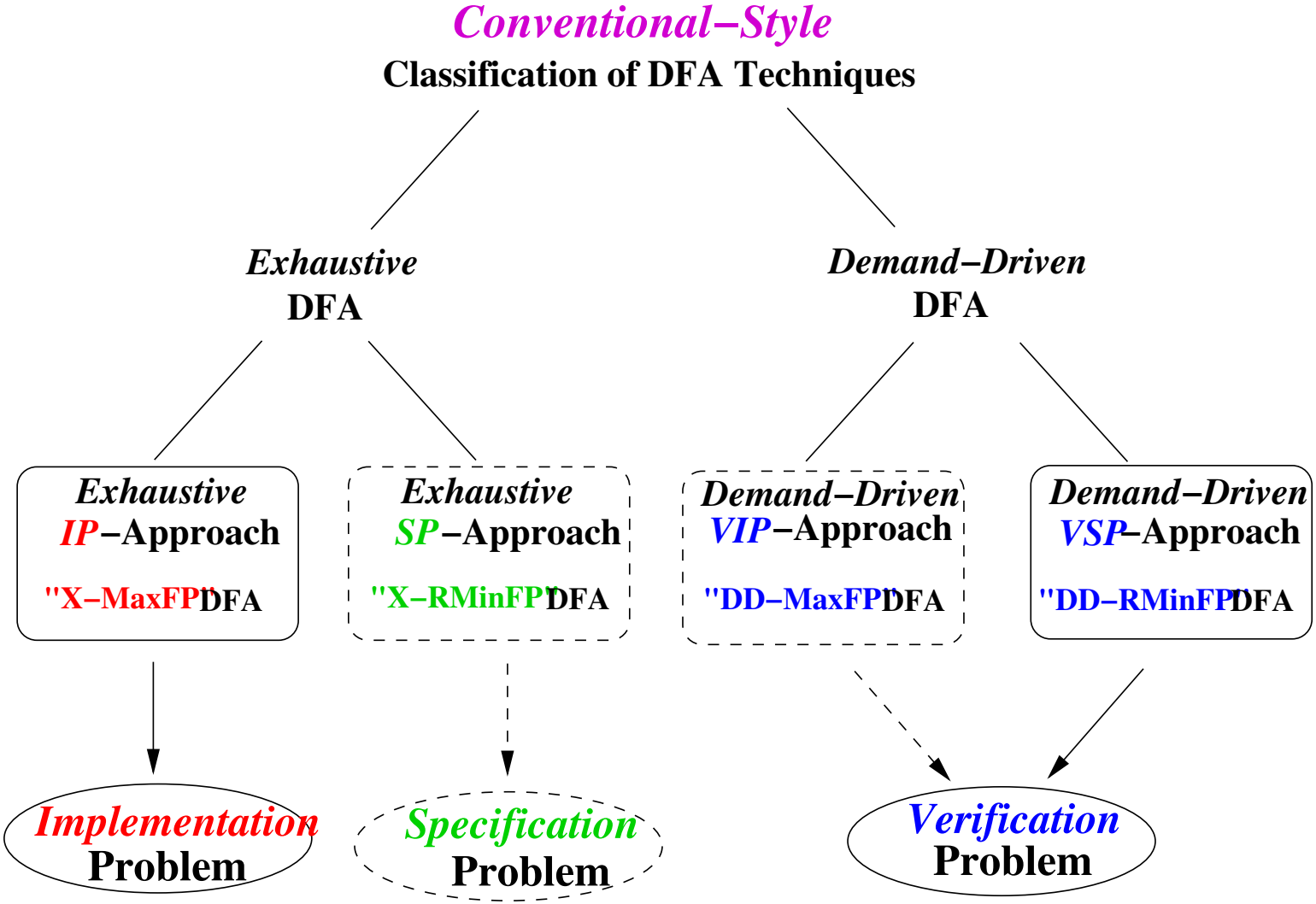


Induzierte Lösungsstrategien (2)



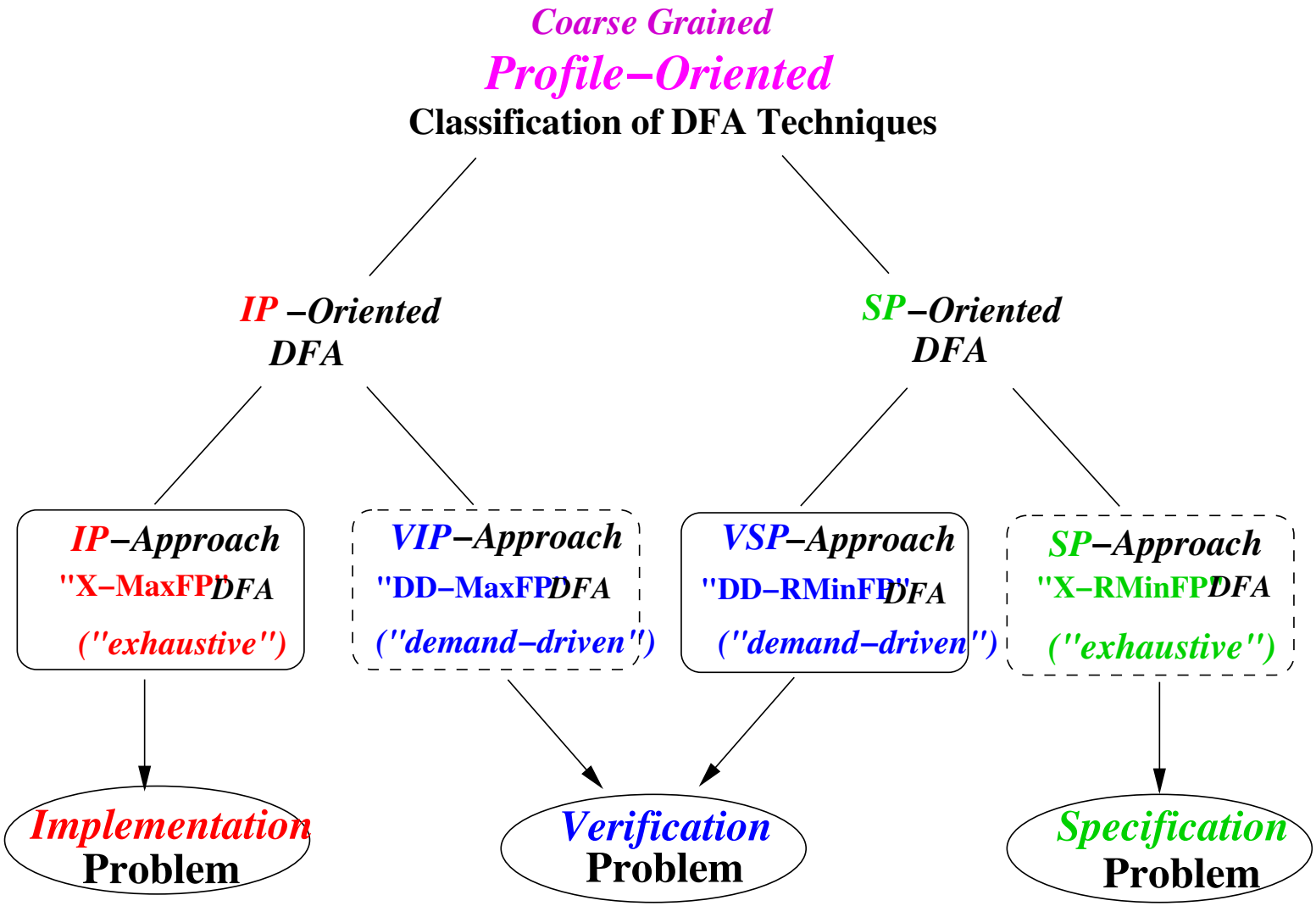
Algorithmenorientierte Sicht (1)

...zur Klassifikation von DFA-Problemen.



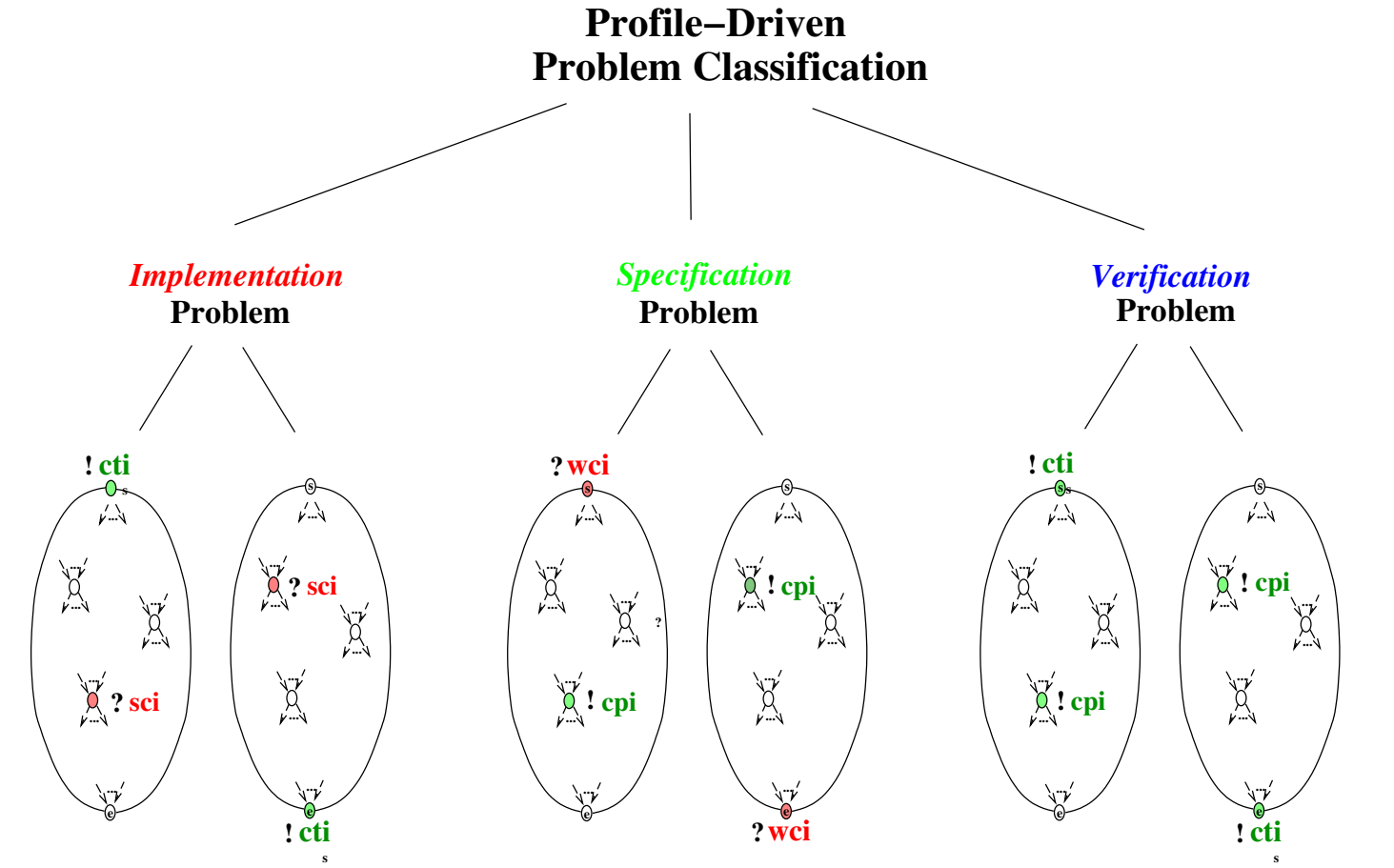
Algorithmenorientierte Sicht (2)

...zur Klassifikation von DFA-Problemen.



Programm- und problemorientierte Sicht

...zur Klassifikation von DFA-Problemen.



! 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 8.9

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 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, 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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 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, Germany, 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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 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 (Ed.), 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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 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 (SC'95), 414-423, 1995.


X. Yuan, Rajiv Gupta, R. Melham. *Demand-driven Data Flow Analysis for Communication Optimization*. Parallel Processing Letters 7(4):359-370, 1997.


X. Zheng, R. 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

Chaotische Fixpunktiteration

Kapitel 9.1

Motivation

Motivation

Viele praktisch relevante Probleme in der Informatik lassen sich durch die

- kleinste (größte) gemeinsame Lösung

von Systemen rekursiver Gleichungen beschreiben:

$$\begin{aligned} x &= f_1(x) \\ &\vdots \\ x &= f_n(x) \end{aligned}$$

System rekursiver Gleichungen

Sei

$$\begin{aligned} x &= f_1(x) \\ &\vdots \\ x &= f_n(x) \end{aligned}$$

ein System rekursiver Gleichungen, wobei

$$\mathcal{F} =_{df} \{f_k : D \to D \mid 1 \leq k \leq n\}$$

eine Familie monotoner Funktionen auf einer wohlfundierten partiellen Ordnung $(D, \sqsubseteq)$ ist.

Fixpunkte vs. Lösungen

Fixpunkte von Funktionen vs. Lösungen von Gleichungen

Wir wollen einsehen, dass das

- Lösen eines Systems rekursiver Gleichungen

$$\begin{aligned} x &= f_1(x) \\ &\vdots \\ x &= f_n(x) \end{aligned}$$

der

- Berechnung eines Fixpunktes von $\mathcal{F}$, d.h. eines gemeinsamen Fixpunkts $x = f_k(x)$ für alle f_k.

entspricht.

Chaotische Iteration

Fixpunktberechnung mittels chaotischer Iteration:

- Ein typischer Iterationsalgorithmus beginnt mit dem initialen Wert $\perp$ für x, dem kleinsten Element von D, und aktualisiert sukzessive den Wert von x durch Anwendung der Funktionen f_k in einer beliebigen Reihenfolge, um so den kleinsten gemeinsamen Fixpunkt von $\mathcal{F}$ zu approximieren.

Diese Vorgehensweise wird als chaotische Iteration bezeichnet.

Bekannte Fixpunkttheoreme aus der Literatur

Möglicherweise am bekanntesten das

- Fixpunkttheorem von Tarski [1955]
  - Garantiert Existenz kleinster Fixpunkte für monotone Funktionen über vollständigen partiellen Ordnungen
  - Iteration: $\vec{x}_0 = \vec{\bot}, \vec{x}_1 = \vec{f}(\vec{x}_0), \vec{x}_2 = \vec{f}(\vec{x}_1), \ldots$, wobei $\vec{x}_i$ den Wert von $\vec{x}$ nach der i-ten Iteration bezeichnet.
  - Vielfach anwendbar, aber dennoch oft zu speziell.

Verallgemeinerungen:

- 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 $\vec{f}$
  - Iteration:
    $\vec{x}_0 = \vec{\bot}, \vec{x}_1 = \vec{f}_{J_0}(\vec{x}_0), \vec{x}_2 = \vec{f}_{J_1}(\vec{x}_1), \ldots$, 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 $\vec{x}_j$, $j \leq i$.

Zum Nachschlagen und -lesen

- Alfred Tarski. *A Lattice-theoretical fixpoint theorem and its applications*. Pacific Journal of Mathematics, Vol. 5, 285-309, 1955.
- F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, France, Dec. 1976.

Ein historischer Abriss findet sich in:

- 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 diesem Kapitel

Vorstellung eines weiteren Fixpunkttheorems, das ohne Monotonie 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.

Kapitel 9.2

Chaotisches Fixpunktiterationstheorem

Vorbereitungen (1)

- Eine partielle Ordnung $(D, \sqsubseteq)$ ist ein Paar aus einer Menge D und einer reflexiven, antisymmetrischen und transitiven zweistelligen Relation $\sqsubseteq \subseteq D \times D$.
- Eine Folge $(d_i)_{i \in \mathbb{N}}$ von Elementen $d_i \in D$ heißt (aufsteigende) Kette, falls $\forall i \in \mathbb{N}.\ d_i \sqsubseteq d_{i+1}$.
- Eine Kette $T =_{df} (d_i)_{i \in \mathbb{N}}$ heißt stationär, falls $\{d_i \mid i \in \mathbb{N}\}$ endlich ist.
- Eine partielle Ordnung $\sqsubseteq$ heißt wohlfundiert, falls jede Kette stationär ist.

Vorbereitungen (2)

- Eine Funktion $f : D \to D$ auf D heißt
  - monoton, falls $\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$.
  - inflationär (vergrößernd), falls $\forall d \in D.\ d \sqsubseteq f(d)$
- Ist $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine Familie von Funktionen und $s = (s_1, \ldots, s_n) \in \mathbb{N}^*$, dann ist f_s definiert durch die Komposition

$$f_s =_{df} f_{s_n} \circ \cdots \circ f_{s_1}$$

Strategien, Iterationsfolgen, Fairness

Definition 9.2.1 (Strategie, Chaotische Iterationsfolge, Fairness)

Sei $(D, \sqsubseteq)$ eine partielle Ordnung und $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine Familie inflationärer Funktionen $f_k : D \to D$.

- Eine Strategie ist eine beliebige Funktion $\gamma : \mathbb{N} \to \mathbb{N}$.
- Eine Strategie γ und ein Element $d \in D$ induzieren eine chaotische Iteration $f_\gamma(d) = (d_i)_{i \in \mathbb{N}}$ von Elementen $d_i \in D$, die induktiv definiert sind durch $d_0 = d$ und $d_{i+1} = f_{\gamma(i)}(d_i)$.
- Eine Strategie γ heißt fair gdw

$$\forall i, k \in \mathbb{N}. \, (f_k(d_i) \neq d_i \text{ impliziert } \exists j > i. \, d_j \neq d_i)$$

Ein abgeschwächter Monotoniebegriff

Definition 9.2.2 (Verzögerungsmonotonie)

Sei $(D, \sqsubseteq)$ eine partielle Ordnung und $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine Familie von Funktionen $f_k : D \to D$. Dann heißt $\mathcal{F}$ verzögert-monoton, falls für alle $k \in \mathbb{N}$ gilt:

$$d \sqsubseteq d' \text{ impliziert } \exists s \in \mathbb{N}^*. f_k(d) \sqsubseteq f_s(d')$$

Lemma 9.2.3

$\mathcal{F}$ ist verzögert-monoton, wenn alle f_k im üblichen Sinn monoton sind.

Das "monotoniefreie" Fixpunkttheorem

Theorem 9.2.4 (Chaotische Fixpunktiteration)

Sei $(D, \sqsubseteq)$ eine wohlfundierte partielle Ordnung mit kleinstem Element $\bot$, $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine verzögert-monotone Familie inflationärer Funktionen und $\gamma : \mathbb{N} \to \mathbb{N}$ eine faire Strategie.

Dann gilt:

1. Der kleinste gemeinsame Fixpunkt $\mu\mathcal{F}$ von $\mathcal{F}$ existiert und ist gegeben durch $\bigsqcup f_\gamma(\bot)$.
2. $\mu\mathcal{F}$ wird stets in einer endlichen Zahl von Iterationsschritten erreicht.

Generischer Fixpunktalgorithmus 9.2.5

Nichtdeterministischer 'Rumpf'-Algorithmus:

$d := \bot$;
while $\exists\, k \in \mathbb{N}.\ d \neq f_k(d)$ **do**
  choose $k \in \mathbb{N}$ **where** $d \sqsubset f_k(d)$ **in**
    $d := f_k(d)$
  ni
od

Kapitel 9.3

Anwendungen

Kapitel 9.3.1

Vektor-Iteration

Vorbereitungen (1)

- Sei $(C, \sqsubseteq_C)$ eine wohlfundierte partielle Ordnung und $D = C^n$ für ein $n \in \mathbb{N}$, geordnet durch die punktweise Ausdehnung von $\sqsubseteq_C$.
- Sei $f : D \to D$ eine monotone Funktion.
- Anstelle der Iteration $d_1 = f(\bot), d_2 = f(d_1), \ldots$ im Stil von Tarskis Fixpunkttheorem, können wir zu einer Zerlegung von f in seine Komponenten f^k übergehen, d.h. $f(d) = (f^1(d), \ldots, f^n(d))$ unter Ausführung selektiver Aktualisierungen
- Hier und in der Folge benutzen wir obere Indizes i, um die i-te Komponente eines Vektors der Länge n zu bezeichnen.

Vorbereitungen (2)

Definition 9.3.1.1 (Vektor-Iteration)

Eine Vektor-Iteration ist eine Iteration der Form $d_1 = f_{J_0}(\bot), d_2 = f_{J_1}(d_1), \ldots$, wobei $J_i \subseteq \{1, \ldots, n\}$ und

$$f_J(d)^i =_{df} \begin{cases} f^i(d) & \text{falls } i \in J \\ d^i & \text{sonst} \end{cases}$$

eine selektive Aktualisierung der durch J spezifizierten Komponenten durchführt.

Es gilt:

- 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.
- Jedes f_J is monoton, da f monoton ist.

Anwendung von Fixpunkttheorem 9.2.4

...zur Modellierung der Vektor-Iteration.

Erforderlich: Verallgemeinerung des Strategiebegriffs auf einen Strategiebegriff für Mengen.

Definition 9.3.1.2 ((Faire) Mengenstrategie)

- 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 $\{1, \ldots, n\}$, deren zugehörige Komponenten in Schritt i aktualisiert werden sollen.

- Eine Mengenstrategie heißt fair gdw

  $$\forall i \in \mathbb{N},\, J \subseteq \mathbb{N}.\ (f_J(d_i) \neq d_i \text{ impliziert } \exists j > i.\, d_j \neq d_i)$$

Modellierungsresultate (1)

Lemma 9.3.1.3 (Vektor-Iteration)

Sei $(C, \sqsubseteq_C)$ eine wohlfundierte partielle Ordnung mit kleinstem Element $\bot_C$, sei $n \in \mathbb{N}$ und sei $D = C^n$ geordnet durch die punktweise Ausdehnung von $\sqsubseteq_C$ auf $\sqsubseteq$. Sei $f = (f^1, \ldots, f^n)$ eine monotone Funktion auf D, sei $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{1, \ldots, n\}\}$ mit Funktionen $f_J : D \to D$ wie zuvor definiert und sei $\gamma : \mathbb{N} \to \mathcal{P}(\{1, \ldots, n\})$ eine Mengenstrategie.

Dann gilt: Jede chaotische Iteration $f_\gamma(\bot)$ liefert eine Kette.

Modellierungsresultate (2)

Korollar 9.3.1.4 (Chaotische Vektor-Iteration)

Sei $(C, \sqsubseteq_C)$ eine wohlfundierte partielle Ordnung mit kleinstem Element $\bot_C$, sei $n \in \mathbb{N}$ und sei $D = C^n$ geordnet durch die punktweise Erweiterung von $\sqsubseteq_C$ auf $\sqsubseteq$. Sei $f = (f^1, \ldots, f^n)$ eine monotone Function auf D, sei $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{1, \ldots, n\}\}$ und sei γ eine faire Mengenstrategie.

Dann gilt:

- $\bigsqcup f_\gamma(\bot)$ ist der kleinste Fixpunkt $\mu\mathcal{F}$ von $\mathcal{F}$.
- $\mu\mathcal{F} = \mu f$.
- $\mu\mathcal{F}$ ist stets in einer endlichen Zahl von Iterationsschritten erreicht.

Modellierungsresultate (3)

Bemerkungen:

- Korollar 9.3.1.4 ist ein Spezialfall von Fixpunkttheorem 9.2.4 für Vektor-Iterationen und folgt zusammen mit Lemma 9.3.1.3.
- Für $|\mathcal{F}| = 1$ reduziert sich Korollar 9.3.1.4 auf Tarskis Fixpunkttheorem im Fall wohlfundierter partieller Ordnungen.

Kapitel 9.3.2

Datenflussanalyse

Anwendung von Fixpunkttheorem 9.2.4

...auf intraprozedurale DFA.

Erinnerung:

Das *MaxFP*-Gleichungssystem:

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

Die *MaxFP*-Lösung:

Die größte Lösung des *MaxFP*-Gleichungssystems, bezeichnet mit $inf^*_{c_{\mathbf{s}}}$.

Dual dazu: *MinFP*-Gleichungssystem 9.3.2.1

Das *MinFP*-Gleichungssystem:

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

Die *MinFP*-Lösung:

Die kleinste Lösung des *MinFP*-Gleichungssystems, bezeichnet mit $inf^*_{c_{\mathbf{s}}}$.

Generischer *MinFP*-Fixpunktalg. 9.3.2.2

$inf[\mathbf{s}] := c_{\mathbf{s}}$;
forall $n \in N \backslash \{\mathbf{s}\}$ **do** $inf[n] := \bot$ **od**;
$workset := N$;
while $workset \neq \emptyset$ **do**
  choose $n \in workset$ **in**
    $workset := workset \backslash \{ n \}$;
    $new := inf[n] \sqcup \bigsqcup \{ [\![(m, n)]\!](\mathbf{inf}[m]) \mid m \in pred_G(n) \}$;
    if $new \sqsupset inf[n]$ **then**
      $inf[n] := new$;
      $workset := workset \cup succ_G(n)$
    fi
  ni
od

Zur Fixpunktcharakt. der *MinFP*-Lösung (1)

Vorbereitung

- Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ der betrachtete Flussgraph.
- Sei $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ ein monotones DFA-Funktional, das die lokale abstrakte Semantik von G festlegt.
- Die Menge der Knoten N werde mit der Menge der natürlichen Zahlen $\{1, \ldots, \mathtt{n}\}$ identifiziert, wobei $\mathtt{n}$ die Anzahl der Knoten von N bezeichnet.

Zur Fixpunktcharakt. der *MinFP*-Lösung (2)

Sei $D =_{df} \mathcal{C}^{\mathtt{n}}$ versehen mit der punktweisen Ausdehnung v. $\sqsubseteq_C$.

Dann gilt:

- $(D, \sqsubseteq)$ ist eine wohlfundierte partielle Ordnung.
- Ein Wert $d = (d^1, \ldots, d^{\mathtt{n}})$ stellt eine Annotation des Flussgraphen dar, wobei dem Knoten k der Wert d^k zugewiesen ist.

Zur Fixpunktcharakt. der *MinFP*-Lösung (3)

Für jeden Knoten k des Flussgraphen definieren wir jetzt eine Funktion $f^k : D \to C$ durch

$$f^k(d^1, \ldots, d^n) =_{df} d'^k$$

wobei

$$d'^k = d^k \sqcup \bigsqcup \{ [\![(m,k)]\!](d^m) \mid m \in pred_G(k) \}$$

Intuitiv: f^k beschreibt den Effekt der Berechnung der lokalen abstrakten Semantik am Knoten k.

Charakterisierungsresultate (1)

Lemma 9.3.2.3

Für alle $d \in D$ gilt: d ist eine Lösung des *MinFP*-Gleichungssystems gdw d ist Fixpunkt von $f =_{df} (f^1, \ldots, f^n)$.

Theorem 9.3.2.4 (Korrektheit und Terminierung)

Jeder Lauf des generischen *MinFP*-Fixpunktalgorithmus 9.3.2.2 terminiert mit der *MinFP*-Lösung.

Charakterisierungsresultate (2)

Anmerkungen zum Beweis von Theorem 9.3.2.4

- Der *MinFP*-Fixpunktalgorithmus 9.3.2.2 folgt dem Muster von Rumpfalgorithmus 9.2.5 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.
- Offenbar gilt: f ist monoton.
- Somit sind insgesamt die Voraussetzungen von Korollar 9.3.1.4 erfüllt, womit Theorem 9.3.2.4 folgt.

Ausblick

Weitere Anwendungen des “monotoniefreien” Fixpunkttheorems 9.2.4 in Kapitel 11 und 12 zum Beweis der Optimalität von

- Partially Dead-Code Elimination
- Partially Redundant-Assignment Elimination

Andere Fixpunkttheoreme sind dafür nicht anwendbar.

Kapitel 9.4

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 9 (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 9 (2)


F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, France, Dec. 1976.


Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5:285-309, 1955.


Franklyn Turbak, David Gifford with Mark A. Sheldon. *Design Concepts in Programming Languages*. MIT Press, 2008. (Kapitel 5, Fixed Points)

Kapitel 10

Basisblock- vs. Einzelanweisungsgraphen

Kapitel 10.1

Motivation

Basisblock- vs. Einzelanweisungsgraphen

In diesem Kapitel untersuchen wir die Zweckmäßigkeit unterschiedlicher Programmrepräsentationen.

Dazu betrachten und vergleichen wir Programme in Form von knoten- und kantenbenannten Flussgraphen mit Basisblöcken und Einzelanweisungen und untersuchen ihre jeweiligen

- Vor- und Nachteile für die Programmanalyse

...und gehen somit der Frage nach:

- Basisblock vs. Einzelanweisungsgraphen: Eine Geschmacksfrage?

Nebenbei werden wir kennenlernen:

- Einige weitere Beispiele konkreter Datenflussanalyseprobleme und Datenflussanalysen

Basisblöcke: Vermeintliche Vorteile

...und die ihnen allgemein zugeschriebenen Anwendungsvorteile ("Folk Knowledge"):

Bessere Skalierbarkeit, da

- weniger Knoten in die (potentiell) berechnungsaufwändige iterative Fixpunktberechnung involviert sind und somit
- größere Programme in den Hauptspeicher passen.

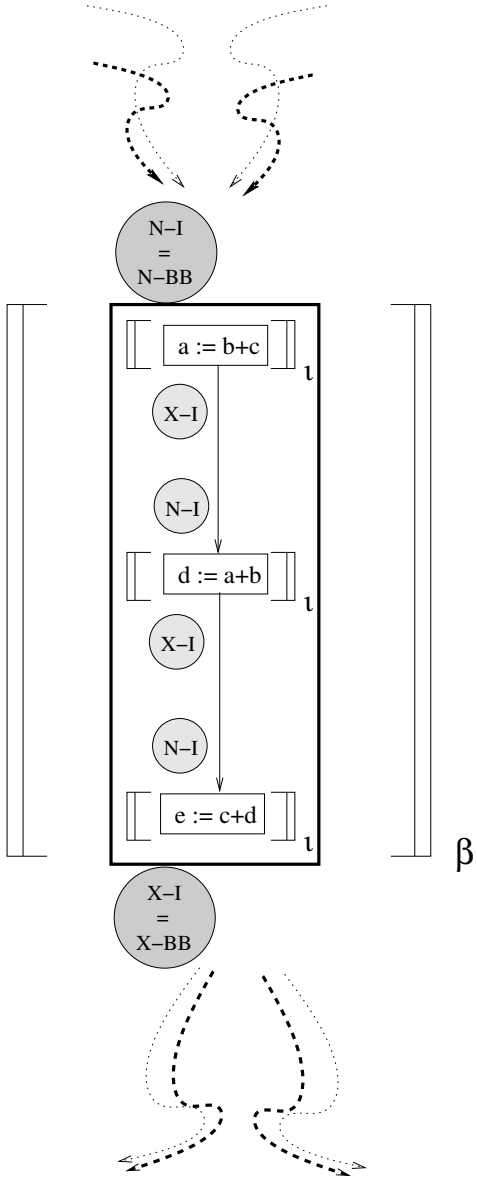
Basisblöcke: Sichere Nachteile

...und die nachfolgend gezeigten Anwendungsnachteile:

- 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 die hierarchie-induzierten Zusatzprobleme zu behandeln (z.B. für dead code elimination, constant propagation, ...); oder "trickbehaftete" Formulierungen nötig macht, um sie zu vermeiden (z.B. für 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. faint variable elimination).

Hierarchisierung durch Basisblöcke

Veranschaulichung:



In der Folge

Untersuchung von

- Vor- und Nachteilen von Basisblock- (BB) gegenüber Einzelanweisungsgraphen (EA)

anhand von Beispielen

- einiger bereits von uns betrachteter
  - Verfügbarkeit von Ausdrücken
  - Einfache Konstanten

  und neuer Datenflussanalyseprobleme
  - Schattenhafte (faint) Variablen

Kapitel 10.1.1

Kantenbenannter Einzelanweisungsansatz

MOP_{EA}-Ansatz

...für kantenbenannte Einzelanweisungsgraphen.

Die *MOP*-Lösung:

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MOP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \bigsqcap \{\, \llbracket\, p\,\rrbracket_\iota(c_s) \mid p \in \mathbf{P}_G[s, n] \,\}$$

$MaxFP_{EA}$-Ansatz

...für kantenbenannte Einzelanweisungsgraphen.

Die *MaxFP*-Lösung:

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \mathtt{inf}^*_{c_s}(n)$$

wobei $\mathtt{inf}^*_{c_s}$ die größte Lösung des *MaxFP*-Gleichungssystems bezeichnet:

$$inf(n) = \begin{cases} c_s & \text{falls } n = s \\ \bigsqcap \{\, \llbracket\, (m,n)\, \rrbracket_\iota(inf(m)) \mid m \in pred_G(n) \,\} & \text{sonst} \end{cases}$$

Kapitel 10.1.2

Knotenbenannter Basisblockansatz

Bezeichnungen

In der Folge bezeichnen wir

- Basisblockknoten mit fett gesetzten Buchstaben (**m**, **n**,...)
- Einzelanweisungsknoten mit normal gesetzten Buchstaben (m, n,...)

Weiters bezeichnen wir mit

- $[\![\ ]\!]_\beta$ und
- $[\![\ ]\!]_\iota$

(lokale) abstrakte Datenflussanalysefunktionale auf Basisblock- bzw. Einzelanweisungs-/Instruktions-Ebene.

MOP_{BB}-Ansatz (1)

...für knotenbenannte Basisblockgraphen.

Die *MOP*-Lösung auf BB-Ebene:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, \mathbf{n} \in \mathbf{N}.\ MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\ N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ X\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n})\ )$$

mit

$$N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\ \llbracket\ p\ \rrbracket_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s}, \mathbf{n}[\ \}$$

$$X\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\ \llbracket\ p\ \rrbracket_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s}, \mathbf{n}]\ \}$$

MOP_{BB}-Ansatz (2)

...und ihre Fortsetzung auf EA-Ebene:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathbf{s}})}(n) =_{df} \\ (\ N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathbf{s}})}(n),\ X\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathbf{s}})}(n)\ )$$

MOP_{BB}-Ansatz (3)

...mit

$$N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \begin{cases} N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_s)}(bb(n)) \\ \quad \text{falls } n = start(bb(n)) \\ \\ \llbracket\ p\ \rrbracket_\iota(N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_s)}(bb(n))) \\ \quad \text{sonst } (p \text{ Präfixpfad} \\ \quad \text{von } start(bb(n)) \\ \quad \text{bis (ausschließlich) } n) \end{cases}$$

$$X\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \llbracket\ p\ \rrbracket_\iota(N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_s)}(bb(n))) \\ (p \text{ Präfix von } start(bb(n)) \\ \text{bis (einschließlich) } n)$$

$MaxFP_{BB}$-Ansatz (1)

...für knotenbenannte Basisblockgraphen:

Die *MaxFP*-Lösung auf BB-Ebene:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\, \forall\, \mathbf{n} \in \mathbf{N}.\ MaxFP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\ N\text{-}MaxFP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ X\text{-}MaxFP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n})\ )$$

mit

$$N\text{-}MaxFP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \mathrm{pre}^{\beta}_{c_{\mathbf{s}}}(\mathbf{n}) \qquad \text{und}$$
$$X\text{-}MaxFP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \mathrm{post}^{\beta}_{c_{\mathbf{s}}}(\mathbf{n})$$

$MaxFP_{BB}$-Ansatz (2)

...wobei $pre^{\beta}_{c_s}$ und $post^{\beta}_{c_s}$ die größten Lösungen des Gleichungssystems

$$pre(\mathbf{n}) = \begin{cases} c_{\mathbf{s}} & \text{falls } \mathbf{n} = \mathbf{s} \\ \bigsqcap \{ post(\mathbf{m}) \mid \mathbf{m} \in pred_{\mathbf{G}}(\mathbf{n}) \} & \text{sonst} \end{cases}$$

$$post(\mathbf{n}) = [\![\mathbf{n}]\!]_{\beta}(pre(\mathbf{n}))$$

bezeichnen.

$MaxFP_{BB}$-Ansatz (3)

...und ihre Fortsetzung auf EA-Ebene:

$$\forall\, c_s \in \mathcal{C}\, \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \\ (\ N\text{-}MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n),\ X\text{-}MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n)\ )$$

mit

$$N\text{-}MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \mathrm{pre}^\iota_{c_s}(n) \qquad \text{und}$$
$$X\text{-}MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \mathrm{post}^\iota_{c_s}(n)$$

BB-*MaxFP*-Ansatz (4)

...wobei $pre^{\iota}_{c_s}$ und $post^{\iota}_{c_s}$ die größten Lösungen des Gleichungssystems

$$pre(n) = \begin{cases} pre^{\beta}_{c_s}(bb(n)) \\ \quad \text{falls } n = start(bb(n)) \\ \\ post(m) \\ \quad \text{sonst } (m \text{ ist hier der eindeutig} \\ \quad \text{bestimmte Vorgänger von } n \\ \quad \text{in } bb(n)) \end{cases}$$

$$post(n) = [\![n]\!]_{\iota}(pre(n))$$

bezeichnen.

Kapitel 10.2

Verfügbarkeit von Ausdrücken

Kapitel 10.2.1

Knotenbenannter Basisblockansatz

Verfügbarkeit von Ausdrücken (1)

...für knotenbenannte BB-Graphen.

Phase I: Die Basisblockebene

Lokale Prädikate (assoziiert mit BB-Knoten):

- BB-XCOMP$_\beta(t)$: β enthält eine Anweisung ι, die t berechnet, und weder ι noch eine auf ι folgende Anweisung in β modifiziert einen Operanden von t.
- BB-TRANSP$_\beta(t)$: β enthält keine Anweisung, die einen Operanden von t modifiziert.

Verfügbarkeit von Ausdrücken (2)

Das BB-Gleichungssystem von Phase I:

$$\text{BB-N-AVAIL}_\beta = \begin{cases} \textbf{falsch} & \text{falls } \beta = \textbf{s} \\ \prod_{\hat{\beta} \in \mathit{pred}(\beta)} \text{BB-X-AVAIL}_{\hat{\beta}} & \text{sonst} \end{cases}$$

$$\text{BB-X-AVAIL}_\beta = \text{BB-N-AVAIL}_\beta \cdot \text{BB-TRANSP}_\beta + \text{BB-XCOMP}_\beta$$

Verfügbarkeit von Ausdrücken (3)

Phase II: Die Anweisungsebene

Lokale Prädikate (assoziiert mit EA-Knoten):

- $\mathrm{COMP}_\iota(t)$: ι berechnet t.
- $\mathrm{TRANSP}_\iota(t)$: ι modifiziert keinen Operanden von t.
- BB-N-AVAIL*, BB-X-AVAIL*: größte Lösung des BB-Gleichungssystem von Phase I.

Das EA-Gleichungssystem von Phase II:

$$\text{N-AVAIL}_\iota = \begin{cases} \text{BB-N-AVAIL}^\star_{bb(\iota)} & \text{falls } \iota = start(bb(\iota)) \\ \text{X-AVAIL}_{pred(\iota)} & \text{sonst} \\ & (\text{beachte: } |pred(\iota)| = 1) \end{cases}$$

$$\text{X-AVAIL}_\iota = \begin{cases} \text{BB-X-AVAIL}^\star_{bb(\iota)} & \text{falls } \iota = end(bb(\iota)) \\ (\text{N-AVAIL}_\iota + \text{COMP}_\iota) \cdot \text{TRANSP}_\iota & \text{sonst} \end{cases}$$

Kapitel 10.2.2

Knotenbenannter Einzelanweisungsansatz

Verfügbarkeit von Ausdrücken

...für knotenbenannte EA-Graphen.

Lokale Prädikate (assoziiert mit EA-Knoten):

- $\text{COMP}_\iota(t)$: ι berechnet t.
- $\text{TRANSP}_\iota(t)$: ι modifiziert keinen Operanden von t.

Das EA-Gleichungssystem:

$$\text{N-AVAIL}_\iota = \begin{cases} \textbf{falsch} & \text{falls } \iota = s \\ \prod_{\hat{\iota} \in pred(\iota)} \text{X-AVAIL}_{\hat{\iota}} & \text{sonst} \end{cases}$$

$$\text{X-AVAIL}_\iota = (\text{N-AVAIL}_\iota + \text{COMP}_\iota) \cdot \text{TRANSP}_\iota$$

Kapitel 10.2.3

Kantenbenannter Einzelanweisungsansatz

Verfügbarkeit von Ausdrücken

...für kantenbenannte EA-Graphen.

Lokale Prädikate (assoziiert mit EA-Kanten):

- $\mathrm{COMP}_{\varepsilon}(t)$: Anweisung ι von Kante ε berechnet t.
- $\mathrm{TRANSP}_{\varepsilon}(t)$: Anweisung ι von Kante ε modifiziert keinen Operanden von t.

Das EA-Gleichungssystem:

$$\mathrm{Avail}_n = \begin{cases} \textbf{falsch} & \text{falls } n = s \\ \prod_{m \in pred(n)} (\mathrm{Avail}_m + \mathrm{COMP}_{(m,n)}) \cdot \mathrm{TRANSP}_{(m,n)} & \\ & \text{sonst} \end{cases}$$

In den folgenden Kapiteln

...zwei weitere Beispiele zur Veranschaulichung des Einflusses der Flussgraphdarstellungsvariante:

- Konstantenausbreitung und -faltung (Constant Propagation and Folding)
- Schattenvariablenelimination (Faint Variable Elimination)

Dabei Formulierung dieser Probleme für die Varianten:

- knotenbenannte Basisblockgraphen
- kantenbenannte Einzelanweisungsgraphen

Kapitel 10.3

Konstantenausbreitung und -faltung

Konstantenausbreitung und -faltung

... am Beispiel "Einfacher Konstanten".

Wir benötigen dazu zwei Hilfsfunktionen:

- Rückwärtssubstitution
- Zustandstransformation(sfunktion)

Rückwärtssubstitution und Zustandstransformation auf Anweisungen

Sei $\iota \equiv (x := t)$ eine Anweisung. Dann definieren wir:

- Rückwärtssubstitution

  $\delta_\iota : \mathbf{T} \to \mathbf{T}$ durch $\delta_\iota(s) =_{df} s[t/x]$ für alle $s \in \mathbf{T}$, wobei $s[t/x]$ die simultane Ersetzung aller Vorkommen von x in s durch t bezeichnet.

- Zustandstransformation

$$\theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

Zusammenhang von δ und θ

Bezeichne $\mathcal{I}$ die Menge aller Anweisungen.

Lemma 10.3.1 (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.

Kapitel 10.3.1

Kantenbenannter Einzelanweisungsansatz

Einfache Konstanten für den EA-Fall

...für kantenbenannte Einzelanweisungsgraphen.

- $\mathrm{CP}_n \in \Sigma$
- $\sigma_0 \in \Sigma$ Anfangszusicherung

Das EA-Gleichungssystem:

$$\forall\, v \in \mathbf{V}.\ \mathrm{CP}_n = \begin{cases} \sigma_0(v) & \text{falls } n = s \\ \prod\{\, \mathcal{E}(\delta_{(m,n)}(v))(\mathrm{CP}_m) \mid m \in \mathit{pred}(n) \,\} & \text{sonst} \end{cases}$$

Kapitel 10.3.2

Knotenbenannter Basisblockansatz

Rückwärtssubstitution und Zustandstransformation auf Pfaden

Ausdehnung von δ und θ auf Pfade (und somit insbesondere auch auf Basisblöcke):

- $\Delta_p : \mathbf{T} \to \mathbf{T}$ definiert durch $\Delta_p =_{df} \delta_{n_q}$ für $q = 1$ und durch $\Delta_{(n_1,\ldots,n_{q-1})} \circ \delta_{n_q}$ für $q > 1$
- $\Theta_p : \Sigma \to \Sigma$ definiert durch $\Theta_p =_{df} \theta_{n_1}$ für $q = 1$ und durch $\Theta_{(n_2,\ldots,n_q)} \circ \theta_{n_1}$ für $q > 1$.

Zusammenhang von Δ und Θ

Bezeichne $\mathcal{B}$ die Menge aller Basisblöcke.

Lemma 10.3.2.1 (Verallgemeinertes 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 für den BB-Fall (1)

...für knotenbenannte Basisblockgraphen.

Phase I: Basisblockebene

Bemerkung:

- $\Delta_\beta(v) =_{df} \delta_{\iota_1} \circ \ldots \circ \delta_{\iota_q}(v)$, wobei $\beta \equiv \iota_1; \ldots; \iota_q$.
- BB-N-CP$_\beta$, BB-X-CP$_\beta$, N-CP$_\iota$, X-CP$_\iota \in \Sigma$
- $\sigma_0 \in \Sigma$ Anfangszusicherung

Einfache Konstanten für den BB-Fall (2)

Das BB-Gleichungssystem von Phase I:

$$\text{BB-N-CP}_\beta = \begin{cases} \sigma_0 & \text{falls } \beta = \mathbf{s} \\ \bigsqcap\{\text{BB-X-CP}_{\hat\beta} \mid \hat\beta \in \mathit{pred}(\beta)\} & \\ & \text{sonst} \end{cases}$$

$$\forall\, v \in \mathbf{V}.\ \text{BB-X-CP}_\beta(v) = \mathcal{E}(\Delta_\beta(v))(\text{BB-N-CP}_\beta)$$

Einfache Konstanten für den BB-Fall (3)

Phase II: Anweisungsebene

Vorberechnete Resultate (aus Phase I):

- BB-N-CP*, BB-X-CP*: die größte Lösung des Gleichungssystems von Phase I.

Einfache Konstanten für den BB-Fall (4)

Das EA-Gleichungssystem von Phase II:

$$\text{N-CP}_\iota \quad = \quad \begin{cases} \text{BB-N-CP}^\star_{bb(\iota)} \\ \quad \text{falls } \iota = start(bb(\iota)) \\ \text{X-CP}_{pred(\iota)} \\ \quad \text{sonst (beachte: } |pred(\iota)| = 1) \end{cases}$$

$$\forall\, v \in \mathbf{V}.\ \text{X-CP}_\iota(v) \quad = \quad \begin{cases} \text{BB-X-CP}^\star_{bb(\iota)}(v) \\ \quad \text{falls } \iota = end(bb(\iota)) \\ \mathcal{E}(\delta_\iota(v))(\text{N-CP}_\iota) \quad \text{sonst} \end{cases}$$

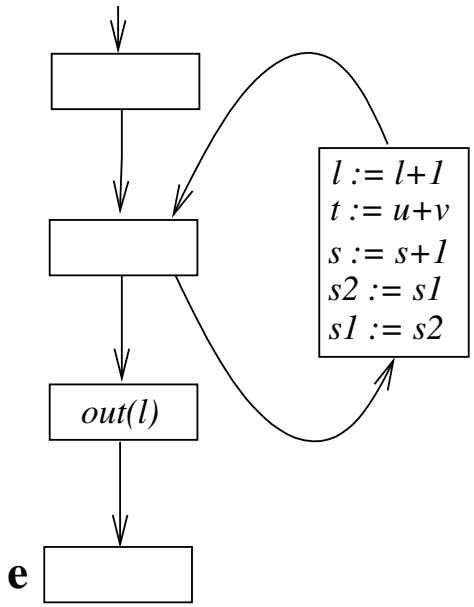
Kapitel 10.4

Schattenvariablen

Schattenvariablen

Anweisung

- $l := l + 1$ ist lebendig.
- $t := u + v$ ist tot.
- $s := s + 1$ sowie $s1 := s2;\ s2 := s1$ sind lebendig, aber schattenhaft (faint).



- faint: schwach, kraftlos, ohnmächtig
  $\rightsquigarrow$ "ein Schatten seinerselbst" $\rightsquigarrow$ Schattenvariable

Schattenvariablenanalyse (1)

...für kantenbenannte Einzelanweisungsgraphen.

Lokale Prädikate (assoziiert mit Einzelanweisungskanten):

- $\text{USED}_\varepsilon(v)$: Anweisung ι von Kante ε benutzt v.
- $\text{MOD}_\varepsilon(v)$: Anweisung ι von Kante ε modifiziert v.
- $\text{Rel-Used}_\varepsilon(v)$: v ist eine Variable, die in der Anweisung ι von Kante ε vorkommt und von dieser Anweisung "zu leben gezwungen" wird (z.B. für ι Ausgabeanweisung).
- $\text{Ass-Used}_\varepsilon(v)$: v ist eine in der Zuweisung ι von Kante ε rechtsseitig vorkommende Variable.

Schattenvariablenanalyse (2)

Das EA-Gleichungssystem:

$$\mathrm{FAINT}_n(v) = \prod_{m \in succ(n)} \overline{\mathrm{Rel\text{-}Used}_{(n,m)}(v)} * (\mathrm{FAINT}_m(v) + \mathrm{MOD}_{(n,m)}(v)) * (\mathrm{FAINT}_m(\mathit{LhsVar}_{(n,m)}) + \overline{\mathrm{Ass\text{-}Used}_{(n,m)}(v)})$$

Schattenvariablenanalyse (3)

...ein Beispiel für ein DFA-Problem, für das eine Formulierung

- auf (knoten- und kantenbenannten) Einzelanweisungs-graphen offensichtlich ist,
- auf (knoten- und kantenbenannten) Basisblockgraphen alles andere als ersichtlich, nicht möglich ist.

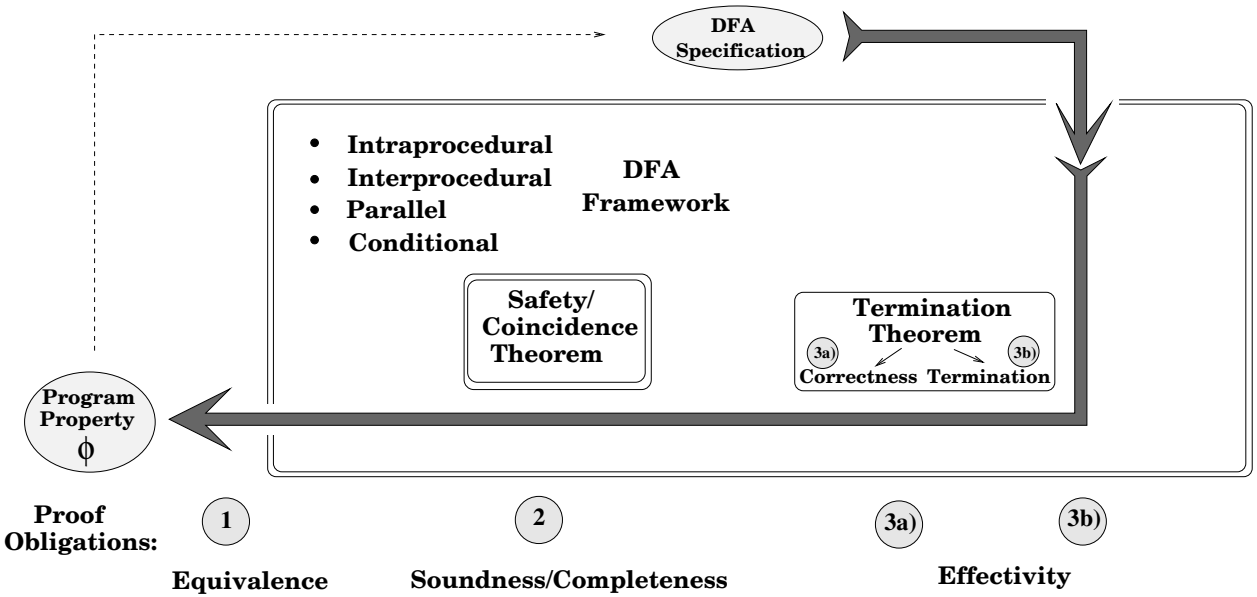
Kapitel 10.5

Fazit

Fazit

Alle 4 Flussgraphrepräsentationen sind grundsätzlich gleichwertig.

Konzeptuell reicht deshalb die allgemeine Rahmen- bzw. Werkzeugkistensicht



und das Wissen, dass sie je nach Aufgabe unterschiedlich zweckmäßig sind und unterschiedlich aufwändige Spezifikations-, Implementierungs- und Beweisverpflichtungen bewirken.

Kapitel 10.6

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 10

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.

Teil III

Transformation und Optimalität

Intuition

$$\text{Optimalität} = \text{Korrektheit} + \text{Vollständigkeit}$$

Kapitel 11

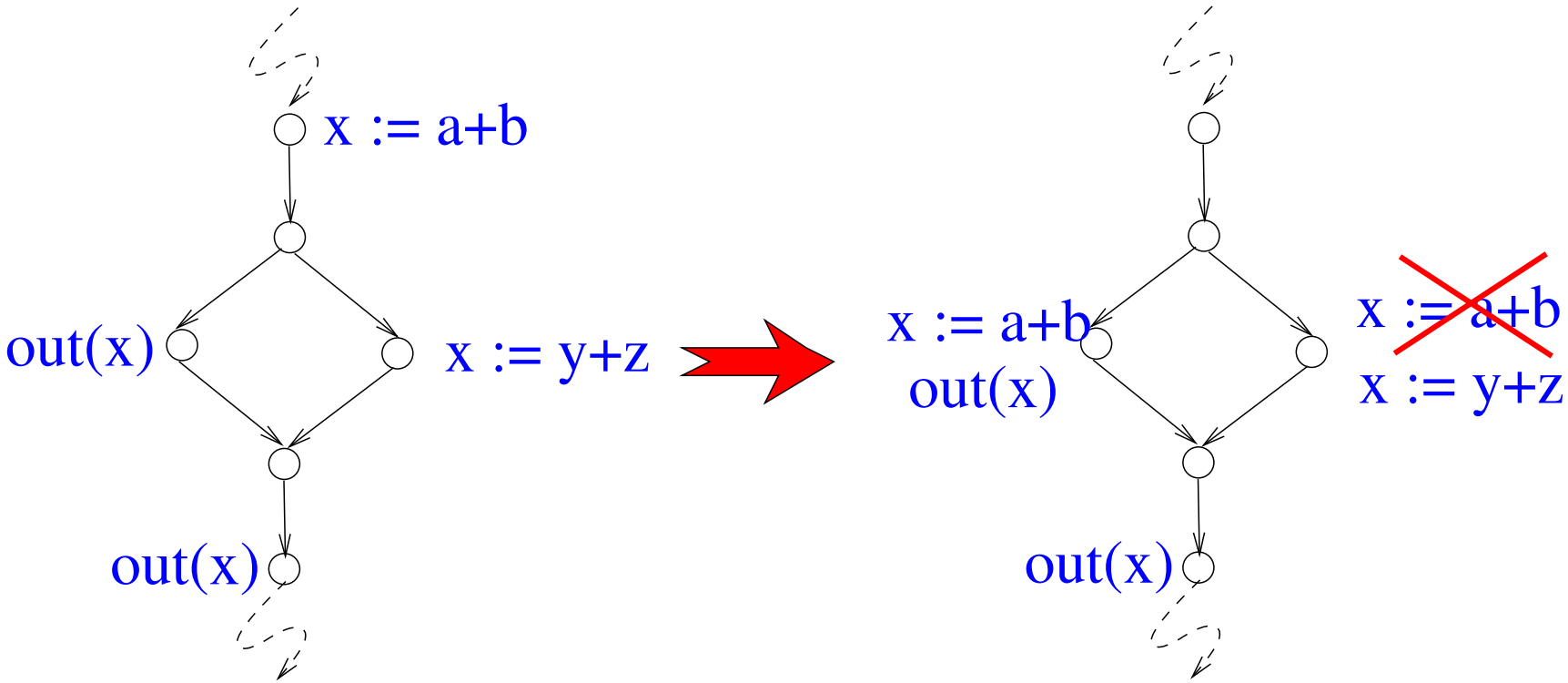
Elimination partiell toter Anweisungen

Kapitel 11.1

Motivation

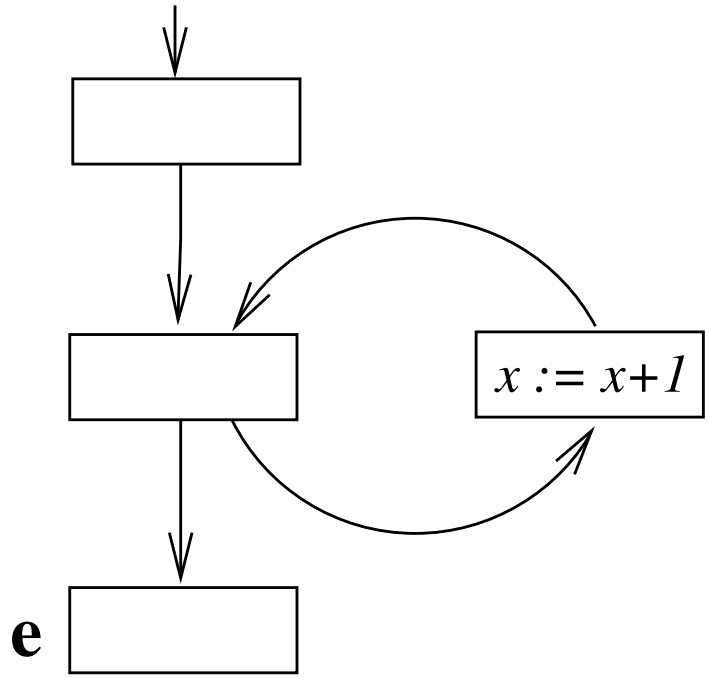
Elimination partiell toter Anweisungen

Veranschaulichendes Beispiel:



Schattenhafter Code

Die Anweisung $x := x + 1$ ist schattenhaft, aber nicht tot.



Elimination partiell toten/schattenhaften Codes

Toter und schattenhafter Code induzieren zwei unterschiedliche (Optimierungs-) Transformationen:

- Elimination partiell toten Codes

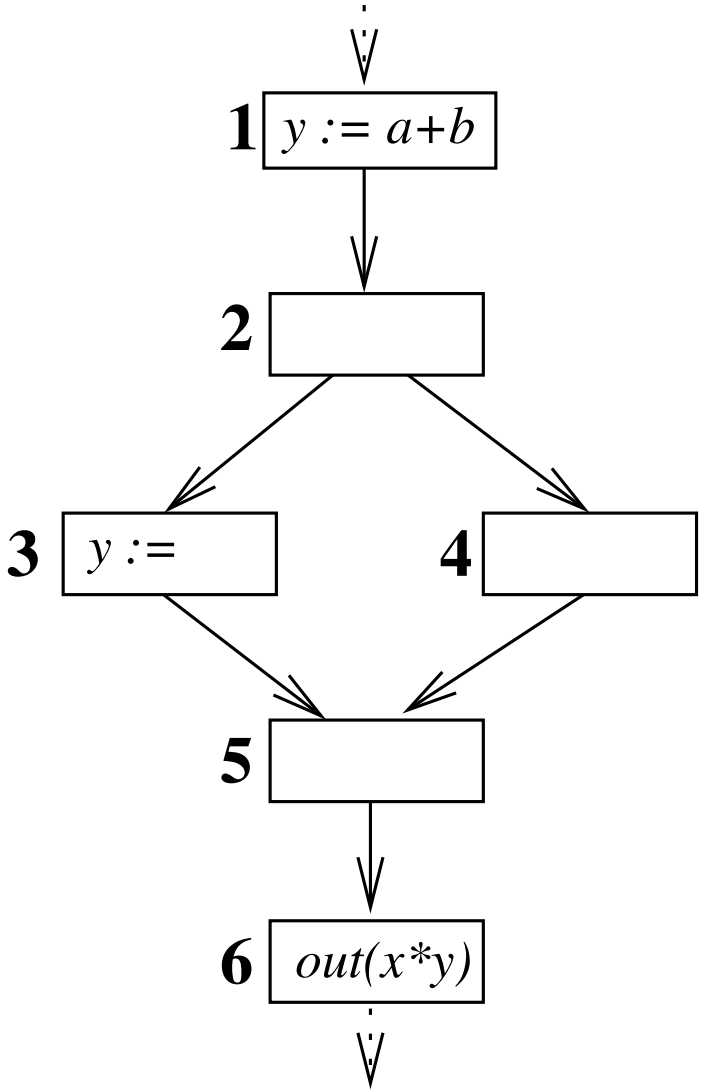
  $\rightsquigarrow$ Partial Dead-Code Elimination (PDCE)

- Elimination partiell schattenhaften Codes

  $\rightsquigarrow$ Partial Faint-Code Elimination (PFCE)

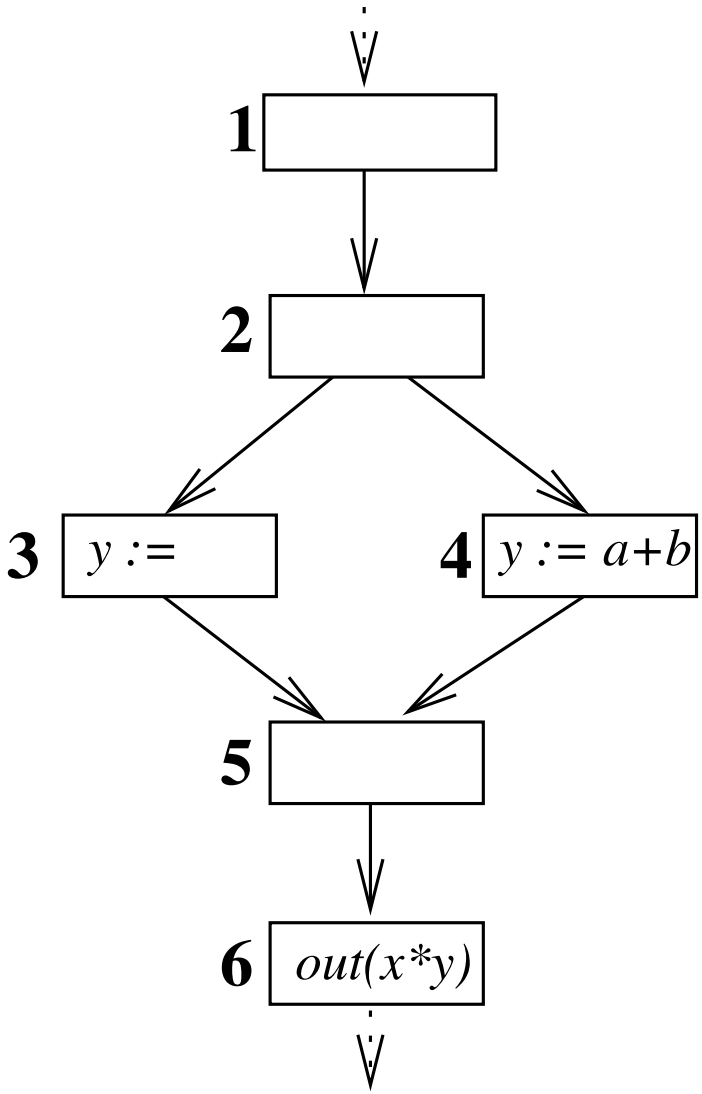
Elimination partiell toten Codes (1)

Ausgangsprogramm:



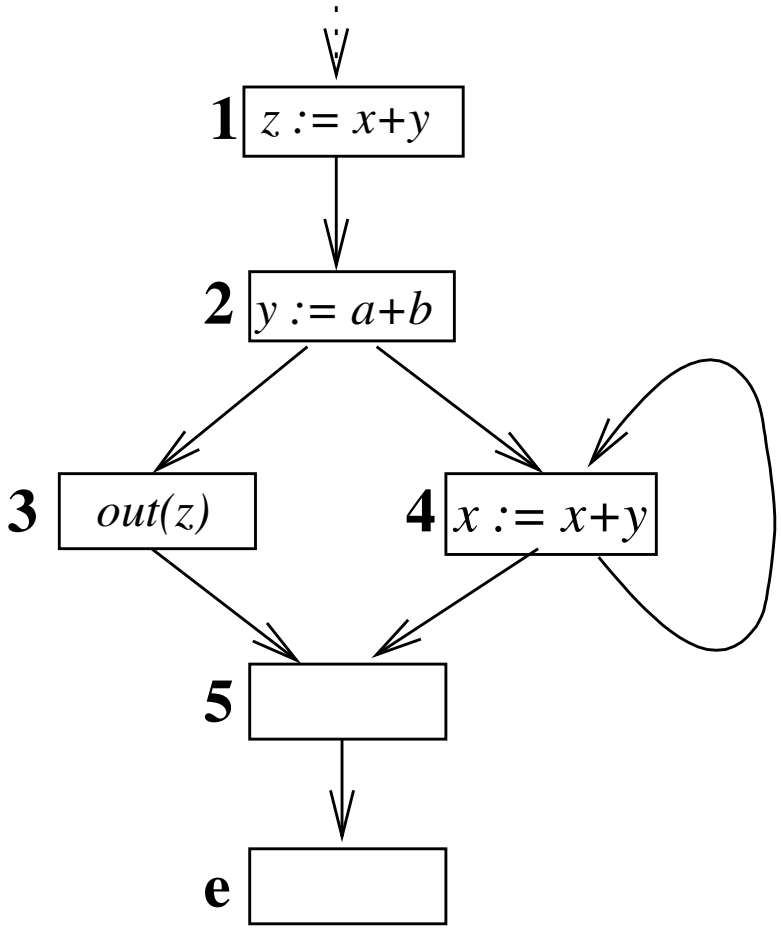
Elimination partiell toten Codes (2)

Transformiertes/optimiertes Programm:



Elimination partiell schattenhaften Codes (1)

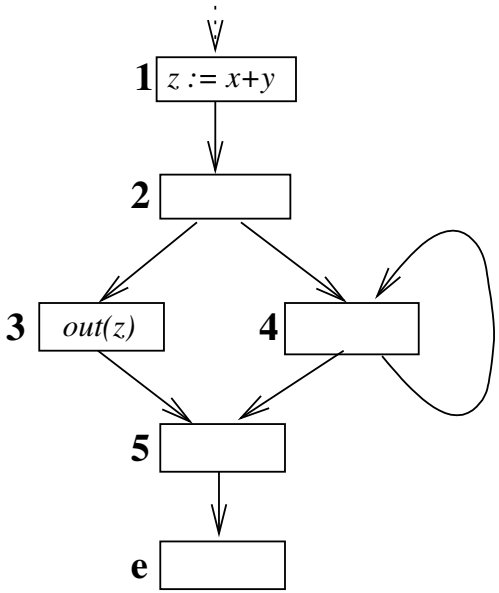
Ausgangsprogramm:



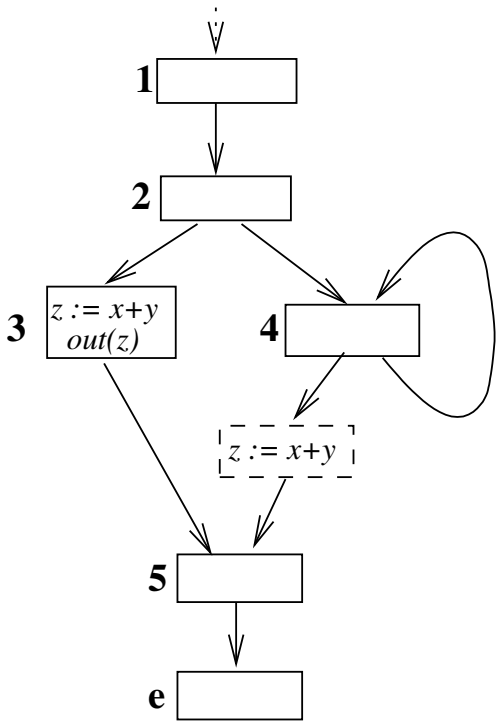
Elimination partiell schattenhaften Codes (2)

Transformiertes/optimiertes Programm:

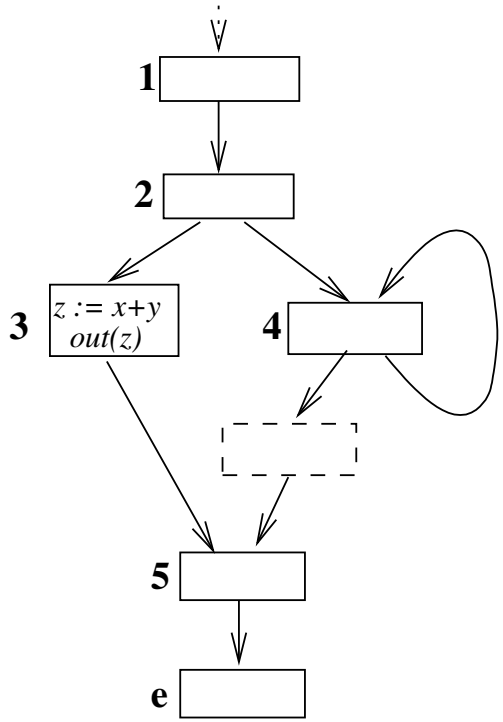
Elimination schattenhaften Codes



Anweisungssenkung



Elimination toten Codes



Elimination partiell schattenhaften Codes (3)

Beachte:

- "Echt" partiell schattenhaften Code gibt es nicht.

Die Elimination schattenhaften Codes

- kann aber durch die Beseitigung von Codesenkungs-blockaden die Elimination weiteren partiell toten Codes ermöglichen.

In diesem Sinne ist

- Elimination partiell schattenhaften Codes

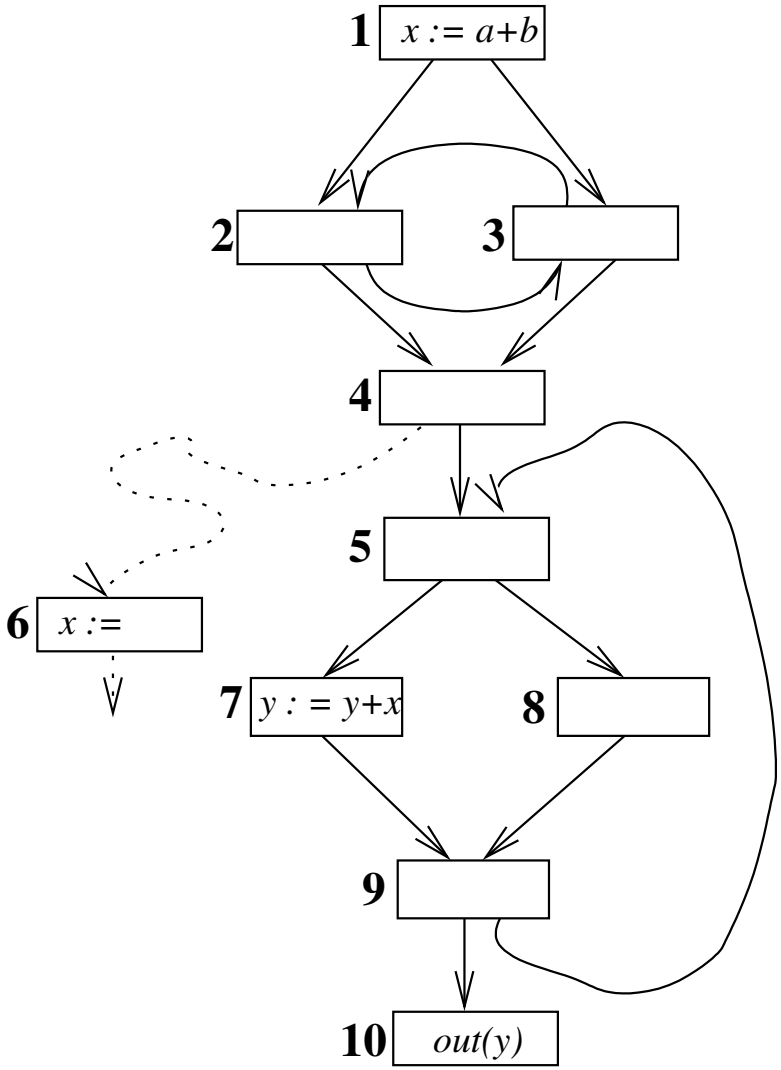
zu verstehen.

Kapitel 11.2

PDCE/PFCE: Transformation und Optimalität

Kritische Kanten (1)

...be-/verhindern die Transformation:

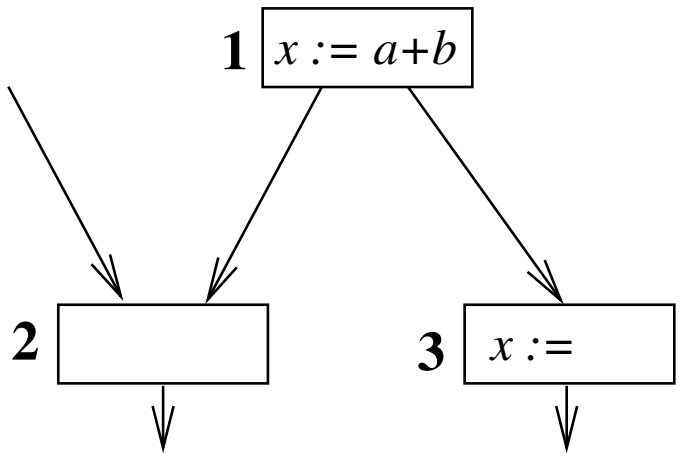


Kritische Kanten (2)

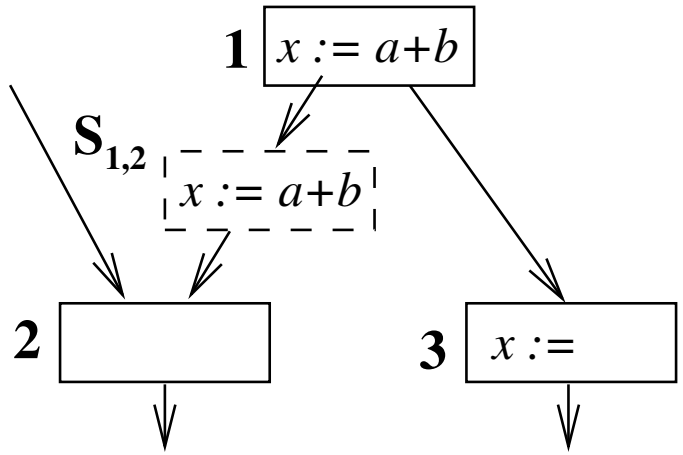
Definition 11.2.1 (Kritische Kanten)

Eine Kante heißt kritisch gdw sie von einem Knoten mit mehr als einem Nachfolger zu einem Knoten mit mehr als einem Vorgänger führt.

a)



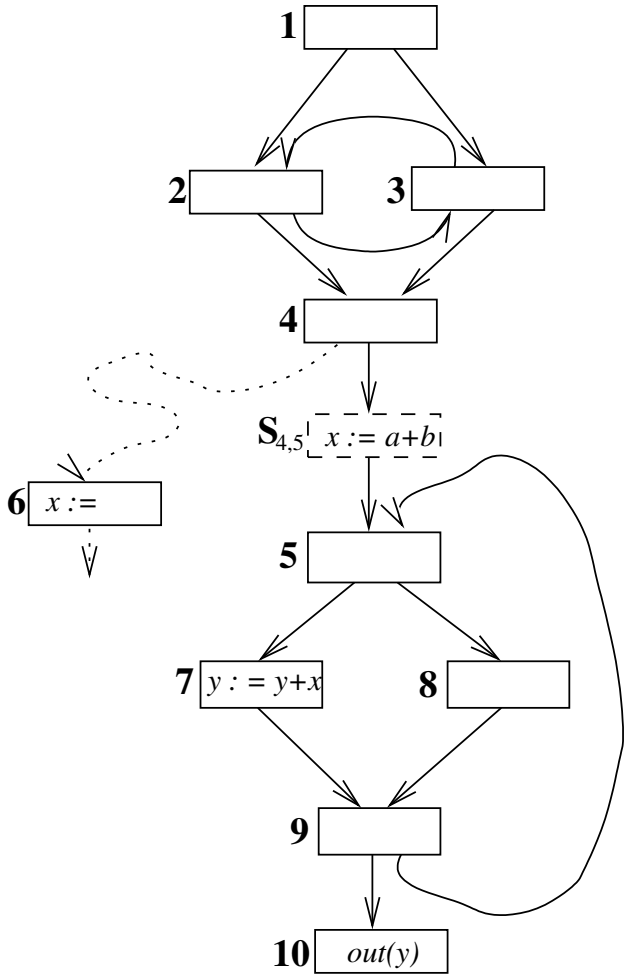
b)



Kritische Kanten (3)

Für bestmögliche Transformationsresultate, insbesondere kein Schieben von Anweisungen in Schleifen:

- Kritische Kanten spalten!



Anweisungsmuster

In der Folge bezeichne:

- α ein sog. Anweisungsmuster: $\alpha \equiv x := t$
- $\mathcal{AP}$ die Menge aller Anweisungsmuster

Senken von Anweisungen

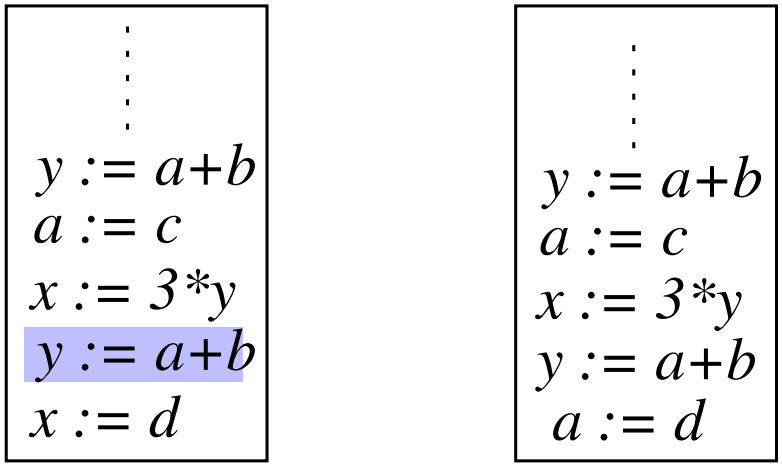
Definition 11.2.2 (Anweisungssenkung)

Eine α-Anweisungssenkung (assignment sinking) ist eine Programmtransformation, die

- einige α-Vorkommen eliminiert,
- neue α-Vorkommen am Eingang oder Ausgang einiger von einem Basisblock mit einem eliminierten α-Vorkommen aus erreichbaren Basisblöcke einsetzt.

Senkungskandidat

...in Basisblöcken:



Beachte:

- Nur das markierte Vorkommen von $y := a + b$ ist ein Senkungskandidat; die drei anderen Vorkommen von $y := a + b$ sind lokal blockiert.

Lokale Barrieren

Definition 11.2.3 (Lokale Barrieren)

Eine α-Anweisungssenkung wird blockiert von einer Anweisung, die

- einen Operanden von t modifiziert oder
- die Variable x liest oder modifiziert.

Zulässiges Senken von Anweisungen

Definition 11.2.4 (Zulässige Anweisungssenkung)

Ein α-Anweisungssenkung ist zulässig gdw

1. Die eliminierten α-Vorkommen werden ersetzt, d.h. auf jedem von einem Knoten n mit eliminiertem α-Vorkommen ausgehenden Pfad zum Endknoten **e** gibt es einen Knoten m, an dem ein neues α-Vorkommen eingesetzt wird und α von keiner Anweisung zwischen n and m blockiert ist.
2. Jedes neue Vorkommen von α ist gerechtfertigt, d.h., auf jedem vom Startknoten **s** aus einen Knoten n mit neu eingesetztem α-Vorkommen erreichenden Pfad gibt es einen Knoten m, an dem ein (ursprüngliches) α-Vorkommen eliminiert worden ist und α von keiner Anweisung zwischen m and n blockiert ist.

Elimination von Anweisungen

Definition 11.2.5 (Anweisungselimination)

Eine α-Anweisungselimination (assignment elimination) ist eine Programmtransformation, die einige α-Vorkommen aus dem Argumentprogramm streicht.

Zulässige Elimination von Anweisungen

Definition 11.2.6 (Zulässige Anweisungselimination)

Eine α-Anweisungselimination ist zulässig gdw sie streicht einige

- tote
- schattenhafte

α-Vorkommen aus dem Argumentprogramm.

Wir bezeichnen diese beiden Transformationen als

- Elimination toten Codes

  $\rightsquigarrow$ Dead-Code Elimination (DCE)
- Elimination schattenhaften Codes

  $\rightsquigarrow$ Faint-Code Elimination (FCE)

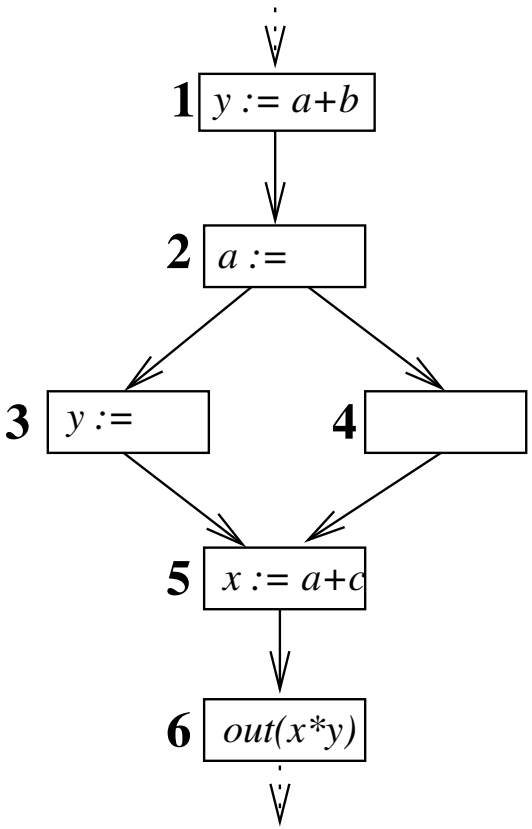
Effekte zweiter Ordnung

Effekte zweiter Ordnung (engl. second-order effects):

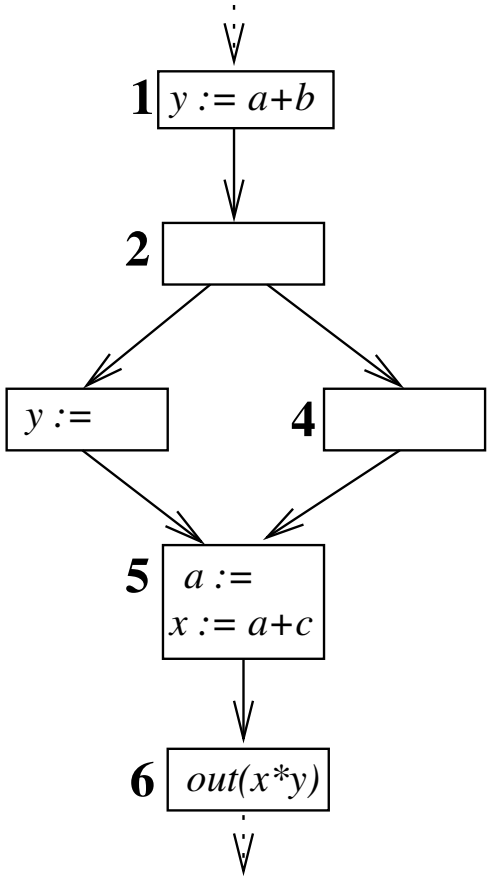
- Senkungs-Eliminations-Effekte (Zieleffekt)
  $\rightsquigarrow$ Sinking-Elimination effects (SE)
- Senkungs-Senkungs-Effekte
  $\rightsquigarrow$ Sinking-Sinking effects (SS)
- Eliminations-Senkungs-Effekte
  $\rightsquigarrow$ Elimination-Sinking effects (ES)
- Eliminations-Eliminations-Effekte (Zieleffekt)
  $\rightsquigarrow$ Elimination-Elimination effects (EE)

Beispiel eines Senkungs-Senkungs-Effekts

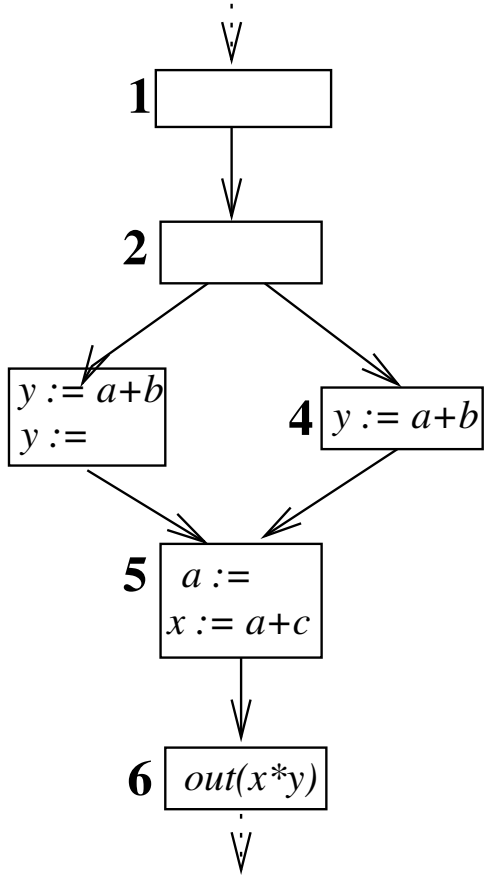
Ausgangsprogramm



1. Senkung

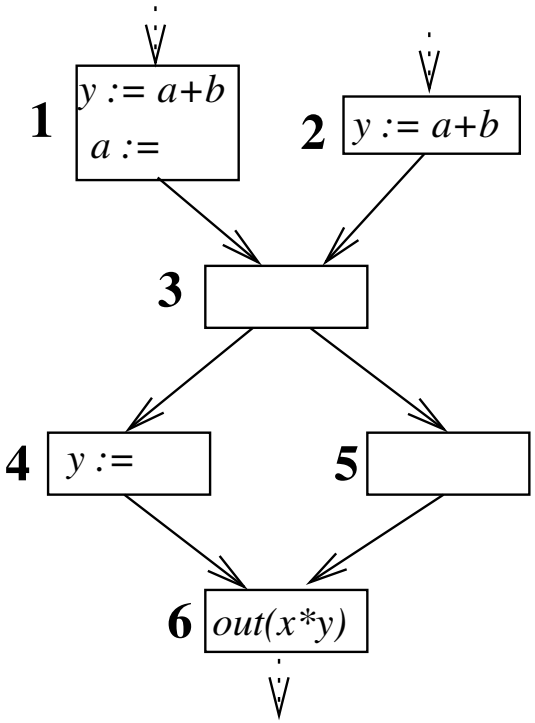


2. Senkung

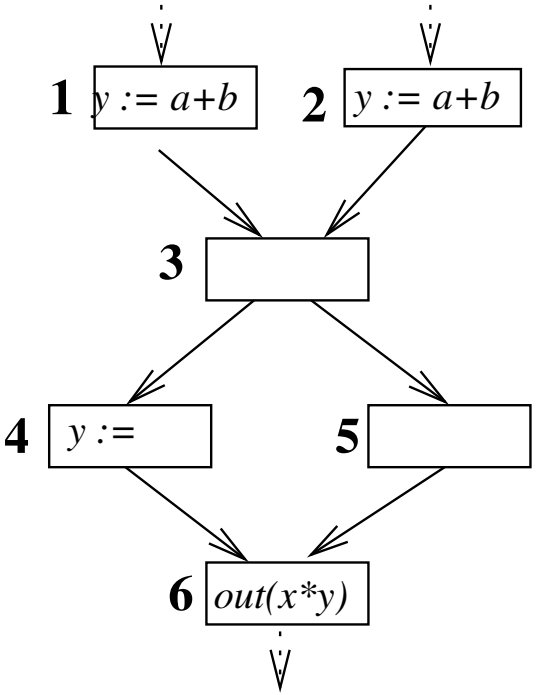


Beispiel eines Eliminations-Senkungs-Effekt

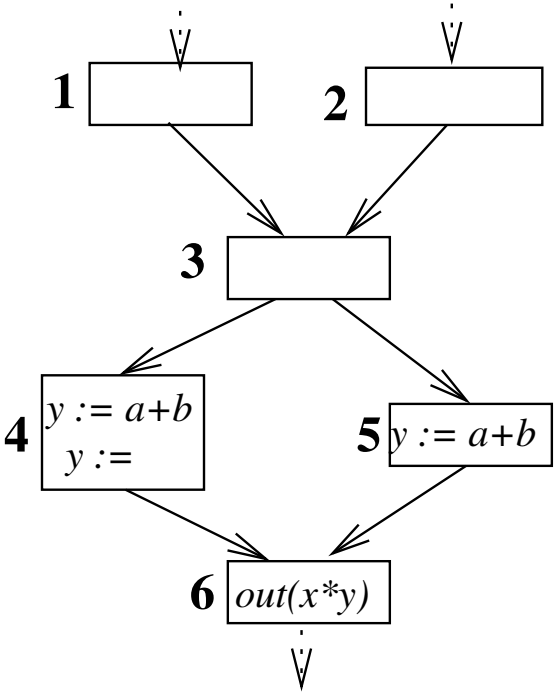
Ausgangsprogramm



Elimination

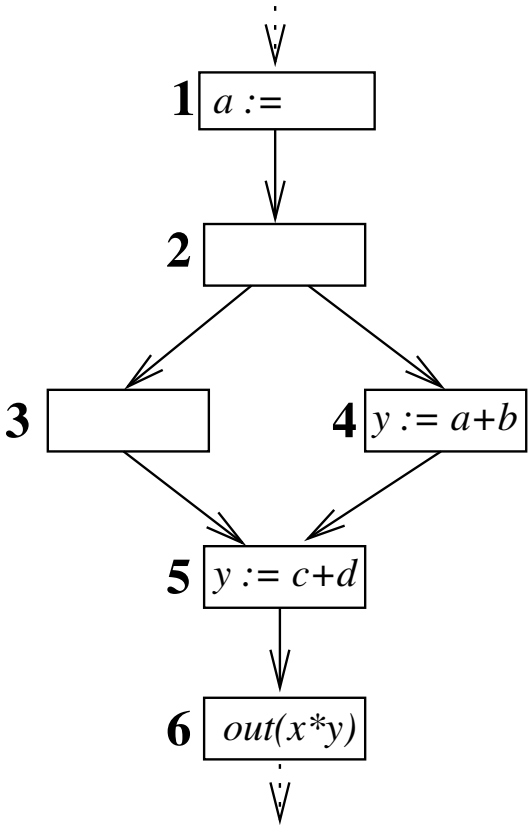


Senkung

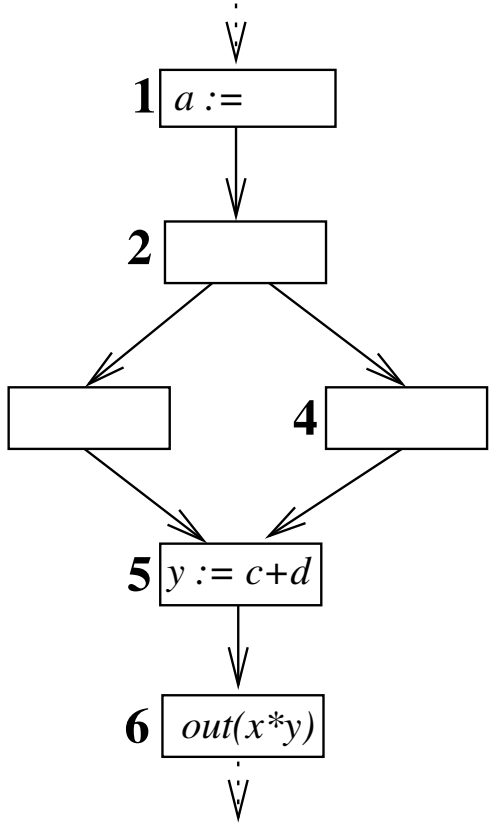


Beispiel eines Eliminations-Eliminations-Effekt

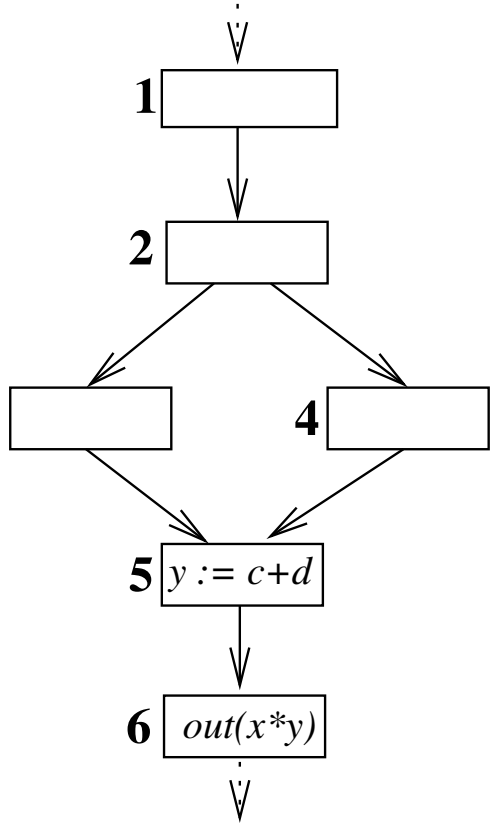
Ausgangsprogramm



1. Elimination

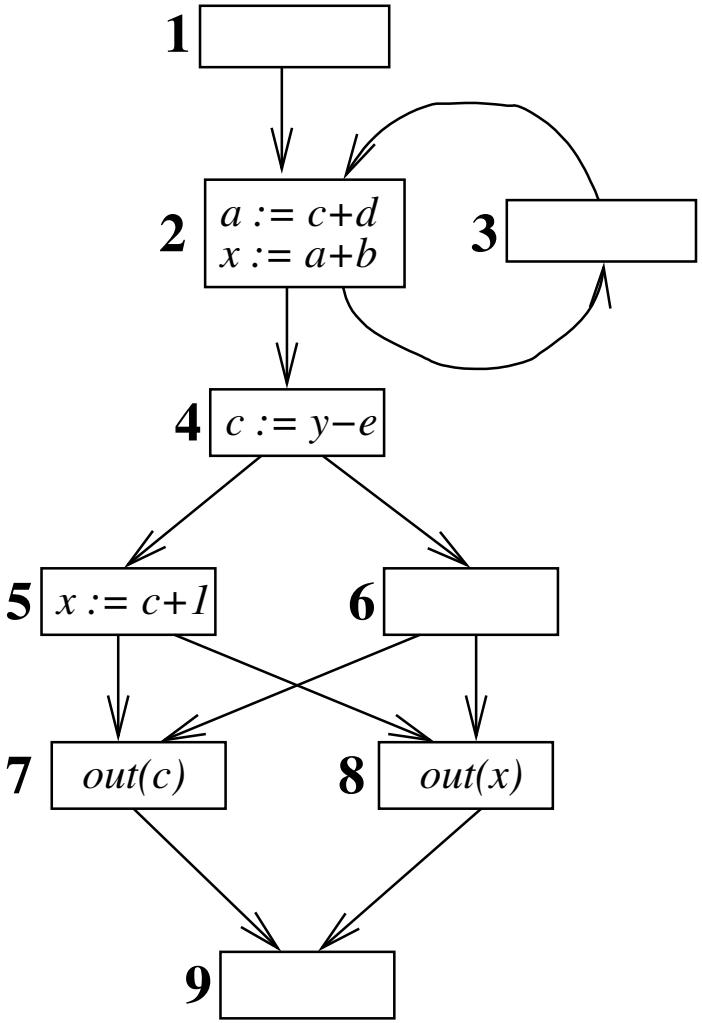


2. Elimination



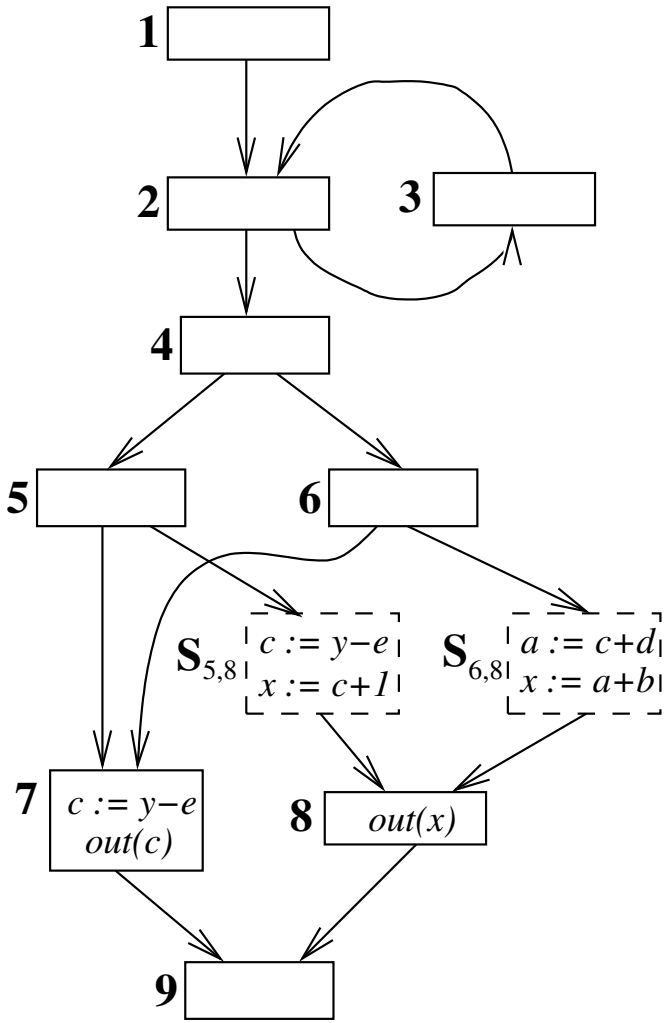
Kombinationswirkung von Effekten zweiter Ordnung (1)

Ausgangsprogramm:



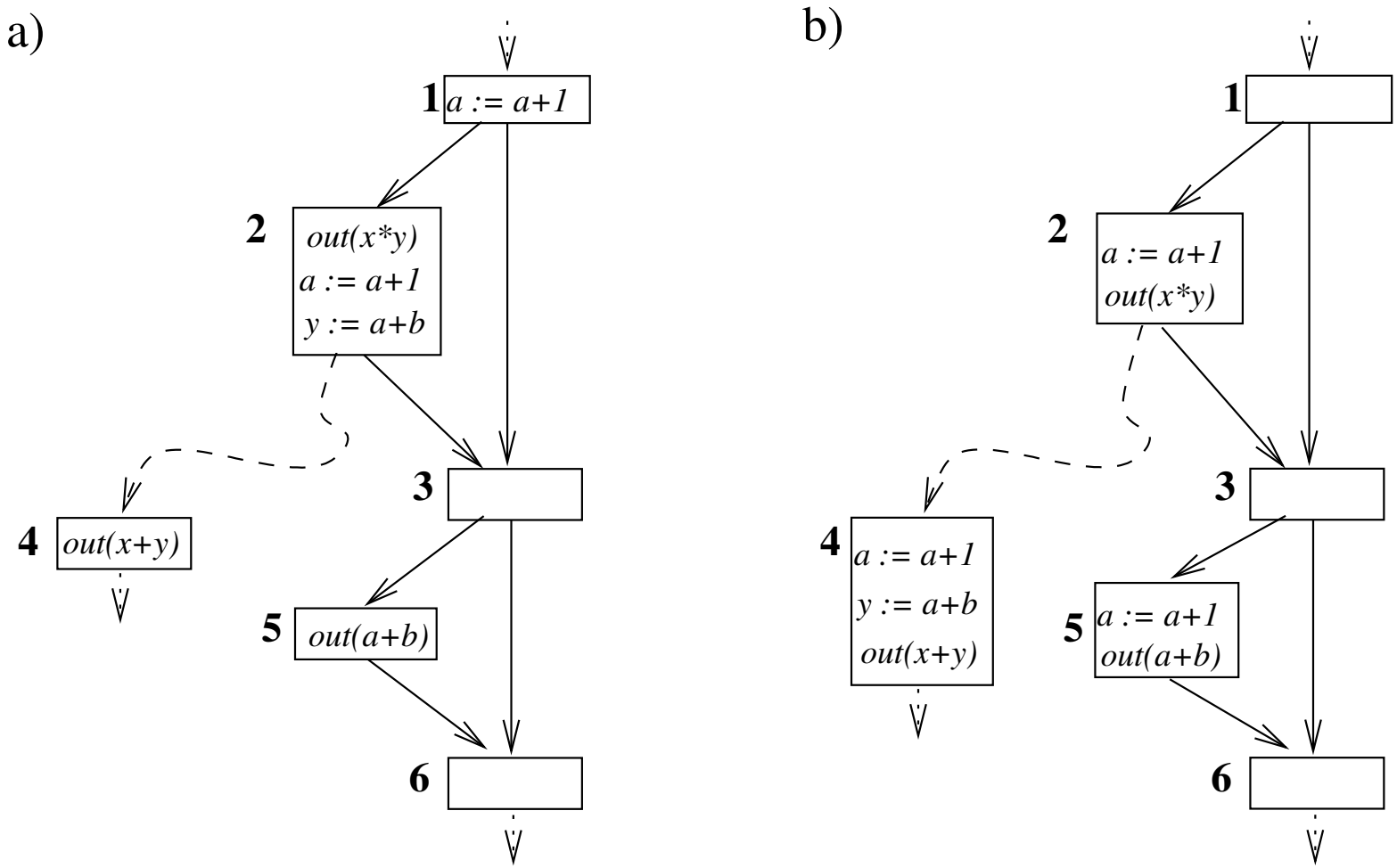
Kombinationswirkung von Effekten zweiter Ordnung (2)

Transformiertes/optimiertes Programm:



Kombinationswirkung von Effekten zweiter Ordnung (3)

Im allgemeinen m2n-Senkungs-Eliminationswirkungen:



Die PDCE/PDFE-Transformationen

Definition 11.2.7 (PDCE/PDFE-Transformationen)

Die Elimination partiell toten/schattenhaften Codes (partial dead (faint) code elimination) PDCE/PFCE ist eine beliebige Abfolge zulässiger

- Anweisungssenkungen und
- Anweisungseliminationen von
  - toten
  - schattenhaften

  Anweisungen.

Bezeichnungen und Schreibweisen (1)

- $G \vdash_{PDCE} G'$ bzw. $G \vdash_{PFCE} G'$:

  G' resultiert aus G durch Anwendung einer zulässigen Anweisungssenkungs- oder Eliminationstransformation (tot bzw. schattenhaft).

- $\tau \in \{PDCE, PFCE\}$:

  Bezeichner für PDCE bzw. PFCE.

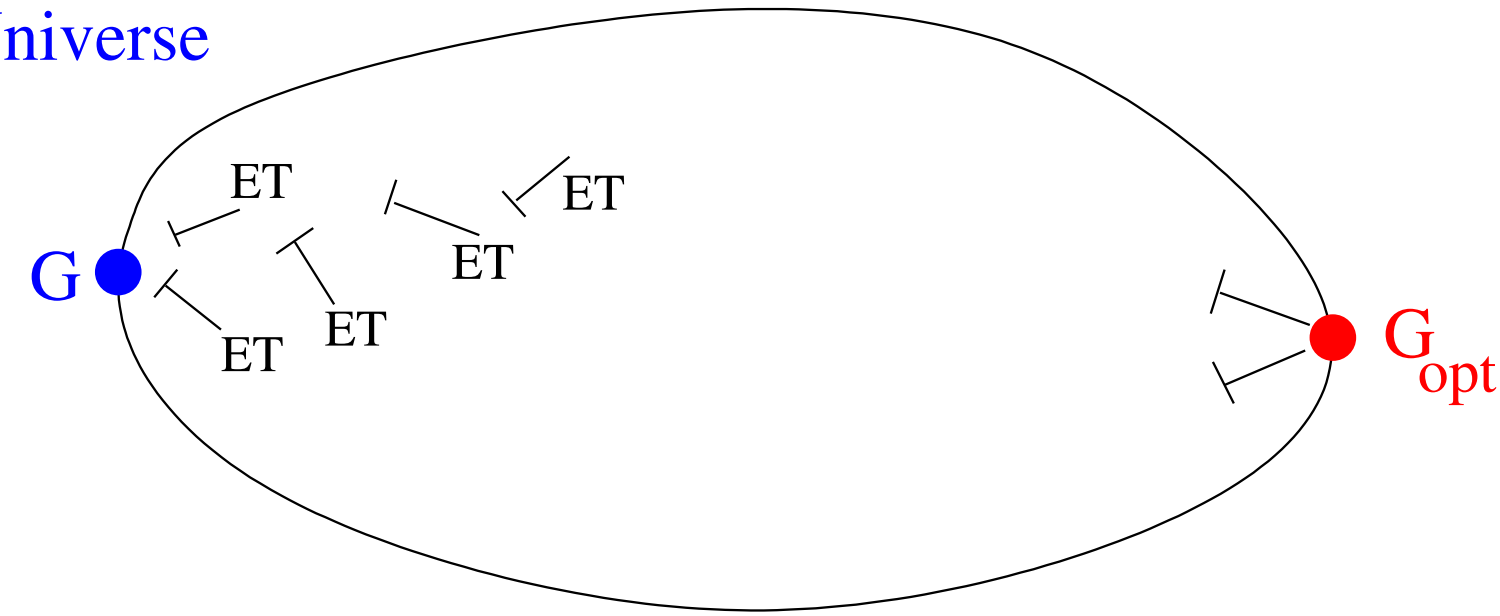
- $\mathcal{G}_\tau =_{df} \{ G' \mid G \vdash_\tau^* G' \}$:

  Das aus G durch sukzessive Anwendung von PDCE- bzw. PFCE-Elementartransformationen aufgespannte Universum.

Bezeichnungen und Schreibweisen (2)

Veranschaulichung des Universums:

- ET steht für Elementar-Transformation,
  d.h. Anweisungssenkung und Anweisungselimination.



Vergleichsrelation "besser" für Programme

Definition 11.2.8 (besser)

Seien $G', G'' \in \mathcal{G}_\tau$. Dann heißt G' besser als G'', in Zeichen $G'' \sqsubseteq_{\sim} G'$, gdw

$$\forall\, p \in \mathbf{P}[\mathbf{s}, \mathbf{e}]\ \forall \alpha \in \mathcal{AP}.\ \alpha\#(p_{G'}) \leq \alpha\#(p_{G''})$$

wobei $\alpha\#(p_{G'})$ und $\alpha\#(p_{G''})$ jeweils die Anzahl der α-Vorkommen auf p in G' bzw. G'' bezeichnen.

Beachte:

- Anweisungssenkungen und -eliminationen erhalten die Verzweigungsstruktur eines Programms G. Die einem Pfad in G entsprechenden Pfade in G' und G'' können deshalb einfach identifiziert werden.

Eigenschaften der Relation “besser”

Lemma 11.2.9

Die Vergleichsrelation $\sqsubseteq_{\sim}$ ist eine

- Quasiordnung (d.h. reflexiv und transitiv, aber nicht antisymmetrisch).

Der Optimalitätsbegriff

Definition 11.2.10 (PDCE/PFCE-Optimalität)

Ein Programm $G^* \in \mathcal{G}_\tau$ ist optimal gdw G^* besser ist als jedes andere Programm aus $\mathcal{G}_\tau$.

Monotonie und Dominanz

Sei

- $\vec{\sqsubseteq}_\tau =_{df} (\sqsubseteq \cap \vdash_\tau)^*$
- $\mathcal{F}_\tau \subseteq \{f \mid f : \mathcal{G}_\tau \to \mathcal{G}_\tau\}$ eine endliche Familie von Funktionen mit
  1. Monotonie:
     $\forall\, G', G'' \in \mathcal{G}_\tau\ \forall\, f \in \mathcal{F}_\tau.$
     $G' \vec{\sqsubseteq}_\tau G'' \Rightarrow f(G') \vec{\sqsubseteq}_\tau f(G'')$
  2. Dominanz:
     $\forall\, G', G'' \in \mathcal{G}_\tau.\ G' \vdash_\tau G'' \Rightarrow \exists\, f \in \mathcal{F}_\tau.\ G'' \vec{\sqsubseteq}_\tau f(G')$

PDCE/PFCE-Hauptergebnisse

Theorem 11.2.11 (Existenz optimalen Programms)

$\mathcal{G}_\tau$ besitzt ein optimales Element (bezüglich $\sqsubseteq_\sim$), das von jeder Folge von Funktionsanwendungen berechnet wird, die alle Elemente aus $\mathcal{F}_\tau$ 'hinreichend' oft enthält.

Beweis mithilfe von Monotonie, Dominanz und Fixpunkt-theorem 9.2.4.

Anwendung auf PDCE und PFCE

- PDCE und PFCE erfüllen die Voraussetzungen von Optimalitätstheorem 11.2.11.
- Das optimale Programm in $\mathcal{G}_\tau$ bezüglich PDCE und PFCE ist bis auf irrelevante Umreihungen von Anweisungen in Basisblöcken eindeutig bestimmt.

Insgesamt ergibt sich daraus die

- Korrektheit und Optimalität

von PDCE- und PDFE-Transformation.

Zweiter Korrektheits- und Optimalitätsbeweis

PDCE/PFCE-Transformationsidee:

Konzeptuell können wir die Elimination partiell toter/schattenhafter Anweisungen (PDCE/PFCE) in folgender Weise verstehen:

- PDCE $= (\text{AS} + \text{DCE})^*$
- PFCE $= (\text{AS} + \text{FCE})^*$

Bezeichnungen (1)

In der Folge bezeichnen wir die aus vorstehender Transformationsidee abgeleiteten Algorithmen für die Elimination partiell

- toter und
- schattenhafter

Anweisungen mit

- pdce und
- pfce.

Bezeichnungen (2)

Wir bezeichnen weiters die aus einem Programm G durch Anwendung von pdce und pfce resultierenden Programme mit

- G_{pdce} und
- G_{pfce}

und die von den Elementartransformationen von pdce und pfce aufgespannten Universen für G mit

- $\mathcal{G}_{PDCE}$ und
- $\mathcal{G}_{PFCE}$.

Veranschaulichung

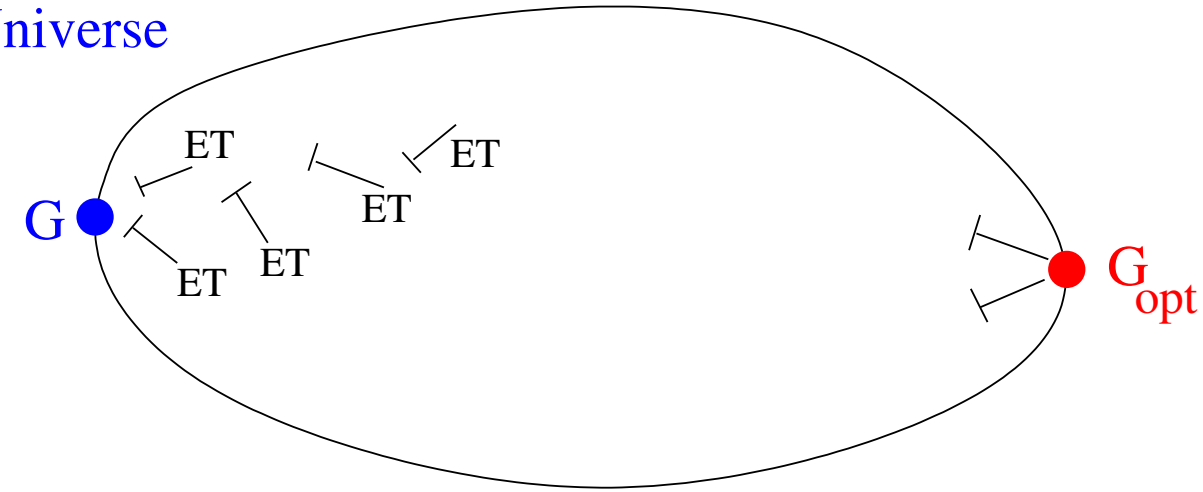
PDCE/PFCE-Ableitungsrelation $\vdash$:

- PDCE: $G \vdash_{AS,DCE} G'$ (d.h. ET$=_{df}$ {AS,DCE})
- PFCE: $G \vdash_{AS,FCE} G'$ (d.h. ET$=_{df}$ {AS,FCE})

Korrektheit und Optimalität von pdce/pfce folgen aus:

Theorem 11.2.12 (Konfluenz und Terminierung)

Die PDCE/PFCE-Ableitungsrelationen $\vdash_{AS,DCE}$ und $\vdash_{AS,FCE}$ sind konfluent und terminierend.



PDCE/PDFE-Korrektheit und -Optimalität

Theorem 11.2.13 (Korrektheit)

1. $G_{pdce} \in \mathcal{G}_{PDCE}$
2. $G_{pfce} \in \mathcal{G}_{PFCE}$

Theorem 11.2.14 (Optimalität)

1. G_{pdce} ist optimal in $\mathcal{G}_{PDCE}$.
2. G_{pfce} ist optimal in $\mathcal{G}_{PFCE}$

Kapitel 11.3

Implementierung von PDCE/PFCE

Vorbereitungen zur DFA-Spezifikation

Wir benötigen 6 Hilfsprädikate für die Spezifikation der benötigten lokalen abstrakten Semantiken und der darauf aufbauenden DFAs.

Im einzelnen folgende 6 lokale Prädikate:

- USED, Rel-Used, Ass-Used und MOD
- LOC-DELAYED und LOC-BLOCKED

Darauf aufbauende DFAs:

- Elimination toten Codes

  $\rightsquigarrow$ Dead-Code Elimination (DCE)
- Elimination schattenhaften Codes

  $\rightsquigarrow$ Faint-Code Elimination (FCE)
- Anweisungssenkung

  $\rightsquigarrow$ Assignment Sinking (AS $\rightsquigarrow$ Delayability)

Bedeutung der lokalen DCE/FCE-Prädikate

Zur Bedeutung der vier lokalen DCE/FCE-Prädikate:

- $\text{USED}_\iota(x)$: x wird rechtsseitig in Anweisung ι benutzt.
- $\text{Rel-Used}_\iota(x)$: x wird rechtsseitig in der "relevanten", d.h. "zum Leben zwingenden" Anweisung ι benutzt.
- $\text{Ass-Used}_\iota(x)$: x wird rechtsseitig in der Zuweisung ι benutzt.
- $\text{MOD}_\iota(x)$: x wird linksseitig in der Anweisung ι benutzt.

Die DCE-Analyse

Die Analyse toter Variablen (DCE):

$$\text{N-DEAD}_\iota = \overline{\text{USED}_\iota} * (\text{X-DEAD}_\iota + \text{MOD}_\iota)$$

$$\text{X-DEAD}_\iota = \prod_{\hat{\iota} \in \mathit{succ}(\iota)} \text{N-DEAD}_{\hat{\iota}}$$

Die FCE-Analyse

Die Analyse schattenhafter Variablen (FCE):

(Simultan für alle Variablen x)

$$\begin{aligned}
\text{N-FAINT}_\iota(x) &= \overline{\text{Rel-Used}_\iota(x)} \ * \\
&\quad (\text{X-FAINT}_\iota(x) \ + \ \text{MOD}_\iota(x)) \ * \\
&\quad (\text{X-FAINT}_\iota(\mathit{LhsVar}_\iota) + \overline{\text{Ass-Used}_\iota(x)})
\end{aligned}$$

$$\text{X-FAINT}_\iota(x) = \prod_{\hat{\iota} \in \mathit{succ}(\iota)} \text{N-FAINT}_{\hat{\iota}}(x)$$

wobei LhsVar_ι die linksseitige Variable von Zuweisung ι bezeichnet.

Bedeutung der lokalen AS-Prädikate

Zur Bedeutung der zwei lokalen AS-Prädikate:

- LOC-DELAYED$_n(\alpha)$: Es gibt einen α-Anweisungssenkungs-Kandidaten (sinking candidate) in n.
- LOC-BLOCKED$_n(\alpha)$: Die Senkung von α ist durch eine Anweisung an n blockiert.

Die AS-Analyse

Das Anweisungssenkungs-Gleichungssystem:

$$\text{N-DELAYED}_n = \begin{cases} \textbf{falsch} & \text{if} \quad n = \textbf{s} \\ \prod_{m \in pred(n)} \text{X-DELAYED}_m & \text{otherwise} \end{cases}$$

$$\text{X-DELAYED}_n = \text{LOC-DELAYED}_n + \text{N-DELAYED}_n * \overline{\text{LOC-BLOCKED}_n}$$

Einsetzungspunkte gesenkter Anweisungen

Die sich aus der AS-Analyse ergebenden Einsetzungspunkte:

$$\text{N-INSERT}_n \quad =_{df} \quad \text{N-DELAYED}^*_n * \text{LOC-BLOCKED}_n$$

$$\text{X-INSERT}_n \quad =_{df} \quad \text{X-DELAYED}^*_n * \sum_{m \in succ(n)} \overline{\text{N-DELAYED}^*_m}$$

wobei N-DELAYED^*_n und X-DELAYED^*_n die größten Lösungen des Anweisungssenkungs-Gleichungssystems bezeichnen.

Beachte: Die Berechnung der Einsetzungspunkte erfordert keine DFA!

Kapitel 11.4

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 11 (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 11 (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 12

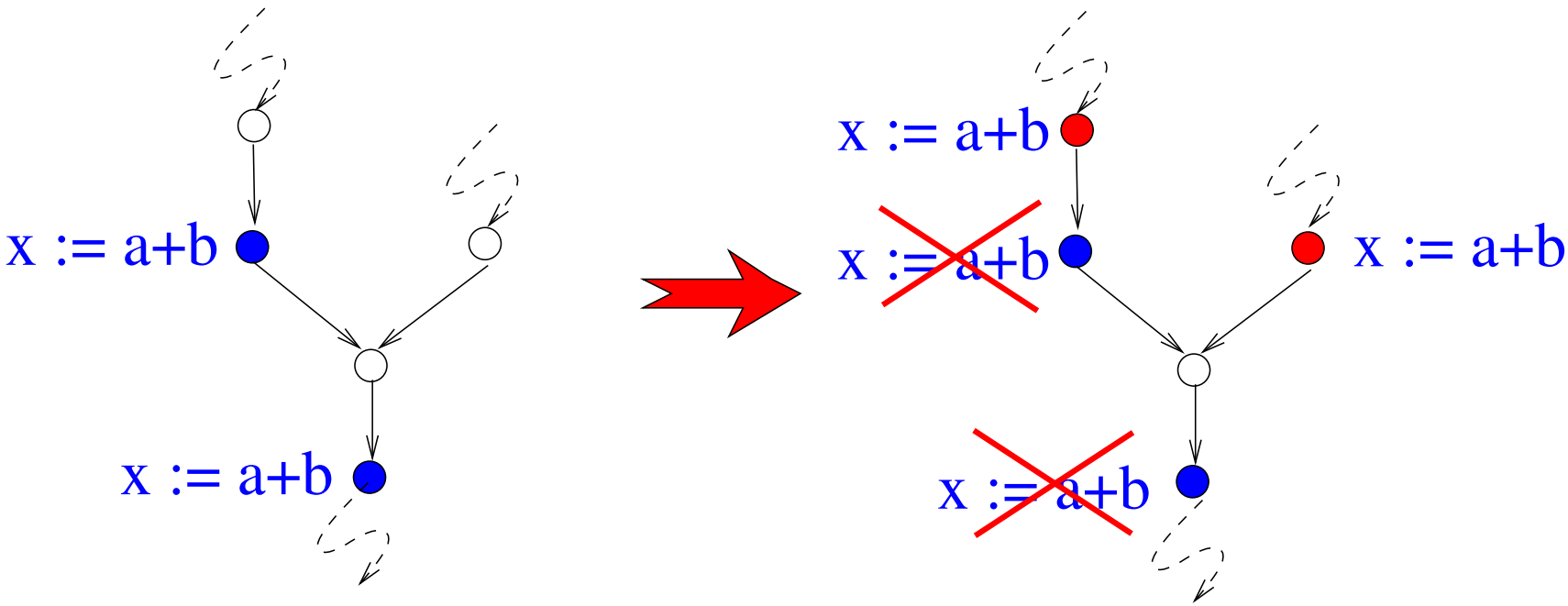
Elimination partiell redundanter Anweisungen

Kapitel 12.1

Motivation

Elimination partiell redundanter Anweisungen

Veranschaulichendes Beispiel:



Transformationsidee

Konzeptuell können wir die Elimination partiell redundanter Anweisungen (PRAE) in folgender Weise verstehen:

- PRAE $= (\mathrm{AH} + \mathrm{RAE})^*$

Analog zu den PDCE/PFCE-Transformationen pdce und pfce gilt auch für die PRAE-Transformation prae ein

- Korrektheits- und
- Optimalitätsresultat.

Veranschaulichung

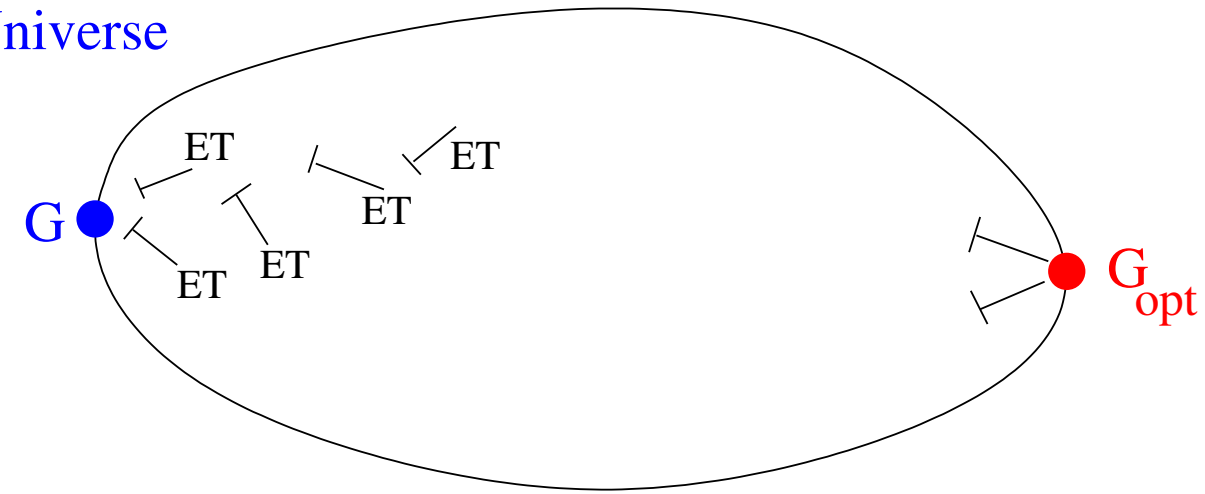
PRAE-Ableitungsrelation $\vdash$:

- PRAE: $G \vdash_{AH,RAE} G'$ (d.h. ET$=_{df}$ {AH,RAE})

Korrektheit und Optimalität von prae folgen aus:

Theorem 12.1.1 (Konfluenz und Terminierung)

Die PRAE-Ableitungsrelation $\vdash_{AH,RAE}$ ist konfluent und terminierend.



PRAE-Korrektheit und -Optimalität

Theorem 12.1.2 (Korrektheit)

$G_{prae} \in \mathcal{G}_{PRAE}$

Theorem 12.1.3 (Optimalität)

G_{prae} ist optimal in $\mathcal{G}_{PRAE}$.

Kapitel 12.2

EAM: Einheitliche PREE/PRAE-Behandlung

Grundtransformationen: PREE und PRAE

Zwei Grundtransformationen zur Redundanzelimination:

- Elimination partiell redundanter Ausdrücke
  - $\rightsquigarrow$ Partially Redundant Expression Elimination (PREE)
  - $\rightsquigarrow$ Expression Motion (EM)
- Elimination partiell redundanter Anweisungen
  - $\rightsquigarrow$ Partially Redundant Assignment Elimination (PRAE)
  - $\rightsquigarrow$ Assignment Motion (AM)

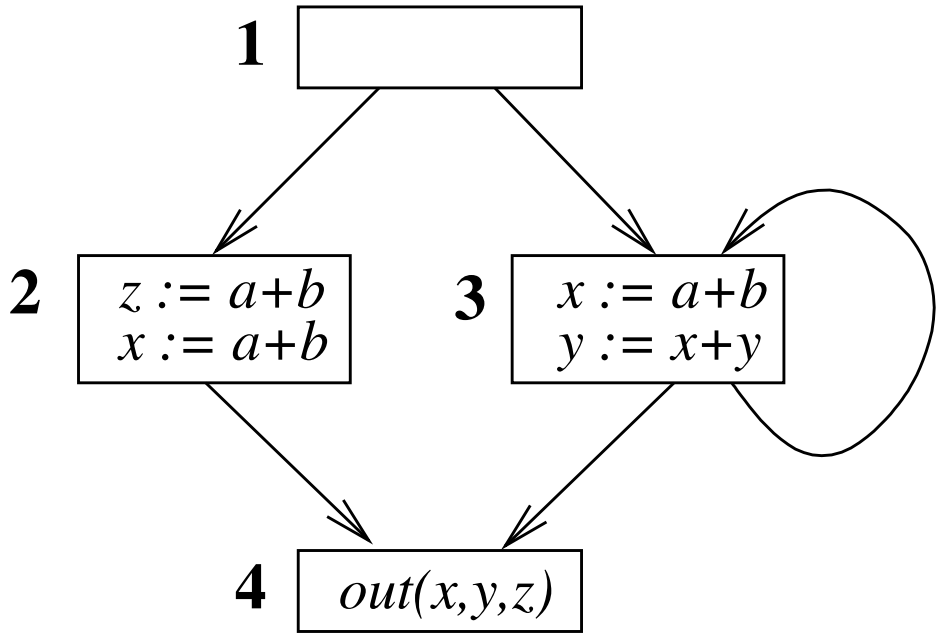
Kombinierte PRE/AE-Transformation: EAM

Kombinierte Transformation zur Redundanzelimination:

- Elimination partiell redundanter Ausdrücke und Anweisungen
  - $\rightsquigarrow$ Partially Redundant Expression and Assignment Elimination (PREAE)
  - $\rightsquigarrow$ Expression and Assignment Motion (EAM)

In der Folge

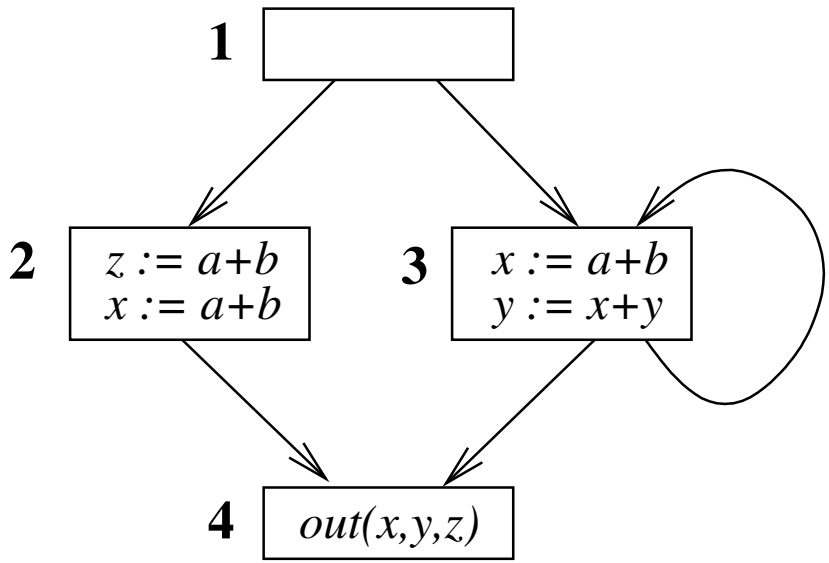
...illustrieren wir die unterschiedlichen Effekte dieser Transformationen anhand eines gemeinsamen Beispiels:



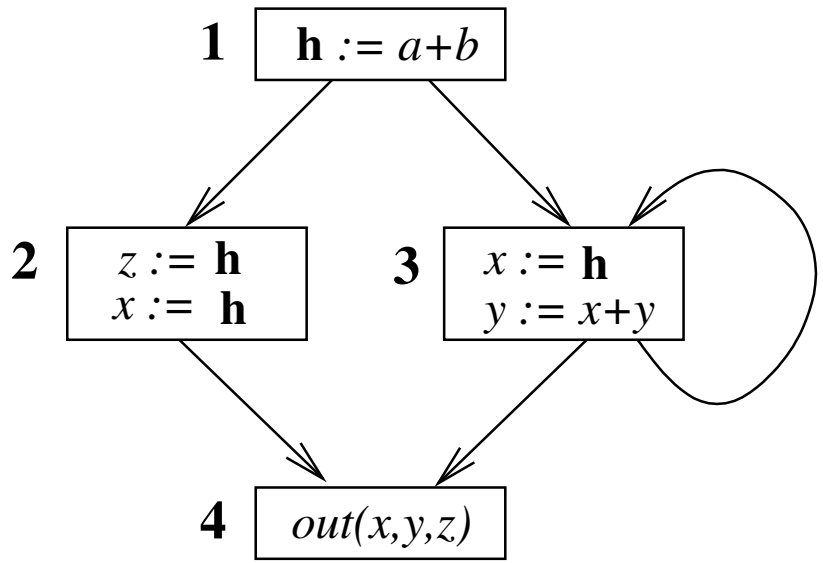
Elimination partiell redundanter Ausdrücke

Der PREE-Effekt auf das laufende Beispiel:

a)



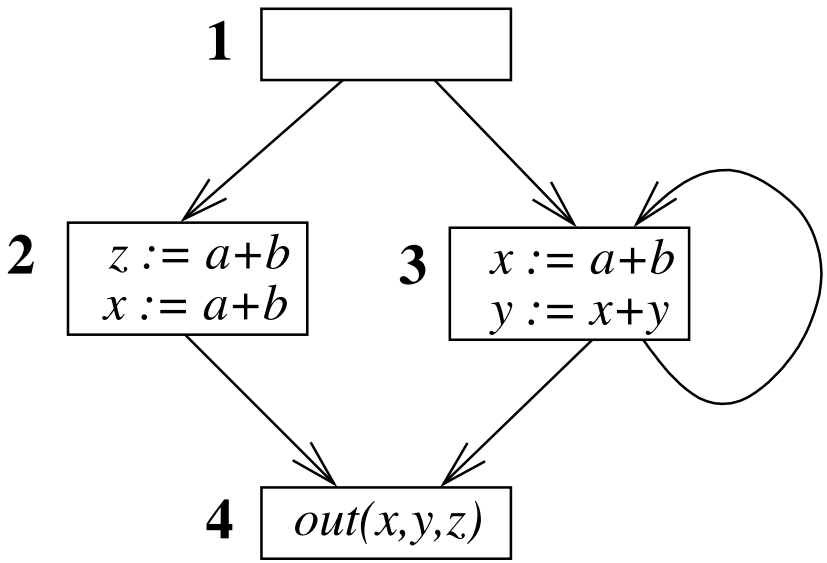
b)



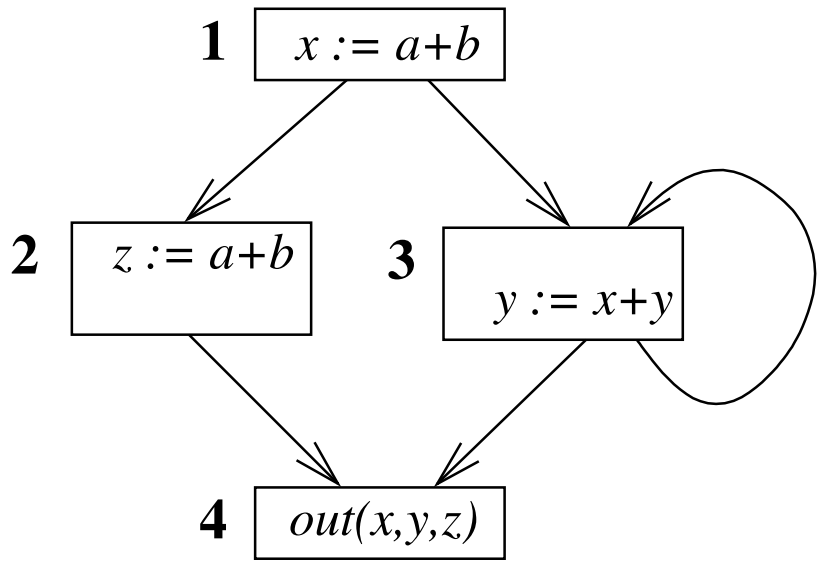
Elimination partiell redundanter Anweisungen

Der PRAE-Effekt auf das laufende Beispiel:

a)

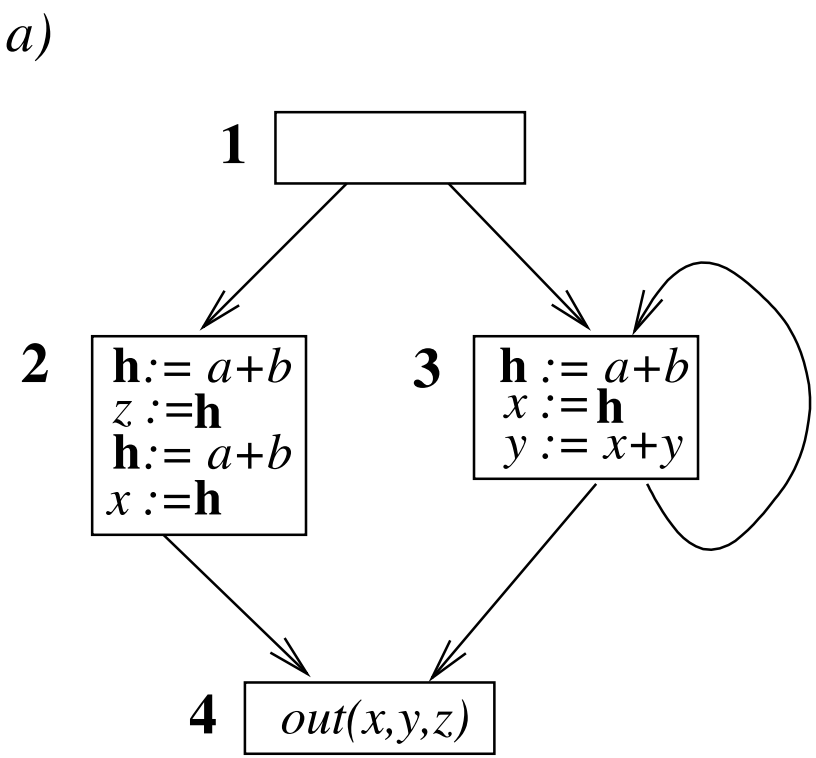


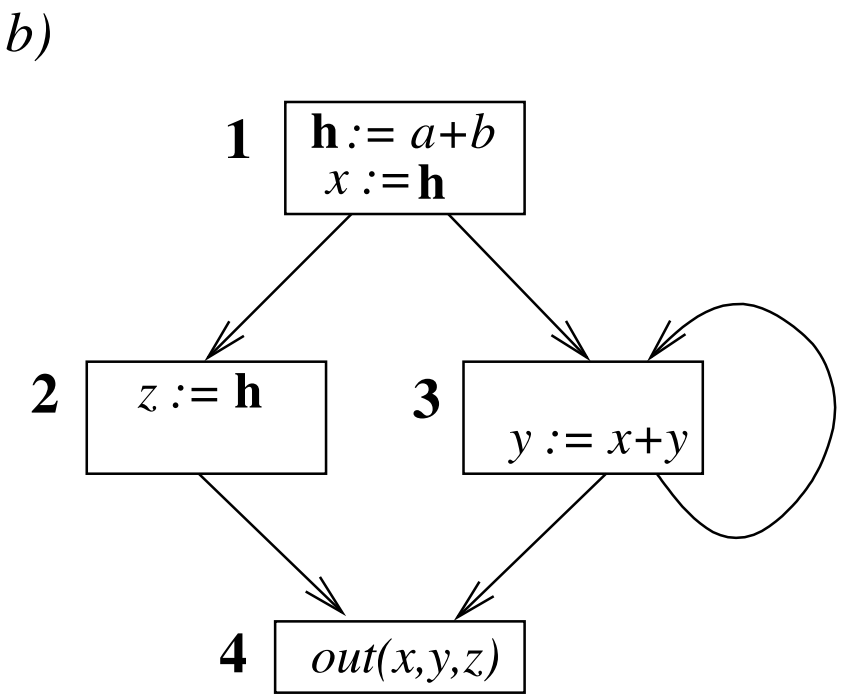
b)



Kombinierte Elimination partiell redundanter Ausdrücke und Anweisungen

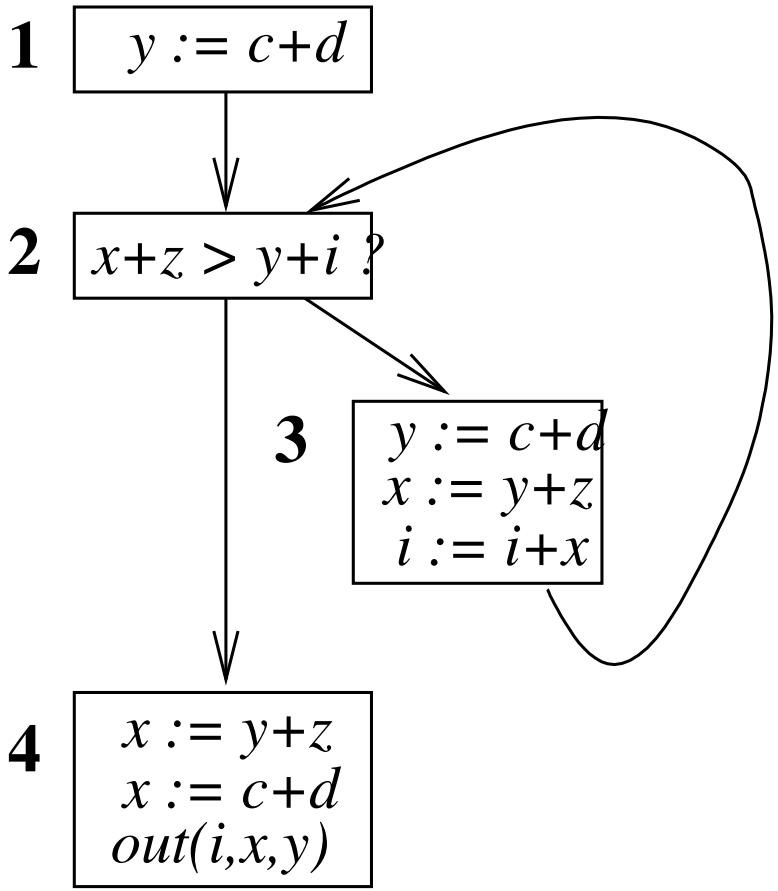
Der EAM(=PREAE)-Effekt auf das laufende Beispiel:





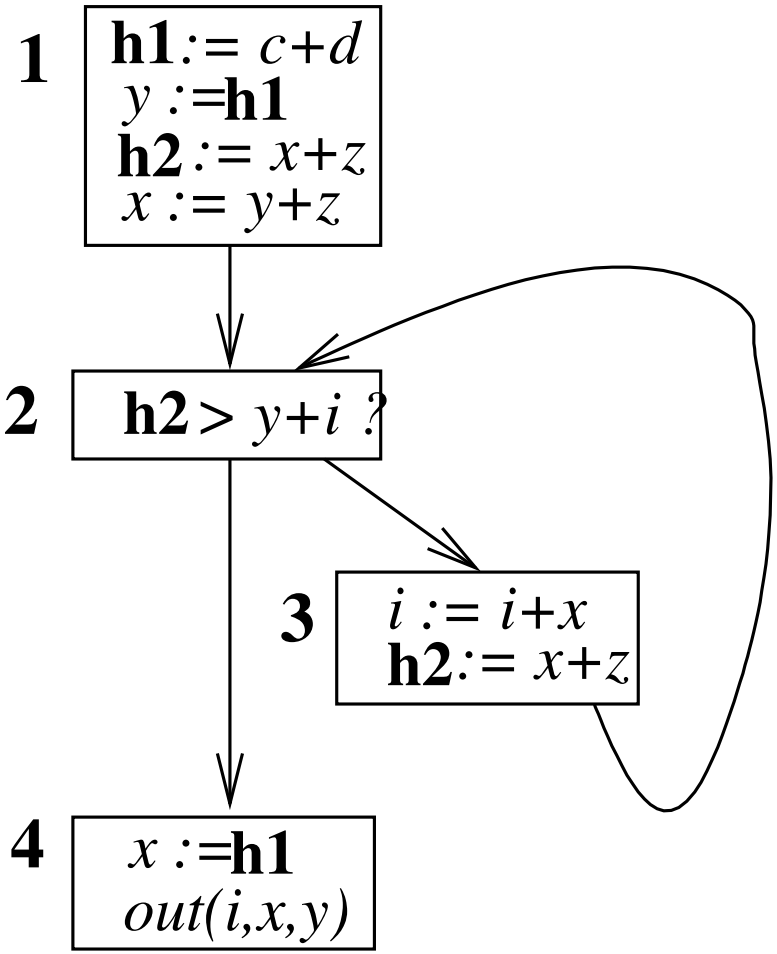
EAM anhand eines größeren Beispiels (1)

Ausgangsprogramm:



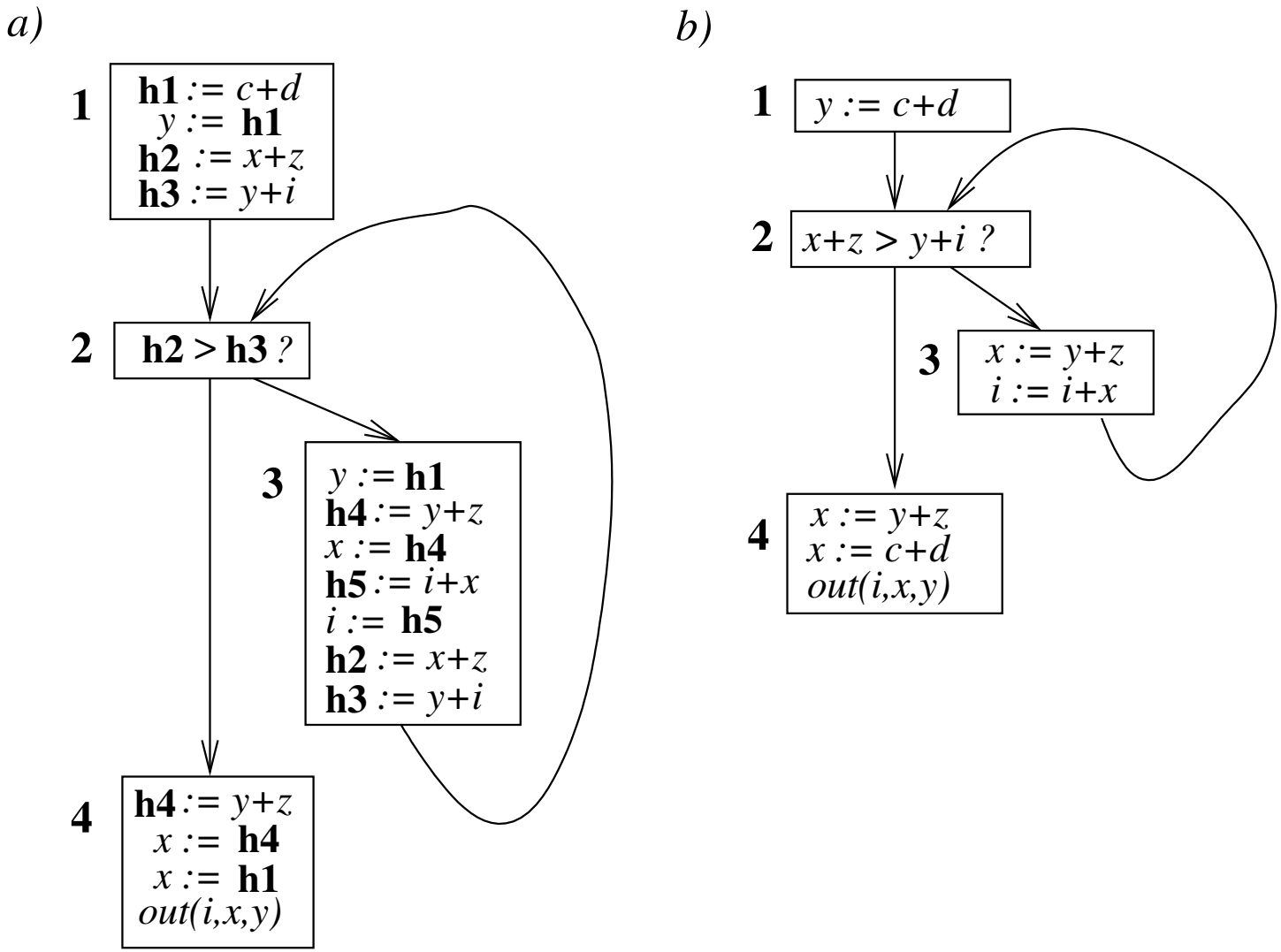
EAM anhand eines größeren Beispiels (2)

Effekt der EAM-Optimierung:



PREE- und PRAE-Effekte zum Vergleich

Die (schwächeren) Effekte von PREE (a) und PRAE (b):



Kapitel 12.3

EAM: Transformation und Optimalität

Der EAM-Algorithmus eam

eam: Ein dreistufiges Verfahren

- Präprozess
  Ersetze jedes Vorkommen einer Anweisung $x := t$ durch die Anweisungssequenz $h_t := t;\ x := h_t$.
- Hauptprozess
  Wende die Transformationen
  - Heben von Anweisungen
    $\rightsquigarrow$ Assignment Hoisting (AH)
  - Eliminieren (total) redundanter Anweisungen
    $\rightsquigarrow$ (Totally) Redundant Assignment Elimination (RAE)

  wiederholt so lange an bis Stabilität eintritt.
- Postprozess
  Aufräumen isolierter Initialisierungen.

Wichtig

Der Präprozess bewirkt

- dass der 3-stufige EAM-Algorithmus die Effekte von PREE und PRAE einheitlich erfasst und abdeckt!

Dabei gilt:

- "Das Ganze ist mehr als die Summe seiner Teile":

$$\text{EAM} > \text{PREE} + \text{PRAE}$$

Effekte zweiter Ordnung für (E)AM

Effekte zweiter Ordnung (engl. second order effects) im (E)AM-Fall:

- Hebungs-Hebungs-Effekte

  $\rightsquigarrow$ Hoisting-Hoisting effects (HH)

- Hebungs-Eliminations-Effekte (Zieleffekt)

  $\rightsquigarrow$ Hoisting-Elimination effects (HE)

- Eliminations-Hebungs-Effekte

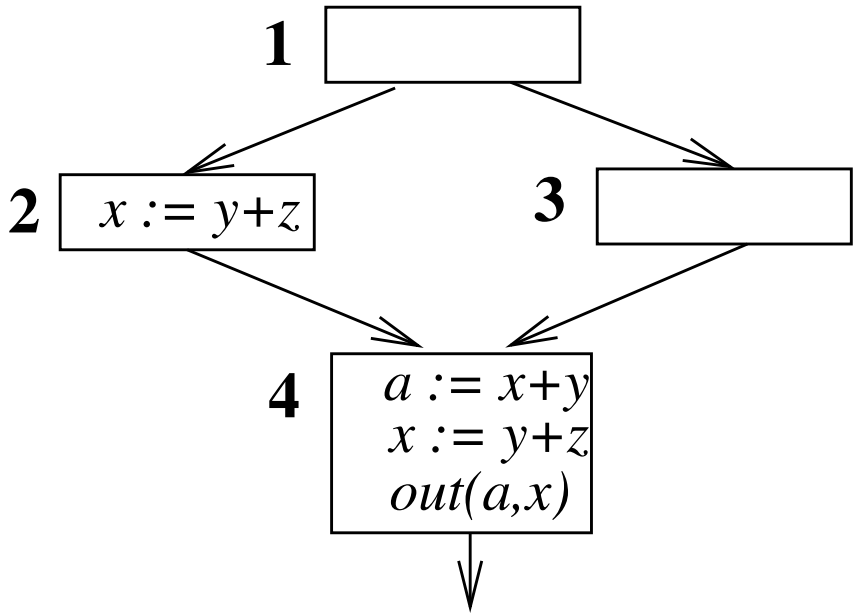
  $\rightsquigarrow$ Elimination-Hoisting effects (EH)

- Eliminations-Eliminations-Effekte (Zieleffekt)

  $\rightsquigarrow$ Elimination-Elimination effects (EE)

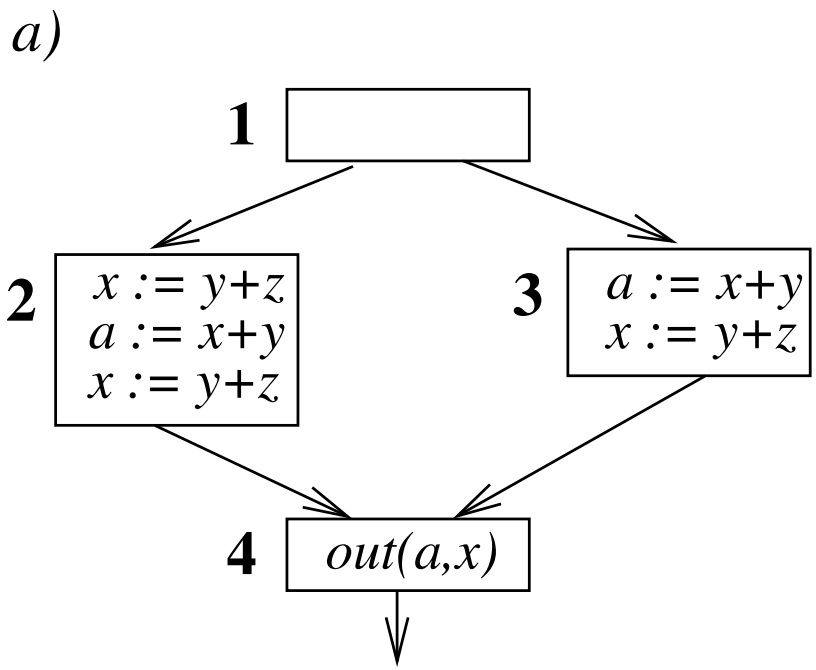
Veranschaulichung von Effekten 2. Ordnung

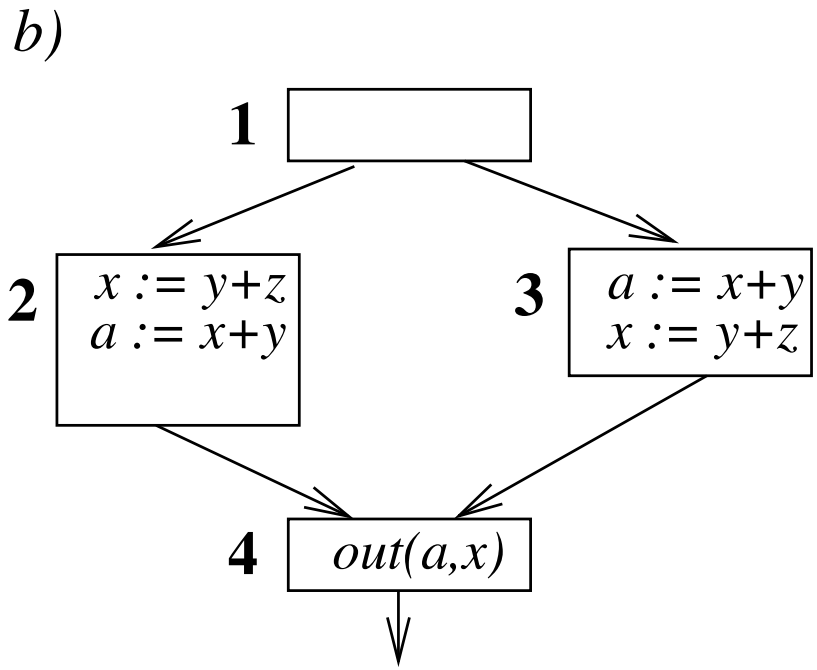
Ausgangsprogramm:



Veranschaulichung von Effekten 2. Ordnung

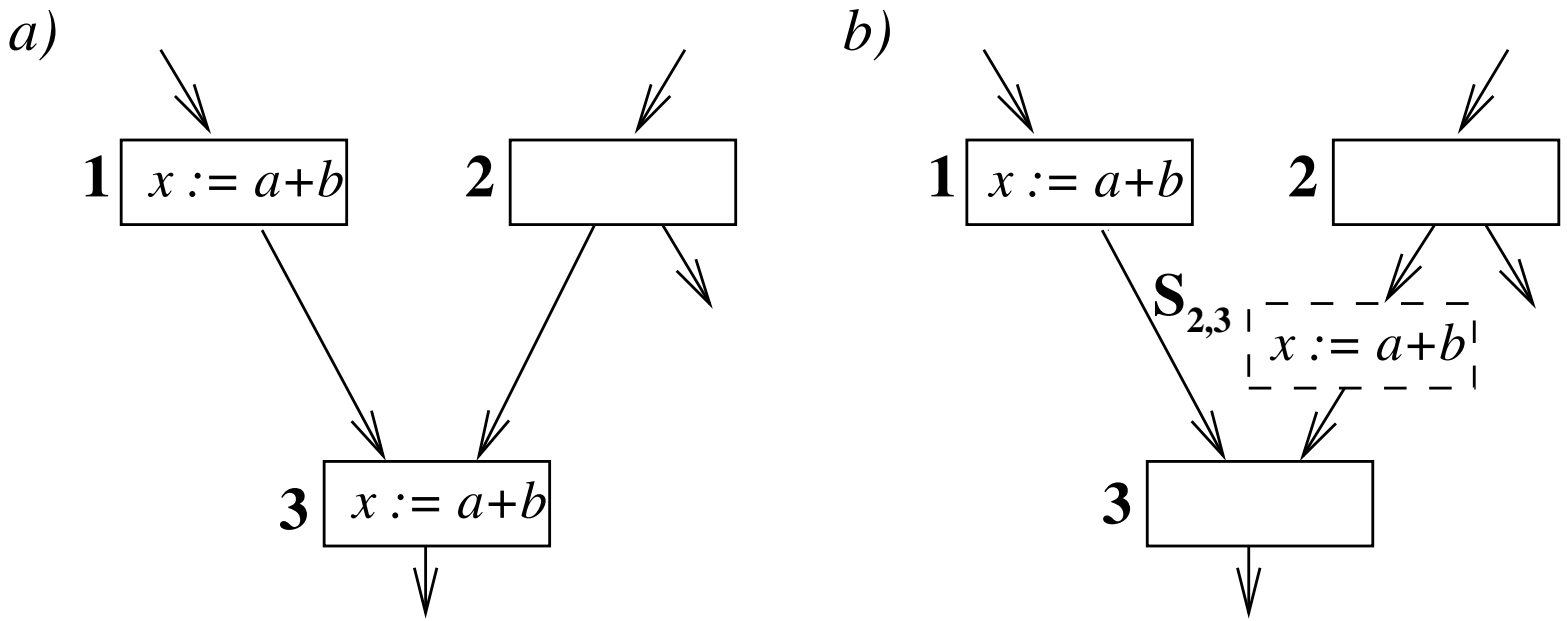
Transformiertes/optimiertes Programm:





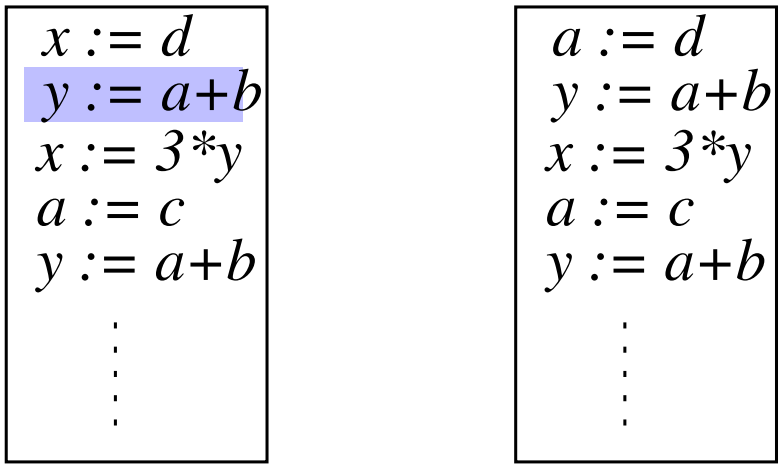
Wie für PDCE/PFCE

Spalten kritischer Kanten:



Analog zu PDCE/PFCE

Hebungskandidat in Basisblöcken:

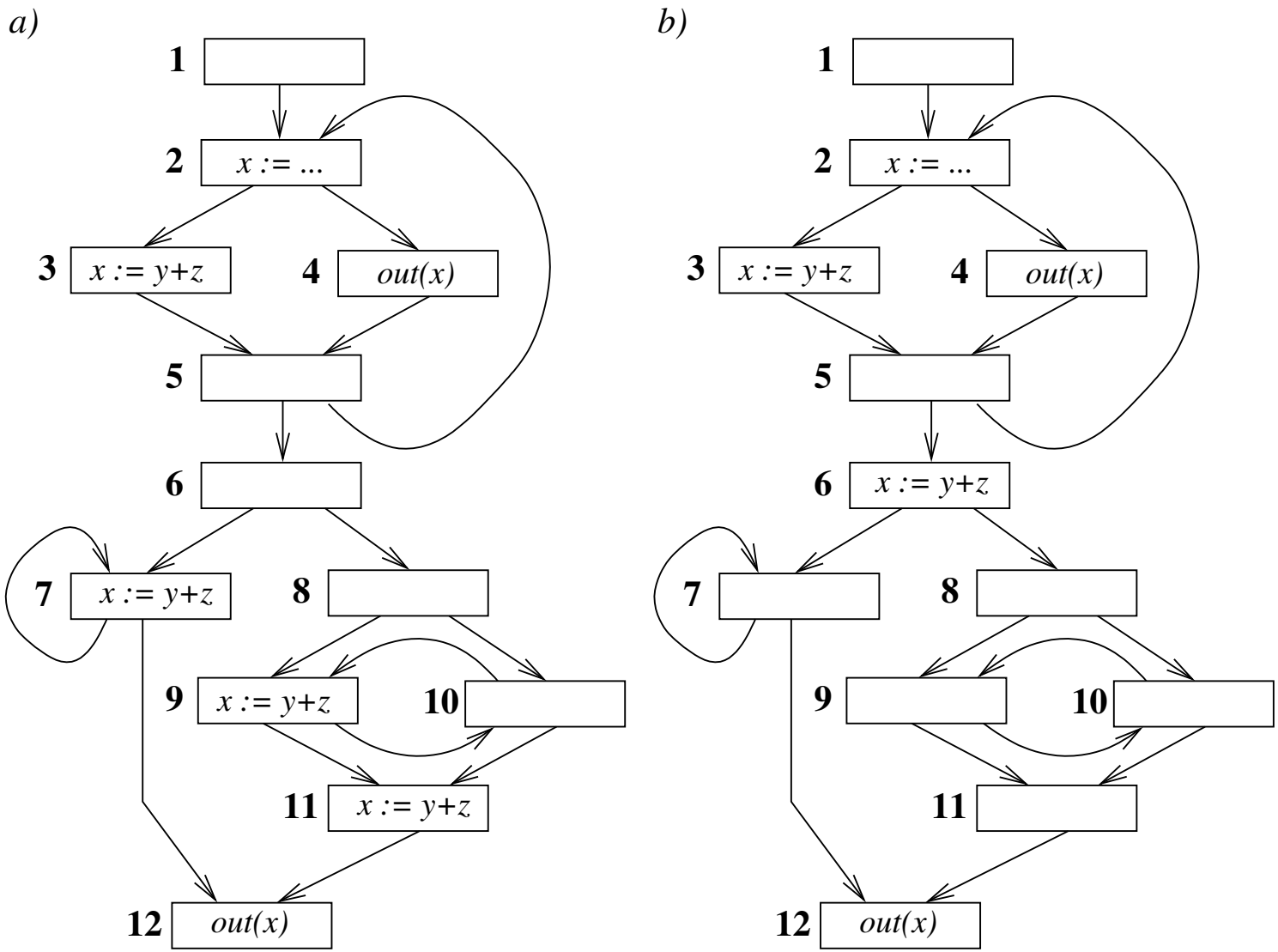


Beachte:

- Nur das markierte Vorkommen von $y := a + b$ ist ein Hebungskandidat; die drei anderen Vorkommen von $y := a + b$ sind lokal blockiert.

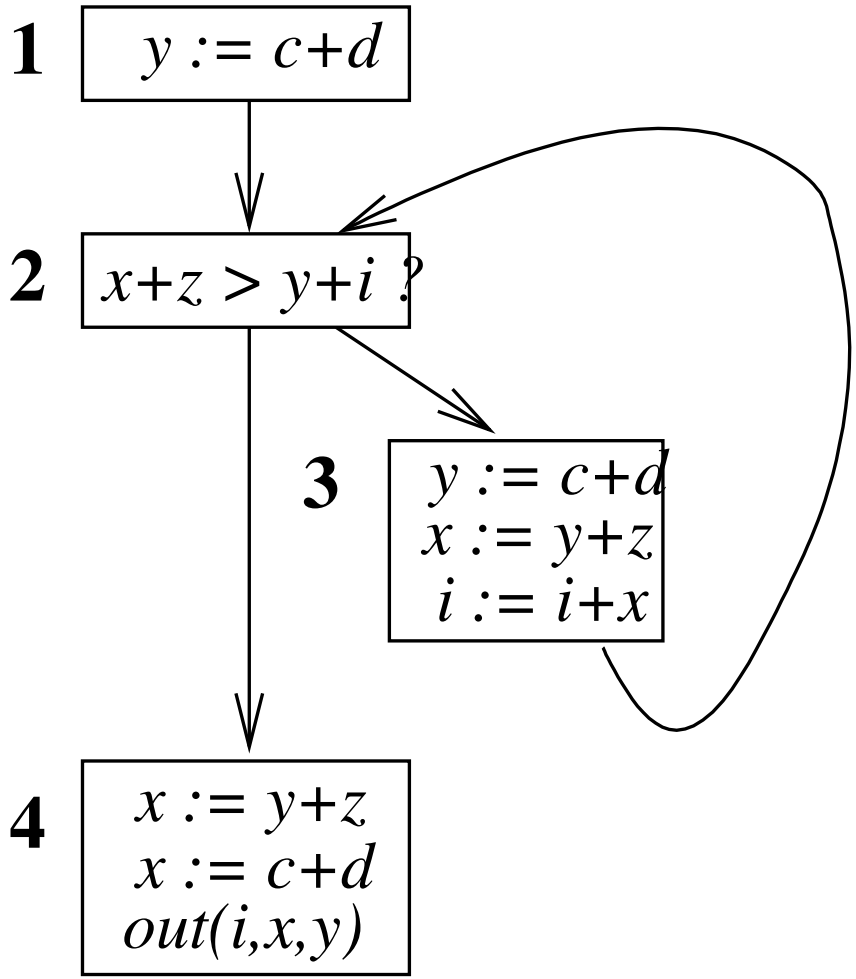
EAM für schleifenbehaftete Programme

Kein Schieben von Anweisungen in Schleifen:



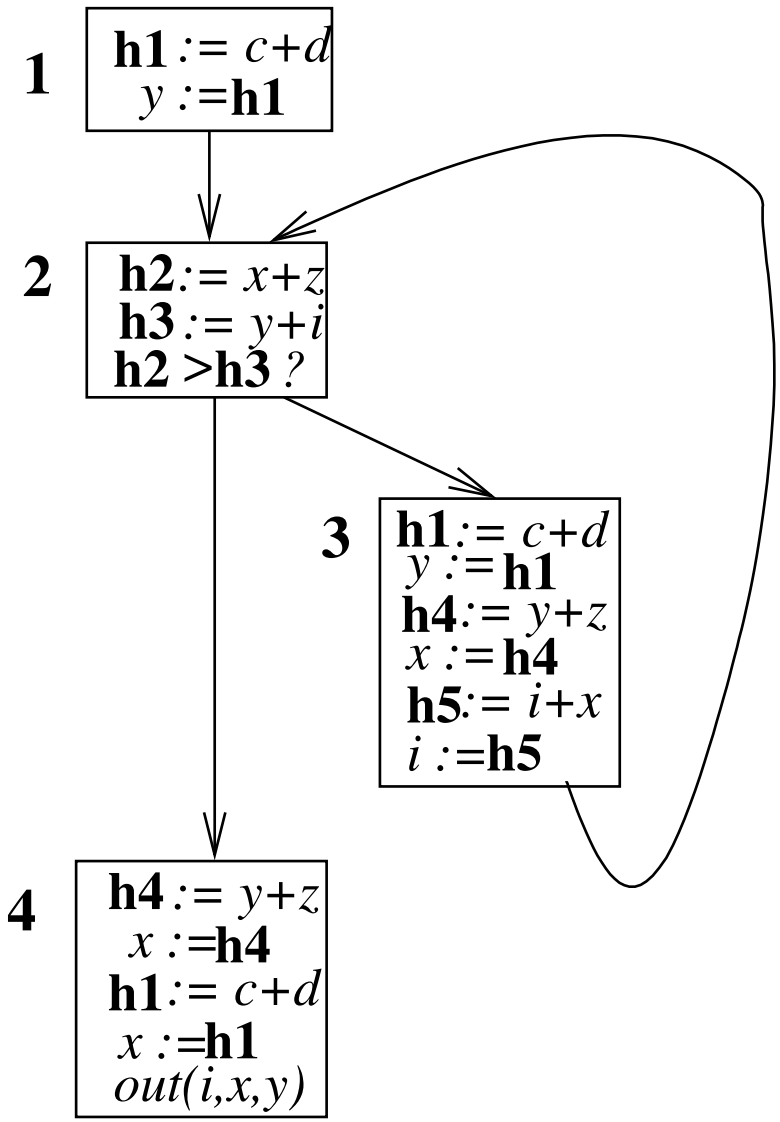
Die EAM-Transformation im Detail (1)

Ausgangsprogramm:



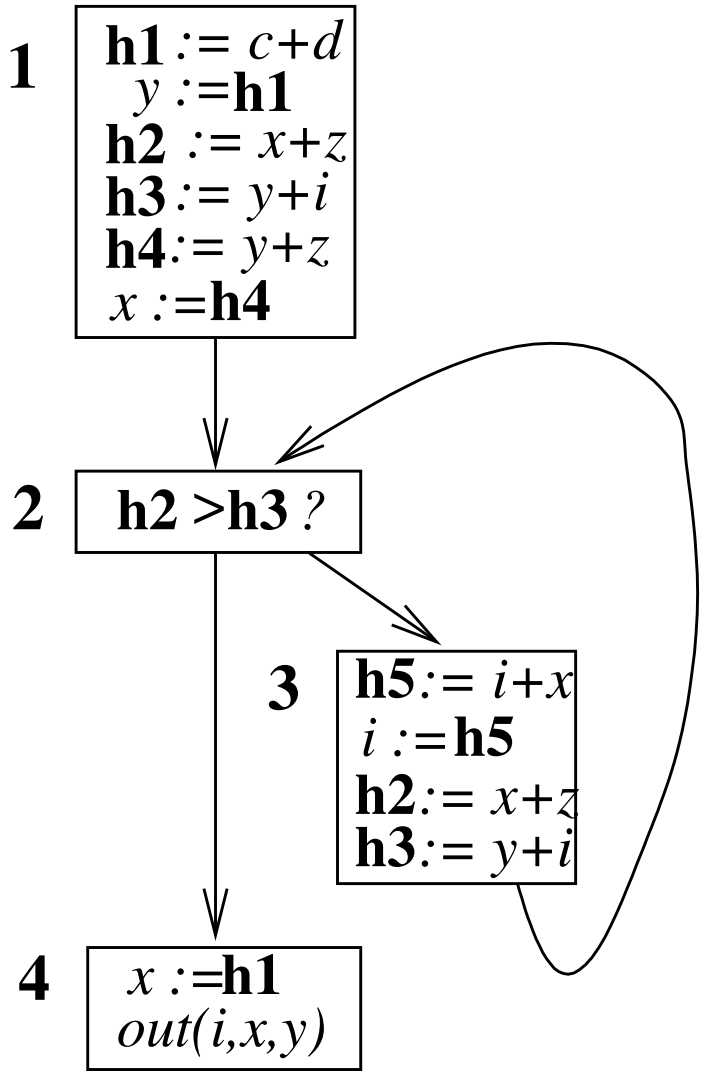
Die EAM-Transformation im Detail (2)

Effekt des Präprozesses:



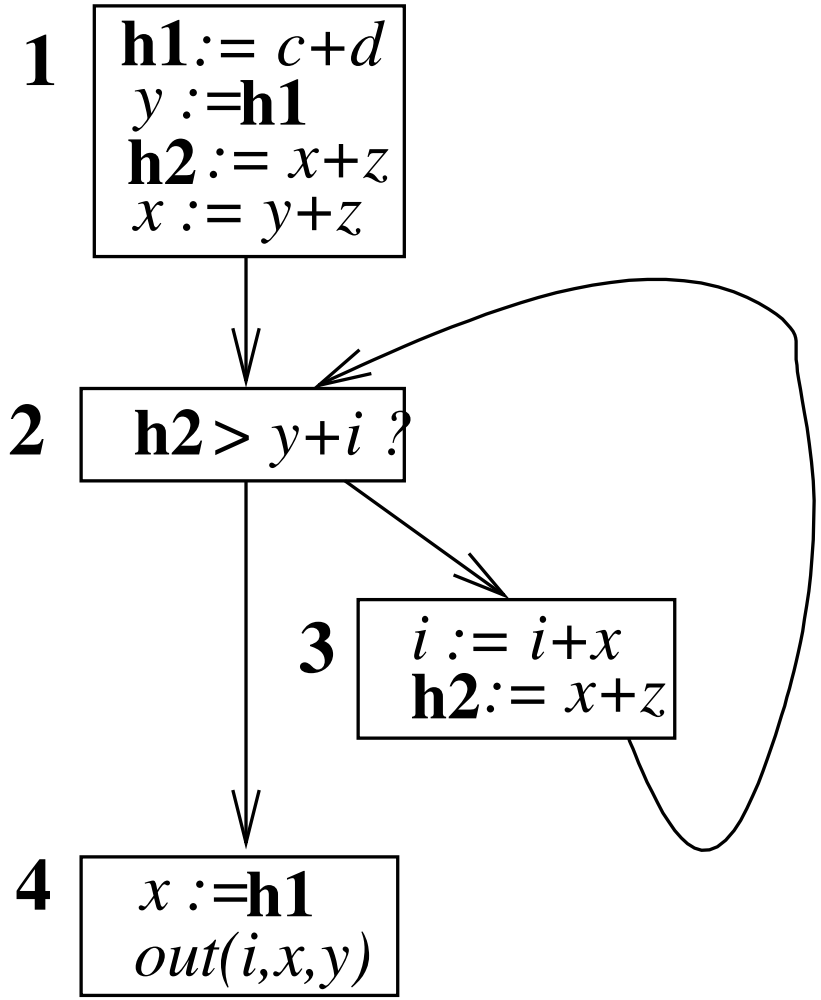
Die EAM-Transformation im Detail (3)

Effekt des Hauptprozesses:



Die EAM-Transformation im Detail (4)

Effekt des Postprozesses und damit des EAM-Gesamteffekts:



EAM-Hauptresultate

Analog zu den PRAE/PDCE/PFCE-Transformationen gelten auch für die EAM-Transformation

- Korrektheits- und
- Optimalitätsresultate.

Anders als für die PRAE/PDCE/PFCE-Transformationen gilt:

- EAM-Optimalität zerfällt in Aussagen über
  - Ausdrücke, Anweisungen und Hilfsvariablen(anzahl)
  - lokale und globale Optimalität.

Bezeichnungen für die folgenden Korrektheits- und Optimalitätstheoreme:

- Sei G ein Programm, G_{eam} das durch Anwendung von eam auf G entstehende Programm und $\mathcal{G}_{EAM}$ das durch EAM-Elementartransformationen aufgespannte Universum.

EAM-Korrektheit

Theorem 12.3.1 (Korrektheit)

$$G_{eam} \in \mathcal{G}_{EAM}$$

EAM-Optimalität: Ausdrucksoptimalität

Theorem 12.3.2 (Ausdrucksoptimalität)

G_{eam} is ausdrucksoptimal in $\mathcal{G}_{EAM}$, d.h., während seiner Ausführung werden höchstens so viele Ausdrücke ausgewertet wie in jedem anderen Programm, das durch Anwendung von PREE- und PRAE-Transformationen entstehen kann.

EAM-Optimalität: Anweisungsoptimalität

Theorem 12.3.3 (Relative Anweisungsoptimalität)

G_{eam} ist relativ anweisungsoptimal in $\mathcal{G}_{EAM}$, d.h. es ist nicht möglich, die Zahl der von G_{eam} zur Laufzeit ausgeführten Anweisungen durch PREE- und PRAE-Transformationen zu verringern.

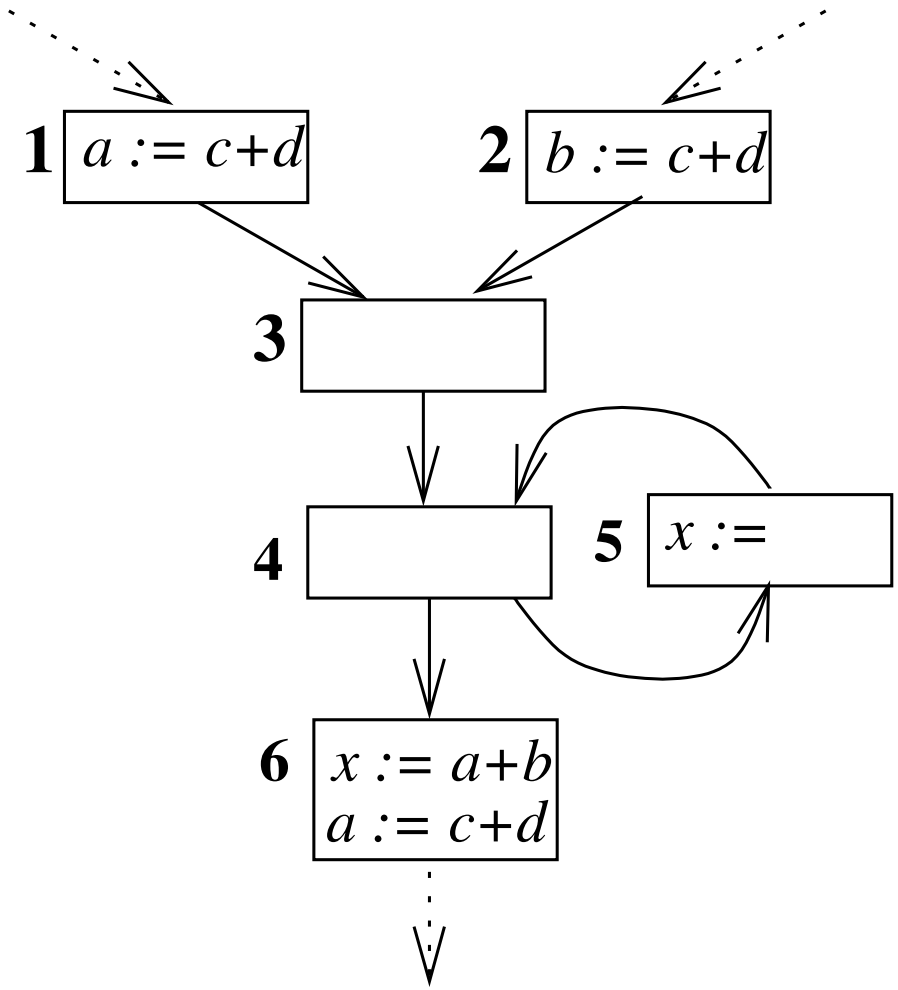
EAM-Optimalität: Hilfsvariablenoptimalität

Theorem 12.3.4 (Relative Hilfsvariablenoptimalität)

G_{eam} ist relative hilfsvariablenoptimal in $\mathcal{G}_{EAM}$, d.h. es ist nicht möglich, die Zahl der Zuweisungen an Hilfsvariablen oder die Länge der Lebenszeiten der Hilfsvariablen in G_{eam} durch Anweisungssenkungen (assignment sinkings) zu verringern.

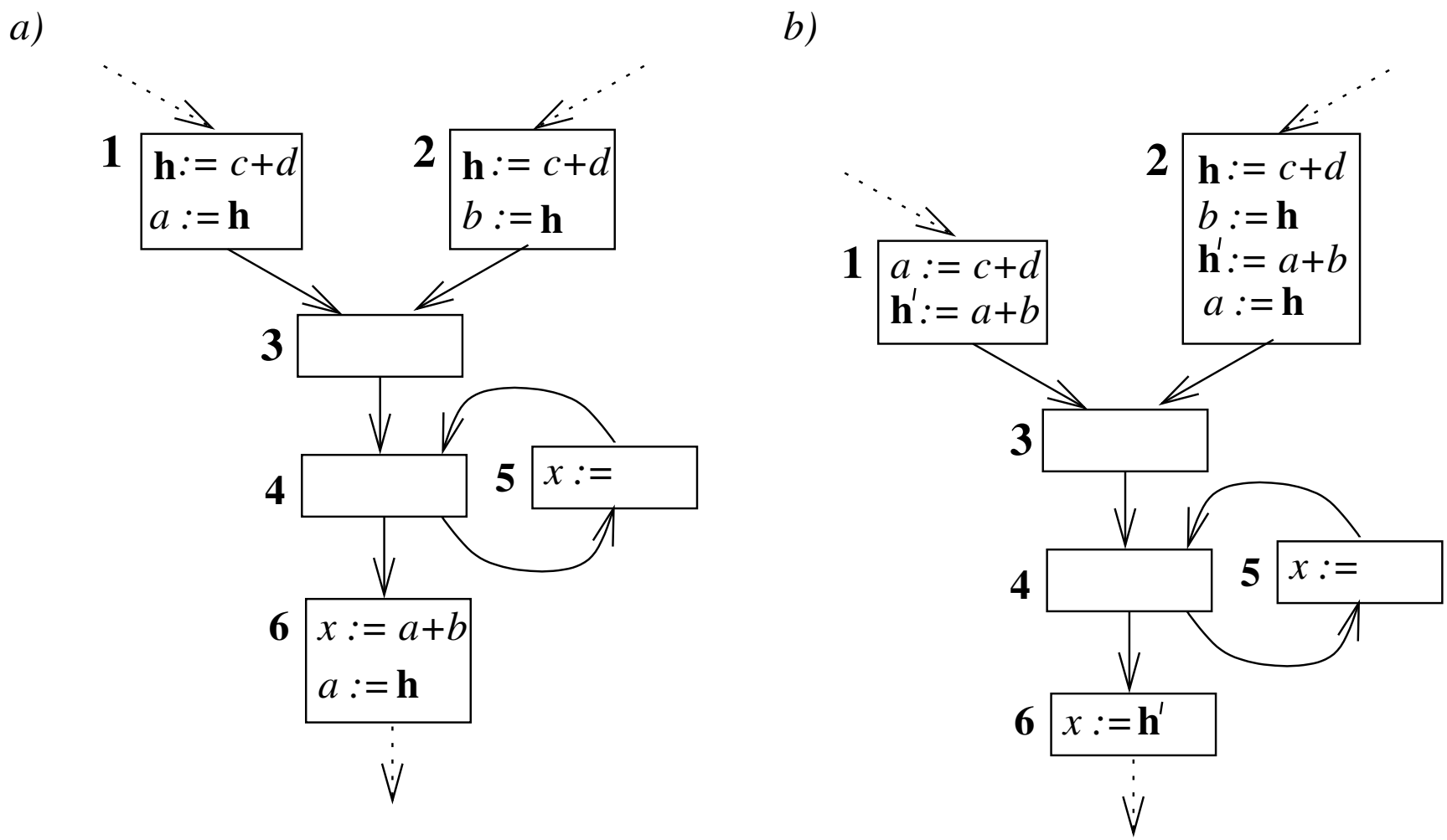
Warum nur relative A/HV-Optimalität?

Betrachte dazu folgendes Programm:



Zur relativen A/HV-Optimalität

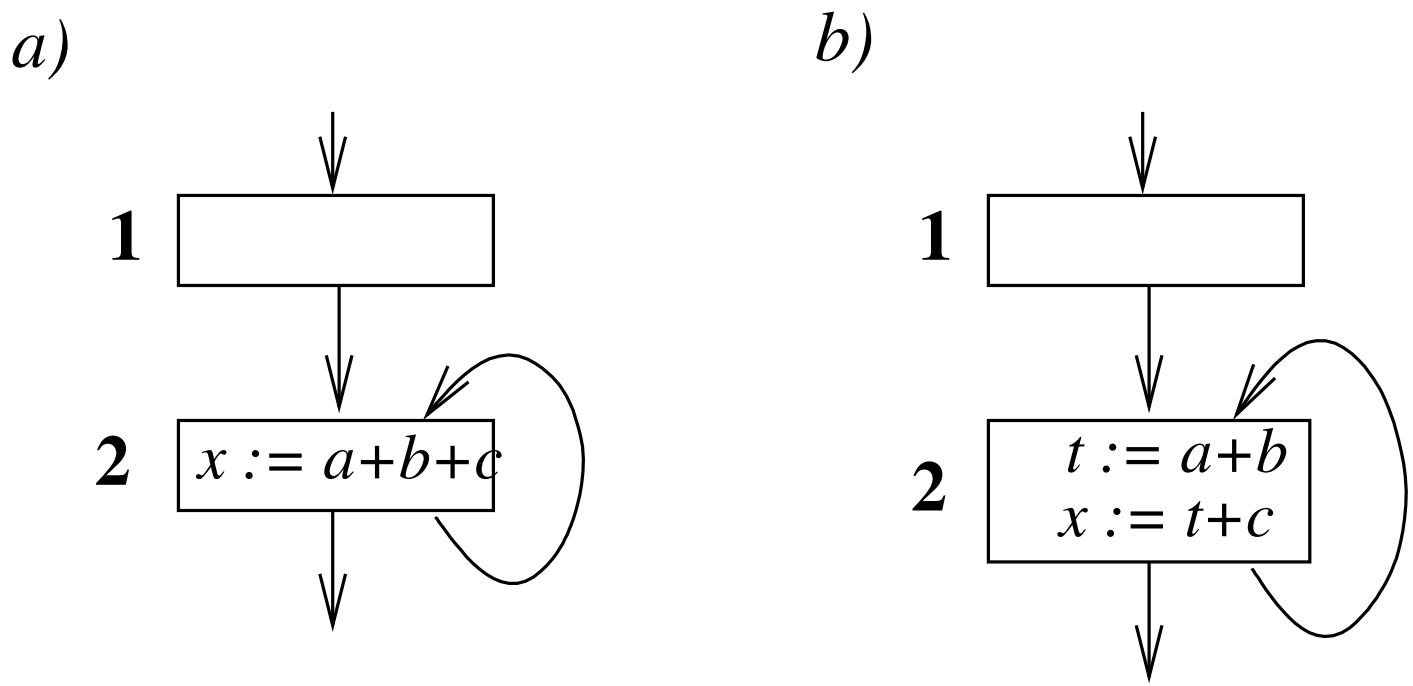
...und folgende zwei unvergleichbare Transformationsresultate:



⇒ Relative A/HV-Optimalität ist das Beste, was möglich ist!

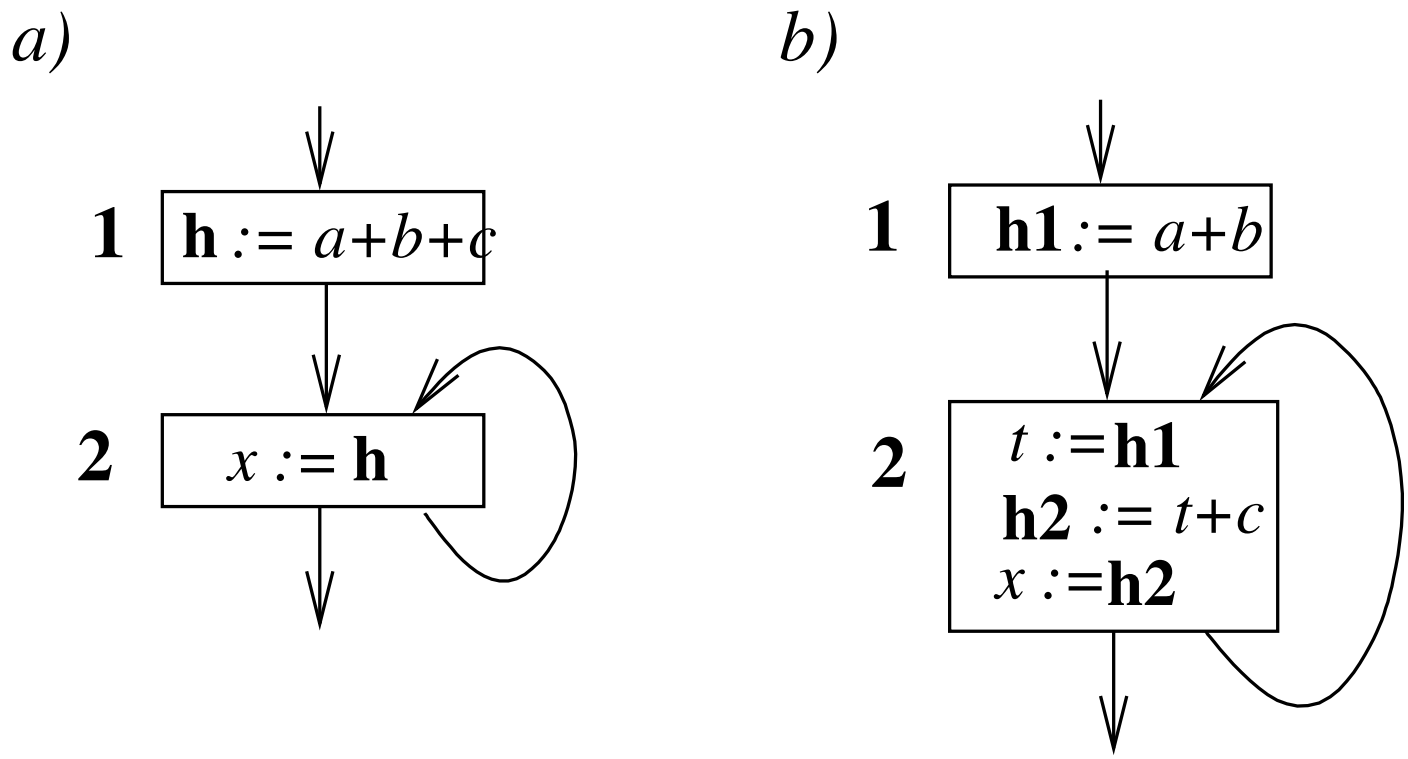
Weitere PREE/PRAE-Phänomene (1)

PREE/PRAE-Phänomene im Zusammenhang mit 3-Adresscode und Nicht-3-Adresscode:



Weitere PREE/PRAE-Phänomene (2)

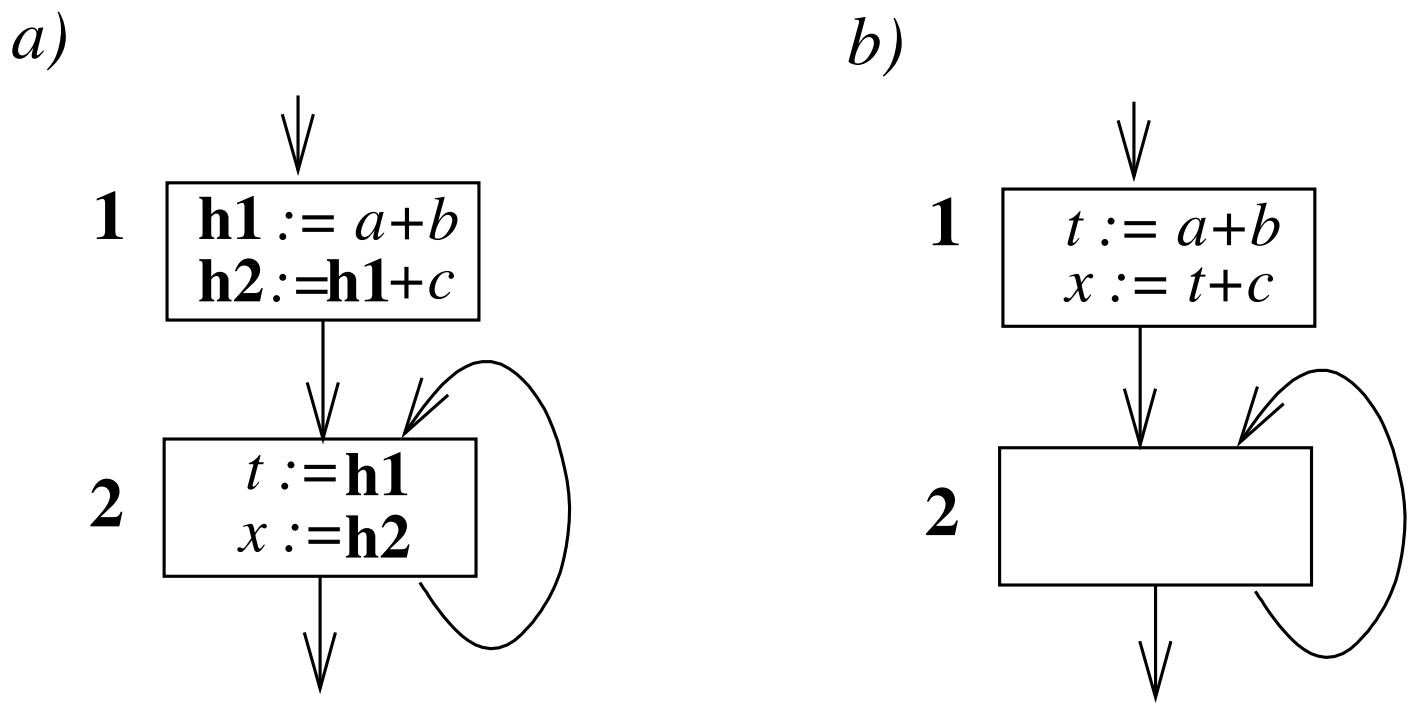
Der Effekt von PREE auf die Programme aus Abb. (a) und (b):



Weitere PREE/PRAE-Phänomene (3)

Die Effekte von

- PREE gefolgt von Konstantenpropagierung (constant propagation (CP)): Abb. (a)
- EAM: Abb. (b)



Kapitel 12.4

Literaturverzeichnis, Leseempfehlungen

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 12 (1)


D. M. Dhamdhere. *Register Assignment using Code Placement Techniques*. Journal of Computer Languages 13(2):75-93, 1988.


D. 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.


D. 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 12 (2)


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.


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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (3)

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 13

Kombination von PRAE und PDCE

Kapitel 13.1

xxx

Motivation

Erinnerung:

Konzeptuell können wir PRAE und PDCE wie folgt verstehen:

- $\mathrm{PRAE} = (\mathrm{AH} + \mathrm{RAE})^*$
- $\mathrm{PDCE} = (\mathrm{AS} + \mathrm{DCE})^*$

Das legt nahe auch die "Summe" aus PRAE und PDCE zu betrachten:

- $\mathrm{AP} = (\mathrm{AH} + \mathrm{RAE} + \mathrm{AS} + \mathrm{DCE})^*$

AP steht dabei für Assignment Placement.

Individuelle PRAE/PDCE-Opt.-Ergebnisse

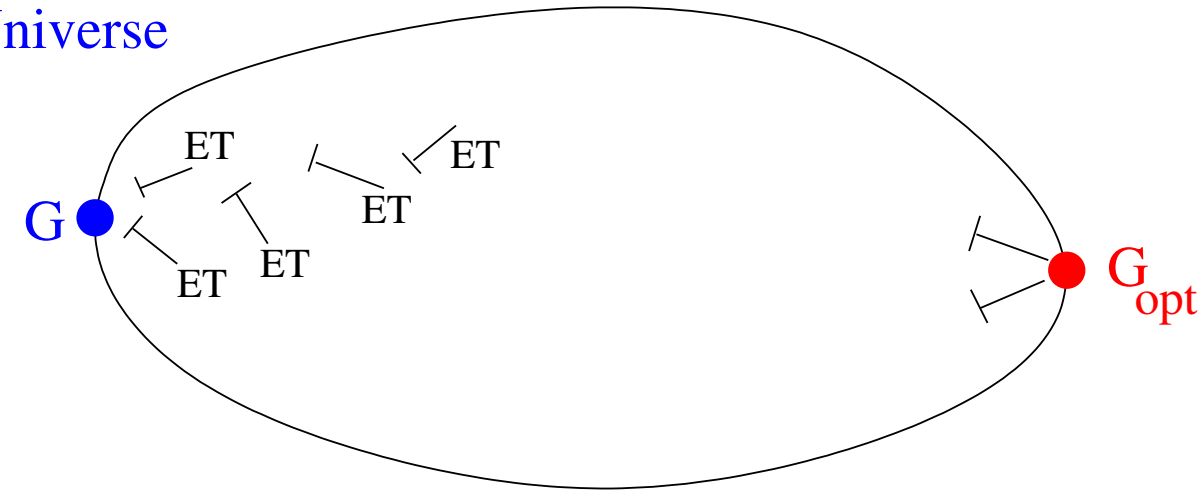
PRAE- bzw. PDCE-Ableitungsrelation $\vdash$:

- PRAE: $G \vdash_{AH,RAE} G'$ (d.h. ET$=_{df}$ {AH,RAE})
- PDCE: $G \vdash_{AS,DCE} G'$ (d.h. ET$=_{df}$ {AS,DCE})

Wir haben gezeigt (s. Kap. 11 und 12):

Theorem 13.1.1 (PRAE-/PDCE-Optimalität)

Die PRAE- und PDCE-spezifischen Ableitungsrelationen $\vdash_{ET}$ sind konfluent und terminierend (und somit optimal).



Kombination von PRAE und PDCE: AP

Betrachte nun:

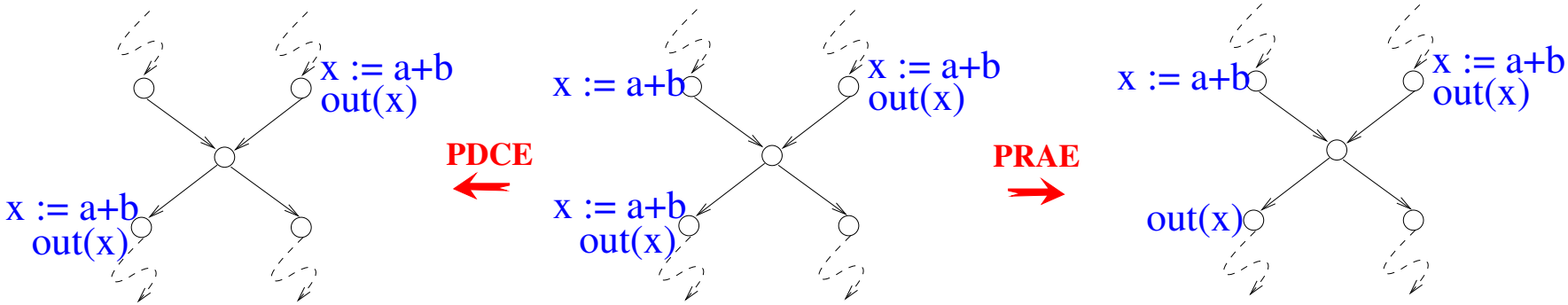
- Assignment Placement AP

$$\mathrm{AP} = (\mathrm{AH} + \mathrm{RAE} + \mathrm{AS} + \mathrm{DCE})^*$$

Erwartung:

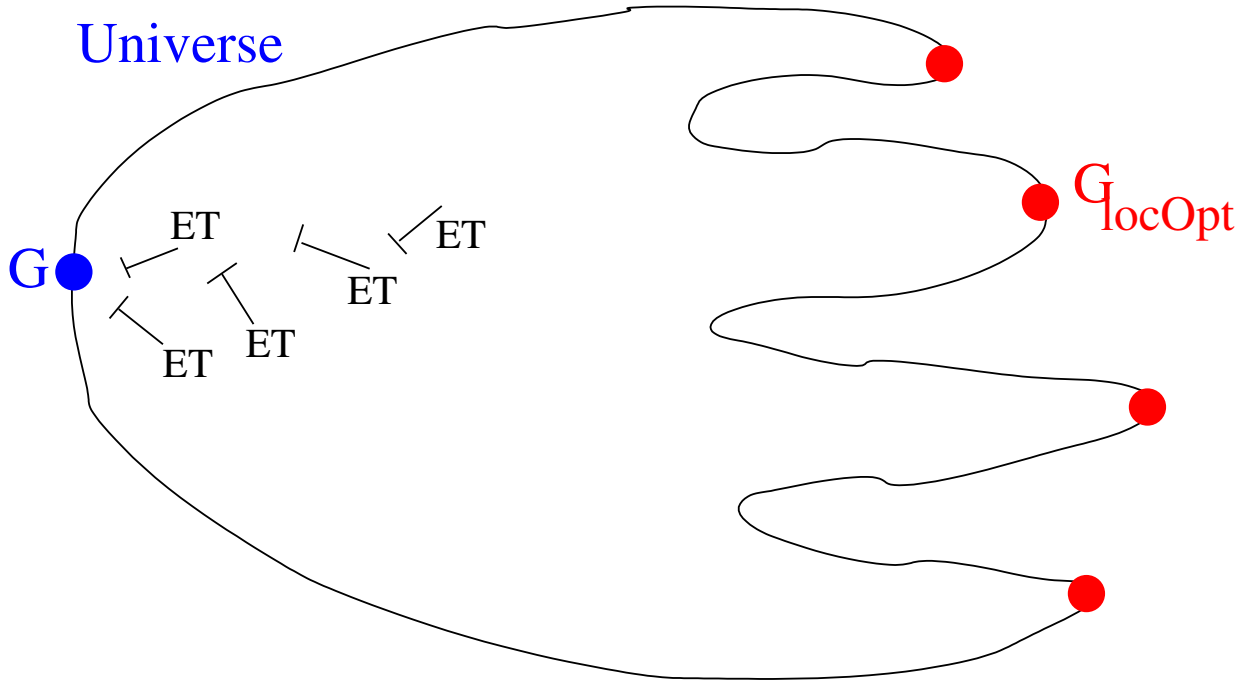
- AP sollte mächtiger sein als PRAE und PDCE individuell!

Das gilt in der Tat. Aber:



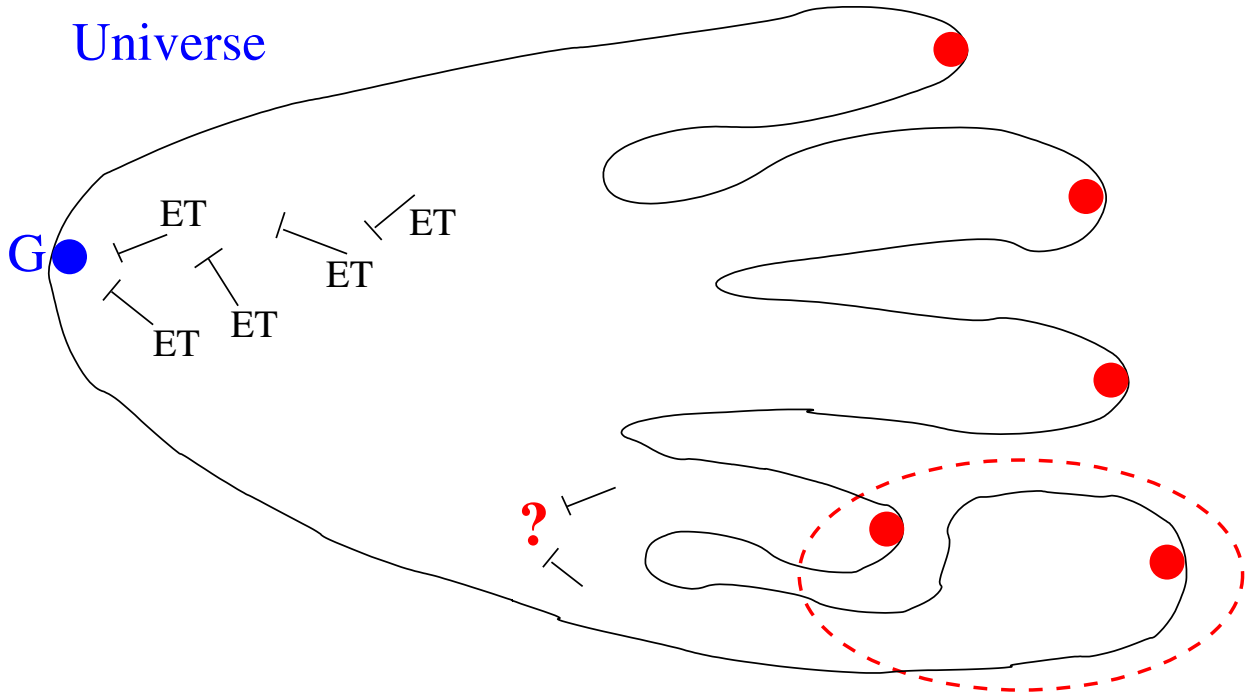
AP: Verlust globaler Optimalität

Konfluenz und damit globale Optimalität sind verloren!



AP: Verlust lokaler Optimalität

In speziellen Szenarien geht sogar lokale Optimalität verloren:



- Knoop, J., and Mehofer, E. Distribution Assignment Placement: Effective Optimization of Redistribution Costs. IEEE Transactions on Parallel and Distributed Systems 13(6):628-647, 2002.

Kapitel 13.2

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 13

Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Code Motion and Code Placement: Just Synonyms?* In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 154-169, 1998.

Jens Knoop, Eduard Mehofer. *Distribution Assignment Placement: Effective Optimization of Redistribution Costs*. IEEE Transactions on Parallel and Distributed Systems 13(6):628-647, 2002.

Kapitel 14

Konstantenfaltung auf dem Wertegraphen

Arbeitsplan

Konstantenfaltung und -ausbreitung

- Hintergrund und Motivation
- Der VG-Konstantenfaltungsalgorithmus
- Der PVG-Konstantenfaltungsalgorithmus

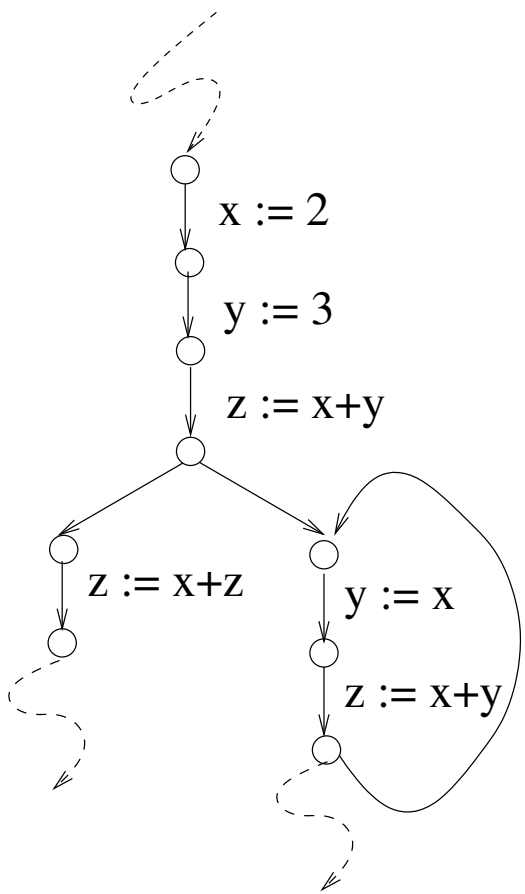
Kapitel 14.1

Motivation

Konstantenfaltung und -ausbreitung

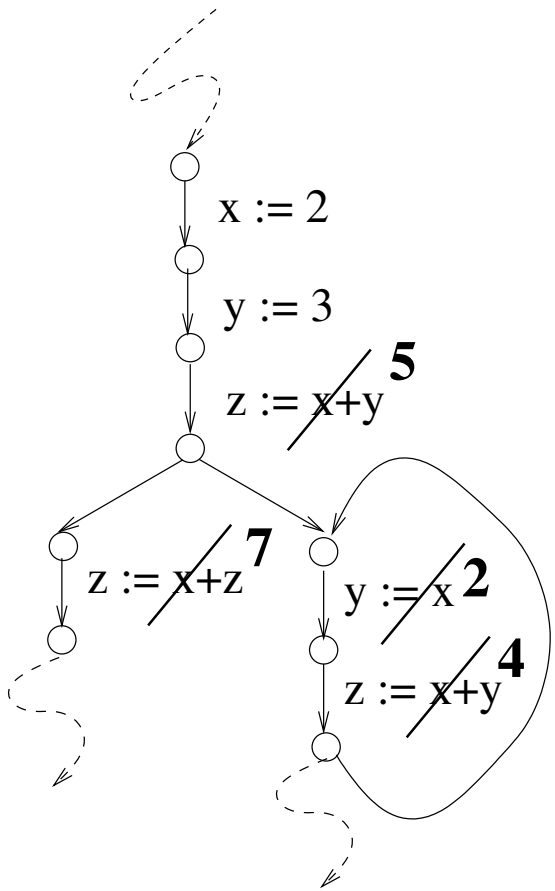
Veranschaulichendes Beispiel

a)



Original program

b)



After simple constant propagation

Ursprung und Entwicklung

...von Algorithmen zu Konstantenfaltung und -ausbreitung.

Prägend:

- Gary A. Kildalls Algorithmus zur Berechnung einfacher Konstanten (engl. simple constants (SC) (POPL'73).

Beachte:

- Der Algorithmus zur Berechnung einfacher Konstanten aus Kapitel 6.6 stimmt im Ergebnis, nicht jedoch in den verwendeten Datenstrukturen mit Kildalls Algorithmus überein.

Erweiterungen von Kildalls SC-Algorithmus (1)

Verbesserungen von Kildalls Algorithmus zielen auf:

- Anwendungsreichweite
  - Interprozedurale Erweiterungen
    - 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 parallele Erweiterungen
    - Lee, Midkiff, Padua (J. of Parallel Prog., 1998)
    - Knoop (Euro-Par'98)
    - ...

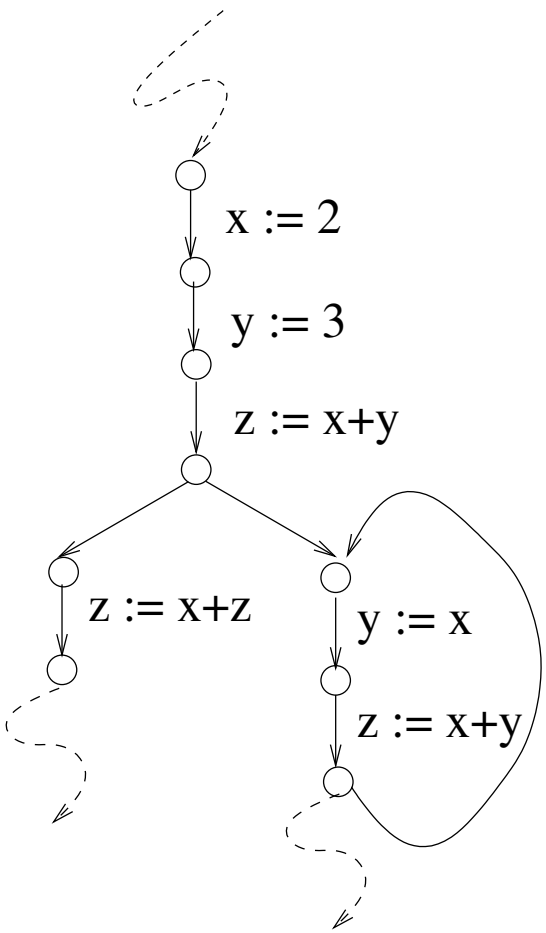
  wobei Anwendungsreichweite zulasten von Ausdruckskraft gewonnen wird: Kopierkonstanten (engl. copy constants), lineare Konstanten (engl. linear constants) anstelle von einfachen Konstanten (engl. simple constants).

Erweiterungen von Kildalls SC-Algorithmus (2)

- Performanz
  - SSA-Form: Wegman, Zadeck (POPL'85)
  - ...
- 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)
  - ...

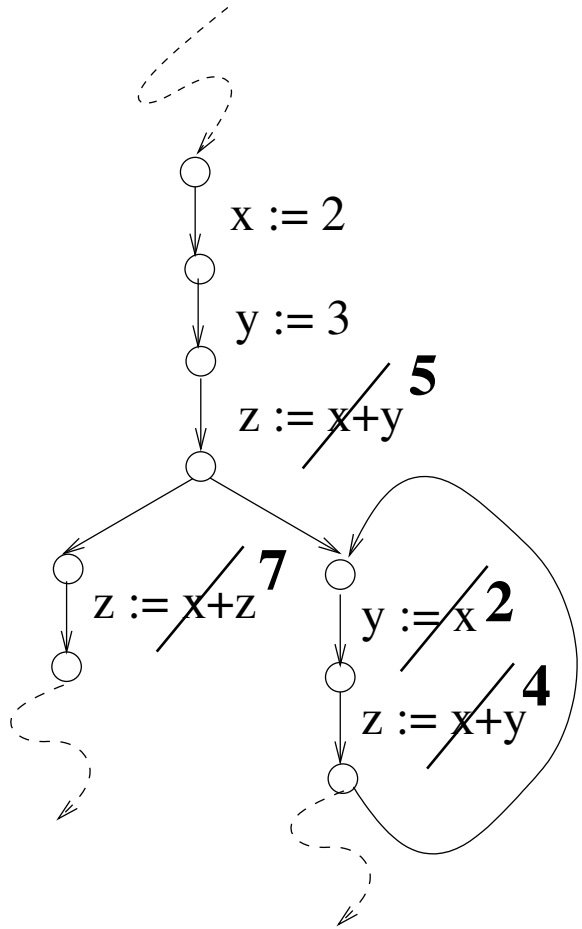
Warum nach größerer Ausdruckskraft streben?

a)



Original program

b)



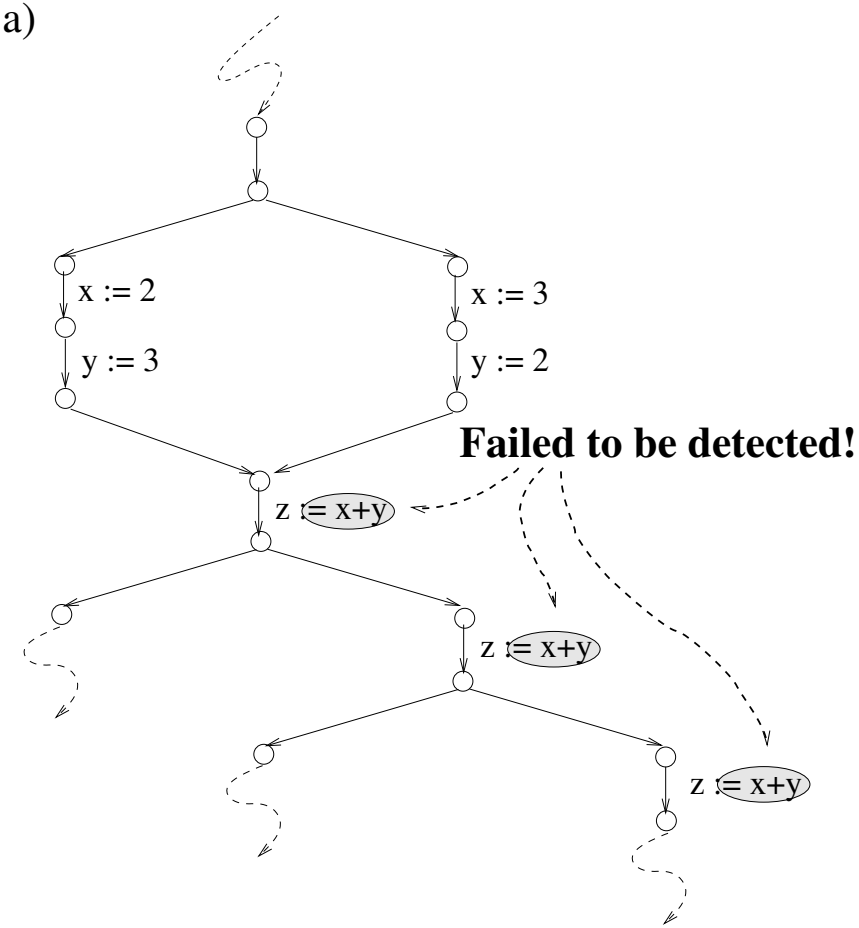
After simple constant propagation

Das SC-Transformationsergebnis ist doch überzeugend, nicht?

Tatsächlich ist es nicht überzeugend

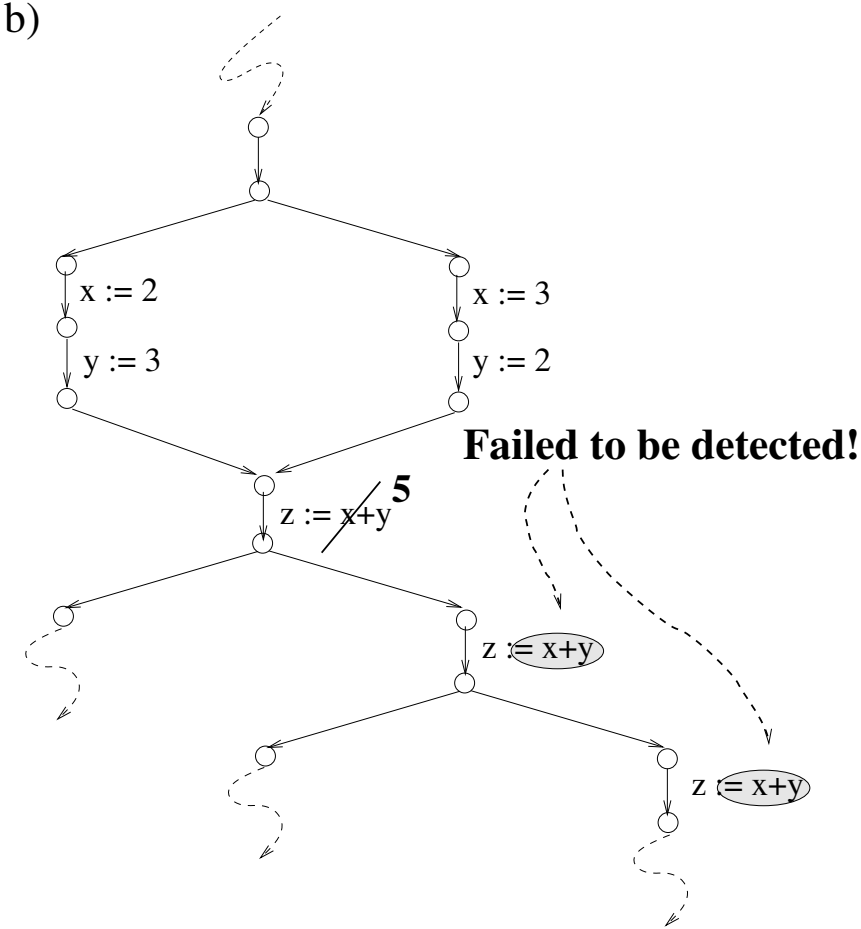
Der Begriff einfacher Konstanten ist schwach:

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 Konstantenfaltung

Es gilt:

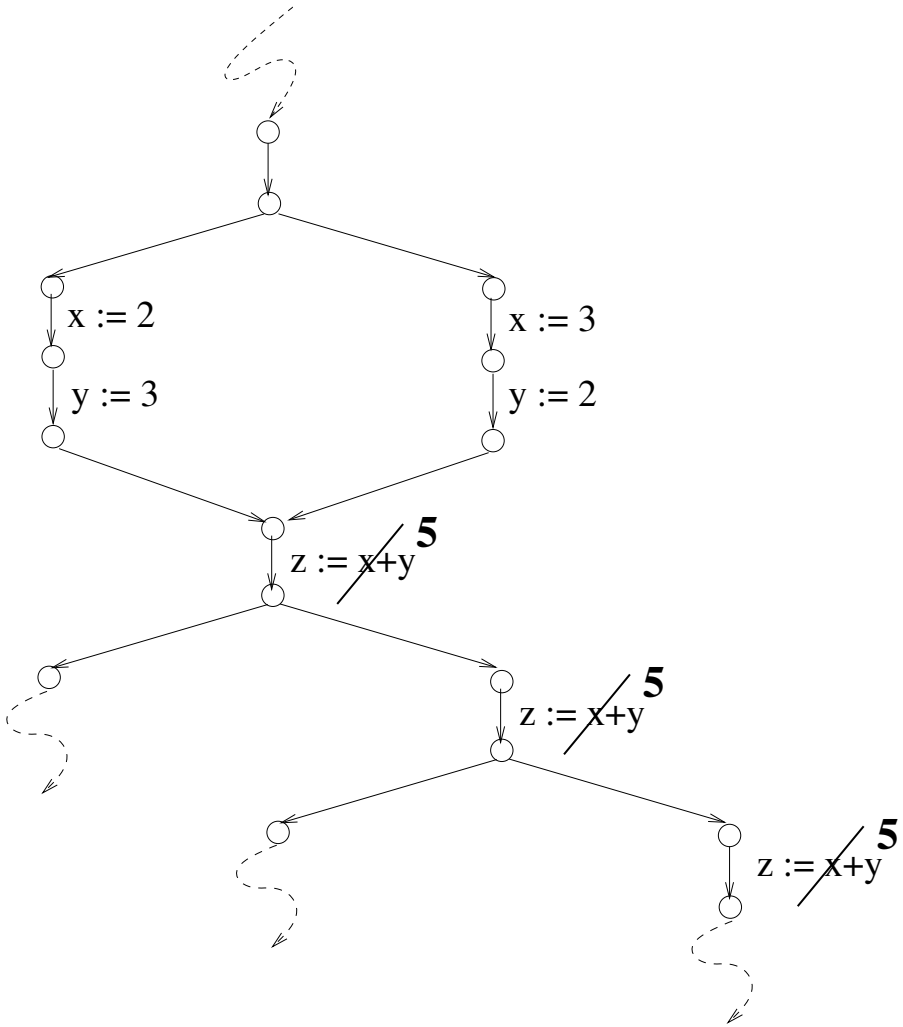
- Konstantenfaltung ist unentscheidbar: Reif, Lewis (POPL 1977)

Andererseits:

- Konstantenfaltung ist entscheidbar auf schleifenfreien Programmen (azyklische Programme (engl. directed acyclic graphs (DAGs)).

Endliche Konstanten (1)

...sind optimal (vollständig) auf schleifenfreien Programmen, d.h. jede Konstante in einem schleifenfreien Programm ist eine endliche Konstante!



Endliche Konstanten (2)

Intuitiv:

- Endliche Konstanten sind eine systematisch, erschöpfend und endlich zu berechnende Erweiterung der von Kam&Ullmanns heuristischer "1-Anweisungsvorschau" erfassten Konstanten.

Schlüsselfakten über endliche Konstanten:

- Für schleifenfreie Programme sind endliche Konstanten optimal.
- Für Programme mit beliebigem Kontrollfluss sind endliche Konstanten eine echte Obermenge einfacher Konstanten.
- Die Berechnungskomplexität endlicher Konstanten ist exponentiell im schlechtesten Fall (auch für schleifenfreie Programme).

Hinsichtlich der Berechnungskomplexität

...endlicher Konstanten sollte man bedenken:

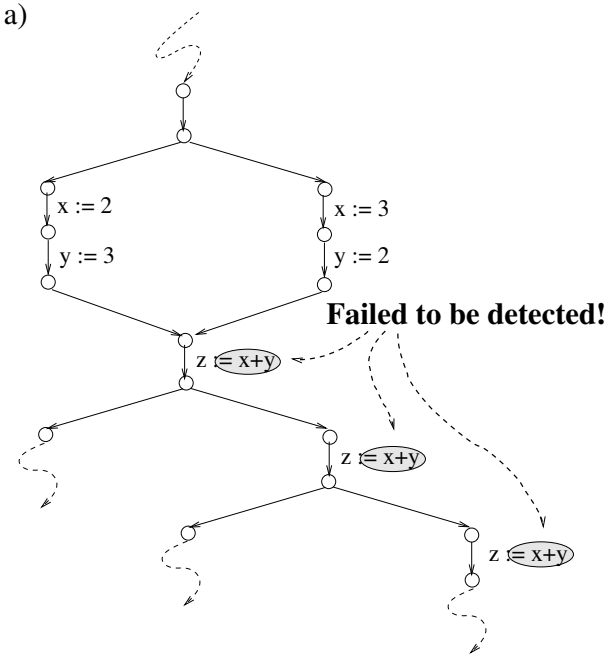
Theorem 14.1.1

Konstantenfaltung und -ausbreitung ist für schleifenfreie Programme co-NP-vollständig.

Knoop, Rüthing (CC'00)
Müller-Olm, Rüthing (ESOP'01)

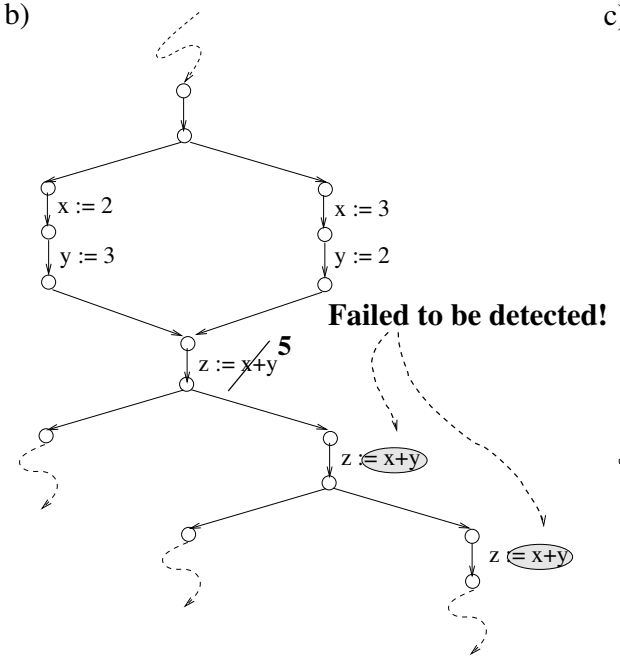
Zurück zum laufenden Beispiel

a)



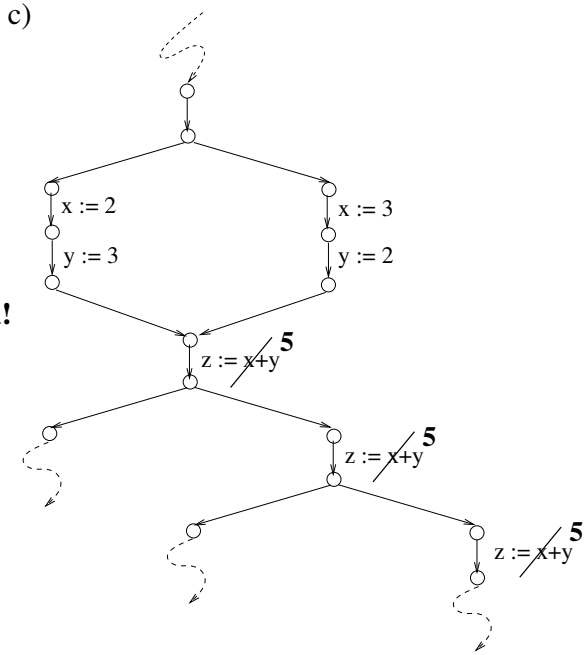
After simple constant propagation
(Note: No effect at all!)

b)



After simple constant propagation
enriched by the "look−ahead−of−one" heuristics
of Kam and Ullman

c)



The effect of the new algorithm

Ein neuer Konstantenfaltungsalgorithmus

...der eine sorgfältige Balance hält zwischen

- Ausdruckskraft und Performanz.

Dieser neue Konstantenfaltungsalgorithmus stützt sich

- auf den Wertegraphen (engl. value graph) von Alpern, Wegman, and Zadeck (POPL'88)
- der sich seinerseits auf eine SSA-Repräsentation von Programmen stützt (SSA = Static Single Assignment Form, Cytron et al. (POPL'89)).

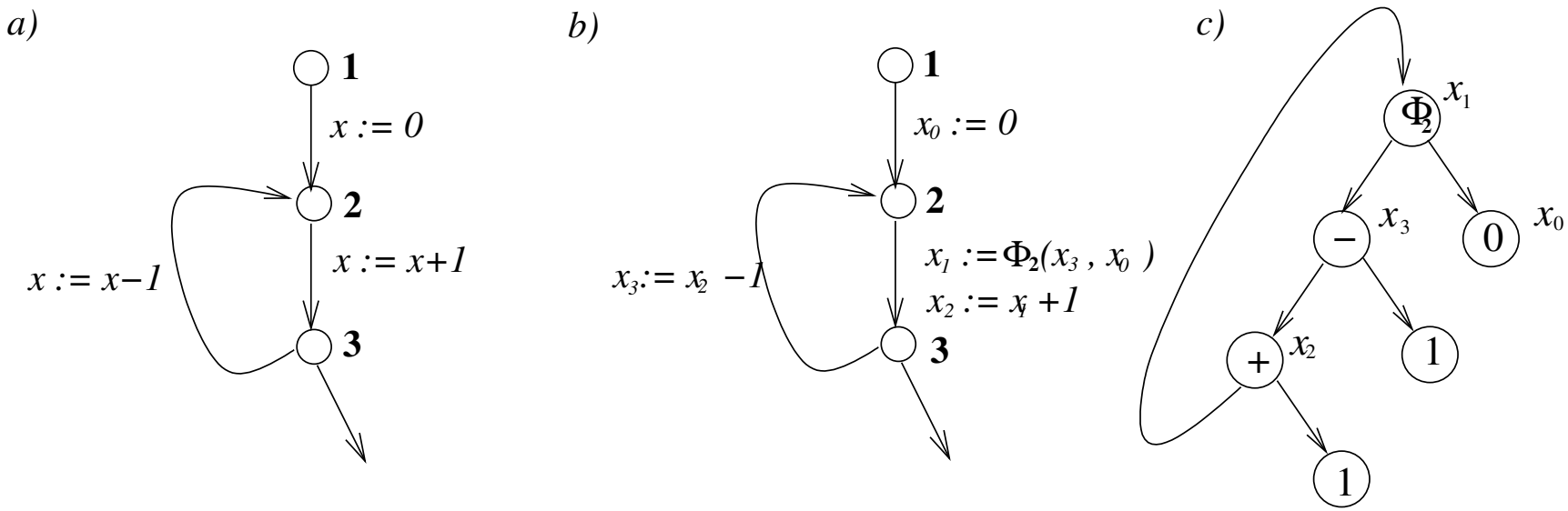
Insgesamt erhalten wir den

- VG-Konstantenfaltungsalgorithmus, einen Konstantenfaltungsalgorithmus mit SSA-Form, nicht auf SSA-Form.

Kapitel 14.2

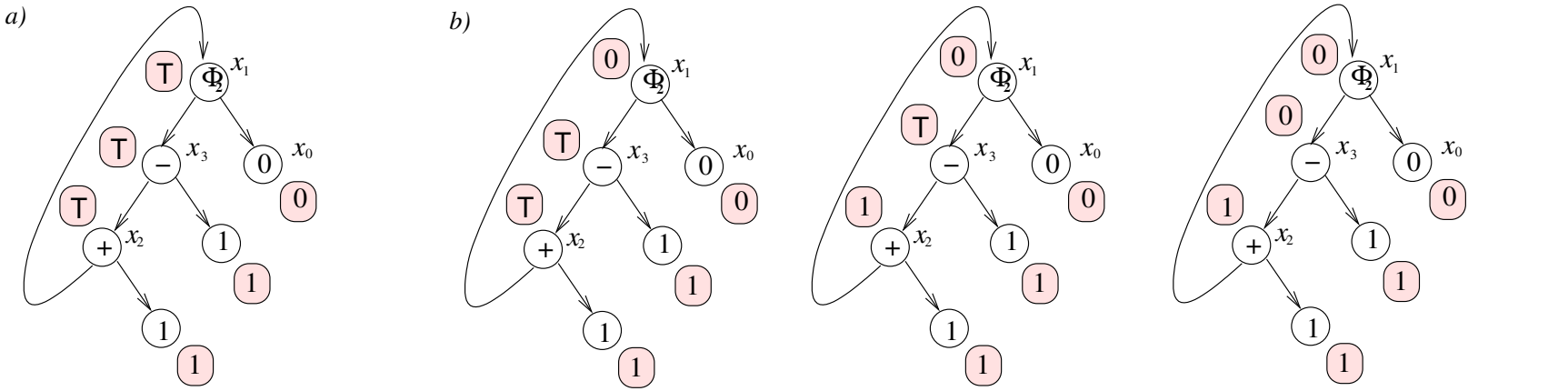
Der VG-Konstantenfaltungsalgorithmus

The Wertegraph von Alpern, Wegman, Zadeck



Ausgangsprogramm SSA-Form Wertegraph

Konstantenfaltung auf dem Wertegraphen



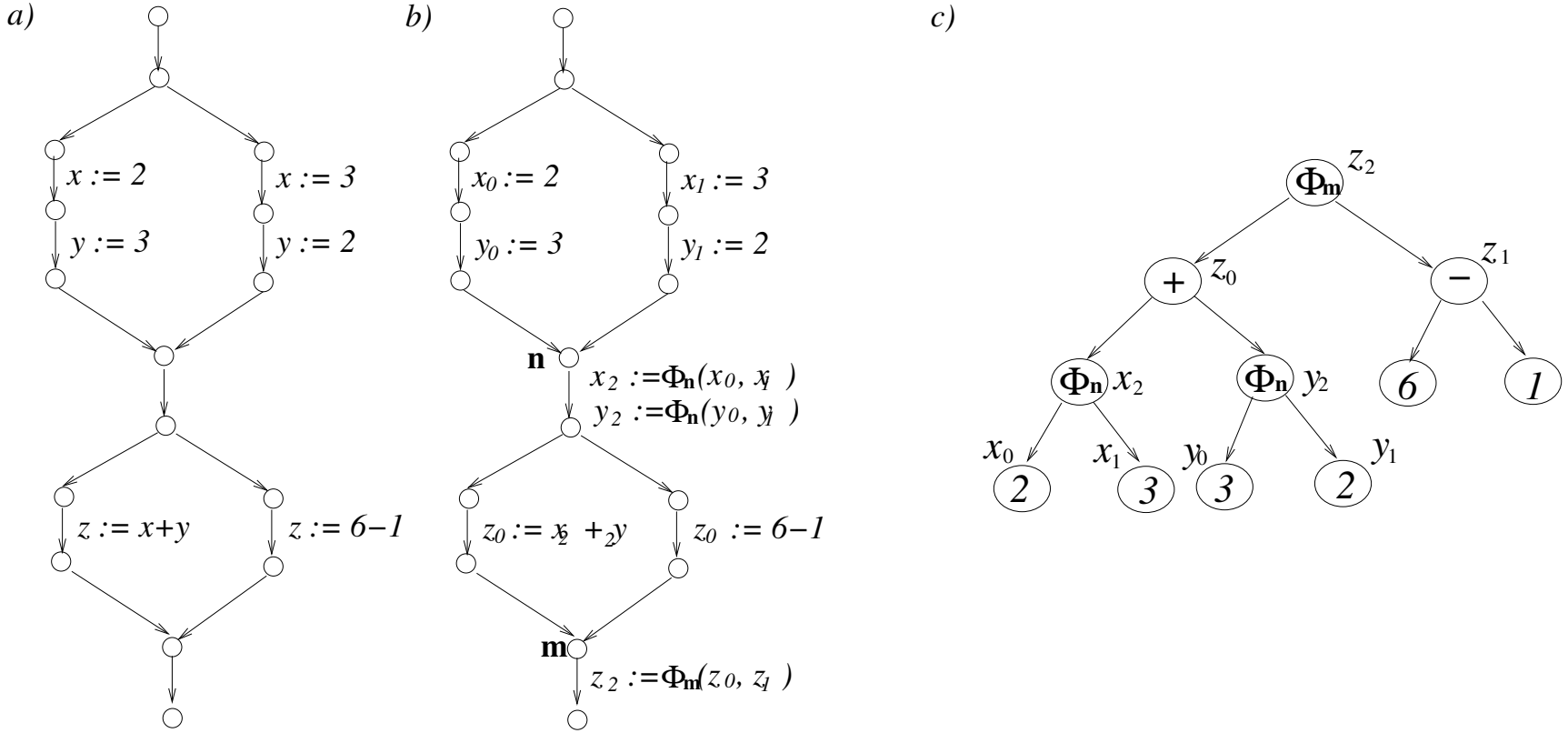
Analyseresultat: x_2 und x_3 sind von konstantem Wert!

Konstantenfaltung auf dem Wertegraphen

...mit zwei Ausprägungen:

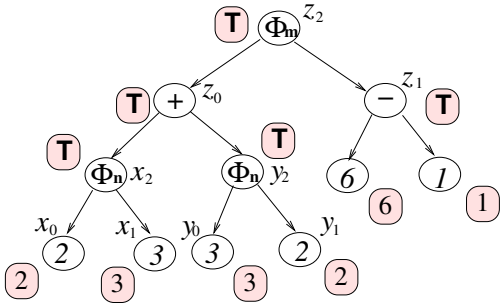
- Der VG-Grundalgorithmus
  ...berechnet einfache Konstanten.
- Der volle VG-Algorithmus
  ...geht über einfache Konstanten und die 1-Vorschauheuristik hinaus.

Beispiel zur Veranschaulichung des vollen VG-Algorithmus

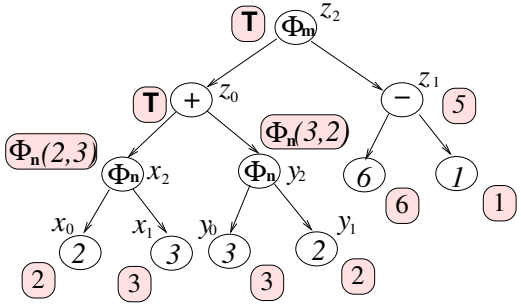


Ausgangsprogramm SSA-Form Wertegraph

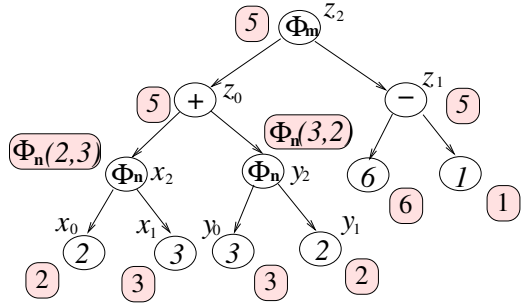
Der volle VG-Algorithmus auf dem Wertegraphen



The start annotation



After the first iteration



After the second iteration. Stable!

Der Clou:

- Einführung von Φ-Konstanten und
- Anpassung der Evaluationsfunktion auf Wertegraphen!

Hauptergebnisse

Für beliebigen, nicht eingeschränkten Kontrollfluss gilt:

- Der volle Algorithmus entdeckt eine Obermenge einfacher Konstanten.

Für azyklischen, schleifenfreien Kontrollfluss gilt:

- Der volle Algorithmus entdeckt die Konstanz jeden Terms, der ausschließlich aus in seinen relevanten Argumenten injektiven Operatoren aufgebaut ist.

Insgesamt gilt:

- Der volle Algorithmus erreicht eine ausgewogene Balance zwischen Ausdruckskraft und Performanz.
- Die Abstützung auf SSA-Form und Wertegraph sind dafür essentiell.

Literaturhinweis

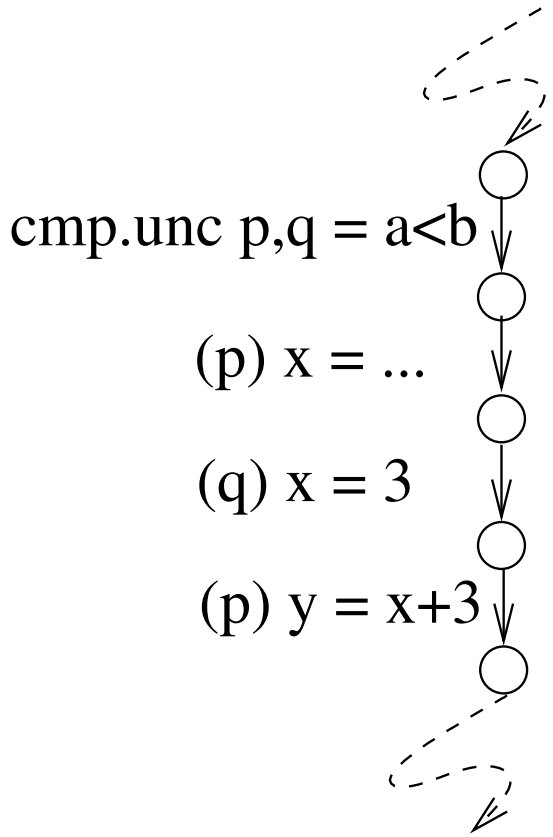
...zum VG-Konstantenfaltungsalgorithmus:

- 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.

Kapitel 14.3

Der PVG-Konstantenfaltungsalgorithmus

Prädikatierter Code

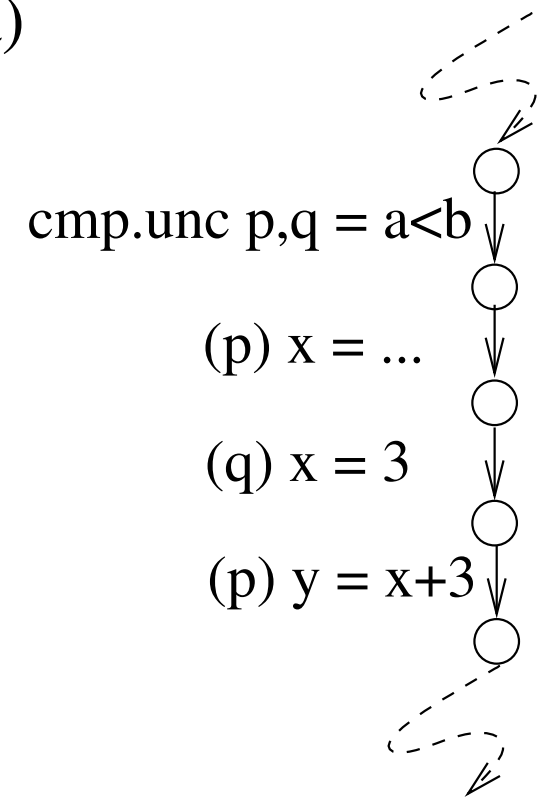


...als Resultat sog. if-Konversion.

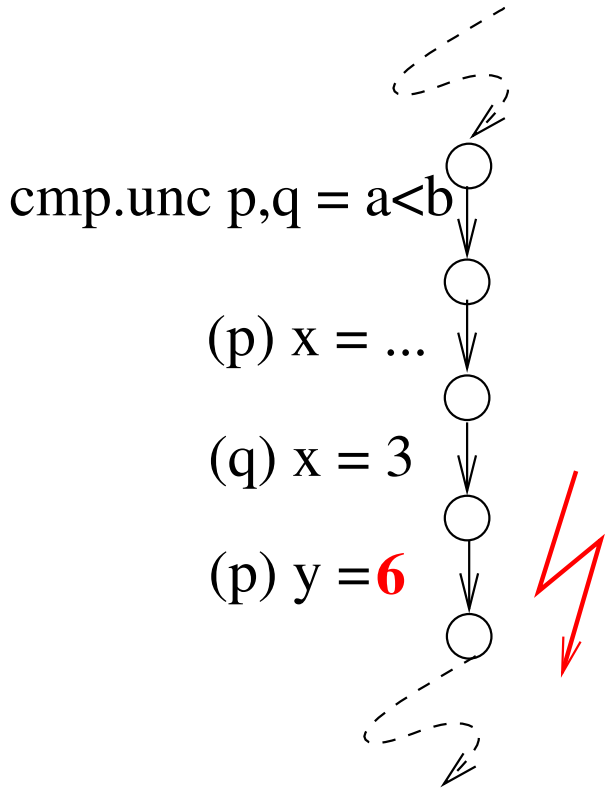
Naive Konstantenfaltung

... auf prädikatiertem Code schlägt fehl:

a)



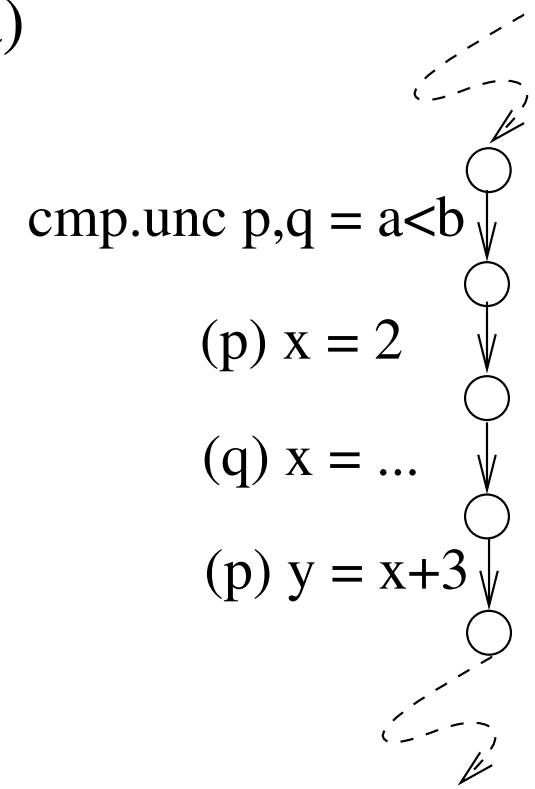
b)



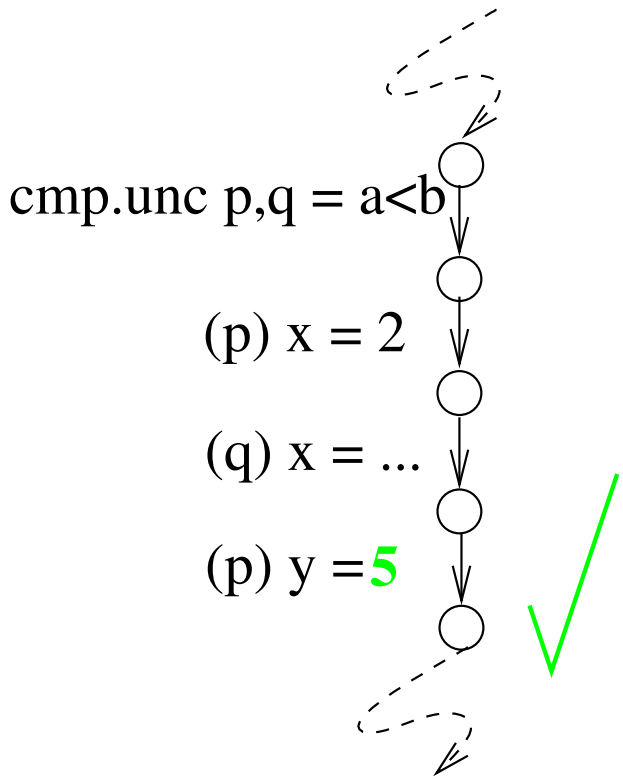
Naive korrekte Konstantenfaltung

...scheint andererseits zu konservativ und zu viele Transformationsmöglichkeiten auszulassen:

a)



b)



Ziel

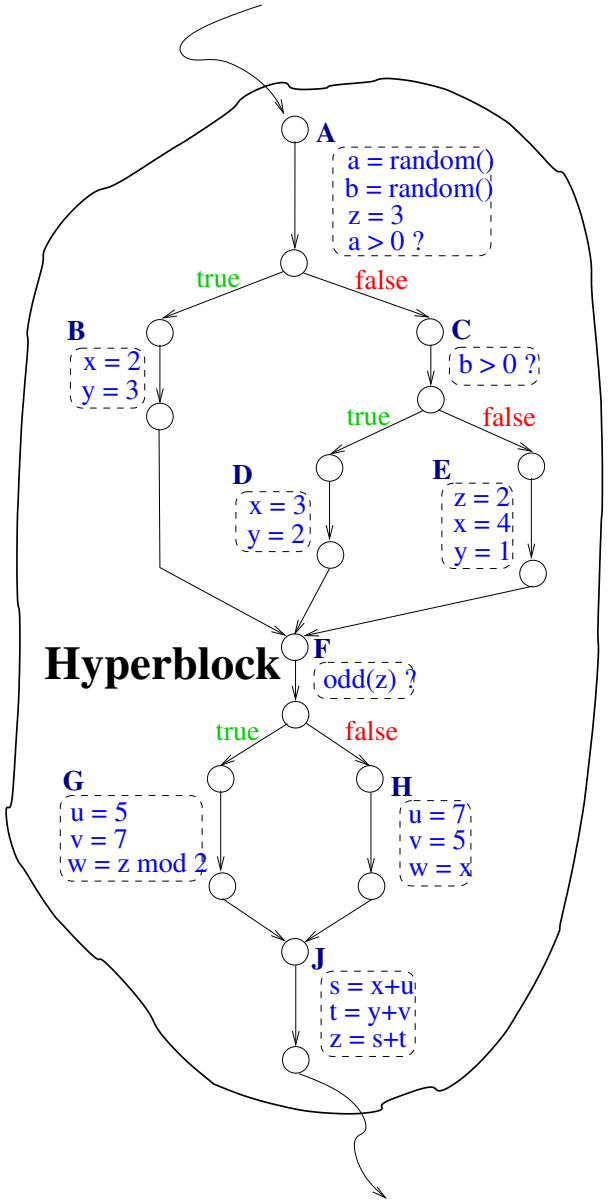
Ein intelligenterer Umgang mit prädikatiertem Code.

Hyperblöcke sind

- wichtige Komponenten prädikatierten Codes.

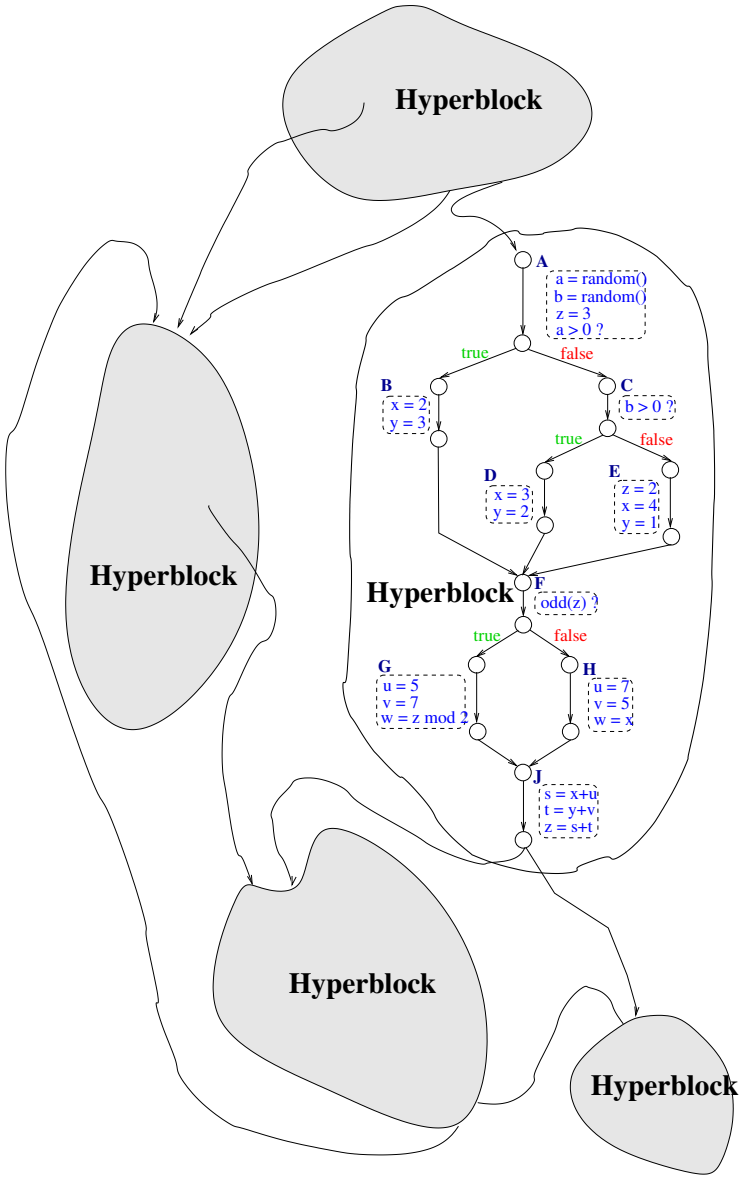
Hyperblöcke

...ein Eintrittspunkt, mehrere Austrittspunkte:



Hyperblockzerlegung eines Programms

Unser durchgehendes Beispiel:



Der PVG-Konstantenausbreitungsalgorithmus

...mit zwei$^+$ Ausprägungen:

- Der PVG-Grundalgorithmus
- Der volle PVG-Algorithmus

zuzüglich einiger

- performanz-gesteigerter Varianten.

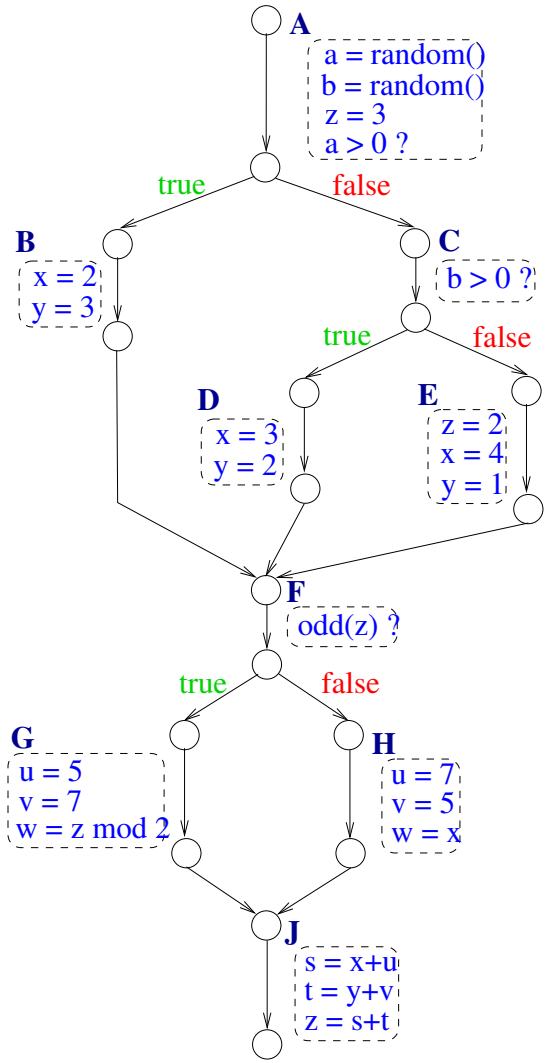
Alle diese Algorithmenvarianten bestehen jeweils aus einer

- lokalen und
- globalen

Analysestufe.

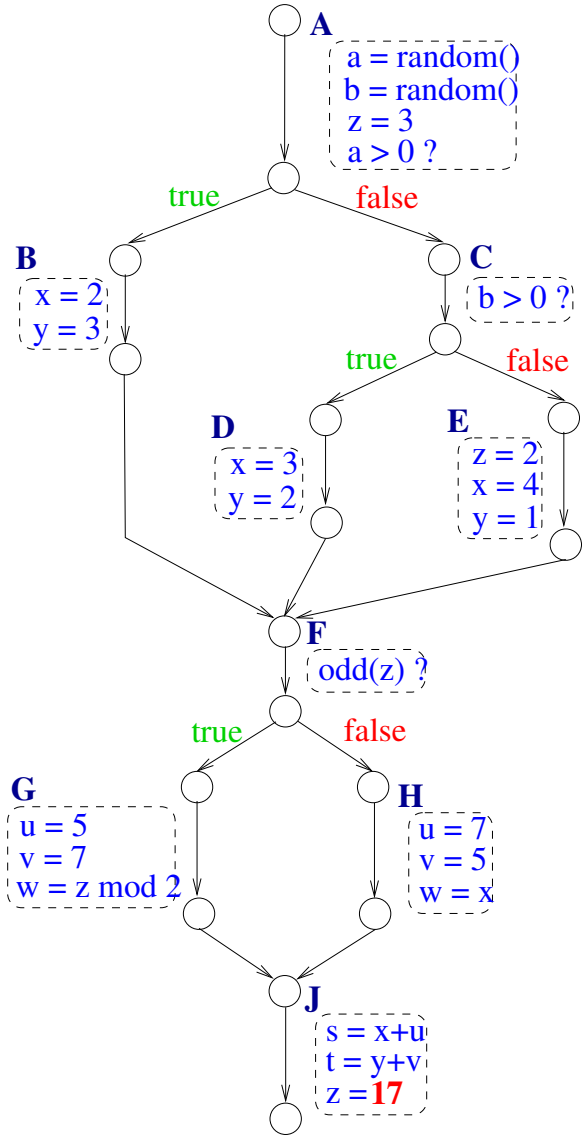
Diskussion der lokalen Analysestufe

Dazu betrachten wir folgenden Hyperblock:



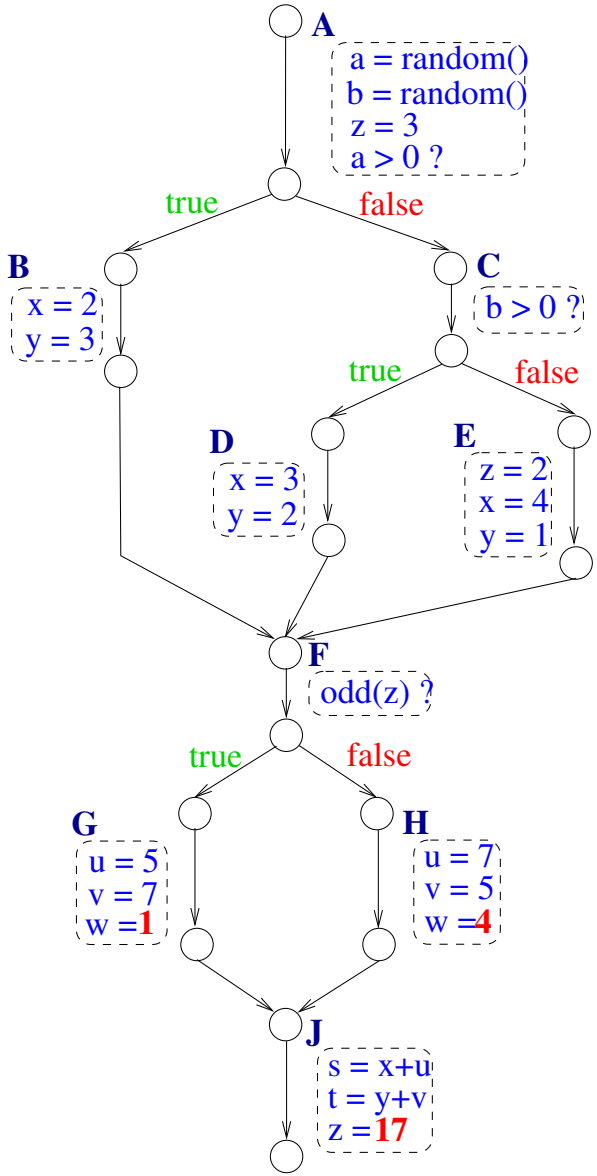
Original Hyperblock

Die PVG-Grundalgorithmustransformation



The Non–Deterministic Path–Precise Basic Optimization

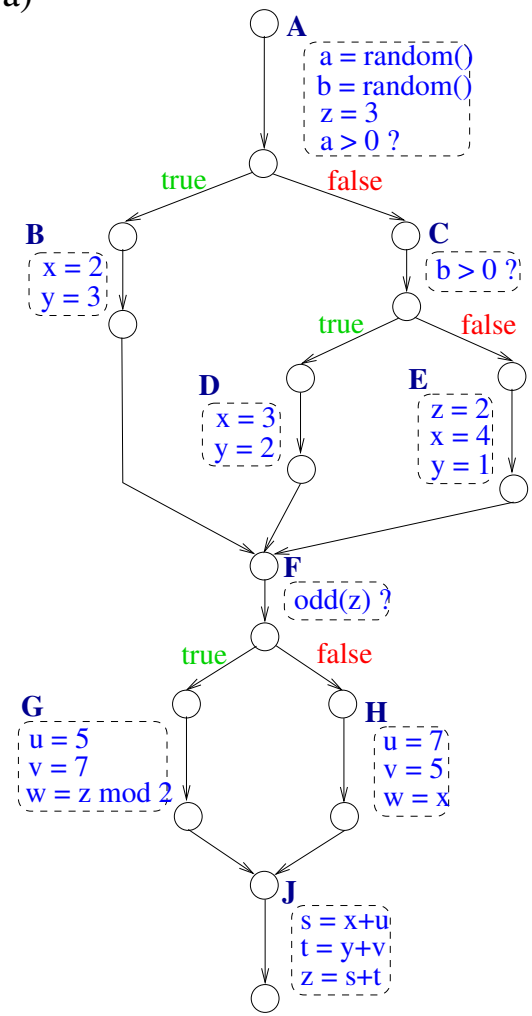
Die volle PVG-Algorithmustransformation



The Deterministic Path–Precise Full Optimization

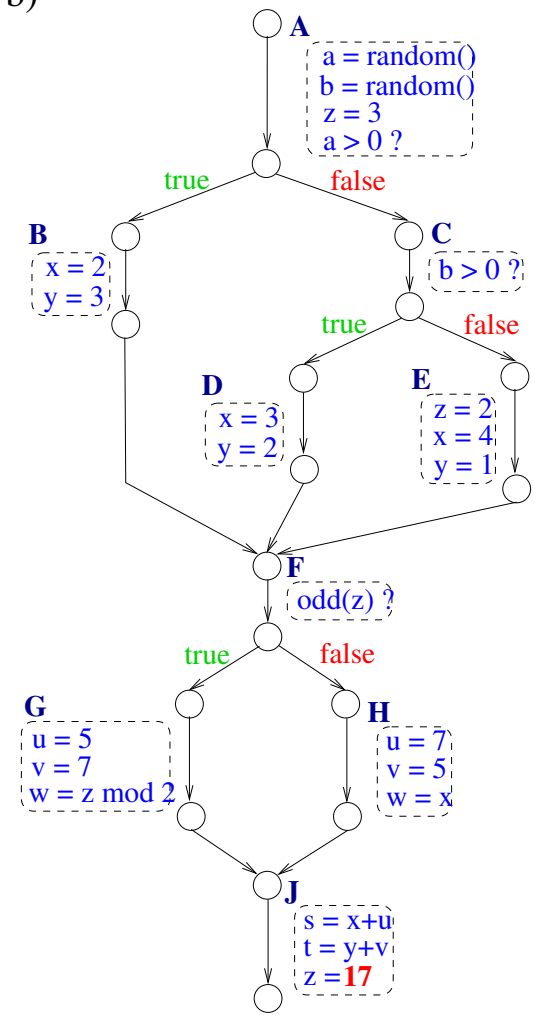
Ausgangsprogramm & beide Transformationen

a)



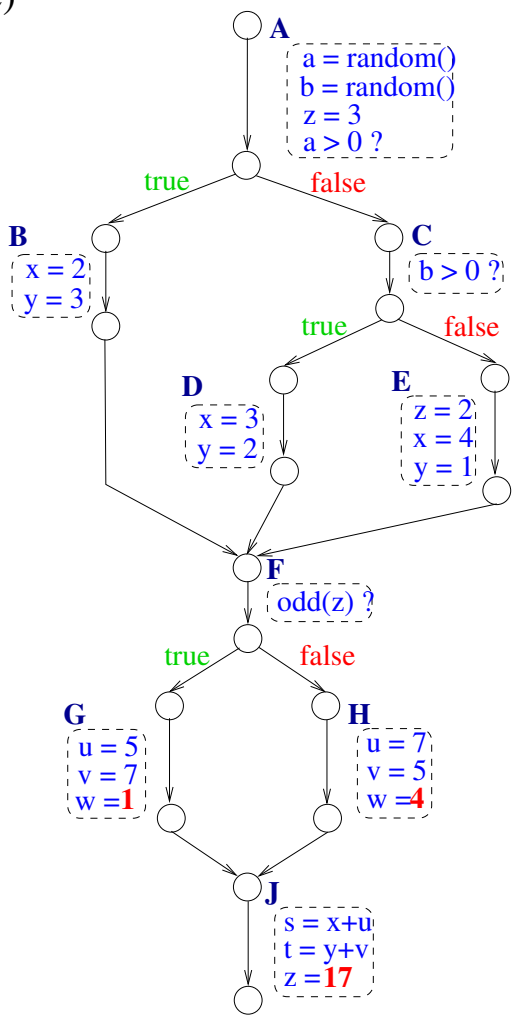
Original Hyperblock

b)



The Non–Deterministic Path–Precise Basic Optimization

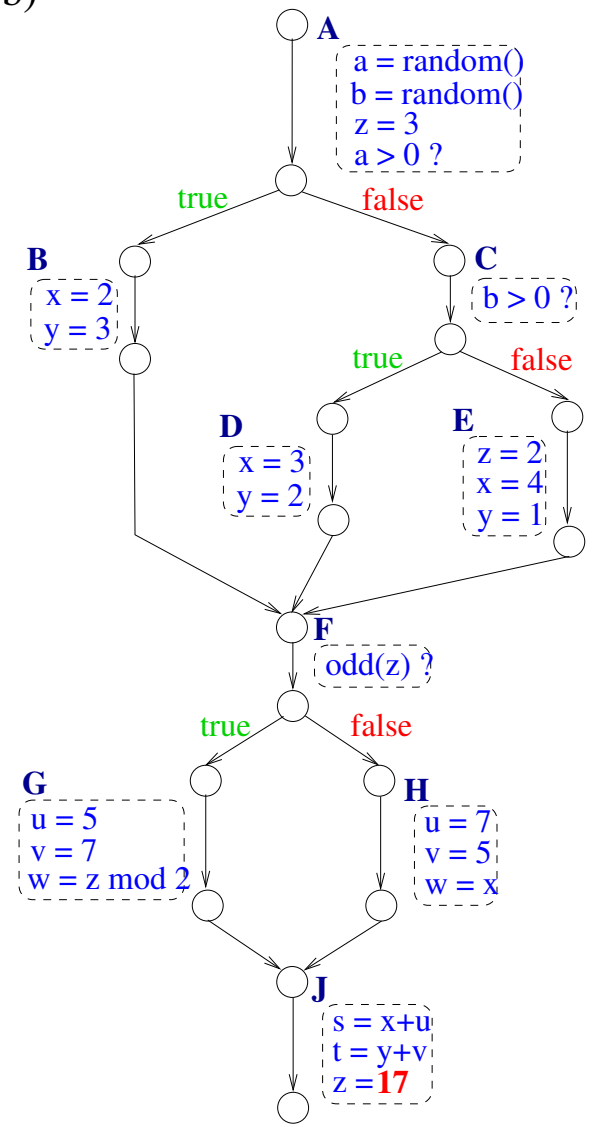
c)



The Deterministic Path–Precise Full Optimization

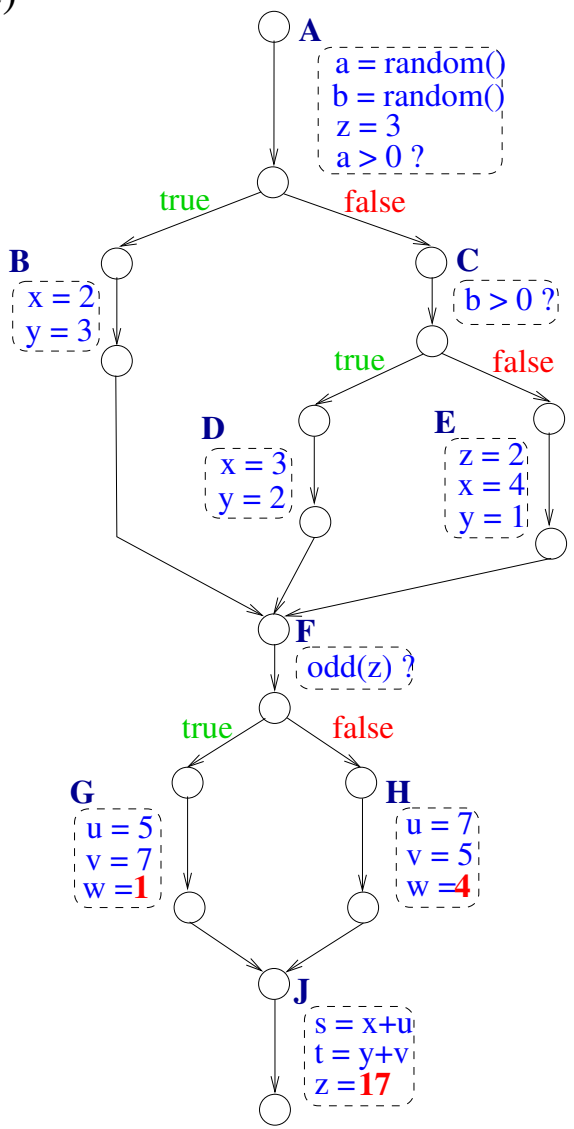
Beide Transformationen auf einen Blick

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.
```

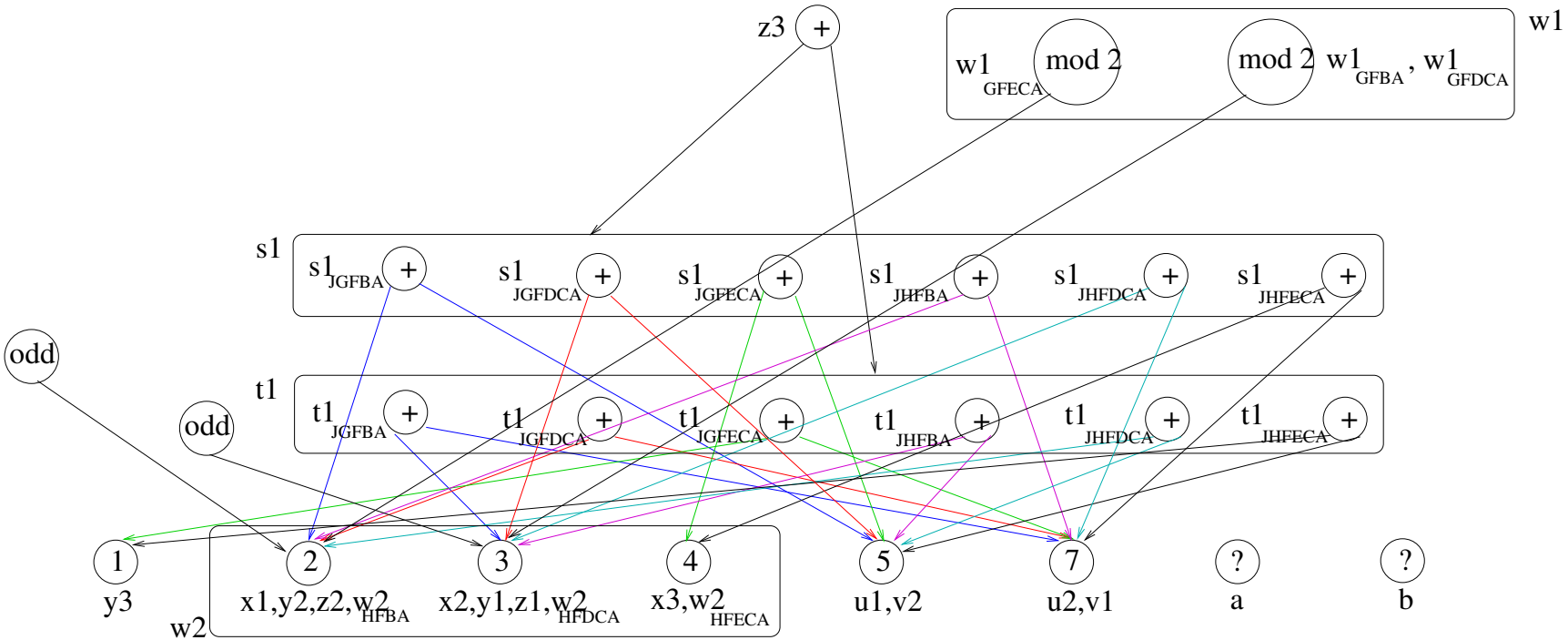
Prädikatierte SSA-Form (PSSA-Form)

...von Carter, Simon, Calder, Ferrante (PACT'99):

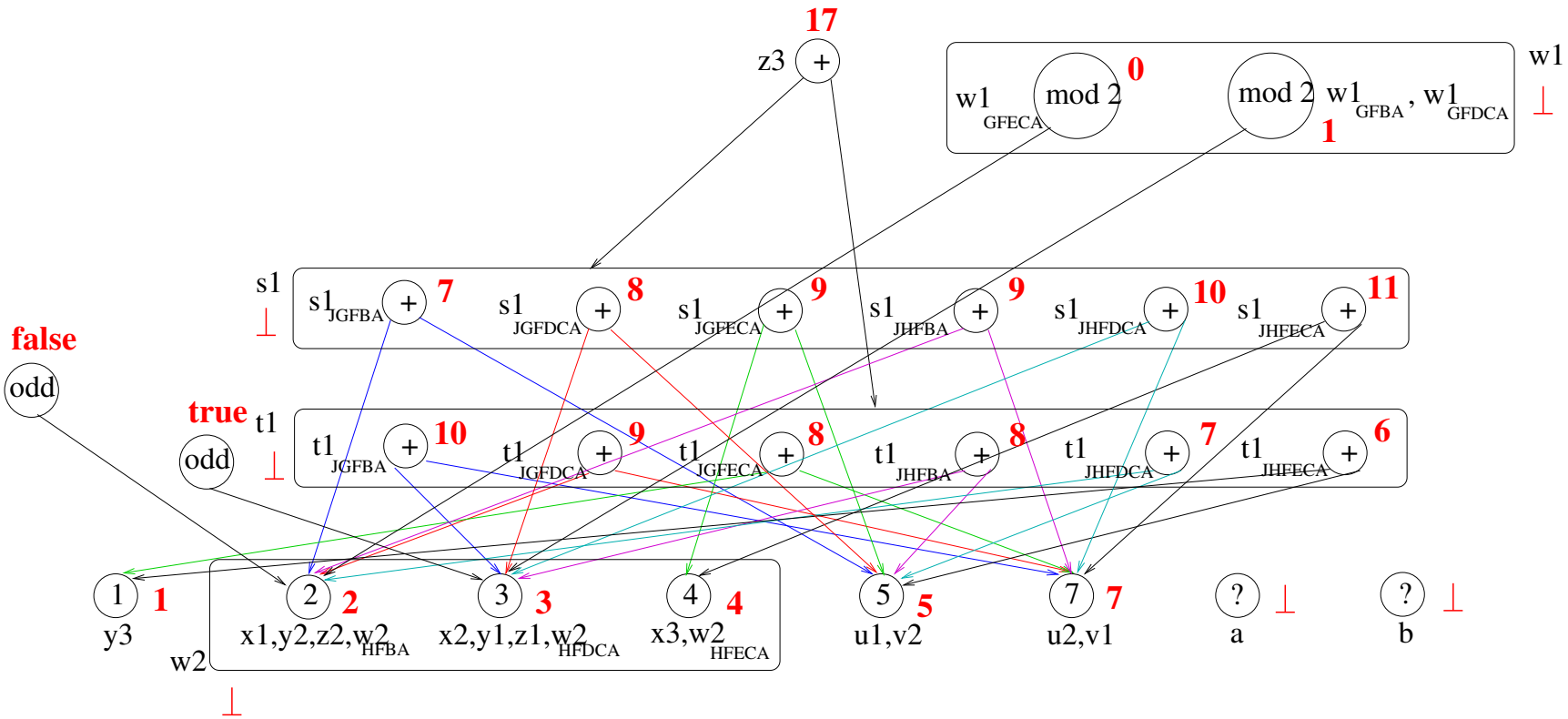
```
begin (p0)     A = OR(TRUE);                    | [*] (HFBA)   w2 = x1;
      (A)      (a1,b1) = (random(),random());   | [*] (HFDCA)  w2 = x2;                 |
      (A)      z1 = 3;                          |     (HFECA)  w2 = x3;
      (A)      cmp.unc BA,CA (a1>0);            |     (H)      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;                          |
```

Die Grundform des prädikatierten Wertegraphen auf PSSA-Grundlage

...ohne Ausnutzung der Wächterprädikate (engl. guarding predicates):

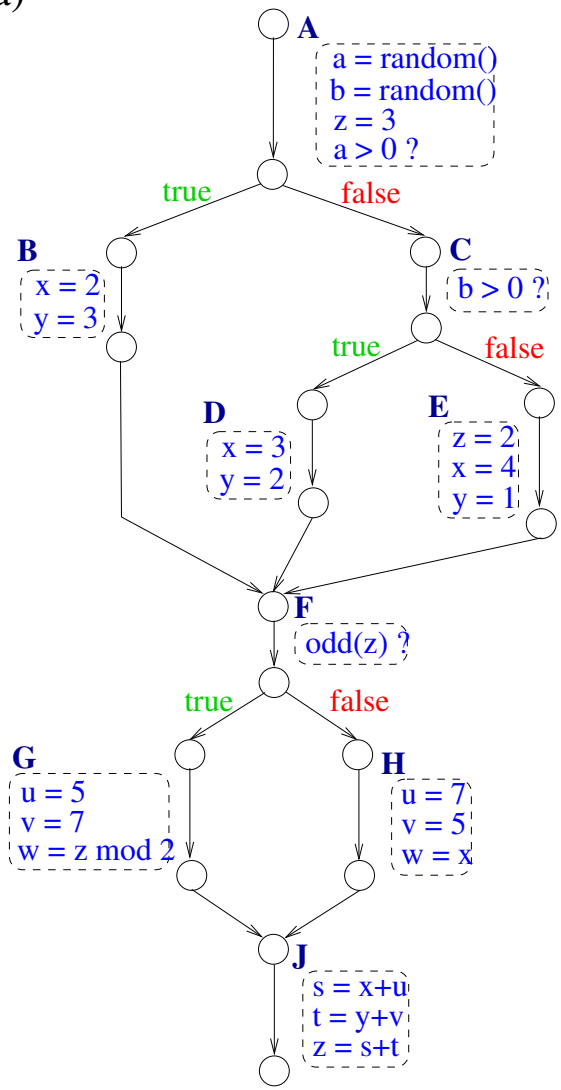


Nach Konstantenfaltung durch PVG-Grundalg.



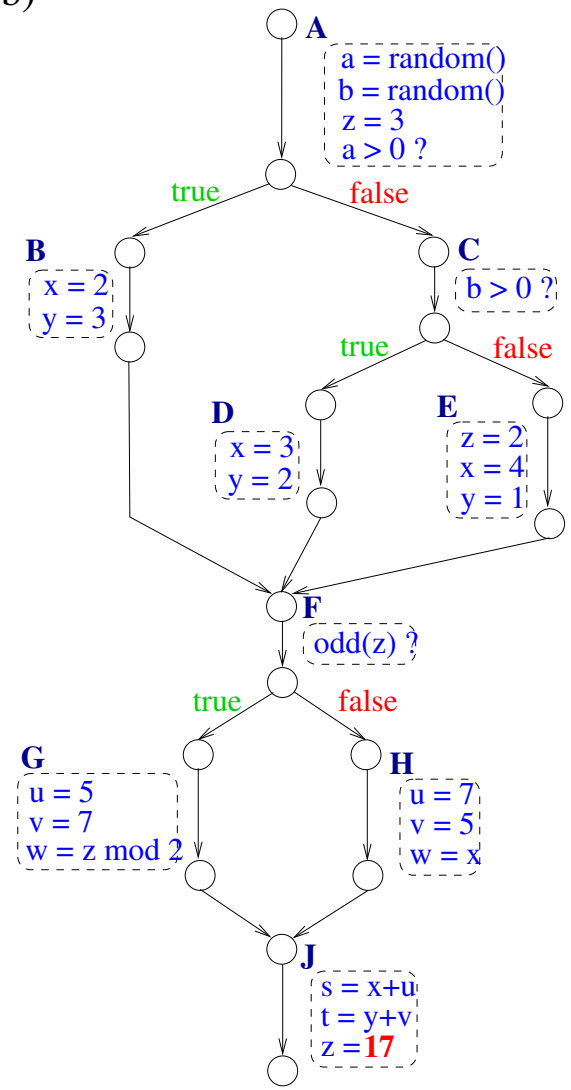
Die PVG-Grundalgorithmustransformation

a)



Original Hyperblock

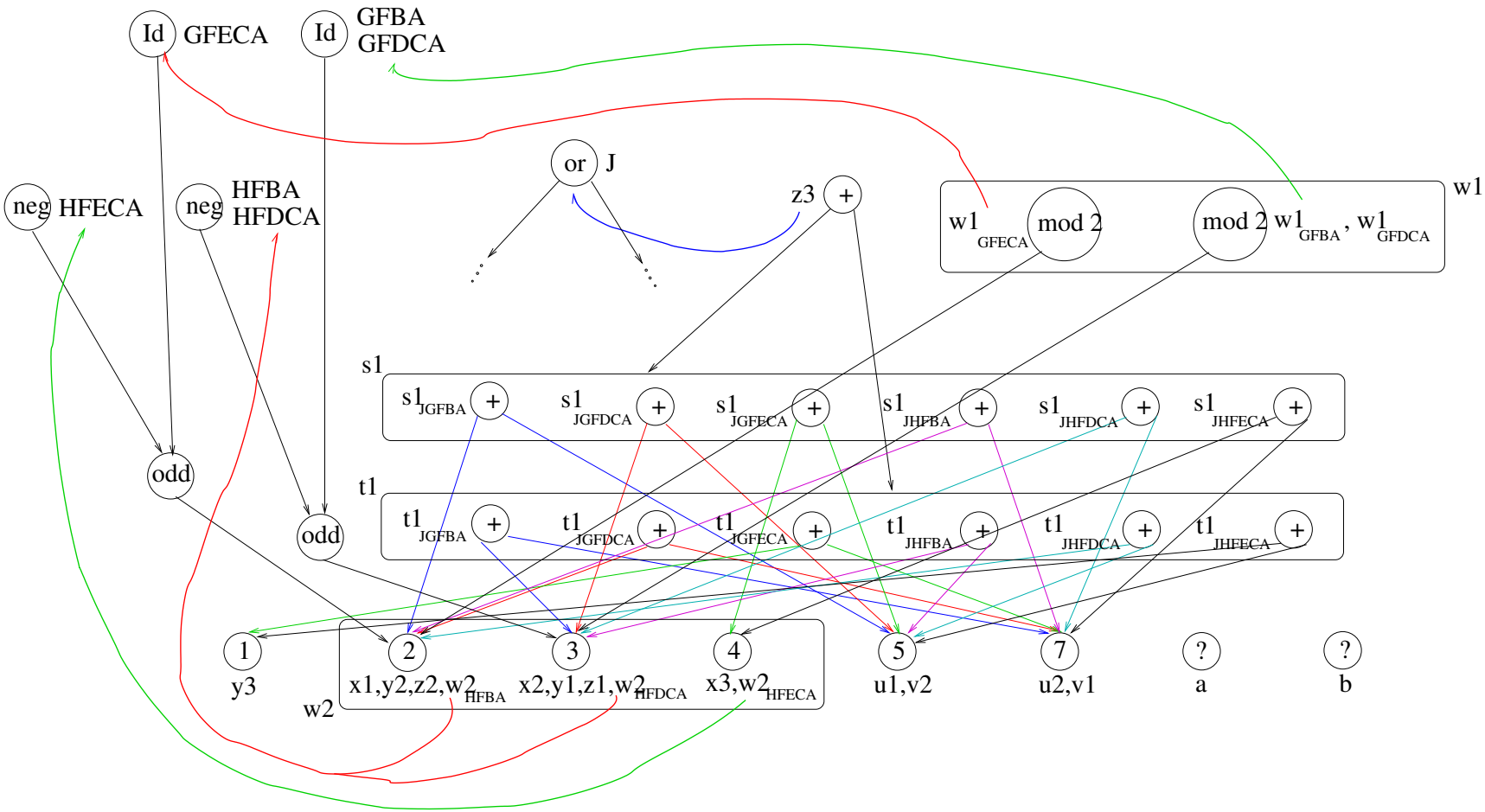
b)



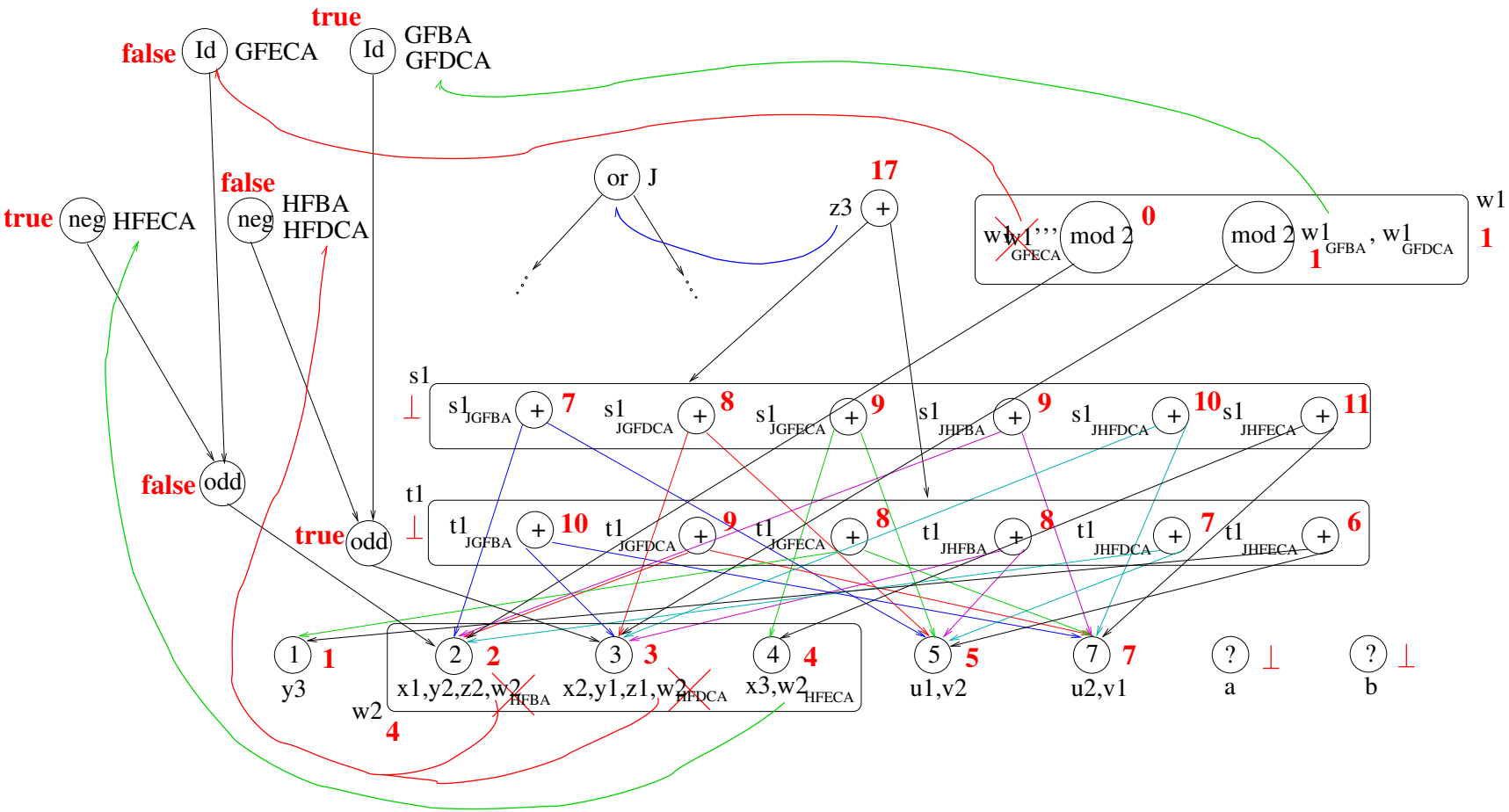
The Non–Deterministic Path–Precise Basic Optimization

Der prädikatierte Wertegraph

...unter Ausnutzung der Wächterprädikate (engl. guarding predicates):

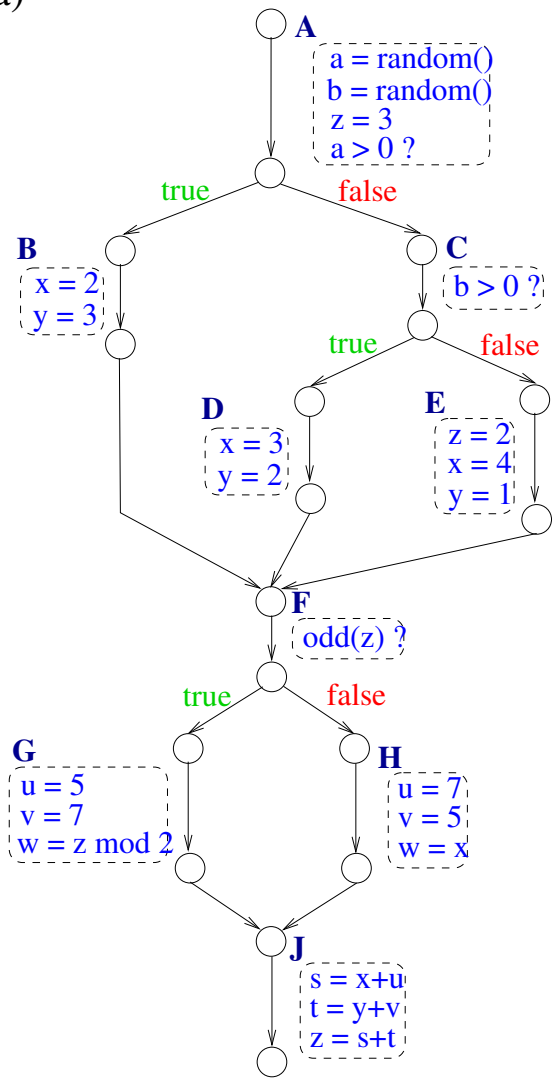


Nach Konstantenfaltung durch vollen PVG-A.



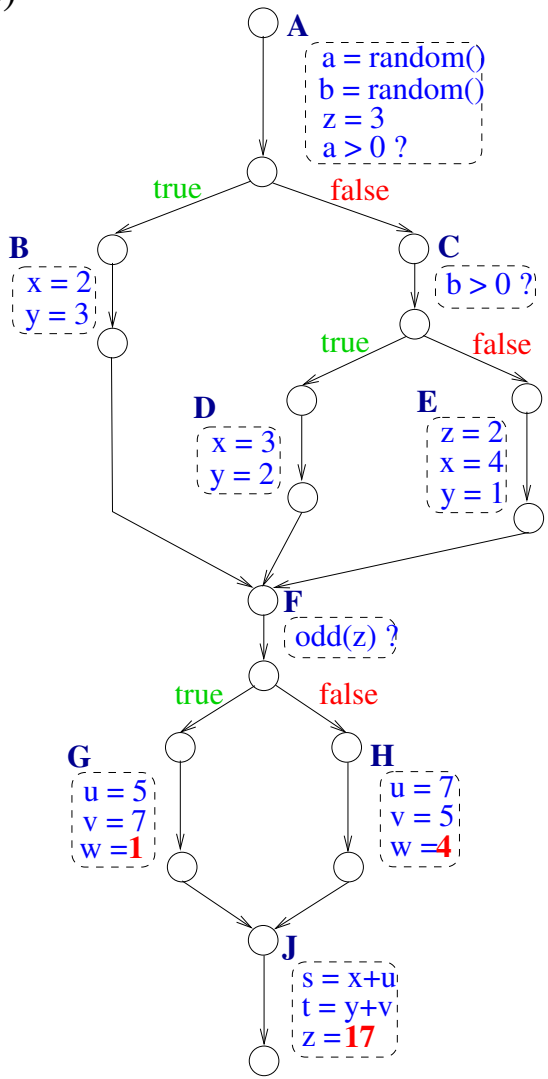
Die volle PVG-Algorithmustransformation

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);  |
```

Hauptergebnisse

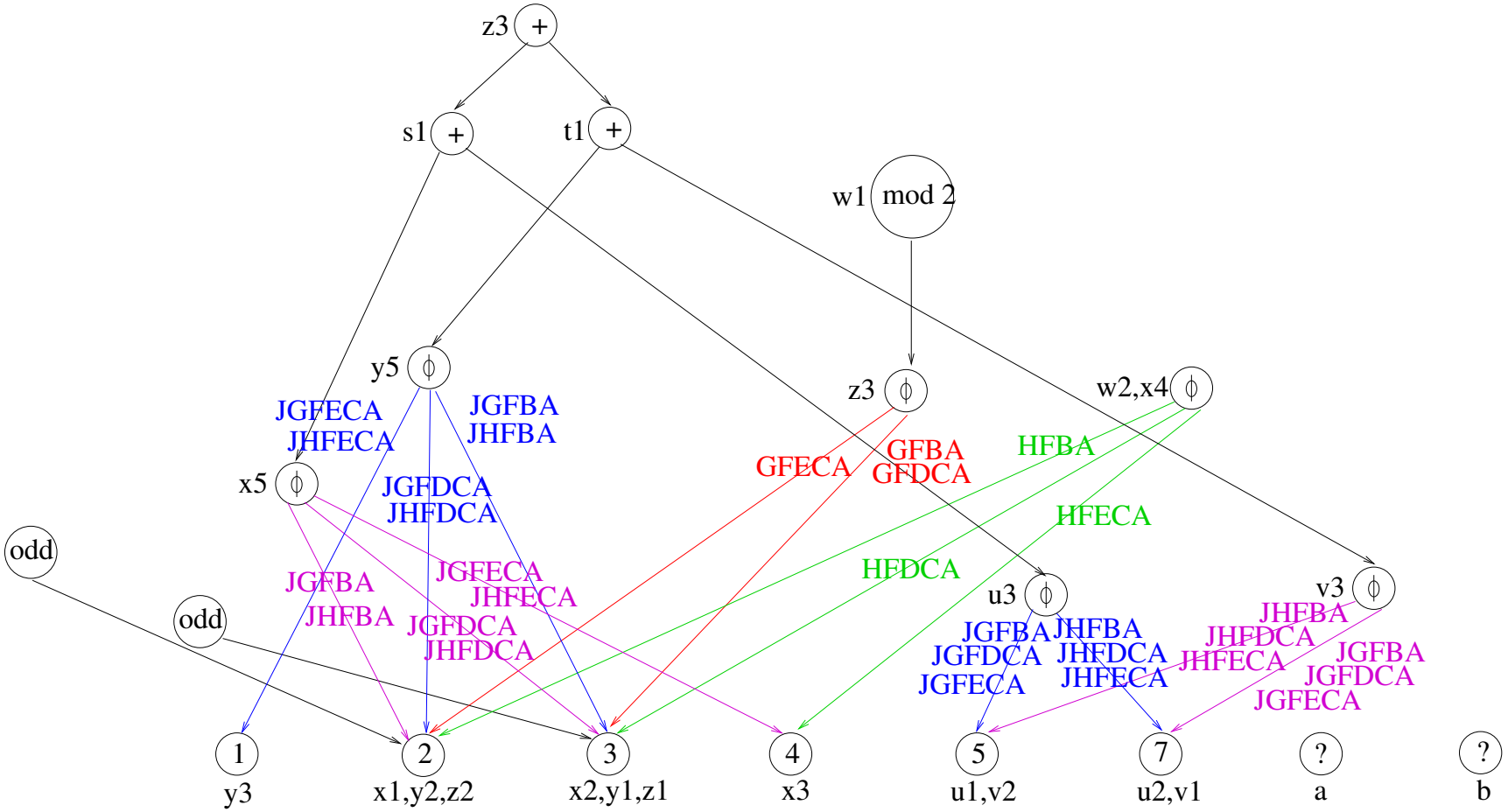
Korrektheit

- Der globale Konstantenfaltungsalgorithmus ist korrekt (sowohl für die Grundform wie für die volle Variante der lokalen Analysestufe).

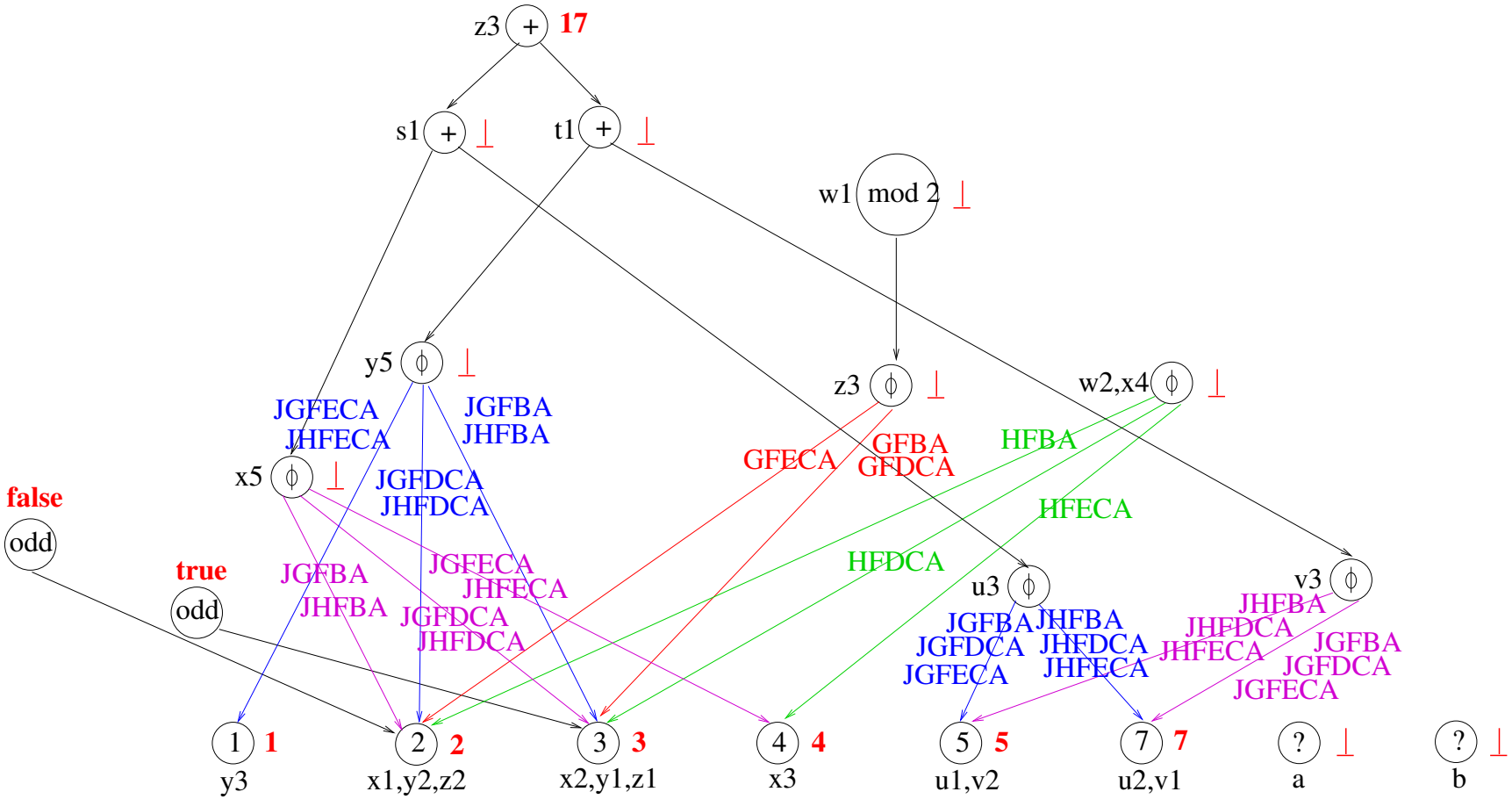
Vollständigkeit/Optimalität

- Der Grundalgorithmus der lokalen Analysestufe ist pfad-präzise bezüglich nicht-deterministischer Interpretation von Verzweigungsbedingungen.
- Der volle Algorithmus der lokalen Analysestufe ist prädikatssensitiv pfad-präzise.

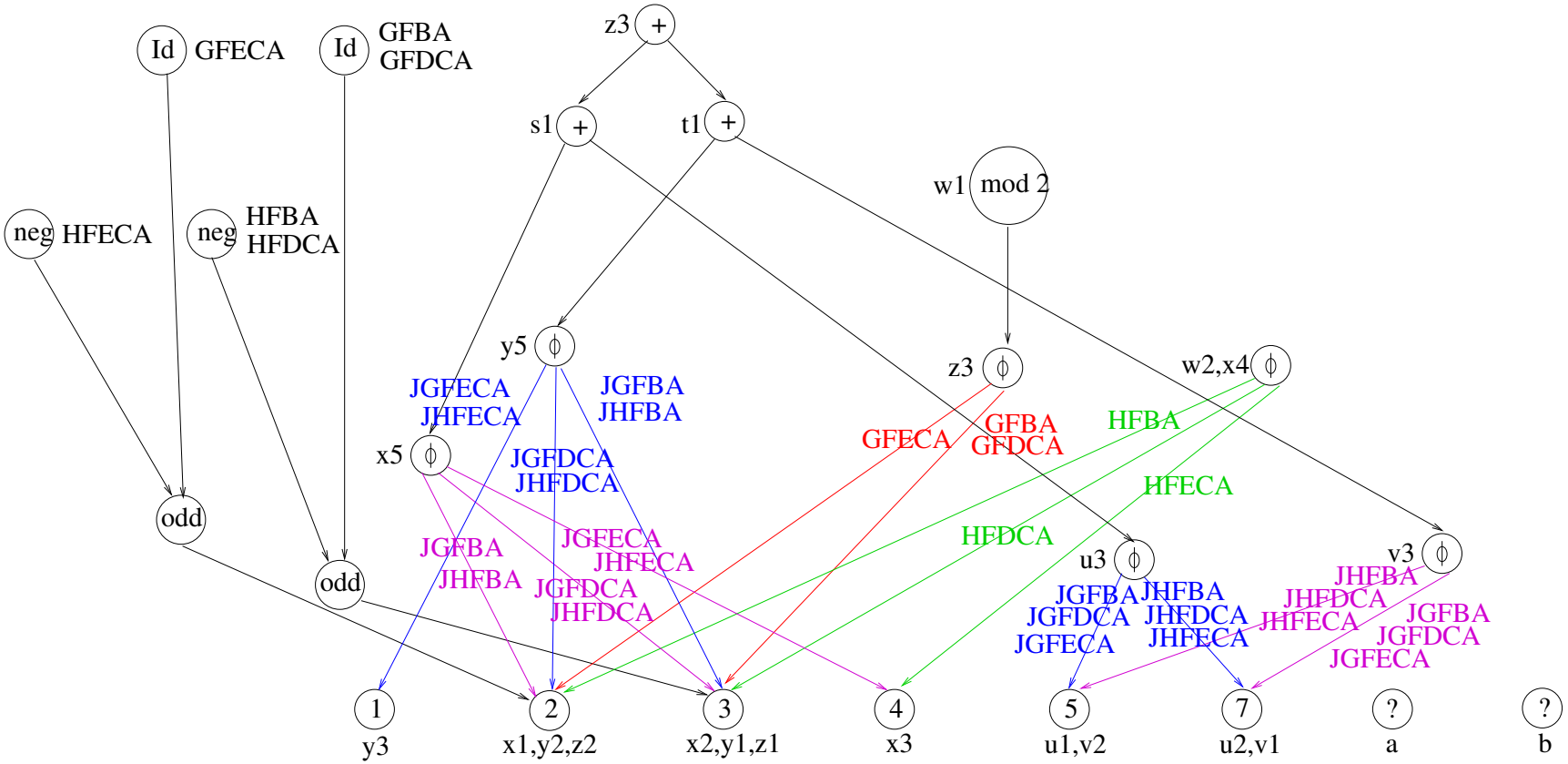
Performanzsteigerung: Grundalgorithmus (1)



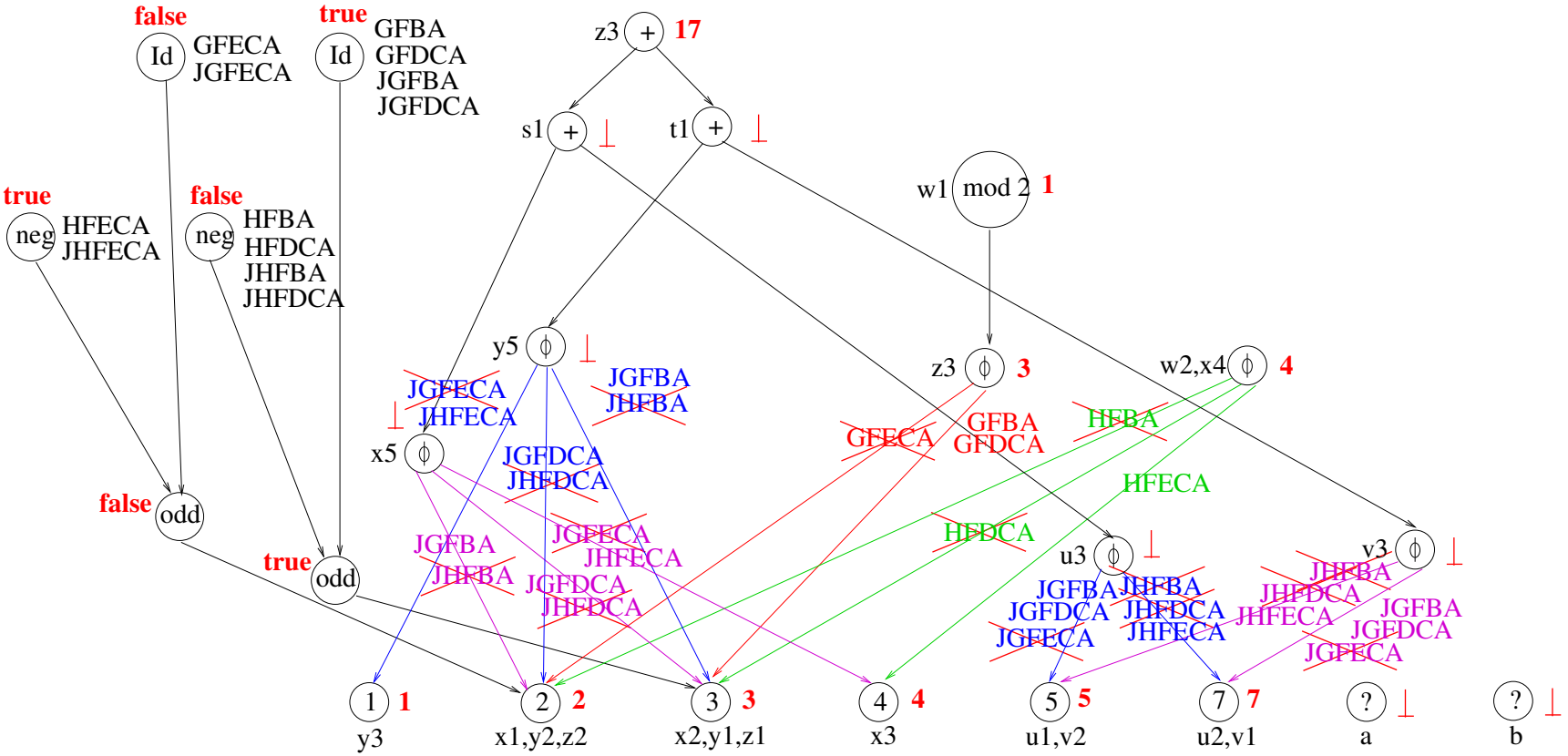
Performanzsteigerung: Grundalgorithmus (2)



Performanzsteigerung: Voller Algorithmus (1)



Performanzsteigerung: Voller Algorithmus (2)



Zusammenfassung und Schlussfolgerungen

Konstantenfaltung und SSA/PSSA-Form:

- sind perfekt aufeinander abgestimmt – SSA/PSSA sind wirklich von Hilfe für Analyse und Transformation!
- Der Schlüssel: Wertegraph und prädikatierter Wertegraph.

Offen für Erweiterungen, z.B.:

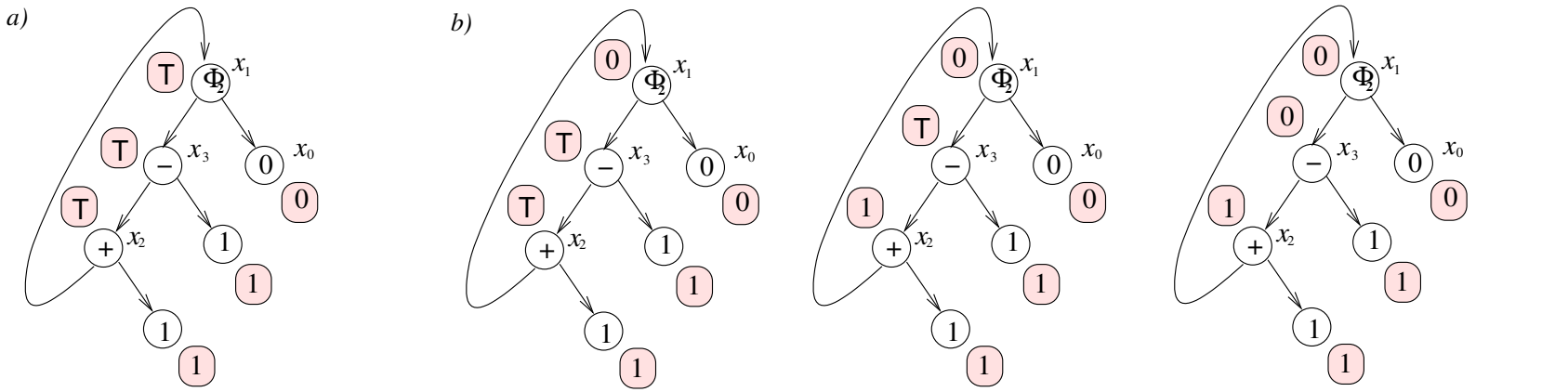
- Wertegraph: Bedingte Konstanten (engl. conditional constants).

Zusammenfassend:

- Konstantenfaltung ist besonders geeignet, Vorteile aus der Nutzung von (P)SSA zur Programmrepräsentation aufzuzeigen.

Konstantenfaltung auf SSA-basiertem 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!

Literaturhinweis

...zum PVG-Konstantenfaltungsalgorithmus:

- Jens Knoop, Oliver Rüthing. Constant Propagation on Predicated Code. Journal of Universal Computer Science 9(8):829-850, 2003. (Sonderausgabe zur SBLP'03).

Kapitel 14.4

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (1)

Bowen Alpern, Mark N. Wegman, F. Ken 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. Ken 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. Ken 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).


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.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (4)


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.


Bernhard Steffen, Jens Knoop. *Finite Constants: Characterizations of a New Decidable Set of Constants*. Theoretical Computer Science 80(2):303-318, 1991.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (5)


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.


Mark N. Wegman, F. Ken Zadeck. *Constant Propagation with Conditional Branches*. ACM Transactions on Programming Languages and Systems 13(2):181-201, 1991.

Teil IV

Abstrakte Interpretation und Modellprüfung

Kapitel 15

Abstrakte Interpretation und DFA

Motivation

Die Theorie abstrakter Interpretation — ein "mondäner" Programmanalyseansatz.

Intuitiv:

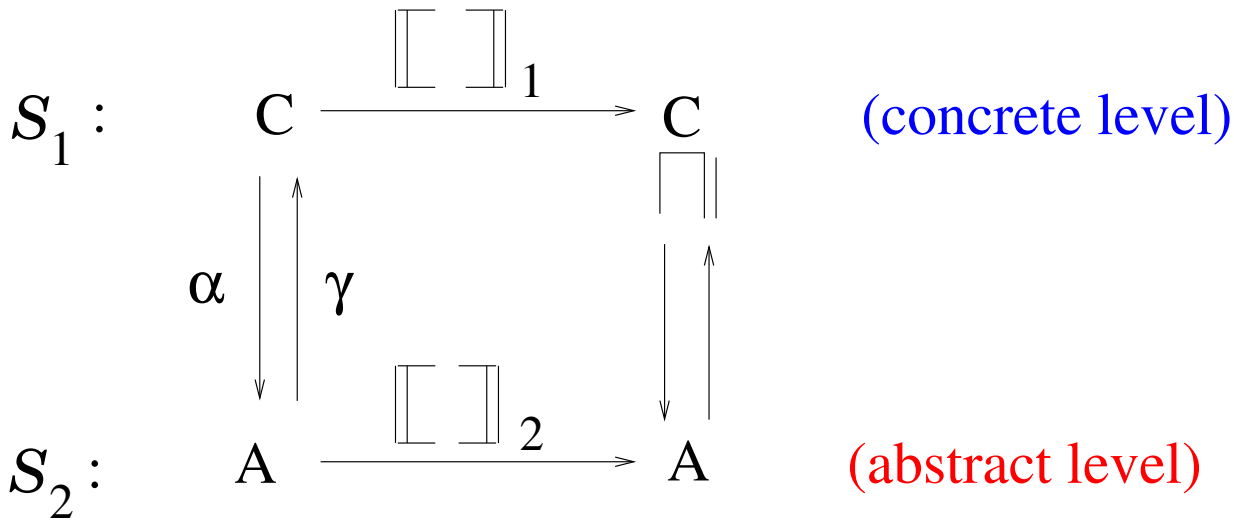
- Der DFA-Ansatz beinhaltet die Spezifikation einer Programmanalyse, deren Korrektheit separat und unabhängig *a posteriori* validiert wird.
- Der abstrakte Interpretationsansatz beinhaltet den Korrektheitsnachweis von Anfang an als integralen Bestandteil der Programmanalyse.

Kapitel 15.1

Theorie abstrakter Interpretation

Theorie abstrakter Interpretation

...ein Ansatz mit 2 Beobachtungsniveaus: Konkret&abstrakt



Bezeichnungen:

- Abstraktionsfunktion $\alpha : \mathcal{C} \to \mathcal{A}$
- Konkretisierungsfunktion $\gamma : \mathcal{A} \to \mathcal{C}$

Wohlzusammenhangsforderung:

- Abstraktions- und Konkretisierungsfunktion müssen eine Galois-Verbindung bilden.

Galois-Verbindungen

Definition 15.1.1 (Galois-Verbindung)

Seien $\mathcal{C} = (C, \sqcup_C, \sqsubseteq_C, \bot_C, \top_C)$ und $\mathcal{A} = (A, \sqcup_A, \sqsubseteq_A, \bot_A, \top_A)$ zwei vollständige Halbverbände und $\alpha : \mathcal{C} \to \mathcal{A}$ und $\gamma : \mathcal{A} \to \mathcal{C}$ eine Abstraktions- und Konkretisierungsfunktion. Dann definieren wir:

Das Viertupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ heißt Galois-Verbindung zwischen $\mathcal{C}$ und $\mathcal{A}$ gdw α und γ sind monoton und erfüllen:

1. $\gamma \circ \alpha \sqsupseteq_C Id_C$
2. $\alpha \circ \gamma \sqsubseteq_A Id_A$

mit Id_C und Id_A Identität auf C und A, d.h. $Id_C(c) = \lambda\, c.\ c$ und $Id_A(a) = \lambda\, a.\ a$ für alle $c \in C$ und $a \in A$.

Eigenschaften von Galois-Verbindungen (1)

Lemma 15.1.2 (Eindeutigkeit von Galois-Verbindungsbeziehungen)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung. Dann gilt:

1. α bestimmt γ eindeutig durch $\gamma(a) = \bigsqcup\{c \mid \alpha(c) \sqsubseteq_A a\}$.
2. γ bestimmt α eindeutig durch $\alpha(c) = \bigsqcap\{a \mid c \sqsubseteq_C \gamma(a)\}$.
3. α ist additiv und γ distributiv.

   Insbesondere gilt: $\alpha(\bot_C) = \bot_A$ und $\gamma(\top_A) = \top_C$.

Eigenschaften von Galois-Verbindungen (2)

Lemma 15.1.3 (Existenz und Eindeutigkeit von Galois-Verbindungsergänzungen)

1. Ist $\alpha : \mathcal{C} \to \mathcal{A}$ additiv, dann gibt es ein $\gamma : \mathcal{A} \to \mathcal{C}$, so dass $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung ist.
2. Ist $\gamma : \mathcal{A} \to \mathcal{C}$ distributiv, dann gibt es ein $\alpha : \mathcal{C} \to \mathcal{A}$, so dass $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung ist.

Lemma 15.1.4 (Keine Informationsverluste und -gewinne)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung. Dann gilt:

1. $\alpha \circ \gamma \circ \alpha = \alpha$
2. $\gamma \circ \alpha \circ \gamma = \gamma$

Galois-Einpassungen

Definition 15.1.5 (Galois-Einpassung)

Seien $\mathcal{C} = (C, \sqcup_C, \sqsubseteq_C, \bot_C, \top_C)$ und $\mathcal{A} = (A, \sqcup_A, \sqsubseteq_A, \bot_A, \top_A)$ zwei vollständige Halbverbände und $\alpha : \mathcal{C} \to \mathcal{A}$ und $\gamma : \mathcal{A} \to \mathcal{C}$ eine Abstraktions- und Konkretisierungsfunktion. Dann definieren wir:

Das Viertupel $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ heißt Galois-Einpassung zwischen $\mathcal{C}$ und $\mathcal{A}$ gdw α und γ sind monoton und erfüllen:

1. $\gamma \circ \alpha \sqsupseteq_C Id_C$
2. $\alpha \circ \gamma = Id_A$

mit Id_C und Id_A Identität auf C und A, d.h. $Id_C(c) = \lambda\, c.\ c$ und $Id_A(a) = \lambda\, a.\ a$ für alle $c \in C$ und $a \in A$.

Interpretation

Ist $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Einpassung zwischen $\mathcal{C}$ und $\mathcal{A}$, so führt eine vorangehende Konkretisierung vor einer Abstraktion

- nicht zu einem Informationsverlust: $\alpha \circ \gamma = Id_A$

Eigenschaften von Galois-Einpassungen

Lemma 15.1.6 (Äquivalenzaussagen)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung. Dann sind die folgenden Aussagen äquivalent:

1. $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Einpassung.
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 ordnungs-ähnlich, d.h.
   $\forall\, a_1, a_2 \in A.\ \gamma(a_1) \sqsubseteq_C \gamma(a_2) \Longleftrightarrow a_1 \sqsubseteq_A a_2$.

Konstruktion von Galois-Einpassungen

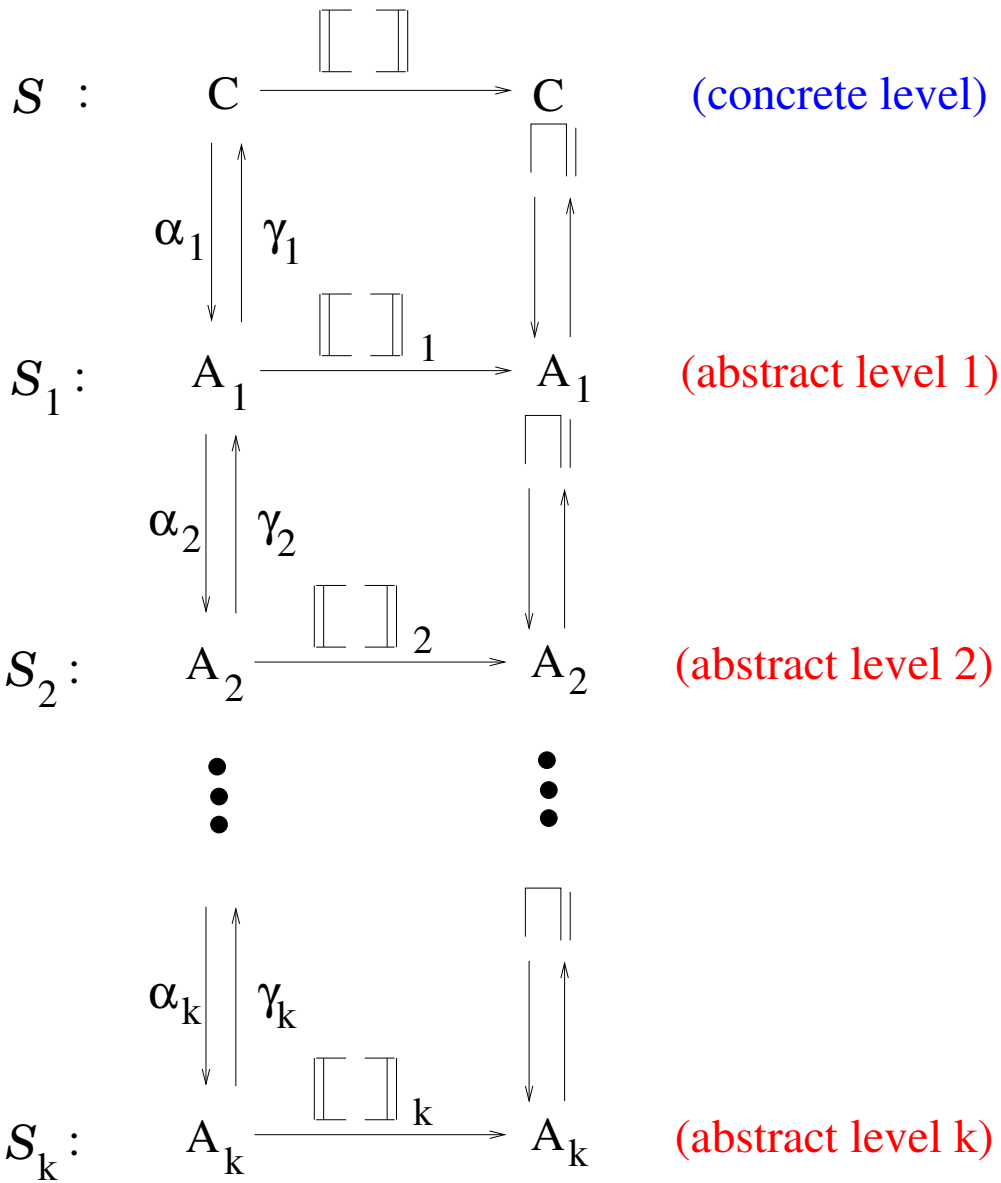
Lemma 15.1.7 (Galois-Einpassungskonstruktion)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung und $\rho : A \to A$ der wie folgt definierte Reduktionsoperator:

$$\forall\, a \in A.\ \rho(a) = \bigsqcap \{a' \mid \gamma(a) = \gamma(a')\}$$

Dann ist $\rho(A) = (\{\rho(a) \mid a \in A\}, \sqcup_A, \sqsubseteq_A, \bot_A, \top_A)\}$ ein vollständiger Halbverband und das 4-Tupel $(\mathcal{C}, \alpha, \gamma, \rho(A))$ eine Galois-Einpassung.

Hierarchie von Galois-Verb./-Einpassungen



Kapitel 15.2

Systematische Konstruktion abstrakter Interpretationen

Übersicht

Im einzelnen betrachten wir 6 Methoden:

- Unabhängige Attributemethode
- Relationale Methode
- Totale Funktionenraummethode
- Monotone Funktionenraummethode
- Direkte Produktmethode
- Direkte Tensorproduktmethode

Die unabhängige Attributemethode

Lemma 15.2.1 (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

$$\begin{aligned} \alpha(c_1, c_2) &= (\alpha_1(c_1), \alpha_2(c_2)) \\ \gamma(a_1, a_2) &= (\gamma_1(a_1), \gamma_2(a_2)) \end{aligned}$$

eine Galois-Verbindung.

Die relationale Methode

Lemma 15.2.2 (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, wobei $\mathcal{P}$ den Potenzmengenoperator bezeichnet.

Dann ist auch $(\mathcal{P}(C_1 \times C_2), \alpha, \gamma, \mathcal{P}(A_1 \times A_2))$ mit

$$\begin{aligned} \alpha(CC) &= \bigcup \{\alpha_1(\{c_1\}) \times \alpha_2(\{c_2\}) \,|\, (c_1, c_2) \in CC\} \\ \gamma(AA) &= \{(c_1, c_2) \,|\, \alpha_1(\{c_1\}) \times \alpha_2(\{c_2\}) \subseteq AA\} \end{aligned}$$

für $CC \subseteq C_1 \times C_2$ und $AA \subseteq A_1 \times A_2$ eine Galois-Verbindung.

Die totale Funktionenraummethode

Lemma 15.2.3 (Totale Funktionenraummethode)

Sei $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$ eine Galois-Verbindung und M eine Menge.

Dann ist auch $([M \to C], \alpha', \gamma', [M \to A])$ mit

$$\begin{aligned} \alpha'(f) &= \alpha \circ f \\ \gamma'(g) &= \gamma \circ g \end{aligned}$$

eine Galois-Verbindung.

Die monotone Funktionraummethode

Lemma 15.2.4 (Monotone Funktionenraum-methode)

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 $([C_1 \to C_2], \alpha, \gamma, [A_1 \to A_2])$ mit

$$\begin{aligned} \alpha(f) &= \alpha_2 \circ f \circ \gamma_1 \\ \gamma(g) &= \gamma_2 \circ g \circ \alpha_1 \end{aligned}$$

eine Galois-Verbindung.

Die direkte Produktmethode

Lemma 15.2.5 (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

$$\begin{aligned} \alpha(c) &= (\alpha_1(c), \alpha_2(c)) \\ \gamma(a_1, a_2) &= \gamma_1(a_1) \sqcap \gamma_2(a_2) \end{aligned}$$

eine Galois-Verbindung.

Die direkte Tensorproduktmethode

Lemma 15.2.6 (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, wobei $\mathcal{P}$ den Potenzmengenoperator bezeichnet.

Dann ist auch $(\mathcal{P}(C), \alpha, \gamma, \mathcal{P}(A_1 \times \mathcal{A}_2))$ mit

$$\begin{aligned} \alpha(C') &= \bigcup \{\alpha_1(\{c\}) \times \alpha_2(\{c\}) \,|\, c \in C'\} \\ \gamma(AA) &= \{c \,|\, \alpha_1(\{c\}) \times \alpha_2(\{c\}) \subseteq AA\} \end{aligned}$$

für $C' \subseteq C$ und $AA \subseteq A_1 \times A_2$ eine Galois-Verbindung.

Kapitel 15.3

Korrektheit abstrakter Interpretationen

Korrektheit

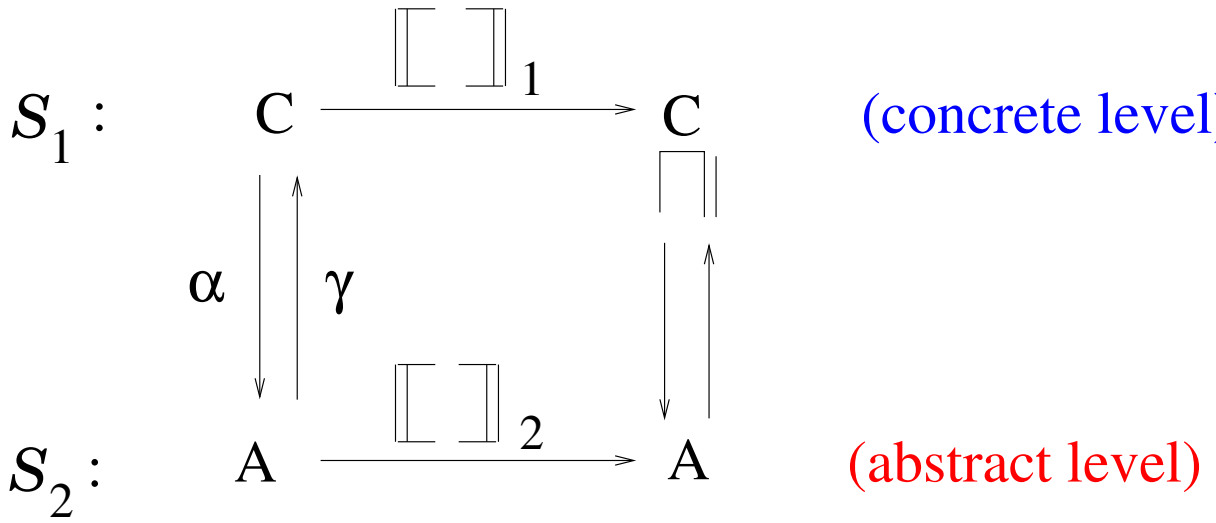
Der in seiner Grund- bzw. erweiterten Form auf

- 2 bzw. einer höheren Zahl von Beobachtungsniveaus

beruhende abstrakte Interpretationsansatz erlaubt die

- Korrektheit einer abstrakten Interpretation auf einem niedrigeren Beobachtungsniveau relativ zu einer anderen abstrakten Interpretation auf einem höheren Beobachtungsniveau

formal zu fassen und nachzuweisen.



Vollständigkeit und Optimalität (1)

In diesem Sinne ist der auf Patrick und Radhia Cousot (POPL'77) zurückgehende Ansatz abstrakter Interpretation in seiner "klassischen" Ausprägung

- semantik-orientiert (unter Bezug auf eine Standardsemantik (besonders wichtig hierbei: Die sog. Aufsammelsemantik (engl. collecting semantics)).

Der semantik-orientierte Ansatz unterstützt nicht *per se* einen

- Vollständigkeits- und Optimalitätsbegriff

und dessen formalen Nachweis.

Vollständigkeit und Optimalität (2)

Zur Betrachtung von Vollständigkeit und Optimalität gehen wir in der Folge von der semantik-orientierten zu einer auf Bernhard Steffen (TAPSOFT'87, MFCS'89) zurückgehenden

- beobachtungs-orientierten Perspektive (ohne Bezug auf eine Standardsemantik)

über.

Kapitel 15.4

Vollständigkeit und Optimalität

Vorbereitungen

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 15.4.1 (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 $([\![\ ]\!]_l, \mathcal{C})$ lokale abstrakte Semantik bzw. lokale abstrakte Interpretation.

Globale abstrakte Semantik

Definition 15.4.2 (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 15.4.3 (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 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 Konkretisierungsfunktion 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 reversem Datenflussanalyse-funktional).
- 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 15.4.4 (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:

$$A \circ \llbracket\ G\ \rrbracket_1 = \llbracket\ G\ \rrbracket_2 \circ A$$

Beobachtungsniveau und Beobachtung

Definition 15.4.5 (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 15.4.6 (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.

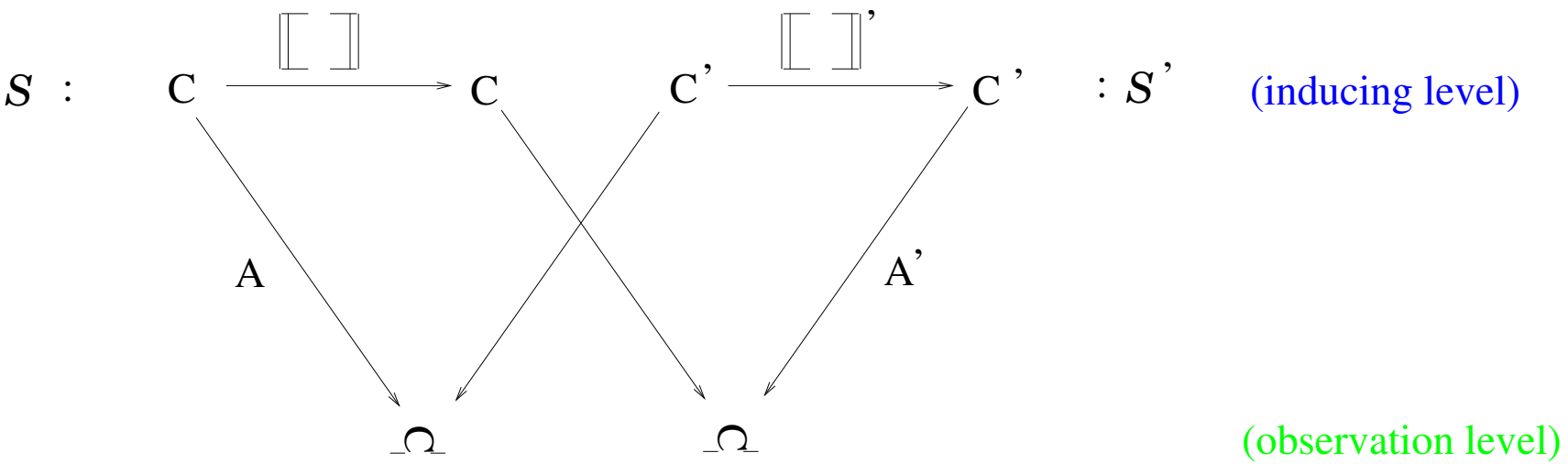
Beobachtungsäquivalenz

Definition 15.4.7 (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:



Eigenschaften der Relation $\approx_\Omega$

Lemma 15.4.8 (Äquivalenzrelation)

Die Relation $\approx_\Omega$ ist eine Äquivalenzrelation auf der Menge aller Modelle von Ω.

Lokale Optimalität (1)

Definition 15.4.9 (Lokale Optimalität)

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 (2)

Definiton 15.4.9 (Lokale Optimalität (fgs.))

4. $\mathcal{S}_2$ heißt 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 15.4.10 (Voll abstrakte Modelle)

Seien $\mathcal{S}_1$, Ω und A mit $\mathcal{S}_1 \rightarrow_A \Omega$.

Ein Paar $(\mathcal{S}_2, A_2)$ mit $\mathcal{S}_2 \rightarrow_A \Omega$ 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 der Beziehung $\mathcal{S}_1 \rightarrow_A \Omega$ stehen, mit $\Phi(\mathcal{S}_1, A, \Omega)$.

Existenz und Eindeutigkeit

Theorem 15.4.11 (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 15.4.12 (Abschneidetheorem)

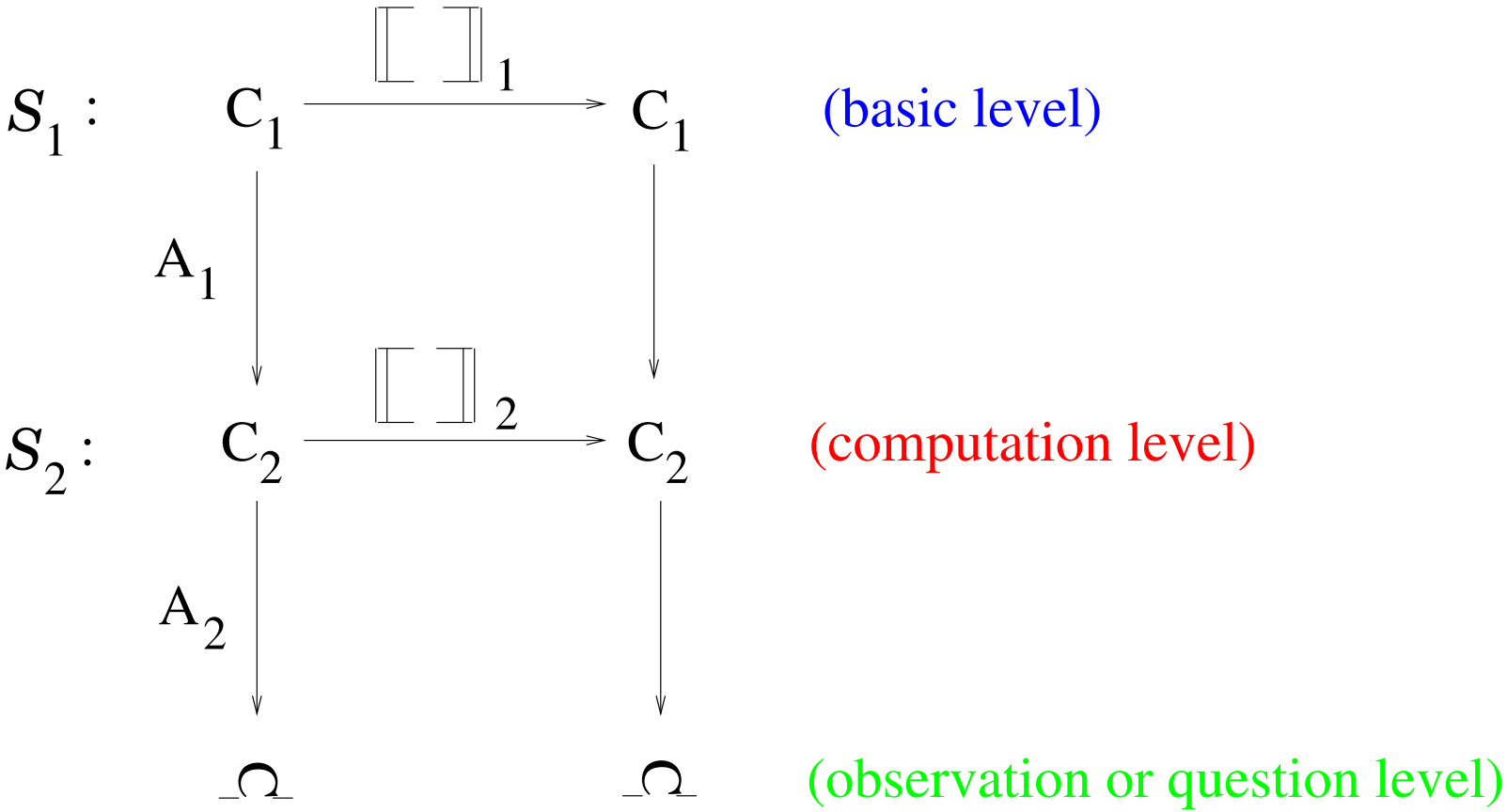
Sei $(\mathcal{S}_2, A_2) \in \Phi(\mathcal{S}_1, A_2 \circ A_1, \Omega)$ mit $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$. Dann gilt:

$$\forall\, G \in \mathbf{FG}.\ A_1 \circ [\![\, G \,]\!]_1 \circ A_1^a = [\![\, G \,]\!]_2$$

Insbesondere gilt weiters:

$$(\mathcal{S}_1, A_2 \circ A_1) \approx_\Omega (\mathcal{S}_2, A_2)$$

Veranschaulichung



Äquivalenz

Theorem 15.4.13 (Äquivalenz)

Seien $(\mathcal{S}, A)$ und $(\mathcal{S}', A')$ zwei Modelle von Ω. Dann gilt:

$$(\mathcal{S}, A) \approx_\Omega (\mathcal{S}', A') \Longleftrightarrow \Phi(\mathcal{S}, A, \Omega) = \Phi(\mathcal{S}', A', \Omega)$$

Interpretation:

- Zusammen mit dem Existenz- und Eindeutigkeitstheorem 15.3.11 und dem Abschneidetheorem 15.3.12 liefert das Äquivalenztheorem 15.3.13, dass voll abstrakte Modelle (bis auf Isomorphie) die "abstraktesten" Repräsentanten ihrer Beobachtungsäquivalenzklasse sind.

Interpretation und Quintessenz (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 korrekt und zugleich das Gesuchte

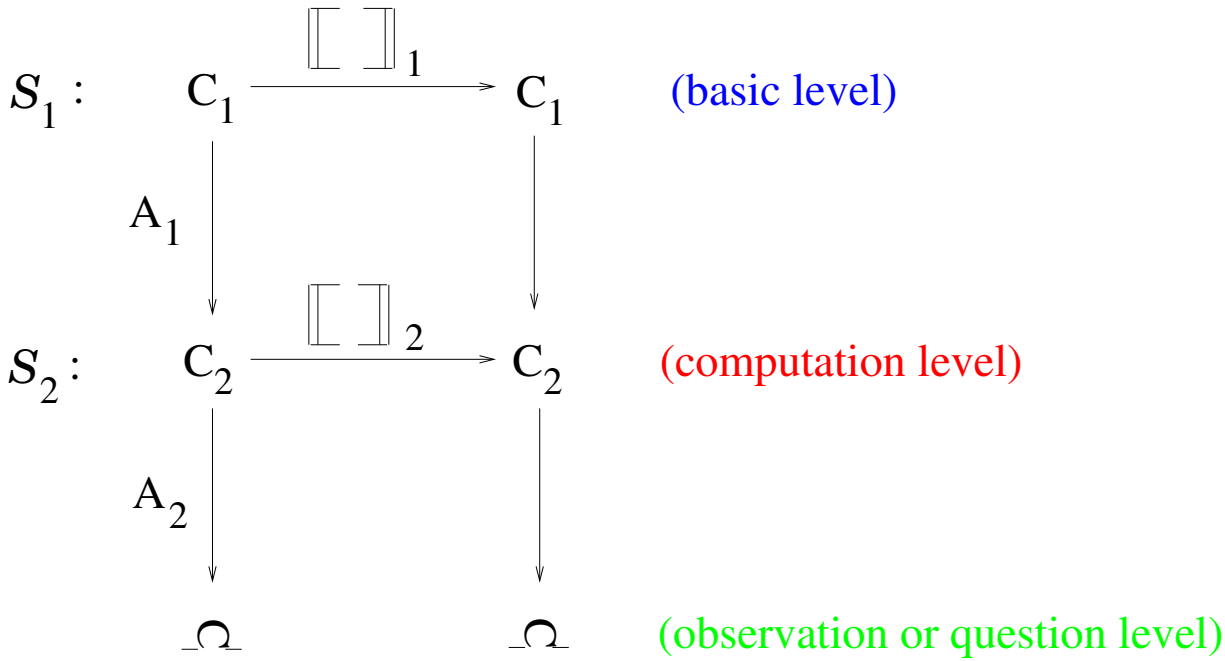
- vollständige bzw. optimale Modell.

Interpretation und Quintessenz (2)

Dieses voll abstrakte Modell liegt hierarchisch eingebettet innerhalb des

- 3-Niveaumodells.

In diesem Sinn ist das 3-Niveaumodell hinreichend allgemein für den nicht auf Hierarchien abstrakter Interpretationen beschränkten Begriff der Beobachtungsäquivalenz.



Kapitel 15.5

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 (Eds). Prentice Hall, 63-102, 1987.
- Patrick Cousot. *Methods and Logics for Proving Programs*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Ed.), 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*. Autom. Softw. Eng. 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 Sience 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 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 DFA

Kapitel 16.1

xxx

Motivation

Das Grundproblem der Modellprüfung.

Gegeben:

- Ein Modell $\mathcal{M}$
- Eine Eigenschaft $\mathcal{E}$ ausgedrückt als eine Formel ϕ

Modellprüfungsfrage:

- Besitzt Modell $\mathcal{M}$ Eigenschaft $\mathcal{E}$, erfüllt $\mathcal{M}$ Formel ϕ ?

$$\mathcal{M} \models \phi$$

Modellprüfer und Modellprüfung

Modellprüfer:

- Eine Methode, ein Werkzeug, das die Modellprüfungsfrage beantwortet.

Modellprüfung:

- Ansetzen eines Modellprüfers MP auf ein Paar 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.

    Zweckmäßig: Ausgabe eines (minimalen) Gegenbeispiels, das die Verletzung der Formel zeigt (CEGAR (counter-example-guided abstraction refinement)-Ansatz).
  - $\mathcal{M} \models_{MP} \phi$ wird weder nachgewiesen noch widerlegt: Modellprüfer ist unvollständig für Modell- und Formelsprache.

Modellsprachen

Modellsprachen:

- Transitionssysteme, Kripke-Strukturen

und spezielle Ausprägungen:

- Automaten
- Flussgraphen
- (Programm-) Zustandsgraphen
- ...

Modelle können sein

- endlich: Endliche Modellprüfung
- unendlich: Unendliche Modellprüfung

Notorisches (und schwierig handhabbares) Problem:

- Explosion des Zustandsraums: Zustandsraumexplosion

Formelsprachen

Formelsprachen:

- Temporale, modale Logiken (linear time logics, branching time logics)
  - LTL, CTL, CTL*
  - μ-Kalkül
  - ...

Notorisches (und schwierig handhabbares) Problem:

- Balancierung von Ausdruckskraft und Entscheidbarkeit, insbesondere effizienter Entscheidbarkeit.

Modaler μ-Kalkül (1)

...um Rückwärtsmodalitäten erweiteter μ-Kalkül:

Syntax:

$$\Phi \ ::= \ tt \mid X \mid \Phi \wedge \Phi \mid \neg\Phi \mid \beta \mid [\alpha]\Phi \mid \overline{[\alpha]}\Phi \mid \nu X.\,\Phi$$

Semantik:

$$\begin{aligned}
[\![\, tt \,]\!]e &= \mathcal{S} \\
[\![\, X \,]\!]e &= e(X) \\
[\![\, \Phi_1 \wedge \Phi_2 \,]\!]e &= [\![\, \Phi_1 \,]\!]e \wedge [\![\, \Phi_2 \,]\!]e \\
[\![\, \neg\Phi \,]\!]e &= \mathcal{S} \backslash [\![\, \Phi \,]\!]e \\
[\![\, \beta \,]\!]e &= \{p \in \mathcal{S} \mid \beta \in \lambda(p)\} \\
[\![\, [\alpha]\Phi \,]\!]e &= \{p \in \mathcal{S} \mid \forall\, q \in \mathit{Succ}_\alpha.\ q \in [\![\, \Phi \,]\!]e\} \\
[\![\, \overline{[\alpha]}\Phi \,]\!]e &= \{p \in \mathcal{S} \mid \forall\, p \in \mathit{Pred}_\alpha.\ p \in [\![\, \Phi \,]\!]e\} \\
[\![\, \nu X.\,\Phi \,]\!]e &= \bigcup\{S' \subseteq \mathcal{S} \mid S' \subseteq [\![\, \Phi \,]\!]e[S'/X]\}
\end{aligned}$$

Modaler μ-Kalkül (2)

Abgeleitete Operatoren:

$$
\begin{aligned}
\mathit{ff} &= \neg \mathit{tt} \\
\Phi_1 \vee \Phi_2 &= \neg(\neg\Phi_1 \wedge \neg\Phi_2) \\
\langle\alpha\rangle\Phi &= \neg[\alpha](\neg\Phi) \\
\overline{\langle\alpha\rangle}\Phi &= \neg\overline{[\alpha]}(\neg\Phi) \\
\mu X.\,\Phi &= \nu X.\,\neg(\Phi[\neg X/X]) \\
\Phi \succ \Psi &= \neg\Phi \vee \Psi
\end{aligned}
$$

Modaler μ-Kalkül (3)

Höher-abstrakte abgeleitete Operatoren:

$$\begin{aligned} \mathbf{AG}\ \Phi &= \nu X.(\Phi \wedge [.]X) \\ \Phi\ \mathbf{U}\ \Psi &= \nu X.(\Psi \vee (\Phi \wedge [.]X) \end{aligned}$$

$$\begin{aligned} \overline{\mathbf{AG}}\ \Phi &= \nu X.(\Phi \wedge \overline{[.]}X) \\ \Phi\ \overline{\mathbf{U}}\ \Psi &= \nu X.(\Psi \vee (\Phi \wedge \overline{[.]}X) \end{aligned}$$

Analogie DFA – Modellprüfung

Datenflussanalyse:

DFA-Algorithmus für Eigenschaft E :

Programme $\rightarrow$ Menge der E erfüllenden Programmpunkte

Modellprüfung:

Modellprüfer : (Modallogische) Formel $\times$ Modell

$\rightarrow$ Menge der die Formel erfüllenden Zustände

Intuitiv:

- Ein DFA-Algorithmus für E ist ein bezüglich E partiell ausgewerteter oder bezüglich E spezialisierter Modellprüfer!

Anwendung: PREE

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:

$$OCP =_{df} EAR \wedge NEC$$

Theorem 16.1.1 (Korrektheit und Optimalität)

Das Ersetzen der originalen Berechnungen durch neue an den durch OCP gegebenen Programmpunkten ist korrekt (d.h. semantikerhaltend) und optimal (d.h. mindestens so gut wie jede andere korrekte Platzierung der Berechnungen).

Zusammenfassung (1)

Diese 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 der 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 16.2

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 Deduction, A. Robinson, A. 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. *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, 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 AI

Kapitel 17.1

xxx

Zustandsexplosionsproblem

Zentrale Herausforderung von Modellprüfung in der Praxis:

- Das sog. Zustandsexplosionsproblem

Im Kern:

- Die Zahl der Zustände im Zustandsraum wächst exponentiell in der Zahl der parallelen/nebenläufigen Komponenten

Verschiedene Anätze zur Behandlung des Zustandsexplosionsproblem

- 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 Ordnungstechniken, z.B. Godefroid, 1996; Peled, 1993; Valmari, 1992.
- Kompositionelle Techniken, Clarke et al., 1989; Santone, 2002.
- Abstraktionstechniken, Barbuti et al., 1999; Clarke et al., 1994.

Kapitel 17.2

Literaturverzeichnis, Leseempfehlungen

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (1)


R. Barbuti, N.D. Francesco, Antonella Santone, G. Vaglini. *Selective Mu-Calculus and Formula-based Equivalence of Transition Systems*. J. Comput. Syst. Sci. 59(3):537-556, 1999.


A. Bouajjani, J.-C. Fernandez, N. Halbwachs. *Minimal Model Generation*. In Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV'90), Springer-V., LNCS 531, 197-203, 1990.


Edmund M. Clarke, Orna Grumberg, D.E. Long. *Compositional Model Checking*. In Proceedings LICS'89, IEEE Computer Society, 353-362, 1989.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (2)


Edmund M. Clarke, Orna Grumberg, D.E. Long. *Model Checking and Abstraction*. ACM Transactions on Programming Languages and Systems 16(5):1512-1542, 1994.


P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*. Springer-V., LNCS 1032, 1996.


Susanne Graf, Bernhard Steffen, Gerald Lüttgen. *Compositional Minimization of Finite State Systems using Interface Specifications*. Formal Aspects of Computing 8(5):607-616, 1996.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (3)


C. Jard, T. 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.


K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.


Doron Peled. *All from One, One for All: On Model Checking Using Representatives*. In Proceedings of the 5th International Workshop on Computer Aided Verification (CAV'93), Springer-V., LNCS 697, 409-423, 1993.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (4)


Antonella Santone. *Automatic Verification of Concurrent Systems Using a Formula-based Compositional Approach*. Acta Informatica 38(8), 531-564, 2002.


Colin Stirling, D. Walker. *Local Model Checking in the Modal Mu-Calculus*. Theoretical Computer Science 89(1):161-177, 1991.


A. Valmari. *A Stubborn Attack on State Explosion*. Formal Methods in System Design 1(4):297-322, 1992.

Teil V

Abschluss und Ausblick

Kapitel 18

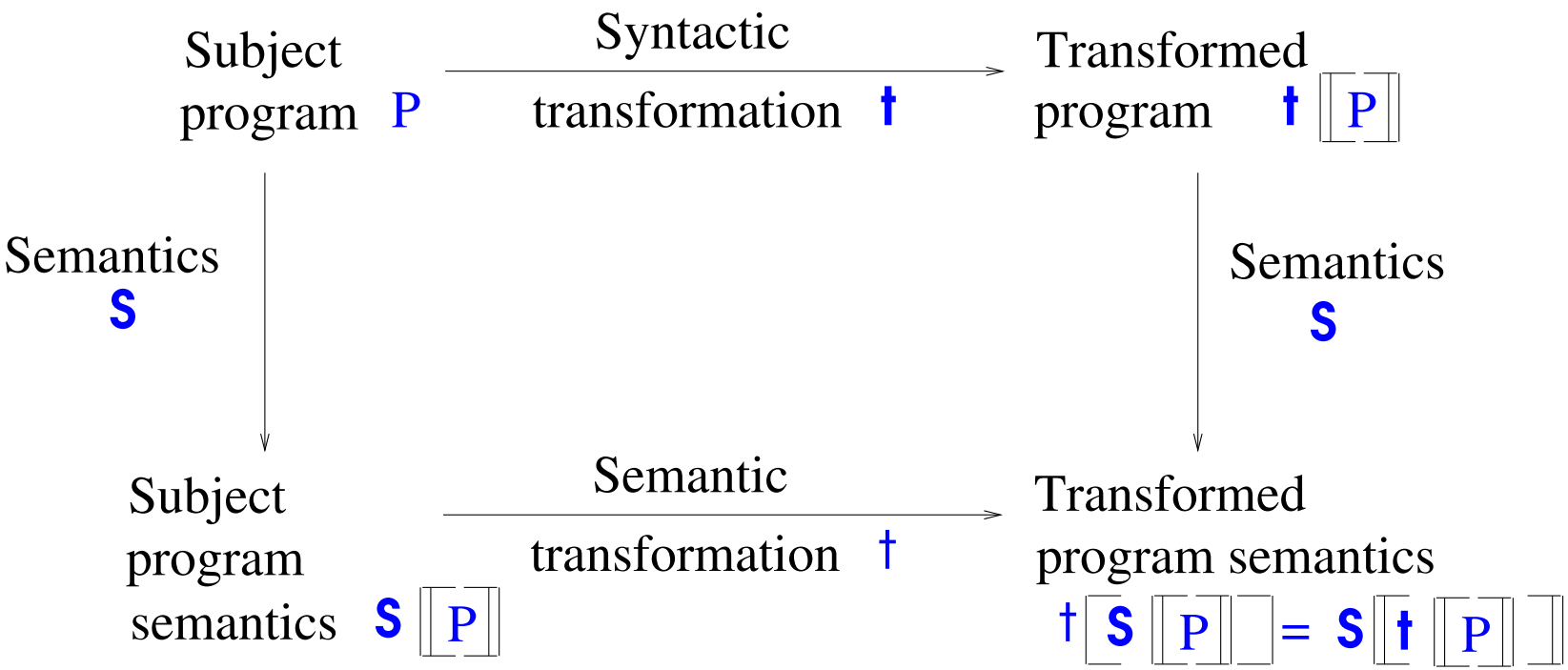
Resümee und Perspektiven

Kapitel 18.1

xxx

Analyse, Verifikation und Transformation

...bewiesen (beweisbar) korrekt und vollständig/optimal in einem einheitlichen Rahmen:



(aus Cousot&Cousot, POPL 2002)

Wichtig für verschiedenste Gebiete, darunter:

- Optimierende Übersetzer (Performanz, Energie,...)
- Software-Verifikation
- Übersetzer-Verifikation, Betriebssystem-Verifikation,...
- Software
  - -Spezifikation, -Analyse, -Validierung, -Generierung (insbesondere auch modellgetriebene Spezifikation, Analyse, Verifikation, Generierung, Testung,...)
  - -Verstehen
    - Refaktorisierung
    - (Re-)engineering, Reverse Engineering
    - Dokumentation
  - -(Kunden)anpassung, -spezialisierung (customization)
- Safety and Security (security policy enforcement,...)
- Datenschürfung und -ausbeutung (data mining)
- Hardware-Verifikation
- ...
- Grüne Informationstechnologie

All dies

- Statisch und dynamisch.
- Auf Programm-Ebene im Kleinen.
- Auf System-Ebene im Großen
  - Systeme von Systemen
  - Hard- und Software-Systeme von Systemen
    - Eingebettete Systeme
    - Cyberphysikalische Systeme
    - Verteilte Systeme: Service-orientierte Systeme, Wolken-Systeme, Mehrkern-(HW)-Systeme,...
    - Echtzeit-Systeme
    - ...
- auf Spezifikations-, Modellierungs-, Programmier-, Zwischensprach- und Binärcode-Ebene.

Unverzichtbar

Rigorose Fundierung

- Formale Methoden

Wirksame Werkzeug-Unterstützung

- Hochskalierende "Denk"-Werkzeuge
  - Vollautomatisch
    - Knopfdruckanalyse, -verifikation und -transformation
  - Halbautomatisch
    - Interaktive, benutzergeleitete Analyse, Verifikation und Transformation
- Orchestrierung und geordnetes Zusammenspiel und -wirkung über Methodengrenzen hinweg (z.B. Abstrakte Interpretation, Modellprüfung, Theorembeweisen,...)

Wichtige Konferenzen und Zeitschriften (1)

- Annual International Conference on Verification, Model-Checking, Abstract Interpretation (VMCAI) Series, Springer-V., LNCS series, since 2000.
- Annual International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Springer-V., LNCS series, since 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, since 2008.

Wichtige Konferenzen und Zeitschriften (2)

- Annual International Symposium on Formal Methods (FM) Series, Springer-V., LNCS series, since 1995.
- Biennial International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA) Series, Springer-V., LNCS series, since 2004.
- International Journal on Software Tools for Technology Transfer (STTT), Springer-V, since 1999.

Kapitel 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.


Cristian Cadar, Koushik Sen. *Symbolic Execution for Software Testing: Three Decades Later*. Communications of the ACM 56(2):82-90, 2013.


Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.

Vertiefende und weiterführende Leseempfehlungen für Kapitel 18 (3)


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.


Flemming Nielson. *Program Transformations in a Denotational Setting*. ACM Transactions on Programming Languages and Systems 7:359-379, 1985.


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 (4)

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 und Leseempfehlungen

...zum vertiefenden und weiterführenden Selbststudium.

- I Lehrbücher
- II Artikel, Dissertationen, Sammelbände
- III Webseiten

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.


A. Arnold, I. 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.


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.

I Lehrbücher (3)


B. A. Davey, H. A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge University Press, 1990.


Martin Davis. *Computability and Unsolvability*. Dover Publications, 1982.


Gilles Dowek. *Principles of Programming Languages*. Springer-V, 2009.


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


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


George E. Hughes, Max J. Cresswell. *An Introduction to Modal Logic*. Methuan, 1968.

I Lehrbücher (4)


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.


Fred Kröger, Stephan Merz. *Temporal Logic and State Systems*. Springer-V., 2008.


Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009.


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


Robert Lover. *Elementary Logic for Software Development*. Springer-V., 2008.

I Lehrbücher (5)

- 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.
- Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. 2. Auflage, Springer-V., 2005.
- Doron A. Peled. *Software Reliability Methods*. Springer-V., 2001.

I Lehrbücher (6)


Bernhard Steffen, Oliver Rüthing, Malte Isberner. *Grundlagen der höheren Informatik. Induktives Vorgehen*. Springer-V., 2014.


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


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 Artikel, Dissertationen, Sammelbände (1)


Samson Abramsky, Chris Hankin. *An Introduction to Abstract Interpretation*. In Abstract Interpretation of Declarative Languages, Samson Abramsky, Chris Hankin (Eds). 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.


Bowen Alpern, Mark N. Wegman, F. Ken 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.

II Artikel, Dissertationen, Sammelbände (2)


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.


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.

II Artikel, Dissertationen, Sammelbände (3)

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.

II Artikel, Dissertationen, Sammelbände (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.


Bernhard Beckert, Reiner Hähnle, Peter H. Schmitt (Hrsg.). *Verification of Object-Oriented Software: The KeY Approach*. LNCS 4334, Springer-V., 2007.

II Artikel, Dissertationen, Sammelbände (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.

II Artikel, Dissertationen, Sammelbände (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.

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.

II Artikel, Dissertationen, Sammelbände (7)

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'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.

Francesco Buccafurri, Thomas Eiter, Georg Gottlob, Nicola Leone. *Enhancing Model Checking in Verification by AI Techniques*. Artificial Intelligence 112(1-2):57-104, 1999.

II Artikel, Dissertationen, Sammelbände (8)


Cristian Cadar, Koushik Sen. *Symbolic Execution for Software Testing: Three Decades Later*. Communications of the ACM 56(2):82-90, 2013.


Edmund M. Clarke. *Programming Language Constructs for which it is Impossible to Obtain Good Hoare Axiom Systems*. Journal of the ACM 26(1):129-147, 1979.


Edmund M. Clarke. *The Birth of Model Checking*. In *25 Years of Model Checking*. Orna Grumberg, Helmut Veith (Hrsg.), Springer-V., LNCS 5000, 1-26, 2008.


Edmund M. Clarke, Stephen M. German, Joseph Y. Halpern. *Effective Axiomatizations of Hoare Logics*. Journal of the ACM 30(1):612-636, 1983.

II Artikel, Dissertationen, Sammelbände (9)


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, H. Schlingloff. *Model Checking*. In Handbook of Automated Deduction, A. Robinson, A. Voronkov (Hrsg.), Vol. II, Elsevier, 1635-1790, 2000.


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.

II Artikel, Dissertationen, Sammelbände (10)

- Patrick Cousot. *Methods and Logics for Proving Programs*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Ed.), 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*. Autom. Softw. Eng. 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.

II Artikel, Dissertationen, Sammelbände (11)

- 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.

II Artikel, Dissertationen, Sammelbände (12)


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.

II Artikel, Dissertationen, Sammelbände (13)


Patrick Cousot, Radhia Cousot. *Abstract Interpretation Frameworks*. Journal of Logic and Computation 2(4):511-547, 1992.


Patrick Cousot, Radhia Cousot. *Temporal Abstract Interpretation*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 12-25, 2000.


Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.

II Artikel, Dissertationen, Sammelbände (14)


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 Sience 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 21st Symposium on Programming (ESOP 2012), Springer-V., LNCS 7211, 169-193, 2012.

II Artikel, Dissertationen, Sammelbände (15)

Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Ken 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. Ken Zadeck. *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph*. ACM Transactions on Programming Languages and Systems 13(4):451-490, 1991.

Marvin Damschen, Lars Bauer, Jörg Henkel. *Timing Analysis of Tasks on Runtime Reconfigurable Processors*. In IEEE Transactions on Very Large Scale Integration Systems, 2016.

II Artikel, Dissertationen, Sammelbände (16)


D. M. Dhamdhere. *Register Assignment using Code Placement Techniques*. Journal of Computer Languages 13(2):75-93, 1988.


D. 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.

II Artikel, Dissertationen, Sammelbände (17)


D. 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.


Evelyn Duesterwald. *A Demand-driven Approach for Efficient Interprocedural Data-Flow Analysis*. PhD thesis, University of Pittsburgh, 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.

II Artikel, Dissertationen, Sammelbände (18)


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.


Stephen A. Edwards, Edward A. Lee. *The Case for the Precision-timed (PRET) Machine*. In Proc. of Design Automat. Conference, ACM, 264-265.


E. Allen Emerson. *Temporal and Modal Logic*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Hrsg.), Elsevier, 995-1072, 1990.

II Artikel, Dissertationen, Sammelbände (19)


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.


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 IEEE Conference Record of teh 15th Annual Symposium on Switching and Automata Theory (SWAT'74), 43-51, 1974.

II Artikel, Dissertationen, Sammelbände (20)


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.

II Artikel, Dissertationen, Sammelbände (21)

- Seymour Ginsburg, Sheila Greibach. *Deterministic Context Free Languages*. Information and Control 9(6):620-648, 1966.
- Orna Grumberg, Helmut Veith. *25 Years of Model Checking: History, Achievements, Perspectives*. Springer-V., LNCS 5000, 2008.
- J. 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.
- J. Gustafsson, A. Betts. *The Mälardalen WCET Benchmarks: Past, Present, and Future*. In Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2011), 136-146, 2010.
- Reiner Hähnle, Richard Bubel. *A Hoare-Style Calculus*

II Artikel, Dissertationen, Sammelbände (22)


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.


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.

II Artikel, Dissertationen, Sammelbände (23)


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.


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`

II Artikel, Dissertationen, Sammelbände (24)


Neil D. Jones, Flemming Nielson. *Abstract Interpretation: A Semantics-based Tool for Program Analysis*. In Handbook of Logic in Computer Science, Volume 4, Oxford University Press, 1995.


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


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

II Artikel, Dissertationen, Sammelbände (25)


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.


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.

II Artikel, Dissertationen, Sammelbände (26)

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, Germany, 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.

II Artikel, Dissertationen, Sammelbände (27)


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.

II Artikel, Dissertationen, Sammelbände (28)


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.

II Artikel, Dissertationen, Sammelbände (29)


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.

II Artikel, Dissertationen, Sammelbände (30)


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, 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`

II Artikel, Dissertationen, Sammelbände (31)

Laura Kovács, Tudor Jebelean. *Generation of Invariants in Theorema*. In Proceedings of the 10th International Symposium of Mathematics and its Applications, 407-415, 2003. `www.risc.jku.at/publications/download/risc_2053/2003-11-06-A.pdf`

Dexter Kozen, Jerzy Tiuryn. *On the Completeness of Propositional Hoare Logic*. Information Sciences 139(3-4):187-195, 2001.

Jean-Louis Lassez, V.L. Nguyen, Liz Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.

II Artikel, Dissertationen, Sammelbände (32)


T. Leveque, E. Borde, A. Marref, J. 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.


Y.-T. Li, S. 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

II Artikel, Dissertationen, Sammelbände (33)


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.

II Artikel, Dissertationen, Sammelbände (34)


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


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.


Flemming Nielson. *Program Transformations in a Denotational Setting*. ACM Transactions on Programming Languages and Systems 7(3):359-379, 1985.

II Artikel, Dissertationen, Sammelbände (35)


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.


Hanne Riis Nielson. *Hoare Logic's for Run-time Analysis of Programs*. PhD thesis, Edinburgh University, 1984.


Hanne Riis Nielson. *A Hoare-like Proof System for Run-Time Analysis of Programs*. Science of Computer Programming 9:107-136, 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*.

II Artikel, Dissertationen, Sammelbände (36)


Ernst-Rüdiger Olderog, Reinhard Wilhelm. *Turing und die Verifikation*. Informatik Spektrum 35(4):271-279, 2012.


G. Ottosson, M. Sjodin. *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, 1997.


Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Lecture notes, DAIMI FN-19, Aarhus University, Denmark, 1981, reprinted 1991.


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


Gordon D. Plotkin. *A Structural Approach to Operational*

II Artikel, Dissertationen, Sammelbände (37)


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, R.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.


John H. Reif, Harry R. Lewis. *Symbolic Evaluation and the*

II Artikel, Dissertationen, Sammelbände (38)


Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, Bernd Becker. *A Definition and Classification of Timing Anomalies*. In Proceedings 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.


Thomas Reps. *Demand Interprocedural Program Analysis using Logic Databases*. In Applications of Logic Databases, R. Ramakrishnan (Ed.), Kluwer Academic Publishers, 1994.

II Artikel, Dissertationen, Sammelbände (39)


Dirk Richter. *Programmanalysen zur Verbesserung der Softwaremodellprüfung*. Dissertation, Universität Halle-Wittenberg, 2012.


F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, France, Dec. 1976.

II Artikel, Dissertationen, Sammelbände (40)


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.


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.

II Artikel, Dissertationen, Sammelbände (41)


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.


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.

II Artikel, Dissertationen, Sammelbände (42)


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.


Bernhard Steffen. *Generating Data Flow Analysis Algorithms from Modal Specifications*. International Journal on Science of Computer Programming 21:115-139, 1993.

II Artikel, Dissertationen, Sammelbände (43)


Bernhard Steffen. *Property-Oriented Expansion*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 22-41, 1996.


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.

II Artikel, Dissertationen, Sammelbände (44)

- Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5:285-309, 1955.
- H. Theiling. *ILP-based Interprocedural Path Analysis*. 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.

II Artikel, Dissertationen, Sammelbände (45)

Mark N. Wegman, F. Ken Zadeck. *Constant Propagation With Conditional Branches*. ACM Transactions on Programming Languages and Systems 13(2):181-201, 1991.

II Artikel, Dissertationen, Sammelbände (46)

- 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.
- X. Yuan, Rajiv Gupta, R. Melham. *Demand-driven Data*

II Artikel, Dissertationen, Sammelbände (47)

X. Zheng, R. 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.

III Webseiten (1)

aiT Worst-Case Execution Time Analyzers. Website: http://www.absint.com/ait, 2016. [Online; accessed 1-August-2016]

Appendix

Appendix A

Mathematical Foundations

A.1

Relations

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

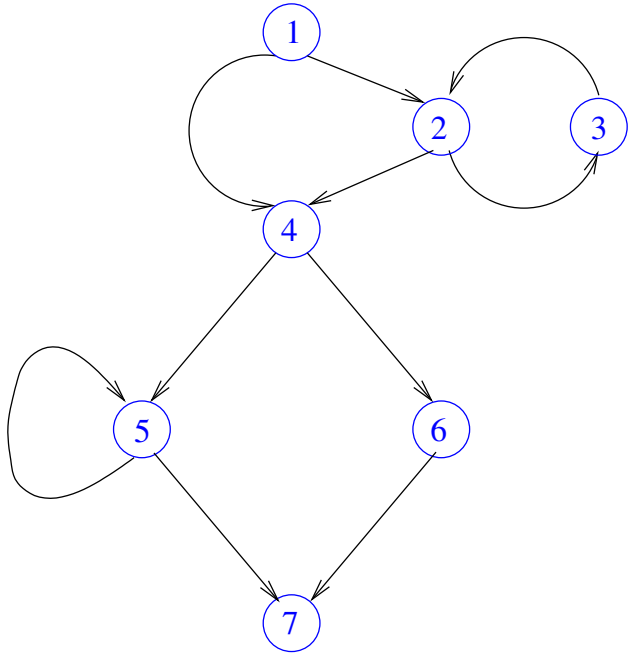
Definition A.1.5 (Properties of Relations on M)

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

Ordered Sets

A.2.1

Pre-Orders, Partial Orders, and More

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?

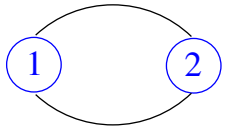
a)

{ }

b)

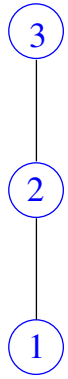
c)

d)

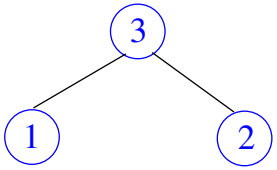


e)

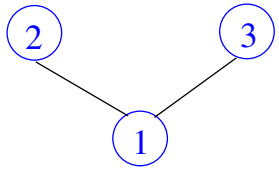
f)



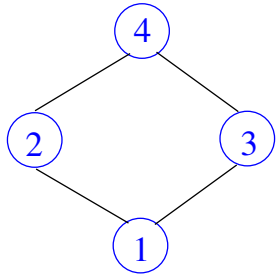
g)



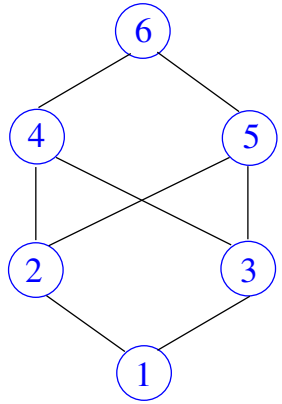
h)



i)



j)



A.2.2

Bounds and Extremal Elements

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?

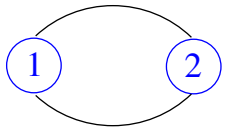
a)

{ }

b)

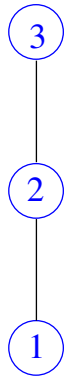
c)

d)

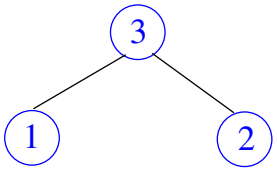


e)

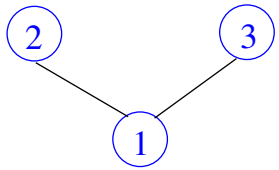
f)



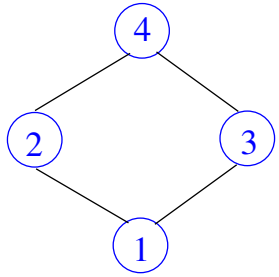
g)



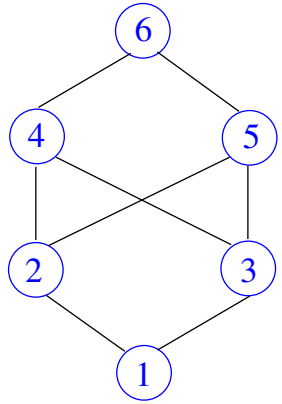
h)



i)



j)



A.2.3

Noetherian Orders, Artinian Orders, and Well-founded Orders

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

Chains

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

Directed Sets

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

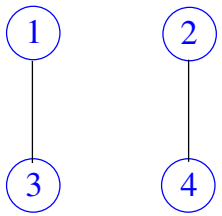
Let D be a strongly directed set. If D is finite, then $\bigsqcup D$ *exists* $\in D$ and is the greatest element of D.

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.

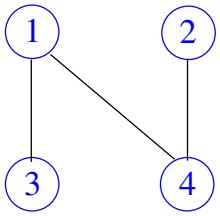
Exercise (1)

Which of the below partial orders are (strongly) directed sets?
Which of their subsets are (strongly) directed sets?

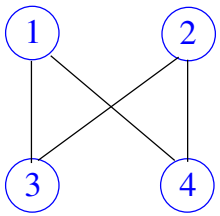
a)



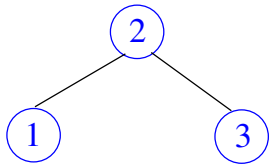
b)



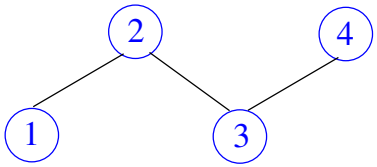
c)



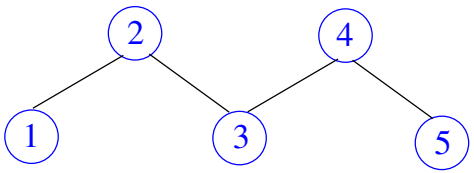
d)



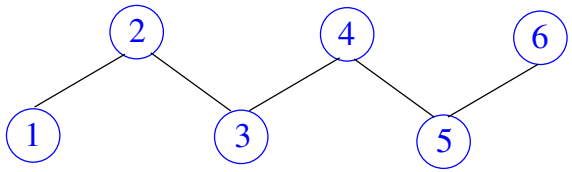
e)



f)



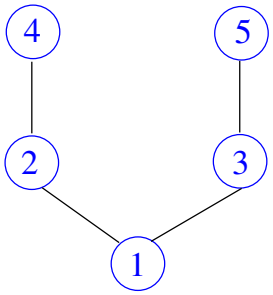
g)



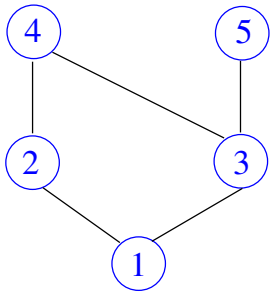
Exercise (2)

Which of the below partial orders are (strongly) directed sets?
Which of their subsets are (strongly) directed sets?

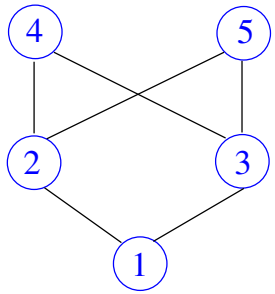
a)



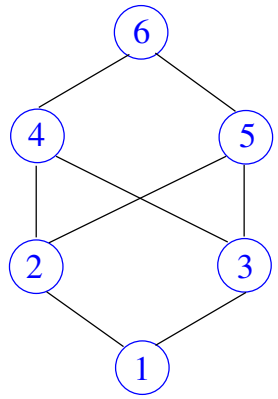
b)



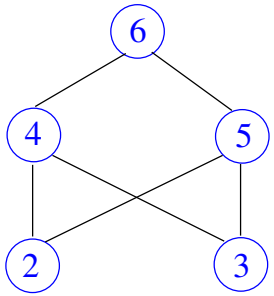
c)



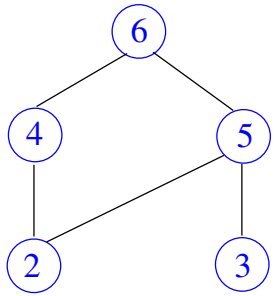
d)



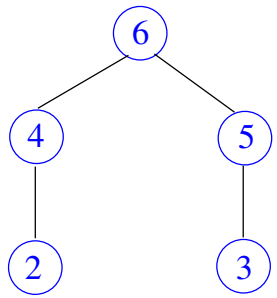
e)



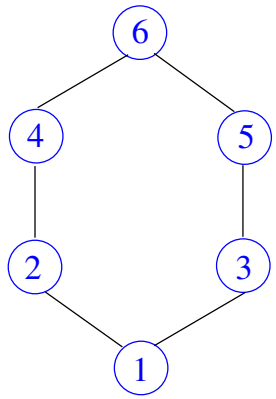
f)



g)



h)



A.2.6

Maps on Partial Orders

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

Order Homomorphisms and Order Isormorphisms between Partial Orders

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.

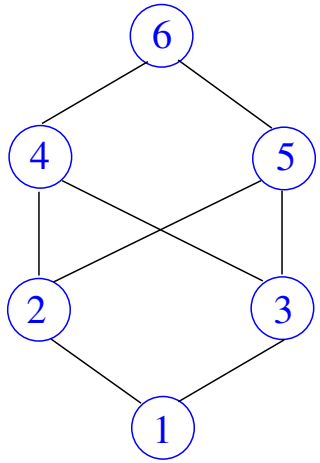
Intuitively: Partial orders, which are order isomorphic, are "essentially the same."

A.2.8

Hasse Diagrams

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
Complete Partially Ordered Sets

A.3.1

CCPOs and DCPOs

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

$$\forall\, s, s'' \in S.\ s \sqsubseteq_{pfx} s'' \iff_{df}$$
$$s = s'' \vee (s \textit{ finite } \wedge \exists\, s' \in S.\ s +\!\!+ s' = s'')$$

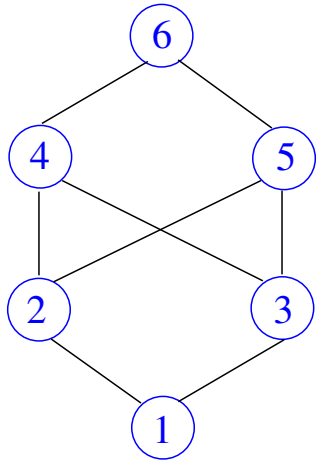
  is a CCPO and a DCPO.
- $(\{-n \mid n \in \mathbb{N}\}, \leq)$ is a pre-CCPO and a pre-DCPO but not a CCPO and not a DCPO.

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)

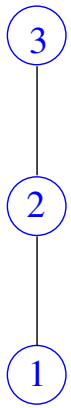
{ }

b)

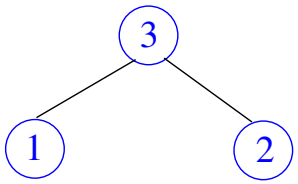
c)

d)

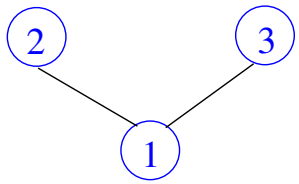
e)



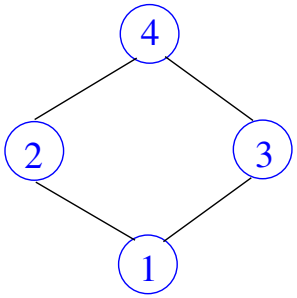
f)



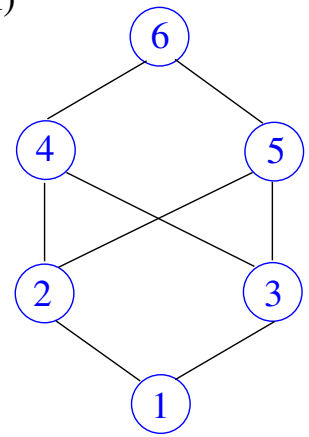
g)



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.3.1.7 (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.3.1.8 (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.3.1.9 (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.3.1.10 (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

Constructing Complete Partial Orders

Common CCPO and DCPO Constructions

The following construction principles hold for

- CCPOs
- DCPOs

Therefore, we simply write CPO.

Common CPO Constructions: Flat CPOs

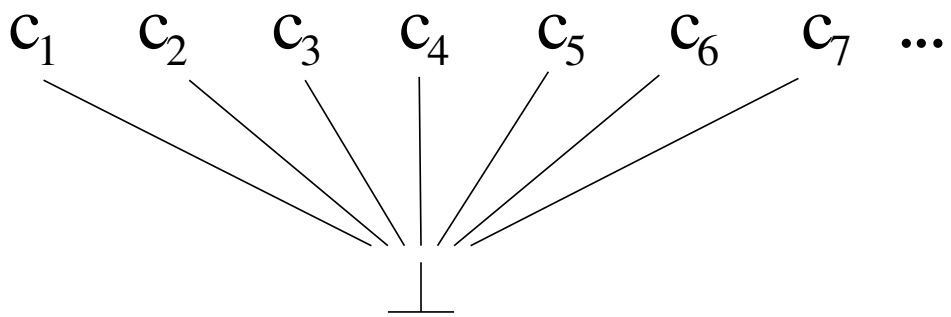
Lemma A.3.2.1 (Flat CPO Construction)

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

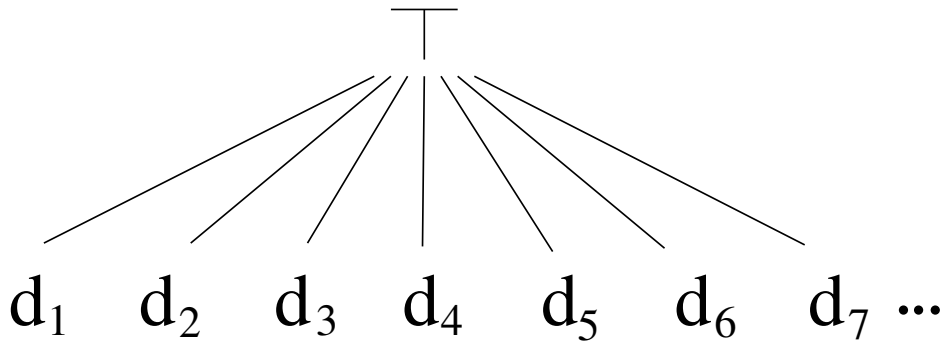
Lemma A.3.2.2 (Flat Pre-CPO Construction)

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 $(\bigoplus_\perp P_i, \sqsubseteq_{\oplus_\perp})$, where

- $\bigoplus_\perp P_i =_{df} P_1 \dot\cup P_2 \dot\cup \ldots \dot\cup P_n \dot\cup \{\perp\}$ is the disjoint union of all P_i, $1 \leq i \leq n$, and a fresh bottom element $\perp$
- $\sqsubseteq_{\oplus_\perp}$ is defined by

  $\forall\, p, q \in \bigoplus_\perp P_i.\ p \sqsubseteq_{\oplus_\perp} q \iff_{df}$

  $p = \perp \ \vee\ (\exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge\ p \sqsubseteq_i q)$

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

Lattices

A.4.1

Lattices, Complete Lattices, and Complete Semi-Lattices

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.

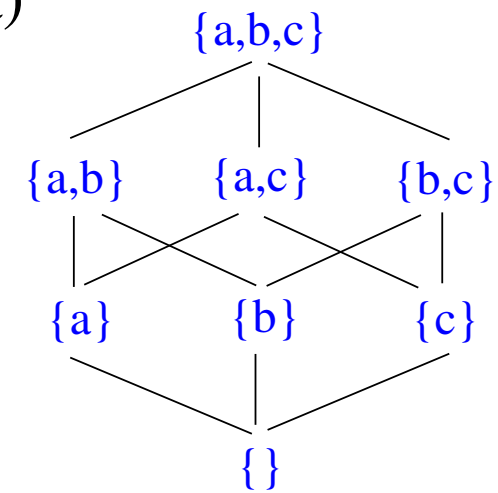
Corollary A.4.1.15 (Finite Lattices, CCPOs, DCPOs)

If $(P, \sqsubseteq)$ is a finite lattice, then $(P, \sqsubseteq)$ is a CCPO and a DCPO.

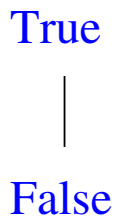
Note: Lemma A.4.1.14 does not hold for lattices.

Examples of Complete Lattices

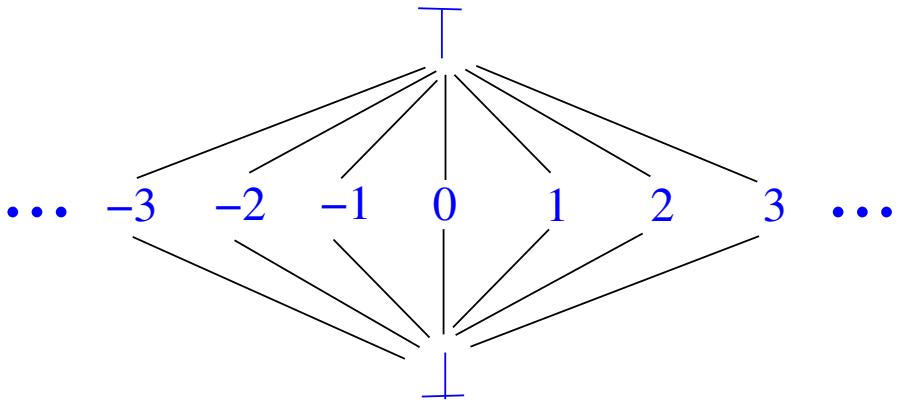
a)



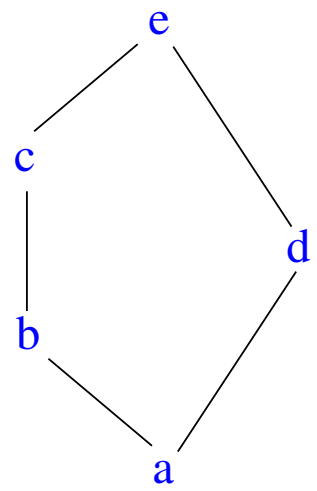
b)



c)

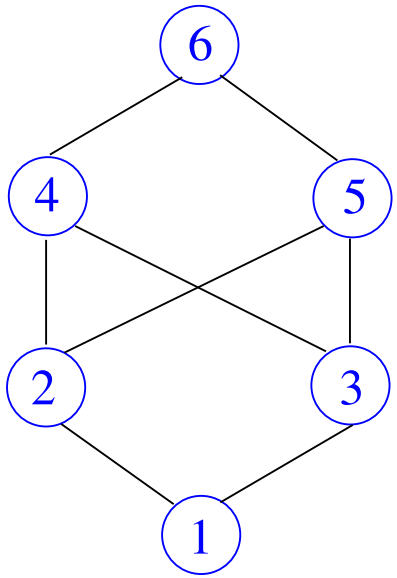


d)



(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)

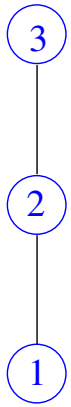
{ }

b)

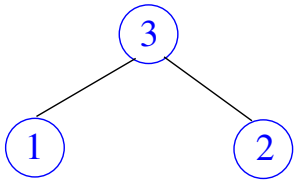
c)

d)

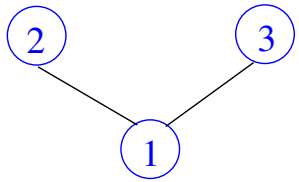
e)



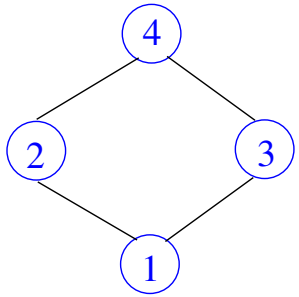
f)



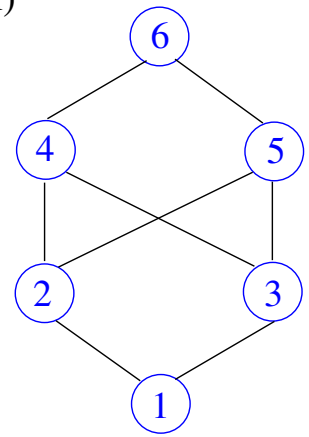
g)



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
$$\forall P' \subseteq P.\ f(\bigsqcap P') = \bigsqcap f(P')$$
(Preservation of greatest lower bounds)
- additive (or $\sqcup$-continuous) iff f is monotonic and
$$\forall P' \subseteq P.\ f(\bigsqcup P') = \bigsqcup f(P')$$
(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

Lattice Homomorphisms, Lattice Isomorphisms

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

Modular, Distributive, and Boolean Lattices

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

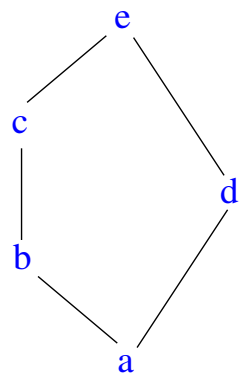
$$\forall\, p, q, r \in P.\ p \sqsubseteq r \Rightarrow p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap r$$

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.

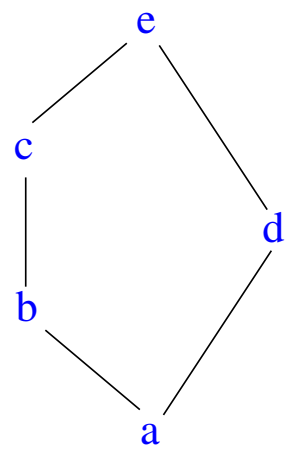
Characterizing Distributive Lattices

Let $(P, \sqsubseteq)$ be a lattice.

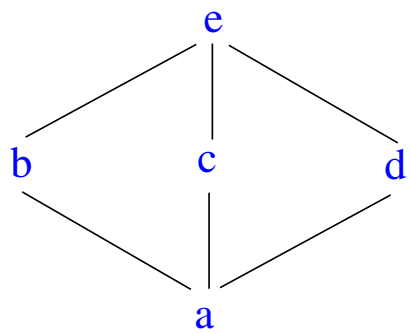
Theorem A.4.3.8 (Characterizing Distributive Lat.)

$(P, \sqsubseteq)$ 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

Constructing Lattices

Lattice Constructions: Flat Lattices

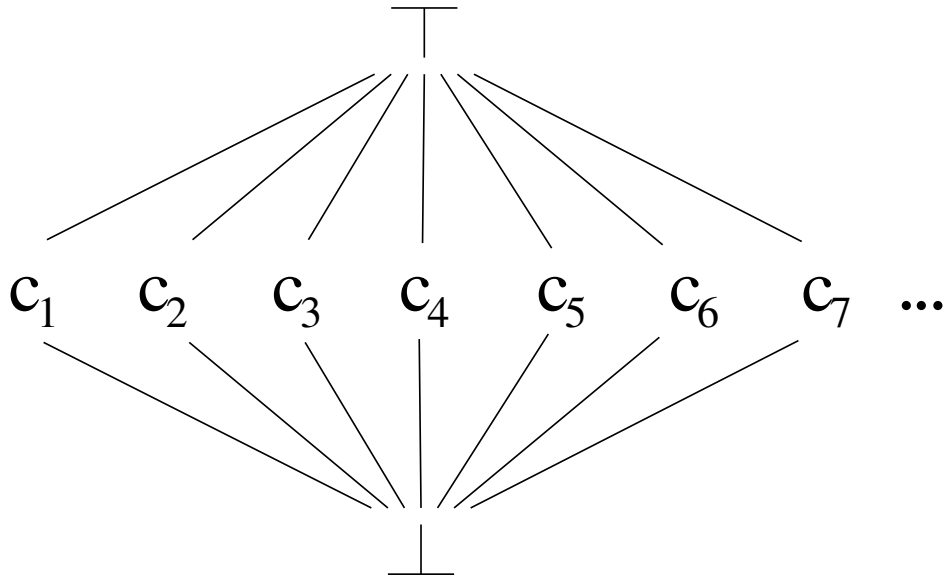
Lemma A.4.4.1 (Flat Construction)

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

Algebraic and Order-theoretic View of Lattices

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 "· carries more/less information than ·," "· is more/less defined than ·," "· is stronger/weaker than ·," etc., which often fits naturally to problems investigated and dealt with in computer science.

A.5

Fixed Point Theorems

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

Fixed Point Theorems for Complete Partial Orders

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

Fixed Point Theorems for Lattices

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

Fixed Point Induction

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: $\phi(\bot) = \mathit{true}$.

Proof. The admissibility of ϕ implies $\phi(\bigsqcup \emptyset) = \mathit{true}$. 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

References, Further Reading

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.


Paul Hudak. *The Haskell School of Expression: Learning Functional Programming through Multimedia*. Cambridge University Press, 2000. (Chapter 11, Proof by Induction; Chapter 14.6, Inductive Properties of Infinite Lists)


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)

Appendix A: Further Reading (5)


David Makinson. *Sets, Logic and Maths for Computing*. Springer-V., 2008. (Chapter 1, Collecting Things Together: Sets; Chapter 2, Comparing Things: Relations)


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)

Appendix A: Further Reading (6)


Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. Springer-V., 2nd edition, 2005. (Appendix A, Partially Ordered Sets)


Peter Pepper, Petra Hofstedt. *Funktionale Programmierung: Sprachdesign und Programmiertechnik*. Springer-V., 2006. (Kapitel 10, Beispiel: Berechnung von Fixpunkten; Kapitel 10.2, Ein bisschen Mathematik: CPOs und Fixpunkte)


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)

Appendix A: Further Reading (7)


Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5(2):285-309, 1955.


Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley/Pearson, 3rd edition, 2011. (Chapter 9, Reasoning about Programs; Chapter 17.9, Proof revisited)


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)