Analyse und Verifikation

LVA 185.276, VU 2.0, ECTS 3.0
SS 2012

Jens Knoop

Technische Universität Wien
Institut für Computersprachen

Inhaltsverzeichnis

Inhaltsverzeichnis (1)

Inhaltsverzeichnis (2)

Inhaltsverzeichnis (3)

Inhaltsverzeichnis (4)

Inhaltsverzeichnis (5)

Teil III: Transformation und Optimalität

Inhaltsverzeichnis (6)

Inhaltsverzeichnis (7)

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.

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

Literaturhinweise (2)

Ergänzend, vertiefend und weiterführend:

- Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. Verification of Sequential and Concurrent Programs. Springer-V., 3. Auflage, 2009.
- Ernst-Rüdiger Olderog, Bernhard Steffen. Formale Semantik und Programmverifikation. In Informatik-Handbuch, P. Rechenberg, G. Pomberger (Hrsg.), Carl Hanser Verlag, 129 - 148, 1997.
- Krzysztof R. Apt, Ernst-Rüdiger Olderog. Programmverifikation – Sequentielle, parallele und verteilte Programme. Springer-V., 1994.

Literaturhinweise (3)

- Jacques Loeckx and 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, 431 - 483, 1981.

Unser Beispielszenario

- Modellsprache 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

Intuition

- **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 in natürlichsprachlichen Dokumenten aufzudecken und aufzulösen
- bietet die Grundlage für 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 ($\rightsquigarrow$ Anw.programmierung)
    - Beweiskalküle für partielle & 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!

WHILE-Syntax und Semantik 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 $\Sigma =_{df} \{\sigma \mid \sigma : \mathbf{Var} \to D\}$ Menge aller Zustände über der Variablenmenge **Var** und geeignetem Datenbereich D.

(In der Folge werden wir für D oft die Menge der ganzen Zahlen $\mathbb{Z}$ betrachten. Notationelle Konvention: Der Pfeil $\to$ bezeichnet totale Funktionen, der Pfeil $\hookrightarrow$ partielle Funktionen.)

Syntax von Ausdrücken

Zahlausdrücke

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

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:

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

Ausblick

In der Folge werden wir im Detail betrachten:

- 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: ...*abweichender Fokus*

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

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

Kapitel 1.3

Semantik von Ausdrücken

Zahl-, arithmetische und Wahrheitswertausdrücke

Die Semantik von WHILE stützt sich ab auf die Semantik von

- Zahlausdrücken
- arithmetischen Ausdrücken
- Wahrheitswertausdrücken (Booleschen Ausdrücken)

Semantik von Zahlausdrücken

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

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

Bemerkung:

- 0, 1, 2, ... bezeichnen syntaktische Entitäten, Darstellungen von Zahlen; $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, ... bezeichnen semantische Entitäten, hier ganze Zahlen.
- *plus*, *mal*, *minus* bezeichnen semantische Operationen, hier die übliche Addition, Multiplikation und Vorzeichenwechsel auf den ganzen Zahlen.
- Die Semantik von Zahlausdrücken ist zustandsunabhängig.

Semantik arithmetischer und Wahrheitswert-ausdrücke

Semantik

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

Dabei bezeichnen

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

Semantik arithmetischer Ausdrücke

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

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

wobei

- $plus,\ mal,\ minus : \mathbb{Z} \times \mathbb{Z} \hookrightarrow \mathbb{Z}$ die übliche Addition, Multiplikation und Subtraktion auf den ganzen Zahlen $\mathbb{Z}$ bezeichnen.

Semantik Boolescher Ausdrücke (1)

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

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

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

Semantik Boolescher Ausdrücke (2)

...wobei

- **true** und **false** die Wahrheitswertkonstanten “wahr” und “falsch” sowie
- $conj$, $disj$: $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ und neg : $\mathbb{B} \to \mathbb{B}$ die übliche zweistellige logische Konjunktion und Disjunktion und einstellige Negation auf der Menge der Wahrheitswerte und
- $equal$: $\mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$ die übliche Gleichheitsrelation auf der Menge der ganzen Zahlen

bezeichnen.

Beachte auch hier den Unterschied zwischen den syntaktischen Entitäten *true* und *false* und ihren semantischen Gegenstücken **true** und **false**.

Vereinbarung

In der Folge seien die

- Semantik arithmetischer Ausdrücke:

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

- Semantik Boolescher Ausdrücke:

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

wie zuvor und die Menge der (Speicher-) Zustände wie folgt festgelegt:

- (Speicher-) Zustände: $\Sigma =_{df} \{ \, \sigma \mid \sigma : \mathbf{Var} \to \mathbb{Z} \, \}$

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 $a \in$ **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 $b \in$ **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/Semantische Substitution

Von zentraler Bedeutung: Der Substitutionsbegriff!

Substitutionen

- Syntaktische Substitution
- Semantische Substitution
- Substitutionslemma

Syntaktische Substitution

Definition (1.4.3)

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:

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

Semantische Substitution

Definition (1.4.4)

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

Wichtig:

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(3)

Zentral:

- Vollständige Induktion
- Verallgemeinerte Induktion
- Strukturelle Induktion

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

Induktive Beweisprinzipien 2(3)

Zur Erinnerung hier wiederholt:

Die Prinzipien der

- vollständigen Induktion

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

- verallgemeinerten Induktion

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

- strukturellen Induktion

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

Bemerkung: $\succ$ bezeichnet die logische Implikation; A erinnert an Aussage, S an (induktiv definierte) Struktur, Komp(s) an Menge der Komponenten von s.

Induktive Beweisprinzipien 3(3)

Bemerkung:

- 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 arithmethischer 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

Semantikdefinitionsstile (1)

Es gibt unterschiedliche Stile, die Semantik einer Programmiersprache festzulegen. Sie richten sich an unterschiedliche Adressaten und deren spezifische Sicht auf die Sprache.

Insbesondere unterscheiden wir den

- denotationellen
- operationellen
- axiomatischen

Stil.

Semantikdefinitionsstile (2)

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

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 1 (1)

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

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.

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

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 1 (2)

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)

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 insbesondere, 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 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*. Lecture notes, DAIMI FN-19, Aarhus University, Denmark, 1981, reprinted 1991.

Strukturell operationelle Semantik 1(2)

i.S.v. Gordon D. Plotkin.

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

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

das jedem WHILE-Programm π als Bedeutung eine partiell definierte Zustandstransformation zuordnet.

Dieses Zustandstransformationsfunktional $[\![\, . \,]\!]_{sos}$ werden wir jetzt im Detail definieren.

Strukturell operationelle Semantik 2(2)

Intuitiv:

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

Zentral:

- Der Begriff der Konfiguration!

Konfigurationen

- Wir unterscheiden:
  - Nichtterminale bzw. (Zwischen-) Konfigurationen γ der Form $\langle \pi, \sigma \rangle$:
    ...(Rest-) Programm π ist auf den (Zwischen-) Zustand σ anzuwenden
  - Terminale bzw. finale Konfigurationen γ der Form σ
    ...beschreiben das Resultat nach Ende der Berechnung
- Vereinbarung:
  - $\Gamma =_{df} (\mathbf{Prg} \times \Sigma) \cup \Sigma$ bezeichne die Menge aller Konfigurationen, $\gamma \in \Gamma$

SOS-Regeln von WHILE – Axiome 1(2)

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

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

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

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

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

SOS-Regeln von WHILE – Regeln 2(2)

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

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

Sprechweisen (1)

Wir unterscheiden

- Prämissenlose Axiome der Form

$$\frac{}{\text{Konklusion}}$$

- Prämissenbehaftete Regeln der Form

$$\frac{\text{Prämisse}}{\text{Konklusion}}$$

ggf. mit Randbedingungen (Seitenbedingungen) wie z.B. in Form von $[\![b]\!]_B(\sigma) = \textbf{false}$ in der Regel $[\text{if}^{ff}_{sos}]$.

Sprechweisen (2)

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

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

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

Terminierende vs. divergierende Berechnungsfolgen

- 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(7)

Sei

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

Betrachte

- die von π angesetzt auf σ, d.h. die von der Anfangskonfiguration

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

  induzierte Berechnungsfolge

Beispiel 2(7)

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

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

Beispiel 3(7)

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

Beispiel 4(7)

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

Beispiel – Detailbetrachtung 5(7)

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

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

steht vereinfachend für:

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

Beispiel – Detailbetrachtung 6(7)

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

steht vereinfachend für:

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

Beispiel – Detailbetrachtung 7(7)

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

steht vereinfachend für:

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

Determinismus der SOS-Regeln

Lemma (2.1.1)

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

Erinnerung: $\succ$ bezeichnet hier die logische Implikation.

Corollary (2.1.2)

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

Korollar 2.1.2 erlaubt uns jetzt festzulegen:

Die strukturell operationelle Semantik von WHILE ist gegeben durch das Funktional

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

welches definiert ist durch:

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

Beachte: $\Rightarrow^*$ bezeichnet die reflexiv-transitive Hülle von $\Rightarrow$.

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.

$\rightsquigarrow$ typisches Beweisprinzip im Zshg. mit Aussagen zur SO-Semantik.

Anwendung

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

Ein Beispiel dafür ist der Beweis von

Lemma (2.1.3)

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

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

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(2)

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

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

das jedem WHILE-Programm π als Bedeutung eine partiell definierte Zustandstransformation zuordnet.

Dieses Zustandstransformationsfunktional $[\![\, . \,]\!]_{ns}$ werden wir jetzt im Detail definieren.

Natürliche Semantik 2(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(2)

...der Modellsprache WHILE:

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

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

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

NS-Regeln von WHILE 2(2)

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

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

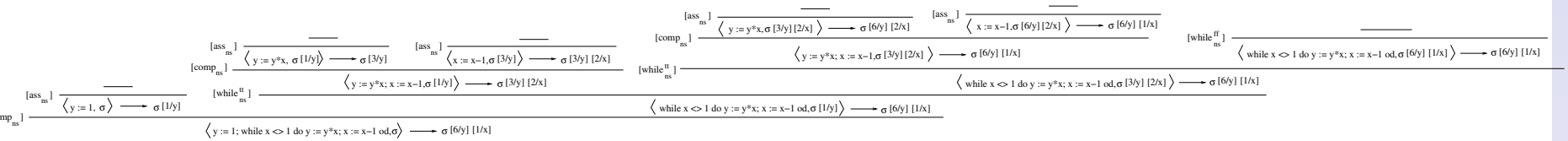
$$[\text{while}^{tt}_{ns}] \quad \frac{\langle \pi, \sigma \rangle \to \sigma', \langle \text{while } b \text{ do } \pi \text{ od}, \sigma' \rangle \to \sigma''}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \to \sigma''} \qquad [\![b]\!]_B(\sigma) = \textbf{true}$$

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

Beispiel 1(2)

Sei $\sigma \in \Sigma$ mit $\sigma(x) = 3$.
Dann gilt:

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



Beispiel 2(2)

Das gleiche Beispiel in etwas gefälligerer Darstellung:

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

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

Determinismus der NS-Regeln

Lemma (2.2.1)

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

Corollary (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 $[\![\ ]\!]_{ns}$

Korollar 2.2.2 erlaubt uns festzulegen:

Die natürliche Semantik von WHILE ist gegeben durch das Funktional

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

welches 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 (ggf. vorhandenen) Randbedingungen der Regel erfüllt sind.

$\rightsquigarrow$ typisches Beweisprinzip im Zshg. mit Aussagen zur N-Semantik.

Anwendung

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

Ein Beispiel dafür 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 $\langle \pi, \sigma \rangle \Rightarrow \gamma$ mit γ von der Form $\langle \pi', \sigma' \rangle$ oder σ': Transition beschreibt den ersten Schritt d. Berechnungsf. v. π angesetzt auf σ.

  Dabei sind folgende Übergänge möglich:

  - γ von der Form $\langle \pi', \sigma' \rangle$:
    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 0), so terminiert die Abarbeitung von π in $\langle \pi', \sigma' \rangle$ irregulär.
  - γ von der Form σ':
    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)

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

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

Lemma (2.3.2)

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

Beweis durch Induktion über die Länge der Ableitungsfolge $\langle \pi, \sigma \rangle \Rightarrow^k \sigma'$, d.h. durch Induktion über k.

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

Aus Lemma 2.3.1 und Lemma 2.3.2 folgt:

Theorem (2.3.3)

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

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 2

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)

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 ist gegeben durch ein Funktional

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

  das jedem Programm π aus WHILE als Bedeutung eine partiell definierte Zustandstransformation zuordnet.

Das Zustandstransformationsfunktional $[\![\ ]\!]_{ds}$ werden wir jetzt im Detail definieren.

Vergleich operationelle vs. denotationelle 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, i.e. 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

Denotationelle Semantik / Definierende Gleichungen

...der Modellsprache WHILE:

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

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

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

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

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

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

wobei $\mathit{Id} : \Sigma \to \Sigma$ die identische Zustandstransformation bezeichnet: $\forall\, \sigma \in \Sigma.\ \mathit{Id}(\sigma) =_{df} \sigma$

Die Hilfsfunktion cond

Funktionalität:

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

Definiert durch

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

Zu den Argumenten und zum Resultat von *cond*:

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

Daraus ergibt sich

...für die Bedeutung der Fallunterscheidung:

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

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

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

Erinnerung:

- In unserem Szenario sind $[\![.]\!]_{\mathcal{A}}$ und $[\![.]\!]_{\mathcal{B}}$ 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:

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

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

   while b do π od und if b then (π; while b do π od) else skip fi

   denselben Effekt haben.
2. Somit gilt folgende Gleichheit:
   $[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} =$

   $$cond([\![b]\!]_{\mathcal{B}}, [\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} \circ [\![\pi]\!]_{ds}, Id)$$
3. Anwenden von F auf $[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds}$ liefert ebenfalls:

   $$F\,[\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} =_{df} cond([\![b]\!]_{\mathcal{B}}, [\![\text{while } b \text{ do } \pi \text{ od}]\!]_{ds} \circ [\![\pi]\!]_{ds},$$
4. Aus 2.) und 3.) folgt damit auch folgende Gleichheit:

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

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

   In der Folge werden wir sehen: Der kleinste Fixpunkt!

Denotationelle Semantik / Existenz

Lemma (3.1.1)

*Für alle $\pi \in$ **Prg** ist durch die Gleichungen von Folie "Denotationelle Semantik / Definierende Gleichungen" eine (partielle) Funktion definiert.*

Wir bezeichnen diese Funktion als denotationelle Semantik von π bezeichnet durch $[\![\pi]\!]_{ds}$.

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

Theorem (3.1.2)

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

Beweis folgt aus Lemma 3.1.3, Lemma 3.1.4 und Theorem 2.3.3.

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

Lemma (3.1.3)

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

Beweis durch strukturelle Induktion über den induktiven Aufbau von π.

Lemma (3.1.4)

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

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

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

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 (z.B.) durch

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

Kapitel 3.2

Fixpunktfunktional

Zur Bedeutung von FIX F

In der Folge wollen wir die Bedeutung von

- *FIX F*

genauer 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:

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

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:

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

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

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

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 $[\![\quad]\!]_{ds}$.

Unser Arbeitsplan

Wir benötigen:

- Einige Resultate aus der *Fixpunkttheorie*.

Diese erlauben uns:

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

Dabei wird vorausgesetzt:

- Mathematischer Hintergrund (Ordnungen, CPOs, Stetigkeit von Funktionen) und die benötigten Resultate (Fixpunktsatz).

Dieser Hintergrund wird in Kapitel 3.3 geliefert.

Folgende drei Argumente

...werdend entscheidend sein:

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

Insgesamt ergibt sich daraus 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 $[\Sigma \hookrightarrow \Sigma]$:

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

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* $\bot\ \sigma = \textit{undef}$ *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: $(\bigsqcup Y)\,\sigma = \sigma' \iff \exists\, g \in Y.\ g\ \sigma = \sigma'$

Einschub / Graph einer Funktion

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

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

Es gilt:

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

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 (3.2.3)

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.*

Erinnerung:
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. $\forall\, C' \subseteq C.\ f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$
  (Erhalt der kleinsten oberen Schranken)

Stetigkeitsresultate (2)

Lemma (3.2.4)

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 (3.2.5)

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:

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

ist wohldefiniert!

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

Mengen, Relationen, Ordnungen und Verbände

Mathematische Grundlagen

...im Zusammenhang mit der

1. Definition abstrakter Semantiken für Programmanalysen
2. Definition der denotationellen Semantik von WHILE im Detail

Wichtig insbesondere...

- Mengen, Relationen, Verbände
- Partielle und vollständige partielle Ordnungen
- Schranken, Fixpunkte und Fixpunkttheoreme

Mengen und Relationen 1(2)

Sei M eine Menge und R eine Relation auf M,
d.h. $R \subseteq M \times M$.

Dann heißt R

- reflexiv gdw. $\forall\, m \in M.\ m\, R\, m$
- transitiv gdw. $\forall\, m, n, p \in M.\ m\, R\, n \ \wedge\ n\, R\, p \ \succ\ m\, R\, p$
- antisymmetrisch
  gdw. $\forall\, m, n \in M.\ m\, R\, n \ \wedge\ n\, R\, m \ \succ\ m = n$

Darüberhinaus (in der Folge jedoch nicht benötigt):

- symmetrisch gdw. $\forall\, m, n \in M.\ m\, R\, n \iff n\, R\, m$
- total gdw. $\forall\, m, n \in M.\ m\, R\, n \ \vee\ n\, R\, m$

Mengen und Relationen 2(2)

Eine Relation R auf M heißt

- Quasiordnung gdw. R ist reflexiv und transitiv
- partielle Ordnung gdw. R ist reflexiv, transitiv und antisymmetrisch

Zur Vollständigkeit sei ergänzt:

- Äquivalenzrelation gdw. R ist reflexiv, transitiv und symmetrisch

...eine partielle Ordnung ist also eine antisymmetrische Quasiordnung, eine Äquivalenzrelation eine symmetrische Quasiordnung.

Schranken, kleinste/größte Elemente

Sei $(Q, \sqsubseteq)$ eine Quasiordnung, sei $q \in Q$ und $Q' \subseteq Q$.
Dann heißt q

- obere (untere) Schranke von Q', in Zeichen: $Q' \sqsubseteq q$ ($q \sqsubseteq Q'$), wenn für alle $q' \in Q'$ gilt: $q' \sqsubseteq q$ ($q \sqsubseteq q'$)
- kleinste obere (größte untere) Schranke von Q', wenn q obere (untere) Schranke von Q' ist und für jede andere obere (untere) Schranke $\hat{q}$ von Q' gilt: $q \sqsubseteq \hat{q}$ ($\hat{q} \sqsubseteq q$)
- größtes (kleinstes) Element von Q, wenn gilt: $Q \sqsubseteq q$ ($q \sqsubseteq Q$)

Eindeutigkeit von Schranken

- In partiellen Ordnungen sind kleinste obere und größte untere Schranken eindeutig bestimmt, wenn sie existieren.
- Existenz (und damit Eindeutigkeit) vorausgesetzt, wird die kleinste obere (größte untere) Schranke einer Menge $P' \subseteq P$ der Grundmenge einer partiellen Ordnung $(P, \sqsubseteq)$ mit $\bigsqcup P'$ ($\bigsqcap P'$) bezeichnet. Man spricht dann auch vom Supremum und Infimum von P'.
- Analog für kleinste und größte Elemente. Existenz vorausgesetzt, werden sie üblicherweise mit $\bot$ und $\top$ bezeichnet.

Verbände und vollständige Verbände

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.
Dann heißt $(P, \sqsubseteq)$

- Verband, wenn jede endliche Teilmenge P' von P eine kleinste obere und eine größte untere Schranke in P besitzt
- vollständiger Verband, wenn jede Teilmenge P' von P eine kleinste obere und eine größte untere Schranke in P besitzt

...(vollständige) Verbände sind also spezielle partielle Ordnungen.

Vollständige partielle Ordnungen

...ein etwas schwächerer, aber in der Informatik oft ausreichender und daher angemessenerer Begriff.

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Dann heißt $(P, \sqsubseteq)$

- vollständig, kurz CPO (von engl. complete partial order), wenn jede aufsteigende Kette $K \subseteq P$ eine kleinste obere Schranke in P besitzt.

Es gilt:

- Eine CPO $(C, \sqsubseteq)$ (genauer wäre: "kettenvollständige partielle Ordnung (engl. chain complete partial order (CCPO)") besitzt stets ein kleinstes Element, eindeutig bestimmt als Supremum der leeren Kette und üblicherweise mit $\bot$ bezeichnet: $\bot =_{df} \bigsqcup \emptyset$.

Ketten

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Eine Teilmenge $K \subseteq P$ heißt

- Kette in P, wenn die Elemente in K total geordnet sind. Für $K = \{k_0 \sqsubseteq k_1 \sqsubseteq k_2 \sqsubseteq \ldots\}$ ($\{k_0 \sqsupseteq k_1 \sqsupseteq k_2 \sqsupseteq \ldots\}$) spricht man auch genauer von einer aufsteigenden (absteigenden) Kette in P.

Eine Kette K heißt

- endlich, wenn K endlich ist, sonst unendlich.

Kettenendlichkeit, endliche Elemente

Eine partielle Ordnung $(P, \sqsubseteq)$ heißt

- kettenendlich gdw. P enthält keine unendlichen Ketten

Ein Element $p \in P$ heißt

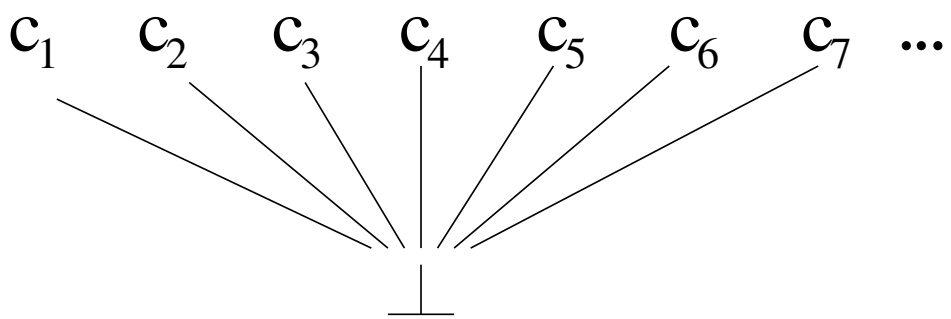
- endlich gdw. die Menge $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$ keine unendliche Kette enthält
- endlich relativ zu $r \in P$ gdw. die Menge $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$ keine unendliche Kette enthält

(Standard-) CPO-Konstruktionen 1(4)

Flache CPOs:

Sei $(C, \sqsubseteq)$ eine CPO. Dann heißt $(C, \sqsubseteq)$

- flach, wenn für alle $c, d \in C$ gilt:
  $c \sqsubseteq d \Leftrightarrow c = \bot \ \vee \ c = d$



(Standard-) CPO-Konstruktionen 2(4)

Produktkonstruktion:

Seien $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ CPOs. Dann sind auch

- das nichtstrikte (direkte) Produkt $(\bigtimes P_i, \sqsubseteq)$ mit
  - $(\bigtimes P_i, \sqsubseteq) = (P_1 \times P_2 \times \ldots \times P_n, \sqsubseteq)$ mit
    $\forall\, (p_1, p_2, \ldots, p_n),$
    $(q_1, q_2, \ldots, q_n) \in \bigtimes P_i.\ (p_1, p_2, \ldots, p_n) \sqsubseteq$
    $(q_1, q_2, \ldots, q_n) \nleftrightarrow \forall\, i \in \{1, \ldots, n\}.\ p_i \sqsubseteq_i q_i$
- und das strikte (direkte) Produkt (smash Produkt) mit
  - $(\bigotimes P_i, \sqsubseteq) = (P_1 \otimes P_2 \otimes \ldots \otimes P_n, \sqsubseteq)$, wobei $\sqsubseteq$ wie oben definiert ist, jedoch zusätzlich gesetzt wird:

$$(p_1, p_2, \ldots, p_n) = \bot \nleftrightarrow \exists\, i \in \{1, \ldots, n\}.\ p_i = \bot_i$$

CPOs.

(Standard-) CPO-Konstruktionen 3(4)

Summenkonstruktion:

Seien $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ CPOs. Dann ist auch

- die direkte Summe $(\bigoplus P_i, \sqsubseteq)$ mit
  - $(\bigoplus P_i, \sqsubseteq) = (P_1 \dot\cup P_2 \dot\cup \ldots \dot\cup P_n, \sqsubseteq)$ disjunkte Vereinigung der P_i, $i \in \{1, \ldots, n\}$ und $\forall\, p, q \in \bigoplus P_i$. $p \sqsubseteq q \gdw \exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge\ p \sqsubseteq_i q$ eine CPO.

    Bem.: Die kleinsten Elemente der $(P_i, \sqsubseteq_i)$, $i \in \{1, \ldots, n\}$ werden meist identifiziert, d.h. $\bot =_{df} \bot_i$, $i \in \{1, \ldots, n\}$

(Standard-) CPO-Konstruktionen 4(4)

Funktionenraum:

Seien $(C, \sqsubseteq_C)$ und $(D, \sqsubseteq_D)$ zwei CPOs und $[C \to D] =_{df} \{f : C \to D \mid f \text{ stetig}\}$ die Menge der stetigen Funktionen von C nach D.
Dann ist auch

- der stetige Funktionenraum $([C \to D], \sqsubseteq)$ eine CPO mit
  - $\forall f, g \in [C \to D].\ f \sqsubseteq g \Longleftrightarrow \forall c \in C.\ f(c) \sqsubseteq_D g(c)$

Funktionen auf CPOs / Eigenschaften

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

Sei $(C, \sqsubseteq)$ eine CPO und sei $f : C \to C$ eine Funktion auf C.
Dann heißt f

- inflationär (vergrößernd) gdw. $\forall\, c \in C.\ c \sqsubseteq f(c)$

Funktionen auf CPOs / Resultate

Mit den vorigen Bezeichnungen gilt:

Lemma (3.3.1)

f ist monoton gdw. $\forall\, C' \subseteq C.\ f(\bigsqcup_C C') \sqsupseteq_D \bigsqcup_D f(C')$

Corollary (3.3.2)

Eine stetige Funktion ist stets monoton, d.h. f stetig $\succ$ f monoton.

(Kleinste und größte) Fixpunkte 1(2)

Sei $(C, \sqsubseteq)$ eine CPO, $f : C \to C$ eine Funktion auf C und sei c ein Element von C, also $c \in C$.
Dann heißt c

- Fixpunkt von f gdw. $f(c) = c$

Ein Fixpunkt c von f heißt

- kleinster Fixpunkt von f gdw. $\forall\, d \in C.\ f(d) = d \succ c \sqsubseteq d$
- größter Fixpunkt von f gdw. $\forall\, d \in C.\ f(d) = d \succ d \sqsubseteq c$

(Kleinste und größte) Fixpunkte 2(2)

Seien $d, c_d \in C$. Dann heißt c_d

- bedingter kleinster Fixpunkt von f bezüglich d gdw. c_d ist der kleinste Fixpunkt von C mit $d \sqsubseteq c_d$, d.h. für alle anderen Fixpunkte x von f mit $d \sqsubseteq x$ gilt: $c_d \sqsubseteq x$.

Bezeichnungen:

Der kleinste bzw. größte Fixpunkt einer Funktion f wird oft mit μf bzw. νf bezeichnet.

Fixpunktsatz

Theorem (3.3.3 – Knaster/Tarski, Kleene)

Sei $(C, \sqsubseteq)$ eine CPO und sei $f : C \to C$ eine stetige Funktion auf C.

Dann hat f einen kleinsten Fixpunkt μf und dieser Fixpunkt ergibt sich als kleinste obere Schranke der Kette (sog. Kleene-Kette) $\{\bot, f(\bot), f^2(\bot), \ldots\}$, d.h.

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

Beweis des Fixpunktsatzes 3.3.3 (1)

Zu zeigen: μf:

1. existiert
2. ist Fixpunkt
3. ist kleinster Fixpunkt

Beweis des Fixpunktsatzes 3.3.3 (2)

1. Existenz
   - Es gilt $f^0 \perp = \perp$ und $\perp \sqsubseteq c$ für alle $c \in C$.
   - Durch vollständige Induktion lässt sich damit zeigen: $f^n \perp \sqsubseteq f^n c$ für alle $c \in C$.
   - Somit gilt $f^n \perp \sqsubseteq f^m \perp$ für alle n, m mit $n \leq m$. Somit ist $\{f^n \perp \mid n \geq 0\}$ eine (nichtleere) Kette in C.
   - Damit folgt die Existenz von $\bigsqcup_{i \in \mathbb{N}_0} f^i(\perp)$ aus der CPO-Eigenschaft von $(C, \sqsubseteq)$.

Beweis des Fixpunktsatzes 3.3.3 (3)

2. Fixpunkteigenschaft

$$
\begin{aligned}
& & f\left(\bigsqcup_{i\in\mathbb{N}_0} f^i(\bot)\right) & \\
(f \text{ stetig}) & = & \bigsqcup_{i\in\mathbb{N}_0} f(f^i\bot) & \\
& = & \bigsqcup_{i\in\mathbb{N}_1} f^i\bot & \\
(K \text{ Kette} \Rightarrow \bigsqcup K = \bot \sqcup \bigsqcup K) & = & \left(\bigsqcup_{i\in\mathbb{N}_1} f^i\bot\right) \sqcup \bot & \\
(f^0 = \bot) & = & \bigsqcup_{i\in\mathbb{N}_0} f^i\bot &
\end{aligned}
$$

Beweis des Fixpunktsatzes 3.3.3 (4)

3. Kleinster Fixpunkt

- Sei c beliebig gewählter Fixpunkt von f. Dann gilt $\bot \sqsubseteq c$ und somit auch $f^n \bot \sqsubseteq f^n c$ für alle $n \geq 0$.
- Folglich gilt $f^n \bot \sqsubseteq c$ wg. der Wahl von c als Fixpunkt von f.
- Somit gilt auch, dass c eine obere Schranke von $\{f^i(\bot) \mid i \in \mathbb{N}_0\}$ ist.
- Da $\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)$ nach Definition die kleinste obere Schranke dieser Kette ist, gilt wie gewünscht $\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \sqsubseteq c$.

Bedingte Fixpunkte

Theorem (3.3.4 – Bedingte Fixpunkte)

Sei $(C, \sqsubseteq)$ eine CPO, sei $f : C \to C$ eine stetige, inflationäre Funktion auf C und sei $d \in C$.

Dann hat f einen kleinsten bedingten Fixpunkt μf_d und dieser Fixpunkt ergibt sich als kleinste obere Schranke der Kette $\{d, f(d), f^2(d), \ldots\}$, d.h.

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

Endliche Fixpunkte

Theorem (3.3.5 – Endliche Fixpunkte)

Sei $(C, \sqsubseteq)$ eine CPO und sei $f : C \to C$ eine stetige Funktion auf C.
Dann gilt: Sind in der Kleene-Kette von f zwei aufeinanderfolgende Glieder gleich, etwa $f^i(\bot) = f^{i+1}(\bot)$, so gilt $\mu f = f^i(\bot)$.

Existenz endlicher Fixpunkte

Hinreichende Bedingungen für die Existenz endlicher Fixpunkte sind:

- Endlichkeit von Definitions- und Wertebereich von f
- f ist von der Form $f(c) = c \sqcup g(c)$ für monotones g über kettenendlichem Wertebereich

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 3

M.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)

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

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 klassische Arbeiten

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

Kapitel 4.1

Partielle und totale Korrektheit

Definition partieller Korrektheit

Sei $\pi \in$ **Prg** ein WHILE-Programm:

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

- gültig (im Sinne der partiellen Korrektheit) oder kurz (partiell) korrekt 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.

Definition totaler Korrektheit

Sei $\pi \in$ **Prg** ein WHILE-Programm:

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

- gültig (im Sinne der totalen Korrektheit) oder kurz (total) korrekt 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"

Partielle und totale Korrektheit

- Die Zustandsmenge

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

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

- Semantik von Korrektheitsformeln:
  Eine Korrektheitsformel $\{p\}\ \pi\ \{q\}$ heißt
  - 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

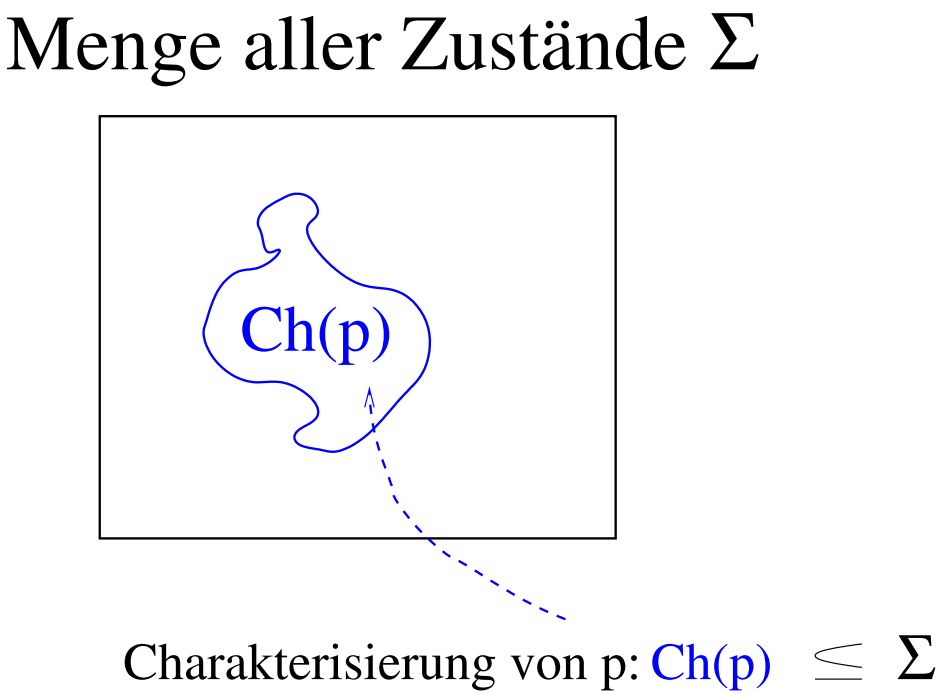
...an einige Sprechweisen:

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 regulär.

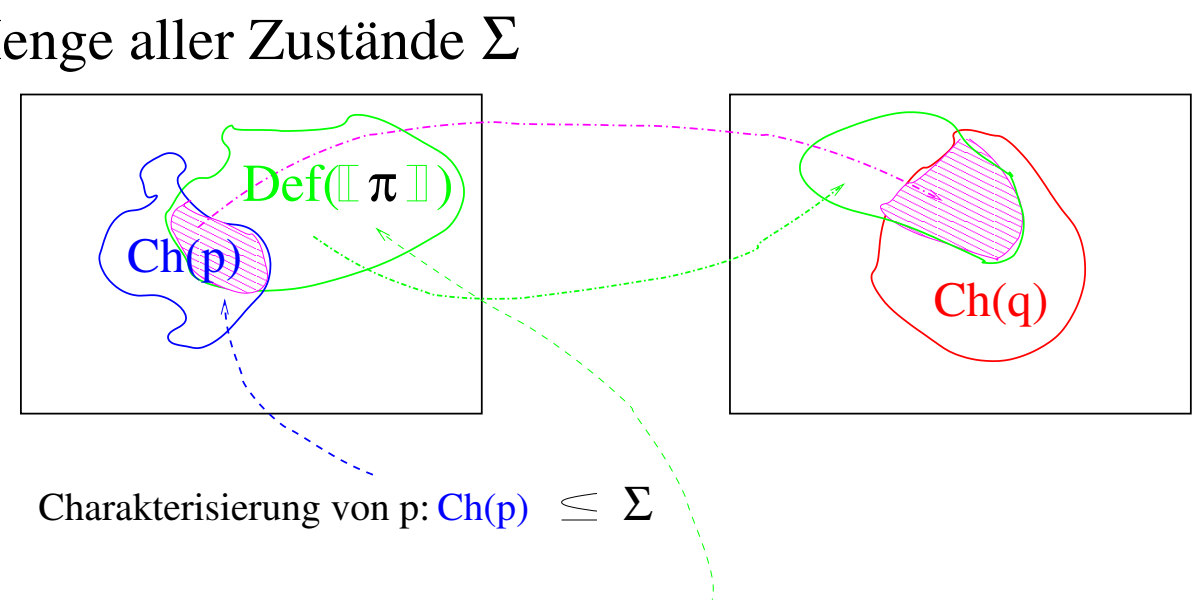
Veranschaulichung 1(3)

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



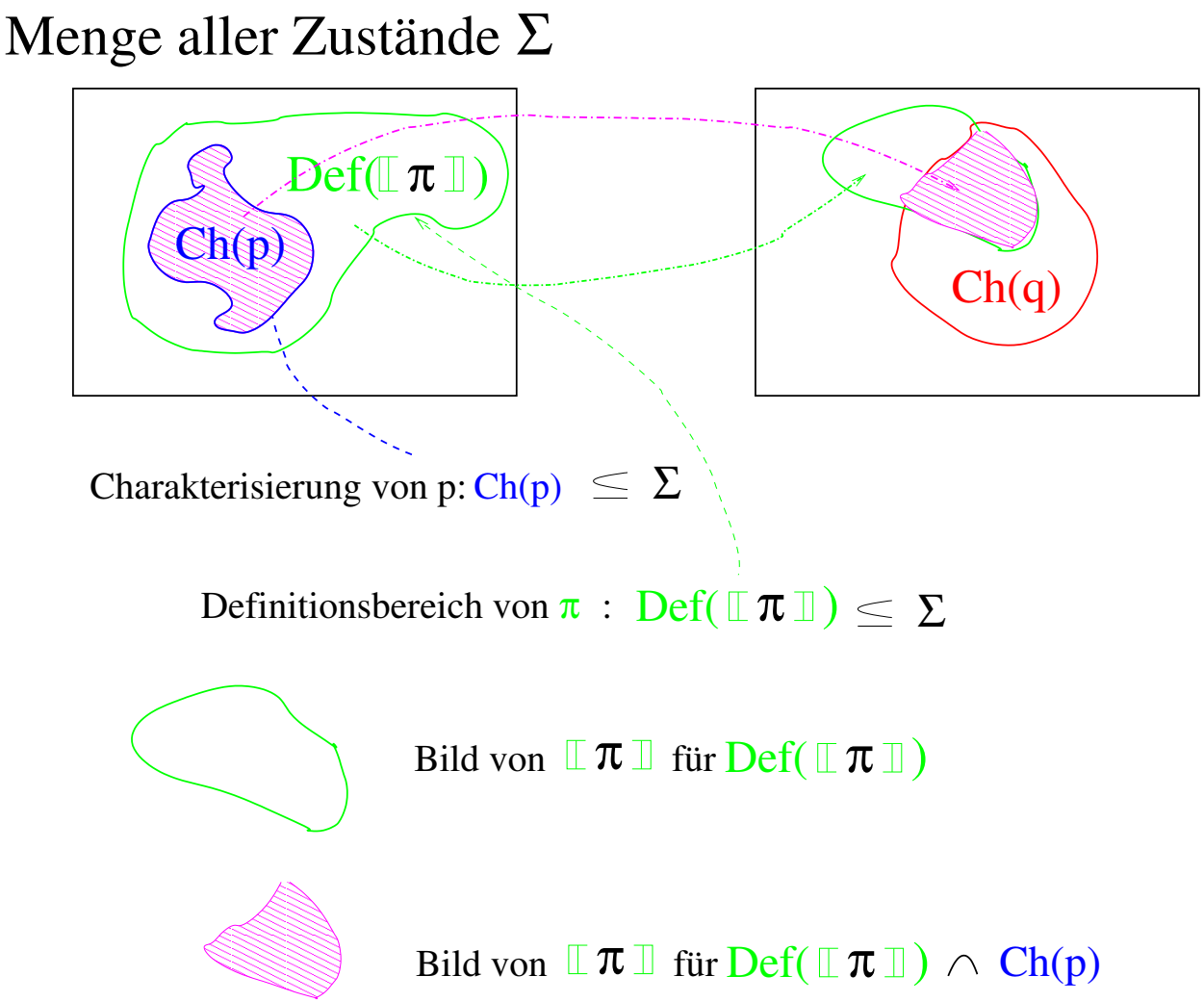
Veranschaulichung 2(3)

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



Veranschaulichung 3(3)

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



Stärkste Nachbedingungen, schwächste Vorbedingungen

In der Folge:

Präzisierung von

- Stärkste Nachbedingungen
- Schwächste Vorbedingungen

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 (4.1.1)

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

Partielle vs. totale Korrektheit

Lemma (4.1.2)

Für deterministische Programme π gilt:

$$[p]\ \pi\ [q] \quad \succ \quad \{p\}\ \pi\ \{q\}$$

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- und stärkste Nachbedingungen

...noch einmal anders betrachtet:

Definition (4.1.3)

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

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

Schwächste Vorbedingungen

Definition (4.1.4)

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.

- $wlp(\pi, q)$ 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.

Stärkste Nachbedingungen 1(3)

Analog zu A ist schwächer als B lässt sich definieren:

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

Zum Überlegen:
Ist es sinnvoll, den Begriff der stärksten (liberalen) Nachbedingung $spo(p, \pi)$ bzw. $slpo(p, \pi)$ "in genau gleicher Weise" zum Begriff der schwächsten (liberalen) Vorbedingung $wp(\pi, q)$ bzw. $wlp(\pi, q)$ zu gegebenem Programm π und Vorbedingung p zu betrachten?

Stärkste Nachbedingungen 2(3)

Betrachte:

Definition (sversuch)

Sei π ein Programm und p eine Formel.
Dann heißt

- $spo(p, \pi)$ stärkste Nachbedingung für totale Korrektheit von π bezüglich (der Vorbedingung) p, wenn

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

  total korrekt ist und $spo(p, \pi)$ die stärkste Formel mit dieser Eigenschaft ist.
- $slpo(p, \pi)$ stärkste liberale Nachbedingung für partielle Korrektheit von π bezüglich (der Vorbedingung) p, wenn

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

Stärkste Nachbedingungen 3(3)

Fragen:

- Gibt es Programme π und Formeln p derart, dass
  - $spo(p, \pi)$
  - $slpo(p, \pi)$

  unterscheidbar, d.h. logisch nicht äquivalent sind?
- Wie passen die hier betrachteten Begriffe von schwächsten Vor- und stärksten Nachbedingungen mit den zuvor in diesem Kapitel betrachteten zusammen?

Kapitel 4.2

Beweiskalkül für partielle Korrektheit

Hoare-Kalkül HK_{PK} für partielle Korrektheit

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

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

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

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-Kalkül HK_{TK} für totale Korrektheit

...identisch mit HK_{PK}, wobei aber Regel [while] ersetzt ist durch:

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

wobei

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

$\rightsquigarrow$ Terminationsordnung!

Zur Vollständigkeit

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

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

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

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

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

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

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)

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)

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 $(\mathbb{N}, >)$

Wohlfundierte Ordnungen (3)

Konstruktionsprinzipien für wohlfundierte 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*

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

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

$$(m_1, m_2) <_{lex} (n_1, n_2) \quad gdw.$$

$$(m_1 <_1 n_1) \ \vee \ (m_1 = n_1 \ \wedge \ m_2 <_2 n_2)$$

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)

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

Intuitiv:

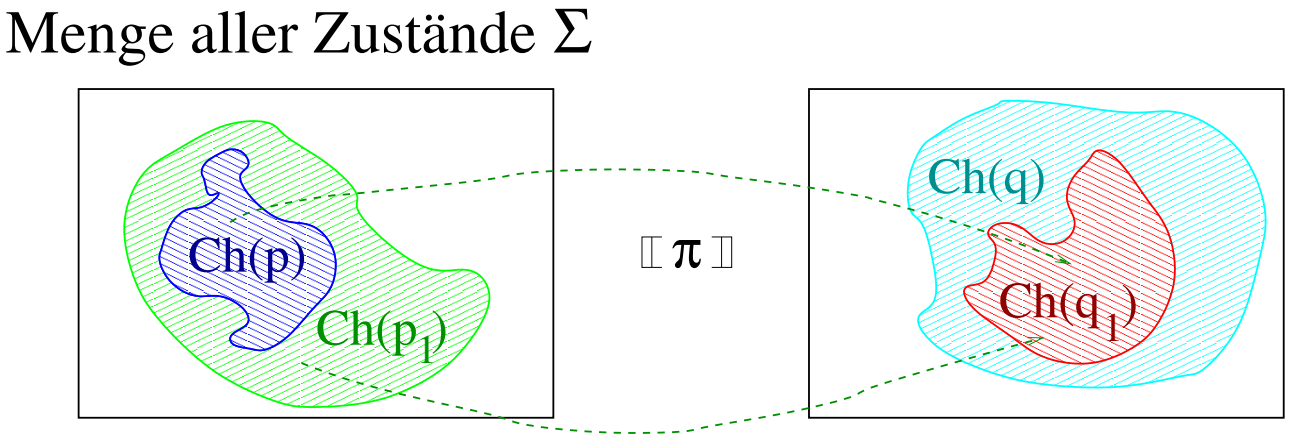
Die Konsequenzregel

- 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
    $p \succ p_1$ ($\Leftrightarrow\ Ch(p) \subseteq Ch(p_1)$)
  - Nachbedingungen abzuschwächen
    (Übergang von q_1 zu q möglich, falls
    $q_1 \succ q$ ($\Leftrightarrow\ Ch(q_1) \subseteq Ch(q)$)

  um so die Anwendung anderer Beweisregeln zu ermöglichen.

Zur Konsequenzregel 2(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:

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

wobei

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

$\rightsquigarrow$ Terminationsordnung!

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

- Prämisse 1: $I \wedge b \succ u[t/v]$
  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

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

- Programmvariablen

  ...Variablen, die in π vorkommen

- logischen Variablen medskip

  ...Variablen, die in π nicht vorkommen

Logische Variablen erlauben

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

Beispiel:

- $\{x = n\}\ y := 1;$ while $x \neq 1$ do $y := y * x; x := x - 1$ 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.

HK_{TK} versus HK_{PK}

Beachte:

HK_{TK} und HK_{PK} sind bis auf die Schleifenregel identisch:

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

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

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

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

Nachtrag zur totalen Korrektheit (1)

Oft, insbesondere für die von uns betrachteten Beispiele, reicht folgende, weniger allgemeine Regel für while-Schleifen, um Terminierung und insgesamt totale Korrektheit zu zeigen.

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

wobei

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

Beachte: Statt beliebiger Terminationsordnungen hier Festlegung auf eine spezielle Noethersche Ordnung als Terminationsordnung, nämlich $(\mathbb{N}, <)$.

Nachtrag zur totalen Korrektheit (2)

Beweistechnische Anmerkung:

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

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

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. Der Gesamtbeweis wird dadurch modular. Oft gilt, dass der Terminationsnachweis einfach ist.

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

Kapitel 4.4

Korrektheit und Vollständigkeit

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

Sei K ein Kalkül für partielle bzw. totale Korrektheit

Zentral sind dann die Fragen der:

- Korrektheit: Ist jede mithilfe von K ableitbare Korrektheitsformel partiell bzw. total korrekt?
- Vollständigkeit: Ist jede partiell bzw. total korrekte Korrektheitsformel mithilfe von K ableitbar?

Speziell:

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

Zur Korrektheit und Vollständigkeit Hoarescher Beweiskalküle

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, dann ist sie auch semantisch gültig. In Zeichen:

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

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

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

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

Theorem (4.4.1 – Korrektheit von HK_{PK}&HK_{TK})

1. *HK_{PK} ist korrekt, d.h. jede mit HK_{PK} ableitbare Korrektheitsformel ist gültig im Sinne partieller Korrektheit:*

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

2. *HK_{TK} ist korrekt, d.h. jede mit HK_{TK} ableitbare Korrektheitsformel ist gültig im Sinne totaler Korrektheit:*

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

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.

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

Für den extensionalen Ansatz gilt:

Theorem (4.4.2 – 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:*

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

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

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

Beweis durch strukturelle Induktion über den Aufbau von π.

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

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

Intuition

- 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

Beweis partieller Korrektheit: Zwei Beispiele

Die beiden Beispiele im Überblick (1)

...Beweis partieller Korrektheit von Hoareschen 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 Zusicherungen

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

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.

In der Folge geben wir die Beweise dafür in baumartiger Notation an.

Bew. part. Korrektheit: Fakultät (1)

Erster Beweis

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

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

wobei

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

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

& : Logisches und

~ : Logisches nicht

Bew. part. Korrektheit: Fakultät (2)

Zweiter Beweis

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

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

$$\dfrac{\dfrac{\dfrac{}{\{a>0\}\ x:=a;\ y:=1;\ \text{WHILE } x>1 \text{ DO } y:=y*x;\ x:=x-1 \text{ OD } \{y=a!\}}\ \textbf{Comp}}{\{a>0\}\ x:=a;\ y:=1;\ \text{WHILE } x>1 \text{ DO } y:=y*x;\ x:=x-1 \text{ OD } \{y=a!\}}}$$

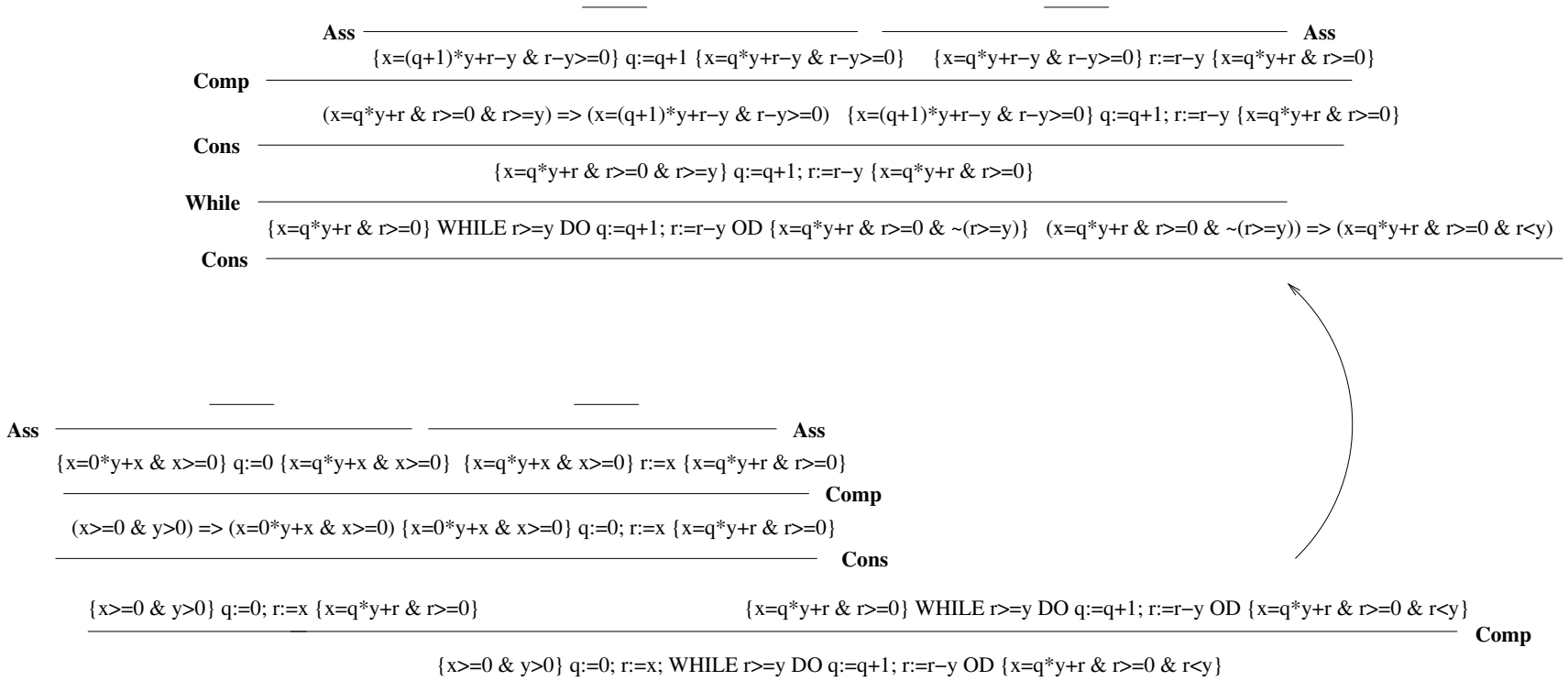
wobei

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

& : Logisches und

~ : Logisches nicht

Bew. partieller Korrektheit: Division



& : 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. linearen Beweisskizze.

Lin. Beweisskizze f. Fakultätsbsp. (1)

Beweise, dass das Hoare-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!

Lin. Beweisskizze f. Fakultätsbsp. (2)

Schritt 1

"Träumen" der Invariante:

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

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

Lin. Beweisskizze f. Fakultätsbsp. (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:

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

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

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

$$y := y * x;$$

$$x := x - 1;$$

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (4)

Behandlung des Rumpfs der while-Schleife im Detail:

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

$$y := y * x;$$

$$x := x - 1;$$

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

Lin. Beweisskizze f. Fakultätsbsp. (5)

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

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

$$y := y * x;$$

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (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!

Lin. Beweisskizze f. Fakultätsbsp. (7)

Schluss der "Beweislücke" in der zugrundeliegenden Theorie:

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

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

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

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

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (8)

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

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

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

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

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

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

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

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

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

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (9)

Schritt 3 Zur gewünschten Nachbedingung verbleibt offenbar ebenfalls eine Beweislücke:

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

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

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

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

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

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

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

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

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

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (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]

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

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

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

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

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (11)

Aus Platzgründen etwas verkürzt dargestellt:

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

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

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

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

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

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

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

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

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

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

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (12)

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

$$\{a > 0\}$$

$$x := a;$$

$$y := 1;$$

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

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

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

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

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

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

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

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

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

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

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (13)

Einmalige Anwendung der [ass]-Regel liefert:

$$\{a > 0\}$$

$$x := a;$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (14)

Abermalige Anwendung der [ass]-Regel liefert:

$$\{a > 0\}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Lin. Beweisskizze f. Fakultätsbsp. (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)

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

Lin. Beweisskizze f. Fakultätsbsp. (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.

Kapitel 4.6

Beweis totaler Korrektheit: Ein Beispiel

Das Beispiel im Überblick

Beweise, dass das Hoare-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: $I \wedge b \succ \sigma(x) \in M$ 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:

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

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 wieder in Form einer linearen Beweisskizze präsentiert...

Bew. totaler Korrektheit: Fakultät (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]

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

Bew. totaler Korrektheit: Fakultät (2)

Behandlung des Rumpfs der while-Schleife im Detail:

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

$$y := y * x;$$

$$x := x - 1;$$

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

Bew. totaler Korrektheit: Fakultät (3)

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

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

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

Bew. totaler Korrektheit: Fakultät (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!

Bew. totaler Korrektheit: Fakultät (5)

Schluss der "Beweislücke" in der zugrundeliegenden Theorie:

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

Bew. totaler Korrektheit: Fakultät (6)

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

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

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

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

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

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

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

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

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

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

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

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

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

Bew. totaler Korrektheit: Fakultät (7)

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

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

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

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

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

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

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

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

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

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

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

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

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

Bew. totaler Korrektheit: Fakultät (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]

$$[y = a!]$$

Bew. totaler Korrektheit: Fakultät (9)

Aus Platzgründen etwas verkürzt dargestellt:

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

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

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

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

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

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

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

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

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

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

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

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

$$[y = a!]$$

Bew. totaler Korrektheit: Fakultät (10)

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

$$[a > 0]$$

$$x := a;$$

$$y := 1;$$

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

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

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

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

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

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

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

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

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

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

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

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

$$[y = a!]$$

Bew. totaler Korrektheit: Fakultät (11)

Einmalige Anwendung der [ass]-Regel liefert:

$$[a > 0]$$

$$x := a;$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$[y = a!]$$

Bew. totaler Korrektheit: Fakultät (12)

Abermalige Anwendung der [ass]-Regel liefert:

$$[a > 0]$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$[y = a!]$$

Bew. totaler Korrektheit: Fakultät (13)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

$$[a > 0]$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$[y = a!]$$

Insgesamt (14)

$$[a > 0]$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$[y = a!]$$

Bew. totaler Korrektheit: Fakultät (15)

Damit haben wir wie gewünscht insgesamt gezeigt:

Die Hoaresche Zusicherung

$$[a > 0]$$

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

$$[y = a!]$$

ist gültig im Sinne totaler Korrektheit.

Kapitel 4.7

Ergänzungen und Ausblick

Linearer vs. baumartiger Beweisstil

Vorteil linearen gegenüber baumartigen Beweisnotationsstils:

- wenig Redundanz
- daher insgesamt knappere Beweise

Sprechweisen im Zshg. mit Hoare-Tripeln (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) und
- π ein Programm ist.

Sprechweisen im Zshg. mit Hoare-Tripeln (2)

In einer Hoareschen Zusicherung von einer der Formen

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

heißen

- p und q Vor- bzw. Nachbedingung.

Sprechweisen im Zshg. mit Hoare-Tripeln (3)

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.

Sprechweisen im Zshg. mit Hoare-Tripeln (4)

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 $y = a!$

Sprechweisen im Zshg. mit Hoare-Tripeln (5)

Die Wortwahl

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

betont jeweils die

- syntaktische bzw.
- semantische Sicht

auf

- $\{p\}\ \pi\ \{q\}$ bzw. $[p]\ \pi\ [q]$

Automatische Ansätze z. Prog.verifikation (1)

...Theorema-Projekt am RISC, Linz: `www.theorema.org`

"The Theorema project aims at extending current computer algebra systems by facilities for supporting mathematical proving. The present early-prototype version of the Theorema software system is implemented in Mathematica. The system consists of a general higher-order predicate logic prover and a collection of special provers that call each other depending on the particular proof situations. The individual provers imitate the proof style of human mathematicians and produce human-readable proofs in natural language presented in nested cells. The special provers are intimately connected with the functors that build up the various mathematical domains.

Automatische Ansätze z. Prog.verifikation (2)

The long-term goal of the project is to produce a complete system which supports the mathematician in creating interactive textbooks, 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."
[...] (Zitat von `http://www.theorema.org`)

Ergänzende, weiterführende und vertiefende 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, Ernst-Rüdiger Olderog. *Programm-verifikation – Sequentielle, parallele und verteilte Programme*. Springer-V., 1994.

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)

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 4 (2)

Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 2nd edition, Springer-V., 2001. (Chapter 9, Programs: Semantics and Verification)

R.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.

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

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 4 (3)

Laura Kovács and 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), Timisoara, Romania, October 1-4, 2003.
`apache.risc.uni-linz.ac.at/internals/ActivityDB/publications/download/risc_464/synasc03.pdf`

Laura Kovács and Tudor Jebelean. *Generation of Invariants in Theorema*. In Proceedings of the 10th International Symphosium of Mathematics and its Applications, Timisoara, Romania, November 6-9, 2003.
`www.theorema.org/publication/2003/Laura/Poli_Timisoara_nov.pdf`

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 4 (4)

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

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

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

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 4 (5)

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)

Ernst-Rüdiger Olderog, Bernhard Steffen. *Formale Semantik und Programmverifikation*. In Informatik-Handbuch, P. Rechenberg, G. Pomberger (Hrsg.), Carl Hanser Verlag, 129 - 148, 1997.

Kapitel 5

Worst-Case Execution Time Analyse

In der Folge

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(2)

...zur Zuordnung von Ausführungszeiten:

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

Die grundlegende Idee 2(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 sind offenbar möglich.

Formalisierung

...dieser grundlegenden Idee in 3 Schritten:

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

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)

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.

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)

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

Zweiter Schritt

Erweiterung und Adaption der

- natürlichen Semantik von WHILE

zur Bestimmung der Ausführungszeit von Programmen.

Idee

Übergang 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.

Natürliche Semantik erweitert um den Ausführungszeitaspekt (1)

...für das Beispiel von WHILE:

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

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

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

Natürliche Semantik erweitert um den Ausführungszeitaspekt (2)

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

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

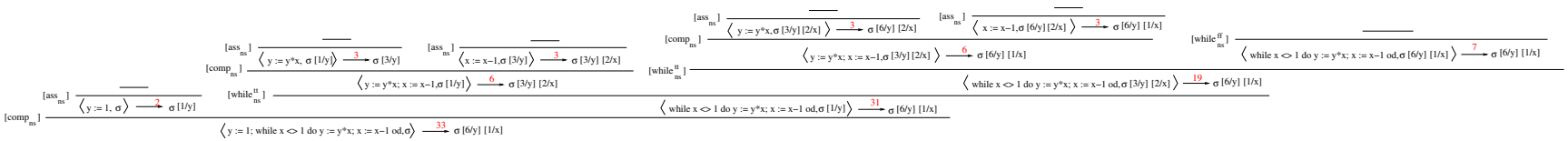
$$[\text{while}^{tt}_{tns}] \quad \frac{\langle \pi, \sigma \rangle \rightarrow^t \sigma', \langle \text{while } b \text{ do } \pi \text{ od}, \sigma' \rangle \rightarrow^{t'} \sigma''}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow^{[\![b]\!]_{TB}+t+t'+2} \sigma''} \quad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

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

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

Sei $\sigma \in \Sigma$ mit $\sigma(x) = 3$.
Dann gilt:

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



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

Das gleiche Beispiel in etwas gefälligerer Darstellung:

$$\dfrac{\dfrac{}{\langle y := 1,\ \sigma\rangle \xrightarrow{2} \sigma\,[1/y]}\,[\mathrm{ass}_{ns}] \qquad \dfrac{\dfrac{}{\langle y := y*x,\ \sigma\,[1/y]\rangle \xrightarrow{3} \sigma\,[3/y]}\,[\mathrm{ass}_{ns}] \qquad \dfrac{}{\langle x := x-1, \sigma\,[3/y]\rangle \xrightarrow{3} \sigma\,[3/y]\,[2/x]}\,[\mathrm{ass}_{ns}]}{\langle y := y*x;\ x := x-1, \sigma\,[1/y]\rangle \xrightarrow{6} \sigma\,[3/y]\,[2/x]}\,[\mathrm{comp}_{ns}] \qquad \dfrac{}{\langle \text{while } x <> 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\,[1/y]\rangle \xrightarrow{31} \sigma\,[6/y]\,[1/x]}\,[\mathrm{while}^{tt}_{ns}]}{\langle y := 1;\ \text{while } x <> 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\rangle \xrightarrow{33} \sigma\,[6/y]\,[1/x]}\,[\mathrm{comp}_{ns}]$$

T^{19}

$$\mathrm{T} \equiv \dfrac{\dfrac{\dfrac{}{\langle y := y*x, \sigma\,[3/y]\,[2/x]\rangle \xrightarrow{3} \sigma\,[6/y]\,[2/x]}\,[\mathrm{ass}_{ns}] \qquad \dfrac{}{\langle x := x-1, \sigma\,[6/y]\,[2/x]\rangle \xrightarrow{3} \sigma\,[6/y]\,[1/x]}\,[\mathrm{ass}_{ns}]}{\langle y := y*x;\ x := x-1, \sigma\,[3/y]\,[2/x]\rangle \xrightarrow{6} \sigma\,[6/y]\,[1/x]}\,[\mathrm{comp}_{ns}] \qquad \dfrac{}{\langle \text{while } x <> 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\,[6/y]\,[1/x]\rangle \xrightarrow{7} \sigma\,[6/y]\,[1/x]}\,[\mathrm{while}^{ff}_{ns}]}{\langle \text{while } x <> 1 \text{ do } y := y*x;\ x := x-1 \text{ od}, \sigma\,[3/y]\,[2/x]\rangle \xrightarrow{19} \sigma\,[6/y]\,[1/x]}\,[\mathrm{while}^{tt}_{ns}]$$

Dritter Schritt

Erweiterung und Adaption 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 Prädikate (wie bisher!) und
- $e \in \mathbf{Aexp}$ ein arithmetischer Ausdruck ist.

Idee (2)

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 von $\pi_1; \pi_2$ 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 zum Ausführungszeitaspekt (1)

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

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

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

wobei u frische logische Variable ist

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

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

wobei (für eine natürliche Zahl k) $p \succ p' \wedge e' \leq k * e$ und $q' \succ q$

Axiomatische Semantik zum Ausführungszeitaspekt (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 $p(z+1) \succ b \wedge e \geq e_1 + e'$, $p(0) \succ \neg\, b \wedge 1 \leq e$
u eine frische logische Variable ist und
z Werte aus den natürlichen Zahlen annimmt (d.h. $z \geq 0$)

Beispiele (1)

Die Korrektheitsformel

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

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

Beispiele (2)

Die Korrektheitsformel

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

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

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 5 (1)

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, 2011.

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 6.5, Assertions for Execution Time)

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

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 5 (2)

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):36.1-36.53, 2008.

Teil II

Analyse

Kapitel 6

Programmanalyse

Programmanalyse

...speziell Datenflussanalyse

Typische Fragen sind:

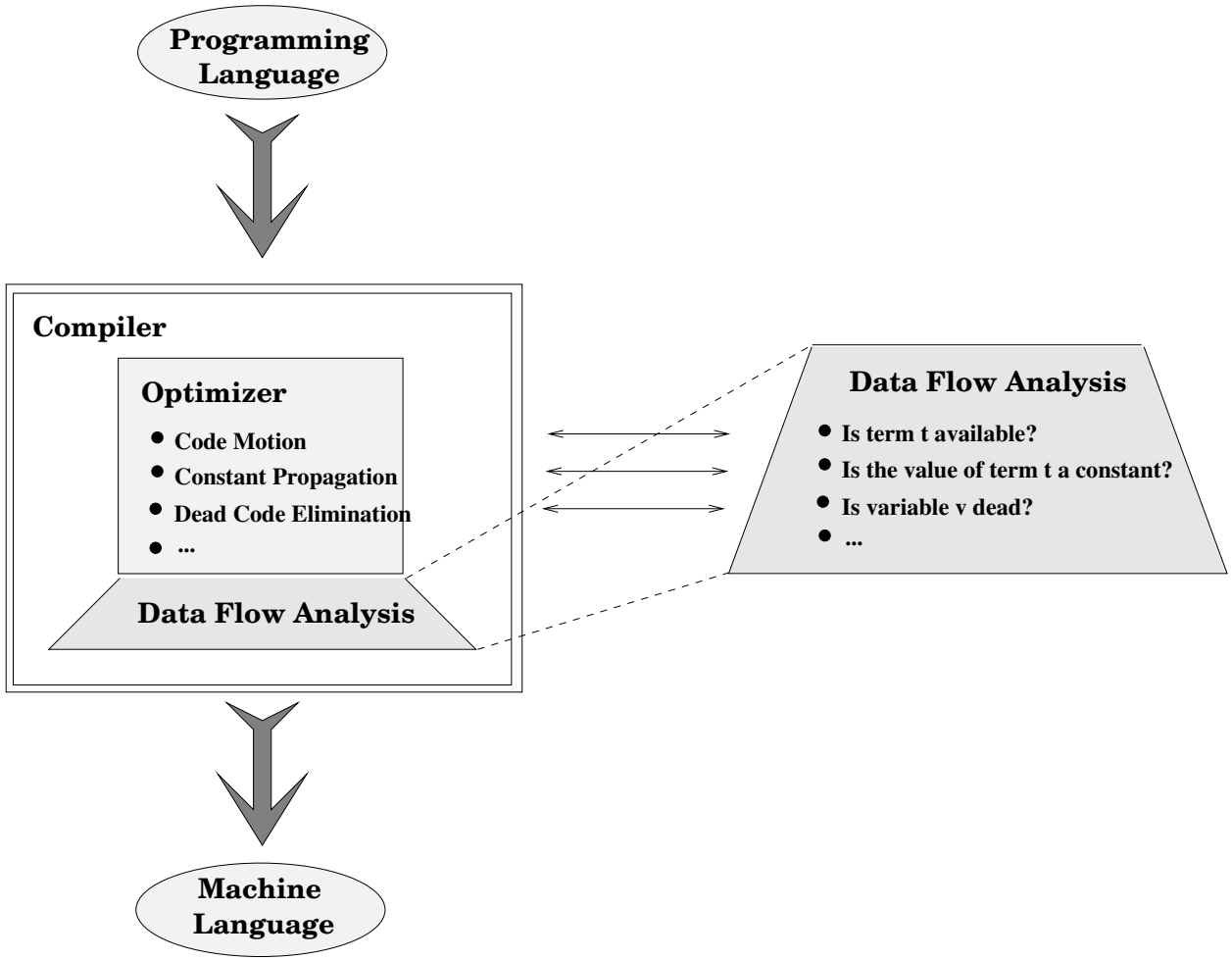
- Welchen Wert hat eine Variable an einer Programmstelle?
  $\rightsquigarrow$ Konstantenausbreitung und Faltung
- Steht der Wert eines Ausdrucks an einer Programmstelle verfügbar?
  $\rightsquigarrow$ (Partielle) Redundanzelimination
- Ist eine Variable tot an einer Programmstelle?
  $\rightsquigarrow$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

Kapitel 6.1

Motivation

Motivation

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



Zentrale Fragen

Grundlegendes:

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

Wie (scheinbar) Nebensächliches:

- Was ist eine angemessene Programmrepräsentation?

Ausblick

Genauer werden wir unterscheiden:

- Intraprozedurale,
- interprozedurale,
- parallele,
- ...

Datenflussanalyse (DFA).

Ausblick (fgs.)

Ingredienzien (intraprozeduraler) Datenflussanalyse:

- (Lokale) abstrakte Semantik
  1. Ein Datenflussanalyseverband $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$
  2. Ein Datenflussanalysefunktional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$
  3. Anfangsinformation/-zusicherung $c_{\mathbf{s}} \in \mathcal{C}$
- Globalisierungsstrategien
  1. "Meet over all Paths"-Ansatz (*MOP*)
  2. Maximaler Fixpunktansatz (*MaxFP*)
- Generischer Fixpunktalgorithmus

Theorie intraprozeduraler DFA

Hauptresultate:

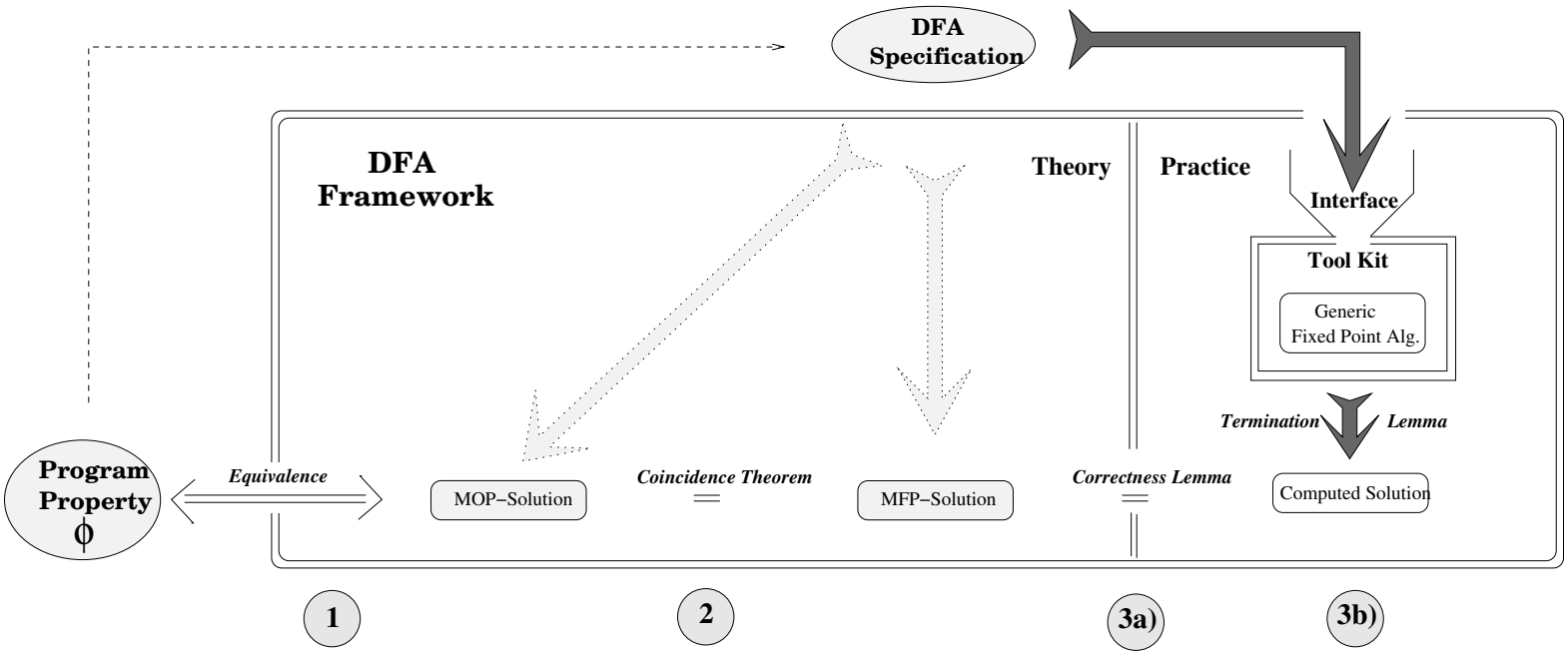
- Sicherheits- (Korrektheits-) Theorem
- Koinzidenz- (Vollständigkeits-) Theorem

Sowie:

- Effektivitäts- (Terminierungs-) Theorem

Praxis intraprozeduraler DFA

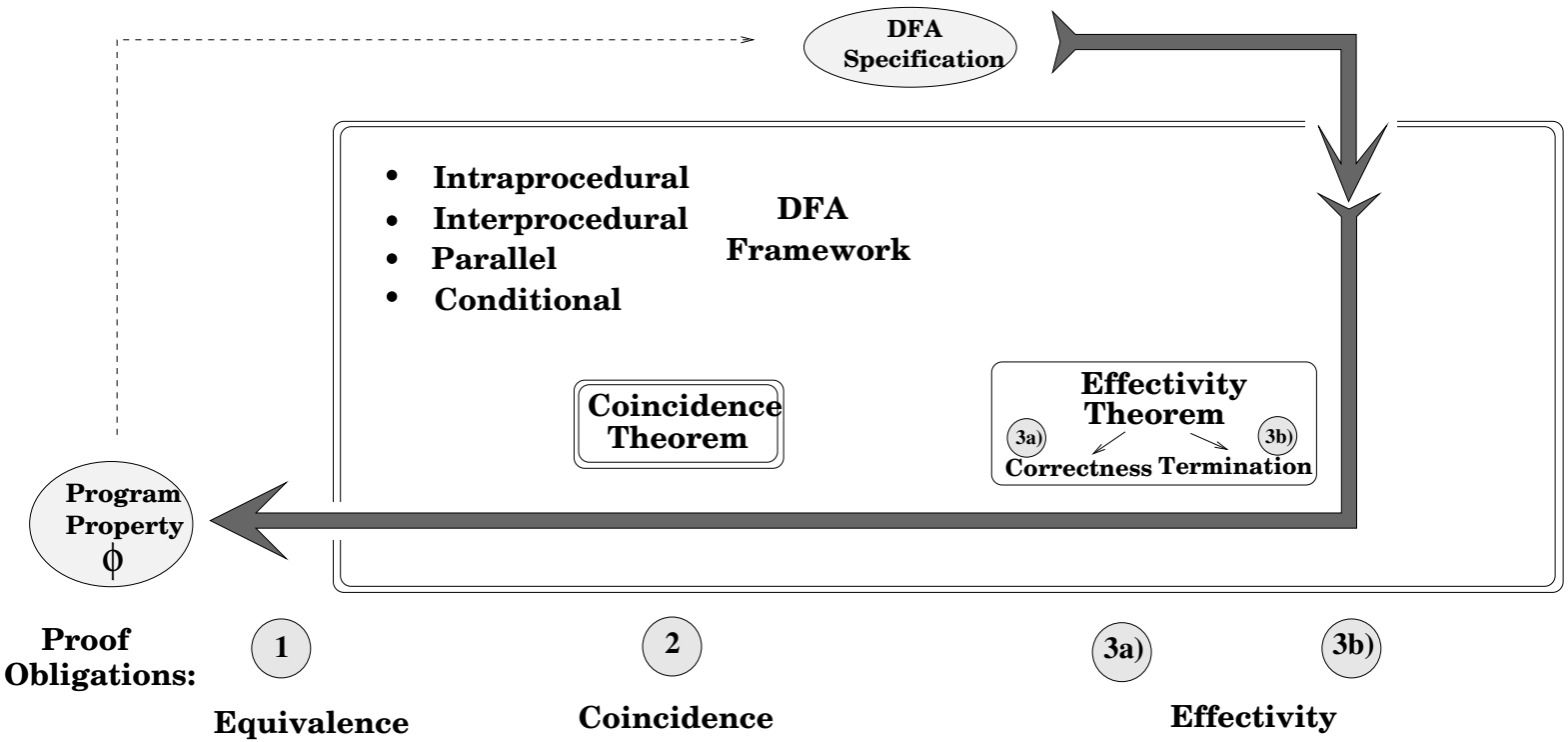
Die intraprozedurale DFA-Framework / DFA-Toolkit - Sicht:



Praxis DFA

Die “intraprozedurale” Einschränkung kann fallen.

Die DFA-Framework / DFA-Toolkit - Sicht gilt allgemein:



Ziel

Optimale Programmoptimierung

...ein weißer Schimmel in der Informatik?

Ohne Fleiß kein Preis!

In der Sprechweise der optimierenden 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

Flussgraphen

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

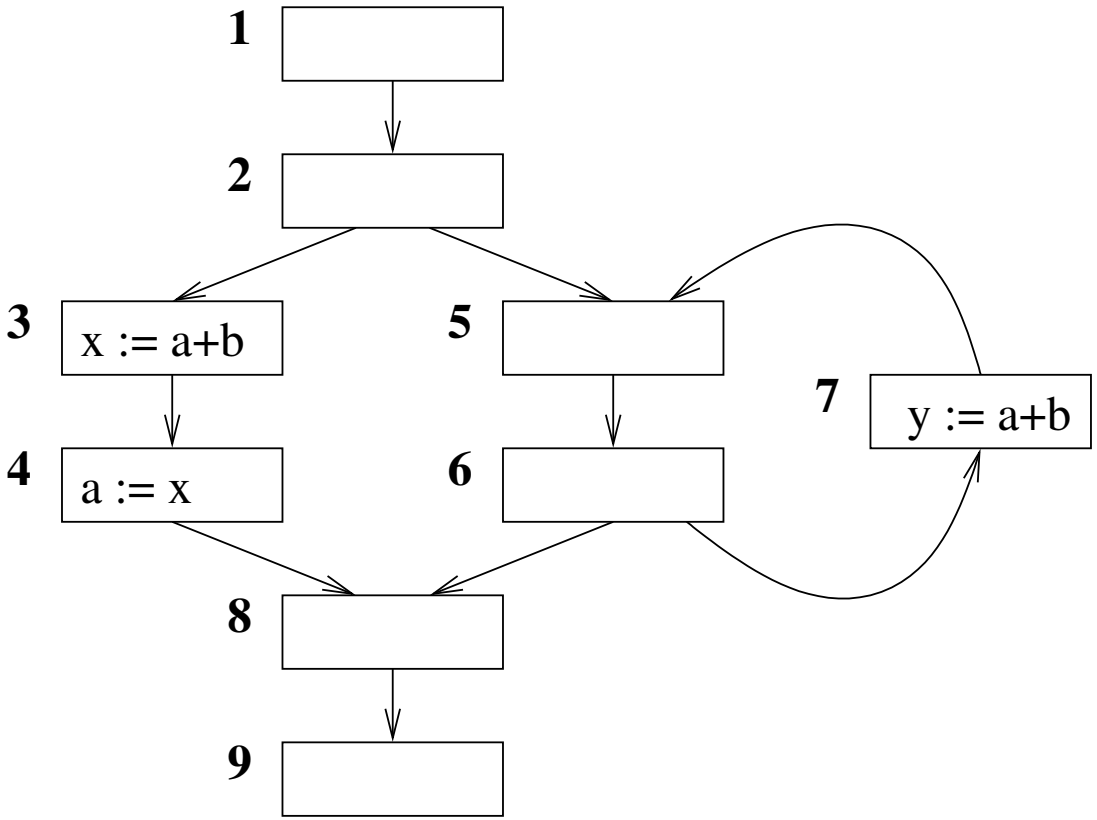
- Knotenmenge (engl. Nodes) N
- Kantenmenge (engl. Edges) $E \subseteq N \times N$
- ausgezeichnetem Startknoten s ohne Vorgänger und
- ausgezeichnetem Endknoten e ohne Nachfolger

Knoten repräsentieren Programmpunkte, Kanten die Verzweigungsstruktur. Elementare Programmanweisungen (Zuweisungen, Tests) können wahlweise durch

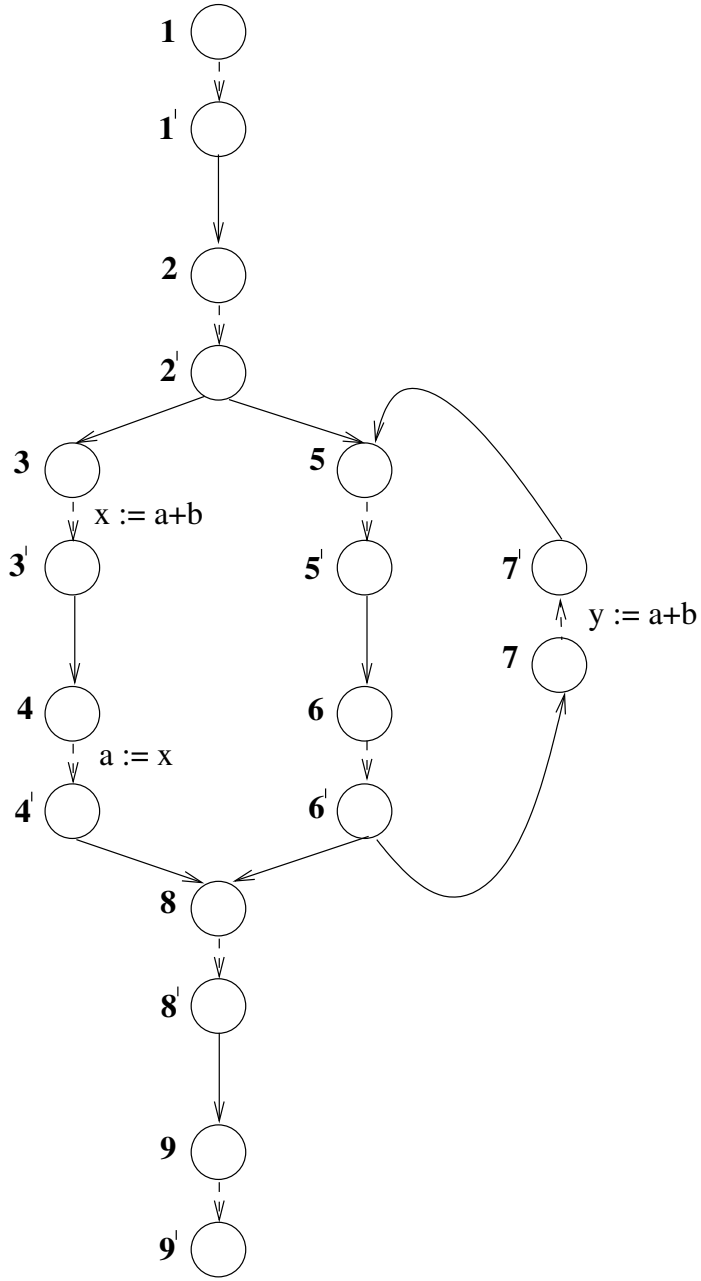
- Knoten ($\rightsquigarrow$ knotenbenannter Flussgraph)
- Kanten ($\rightsquigarrow$ kantenbenannter Flussgraph)

repräsentiert werden.

Bsp.: Knotenbenannter Flussgraph



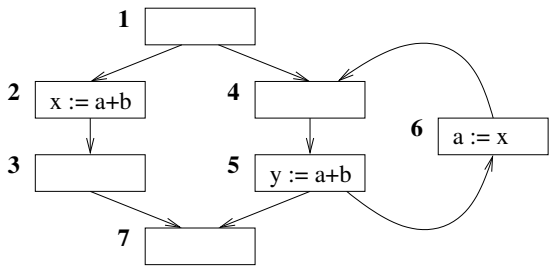
Bsp.: Kantenbenannter Flussgraph



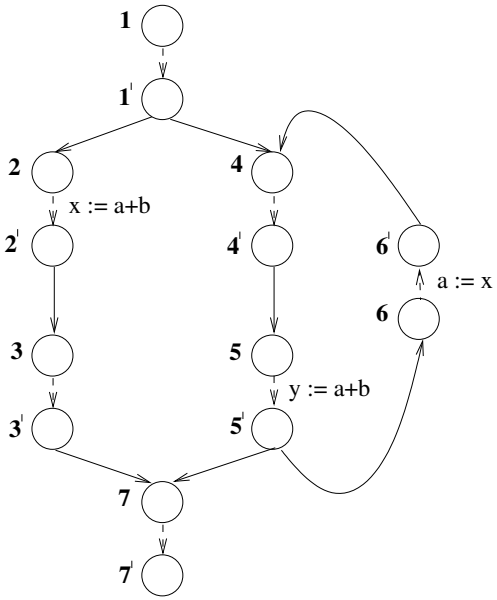
Flussgraphdarstellungsvarianten I

Knoten- vs. kantenbenannte Flussgraphen (hier mit Einzelanweisungsbenennung)

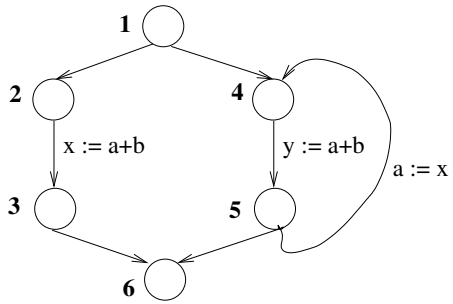
a)



b)



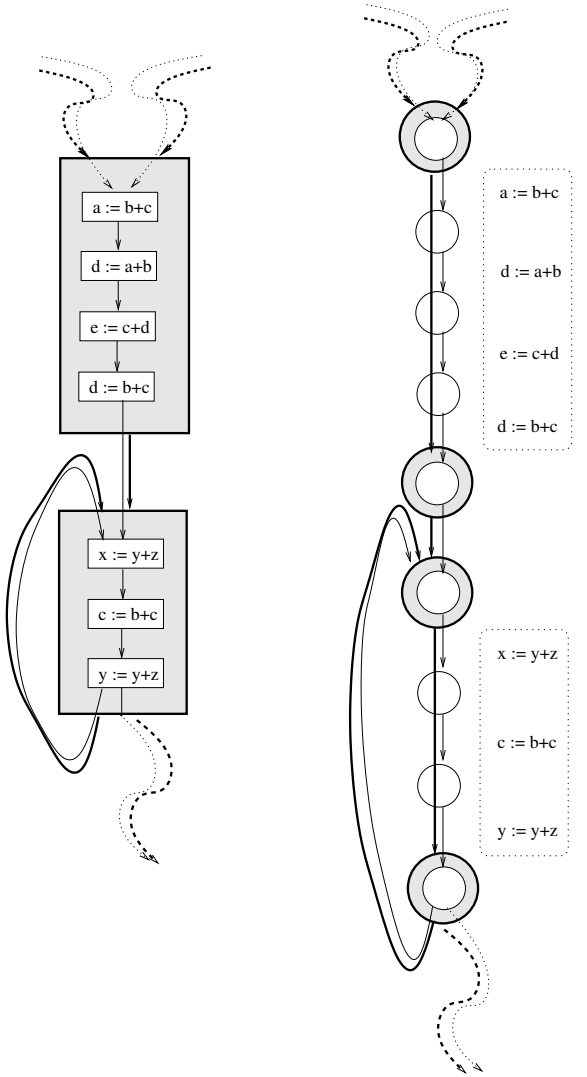
i) Schematisch



ii) "Optimiert"

Flussgraphdarstellungsvarianten II

Knoten- vs. kantenbenannte Flussgraphen
(hier mit Basisblockbenennung)



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

Flussgraphdarstellungsvarianten III

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

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

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

Datenflussanalysespezifikation

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

Globalisierung einer lokalen abstrakten Semantik

Zwei Strategien:

- "Meet over all Paths"-Ansatz (*MOP*)
  $\leadsto$ liefert spezifizierende Lösung
- Maximaler Fixpunktansatz (*MaxFP*)
  $\leadsto$ führt auf berechenbare Lösung

Kapitel 6.3

MOP-Ansatz

Der *MOP*-Ansatz

Zentral:

Ausdehnung der lokalen abstrakten Semantik auf Pfade

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

wobei $Id_{\mathcal{C}}$ die Identität auf $\mathcal{C}$ bezeichnet.

Die *MOP*-Lösung

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ MOP_{c_{\mathbf{s}}}(n) = \bigsqcap \{\, [\![\, p \,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\}$$

Die *MOP*-Lösung: das Maß aller Dinge, die spezifizierende Lösung eines durch $\mathcal{C}$, $[\![\ \ ]\!]$ und $c_{\mathbf{s}}$ gegebenen intraprozeduralen DFA-Problems.

Wermutstropfen

Die Unentscheidbarkeit der *MOP*-Lösung im allgemeinen:

Theorem (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 $\mathcal{C}$, $[\![\ ]\!]$ und $c_{\mathbf{s}}$ gegebene Instanz I dieses Rahmens*
2. *Ausgabe von A ist die MOP-Lösung von I.*

Deshalb betrachten wir jetzt eine zweite Globalisierungsstrategie.

Kapitel 6.4

MaxFP-Ansatz

Der *MaxFP*-Ansatz

Zentral:

Das *MaxFP*-Gleichungssystem

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

Die *MaxFP*-Lösung

$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket, c_s)}(n) =_{df} inf^{*}_{c_s}(n)$

wobei $inf^{*}_{c_s}$ die größte Lösung des *MaxFP* -Gleichungssystems bezüglich $\llbracket\ \rrbracket$ und c_s bezeichnet.

Die *MaxFP*-Lösung: die (unter geeigneten Voraussetzungen) berechenbare Lösung eines durch $\mathcal{C}$, $\llbracket\ \rrbracket$ und c_s gegebenen intraprozeduralen DFA-Problems.

Generischer Fixpunktalgorithmus (1)

Eingabe: (1) Ein Flussgraph $G = (N, E, \mathbf{s}, \mathbf{e})$, (2) eine (lokale) abstrakte Semantik bestehend aus einem Datenflussanalyseverband $\mathcal{C}$, einem Datenflussanalysefunktional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$, und (3) einer Anfangsinformation $c_{\mathbf{s}} \in \mathcal{C}$.

Ausgabe: Unter den Voraussetzungen des Effektivitätstheorems (s. Kap. 6.5) die *MaxFP*-solution. Abhängig von den Eigenschaften des Datenflussanalysefunktionals gilt dann:
(1) $[\![\ ]\!]$ ist distributiv: Variable *inf* enthält für jeden Knoten die stärkste Nachbedingung bezüglich der Anfangsinformation $c_{\mathbf{s}}$.
(2) $[\![\ ]\!]$ 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 (2)

Bemerkung: Die Variable *workset* steuert den iterativen Prozess. Ihre Elemente sind Knoten aus G, deren Annotation jüngst aktualisiert worden ist.

Generischer Fixpunktalgorithmus (3)

(Prolog: Initialisierung von *inf* and *workset*)
FORALL $n \in N \backslash \{\mathbf{s}\}$ DO $inf[n] := \top$ OD;
$inf[\mathbf{s}] := c_{\mathbf{s}}$;
$workset := \{\, \mathbf{s} \,\}$;
(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.

Zu ergänzende Definitionen

...im Zshg. mit dem generischen Fixpunktalgorithmus:

- Absteigende (aufsteigende) Kettenbedingung
- Monotonie und Distributivität von
  - lokalen abstrakten Semantikfunktionen
  - Datenflussanalysefunktionalen

Auf-/absteigende Kettenbedingung

Definition (Ab-/aufsteigende Kettenbedingung)

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

1. die absteigende Kettenbedingung, falls jede absteigende Kette stationär wird, d.h. für jede Kette $p_1 \sqsupseteq p_2 \sqsupseteq \ldots \sqsupseteq p_n \sqsupseteq \ldots$ gibt es einen Index $m \geq 1$ so dass $x_m = x_{m+j}$ für alle $j \in \mathbb{N}$ gilt
2. die aufsteigende Kettenbedingung, falls jede aufsteigende Kette stationär wird, d.h. für jede Kette $p_1 \sqsubseteq p_2 \sqsubseteq \ldots \sqsubseteq p_n \sqsubseteq \ldots$ gibt es einen Index $m \geq 1$ so dass $x_m = x_{m+j}$ für alle $j \in \mathbb{N}$ gilt

Monotonie, Distributivität, Additivität

...von Funktionen auf (Datenflussanalyse-) Verbänden.

Definition (Monotonie, Distributivität, Additivität)

Sei $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ ein vollständiger 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)

Oft nützlich

...ist folgende äquivalente Charakterisierung der Monotonie:

Lemma

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

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

Monotonie und Distributivität

...von Datenflussanalysefunktionalen.

Definition

Ein Datenflussanalysefunktional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ heißt monoton (distributiv) gdw $\forall\, e \in E.\ [\![\ e\ ]\!]$ ist monoton (distributiv).

Kapitel 6.5

Koinzidenz- und Sicherheitstheorem

Hauptresultate: Korrektheit, Vollständigkeit, Effektivität/Terminierung

Zusammenhang von:

- *MOP* - und *MaxFP* -Lösung
  - Korrektheit
  - Vollständigkeit
- *MaxFP* -Lösung und generischem Algorithmus
  - Terminierung mit *MaxFP* -Lösung

Korrektheit

Theorem (6.5.1, Sicherheit)

Die MaxFP-Lösung ist eine sichere (konservative), d.h. untere Approximation der MOP-Lösung, d.h.,

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{c_s}(n) \sqsubseteq MOP_{c_s}(n)$$

falls das Datenflussanalysefunktional $[\![\ ]\!]$ *monoton ist.*

Vollständigkeit (und Korrektheit)

Theorem (6.5.2, Koinzidenz)

Die MaxFP-Lösung stimmt mit der MOP-Lösung überein, d.h.,

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{c_{\mathbf{s}}}(n) = MOP_{c_{\mathbf{s}}}(n)$$

falls das Datenflussanalysefunktional $[\![\ ]\!]$ *distributiv ist.*

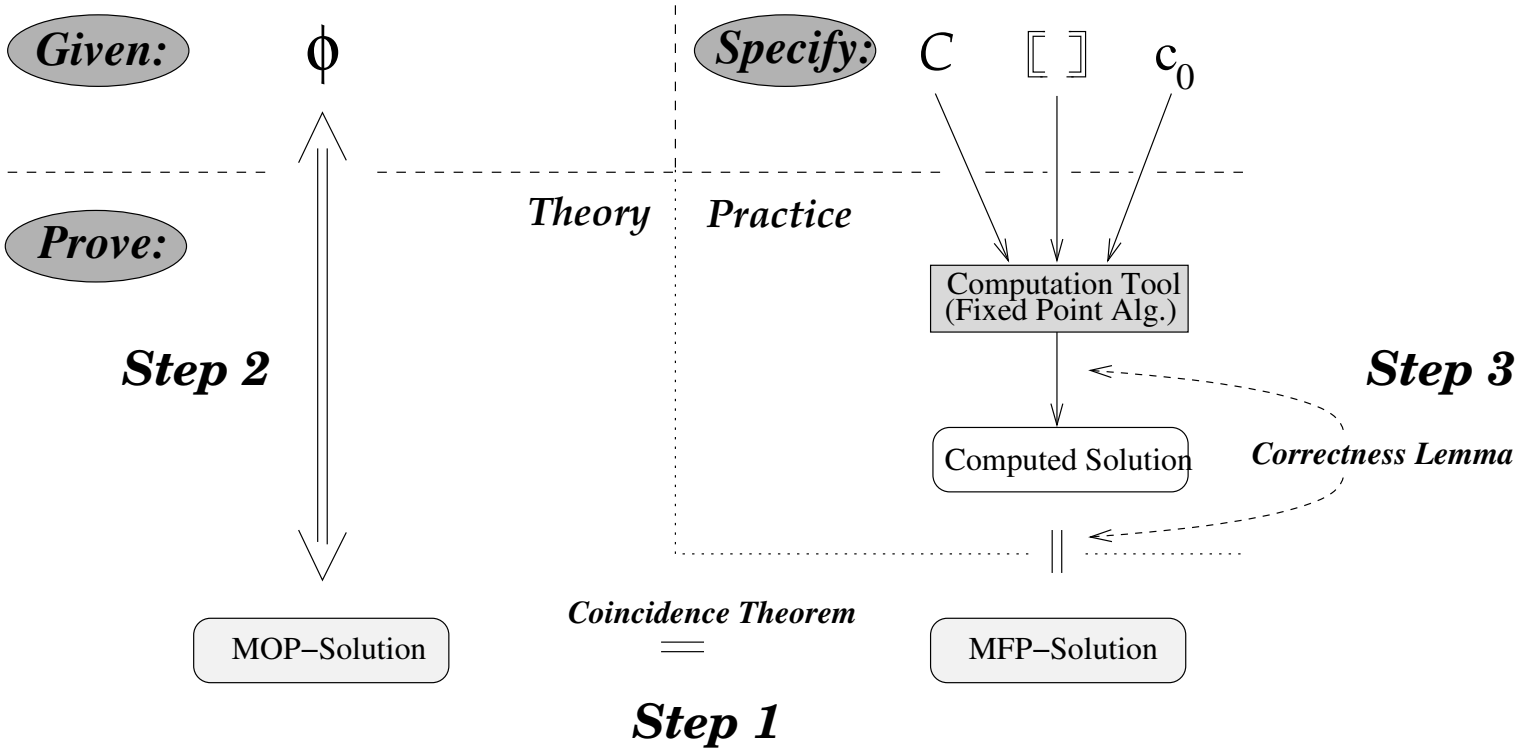
Effektivität/Terminierung

Theorem (6.5.3, Effektivität)

Der generische Fixpunktalgorithmus terminiert mit der MaxFP-Lösung, falls das Datenflussanalysefunktional monoton ist und der Verband die absteigende Kettenbedingung erfüllt.

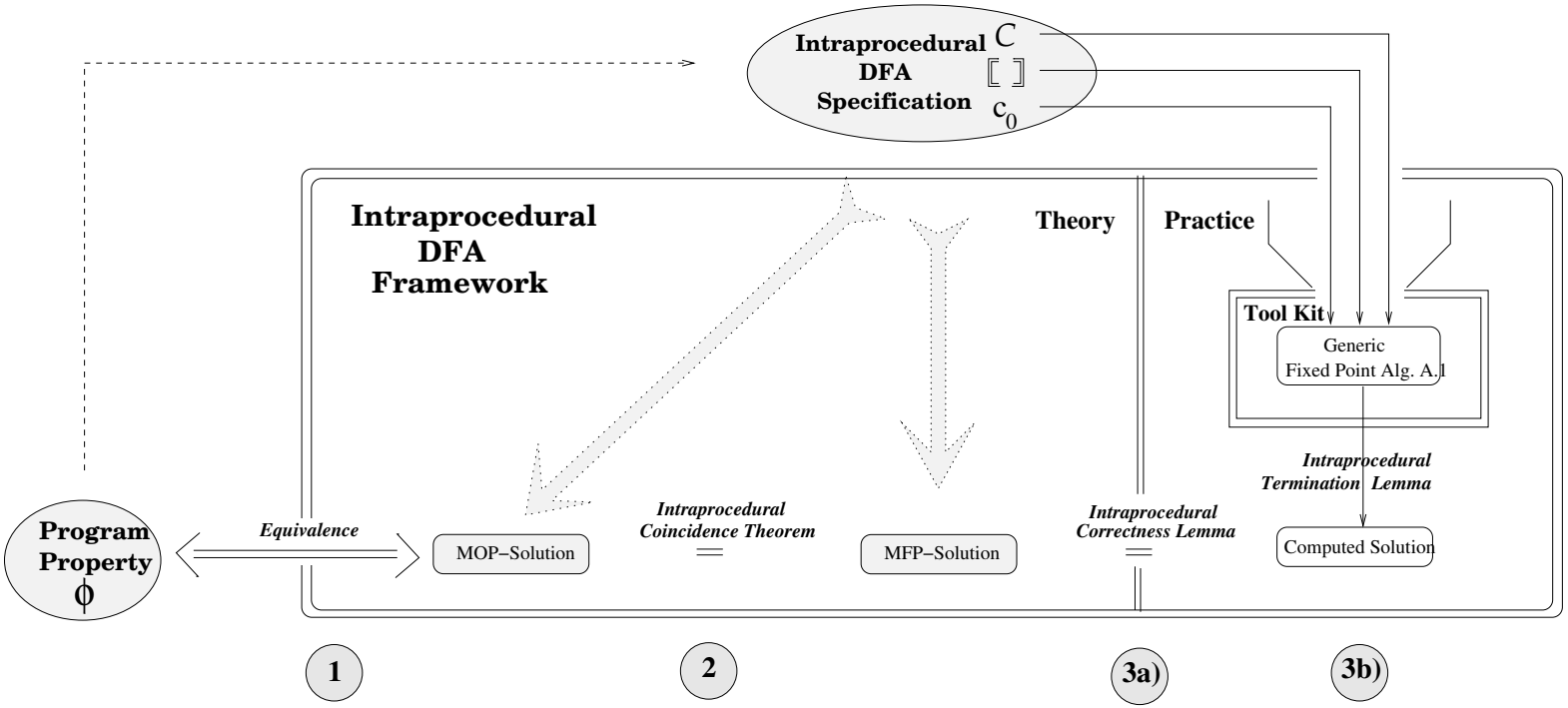
Intraprozedurale DFA im Überblick (1)

Ein Bild sagt mehr als 1000 Worte:



Intraprozedurale DFA im Überblick (2)

Fokussiert auf die Rahmen-/Werkzeugkistensicht:



Kapitel 6.6

Beispiele: Verfügbare Ausdrücke und Einfache Konstanten

Zwei prototypische DFA-Probleme

- Verfügbare Ausdrücke
  $\rightsquigarrow$ kanonisches Bsp. eines distributiven DFA-Problems
- Einfache Konstanten
  $\rightsquigarrow$ kanonisches Bsp. eines monotonen DFA-Problems

Verfügbare Ausdrücke

...ein typisches distributives DFA-Problem.

- Abstrakte Semantik für verfügbare Ausdrücke:
  1. Datenflussanalyseverband:
     $$(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathbb{B}, \wedge, \vee, \leq, \mathbf{false}, \mathbf{true})$$
  2. Datenflussanalysefunktional: $[\![\ ]\!]_{av} : E \to (\mathbb{B} \to \mathbb{B})$
     definiert durch

$$\forall\, e \in E.\ [\![\, e \,]\!]_{av} =_{df} \begin{cases} Cst_{\mathbf{true}} & \text{falls } Comp_e \wedge Transp_e \\ Id_{\mathbb{B}} & \text{falls } \neg Comp_e \wedge Transp_e \\ Cst_{\mathbf{false}} & \text{sonst} \end{cases}$$

Verfügbare Ausdrücke

Dabei bezeichnen:

- $\hat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, \leq, \mathbf{false}, \mathbf{true})$: Verband der Wahrheitswerte mit $\mathbf{false} \leq \mathbf{true}$ und dem logischen "und" und "oder" als Schnitt- bzw. Vereinigungsoperation $\sqcap$ and $\sqcup$.
- $Cst_{\mathbf{true}}$ und $Cst_{\mathbf{false}}$ die konstanten Funktionen "wahr" bzw. "falsch" auf $\hat{\mathbb{B}}$
- $Id_{\mathbb{B}}$ die Identität auf $\hat{\mathbb{B}}$

...und relativ zu einem fest gewählten Kandidatenausdruck t:

- $Comp_e$: t wird von der Instruktion an Kante e berechnet (d.h. t kommt rechtsseitig als (Teil-) Ausdruck vor)
- $Transp_e$: kein Operand von t erhält durch die Instruktion an Kante e einen neuen Wert (d.h. kein Operand von t kommt linksseitig vor: t ist transparent für e)

Verfügbare Ausdrücke

Lemma

$[\![\ ]\!]_{av}$ *ist distributiv.*

Corollary

Für verfügbare Ausdrücke stimmen MOP - und MaxFP -Lösung überein.

Einfache Konstanten

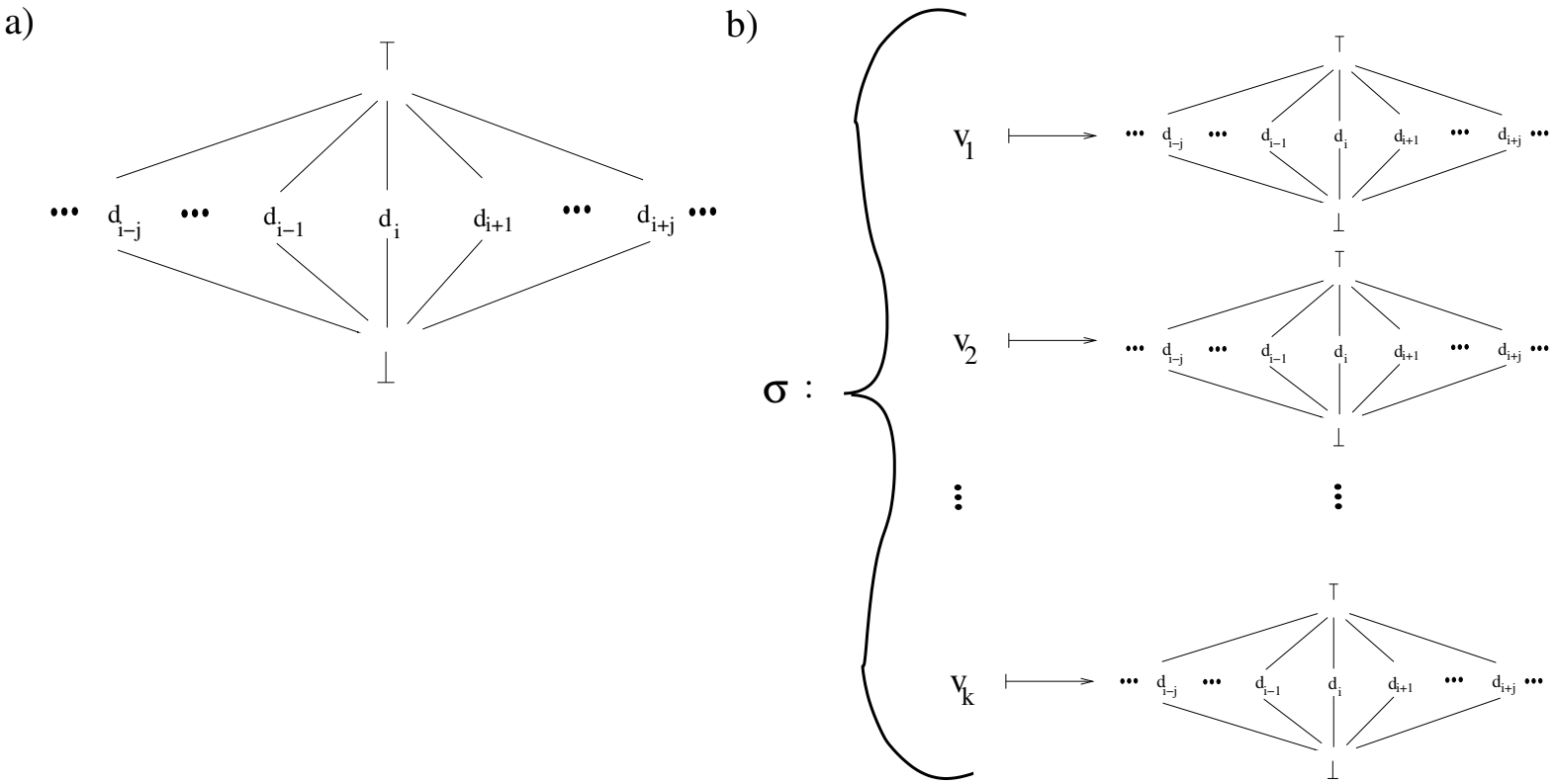
Ein typisches monotones (nicht distributives) DFA-Problem...

Abstrakte Semantik für einfache Konstanten

- Abstrakte Semantik für einfache Konstanten:
  1. Datenflussanalyseverband:
     $$(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\Sigma, \sqcap, \sqcup, \sqsubseteq, \sigma_\bot, \sigma_\top)$$
  2. Datenflussanalysefunktional:
     $[\![\ ]\!]_{sc} : E \to (\Sigma \to \Sigma)$ definiert durch
     $$\forall e \in E.\ [\![e]\!]_{sc} =_{df} \theta_e$$

Datenflussanalyseverband für einfache Konstanten

Der “kanonische” Verband für Konstantenausbreitung/-faltung:



Die Semantik von Termen

Die Semantik von Termen $t \in \mathbf{T}$ ist durch die induktiv definierte Evaluationsfunktion gegeben:

$$\mathcal{E} : \mathbf{T} \to (\Sigma \to \mathbf{D})$$

$$\forall t \in \mathbf{T}\, \forall \sigma \in \Sigma.\, \mathcal{E}(t)(\sigma) =_{df} \begin{cases} \sigma(x) & \text{falls } t = x \in \mathbf{V} \\ I_0(c) & \text{falls } t = c \in \mathbf{C} \\ I_0(op)(\mathcal{E}(t_1)(\sigma), \ldots, \mathcal{E}(t_r)(\sigma)) & \\ \qquad \text{falls } t = op(t_1, \ldots, t_r) & \end{cases}$$

Zu ergänzende Begriffe & Definitionen

...um die Definition der Termsemantik abzuschließen:

- Termsyntax
- Interpretation
- Zustand

Die Syntax von Termen (1)

Sei

- **V** eine Menge von Variablen und
- **Op** eine Menge von n-stelligen Operatoren, $n \geq 0$, sowie $\mathbf{C} \subseteq \mathbf{Op}$ die Menge der 0-stelligen Operatoren, der sog. *Konstanten* in **Op**.

Die Syntax von Termen (2)

Dann legen wir 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

Sei $\mathbf{D}'$ ein geeigneter Datenbereich (z.B. die Menge der ganzen Zahlen), seien $\bot$ und $\top$ zwei ausgezeichnete Elemente mit $\bot, \top \notin \mathbf{D}'$ und sei $\mathbf{D} =_{df} \mathbf{D}' \cup \{\bot, \top\}$.

Eine Interpretation über $\mathbf{T}$ und $\mathbf{D}$ ist ein Paar $I \equiv (\mathbf{D}, I_0)$, wobei

- I_0 eine Funktion ist, die mit jedem 0-stelligen Operator $c \in \mathbf{Op}$ ein Datum $I_0(c) \in \mathbf{D}'$ und mit jedem n-stelligen Operator $op \in \mathbf{Op}$, $n \geq 1$, eine totale Funktion $I_0(op) : \mathbf{D}^n \to \mathbf{D}$ 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 $d_j = \bot$)

Menge der Zustände

$$\Sigma =_{df} \{ \sigma \mid \sigma : \mathbf{V} \to \mathbf{D} \}$$

...bezeichnet die Menge der Zustände, d.h. die Menge der Abbildungen σ von der Menge der Programmvariablen $\mathbf{V}$ auf einen geeigneten (hier nicht näher spezifizierten) Datenbereich $\mathbf{D}$.

Insbesondere

- $\sigma_\perp$: ...bezeichnet den wie folgt definierten total undefinierten Zustand aus Σ: $\forall\, v \in \mathbf{V}.\ \sigma_\perp(v) = \perp$

Zustandstransformationsfunktion

Die Zustandstransformationsfunktion

$$\theta_\iota : \Sigma \to \Sigma, \quad \iota \equiv x := t$$

ist definiert durch:

$$\forall\, \sigma \in \Sigma\ \forall\, y \in \mathbf{V}.\ \theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

Einfache Konstanten

Lemma

$[\![\ ]\!]_{sc}$ *ist monoton.*

Beachte: Distributivität gilt i.a. nicht! (Übungsaufgabe)

Korollar

Für einfache Konstanten stimmen MOP - und MaxFP -Lösung i.a. nicht überein. Die MaxFP -Lösung ist aber stets eine sichere Approximation der MOP -Lösung für einfache Konstanten.

Ergänzende, weiterführende und vertiefende 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)

Ergänzende, weiterführende und vertiefende 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.


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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 6 (3)

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)

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)

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 6 (4)

Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 5, Static Program Analysis)

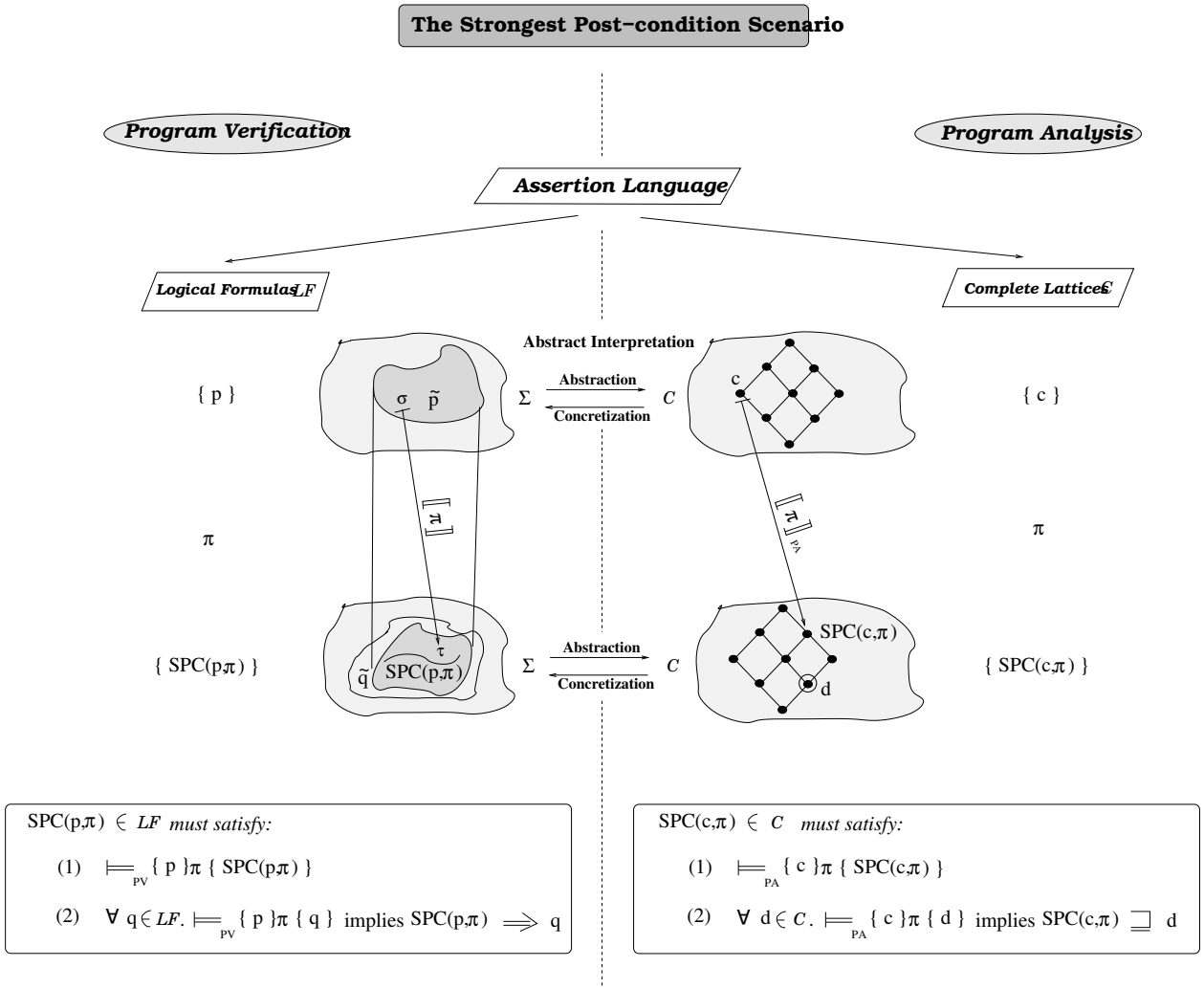
Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 7, Program Analysis; Chapter 8, More on Program Analysis; Appendix B, Implementation of Program Analysis)

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)

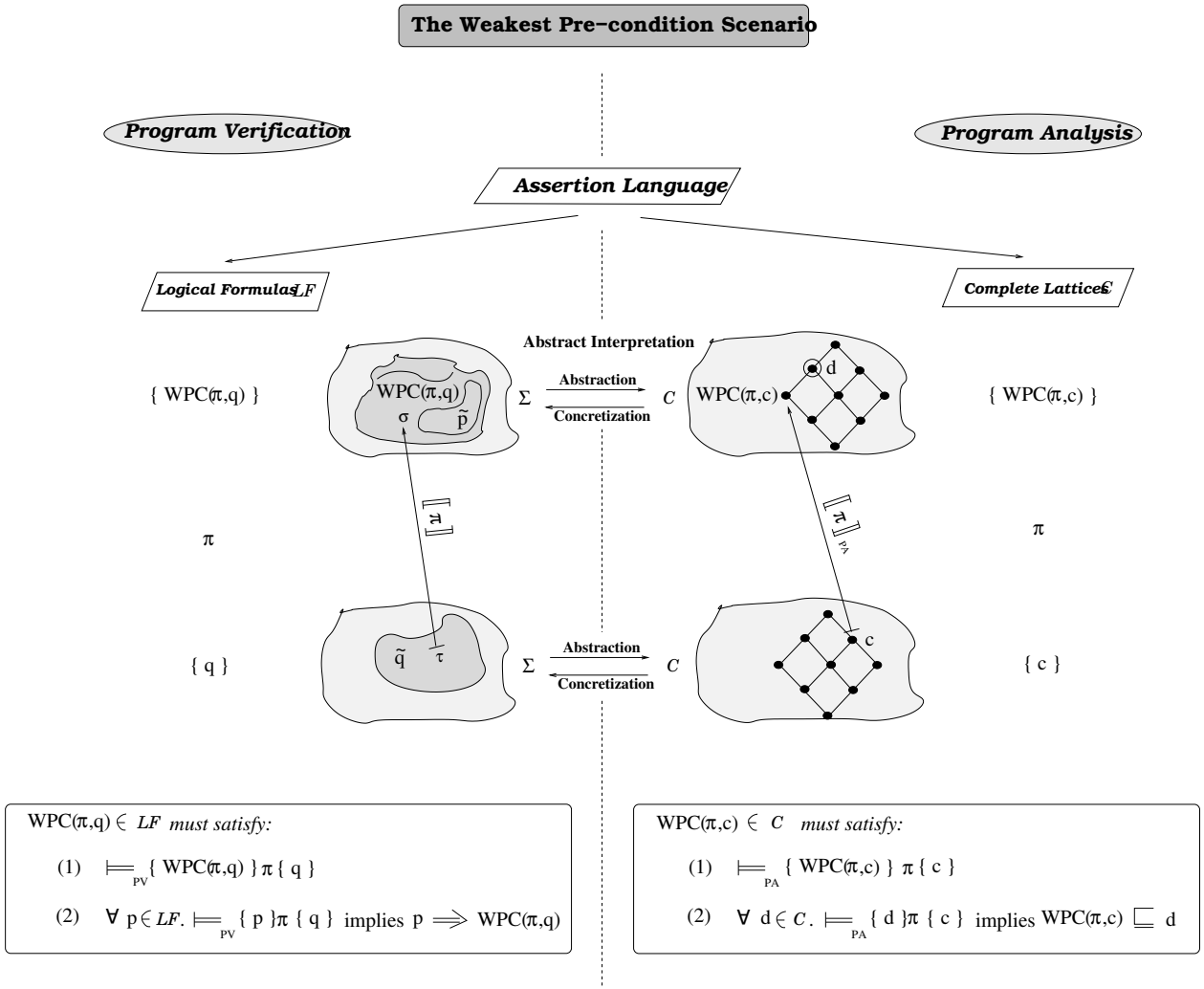
Kapitel 7

Programmverifikation vs. Programmanalyse

Programmverifikation vs. -analyse (1)



Programmverifikation vs. -analyse (2)



Kapitel 8

Reverse Datenflussanalyse

Kapitel 8.1

Grundlagen

Reverse abstrakte Semantik

Sei $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ eine (lokale) abstrakte Semantik auf $\mathcal{C}$. Dann ist die durch $[\![\ ]\!]$ definierte reverse (lokale) abstrakte Semantik wie folgt festgelegt:

- Reverse abstrakte Semantik
  1. Datenflussanalyseverband $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$
  2. Reverses Datenflussanalysefunktional $[\![\ ]\!]_R : E \to (\mathcal{C} \to \mathcal{C})$ definiert durch

$$\forall\, e \in E\ \forall\, c \in \mathcal{C}.\ [\![\, e \,]\!]_R(c) =_{df} \bigsqcap \{\, c' \mid [\![\, e \,]\!](c') \sqsupseteq c \,\}$$

DFA vs. RDFA

Intuition

- 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 bestimmter Datenflussfakt an einer gegebenen 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 $[\![\ ]\!]$ und $[\![\ ]\!]_R$ (1)

Lemma (8.1.1)

Sei $[\![\ ]\!]$ ein Datenflussanalysefunktional. Dann gilt für jede Kante $e \in E$:

1. $[\![e]\!]_R$ *ist wohldefiniert und monoton.*
2. $[\![e]\!]_R$ *ist additiv, falls $[\![e]\!]$ distributiv ist.*

Zusammenhang von $[\![\ ]\!]$ und $[\![\ ]\!]_R$ (2)

Lemma (8.1.2)

Sei $[\![\ ]\!]$ ein Datenflussanalysefunktional. Dann gilt für jede Kante $e \in E$:

1. $[\![e]\!]_R \circ [\![e]\!] \sqsubseteq Id_{\mathcal{C}}$, *falls* $[\![e]\!]$ *monoton ist.*
2. $[\![e]\!] \circ [\![e]\!]_R \sqsupseteq Id_{\mathcal{C}}$, *falls* $[\![e]\!]$ *distributiv ist.*

Sprechweise in der Theorie "Abstrakter Interpretation":

- $[\![e]\!]$ und $[\![e]\!]_R$ bilden eine Galois-Verbindung.

Zusammenhang von $[\![\ ]\!]$ und $[\![\ ]\!]_R$ (3)

Lemma (8.1.3)

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* $succ(\mathbf{q}) = \emptyset$.

Kapitel 8.2

R-JOP-Ansatz

Ausdehnung von $[\![\ ]\!]_R$ auf Pfade

Wir definieren ($p = \langle e_1, \ldots, e_{q-1}, e_q \rangle$):

$$[\![\, p \,]\!]_R =_{df} \begin{cases} Id_{\mathcal{C}} & \text{falls } \lambda_p < 1 \\ [\![\, \langle e_1, \ldots, e_{q-1} \rangle \,]\!]_R \circ [\![\, e_q \,]\!]_R & \text{sonst} \end{cases}$$

Beachte: Die obige Ausdehnung bedeutet einen Rückwärtsdurchlauf von Pfad p.

Der *R-JOP*-Ansatz

Die *R-JOP*-Lösung:

$$\forall\, c_q \in \mathcal{C}\ \forall\, n \in N.\ R\text{-}JOP_{c_q}(n) =_{df} \bigsqcup \{\, [\![\, p \,]\!]_R(c_q) \mid p \in \mathbf{P}[n, \mathbf{q}] \,\}$$

Kapitel 8.3

R-MinFP-Ansatz

Der *R-MinFP*-Ansatz

Das *R-MinFP*-Gleichungssystem:

$$reqInf\,(n) =$$

$$\begin{cases} c_q & \text{falls } n = \mathbf{q} \\ \bigsqcup \{ [\![(n,m)]\!]_R(reqInf\,(m)) \mid m \in succ(n) \} & \text{sonst} \end{cases}$$

Bezeichne $reqInf^*_{c_q}$ die kleinste Lösung dieses Gleichungssystems bzgl. $c_q \in \mathcal{C}$.

Die *R-MinFP*-Lösung:

$$\forall\, c_q \in \mathcal{C}\ \forall\, n \in N.\ R\text{-}MinFP_{c_q}(n) =_{df} reqInf\,^*_{c_q}(n)$$

Der generische *R-MinFP*-Alg. 8.3.1 (1)

Input: (1) A flow graph $G = (N, E, \mathbf{s}, \mathbf{e})$, (2) a program point q, (3) a reverse abstract semantics (i.e., a data-flow lattice $\mathcal{C}$, and a reverse data-flow functional $[\![\ ]\!]_R : E \to (\mathcal{C} \to \mathcal{C})$ induced by a functional $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$), and (4) a component information $c_q \in \mathcal{C}$.

Output: Under the assumption of termination (cf. Theorem 8.4.3), the *R-MinFP*-solution. Depending on the properties of the underlying reverse data-flow functional, this has the following interpretation.

(1) $[\![\ ]\!]_R$ is additive: Variable $reqInf[\mathbf{s}]$ stores the weakest context information of c_q, i.e., the least data-flow fact which must be ensured at the program entry in order to guarantee c_q at q. If this is $\top$, the requested component information cannot be satisfied at all.

Der generische *R-MinFP*-Alg. 8.3.1 (2)

(2) $[\![\ ]\!]_R$ is monotonic: Variable *reqInf*[**s**] stores a lower bound of the weakest context candidate of c_q. Generally, this is not a sufficient context information itself. Hence, except for the special case $\mathit{reqInf}[\mathbf{s}] = \top$, which implies that c_q cannot be satisfied by any consistent context information, nothing can be concluded from the value of *reqInf*[**s**].

Remark: The variable *workset* controls the iterative process. Its elements are nodes of G, whose informations annotating them have recently been updated.

Der generische *R-MinFP*-Alg. (3)

(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. (4)

```
( 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])⊔ reqInf [n];
            IF reqInf [n] ⊏ join
                THEN
                    reqInf [n] := join ;
                    workset := workset ∪{n}
            FI
        OD
    ESOOHC
OD.
```

Kapitel 8.4

Reverses Koinzidenz- und Sicherheitstheorem

Korrektheit

Theorem (8.4.1, Reverse Sicherheit)

Die R-MinFP-Lösung ist eine obere (d.h. sichere) Approximation der R-JOP-Lösung, d.h.,

$$\forall\, c_q \in \mathcal{C}\ \forall\, n \in N.\ R\text{-}MinFP_{c_q}(n) \sqsupseteq R\text{-}JOP_{c_q}(n)$$

Vollständigkeit (und Korrektheit)

Theorem (8.4.2, Reverse Koinzidenz)

Die R-MinFP-Lösung stimmt mit der R-JOP-Lösung überein, d.h.,

$$\forall\, c_q \in \mathcal{C}\ \forall\, n \in N.\ R\text{-}MinFP_{c_q}(n) = R\text{-}JOP_{c_q}(n)$$

falls $[\![\ ]\!]$ *distributiv ist.*

Effektivität / Terminierung

Zusammen mit der stets gegebenen Monotonie der reversen Semantikfunktionen erhalten wir:

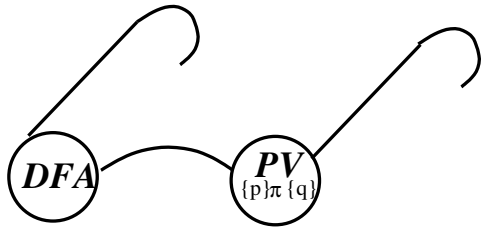
Theorem (8.4.3, Reverse Effektivität)

Algorithmus 8.3.1 terminiert mit der R-MinFP-Lösung, falls der Verband $\mathcal{C}$ die aufsteigende Kettenbedingung erfüllt.

Kapitel 8.5

Analyse und Verifikation: Analogien und Gemeinsamkeiten

(R)DFA vs. Verifikation



Data–flow Analysis | **Program Verification**

MaxFP — $[\![e]\!] \in \sqsubset C \xrightarrow{\text{distributive}} C \sqsupset$ — *Strongest Postcondition View*

Coincidence Theorem ||

$MOP_{c_s}(q) \sqsupseteq c_q$ — $\{p\}\pi\{?\}$

Link Theorem

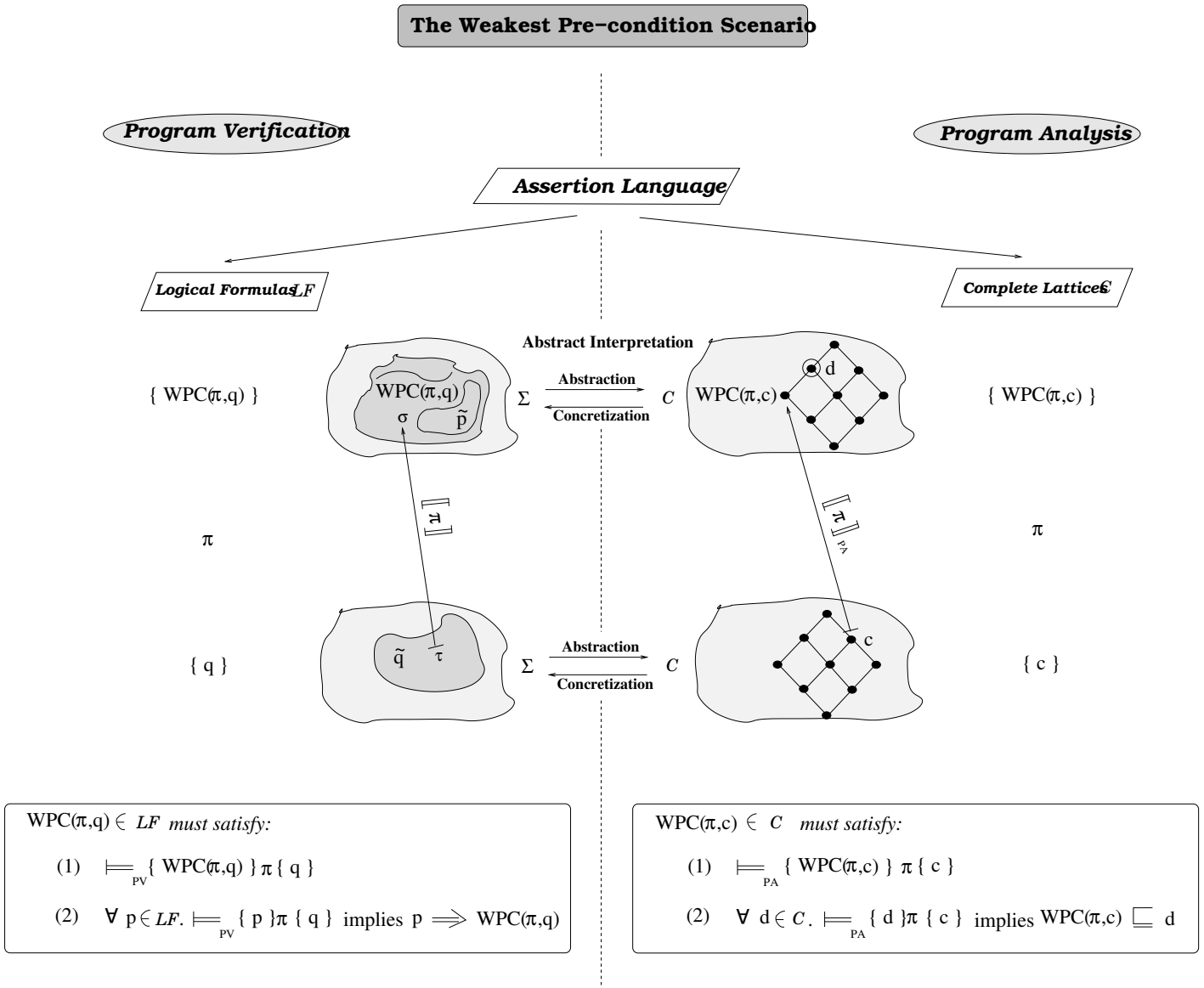
induces

$R\text{–}JOP_{c_q}(s) \sqsubseteq c_s$ — $[\![e]\!]_R(c) =_{df} \bigsqcap \{c' \mid [\![e]\!](c') \sqsupseteq c\}$ — $\{?\}\pi\{q\}$

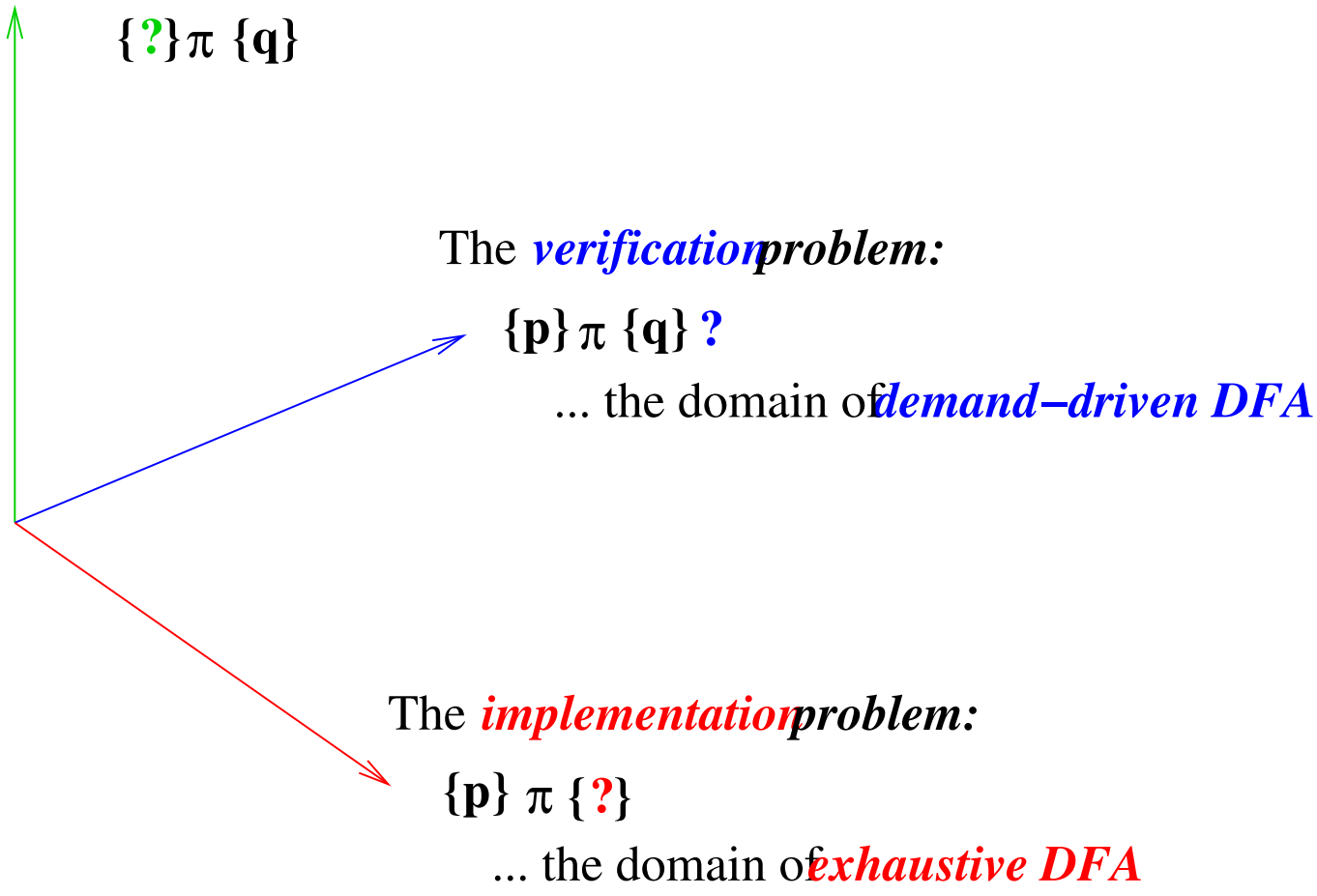
Reverse Coincidence Theorem ||

R–MinFP — *Weakest Precondition View*

Insgesamt: Die gewünschte WPC-Analogie

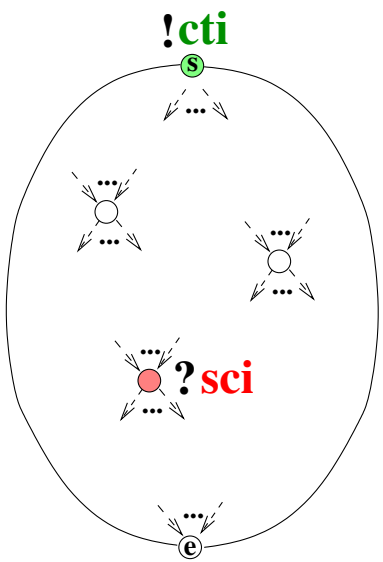


Drei unterschiedliche Problemperspektiven (1)



Drei unterschiedliche Problemperspektiven (2)

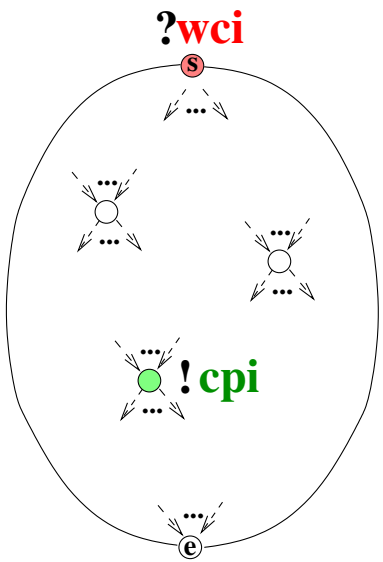
Implementation Problem



! **Given:** Context Information **cti**

? **Sought:** Strongest Component Information **sci**

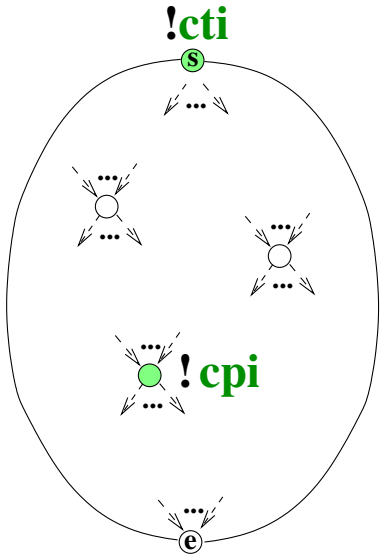
Specification Problem



! **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 8.6

Anwendungen reverser Datenflussanalyse

Beispiele

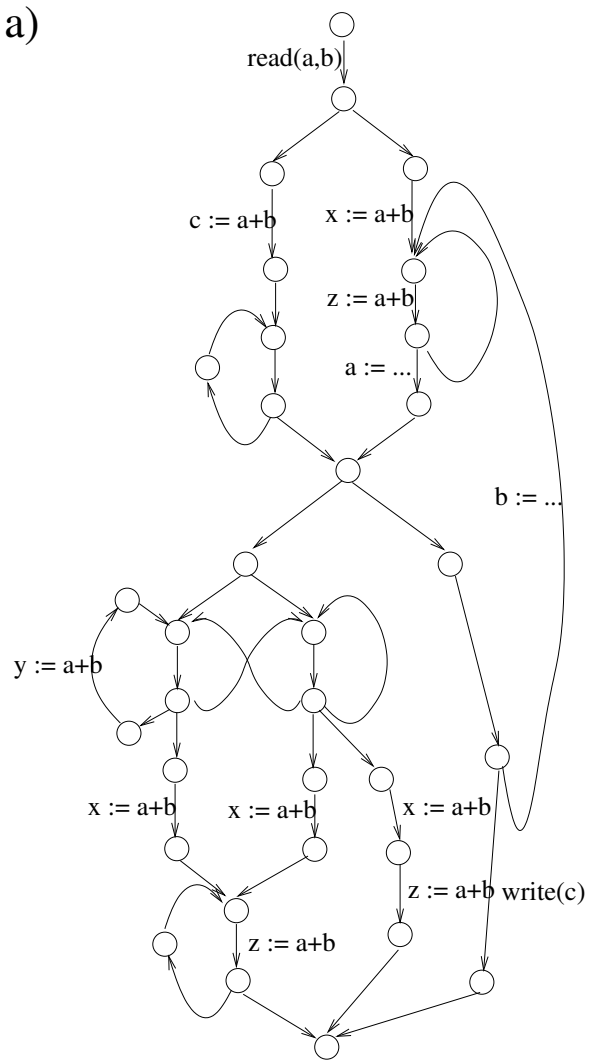
Anwendungen reverser DFA

- Einfache Analysatoren und Optimierer
- "Hot Spot" Programmanalyse und -optimierung
- Debugger
- ...

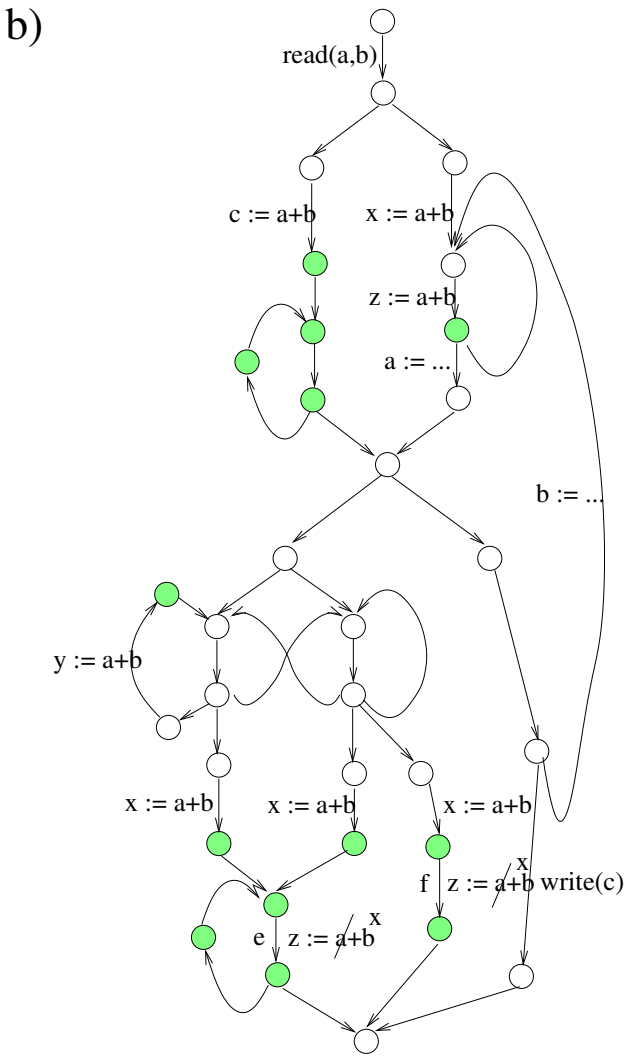
Insbesondere

- Anforderungsgetriebene Datenflussanalyse (Demand-driven data-flow analysis)

Einfacher Analysator und Optimierer



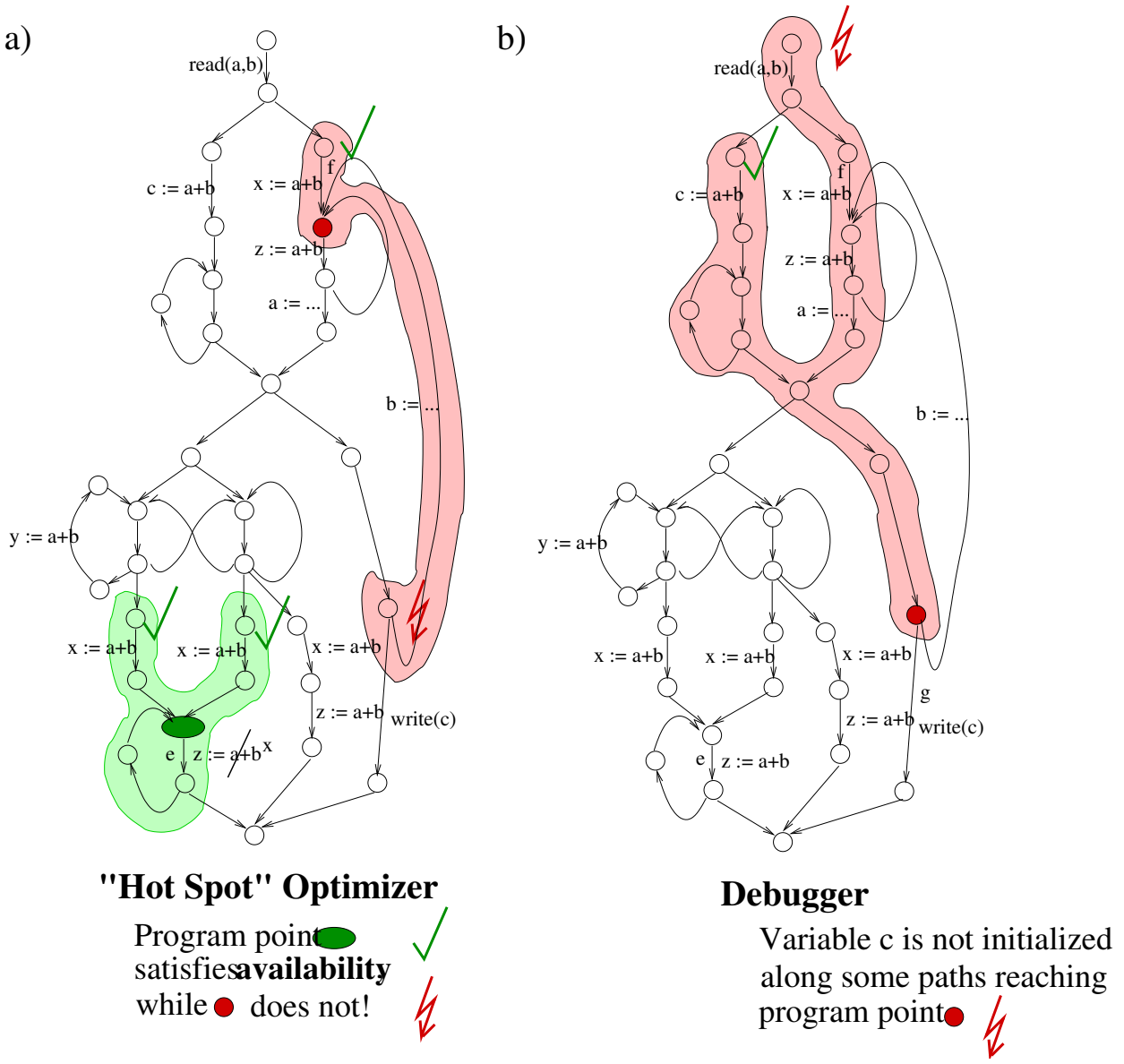
Flow graph



Data−flow information

Program points satisfying **availability**

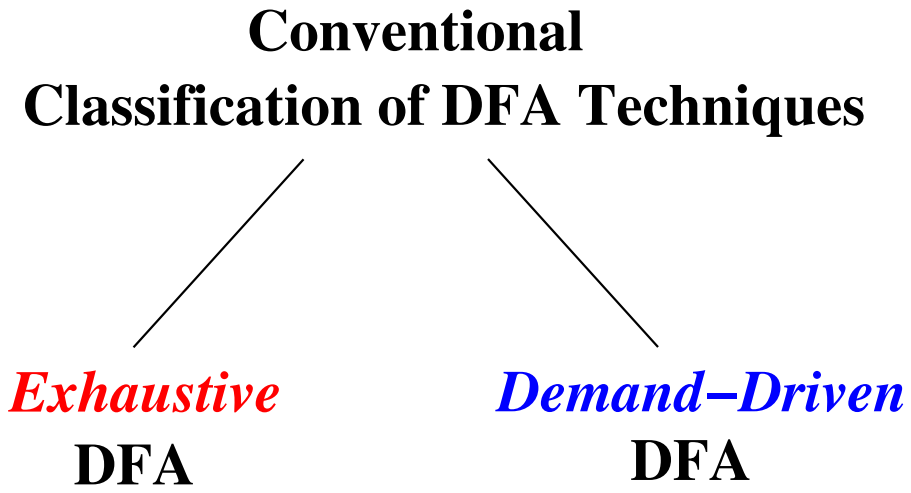
“Hot Spot”-Analysator&Optimierer, Debugger



Erschöpfende vs. anforderungsgetriebene DFA

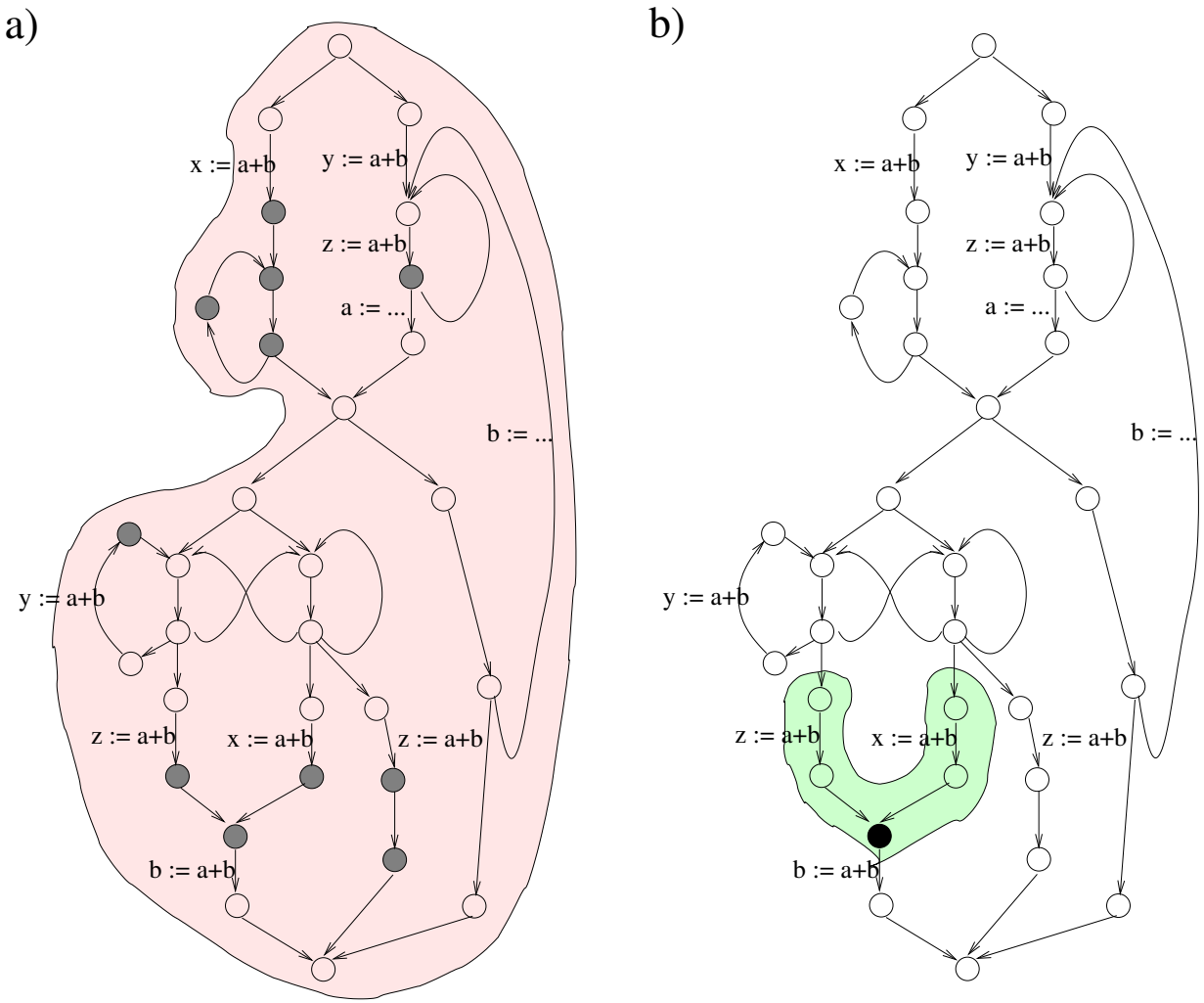
Erschöpfende (XDFA) vs.

anforderungsgetriebene DFA (DD-DFA):



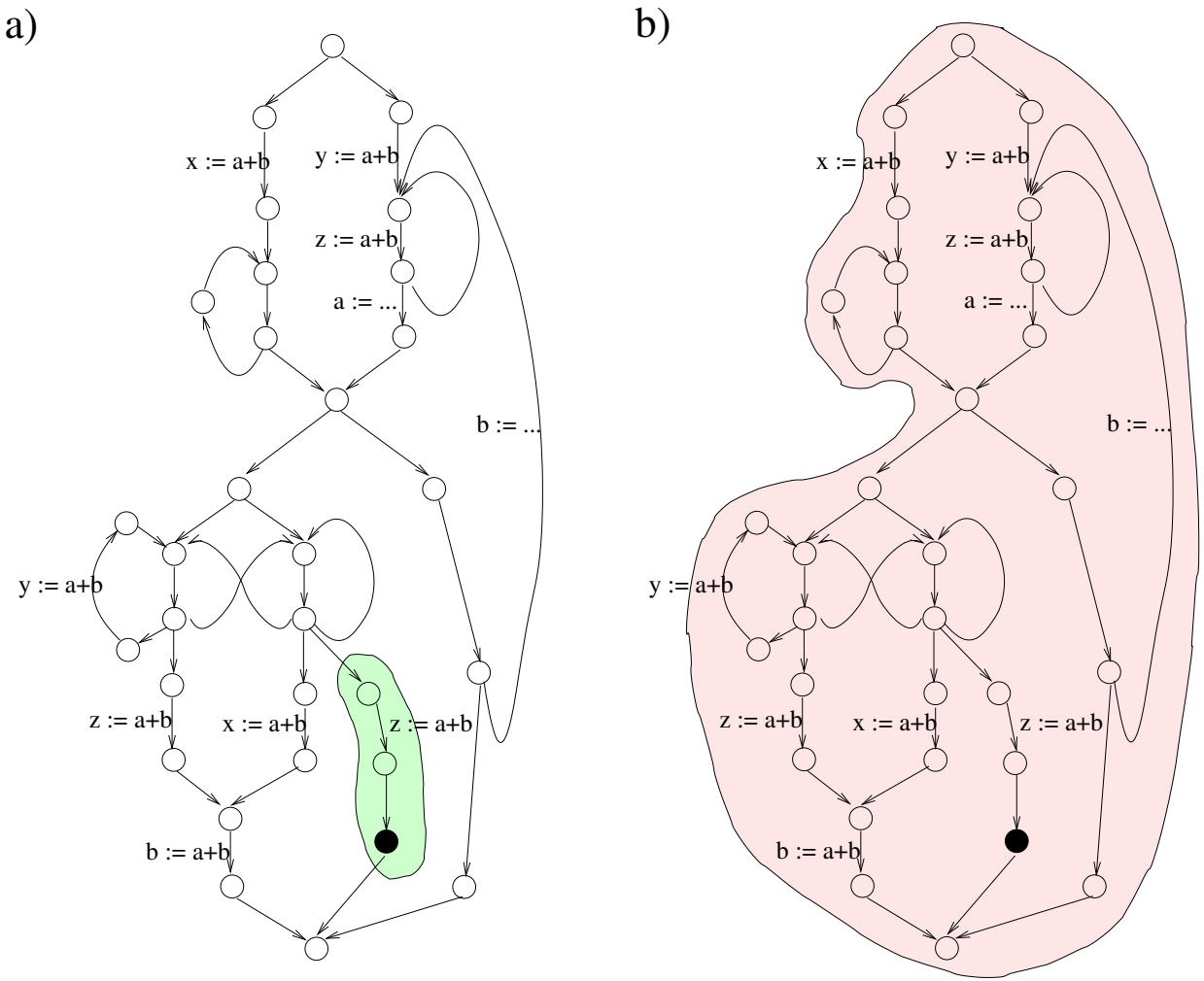
XDFA und DD-DFA im Vergleich (1)

Beispiel 8.5.1: Verfügbarkeit an einem Punkt – Vergleich Berechnungsaufwand erschöpfend (rosa), anforderungsgetrieben (grün)



XDFA und DD-DFA im Vergleich (2)

Beispiel 8.5.2: Verfügbarkeit an einem Punkt – Vergleich Berechnungsaufwand erschöpfend (rosa), anforderungsgetrieben (grün)



Reverse Verfügbarkeit

Beispiel 8.6.1.1: Reverse Verfügbarkeit

Erforderliche Hilfsfunktionen:

$R\text{-}Cst^X_{\mathbf{true}}$, $R\text{-}Cst^X_{\mathbf{false}}$, und $R\text{-}Id_{\mathcal{B}_X}$:

$$\forall\, b \in \mathcal{B}_X.\ R\text{-}Cst^X_{\mathbf{true}}(b) =_{df} \begin{cases} \mathbf{false} & \text{falls } b \in \mathbb{B} \\ \mathit{failure} & \text{sonst (d.h. falls } b = \mathit{failure}) \end{cases}$$

$$\forall\, b \in \mathcal{B}_X.\ R\text{-}Cst^X_{\mathbf{false}}(b) =_{df} \begin{cases} \mathbf{false} & \text{falls } b = \mathbf{false} \\ \mathit{failure} & \text{sonst} \end{cases}$$

$$R\text{-}Id_{\mathcal{B}_X} =_{df} Id_{\mathcal{B}_X}$$

wobei $\mathcal{B}_X =_{df} \{\mathbf{false}, \mathbf{true}, \mathit{failure}\}$.

Reverse Verfügbarkeit (fgs.)

Reverse abstrakte Semantik für Verfügbarkeit:

1. Datenflussanalyseverband:
   $(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathcal{B}_X, \wedge, \vee, \leq, \textbf{false}, \mathit{failure})$
2. Reverses Datenflussanalysefunktional:
   $[\![\ ]\!]_{av_R} : E \to (\ \mathcal{B}_X \to \mathcal{B}_X\ )$ definiert durch

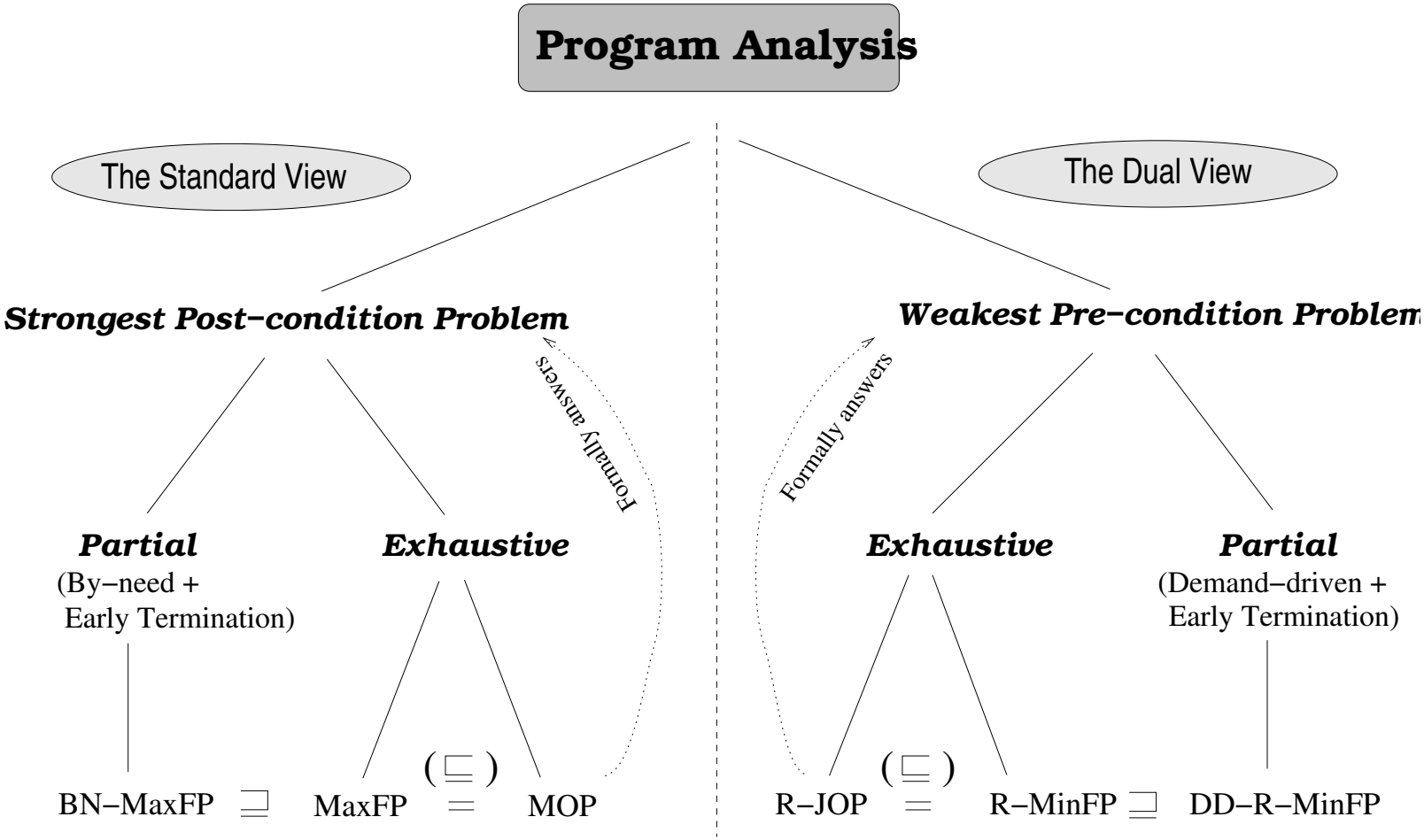
$$\forall\, e \in E.\ [\![\ e\ ]\!]_{av_R} =_{df} \begin{cases} R\text{-}Cst^X_{\textbf{true}} & \text{falls } [\![\ e\ ]\!]_{av} = Cst^X_{\textbf{true}} \\ R\text{-}Id_{\mathcal{B}_X} & \text{falls } [\![\ e\ ]\!]_{av} = Id_{\mathcal{B}_X} \\ R\text{-}Cst^X_{\textbf{false}} & \text{falls } [\![\ e\ ]\!]_{av} = Cst^X_{\textbf{false}} \end{cases}$$

Hinweis: Siehe auch Ergänzungsfolien zu "Hot Spot"-Programmanalyse und -optimierung auf der Webseite zur LVA.

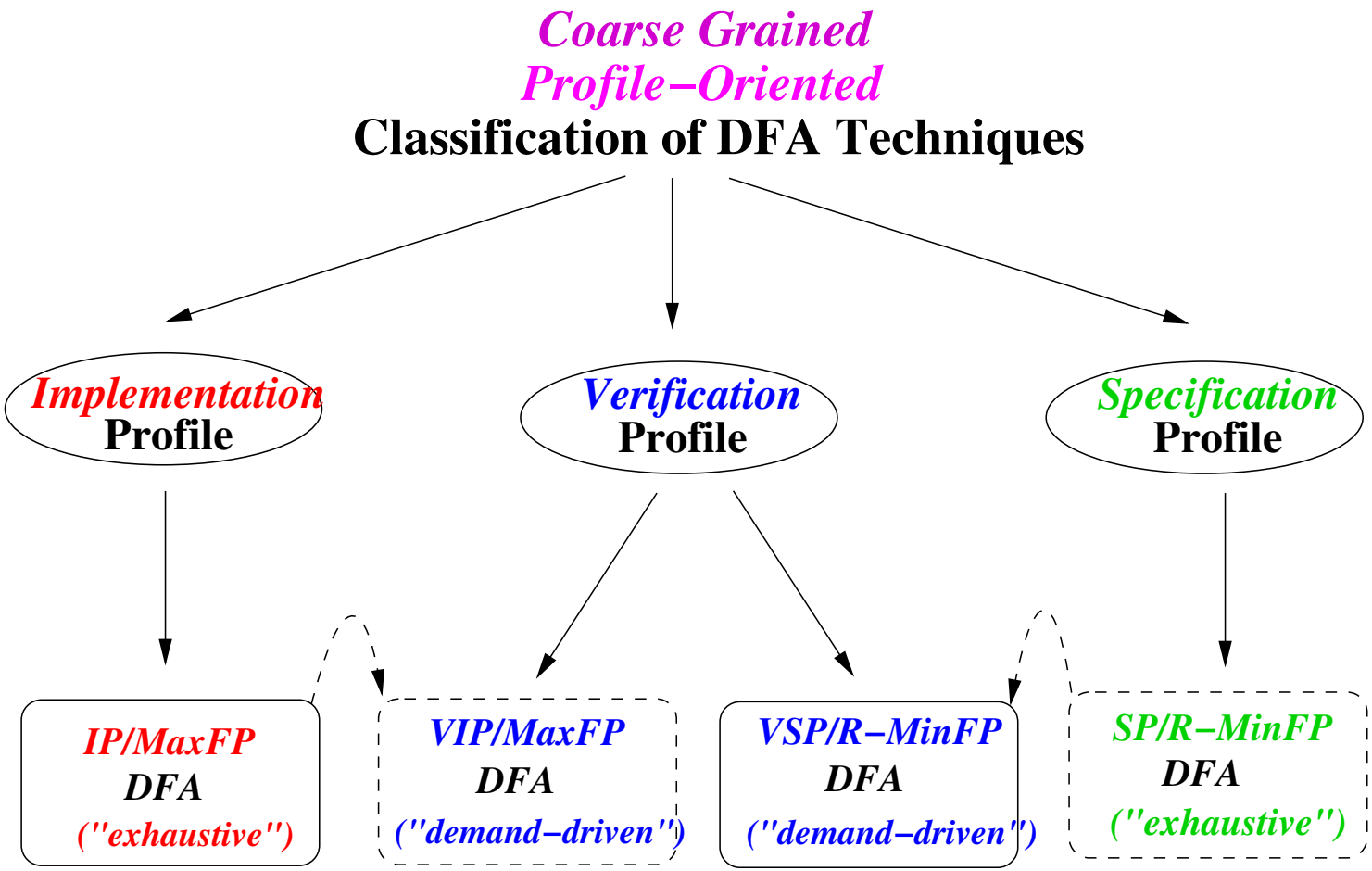
Kapitel 8.7

Zusammenfassung

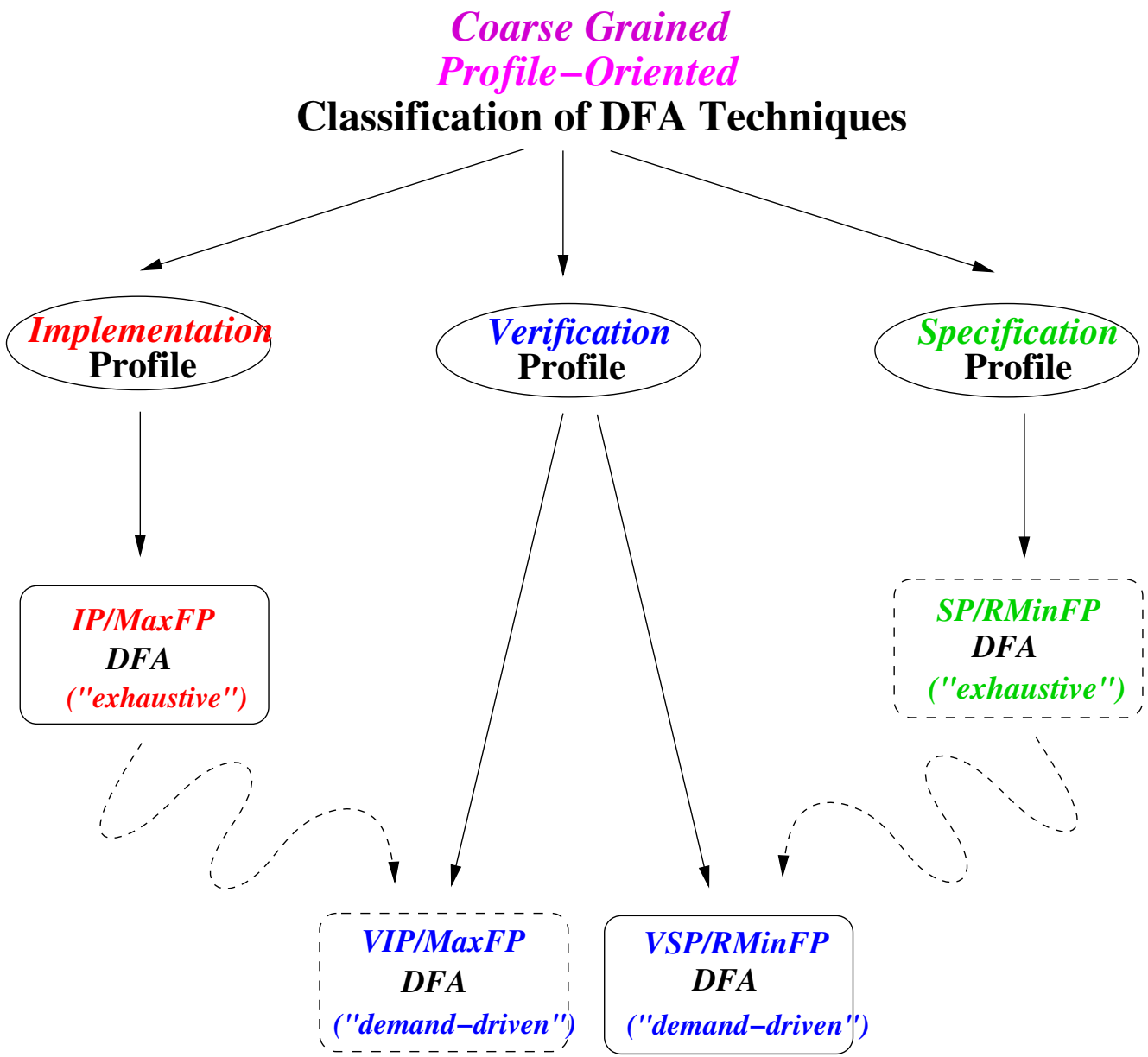
Gegenüberstellung von XDFA und DD-DFA



Induzierte Lösungsstrategien (1)

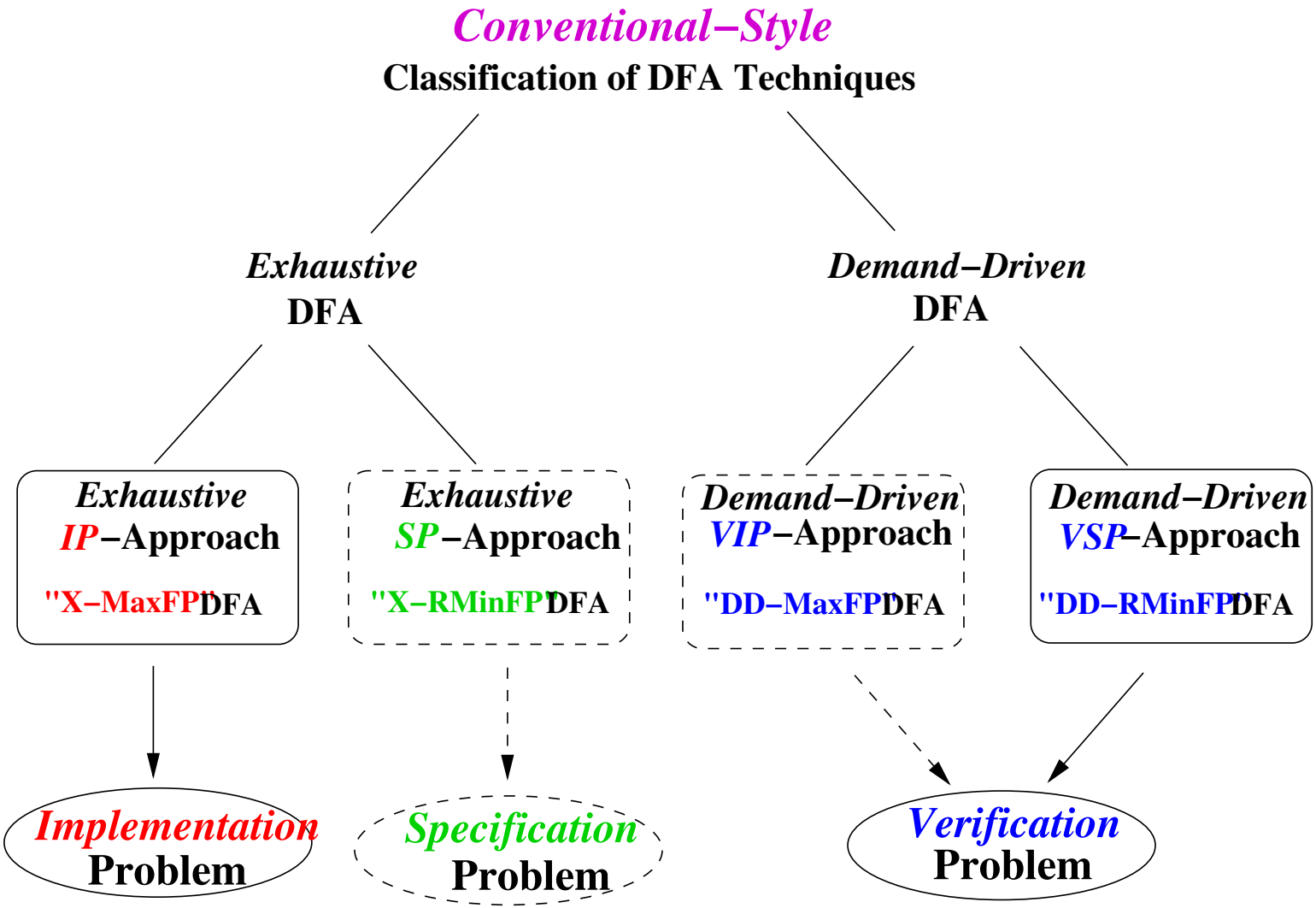


Induzierte Lösungsstrategien (2)



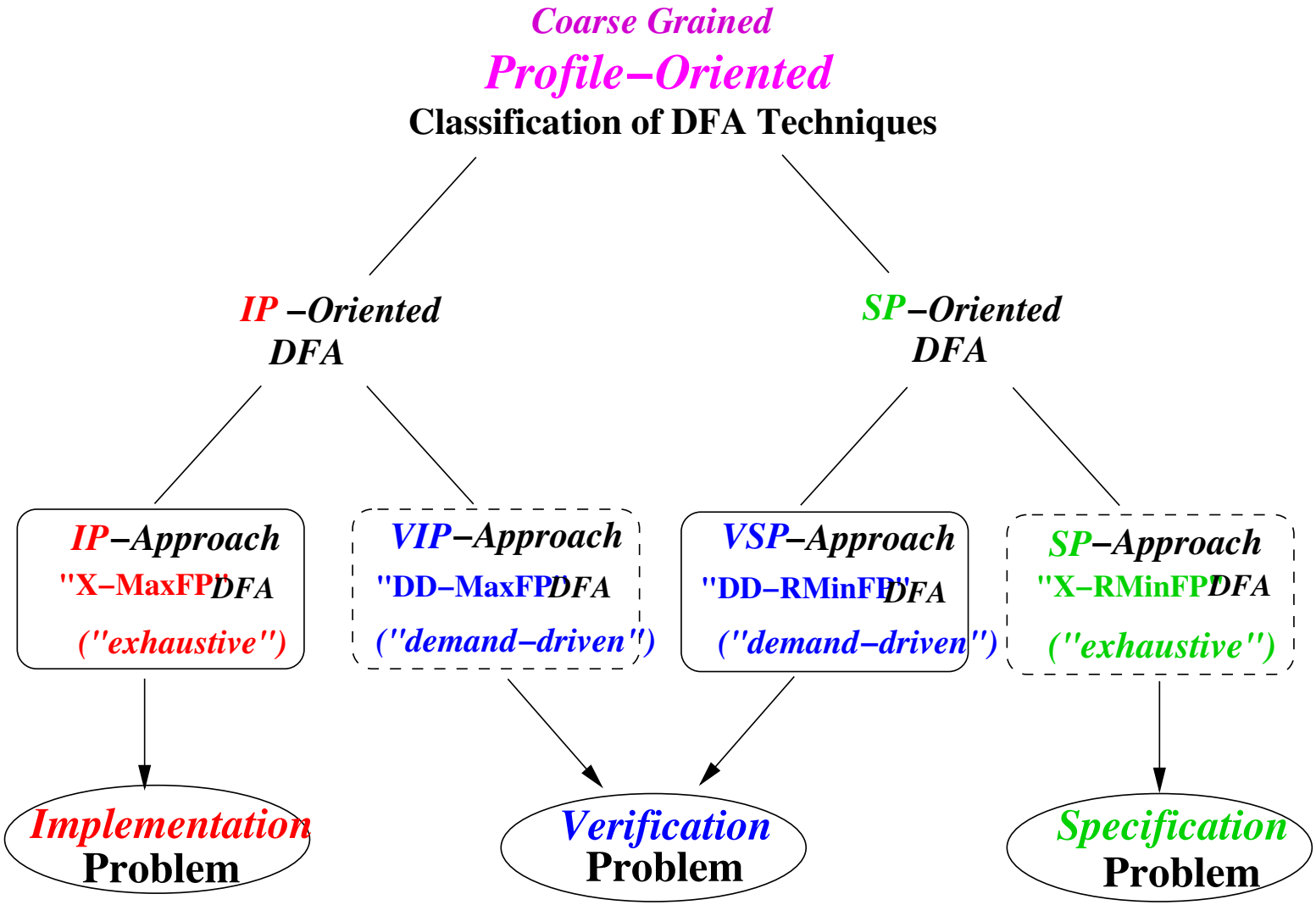
Algorithmenorientierte Sicht (1)

...zur Klassifikation von DFA-Problemen.



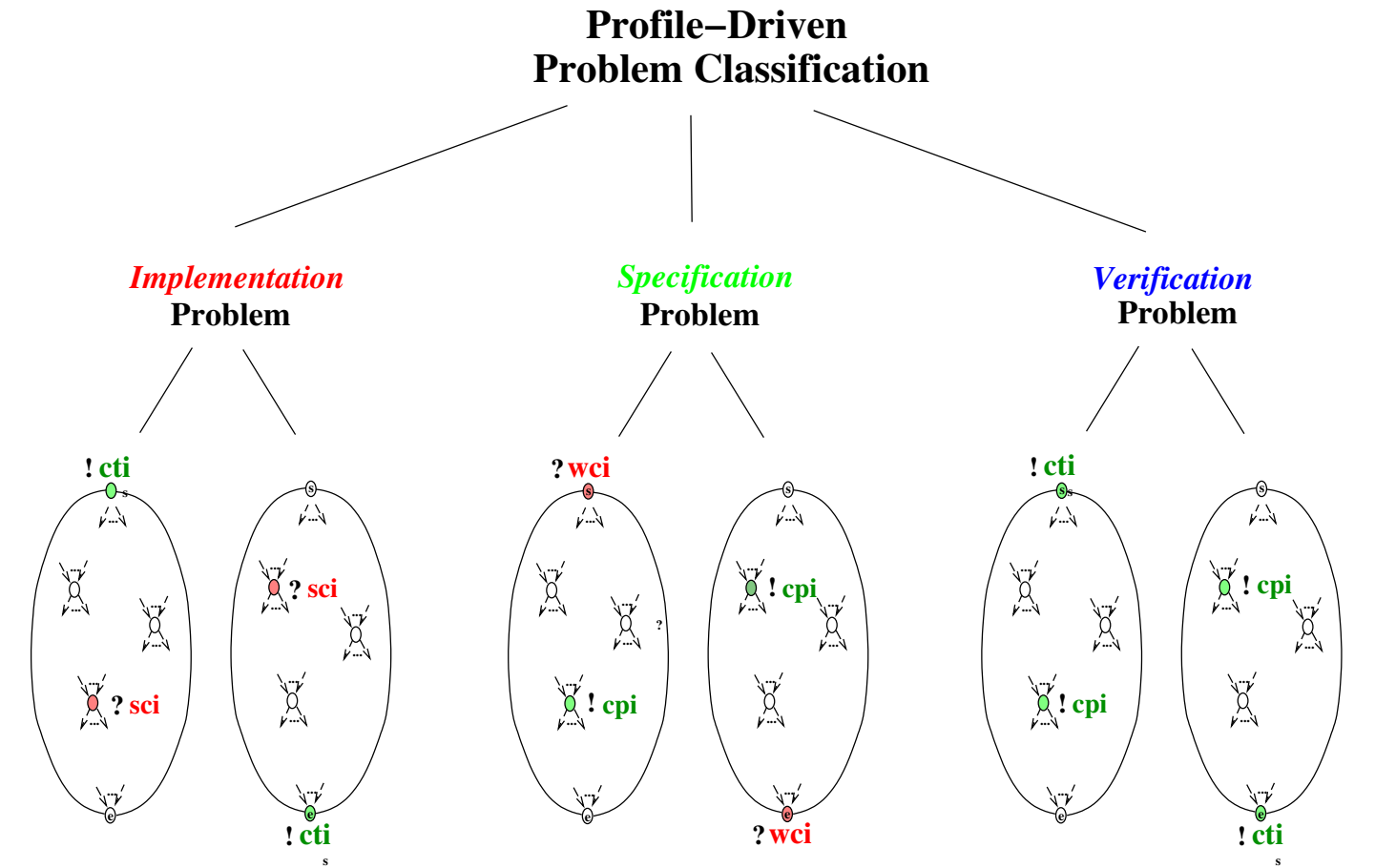
Algorithmenorientierte Sicht (2)

...zur Klassifikation von DFA-Problemen.



Programm- und problemorientierte Sicht

...zur Klassifikation von DFA-Problemen.



Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 8 (1)


G. 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.


W. A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part II - Demand Analysis*. Acta Informatica 10(3):265-272, 1978.


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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 8 (2)


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.


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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 8 (3)

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.

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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 8 (4)

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.

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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 8 (5)

Yuan Lin, David A. Padua. *Demand-driven Interprocedural Array Property Analysis*. In Proceedings of the 12th International Conference on Languages and Compilers for Parallel Computing (LCPC'99), Springer-V., LNCS, 1999.

Thomas Reps. *Solving Demand Versions of Interprocedural Analysis Problems*. In Proceedings of the 5th International Conference on Compiler Construction (CC'95), Springer-V., LNCS 786, 389-403, 1994.

Thomas Reps. *Demand Interprocedural Program Analysis using Logic Databases*. In Applications of Logic Databases, R. Ramakrishnan (Ed.), Kluwer Academic Publishers, 1994.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 8 (6)

Mary Lou Soffa. *Tutorial: Techniques to improve the Scalability and Precision of Data Flow Analysis*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 355-356, 1999.

Peng Tu, David A. Padua. *Gated SSA-based Demand-driven Symbolic Analysis for Parallelizing Computers*. In Proceedings of the International Conference on Supercomputing (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.

Kapitel 9

Chaotische Fixpunktiteration

Motivation

Viele praktisch relevante Probleme in der Informatik lassen sich durch die

- kleinste 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 $\langle D; \sqsubseteq \rangle$ ist.

Fixpunkte vs. Lösungen

Fixpunkte von Funktionen vs. Lösungen von Gleichungen:

Wir wollen einsehen:

Das

- Lösen eines Systems rekursiver Gleichungen

$$\begin{aligned} x &= f_1(x) \\ &\vdots \\ x &= f_n(x) \end{aligned}$$

entspricht der

- Berechnung eines Fixpunktes von $\mathcal{F}$, d.h. eines gemeinsamen Fixpunkts $x = f_k(x)$ für alle f_k.

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.

Wohlbekannte Fixpunkttheoreme aus der Literatur

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]
- ...

Verallgemeinerungen

- 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
    $J_i \subseteq \{1, \ldots, n\}$ 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

- A. 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:

- J.-L. Lassez, V.L. Nguyen, E.A. Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters, Vol. 14, No. 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 CC'96, LNCS 1060, 106-120, 1996.

Vorbereitung (1)

- Eine partielle Ordnung $\langle D; \sqsubseteq \rangle$ ist ein Paar aus einer Menge D und einer reflexiven, antisymmetrischen und transitiven zweistelligen Relation $\sqsubseteq \subseteq D \times D$.
- Eine Folge $(d_i)_{i \in \mathbf{N}}$ von Elementen $d_i \in D$ heißt (aufsteigende) Kette, falls $\forall i \in \mathbf{N}.\ d_i \sqsubseteq d_{i+1}$.
- Eine Kette $T =_{df} (d_i)_{i \in \mathbf{N}}$ heißt stationär, falls $\{d_i \mid i \in \mathbf{N}\}$ endlich ist.
- Eine partielle Ordnung $\sqsubseteq$ heißt wohlfundiert, falls jede Kette stationär ist.

Vorbereitung (2)

- Eine Funktion $f : D \to D$ auf D heißt
  - monoton, falls $\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$.
  - vergrößernd (inflationär), falls $d \sqsubseteq f(d)$ für alle $d \in D$
- 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 (9.1, Strategie, Chaotische Iterationsfolge, Fairness)

Sei $\langle D; \sqsubseteq \rangle$ eine partielle Ordnung und $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine Familie vergrößernder 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, Verzögerungsmonotonie)

Sei $\langle D; \sqsubseteq \rangle$ eine partielle Ordnung und $\mathcal{F} =_{df} (f_k)_{k \in \boldsymbol{N}}$ eine Familie von Funktionen $f_k : D \to D$. Dann heißt $\mathcal{F}$ verzögert-monoton, falls für alle $k \in \boldsymbol{N}$ gilt:

$$d \sqsubseteq d' \text{ impliziert } \exists s \in \boldsymbol{N}^*.\, f_k(d) \sqsubseteq f_s(d')$$

Lemma (9.3)

$\mathcal{F}$ ist verzögert-monoton, wenn alle f_k im üblichen Sinn monoton sind.

Das "monotoniefreie" Fixpunkttheorem

Theorem (9.4, Chaotische Fixpunktiteration)

Sei $\langle D; \sqsubseteq \rangle$ eine wohlfundierte partielle Ordnung mit kleinstem Element $\bot$, $\mathcal{F} =_{df} (f_k)_{k \in \mathbf{N}}$ eine verzögert-monotone Familie vergrößernder Funktionen und $\gamma : \mathbf{N} \to \mathbf{N}$ 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 9.5

Nichtdeterministischer Rumpf-Algorithmus:

$d := \bot;$
while $\exists\, k \in N.\ d \neq f_k(d)$ **do**
  choose $k \in N$ **where** $d \sqsubset f_k(d)$ **in**
    $d := f_k(d)$
  ni
od

Anwendungen von Fixpunkttheorem 9.4

Spezialfall: Vektor-Iterationen

Vorbereitung:

- Sei $\langle C; \sqsubseteq_C \rangle$ eine wohlfundierte partielle Ordnung und $D = C^n$ für ein $n \in \mathbb{N}$, geordnet durch die punktweise Ausdehnung von $\sqsubseteq$ auf $\sqsubseteq_C$.
- 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.

Vektor-Iterationen

Definition (9.6, 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.

Erste Anwendung von Fixpunkttheorem 9.4

...zur Modellierung der Vektor-Iteration.

Vorbereitung: Verallgemeinerung des Strategiebegriffs auf einen Strategiebegriff für Mengen

Definition (9.7, Mengenstrategie, faire Mengenstrategie)

- Eine Mengenstrategie ist eine (beliebige) Funktion $\gamma : \boldsymbol{N} \to \mathcal{P}(\{1, \ldots, n\})$.

  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

$$\forall i \in \boldsymbol{N},\ J \subseteq \boldsymbol{N}.\ (f_J(d_i) \neq d_i \text{ impliziert } \exists j > i.\ d_j \neq d_i)$$

Modellierungsresultate (1)

Lemma (9.8, Vektor-Iterationen)

Sei $\langle C; \sqsubseteq_C \rangle$ 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 $\sqsubseteq$ auf $\sqsubseteq_C$. 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 (9.9, Chaotische Vektor-Iterationen)

Sei $\langle C; \sqsubseteq_C \rangle$ 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 $\sqsubseteq$ auf $\sqsubseteq_C$. 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 9.9 ist ein Spezialfall von Fixpunkttheorem 9.4 für Vektor-Iterationen und folgt zusammen mit Lemma 9.8.
- Für $|\mathcal{F}| = 1$ reduziert sich Korollar 9.9 auf Tarskis Fixpunkttheorem im Fall wohlfundierter partieller Ordnungen.

Zweite Anwendung von Fixpunkttheorem 9.4

...auf intraprozedurale DFA.

Erinnerung:

Das *MaxFP*-Gleichungssystem

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

Die *MaxFP*-Lösung:

Die größte Lösung des *MaxFP*-Gleichungssystems

Dual dazu: Das *MinFP*-Gleichungssystem 9.10

Das *MinFP*-Gleichungssystem

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

Die *MinFP*-Lösung:

Die kleinste Lösung des *MinFP*-Gleichungssystems

Der *MinFP*-Fixpunktalgorithmus 9.11

$inf[\mathbf{s}] := c_{\mathbf{s}}$;
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. d. *MinFP*-Lösung

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. d. *MinFP*-Lösung (fgs.)

Sei $D =_{df} \mathcal{C}^{\mathrm{n}}$ versehen mit der punktweisen Ausdehnung von $\sqsubseteq$.

Dann gilt:

- D ist eine wohlfundierte partielle Ordnung.
- Ein Wert $d = (d^1, \ldots, d^{\mathrm{n}})$ stellt eine Annotation des Flussgraphen dar, wobei dem Knoten k der Wert d^k zugewiesen ist.

Für jeden Knoten k des Flussgraphen definieren wir jetzt eine Funktion $f^k : D \to C$ durch

$$f^k(d^1, \ldots, d^{\mathrm{n}}) =_{df} d'^k$$

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

Lemma (9.12)

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 (9.13, Korrektheit und Terminierung)

Jeder Lauf von MinFP-Algorithmus 9.11 terminiert mit der MinFP-Lösung.

Charakterisierungsresultate (fgs.)

Beweisanmerkungen

- Der *MinFP*-Fixpunktalgorithmus 9.11 folgt dem Muster von Rumpfalgorithmus 9.5 mit $\mathcal{F} = \{f_{\{k\}} \mid 1 \leq k \leq \mathtt{n}\}$.
- Die Verwendung von *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 9.9 erfüllt, womit Theorem 9.13 folgt.

Ausblick

Weitere Anwendungen des "monotoniefreien" Fixpunkttheorems 9.4 in Kapitel 11 und 12 zum Beweis der Optimalität von

- Partially Dead-Code Elimination
- Partially Redundant-Assignment Elimination

Andere Fixpunkttheoreme sind dafür nicht anwendbar.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 9

- Alfons Geser, Jens Knoop, Gerald Lüttgen, Oliver Rüthing, Bernhard Steffen. *Non-monotone Fixpoint Iterations to Resolve Second Order Effects*. In Proceedings CC'96, Springer-V., LNCS 1060, 106-120, 1996.
- J.-L. Lassez, V.L. Nguyen, E.A. Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.
- 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.

Kapitel 10

Basisblock- vs. Einzelanweisungsgraphen

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: Einfach eine Geschmacksfrage?

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: Einfach 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.

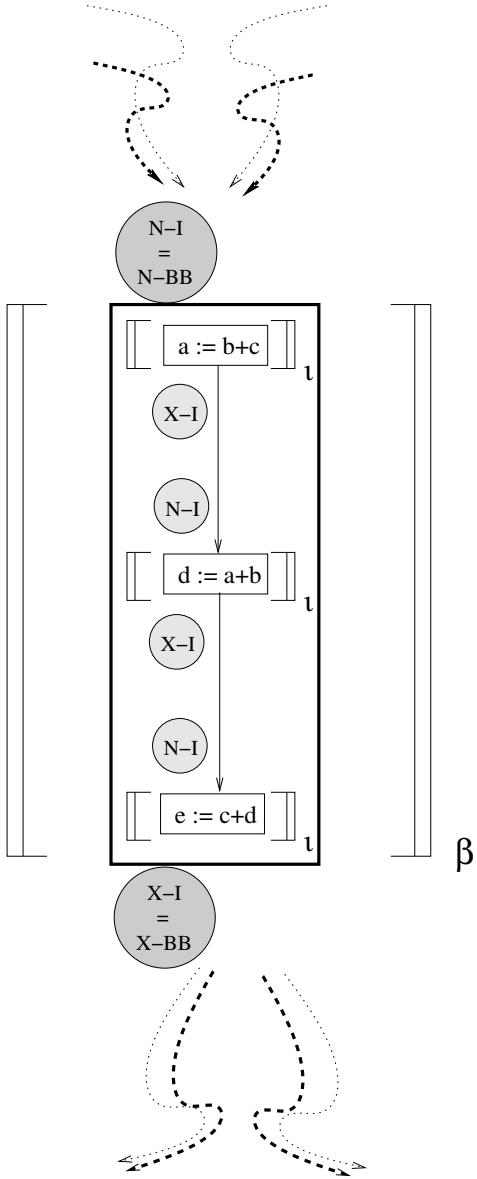
Basisblöcke: Sichere Nachteile

...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

EA-*MOP*-Ansatz

...für kantenbenannte Einzelanweisungsgraphen.

Die *MOP*-Lösung:

$$\forall\, c_s \in \mathcal{C}\ \forall\, n \in N.\ MOP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \bigsqcap \{\, \llbracket\, p\,\rrbracket_\iota(c_s) \mid p \in \mathbf{P}_G[s, n] \,\}$$

EA-*MaxFP*-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:

$$inf(n) = \begin{cases} c_s & \text{falls } n = s \\ \bigsqcap \{\, \llbracket (m,n) \rrbracket_\iota(inf(m)) \mid m \in pred_G(n) \,\} & \text{sonst} \end{cases}$$

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.

BB-*MOP*-Ansatz (1)

...für knotenbenannte Basisblockgraphen.

Die *MOP*-Lösung auf BB-Ebene:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, \mathbf{n} \in \mathbf{N}.\ MOP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\ N\text{-}MOP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ X\text{-}MOP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n})\ )$$

mit

$$N\text{-}MOP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\, [\![\ p\ ]\!]_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s}, \mathbf{n}[\,\}$$

$$X\text{-}MOP_{(\![\ ]\!]_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \bigsqcap \{\, [\![\ p\ ]\!]_\beta(c_{\mathbf{s}}) \mid p \in \mathbf{P_G}[\mathbf{s}, \mathbf{n}]\,\}$$

BB-*MOP*-Ansatz (2)

...und ihre Fortsetzung auf EA-Ebene:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathbf{s}})}(n) =_{df} \\ (\ N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathbf{s}})}(n),\ X\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_{\mathbf{s}})}(n)\ )$$

BB-*MOP*-Ansatz (3)

...mit

$$N\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) \quad =_{df} \quad \begin{cases} N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_s)}(\texttt{block}(n)) \\ \quad \text{falls } n = start(\texttt{block}(n)) \\ \\ \llbracket\ p\ \rrbracket_\iota(N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_s)}(\texttt{block}(n))) \\ \quad \text{sonst } (p \text{ Präfixpfad} \\ \quad \text{von } start(\texttt{block}(n)) \\ \quad \text{bis (ausschließlich) } n) \end{cases}$$

$$X\text{-}MOP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) \quad =_{df} \quad \llbracket\ p\ \rrbracket_\iota(N\text{-}MOP_{(\llbracket\ \rrbracket_\beta, c_s)}(\texttt{block}(n))) \\ (p \text{ Präfix von } start(\texttt{block}(n)) \\ \text{bis (einschließlich) } n)$$

BB-*MaxFP*-Ansatz (1)

...für knotenbenannte Basisblockgraphen:

Die *MaxFP*-Lösung auf BB-Ebene:

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\, \forall\, \mathbf{n} \in \mathbf{N}.\ \mathit{MaxFP}_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \\ (\ \mathit{N\text{-}MFP}_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}),\ \mathit{X\text{-}MFP}_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n})\ )$$

mit

$$\mathit{N\text{-}MFP}_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \mathrm{pre}^{\beta}_{c_{\mathbf{s}}}(\mathbf{n}) \qquad \text{und}$$
$$\mathit{X\text{-}MFP}_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \mathrm{post}^{\beta}_{c_{\mathbf{s}}}(\mathbf{n})$$

BB-*MaxFP*-Ansatz (2)

...wobei $\mathrm{pre}^{\beta}_{c_s}$ und $\mathrm{post}^{\beta}_{c_s}$ die größten Lösungen des Gleichungssystems

$$\begin{aligned} \mathit{pre}(\mathbf{n}) &= \begin{cases} c_{\mathbf{s}} & \text{falls } \mathbf{n} = \mathbf{s} \\ \bigsqcap \{ \mathit{post}(\mathbf{m}) \mid \mathbf{m} \in \mathit{pred}_{\mathbf{G}}(\mathbf{n}) \} & \text{sonst} \end{cases} \\ \mathit{post}(\mathbf{n}) &= [\![\mathbf{n}]\!]_{\beta}(\mathit{pre}(\mathbf{n})) \end{aligned}$$

bezeichnen.

BB-*MaxFP*-Ansatz (3)

...und ihre Fortsetzung auf EA-Ebene:

$$\forall\, c_s \in \mathcal{C}\, \forall\, n \in N.\ MaxFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \\ (\ N\text{-}MFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n),\ X\text{-}MFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n)\ )$$

mit

$$N\text{-}MFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \mathrm{pre}^{\iota}_{c_s}(n) \qquad \text{und}$$
$$X\text{-}MFP_{(\llbracket\ \rrbracket_\iota, c_s)}(n) =_{df} \mathrm{post}^{\iota}_{c_s}(n)$$

BB-*MaxFP*-Ansatz (4)

...wobei $pre^{\iota}_{c_s}$ und $post^{\iota}_{c_s}$ die größten Lösungen des Gleichungssystems

$$pre(n) = \begin{cases} pre^{\beta}_{c_s}(\texttt{block}(n)) \\ \quad \text{falls } n = start(\texttt{block}(n)) \\ \\ post(m) \\ \quad \text{sonst } (m \text{ ist hier der eindeutig} \\ \quad \text{bestimmte Vorgänger von } n \\ \quad \text{in } \texttt{block}(n)) \end{cases}$$

$$post(n) = [\![n]\!]_{\iota}(pre(n))$$

bezeichnen.

Verfügbarkeit von Ausdrücken I (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 I (2)

Das BB-Gleichungssystem von Phase I:

$$\text{BB-N-AVAIL}_\beta = \begin{cases} \textbf{false} & \text{falls } \beta = \textbf{s} \\ \prod_{\hat{\beta} \in \mathit{pred}(\beta)} \text{BB-X-AVAIL}_{\hat{\beta}} & \text{sonst} \end{cases}$$

$$\text{BB-X-AVAIL}_\beta = \text{BB-N-AVAIL}_\beta \cdot \text{BB-TRANSP}_\beta + \text{BB-XCOMP}_\beta$$

Verfügbarkeit von Ausdrücken I (3)

Phase II: Die Anweisungsebene

Lokale Prädikate (assoziiert mit EA-Knoten):

- $\text{COMP}_\iota(t)$: ι berechnet t.
- $\text{TRANSP}_\iota(t)$: 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:

$$\text{N-AVAIL}_\iota = \begin{cases} \text{BB-N-AVAIL}^\star_{\texttt{block}(\iota)} & \text{falls } \iota = \mathit{start}(\texttt{block}(\iota)) \\ \text{X-AVAIL}_{\mathit{pred}(\iota)} & \text{sonst} \\ & (\text{beachte: } |\mathit{pred}(\iota)| = 1) \end{cases}$$

$$\text{X-AVAIL}_\iota = \begin{cases} \text{BB-X-AVAIL}^\star_{\texttt{block}(\iota)} & \text{falls } \iota = \mathit{end}(\texttt{block}(\iota)) \\ (\text{N-AVAIL}_\iota + \text{COMP}_\iota) \cdot \text{TRANSP}_\iota & \text{sonst} \end{cases}$$

Verfügbarkeit von Ausdrücken II

...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:

$$\text{N-AVAIL}_\iota = \begin{cases} \textbf{false} & \text{falls } \iota = s \\ \prod_{\hat{\iota} \in pred(\iota)} \text{X-AVAIL}_{\hat{\iota}} & \text{sonst} \end{cases}$$

$$\text{X-AVAIL}_\iota = (\text{N-AVAIL}_\iota + \text{COMP}_\iota) \cdot \text{TRANSP}_\iota$$

Verfügbarkeit von Ausdrücken III

...für kantenbenannte EA-Graphen.

Lokale Prädikate (assoziiert mit EA-Kanten):

- $\text{COMP}_{\varepsilon}(t)$: Anweisung ι von Kante ε berechnet t.
- $\text{TRANSP}_{\varepsilon}(t)$: Anweisung ι von Kante ε modifiziert keinen Operanden von t.

Das EA-Gleichungssystem:

$$\texttt{Avail}_n = \begin{cases} \textbf{false} & \text{falls } n = s \\ \prod\limits_{m \in pred(n)} (\texttt{Avail}_m + \text{COMP}_{(m,n)}) \cdot \text{TRANSP}_{(m,n)} & \\ & \text{sonst} \end{cases}$$

Zwei weitere Beispiele

...zur Veranschaulichung des Einflusses der Flussgraphdarstellungsvariante:

- Konstantenausbreitung und -faltung (Constant Propagation and Folding)
- Schattenvariablenelimination (Faint Variable Elimination)

In der Folge Formulierung dieser Probleme für die Varianten:

- knotenbenannte Basisblockgraphen und
- kantenbenannte Einzelanweisungsgraphen

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 (10.1, Substitutionslemma)

$$\forall\, t \in \mathbf{T}\ \forall\, \sigma \in \Sigma\ \forall\, \iota \in \mathcal{I}.\ \mathcal{E}(\delta_\iota(t))(\sigma) = \mathcal{E}(t)(\theta_\iota(\sigma))$$

Beweis induktiv über den Aufbau von t.

Einfache Konstanten für den EA-Fall (1)

...für kantenbenannte Einzelanweisungsgraphen.

- $\mathrm{CP}_n \in \Sigma$
- $\sigma_0 \in \Sigma$ Anfangszusicherung

Das EA-Gleichungssystem:

$$\forall\, v \in \mathbf{V}.\ \mathrm{CP}_n = \begin{cases} \sigma_0(v) & \text{falls } n = s \\ \bigsqcap \{\, \mathcal{E}(\delta_{(m,n)}(v))(\mathrm{CP}_m) \mid m \in \mathit{pred}(n) \,\} & \text{sonst} \end{cases}$$

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 (10.2, 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:

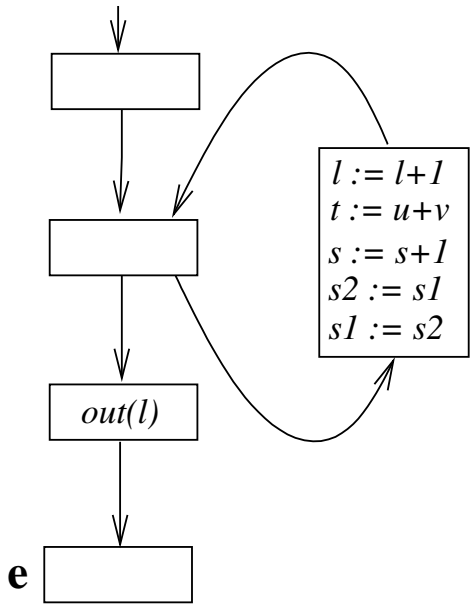
$$\text{N-CP}_\iota \quad = \quad \begin{cases} \text{BB-N-CP}^\star_{\texttt{block}(\iota)} \\ \quad \text{falls } \iota = \mathit{start}(\texttt{block}(\iota)) \\ \text{X-CP}_{\mathit{pred}(\iota)} \\ \quad \text{sonst (beachte: } |\mathit{pred}(\iota)| = 1) \end{cases}$$

$$\forall\, v \in \mathbf{V}.\ \text{X-CP}_\iota(v) \quad = \quad \begin{cases} \text{BB-X-CP}^\star_{\texttt{block}(\iota)}(v) \\ \quad \text{falls } \iota = \mathit{end}(\texttt{block}(\iota)) \\ \mathcal{E}(\delta_\iota(v))(\text{N-CP}_\iota) \quad \text{sonst} \end{cases}$$

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.

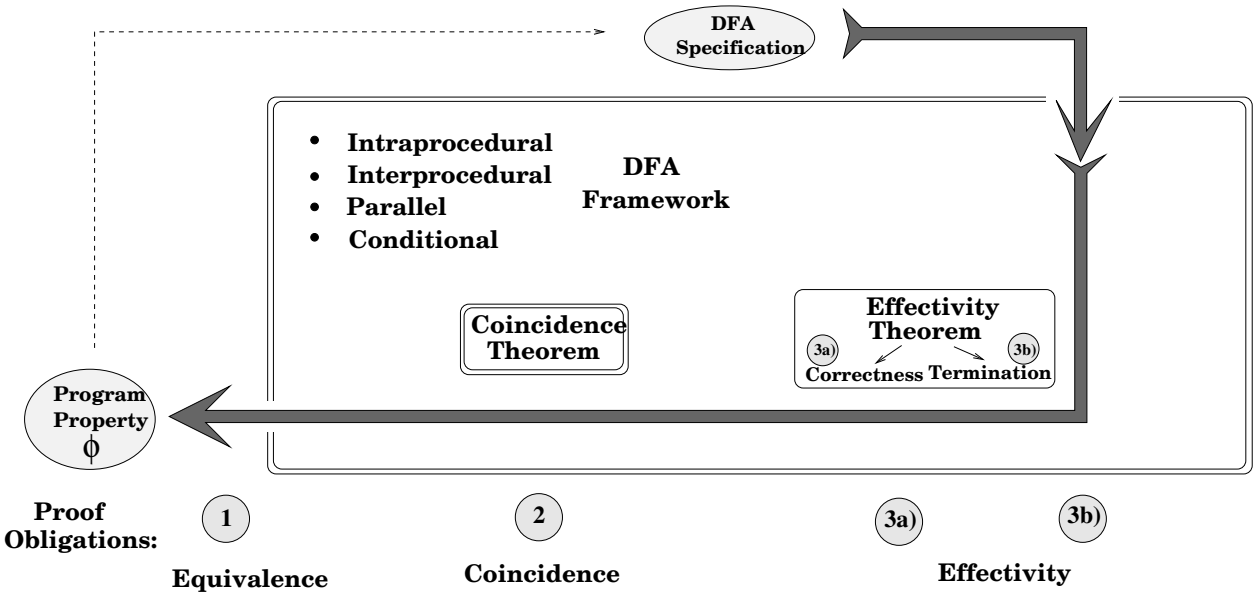
Abschlussbemerkung

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 zur Folge haben.

Ergänzende, weiterführende und vertiefende 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

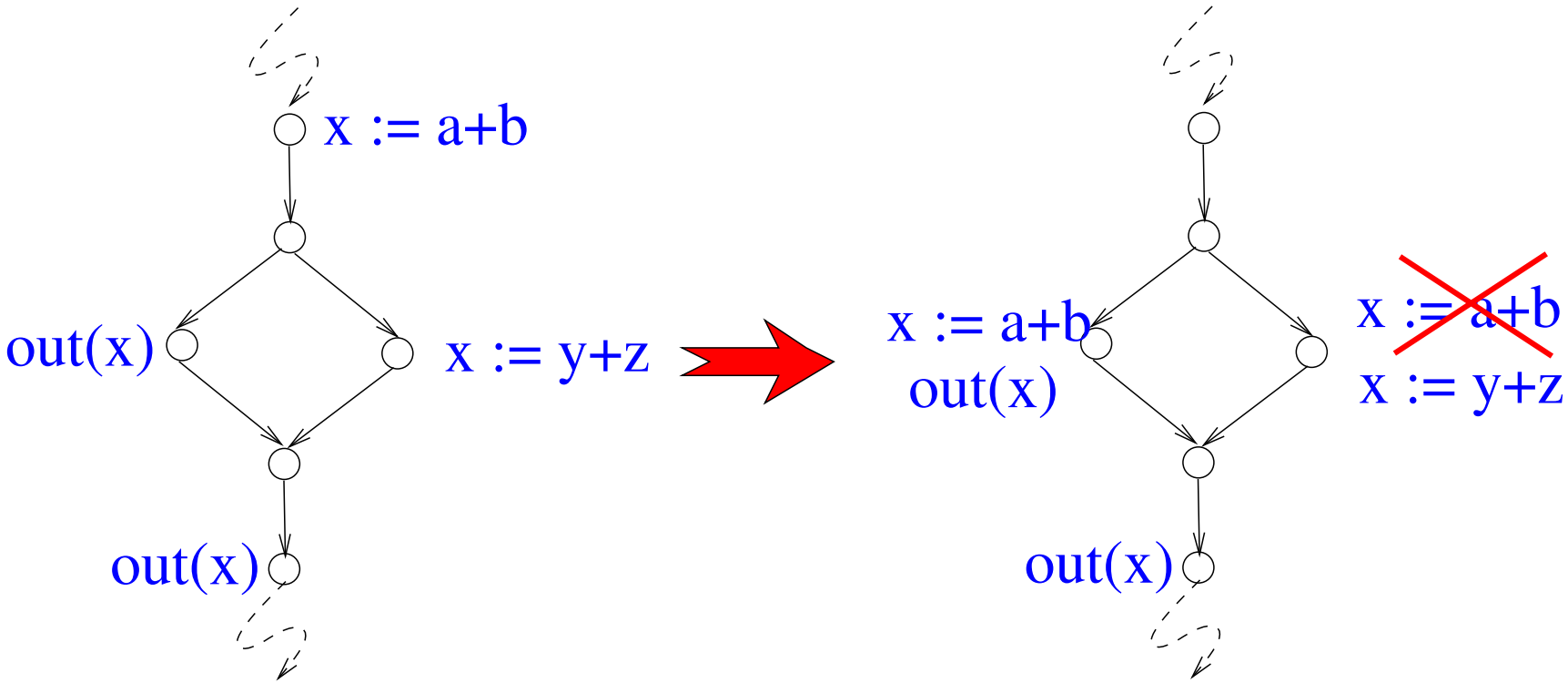
Elimination partiell toter Anweisungen

Kapitel 11.1

Motivation

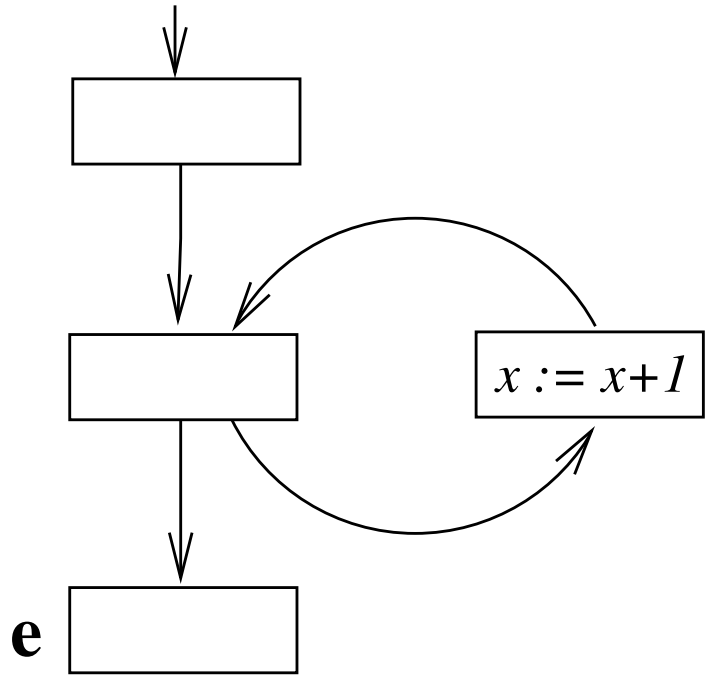
Elimination partiell toter Anweisungen

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

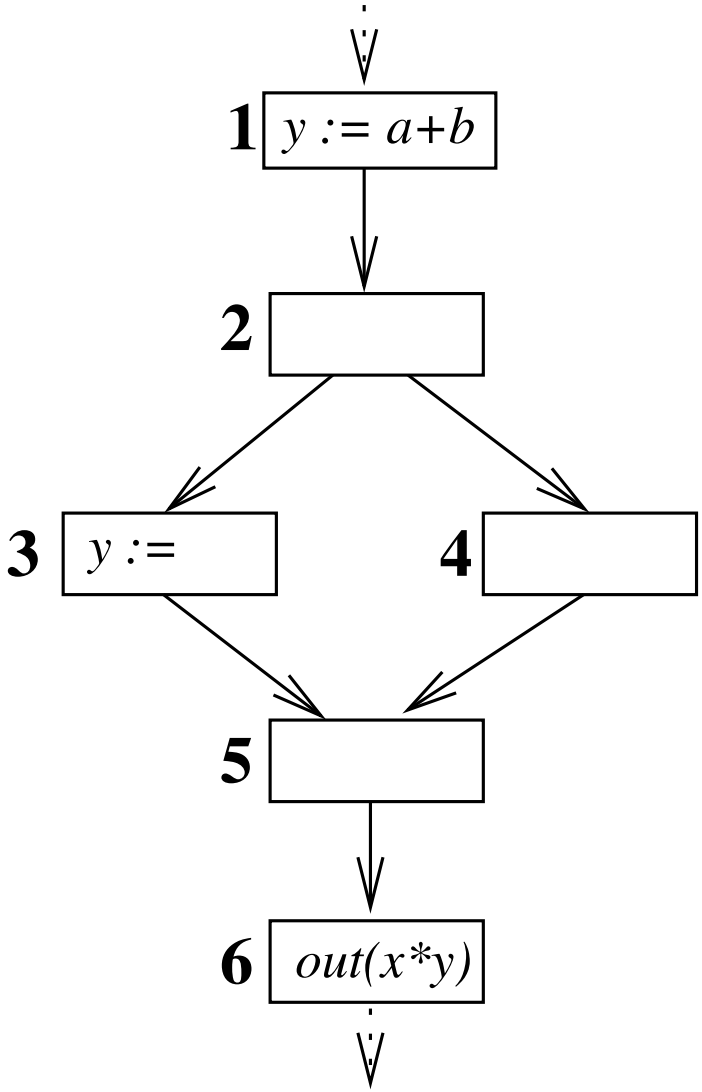
  $\rightsquigarrow$ Partial Dead-Code Elimination (PDCE)

- Elimination partiell schattenhaften Codes

  $\rightsquigarrow$ Partial Faint-Code Elimination (PFCE)

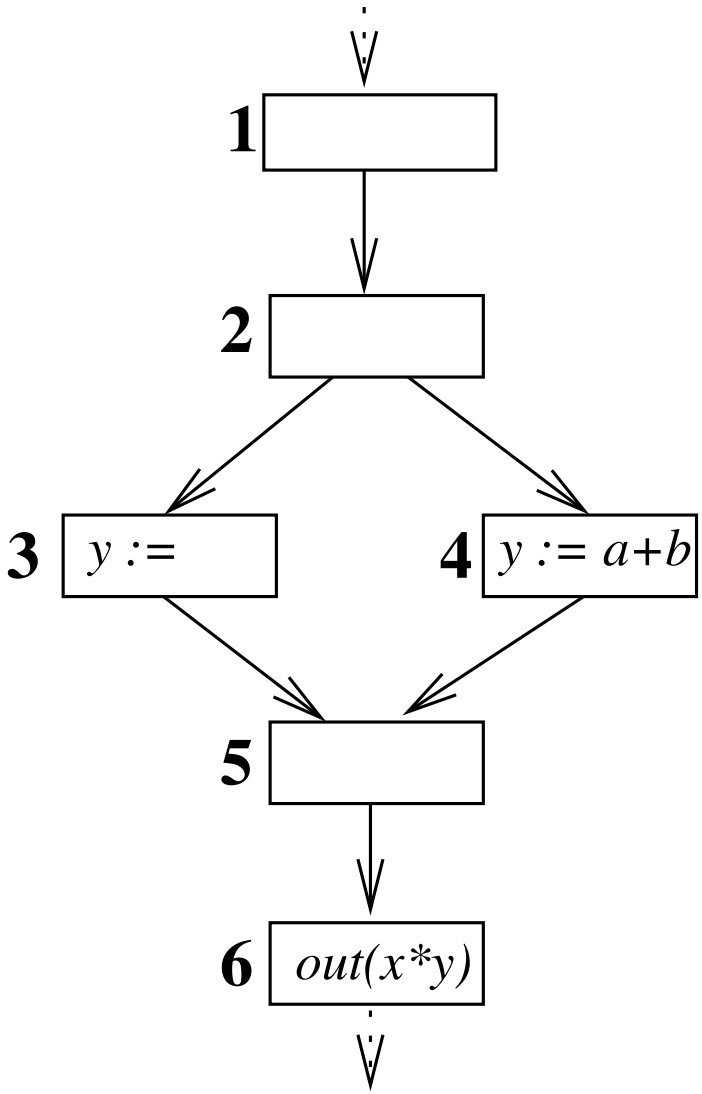
Elimination partiell toten Codes (1)

Ausgangsprogramm:



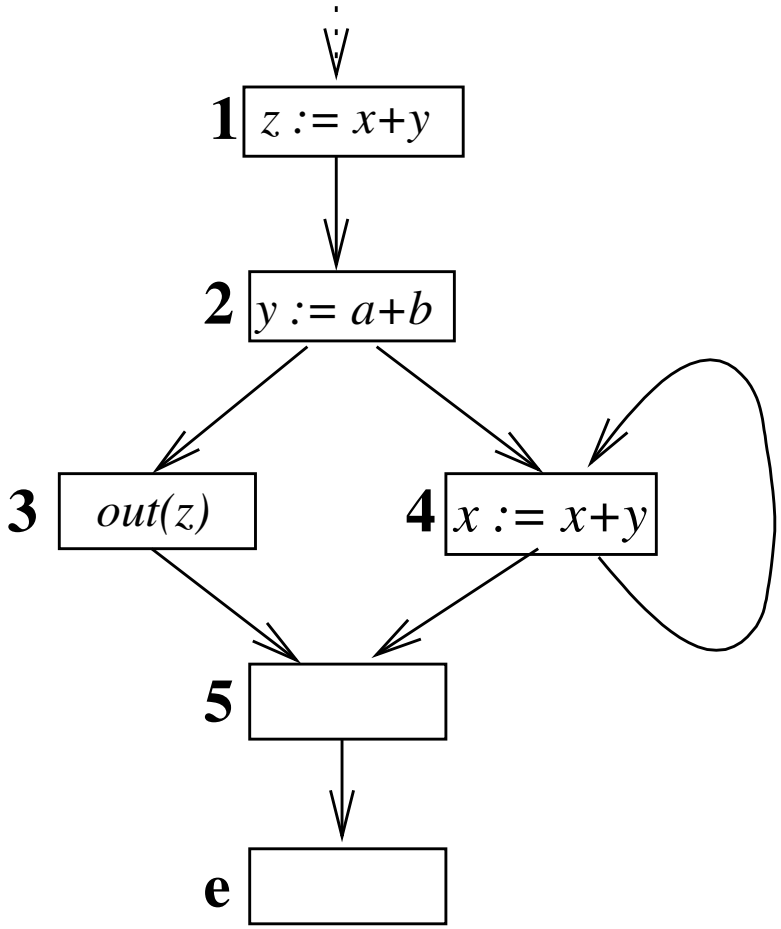
Elimination partiell toten Codes (2)

Transformiertes/optimiertes Programm:



Elimination partiell schattenhaften Codes (1)

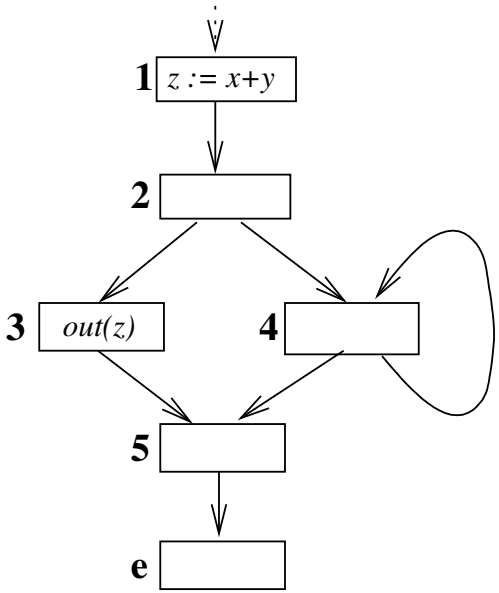
Ausgangsprogramm:



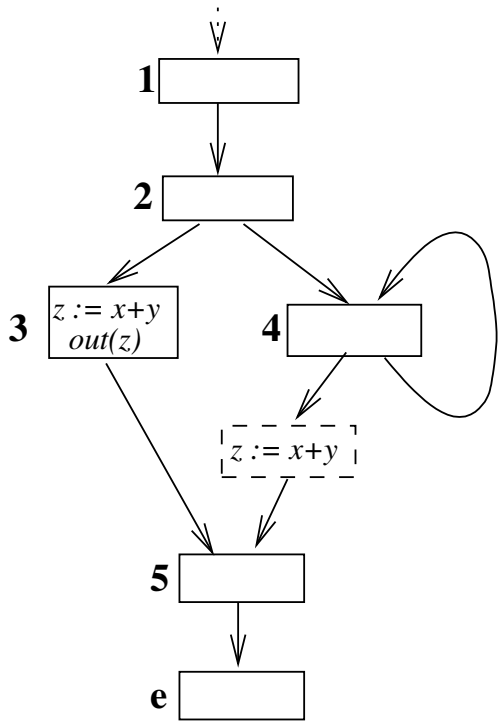
Elimination partiell schattenhaften Codes (2)

Transformiertes/optimiertes Programm:

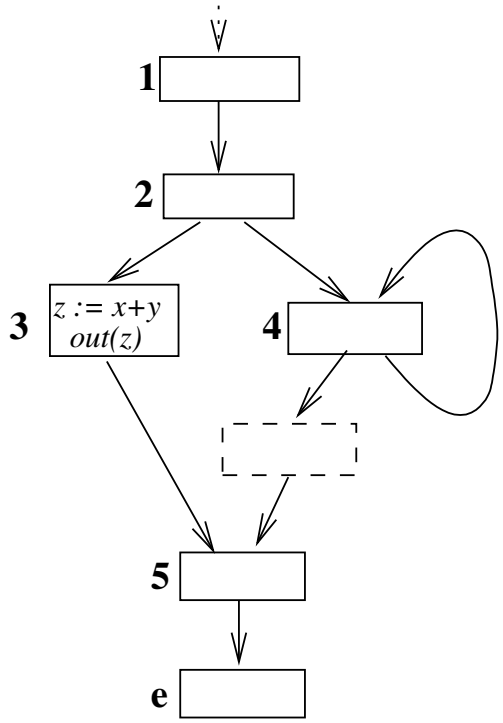
Elimination schattenhaften Codes



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

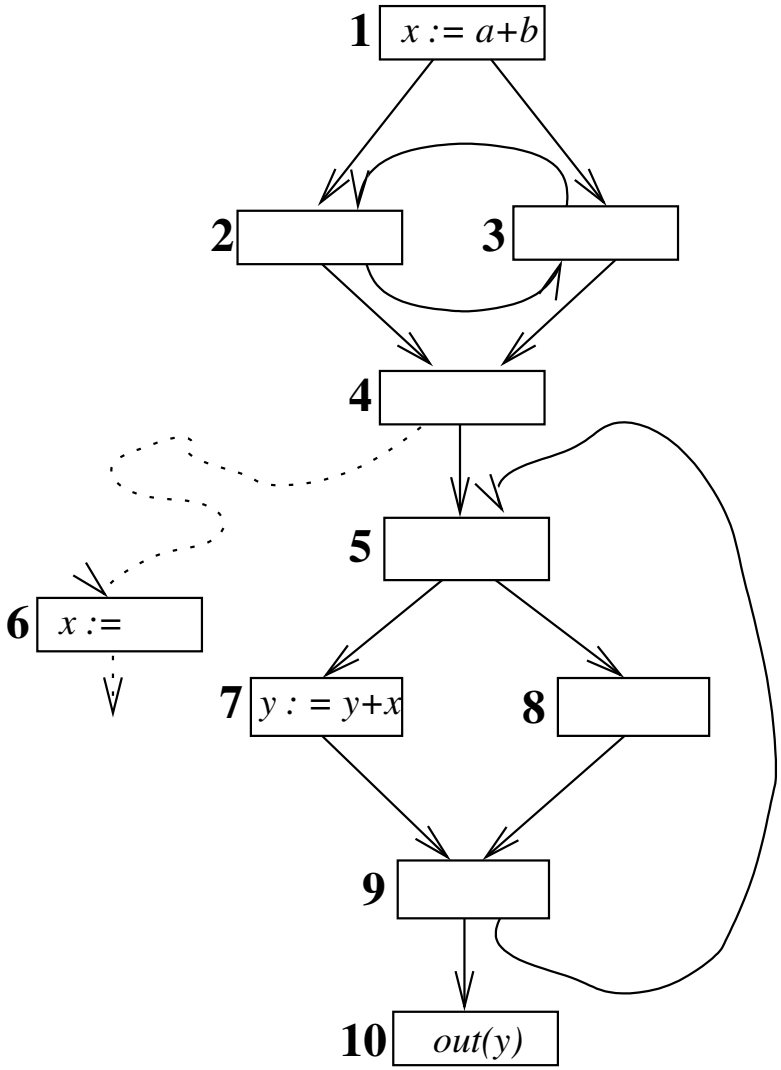
zu verstehen.

Kapitel 11.2

PDCE/PFCE: Transformation und Optimalität

Kritische Kanten (1)

...be-/verhindern die Transformation:

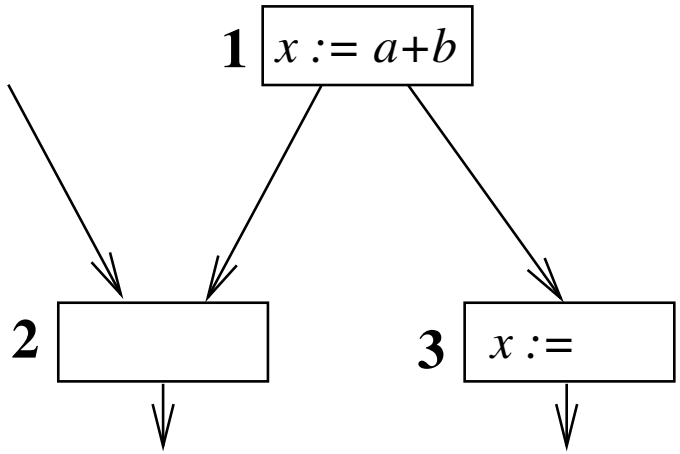


Kritische Kanten (2)

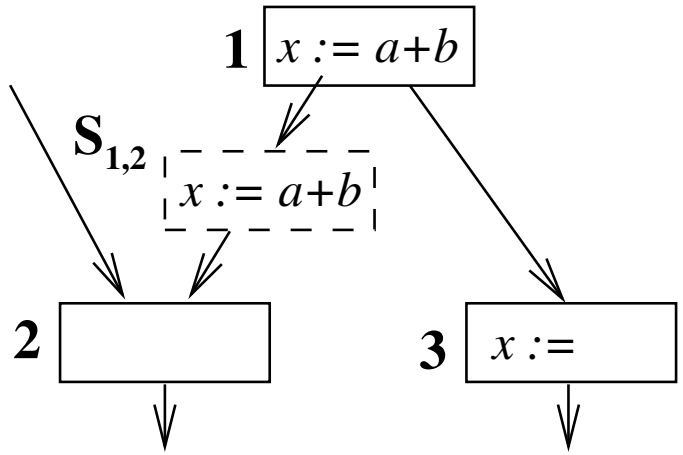
Definition (11.2.1, Kritische Kanten)

Eine Kante heißt kritisch gdw sie von einem Knoten mit mehr als einem Nachfolger zu einem Knoten mit mehr als einem Vorgänger führt.

a)



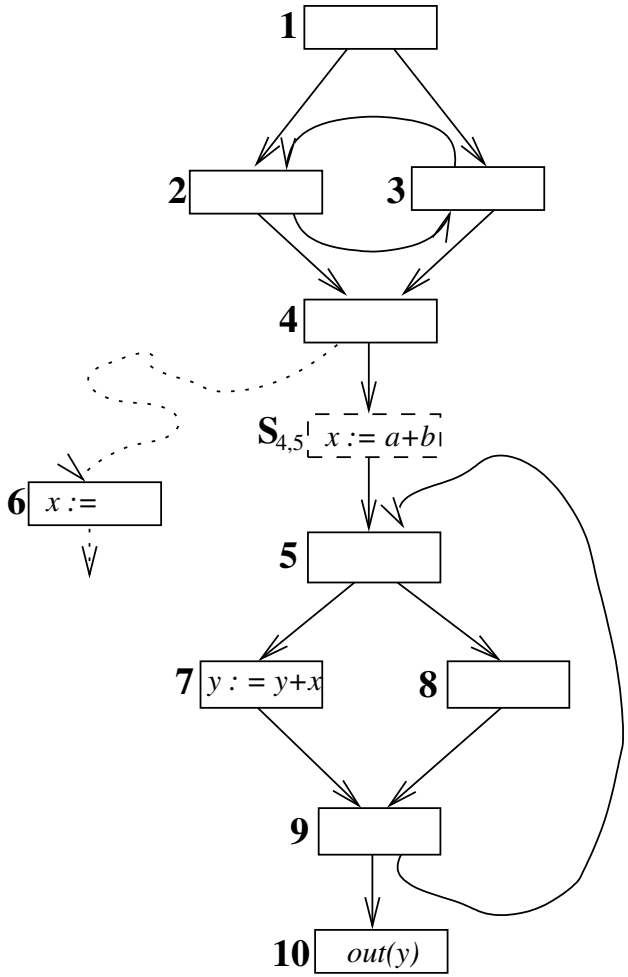
b)



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

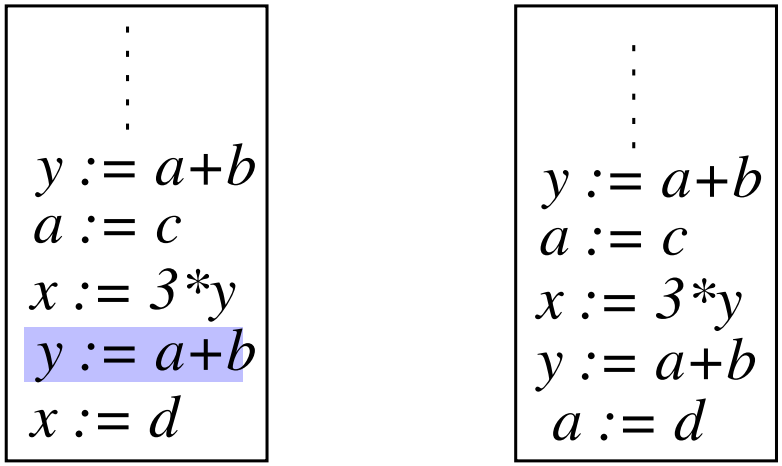
Definition (11.2.2, Anweisungssenkung)

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)

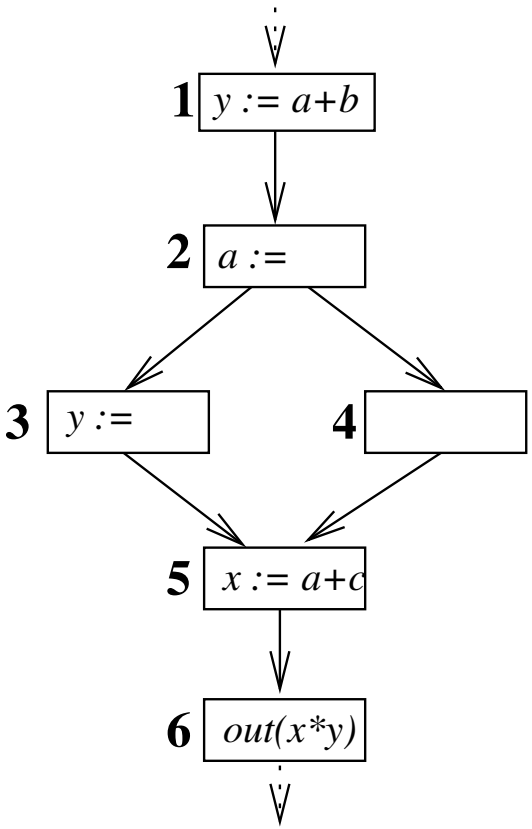
Effekte zweiter Ordnung

Effekte zweiter Ordnung (engl. second-order effects):

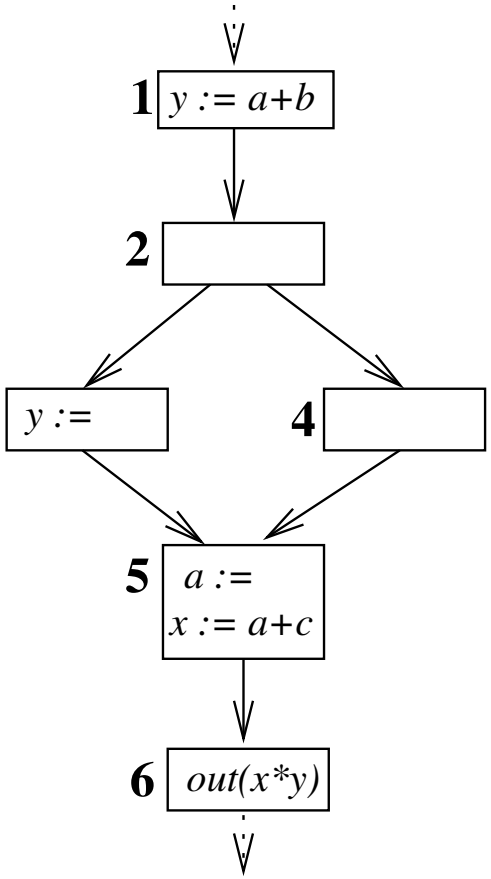
- Senkungs-Eliminations-Effekte (Zieleffekt)
  $\rightsquigarrow$ Sinking-Elimination effects (SE)
- Senkungs-Senkungs-Effekte
  $\rightsquigarrow$ Sinking-Sinking effects (SS)
- Eliminations-Senkungs-Effekte
  $\rightsquigarrow$ Elimination-Sinking effects (ES)
- Eliminations-Eliminations-Effekte (Zieleffekt)
  $\rightsquigarrow$ Elimination-Elimination effects (EE)

Beispiel eines Senkungs-Senkungs-Effekts

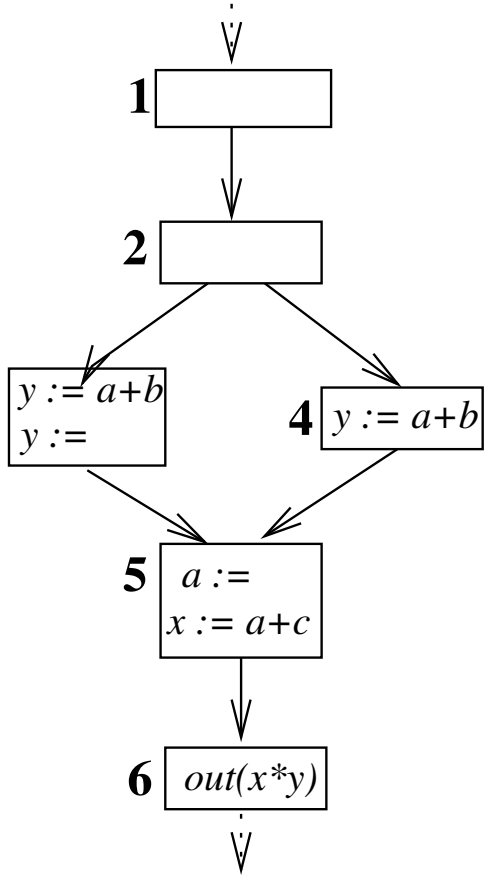
Ausgangsprogramm



1. Senkung

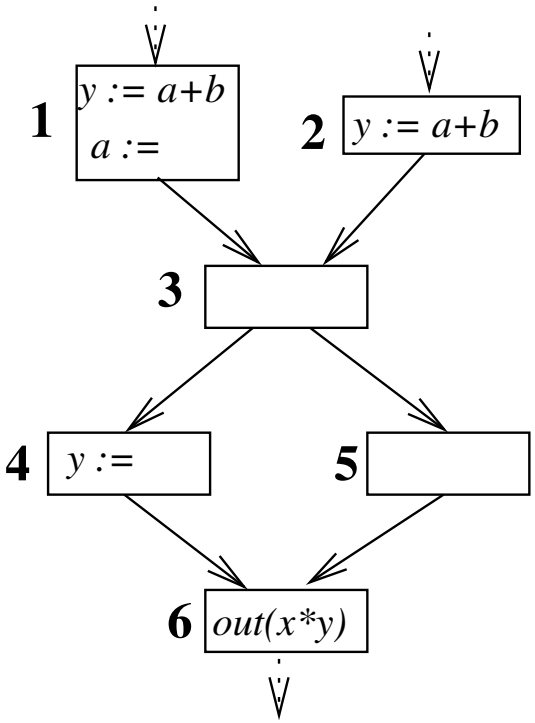


2. Senkung

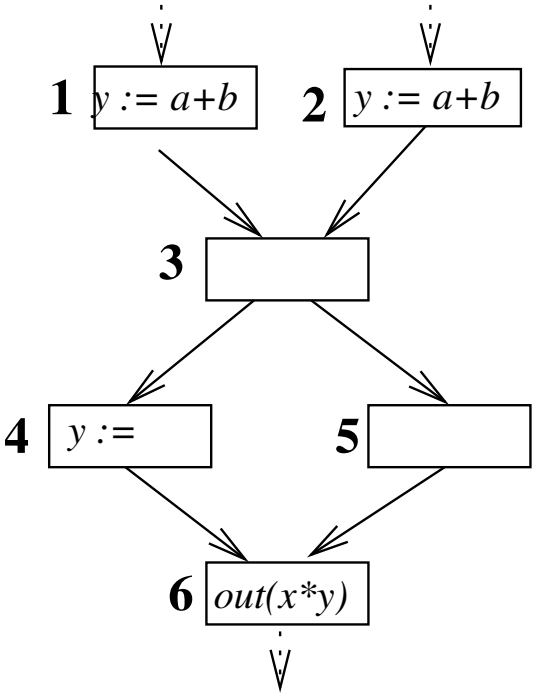


Beispiel eines Eliminations-Senkungs-Effekt

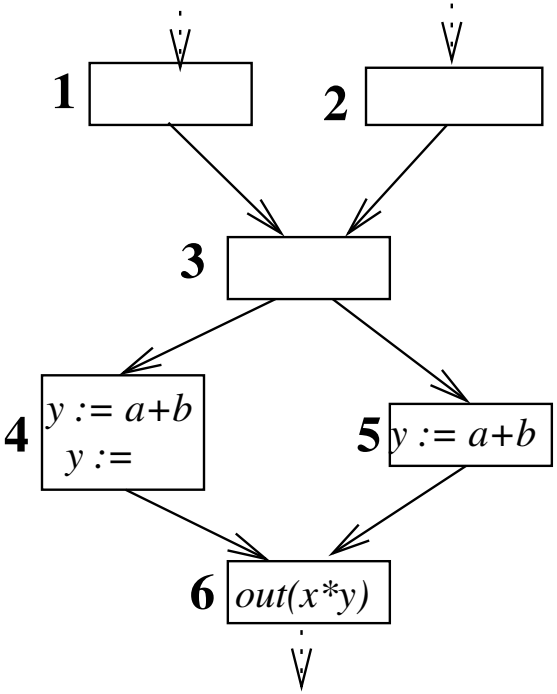
Ausgangsprogramm



Elimination

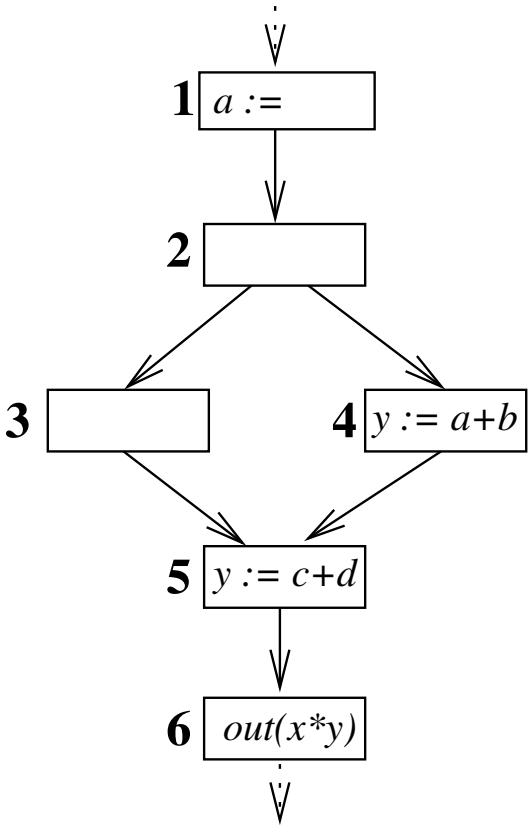


Senkung

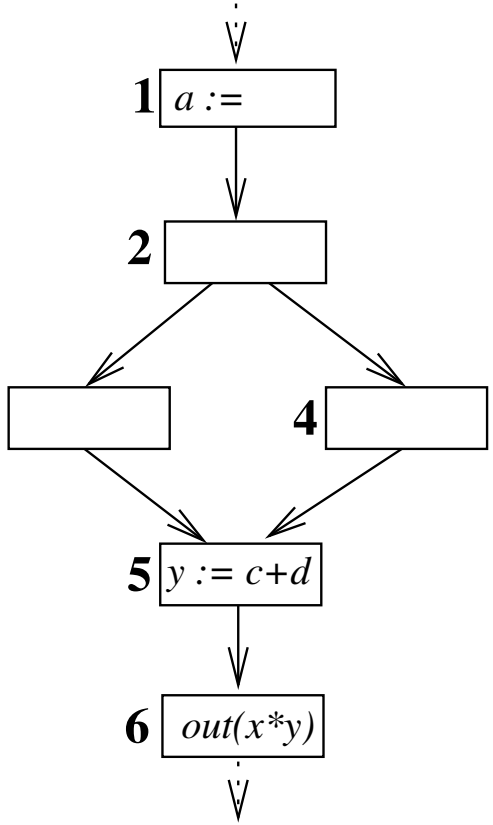


Beispiel eines Eliminations-Eliminations-Effekt

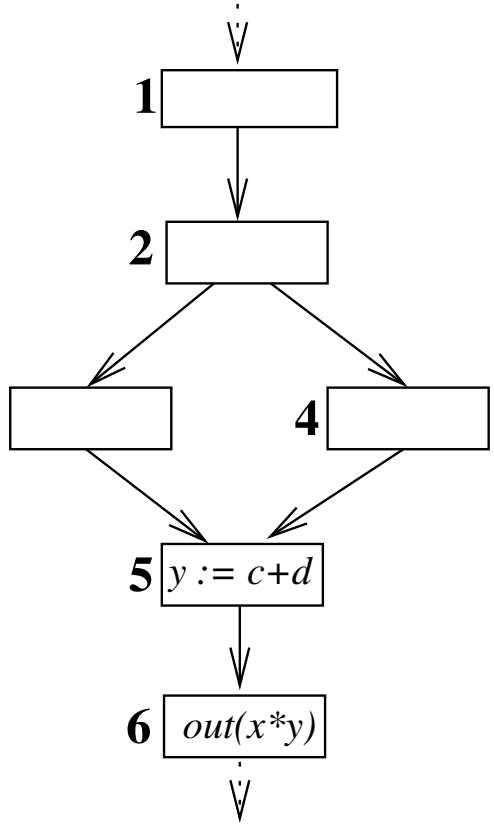
Ausgangsprogramm



1. Elimination

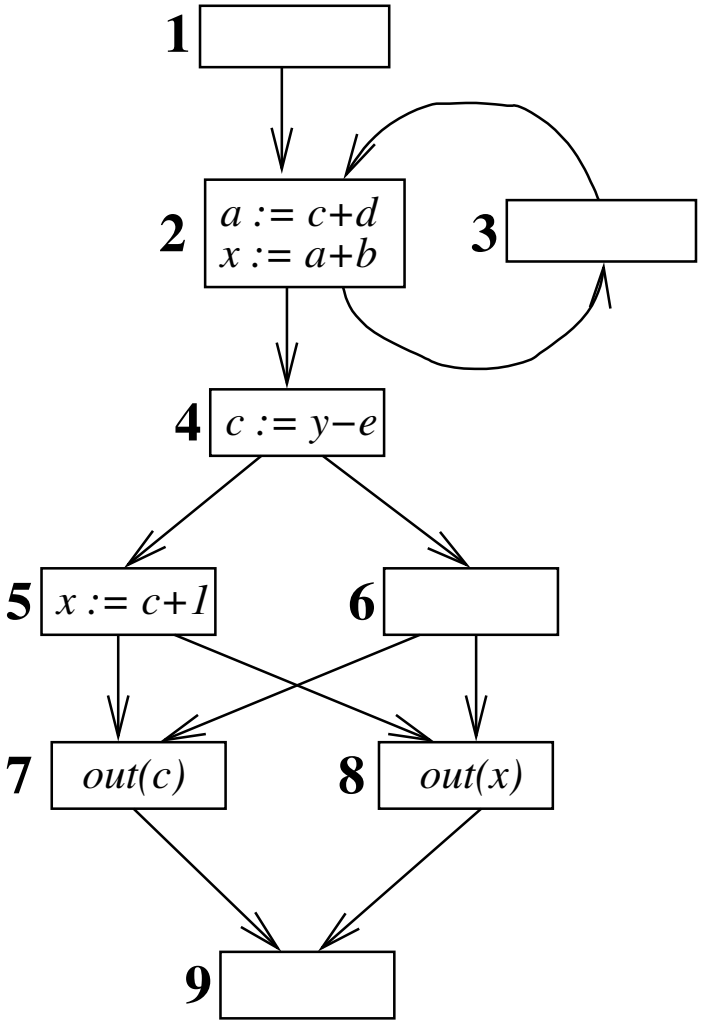


2. Elimination



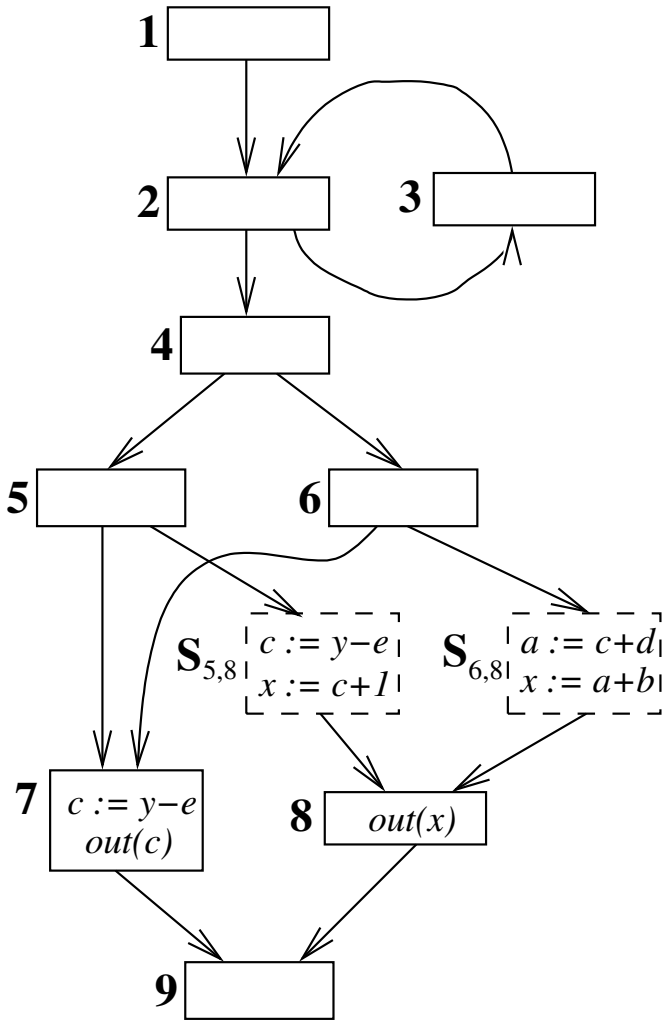
Kombinationswirkung von Effekten zweiter Ordnung (1)

Ausgangsprogramm:



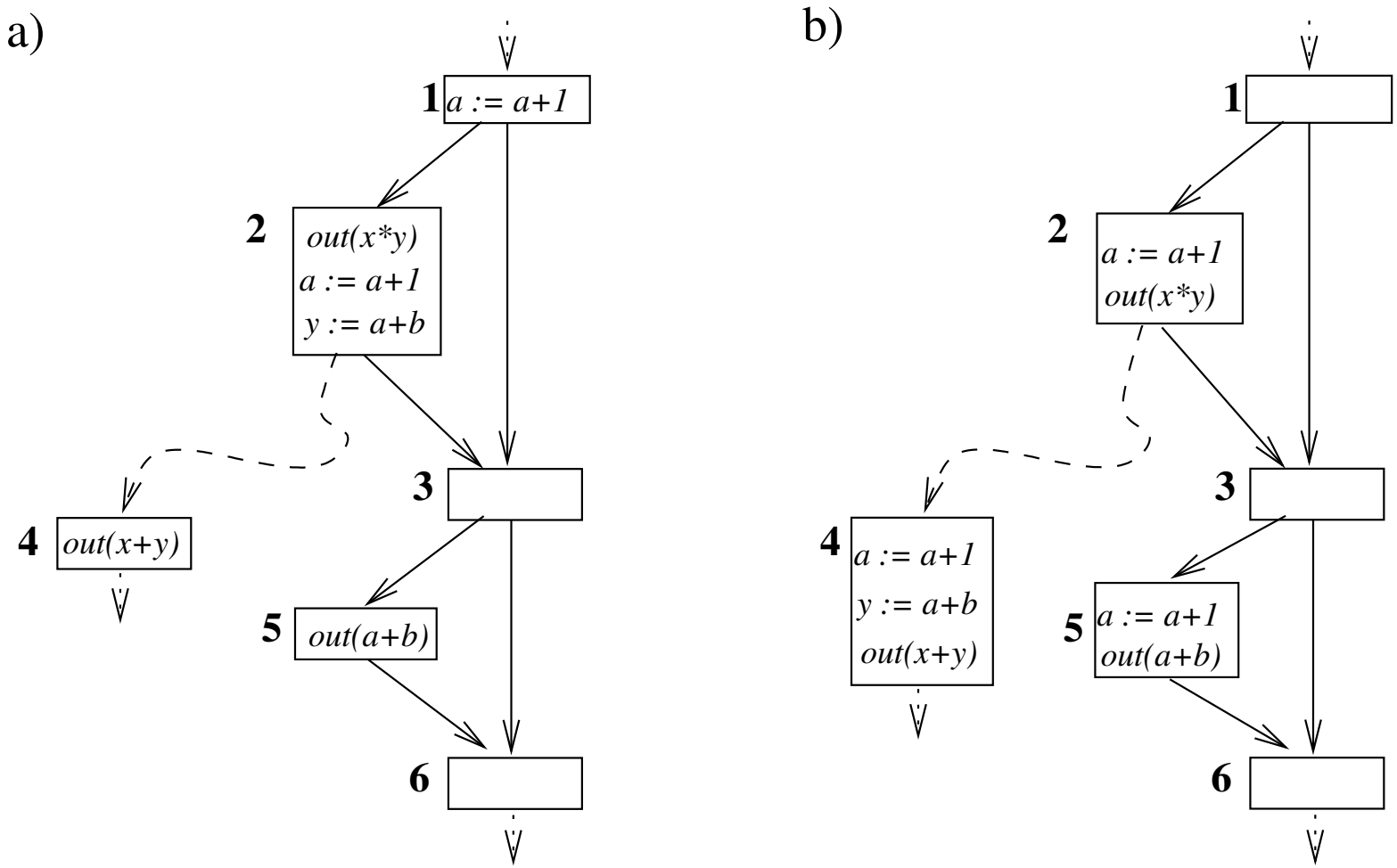
Kombinationswirkung von Effekten zweiter Ordnung (2)

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.

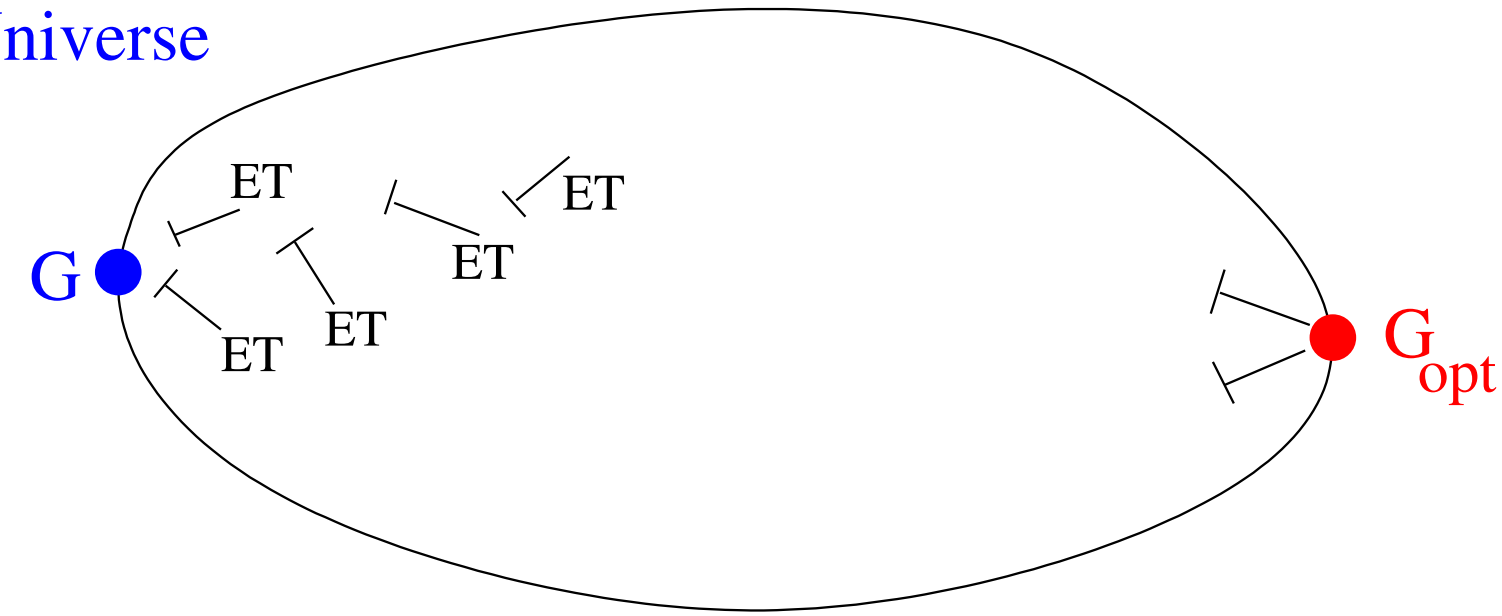
- $\mathcal{G}_\tau =_{df} \{ G' \mid G \vdash_\tau^* G' \}$:

  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 9.4.

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

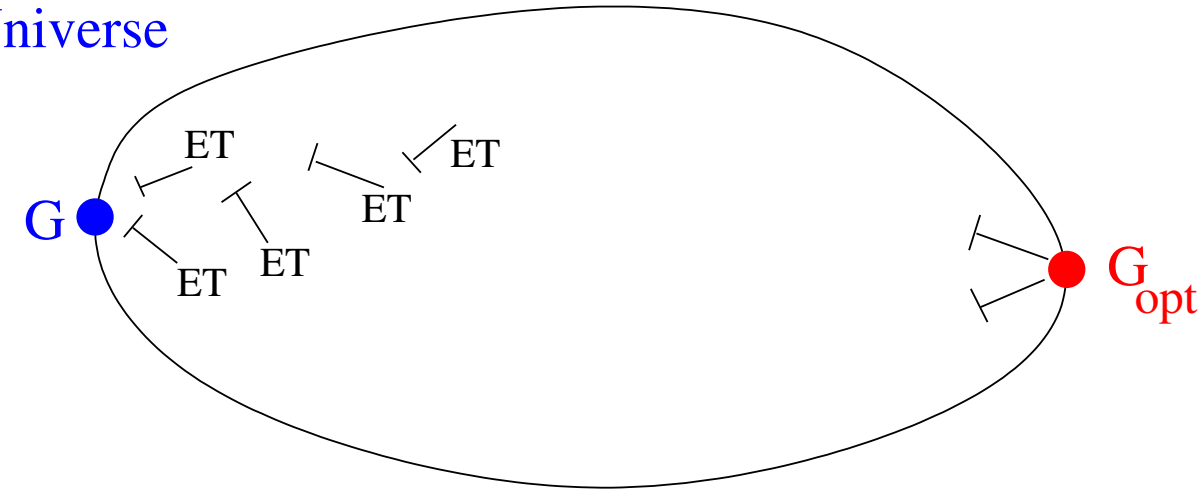
PDCE/PFCE-Ableitungsrelation $\vdash$:

- 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 (11.2.14, Konfluenz und Terminierung)

Die PDCE/PFCE-Ableitungsrelationen $\vdash_{AS,DCE}$ und $\vdash_{AS,FCE}$ sind konfluent und terminierend.



PDCE/PDFE-Korrektheit und -Optimalität

Theorem (11.2.12, Korrektheit)

1. $G_{pdce} \in \mathcal{G}_{PDCE}$
2. $G_{pfce} \in \mathcal{G}_{PFCE}$

Theorem (11.2.13, 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)

$$\begin{aligned}\text{N-FAINT}_\iota(x) \quad = \quad & \overline{\text{REL-USED}_\iota(x)} \ * \\ & (\text{X-FAINT}_\iota(x) \ + \ \text{MOD}_\iota(x)) \ * \\ & (\text{X-FAINT}_\iota(\mathit{LhsVar}_\iota) + \overline{\text{ASS-USED}_\iota(x)})\end{aligned}$$

$$\text{X-FAINT}_\iota(x) \quad = \quad \prod_{\hat{\iota} \in \mathit{succ}(\iota)} \text{N-FAINT}_{\hat{\iota}}(x)$$

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:

$$\text{N-DELAYED}_n = \begin{cases} \textbf{false} & \text{if} \quad n = \textbf{s} \\ \prod_{m \in pred(n)} \text{X-DELAYED}_m & \text{otherwise} \end{cases}$$

$$\text{X-DELAYED}_n = \text{LOC-DELAYED}_n + \text{N-DELAYED}_n * \overline{\text{LOC-BLOCKED}_n}$$

Einsetzungspunkte gesenkter Anweisungen

Die sich aus der AS-Analyse ergebenden Einsetzungspunkte:

$$\text{N-INSERT}_n =_{df} \text{N-DELAYED}^*_n * \text{LOC-BLOCKED}_n$$

$$\text{X-INSERT}_n =_{df} \text{X-DELAYED}^*_n * \sum_{m \in succ(n)} \overline{\text{N-DELAYED}^*_m}$$

wobei N-DELAYED^*_n und X-DELAYED^*_n die größten Lösungen des Anweisungssenkungs-Gleichungssystems bezeichnen.

Beachte: Die Berechnung der Einsetzungspunkte erfordert keine DFA!

Ergänzende, weiterführende und vertiefende 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 of Programming Languages (POPL'94), 1994.

Ergänzende, weiterführende und vertiefende 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.


R. J. Mintz, G. A. Fisher, M. Sharir. *The Design of a Global Optimizer*. In Proceedings of the ACM SIGPLAN'79 Symposium on Compiler Construction (SCC'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

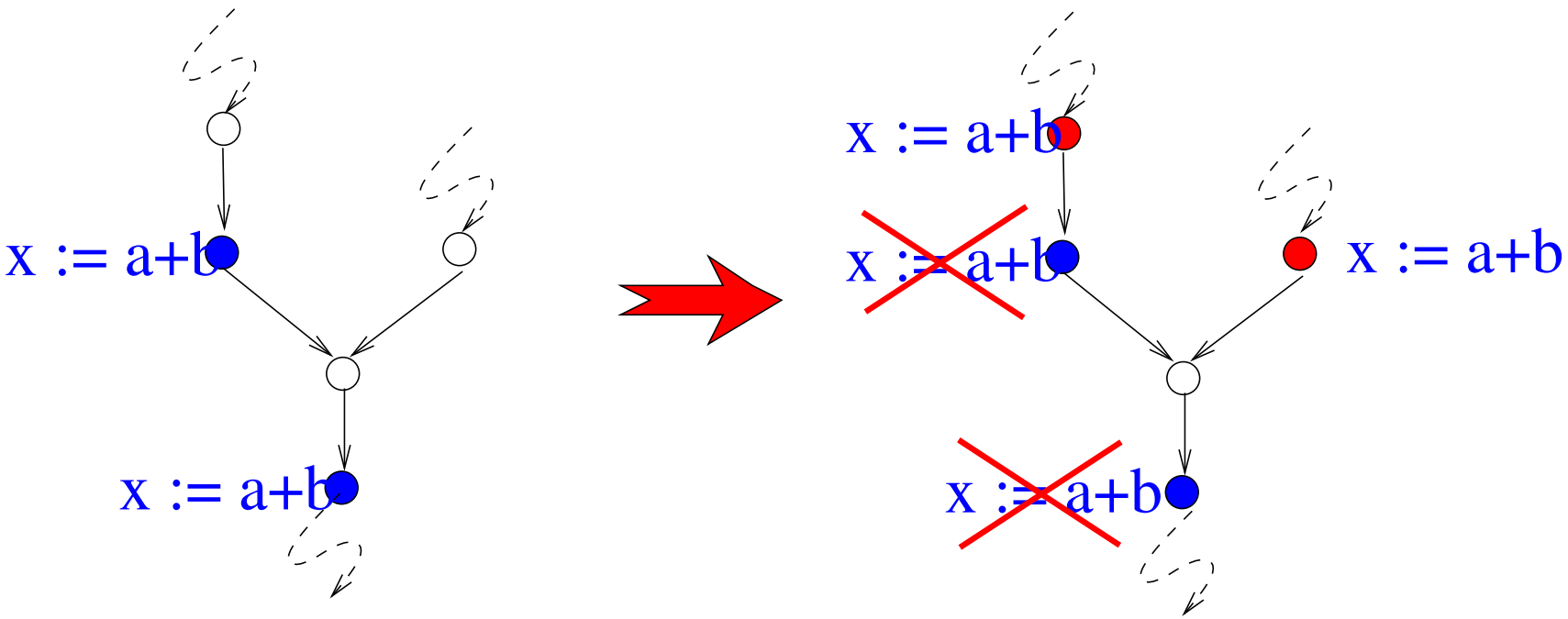
Elimination partiell redundanter Anweisungen

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

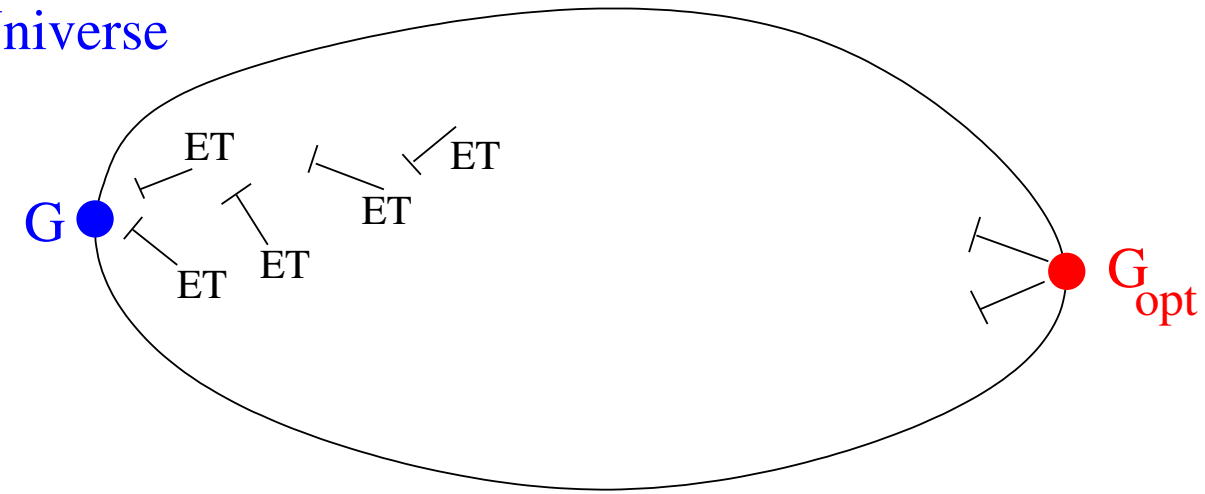
PRAE-Ableitungsrelation $\vdash$:

- 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)

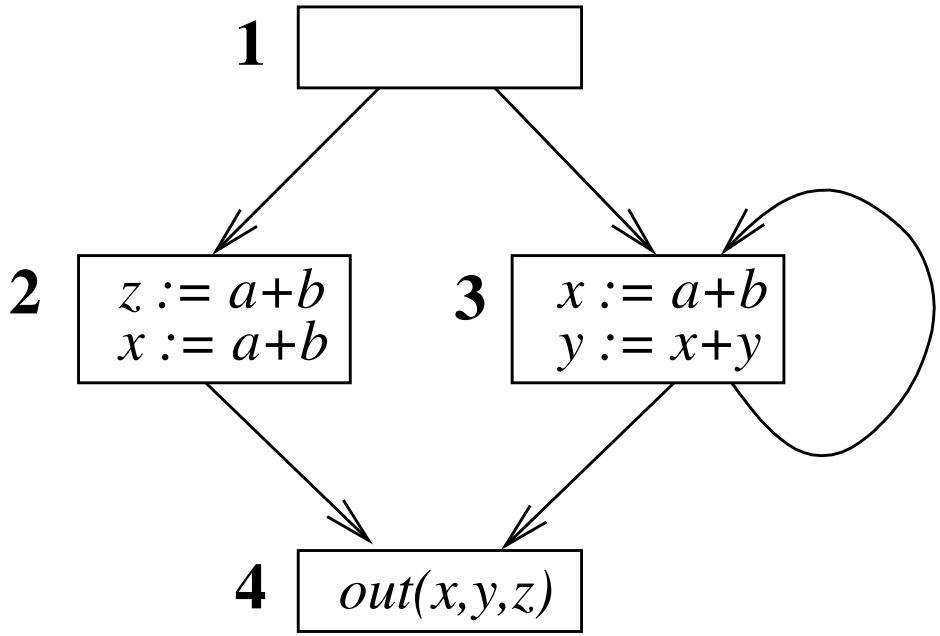
Kombinierte PRE/AE-Transformation: EAM

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

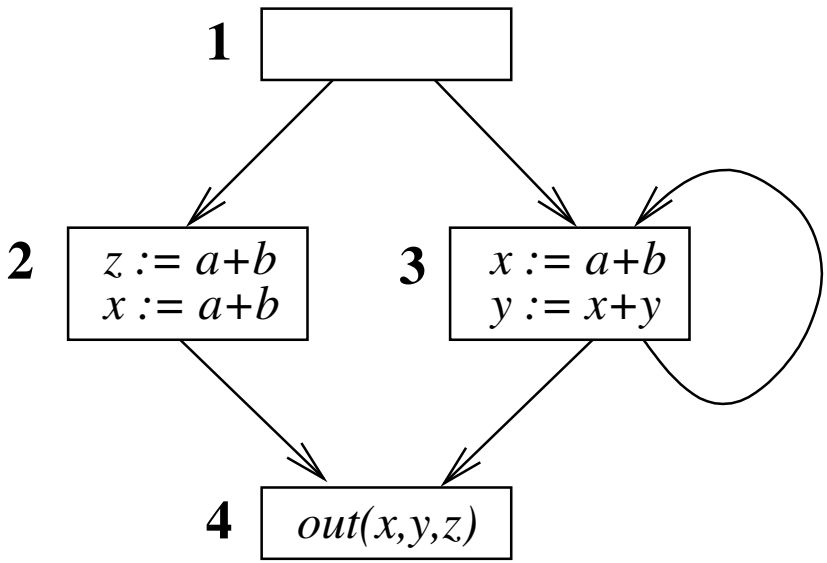
...illustrieren wir die unterschiedlichen Effekte dieser Transformationen anhand eines gemeinsamen Beispiels:



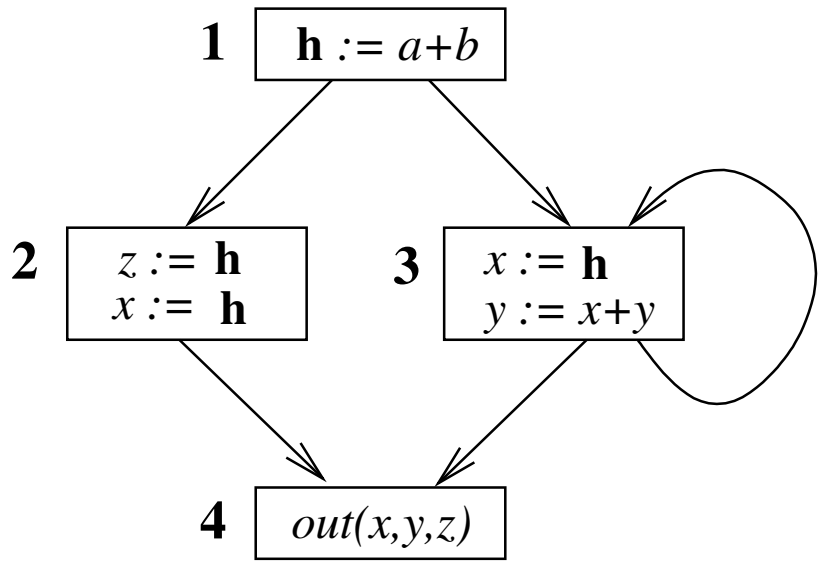
Elimination partiell redundanter Ausdrücke

Der PREE-Effekt auf das laufende Beispiel:

a)



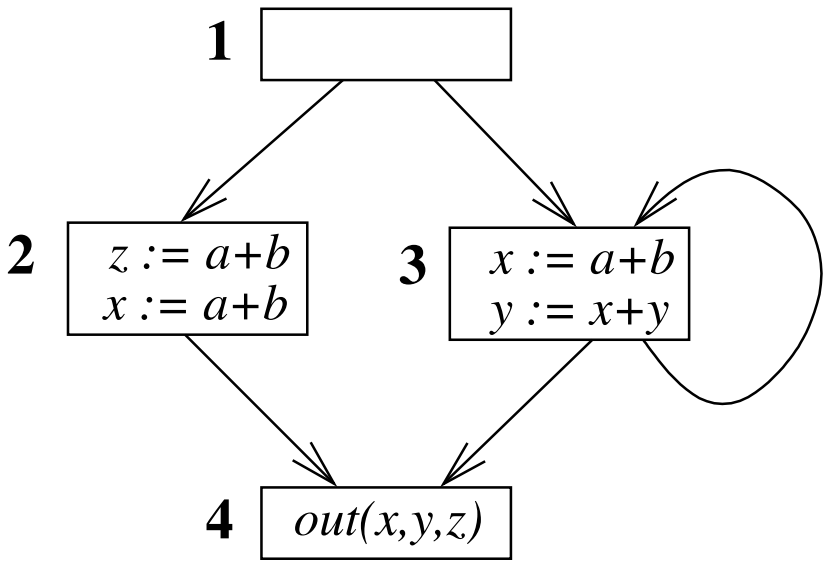
b)



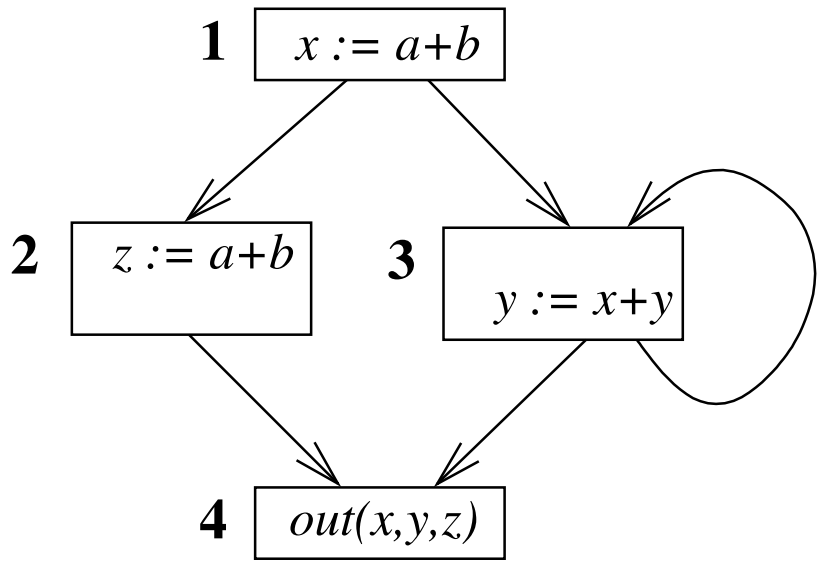
Elimination partiell redundanter Anweisungen

Der PRAE-Effekt auf das laufende Beispiel:

a)

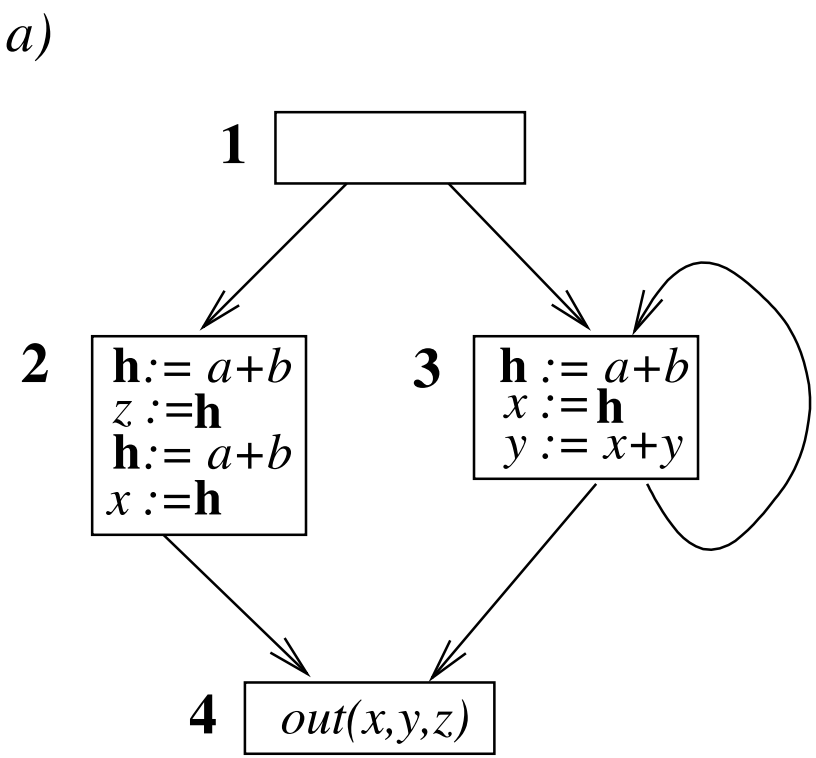


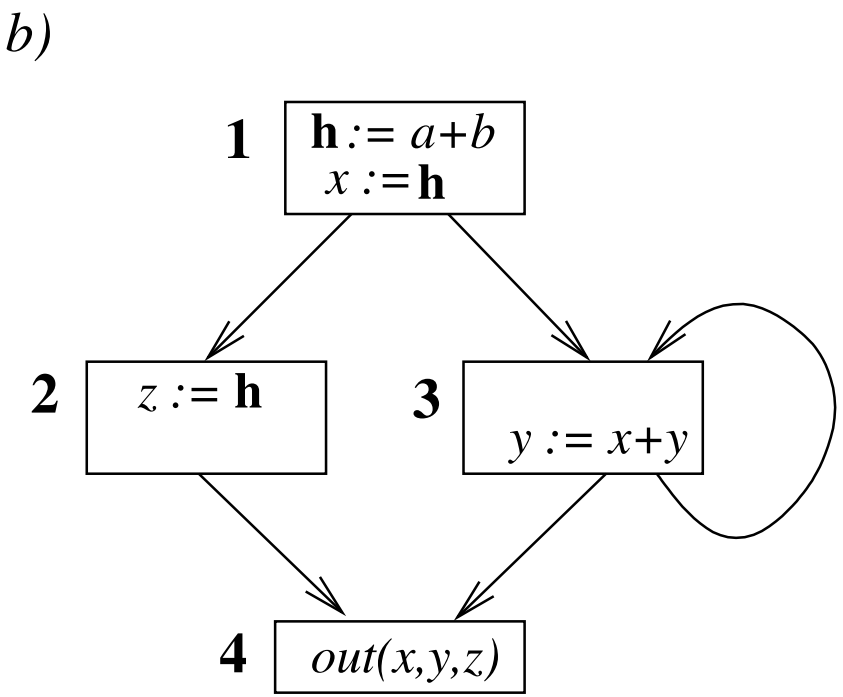
b)



Kombinierte Elimination partiell redundanter Ausdrücke und Anweisungen

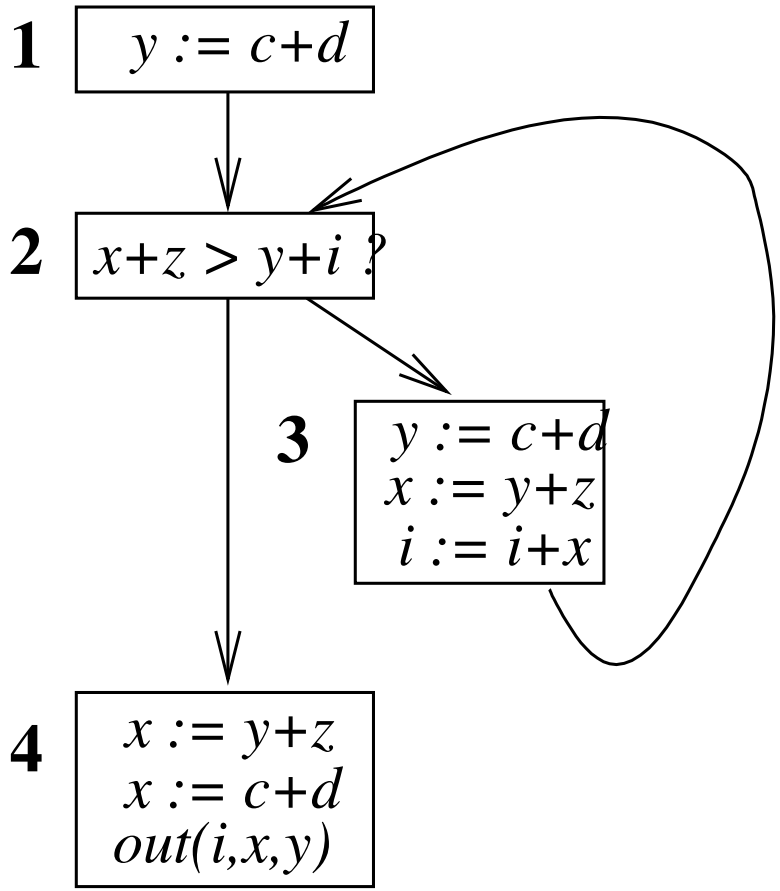
Der EAM(=PREAE)-Effekt auf das laufende Beispiel:





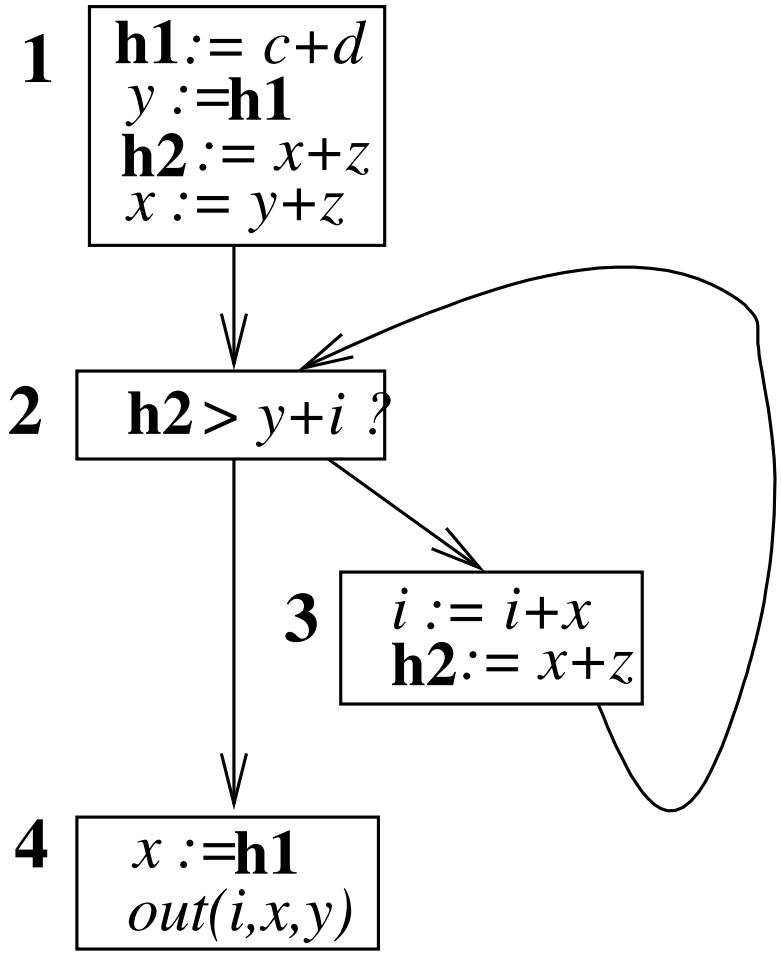
EAM anhand eines größeren Beispiels (1)

Ausgangsprogramm:



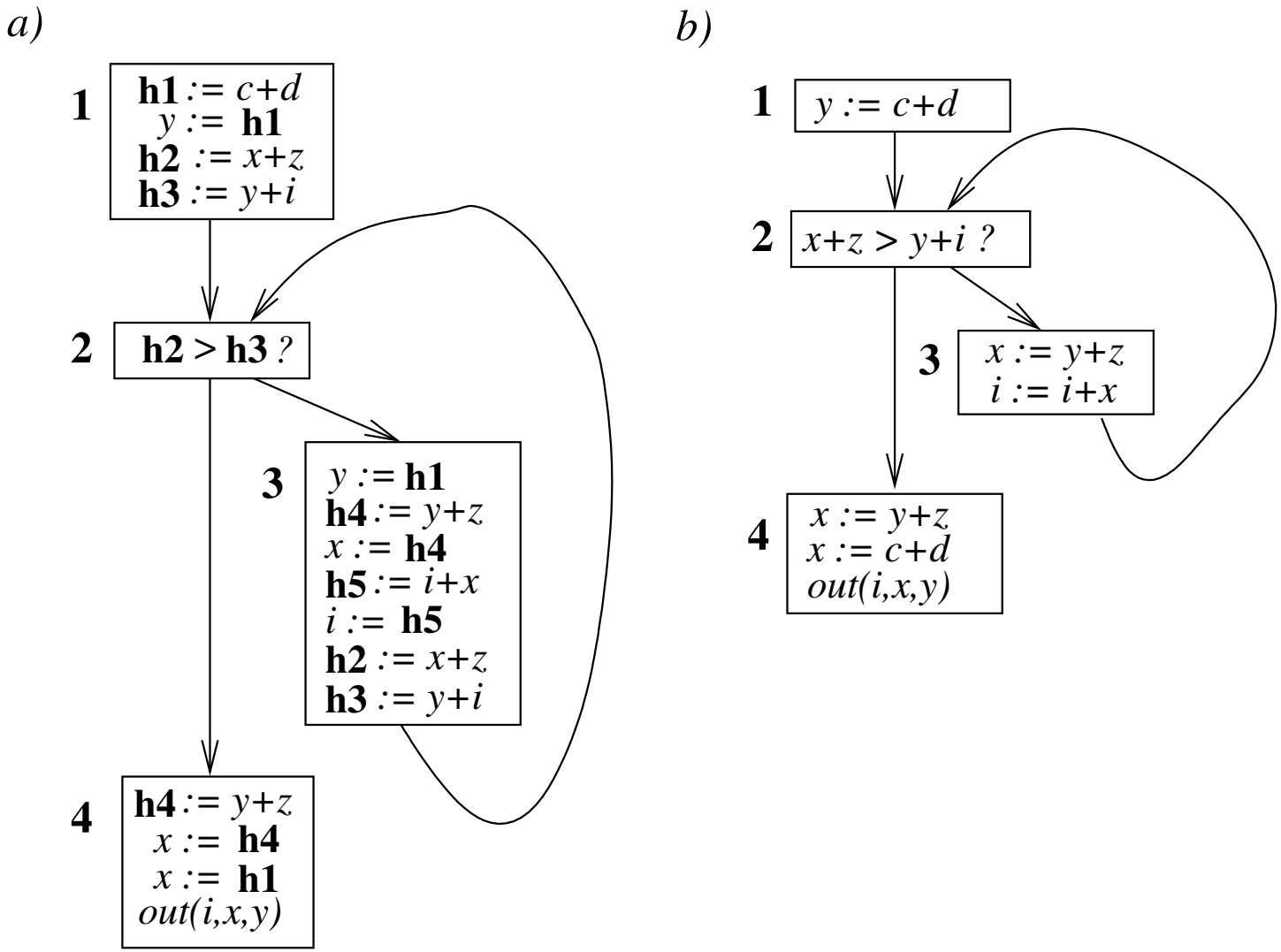
EAM anhand eines größeren Beispiels (2)

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":

$$\text{EAM} > \text{PREE} + \text{PRAE}$$

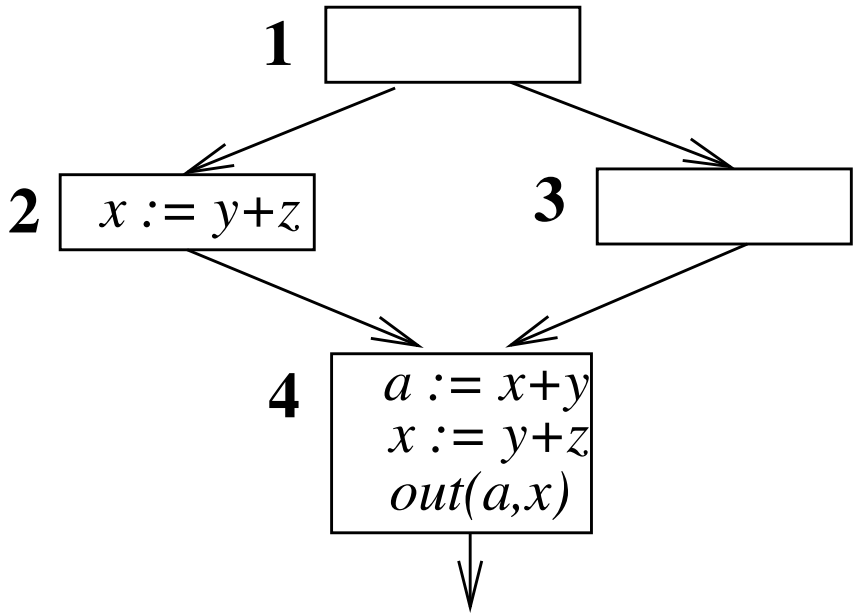
Effekte zweiter Ordnung für (E)AM

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)

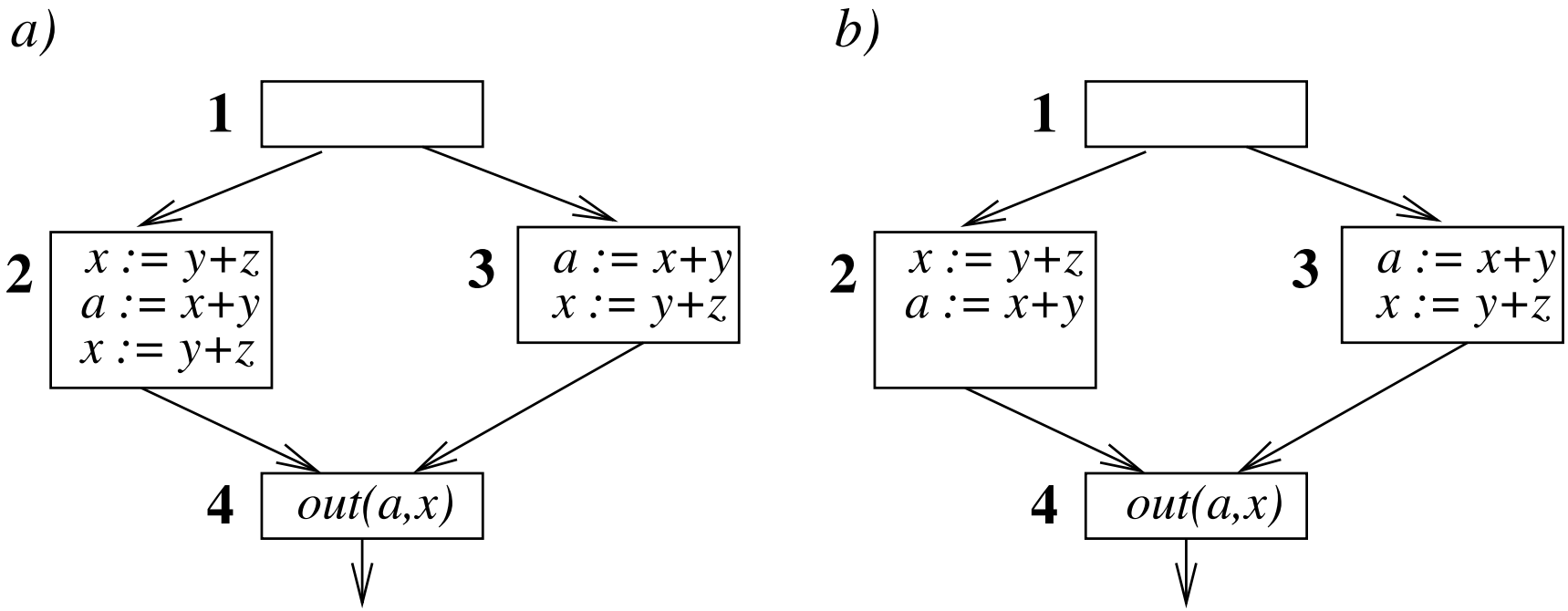
Veranschaulichung von Effekten 2. Ordnung

Ausgangsprogramm:



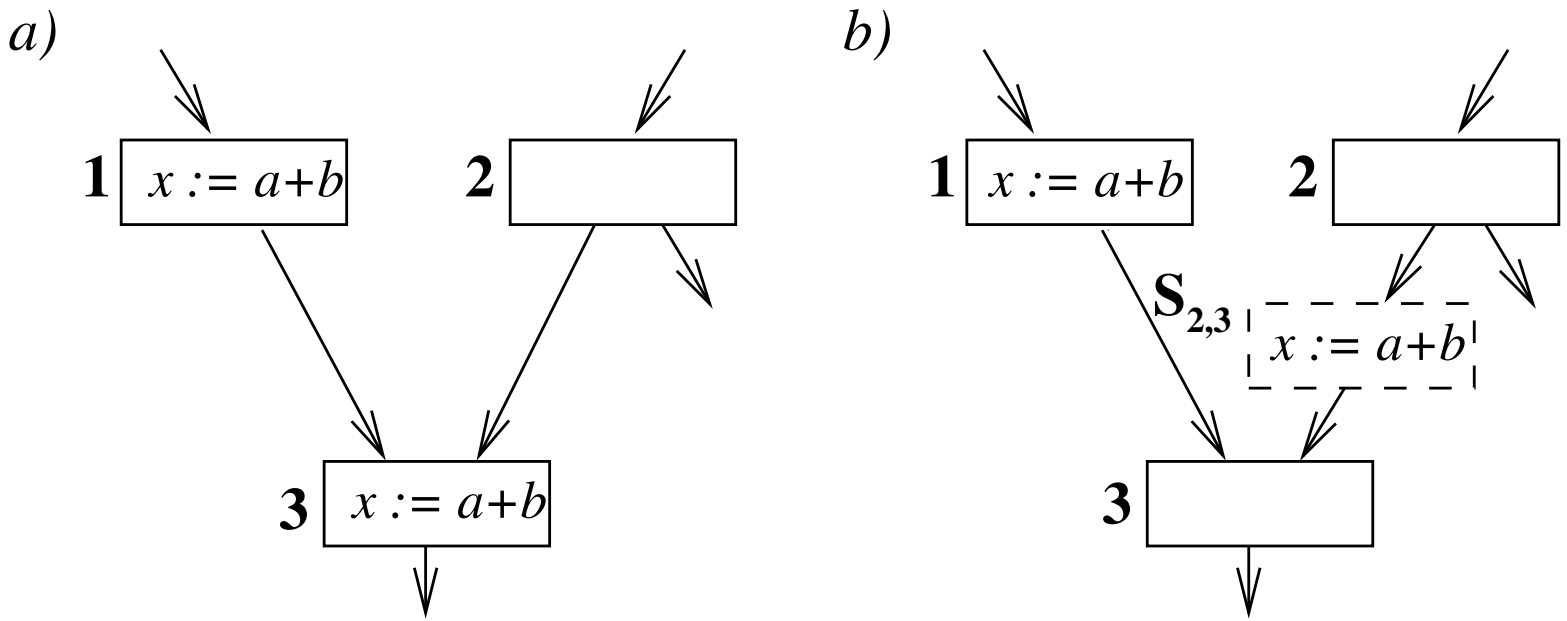
Veranschaulichung von Effekten 2. Ordnung

Transformiertes/optimiertes Programm:



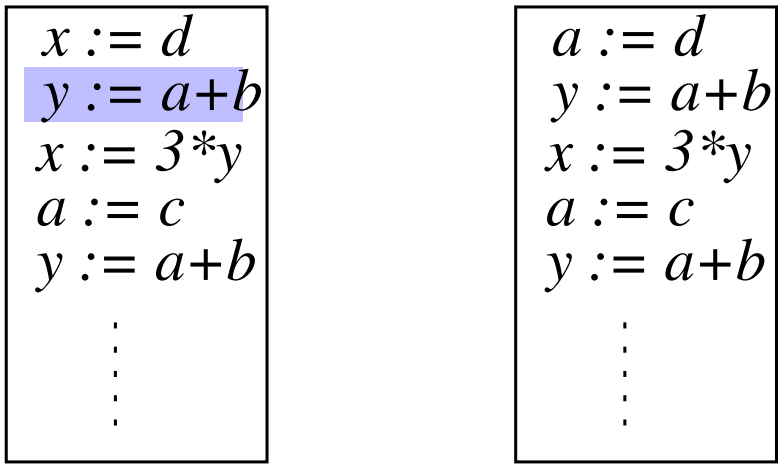
Wie für PDCE/PFCE

Spalten kritischer Kanten:



Analog zu PDCE/PFCE

Hebungskandidat in Basisblöcken:



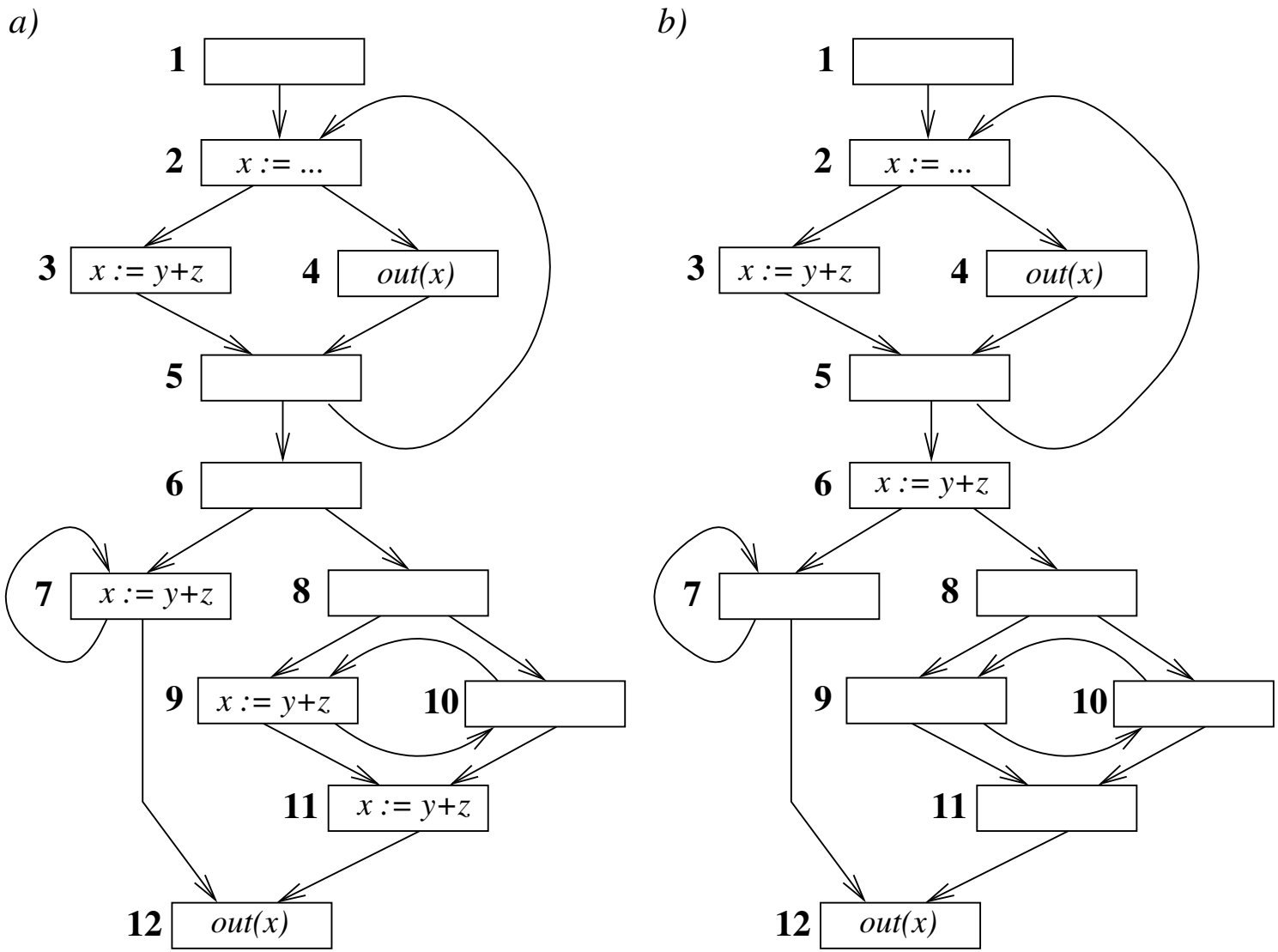
Hebungskandidat

Beachte:

- Nur das markierte Vorkommen von $y := a + b$ ist ein Hebungskandidat; die drei anderen Vorkommen von $y := a + b$ sind lokal blockiert.

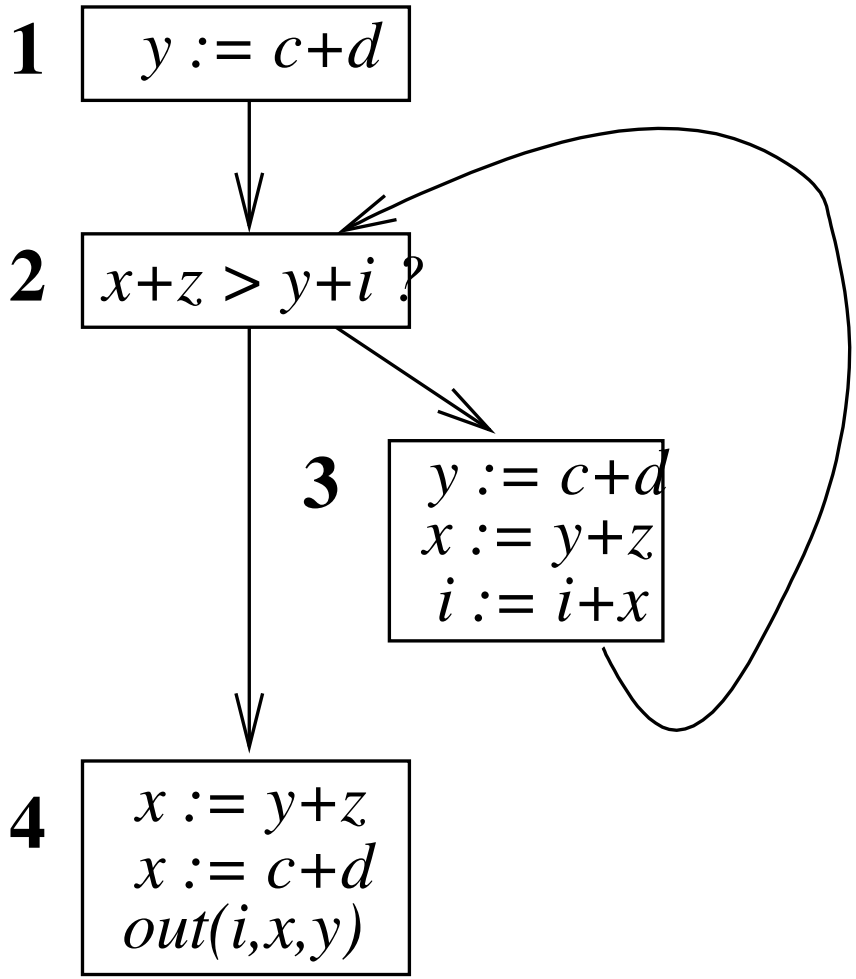
EAM für schleifenbehaftete Programme

Kein Schieben von Anweisungen in Schleifen:



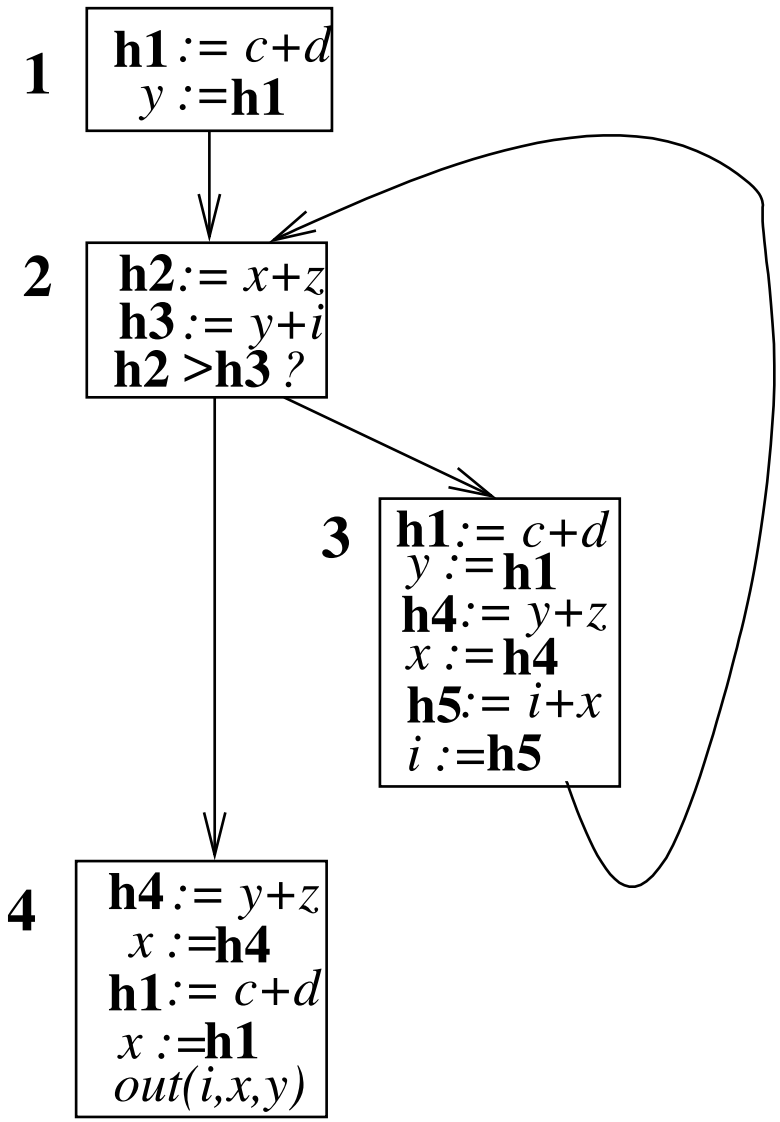
Die EAM-Transformation im Detail (1)

Ausgangsprogramm:



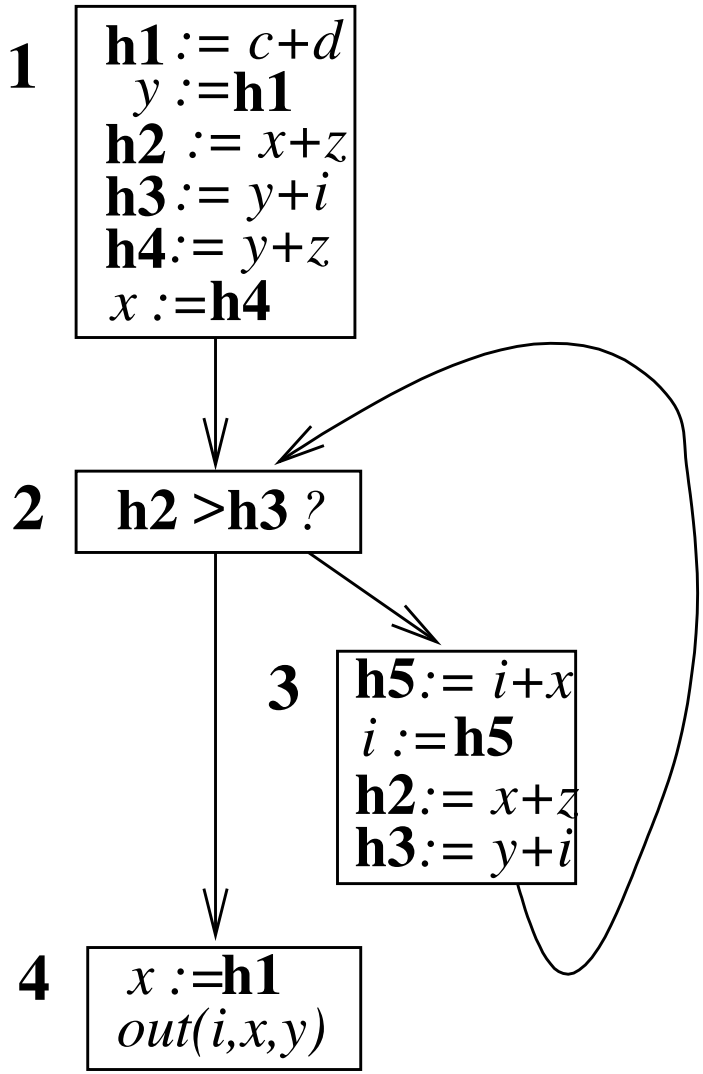
Die EAM-Transformation im Detail (2)

Effekt des Präprozesses:



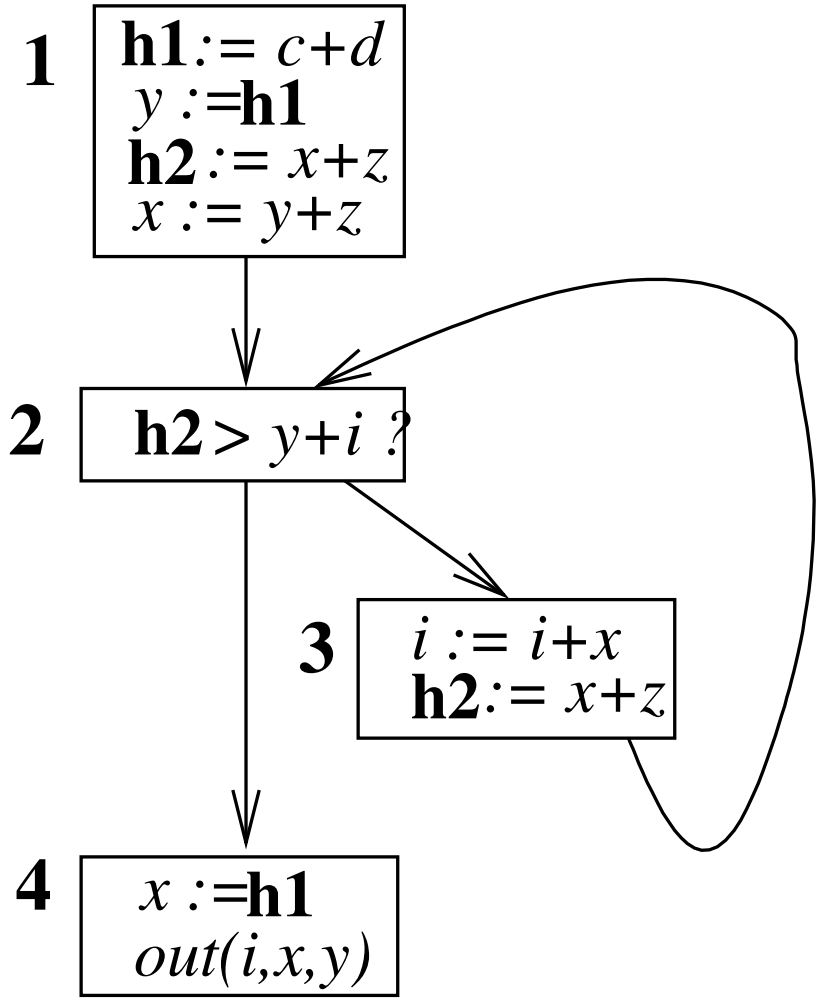
Die EAM-Transformation im Detail (3)

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 (12.2.1, Korrektheit)

$$G_{eam} \in \mathcal{G}_{EAM}$$

EAM-Optimalität: Ausdrucksoptimalität

Theorem (12.2.2, 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 (12.2.3, 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.

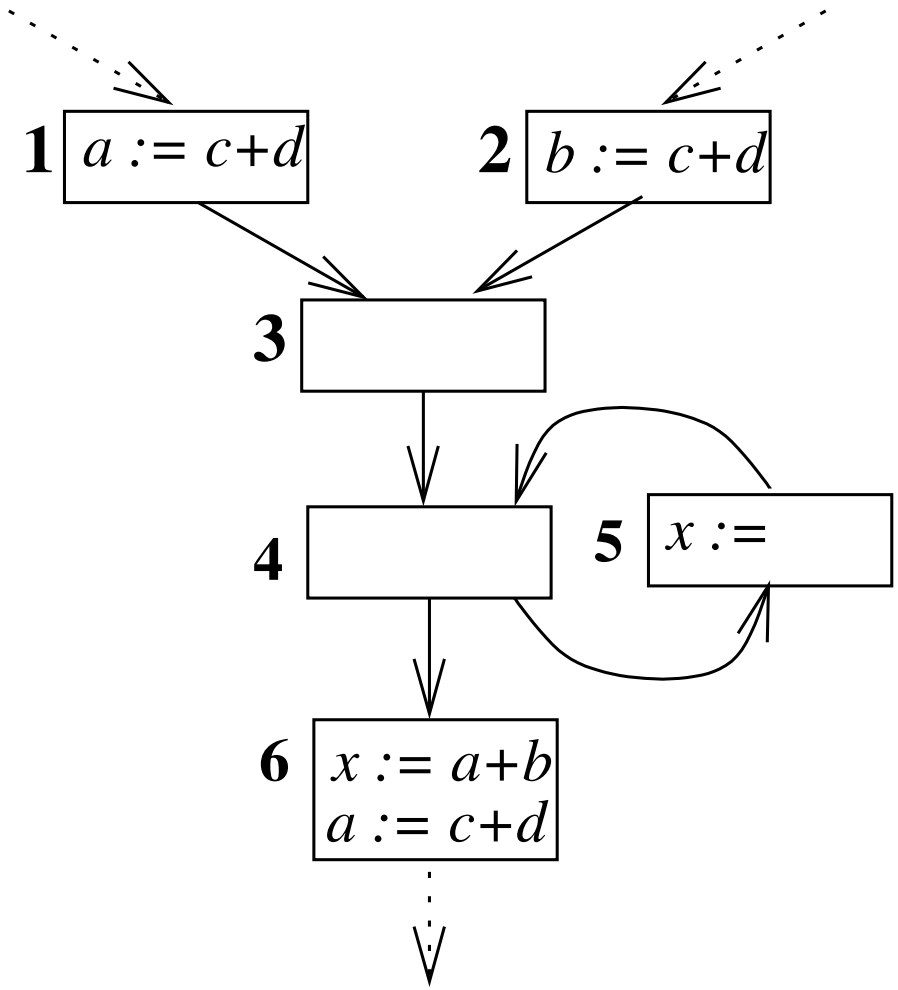
EAM-Optimalität: Hilfsvariablenoptimalität

Theorem (12.2.4, 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.

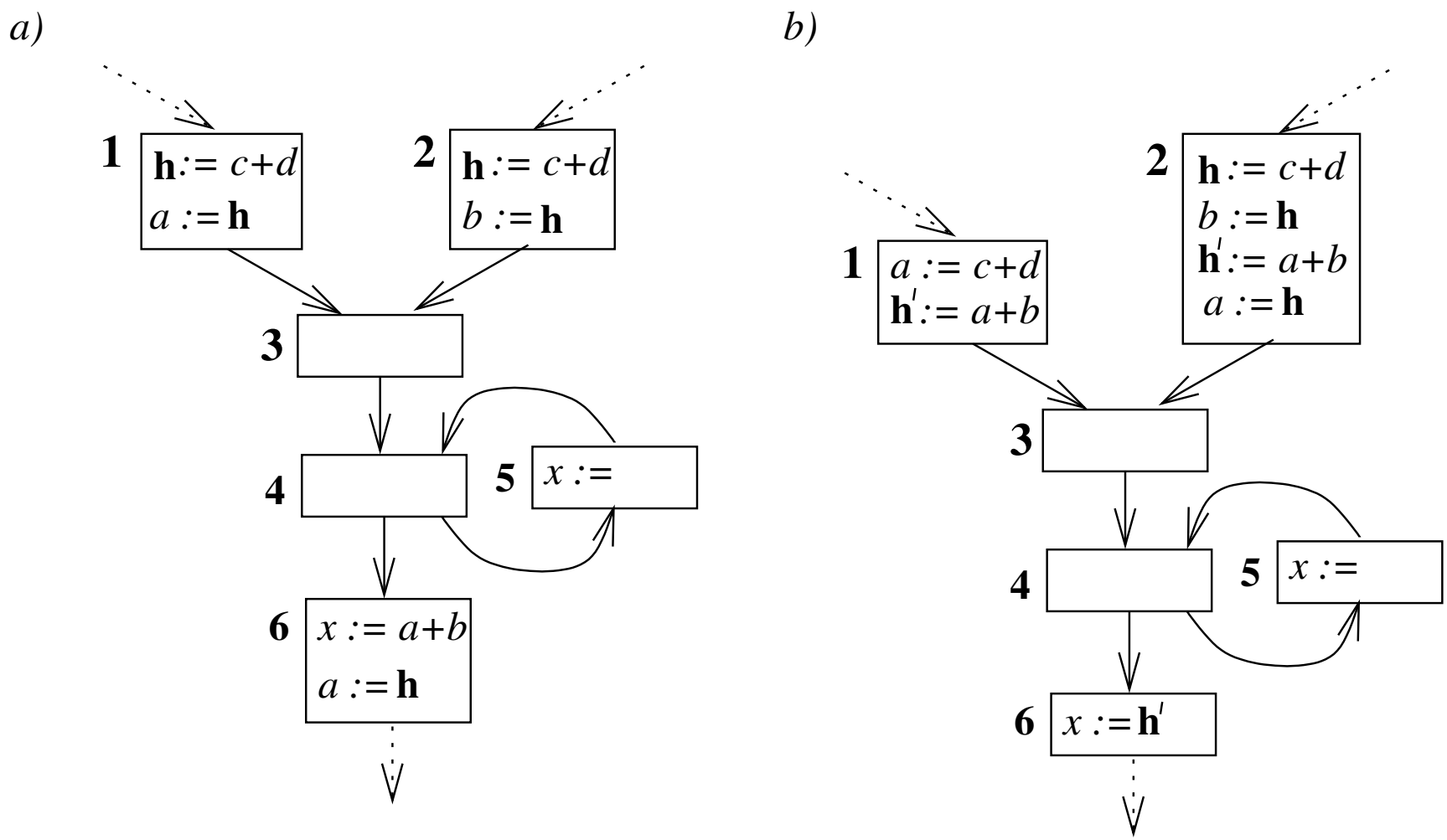
Warum nur relative A/HV-Optimalität?

Betrachte dazu folgendes Programm:



Zur relativen A/HV-Optimalität

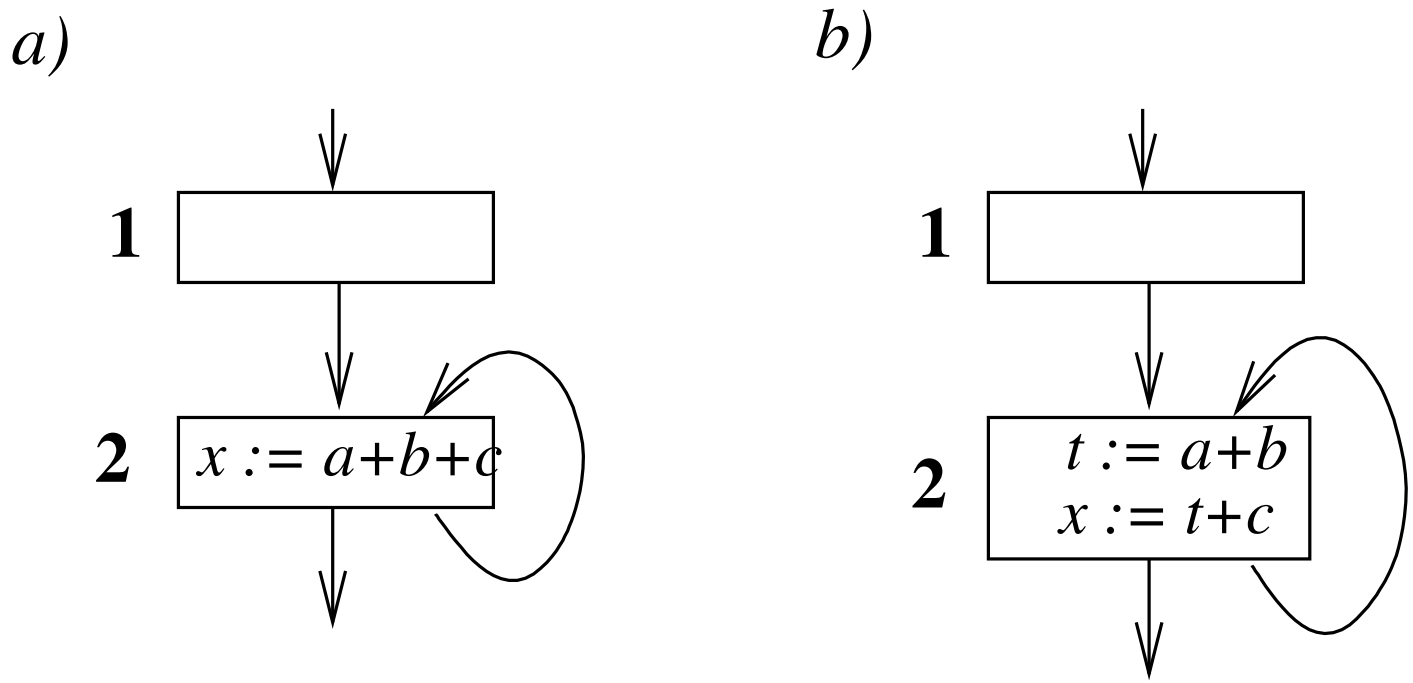
...und folgende zwei unvergleichbare Transformationsresultate:



⇒ Relative A/HV-Optimalität ist das Beste, was möglich ist!

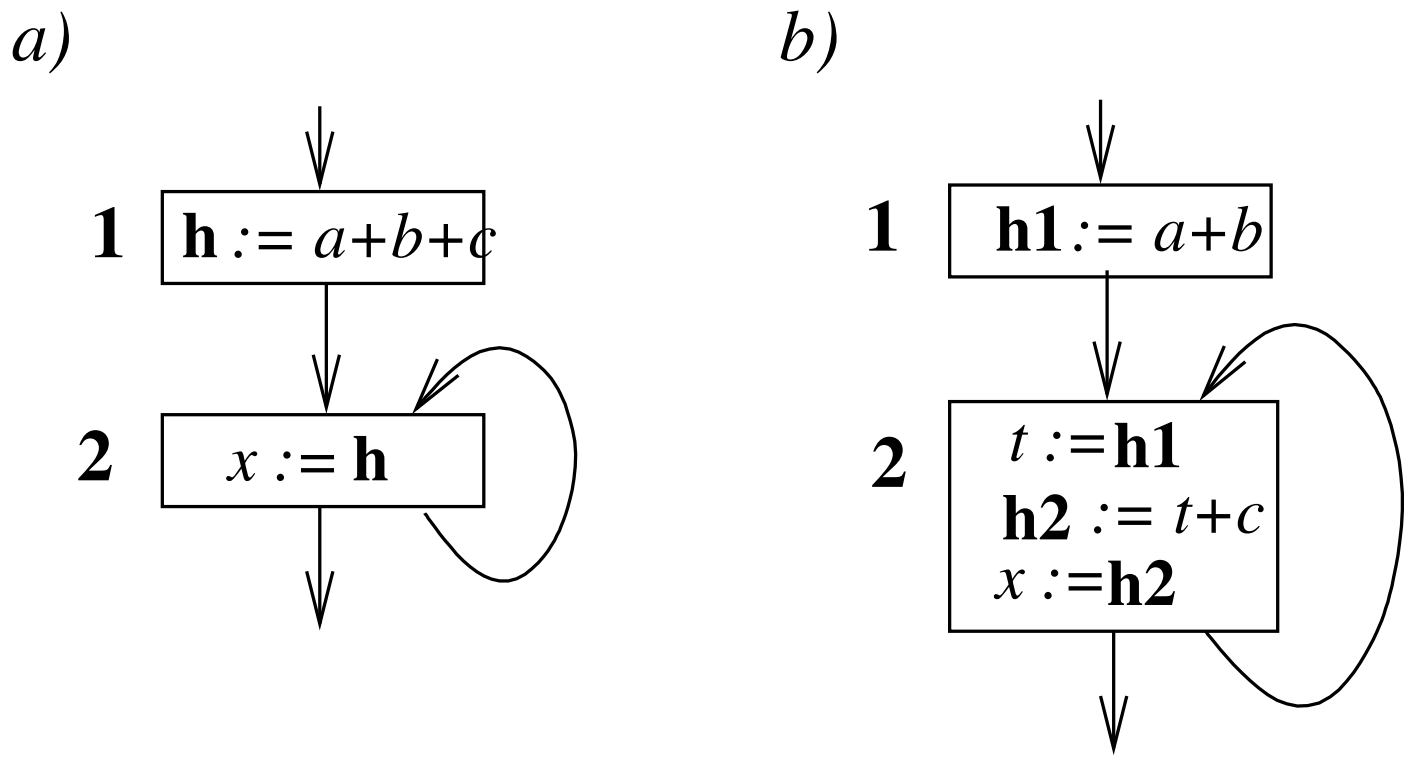
Weitere PREE/PRAE-Phänomene (1)

PREE/PRAE-Phänomene im Zusammenhang mit 3-Adresscode und Nicht-3-Adresscode:



Weitere PREE/PRAE-Phänomene (2)

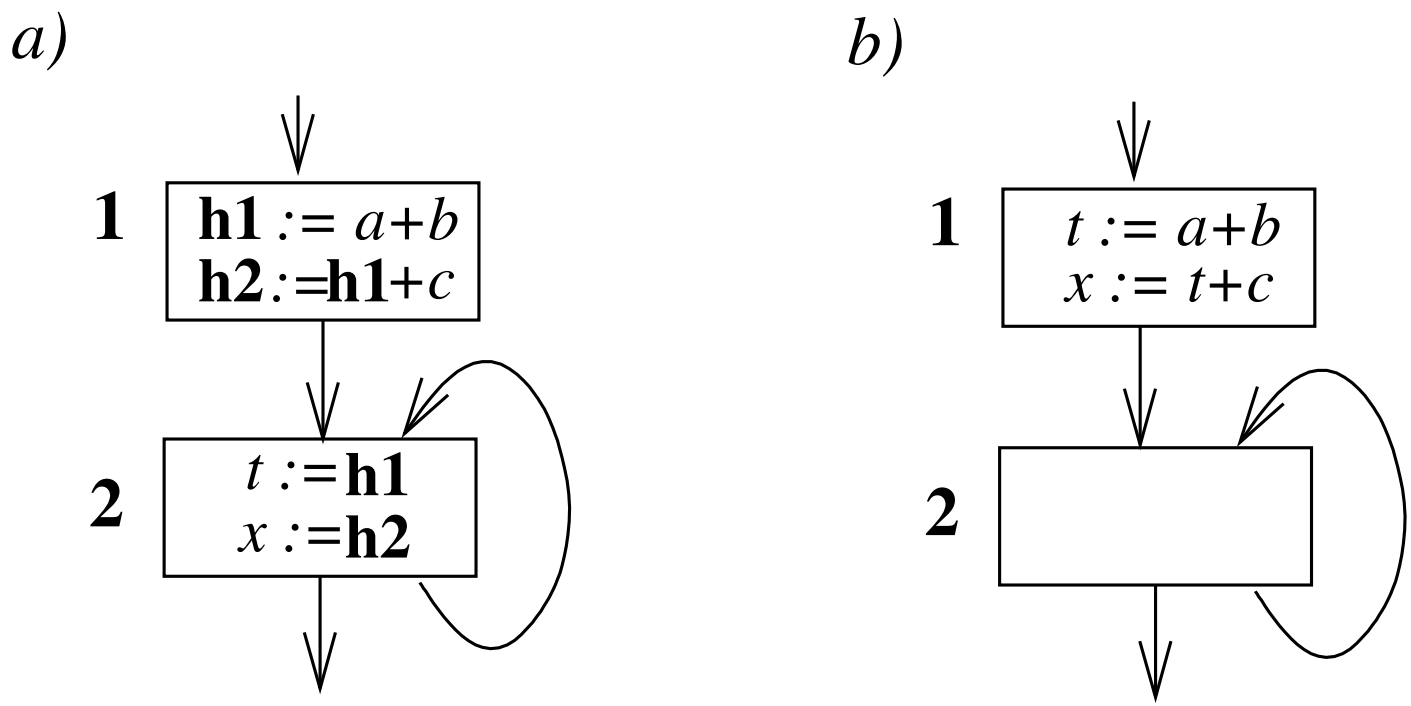
Der Effekt von PREE auf die Programme aus Abb. (a) und (b):



Weitere PREE/PRAE-Phänomene (3)

Die Effekte von

- PREE gefolgt von Konstantenpropagierung (constant propagation (CP)): Abb. (a)
- EAM: Abb. (b)



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.

Ergänzende, weiterführende und vertiefende 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.

Ergänzende, weiterführende und vertiefende 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

Motivation

Erinnerung:

Konzeptuell können wir PRAE und PDCE wie folgt verstehen:

- PRAE = $(\mathrm{AH} + \mathrm{RAE})^*$
- PDCE = $(\mathrm{AS} + \mathrm{DCE})^*$

Motivation

Erinnerung:

Konzeptuell können wir PRAE und PDCE wie folgt verstehen:

- PRAE $= (\mathrm{AH} + \mathrm{RAE})^*$
- PDCE $= (\mathrm{AS} + \mathrm{DCE})^*$

Das legt nahe auch die "Summe" aus PRAE und PDCE zu betrachten:

- AP $= (\mathrm{AH} + \mathrm{RAE} + \mathrm{AS} + \mathrm{DCE})^*$

AP steht dabei für Assignment Placement.

Individuelle PRAE/PDCE-Opt.-Ergebnisse

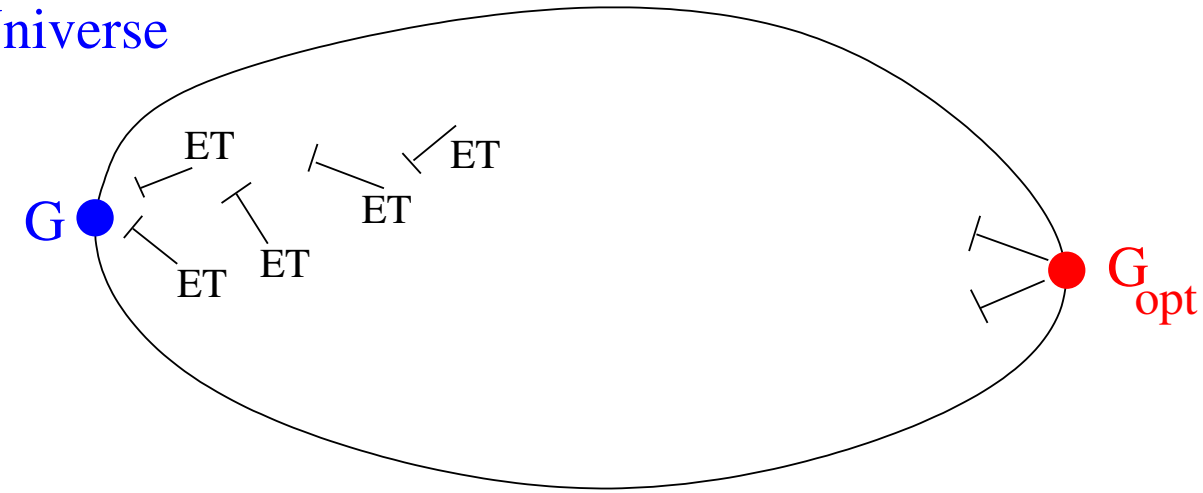
PRAE- bzw. PDCE-Ableitungsrelation $\vdash$:

- 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, 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!

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.

Kombination von PRAE und PDCE: AP

Betrachte nun:

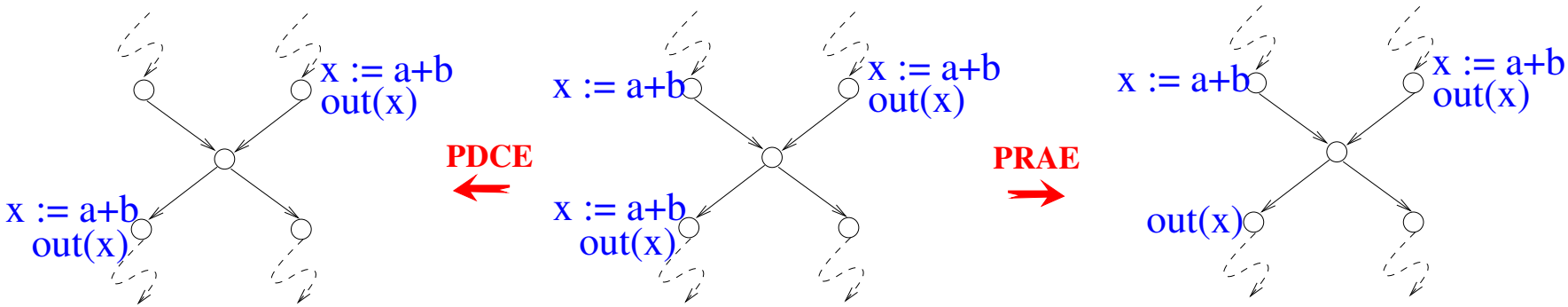
- Assignment Placement AP

$$\mathrm{AP} = (\mathrm{AH} + \mathrm{RAE} + \mathrm{AS} + \mathrm{DCE})^*$$

Erwartung:

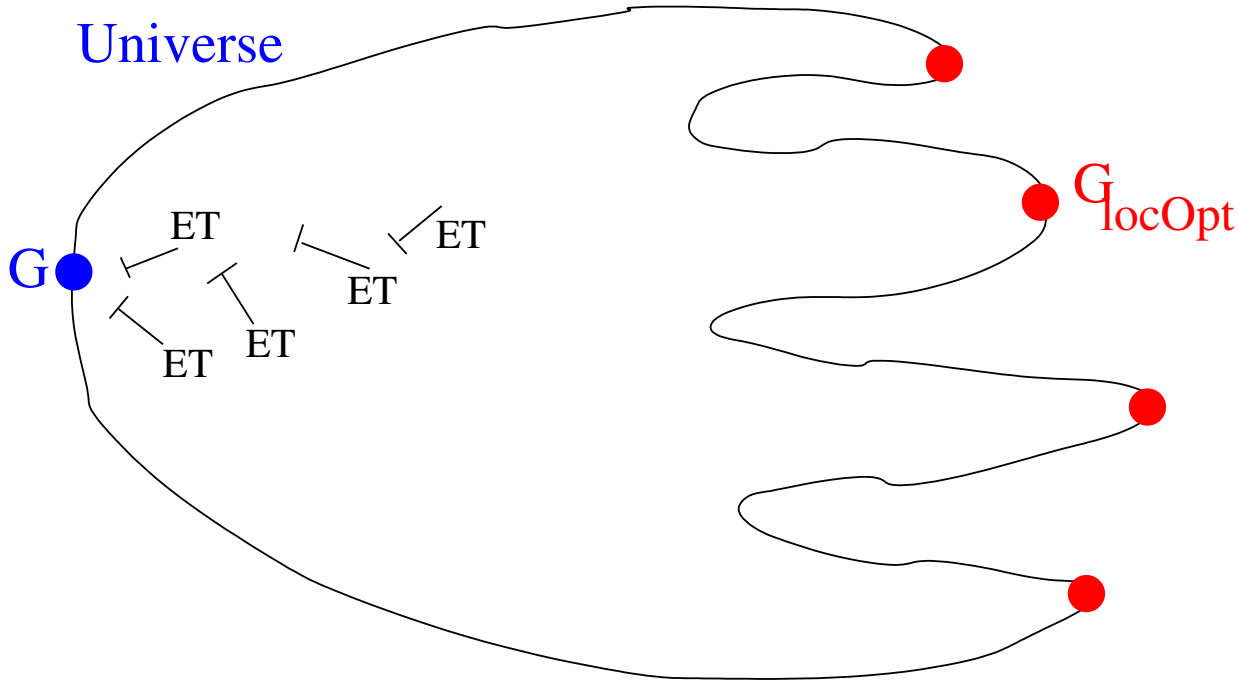
- AP sollte mächtiger sein als PRAE und PDCE individuell!

Das gilt in der Tat. Aber:



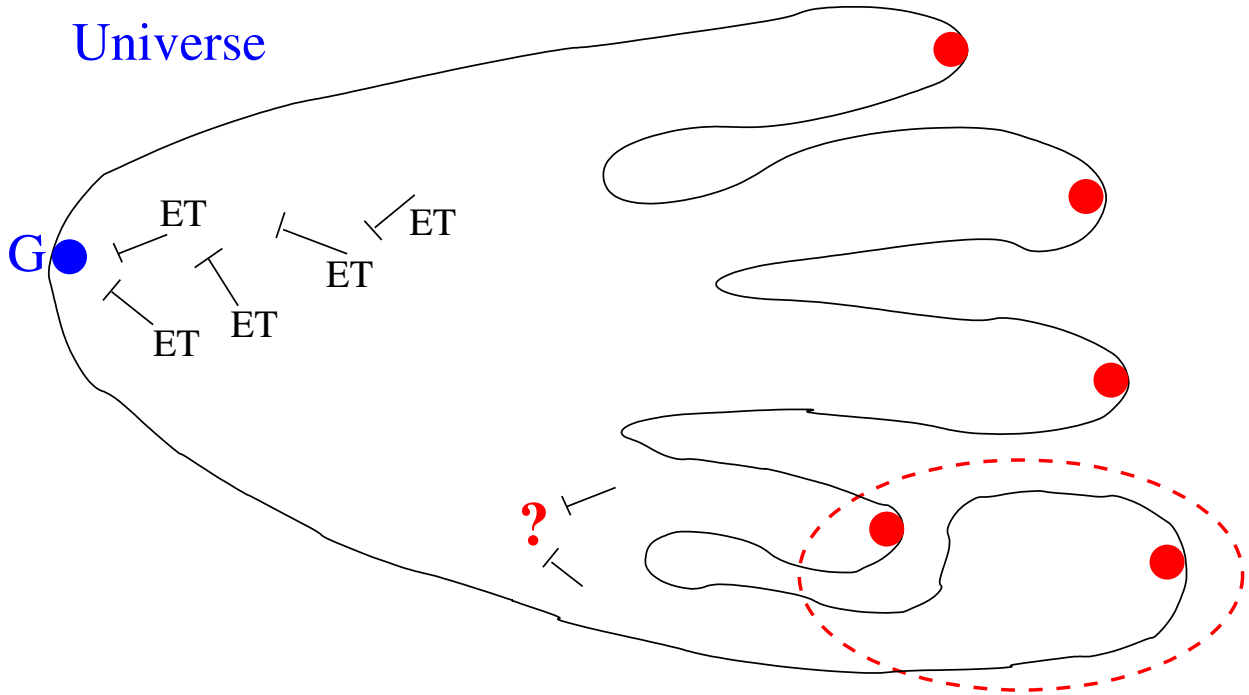
AP: Verlust globaler Optimalität

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.

Ergänzende, weiterführende und vertiefende 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

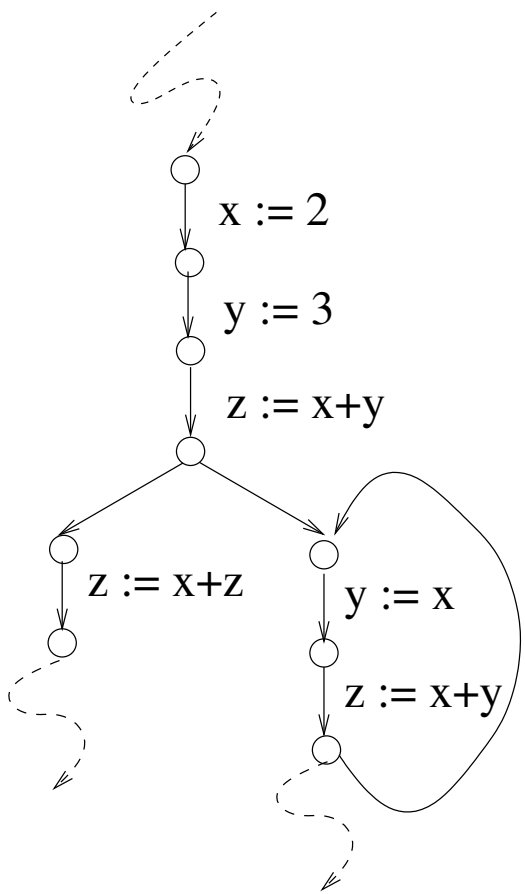
Kapitel 14.1

Motivation

Konstantenfaltung und -ausbreitung

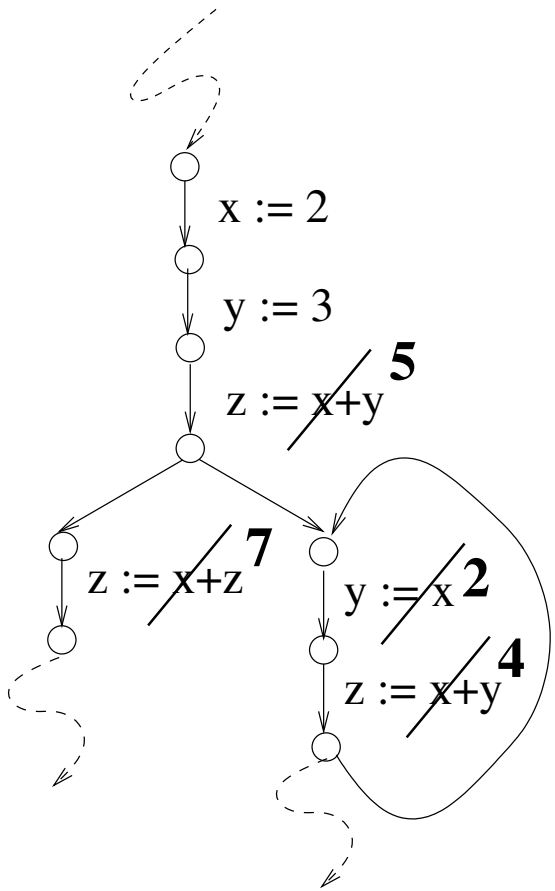
Veranschaulichendes Beispiel

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)
    - ...

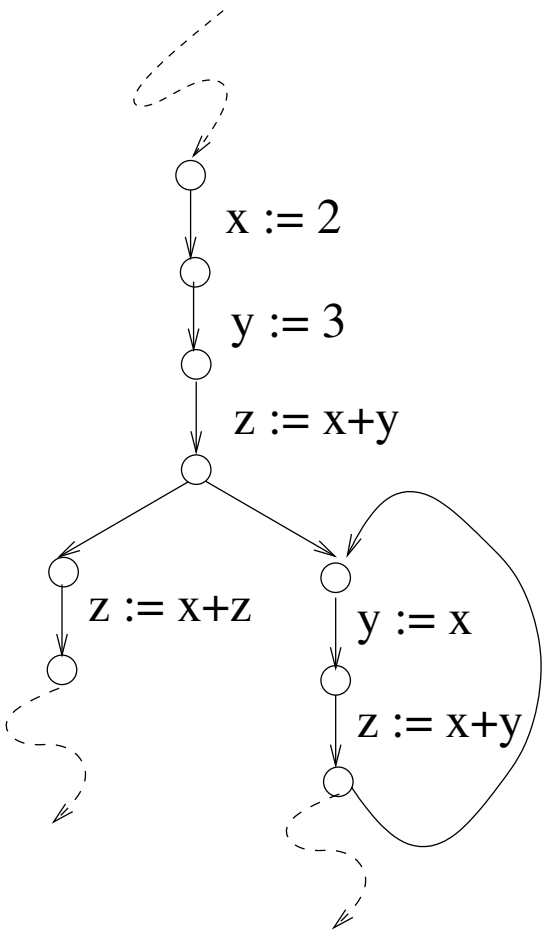
  wobei Anwendungsreichweite zulasten von Ausdruckskraft gewonnen wird: Kopierkonstanzen (engl. copy constants), lineare Konstanten (engl. linear constants) anstelle von einfachen Konstanten (engl. simple constants).

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)
  - ...

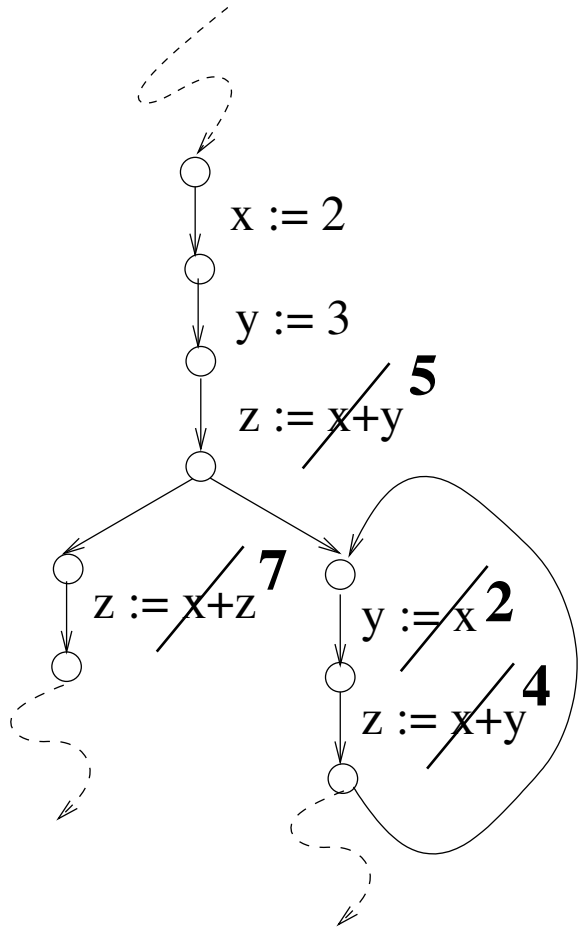
Warum nach größerer Ausdruckskraft streben?

a)



Original program

b)



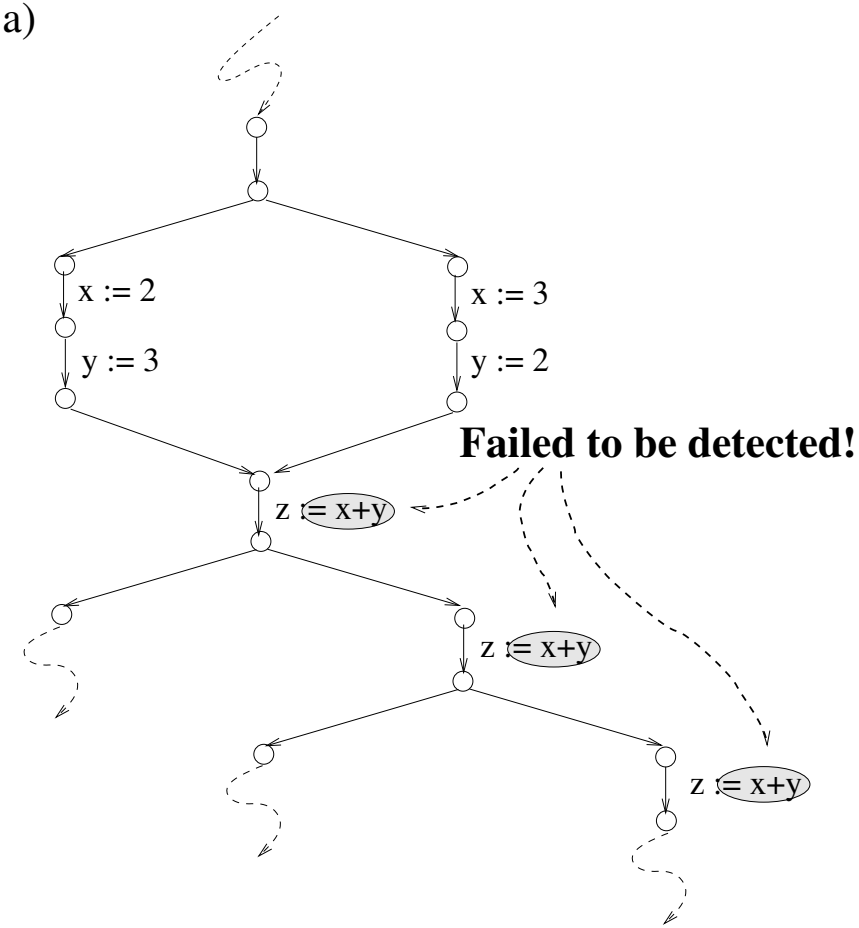
After simple constant propagation

Das SC-Transformationsergebnis ist doch überzeugend, nicht?

Tatsächlich ist es nicht überzeugend

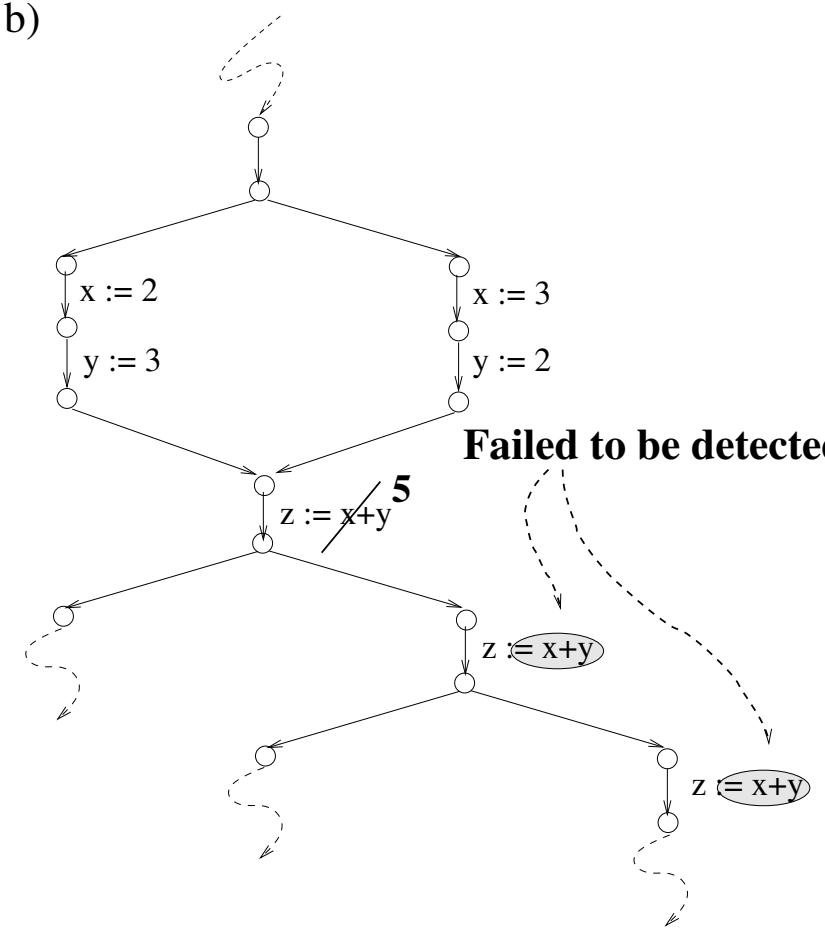
Der Begriff einfacher Konstanten ist schwach:

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:

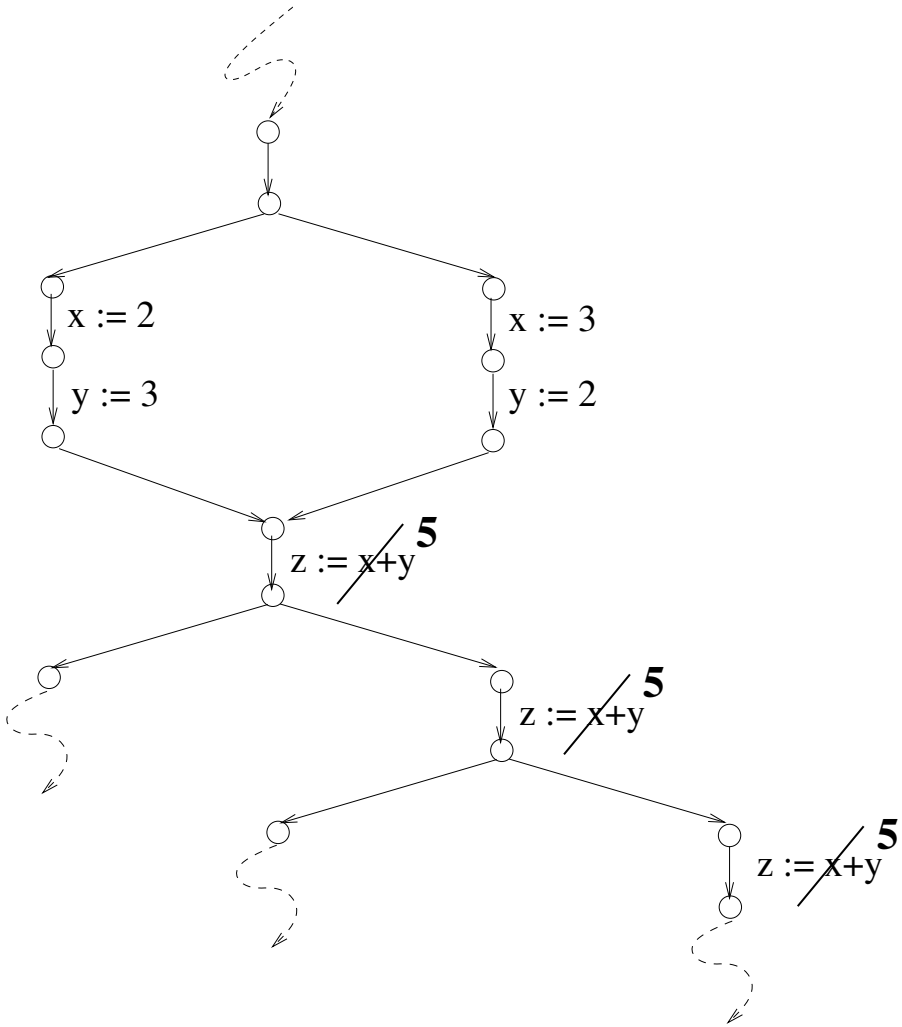
- Konstantenfaltung ist unentscheidbar: Reif, Lewis (POPL 1977)

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:

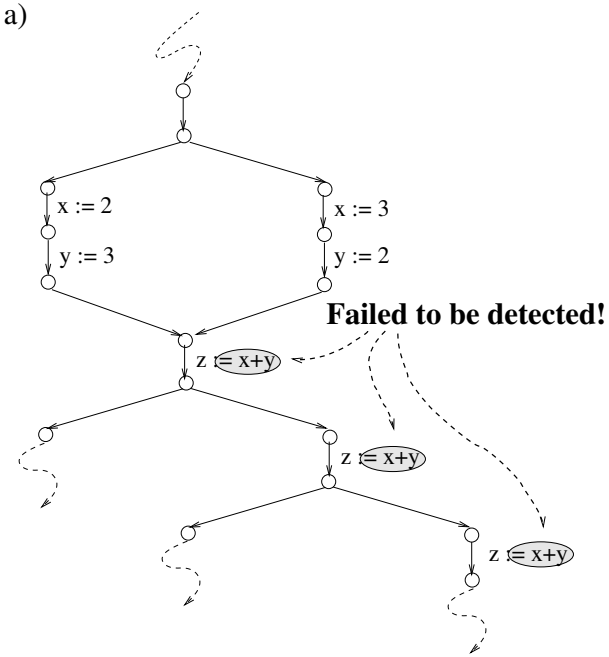
Theorem (14.1.1)

Konstantenfaltung und -ausbreitung ist für schleifenfreie Programme co-NP-vollständig.

Knoop, Rüthing (CC'00)
Müller-Olm, Rüthing (ESOP'01)

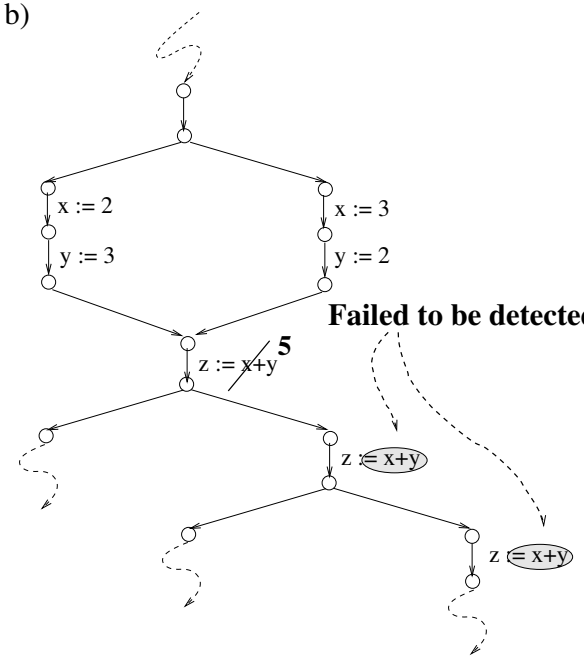
Zurück zum laufenden Beispiel

a)



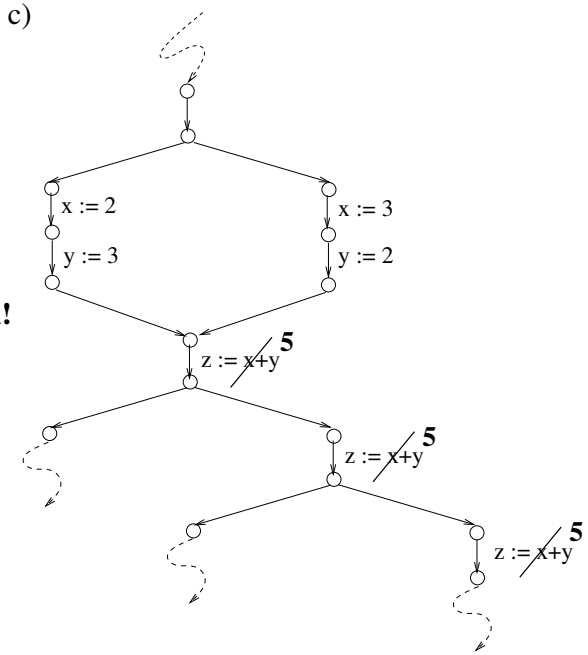
After simple constant propagation
(Note: No effect at all!)

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)).

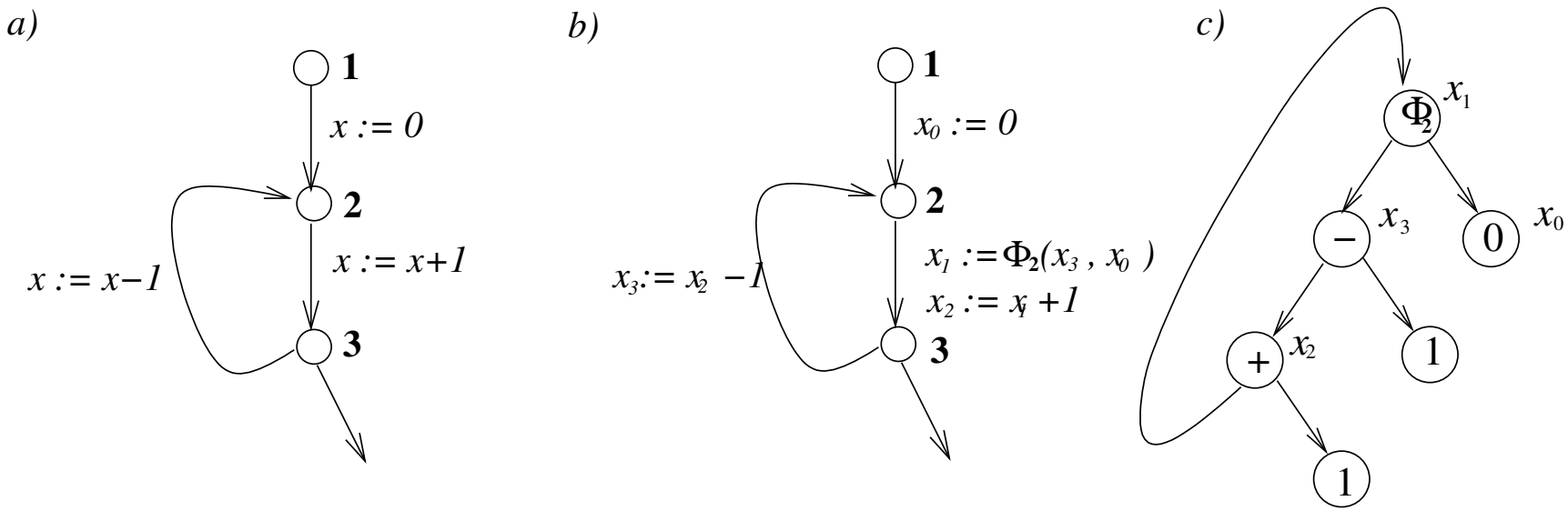
Insgesamt erhalten wir den

- VG-Konstantenfaltungsalgorithmus, einen Konstantenfaltungsalgorithmus mit SSA-Form, nicht auf SSA-Form.

Kapitel 14.2

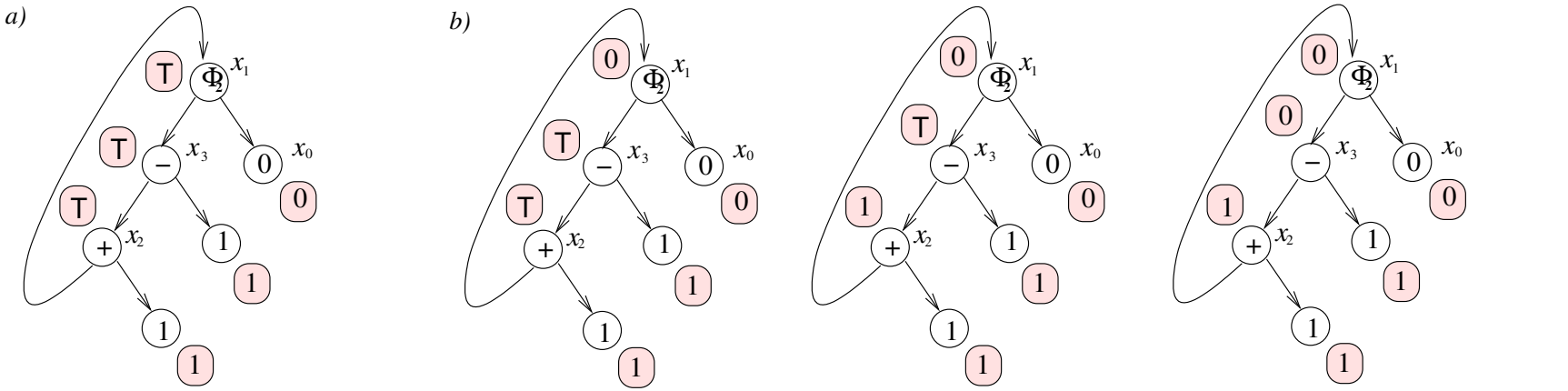
Der VG-Konstantenfaltungsalgorithmus

The Wertegraph von Alpern, Wegman, Zadeck



Ausgangsprogramm SSA-Form Wertegraph

Konstantenfaltung auf dem Wertegraphen



After the initialization step | After the 1st iteration step | After the 2nd iteration step | After the 3rd iteration step: Stable!

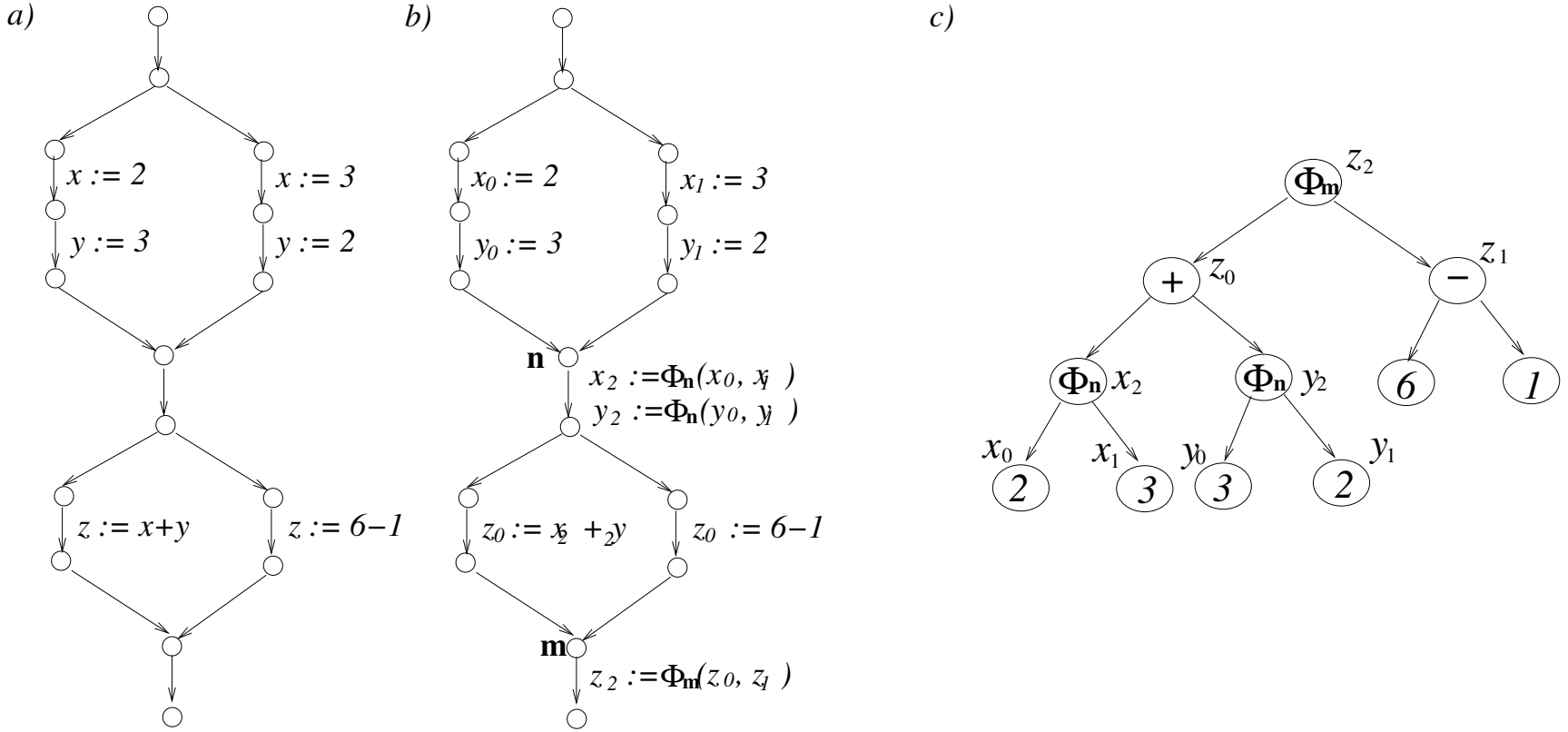
Analyseresultat: x_2 und x_3 sind von konstantem Wert!

Konstantenfaltung auf dem Wertegraphen

...mit zwei Ausprägungen:

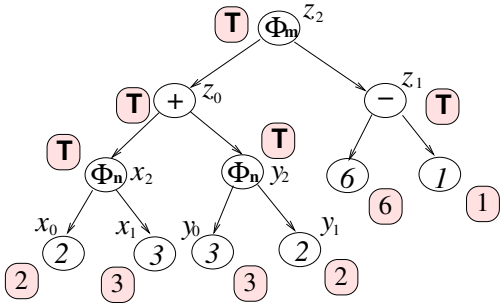
- Der VG-Grundalgorithmus
  ...berechnet einfache Konstanten.
- Der volle VG-Algorithmus
  ...geht über einfache Konstanten und die 1-Vorschauheuristik hinaus.

Beispiel zur Veranschaulichung des vollen VG-Algorithmus

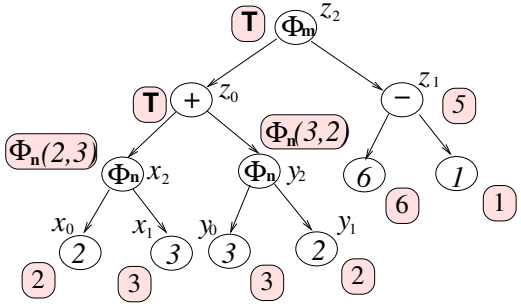


Ausgangsprogramm SSA-Form Wertegraph

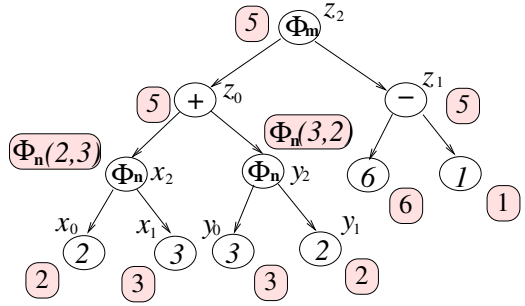
Der volle VG-Algorithmus auf dem Wertegraphen



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

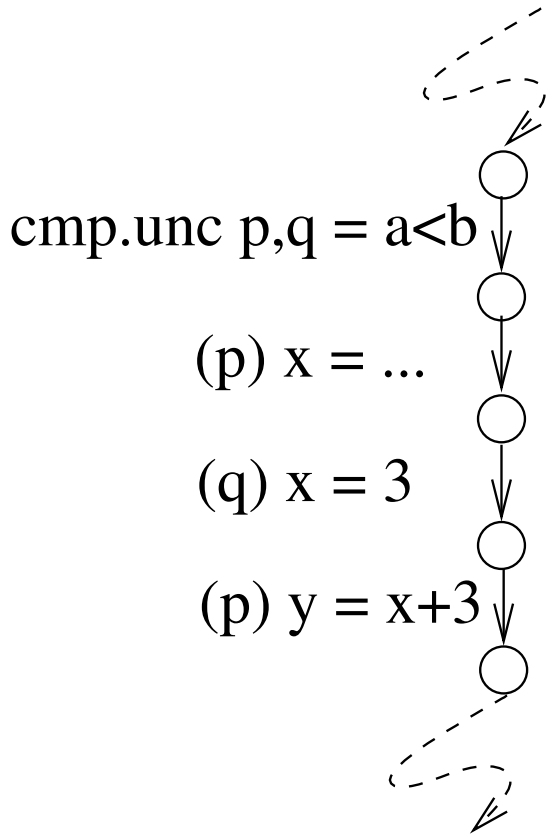
...zum VG-Konstantenfaltungsalgorithmus:

- 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

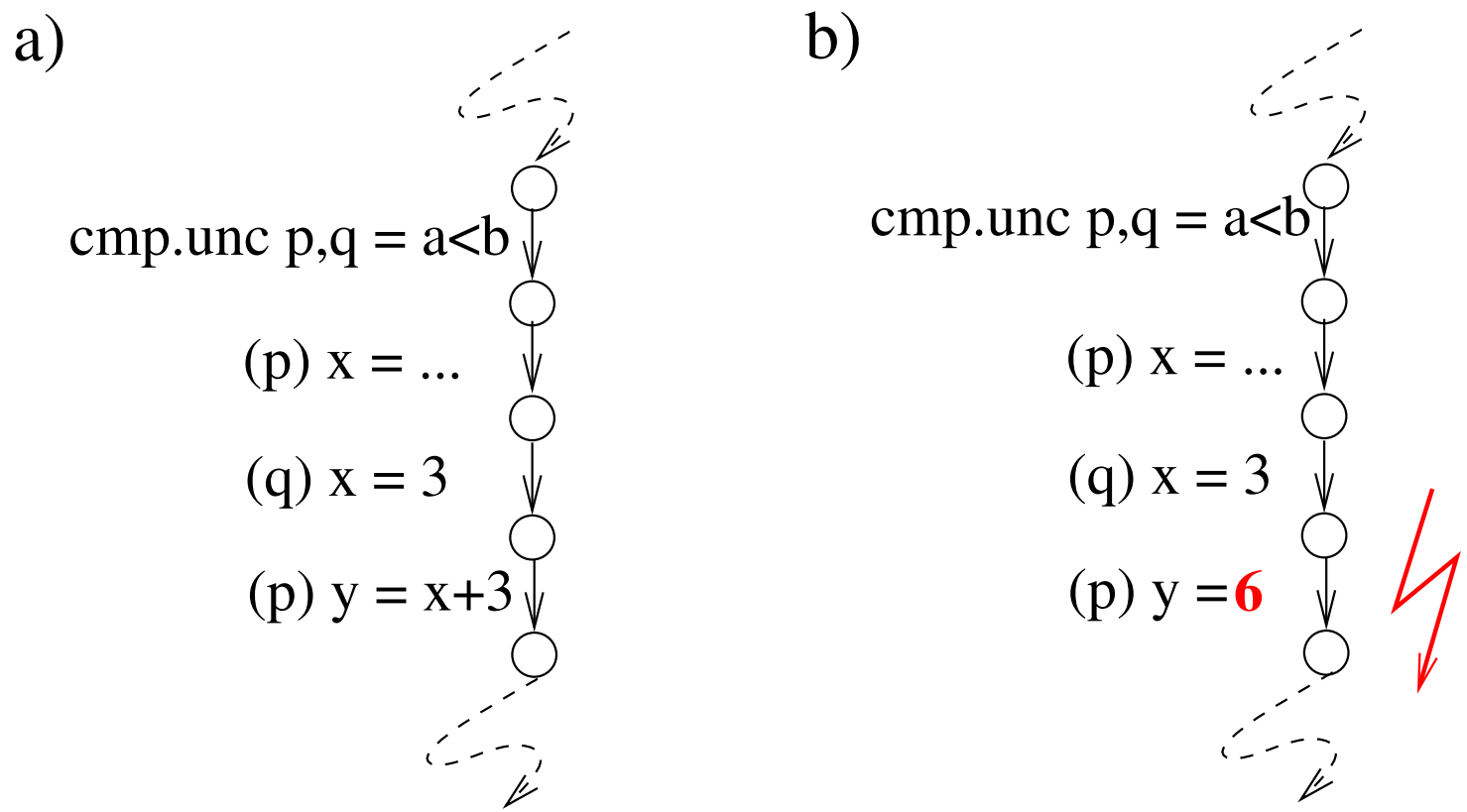
Prädikatierter Code



...als Resultat sog. if-Konversion.

Naive Konstantenfaltung

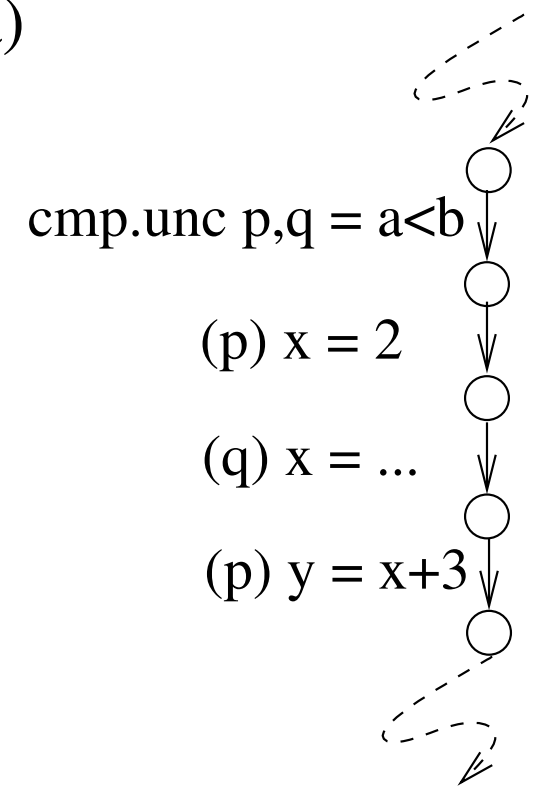
... auf prädikatiertem Code schlägt fehl:



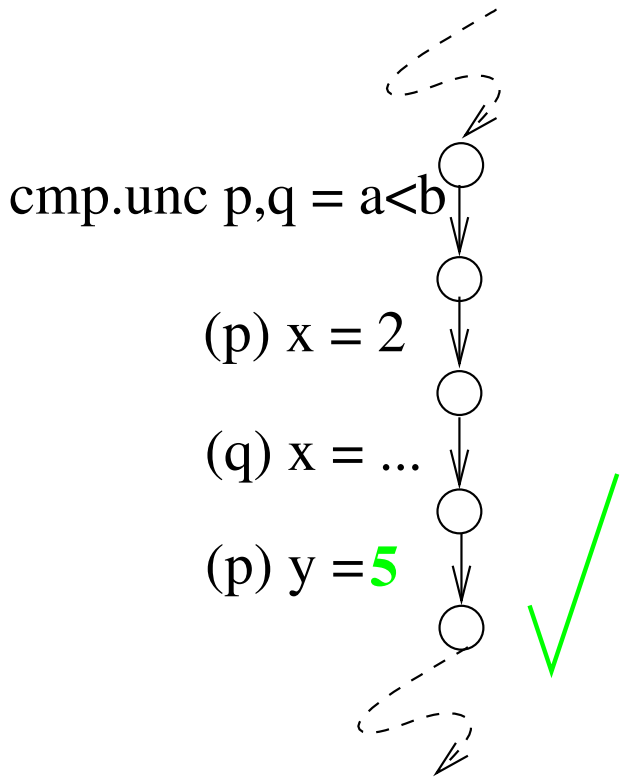
Naive korrekte Konstantenfaltung

...scheint andererseits zu konservativ und zu viele Transformationsmöglichkeiten auszulassen:

a)



b)



Ziel

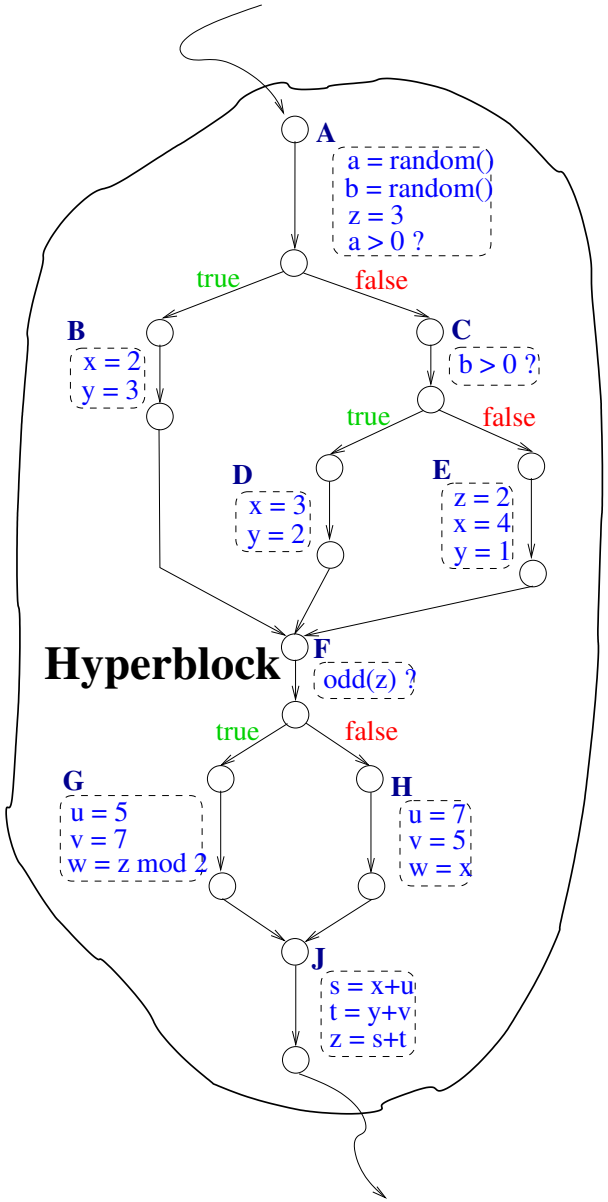
Ein intelligenterer Umgang mit prädikatiertem Code.

Hyperblöcke sind

- wichtige Komponenten prädikatierten Codes.

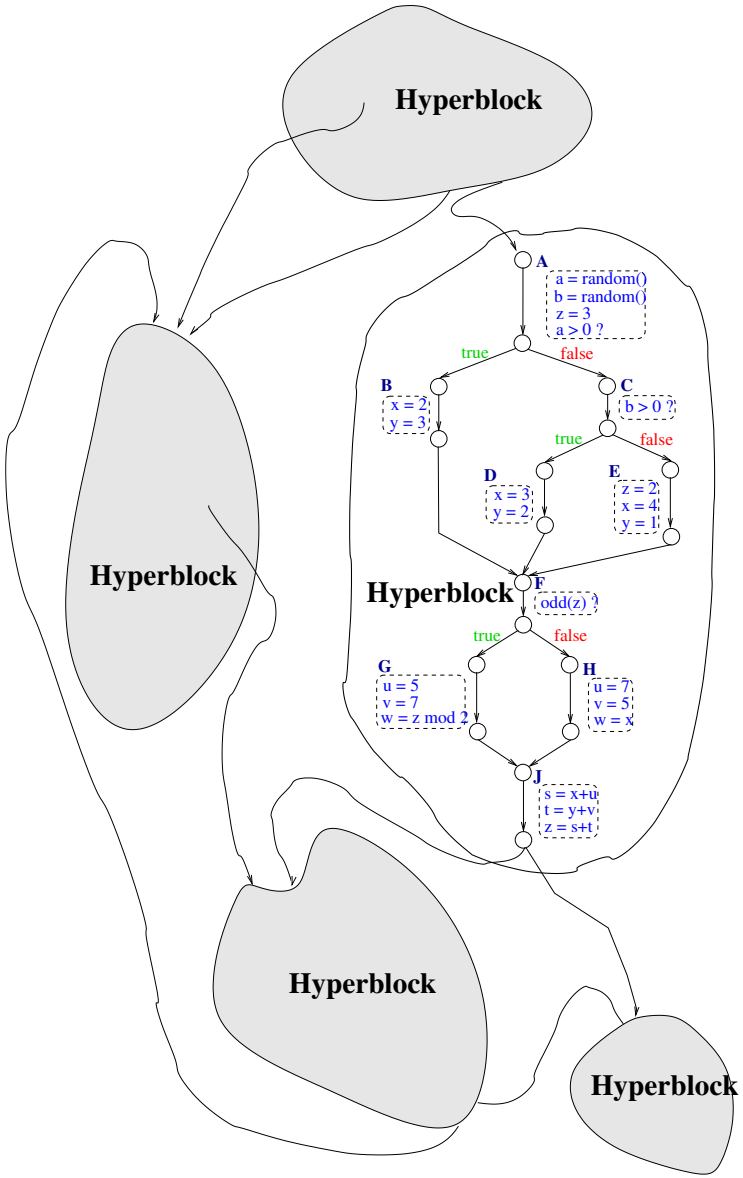
Hyperblöcke

...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.

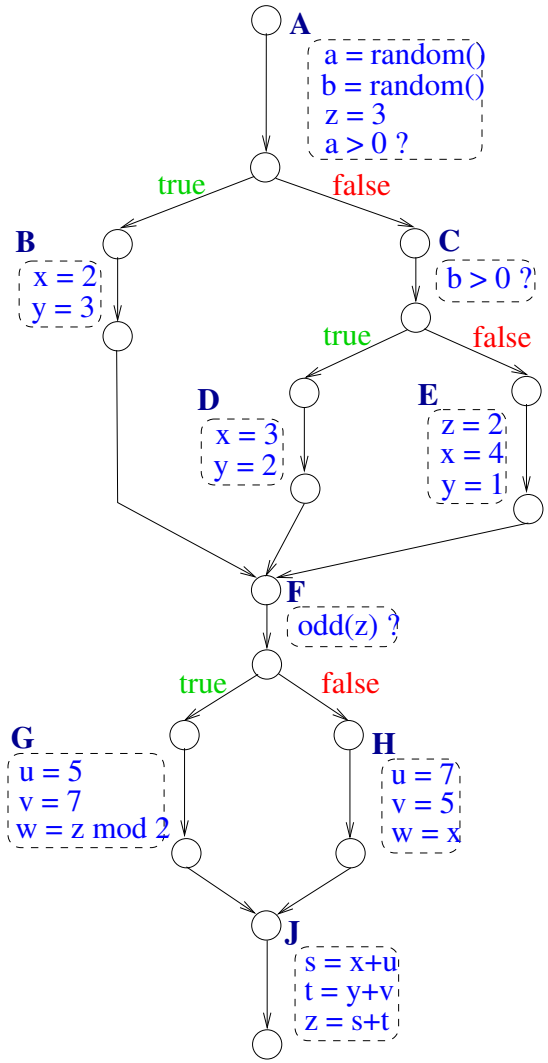
Alle diese Algorithmenvarianten bestehen jeweils aus einer

- lokalen und
- globalen

Analysestufe.

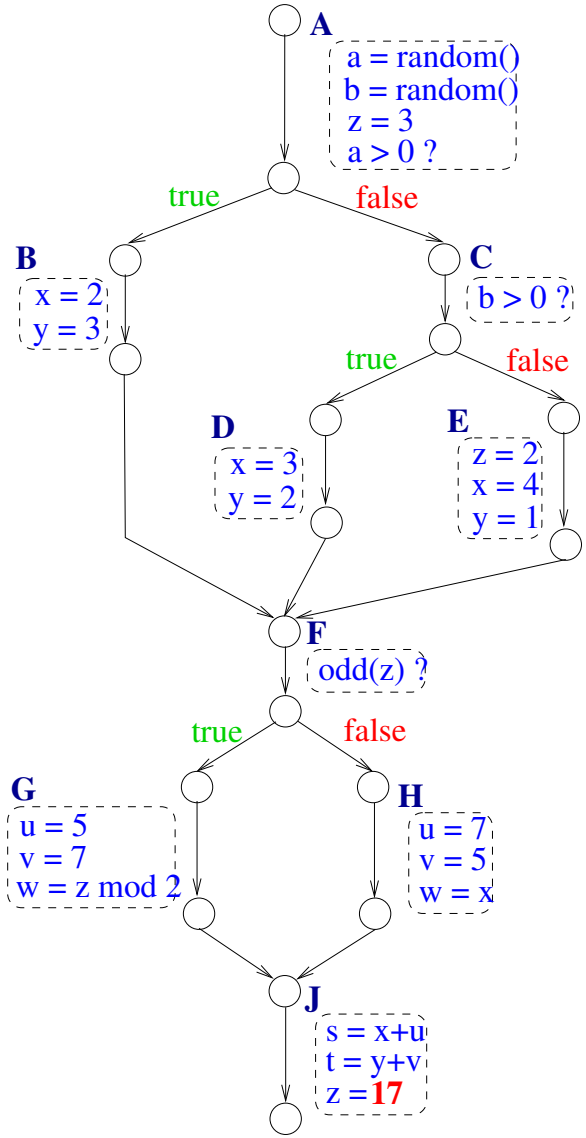
Diskussion der lokalen Analysestufe

Dazu betrachten wir folgenden Hyperblock:



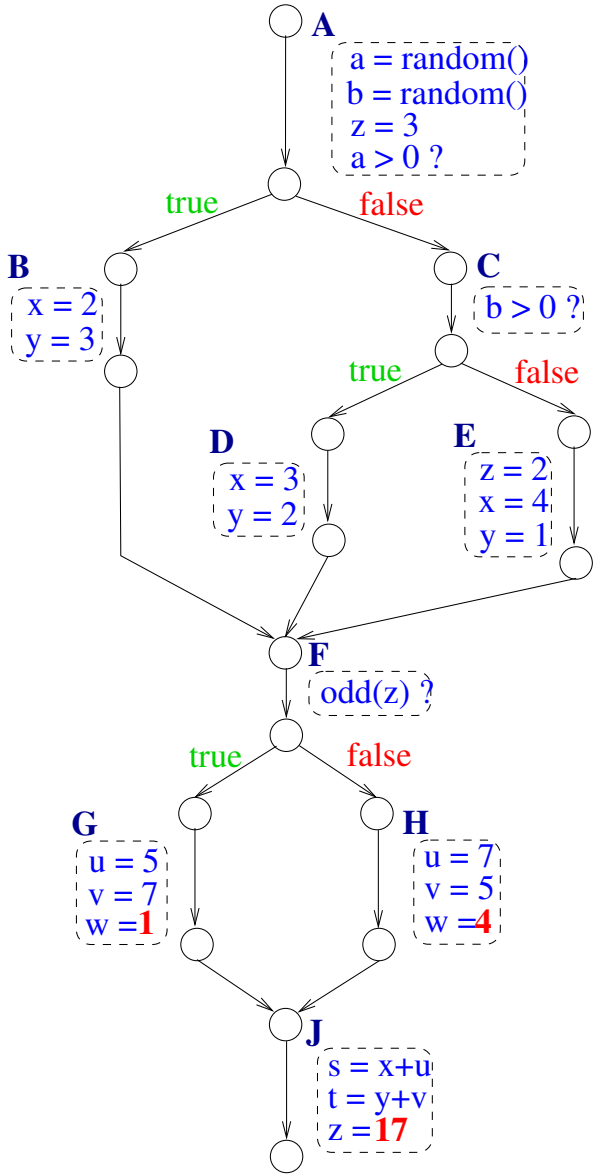
Original Hyperblock

Die PVG-Grundalgorithmustransformation



The Non–Deterministic Path–Precise Basic Optimization

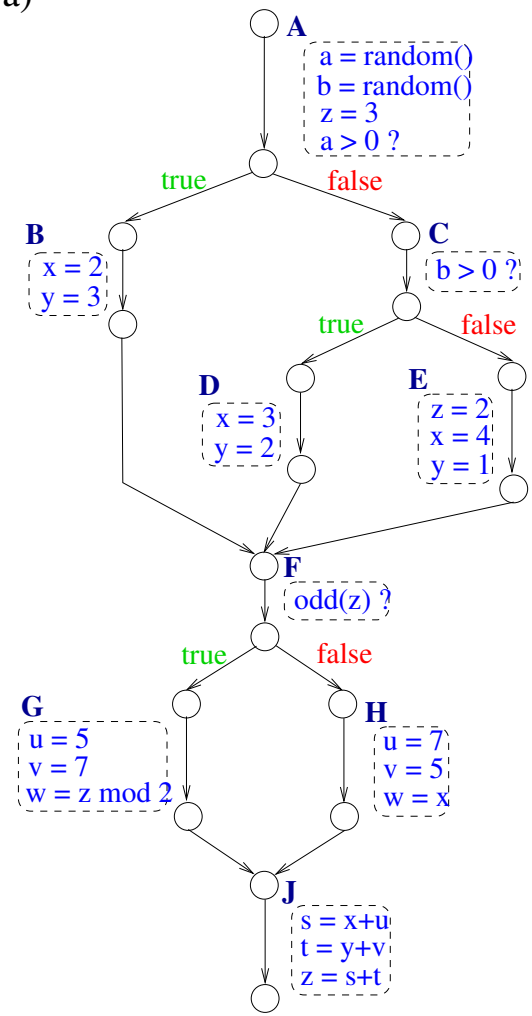
Die volle PVG-Algorithmustransformation



The Deterministic Path–Precise Full Optimization

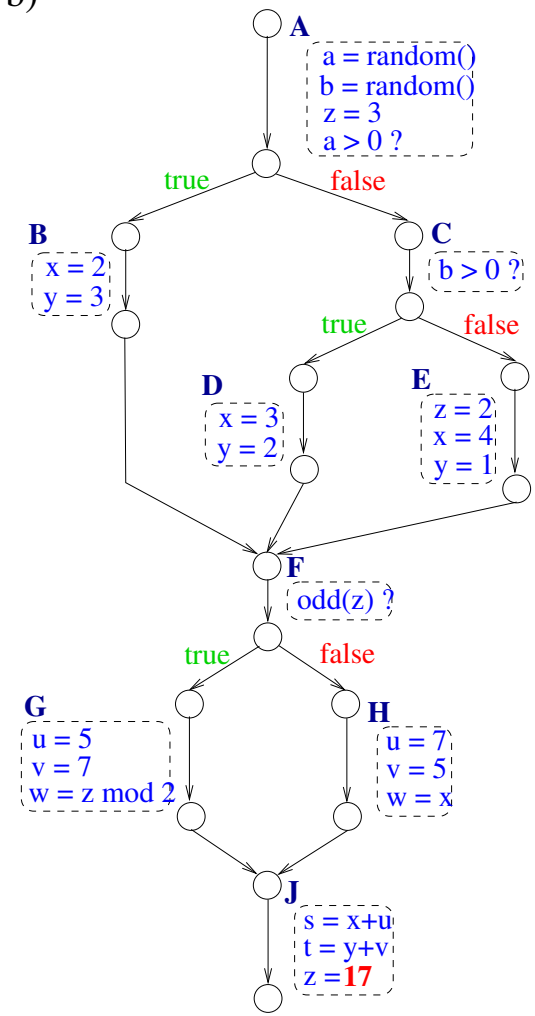
Ausgangsprogramm & beide Transformationen

a)



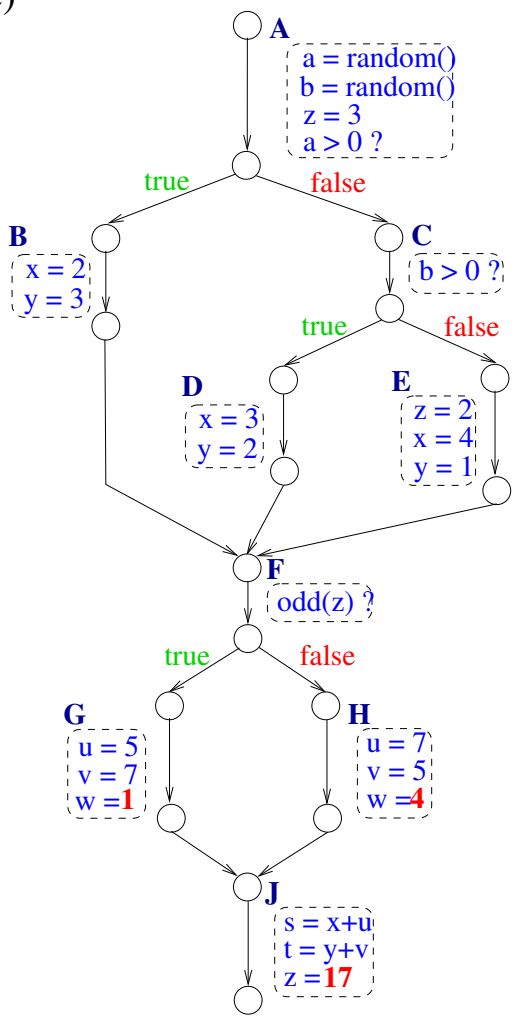
Original Hyperblock

b)



The Non–Deterministic Path–Precise Basic Optimization

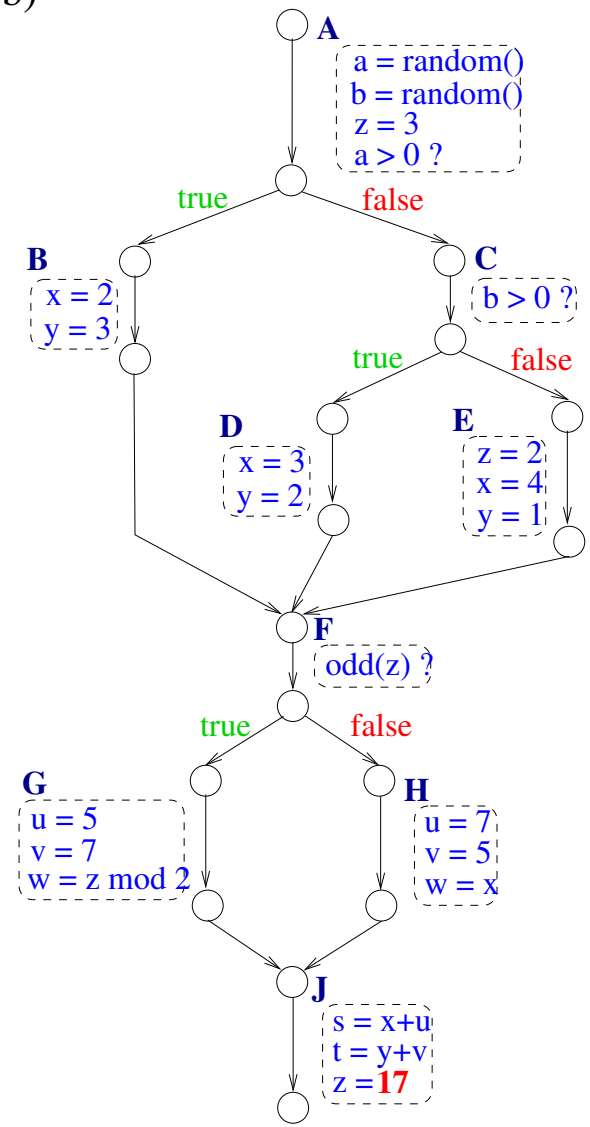
c)



The Deterministic Path–Precise Full Optimization

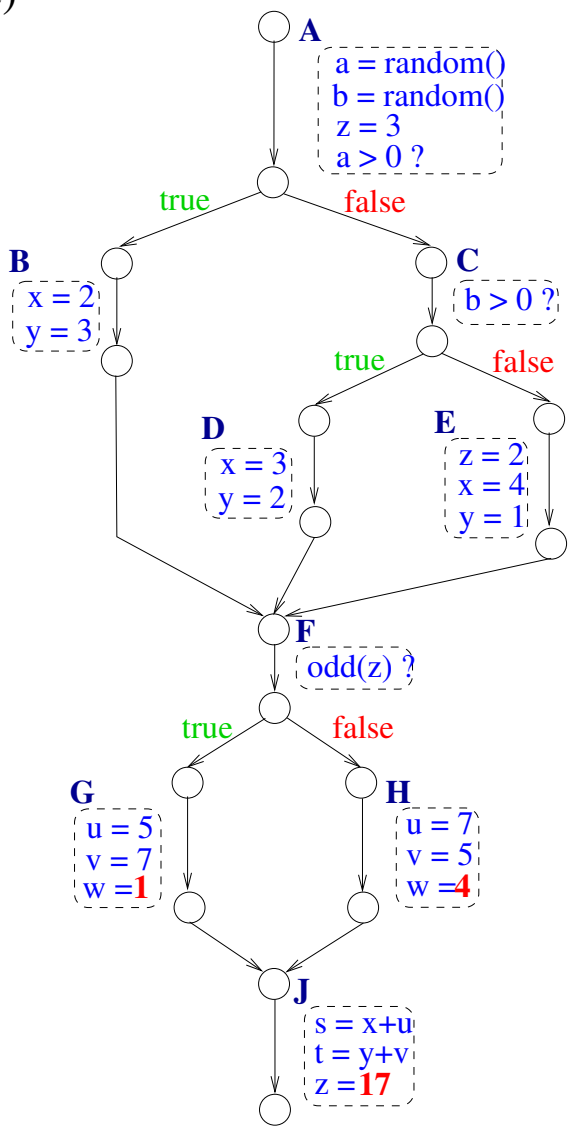
Beide Transformationen auf einen Blick

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.
```

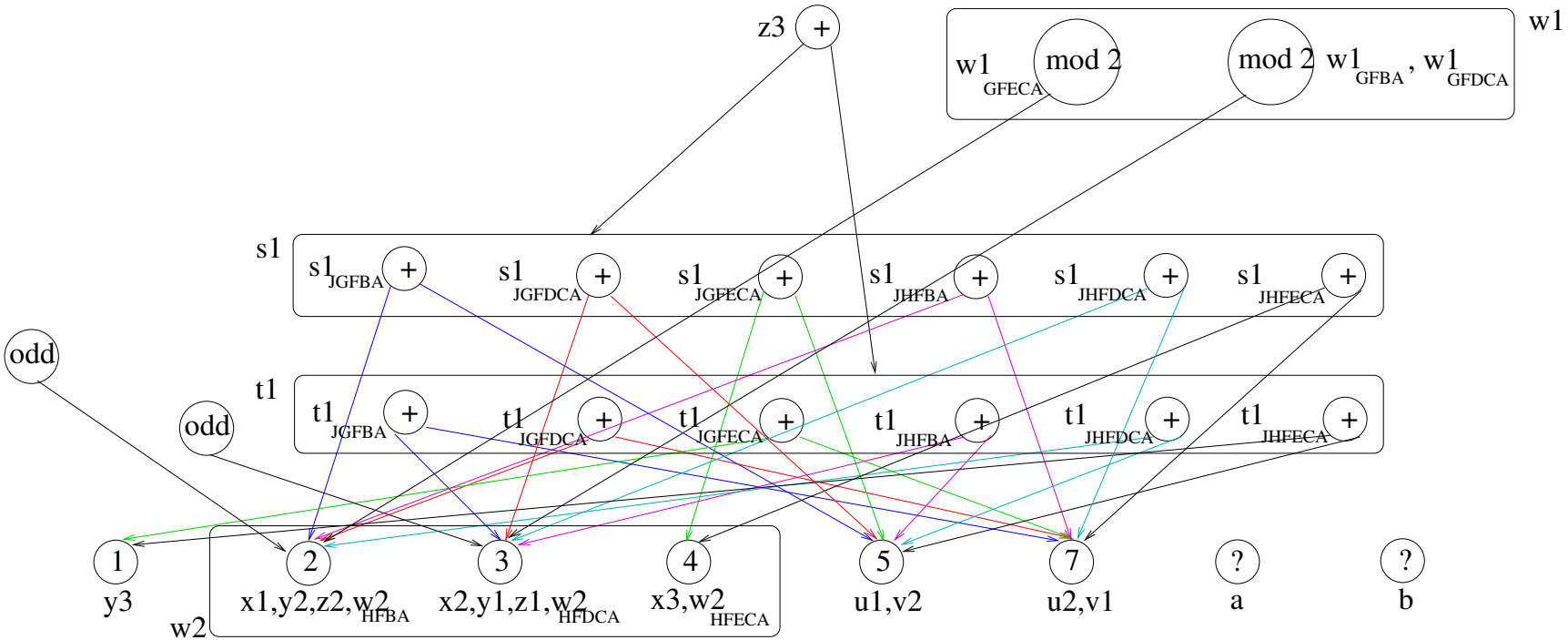
Prädikatierte SSA-Form (PSSA-Form)

...von Carter, Simon, Calder, Ferrante (PACT'99):

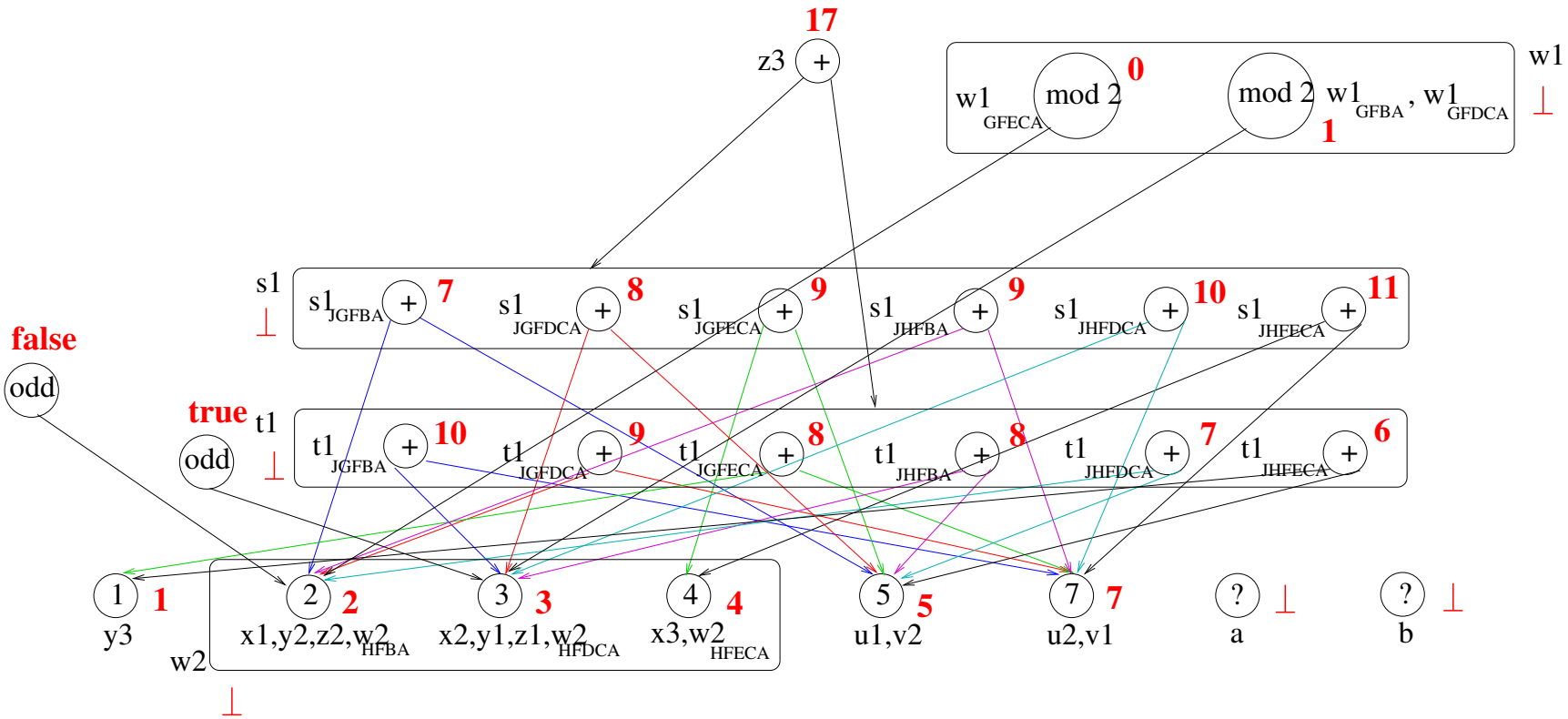
```
begin (p0)     A = OR(TRUE);                    | [*] (HFBA)   w2 = x1;
      (A)      (a1,b1) = (random(),random());   | [*] (HFDCA)  w2 = x2;                 |
      (A)      z1 = 3;                          |     (HFECA)  w2 = x3;
      (A)      cmp.unc BA,CA (a1>0);            |     (H)      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):

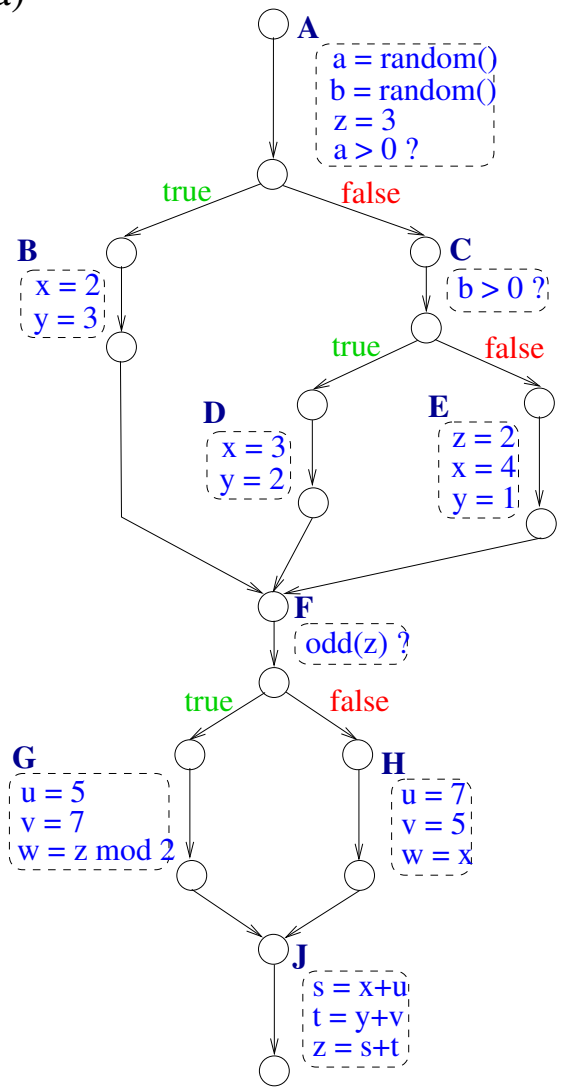


Nach Konstantenfaltung durch PVG-Grundalg.



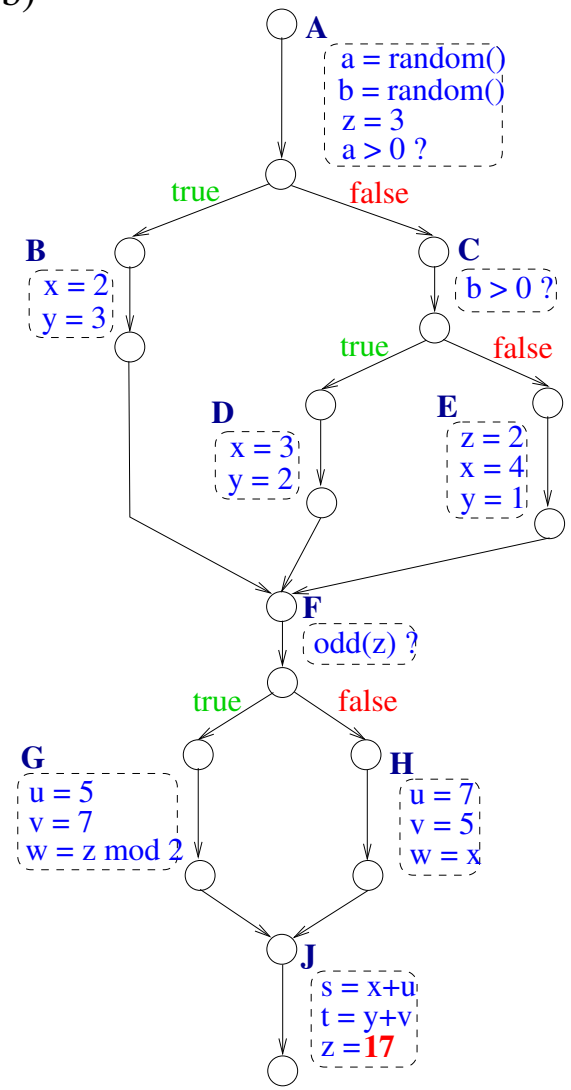
Die PVG-Grundalgorithmustransformation

a)



Original Hyperblock

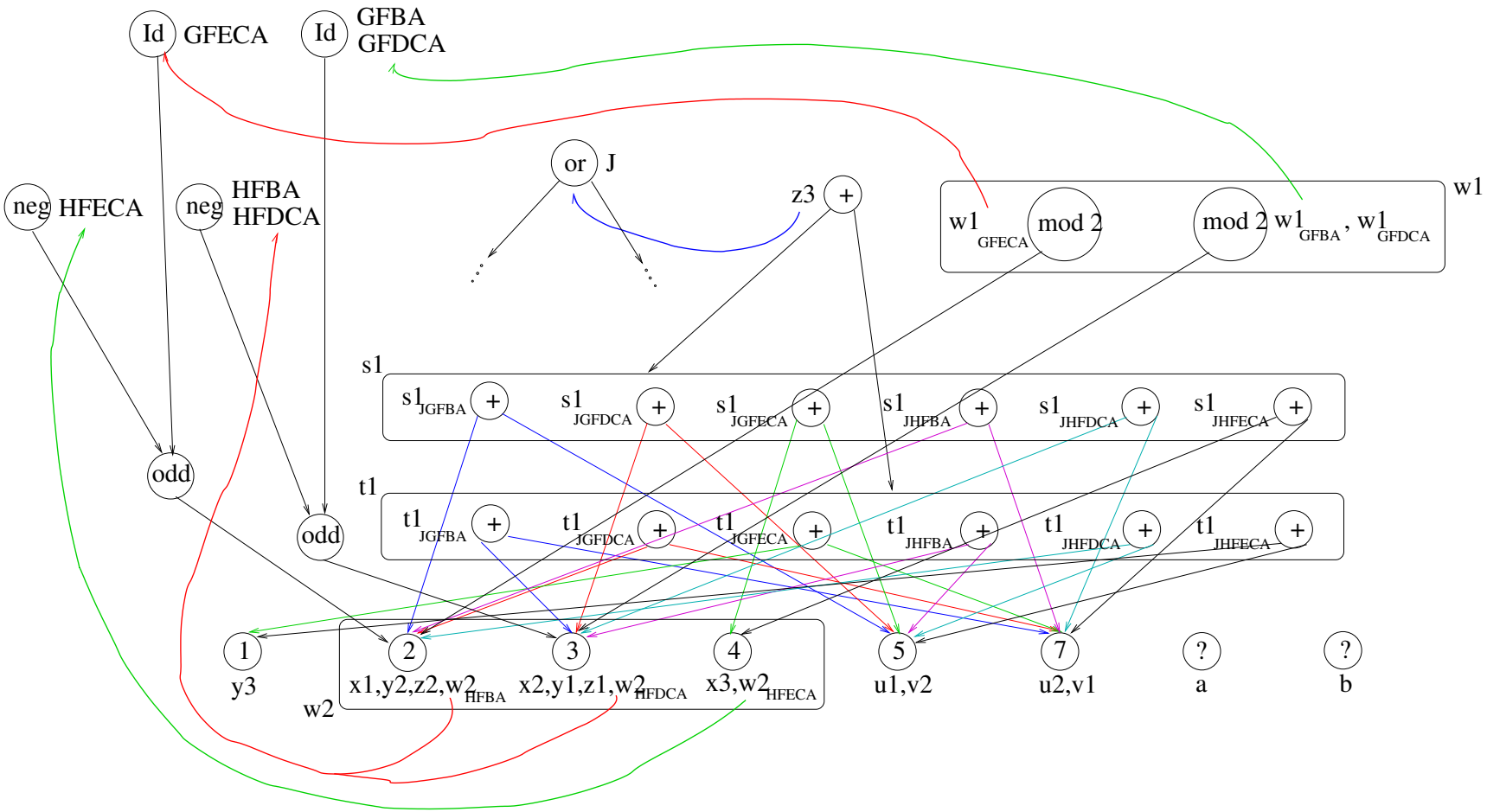
b)



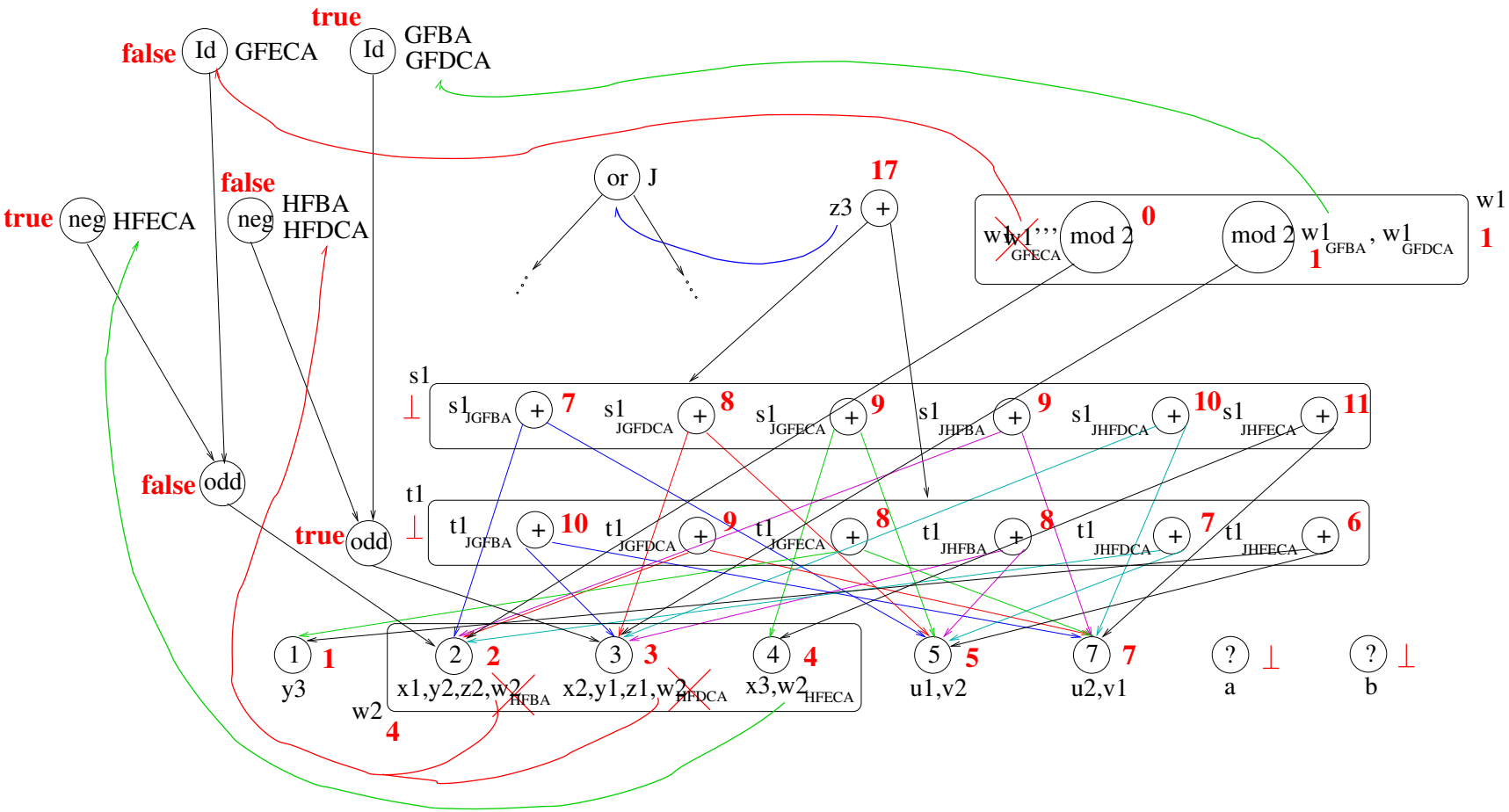
The Non–Deterministic Path–Precise Basic Optimization

Der prädikatierte Wertegraph

...unter Ausnutzung der Wächterprädikate (engl. guarding predicates):

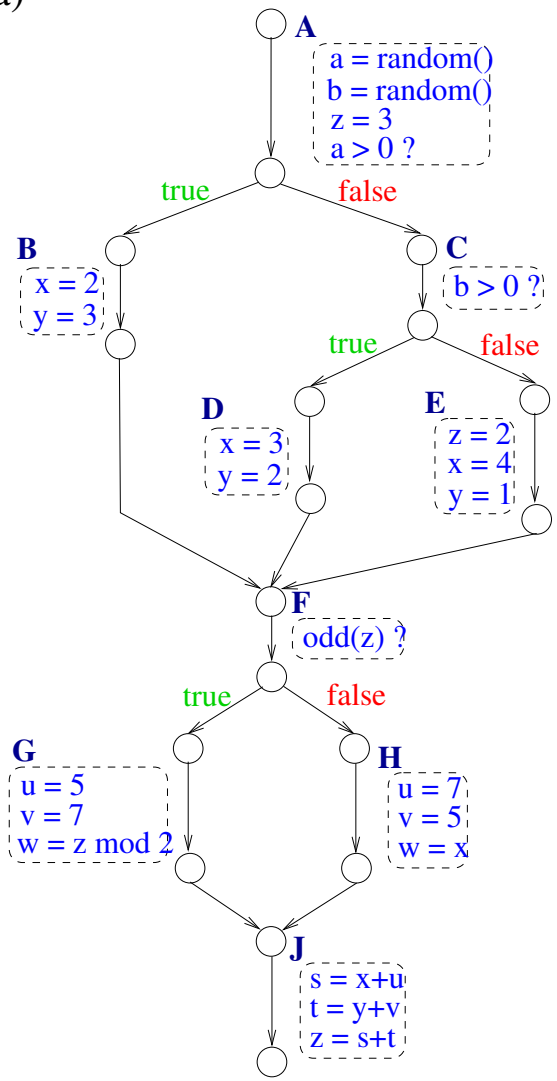


Nach Konstantenfaltung durch vollen PVG-A.



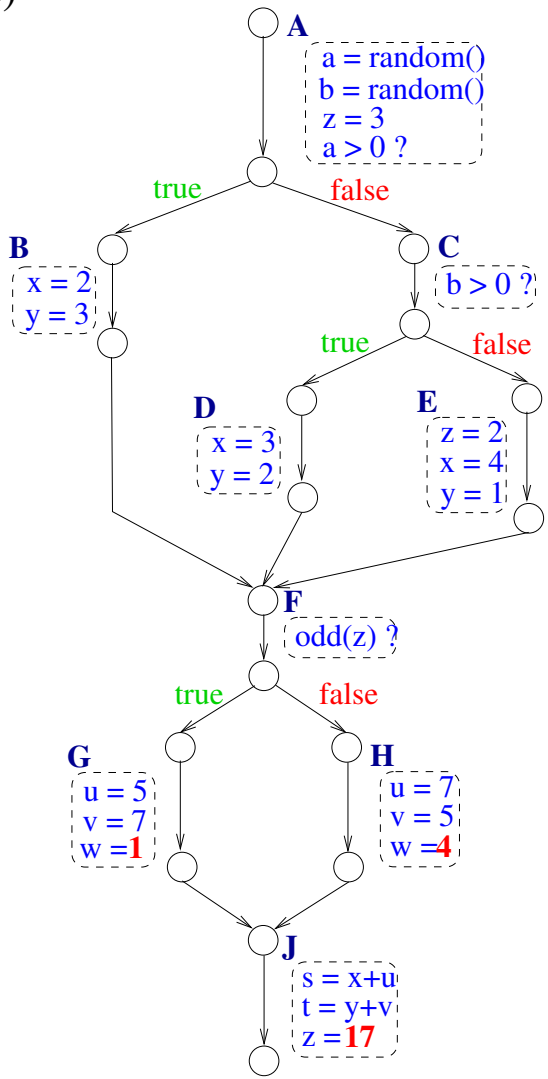
Die volle PVG-Algorithmustransformation

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

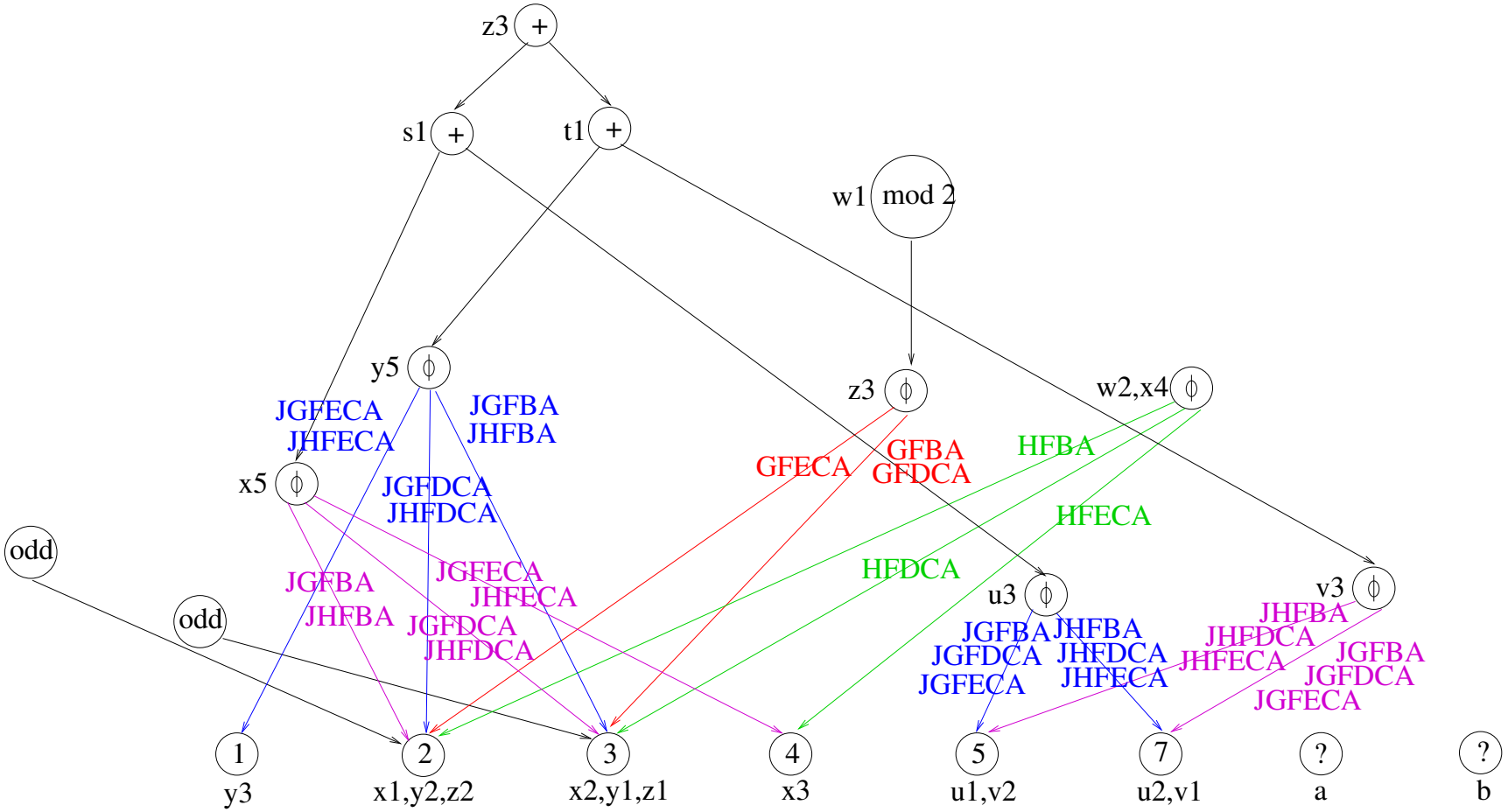
Korrektheit

- Der globale Konstantenfaltungsalgorithmus ist korrekt (sowohl für die Grundform wie für die volle Variante der lokalen Analysestufe).

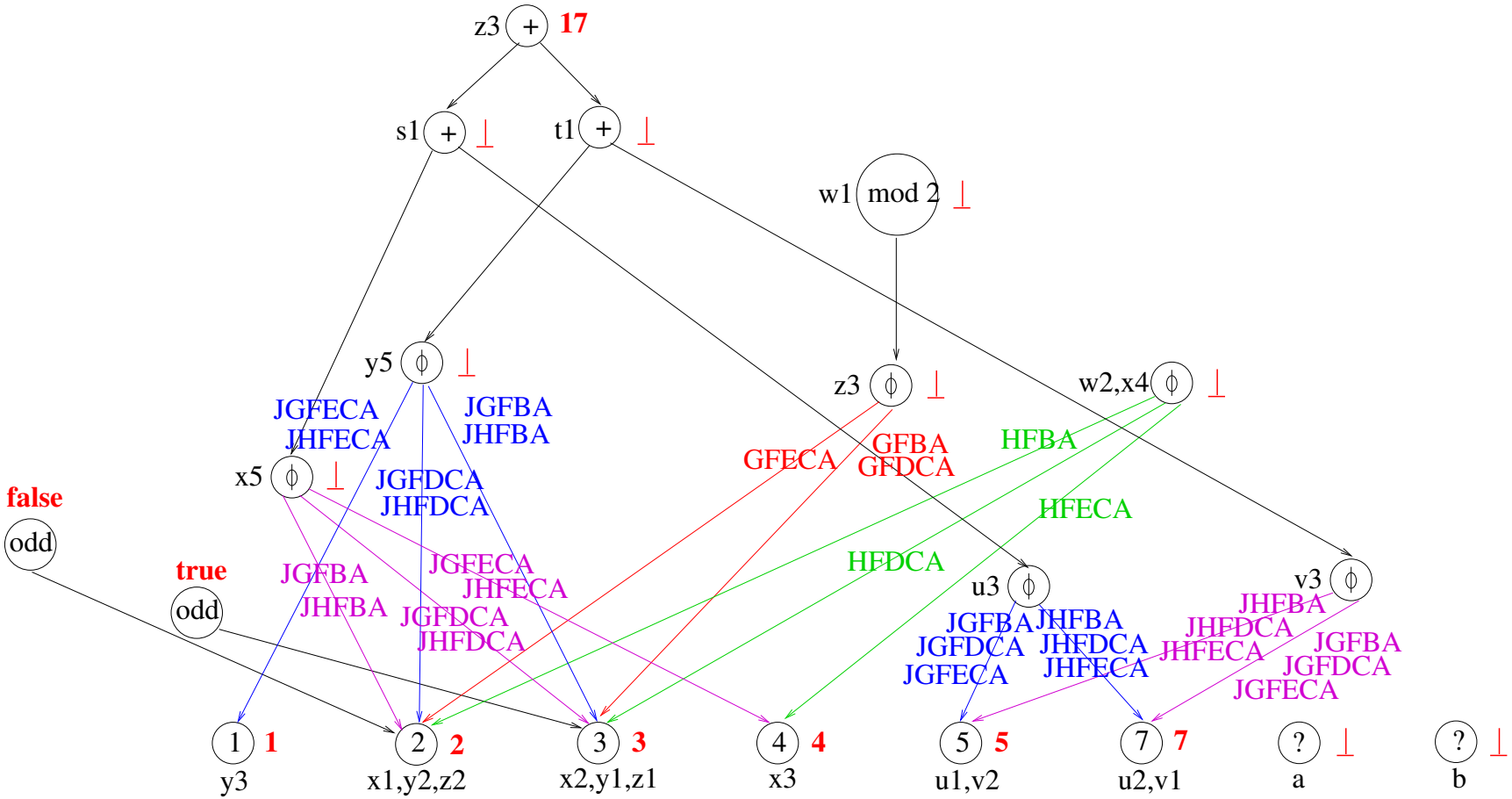
Vollständigkeit/Optimalität

- Der Grundalgorithmus der lokalen Analysestufe ist pfad-präzise bezüglich. nicht-deterministischer Interpretation von Verzweigungsbedingungen.
- Der volle Algorithmus der lokalen Analysestufe ist prädikatssensitiv pfad-präzise.

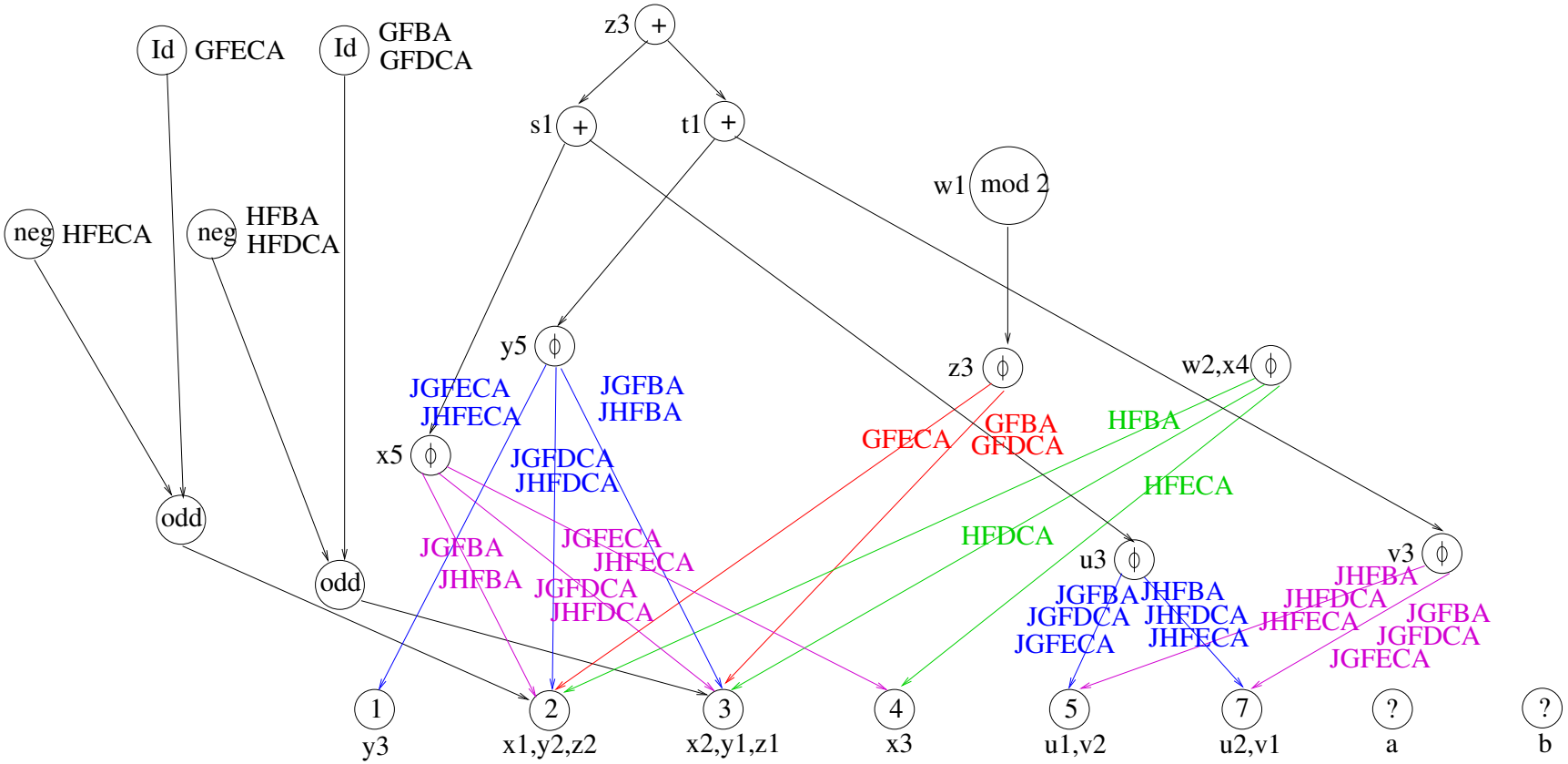
Performanzsteigerung: Grundalgorithmus (1)



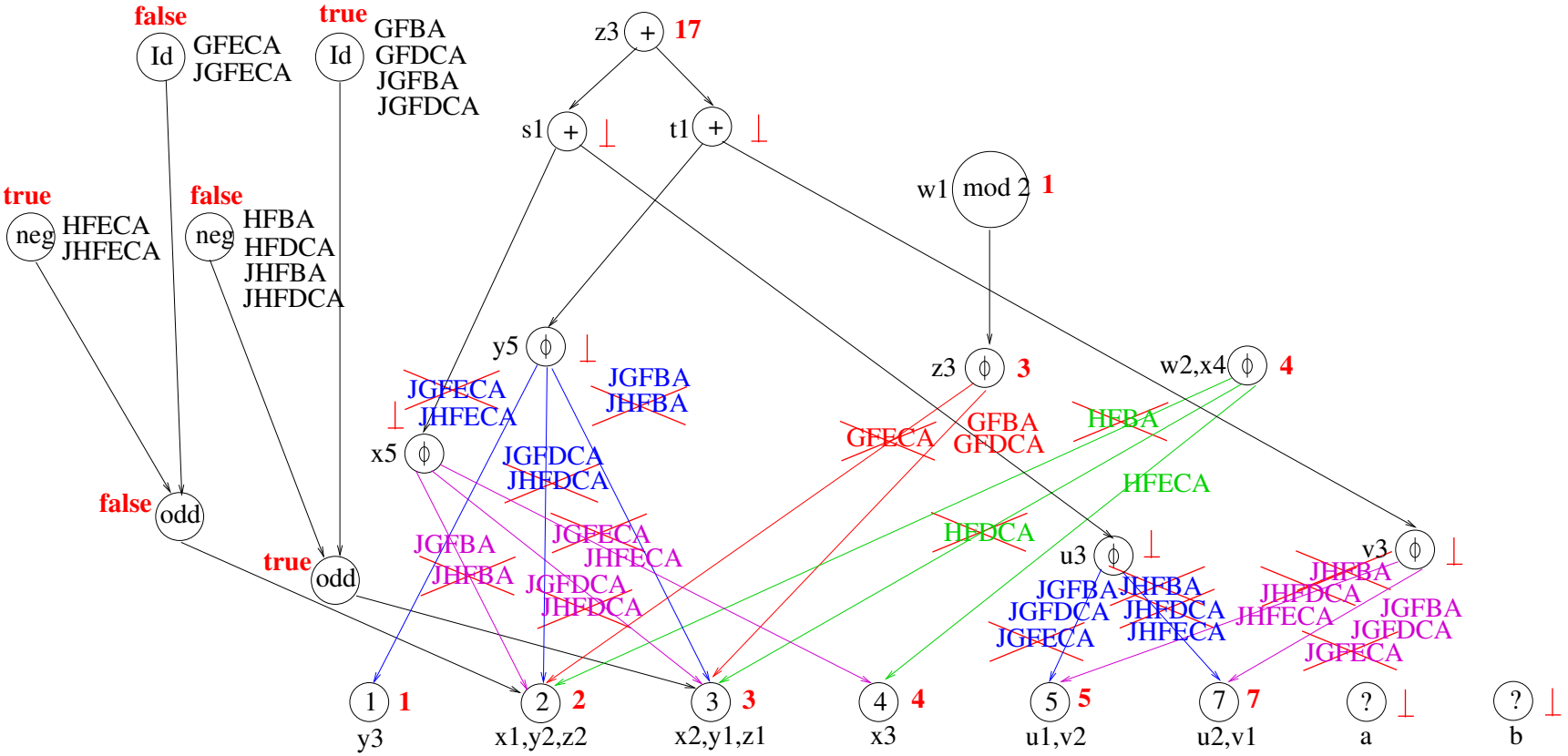
Performanzsteigerung: Grundalgorithmus (2)



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.:

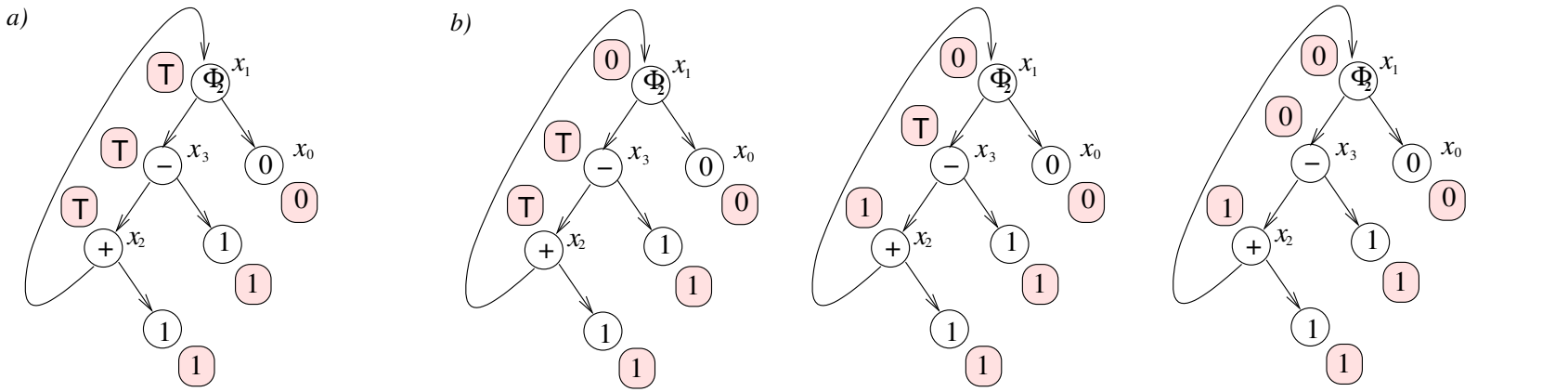
- Wertegraph: Bedingte Konstanten (engl. conditional constants).

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**asy!



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).

Ergänzende, weiterführende und vertiefende 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.

Ergänzende, weiterführende und vertiefende 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.

Ergänzende, weiterführende und vertiefende 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).


J.H. Reif, 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.

Ergänzende, weiterführende und vertiefende 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.

Ergänzende, weiterführende und vertiefende 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:

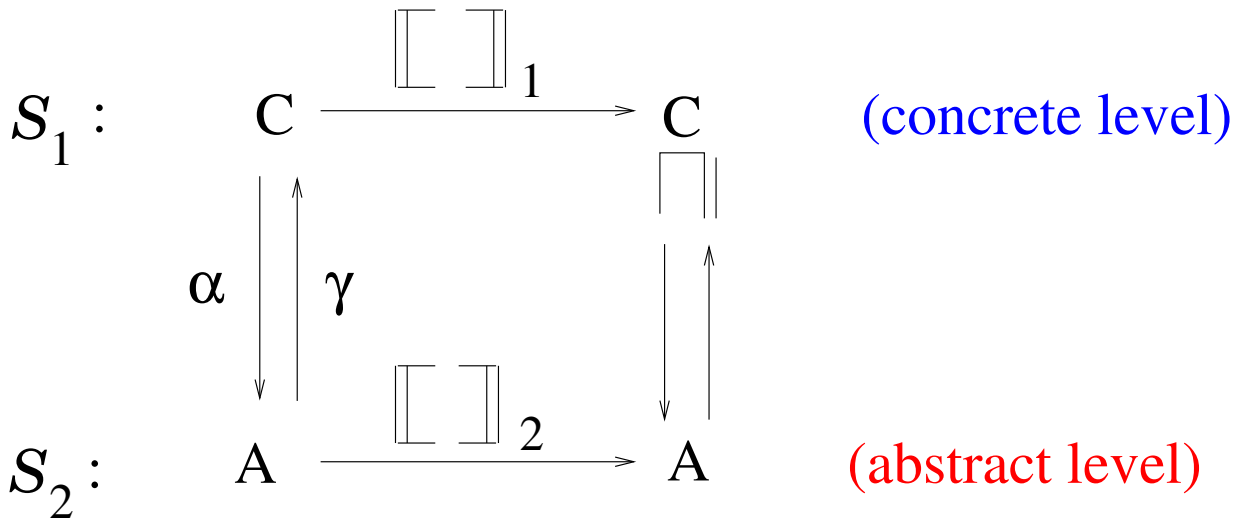
- Der DFA-Ansatz beinhaltet die Spezifikation einer Programmanalyse, deren Korrektheit separat und unabhängig *a posteriori* validiert wird.
- Der abstrakte Interpretationsansatz beinhaltet den Korrektheitsnachweis von Anfang an als integralen Bestandteil der Programmanalyse.

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.*
   $\forall\, a_1, a_2 \in A.\ \gamma(a_1) \sqsubseteq_C \gamma(a_2) \iff a_1 \sqsubseteq_A a_2$.

Konstruktion von Galois-Einpassungen

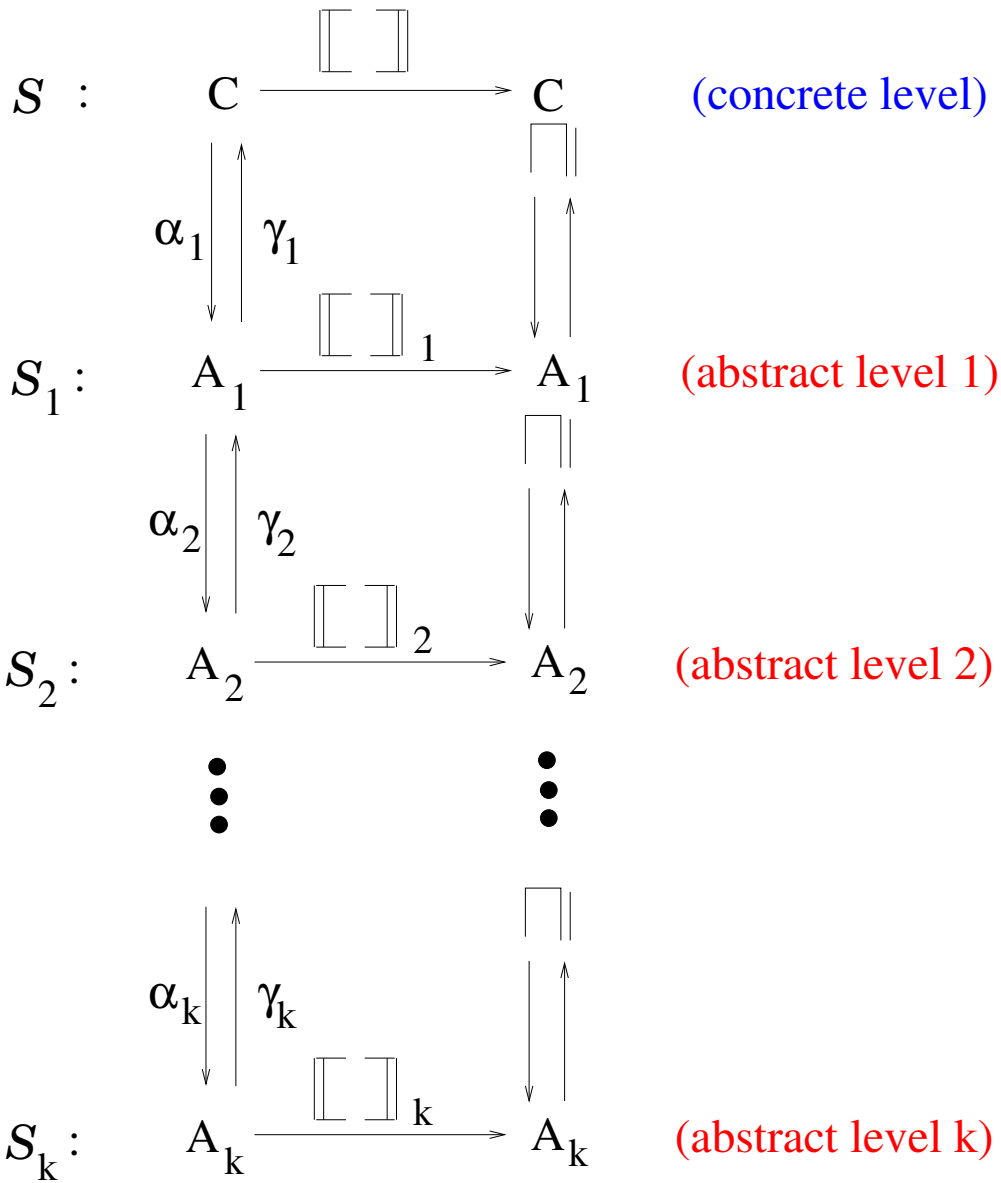
Lemma (15.1.7, Galois-Einpassungskonstruktion)

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*

$$\begin{aligned} \alpha(C') &= \bigcup \{\alpha_1(\{c\}) \times \alpha_2(\{c\}) \mid c \in C'\} \\ \gamma(AA) &= \{c \mid \alpha_1(\{c\}) \times \alpha_2(\{c\}) \subseteq AA\} \end{aligned}$$

für $C' \subseteq C$ *und* $AA \subseteq A_1 \times A_2$ *eine Galois-Verbindung.*

Kapitel 15.3

Korrektheit abstrakter Interpretationen

Korrektheit

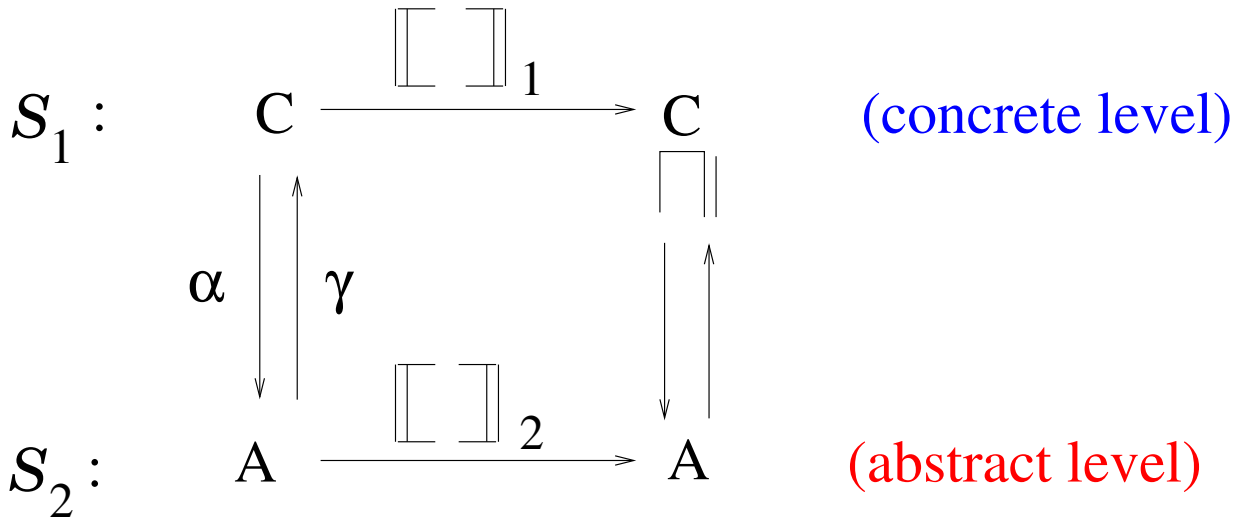
Der in seiner Grund- bzw. erweiterten Form auf

- 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 $\mathbf{s}$ keine Vorgänger, $\mathbf{e}$ keine Nachfolger hat; $\mathbf{P}(G)$ die Menge aller Pfadevon $\mathbf{s}$ nach $\mathbf{e}$ 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 $\mathbf{s}$ und $\mathbf{e}$,
- $\mathcal{C} =_{df} (C, \sqcup, \sqsubseteq, \bot, \top)$ einen vollständigen Halbverband mit kleinstem Element $\bot$ und größtem Element $\top$.

Lokale abstrakte Semantik

Definition (15.3.1, 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 (15.3.2, 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 (15.3.3, 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 (15.3.4, 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 (15.3.5, 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 (15.3.6, 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.

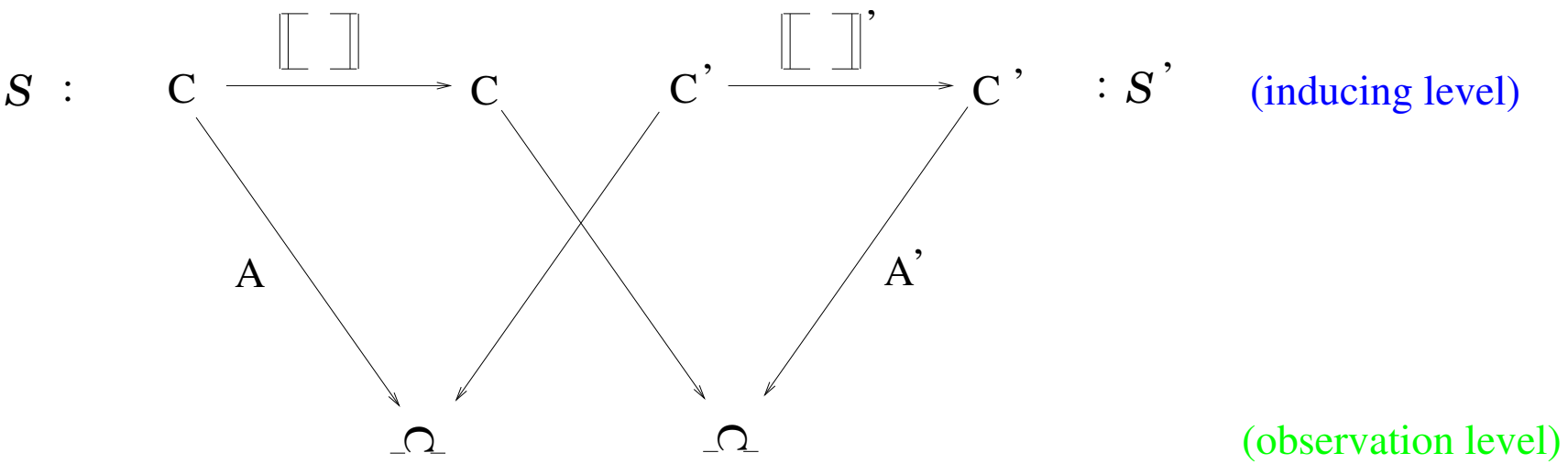
Beobachtungsäquivalenz

Definition (15.3.7, 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 (15.3.8, Äquivalenzrelation)

Die Relation $\approx_\Omega$ ist eine Äquivalenzrelation auf der Menge aller Modelle von Ω.

Lokale Optimalität (1)

Definition (15.3.9, 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)

Definition (15.3.9, 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 (15.3.10, 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 (15.3.11, 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 (15.3.12, Abschneidetheorem)

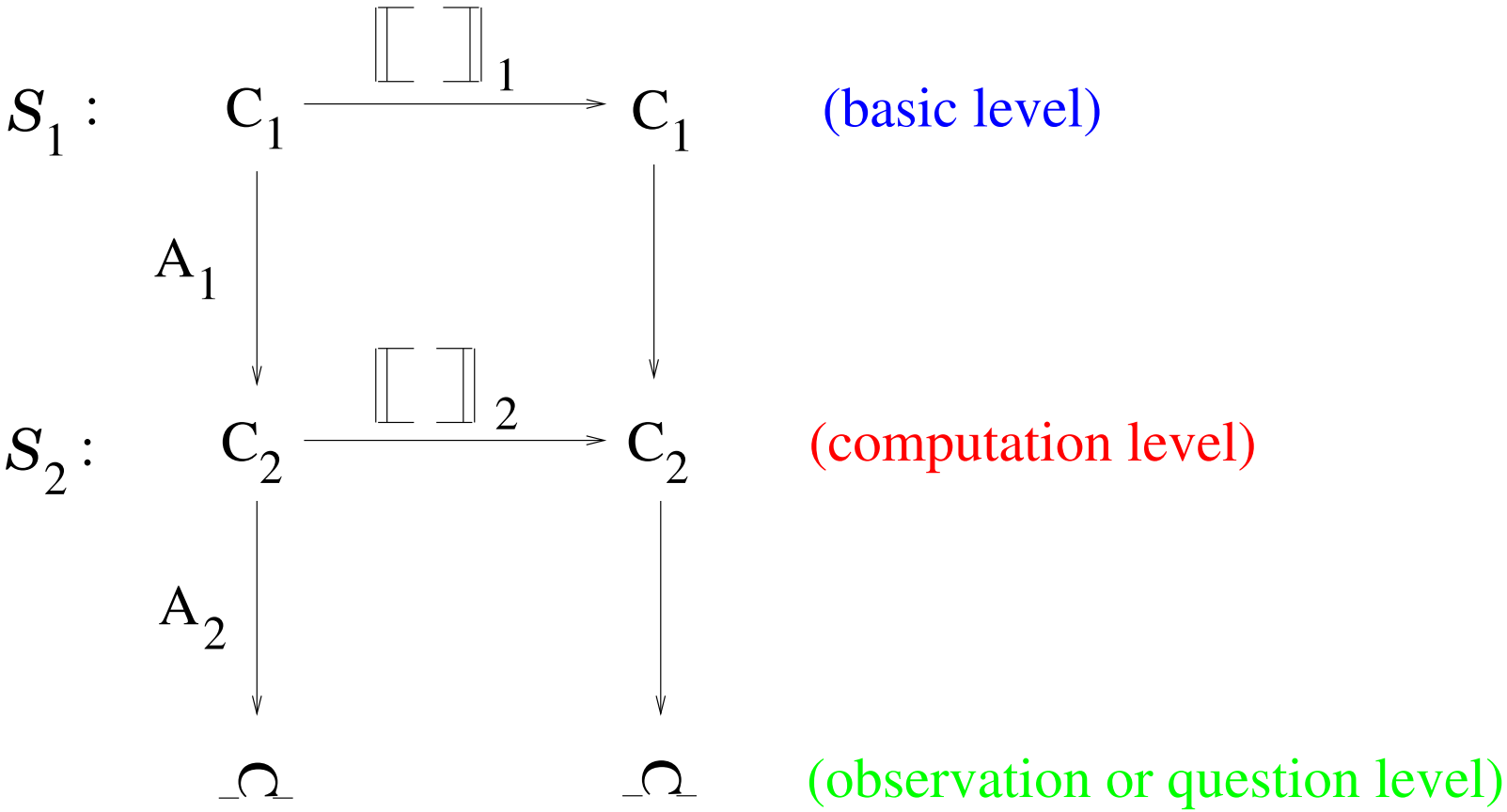
Sei $(\mathcal{S}_2, A_2) \in \Phi(\mathcal{S}_1, A_2 \circ A_1, \Omega)$ *mit* $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$. *Dann gilt:*

$$\forall\, G \in \mathbf{FG}.\ A_1 \circ [\![\ G\ ]\!]_1 \circ A_1^a = [\![\ G\ ]\!]_2$$

Insbesondere gilt weiters:

$$(\mathcal{S}_1, A_2 \circ A_1) \approx_\Omega (\mathcal{S}_2, A_2)$$

Veranschaulichung



Äquivalenz

Theorem (15.2.13, 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

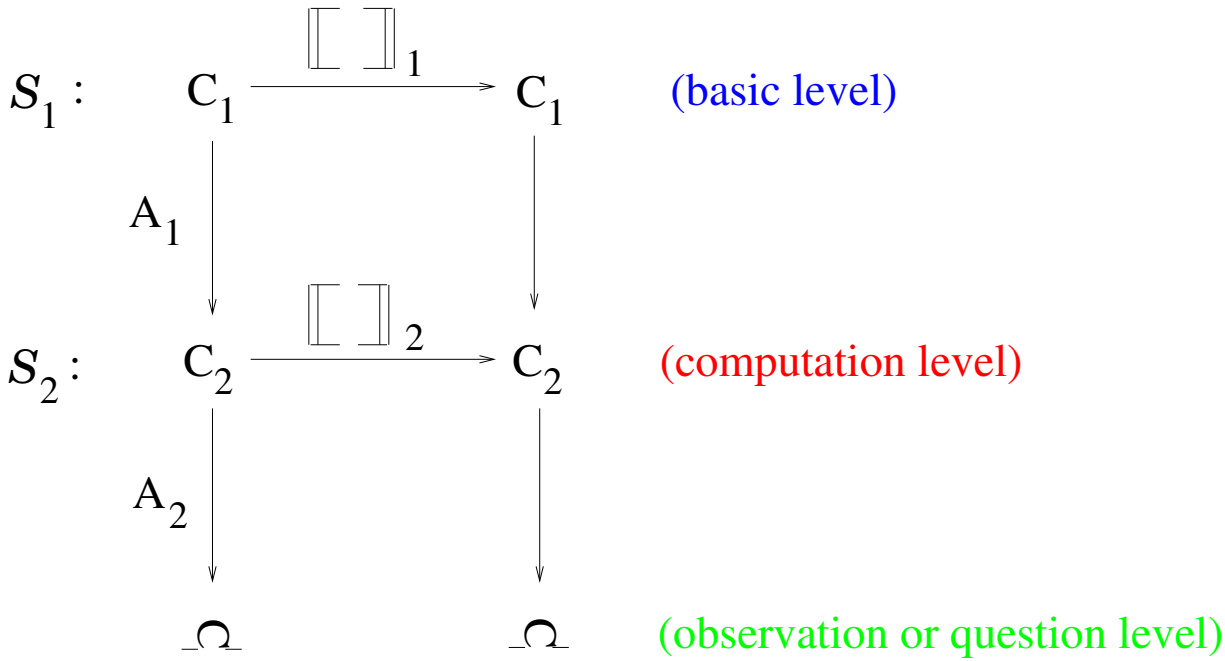
- vollständige bzw. optimale Modell.

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.



Ergänzende, weiterführende und vertiefende 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.

Ergänzende, weiterführende und vertiefende 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.

Ergänzende, weiterführende und vertiefende 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.

Ergänzende, weiterführende und vertiefende 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.

Ergänzende, weiterführende und vertiefende 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, Michael Monerau. *Probabilistic Abstract Interpretation*. In Proceedings 21st Symposium on Programming (ESOP 2012), Springer-V., LNCS 7211, 169-193, 2012.

Ergänzende, weiterführende und vertiefende 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.


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.


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)

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 15 (7)

Bernhard Steffen. *Optimal Run Time Optimization – Proved by a New Look at Abstract Interpretation*. In Proceedings of the 2nd Joint International Conference on the Theory and Practice of Software Development (TAPSOFT'87), Springer-V., LNCS 249, 52-68, 1987.

Bernhard Steffen. *Optimal Data Flow Analysis via Observational Equivalence*. In Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS'89), Springer-V., LNCS 379, 492-502, 1989.

Kapitel 16

Modellprüfung und DFA

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:

$$\text{DFA-Algorithmus für Eigenschaft E} \ :$$
$$\text{Programme} \ \rightarrow \ \text{Menge der E erfüllenden Programmpunkte}$$

Modellprüfung:

$$\text{Modellprüfer} \ : \ \text{(Modallogische) Formel} \times \text{Modell}$$
$$\rightarrow \ \text{Menge der die Formel erfüllenden Zustände}$$

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:

$$OCP =_{df} EAR \wedge NEC$$

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, 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.

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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 16 (1)


B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Peit, L. Petrucci, Ph. Schnoebelen with P. McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-V., 2001.


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.


Orna Grumberg, Helmut Veith. *25 Years of Model Checking: History, Achievements, Perspectives*. Springer-V., LNCS 5000, 2008.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 16 (2)

- Janusz Laski, William Stanley. *Software Verification and Analysis: An Integrated, Hands-On Approach*. Springer-V., 2009.
- 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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 16 (3)

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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 16 (4)


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.

Teil V

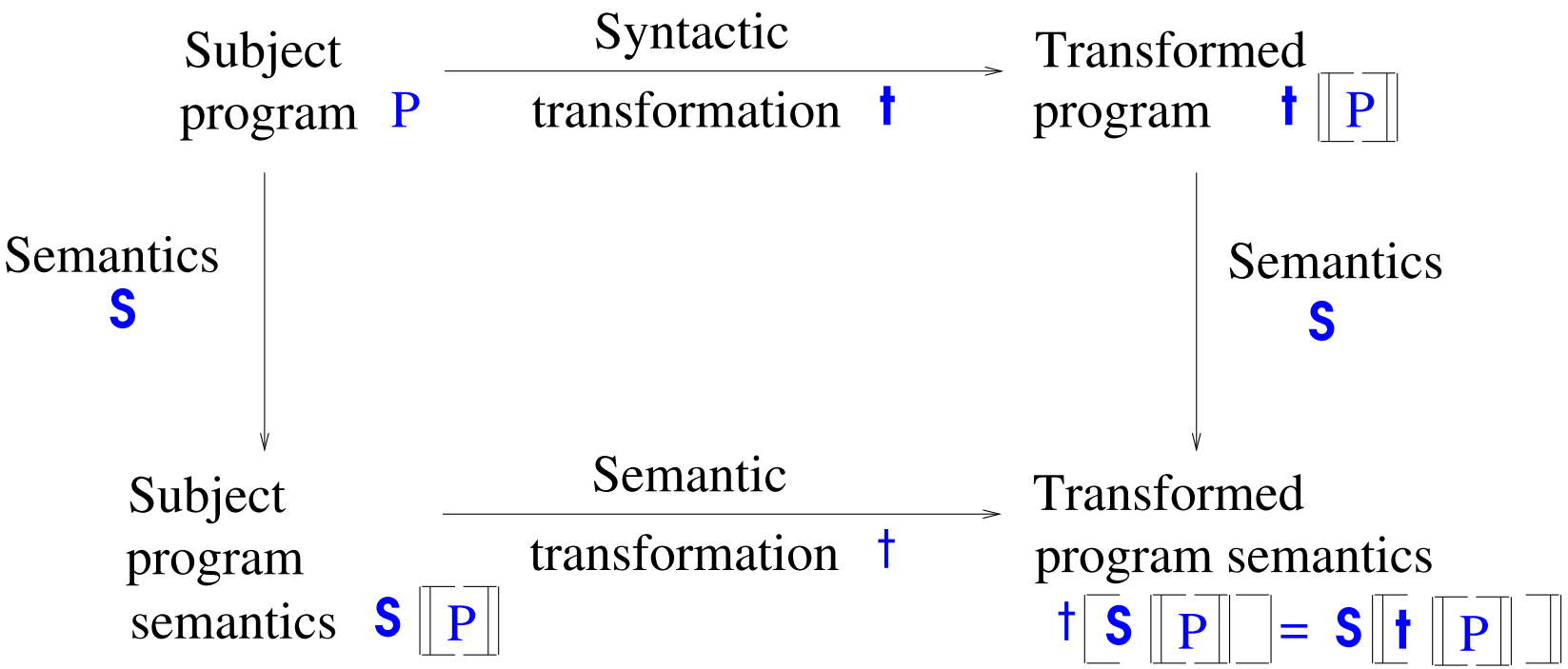
Abschluss und Ausblick

Kapitel 17

Resümee und Perspektiven

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.

Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 17

Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.

Nevin Heintze, Joxan Jaffar, Răzvan Voicu. *A Framework for Combining Analysis and Verification*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 26-39, 2000.

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.

Literatur

Literaturhinweise und Leseempfehlungen

...zum weiterführenden und vertiefenden Selbststudium.

- I Lehrbücher
- II Artikel, Dissertationen, Sammelbände

I Lehrbücher (1)


Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley, 2nd edition, 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*. Springer-V., 3rd edition, 2009.

I Lehrbücher (2)


Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 2nd edition, Springer-V., 2001.


B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Peit, L. Petrucci, Ph. Schnoebelen with P. McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-V., 2001.


Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004.


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


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

I Lehrbücher (3)


Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009.


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


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.

I Lehrbücher (4)


Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007.


Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. 2nd edition, Springer-V., 2005.


Doron A. Peled. *Software Reliability Methods*. Springer-V., 2001.

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.


G. 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, 431 - 483, 1981.


W. A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part II - Demand Analysis*. Acta Informatica 10(3):265-272, 1978.


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.

II Artikel, Dissertationen, Sammelbände (3)


Ras Bodik, Rajiv Gupta. *Partial Dead Code Elimination using Slicing Transformations*. In Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation (PLDI'97), ACM SIGPLAN Notices 32(6):159-170, 1997.


Ras Bodík, Rajiv Gupta, Vivek Sarkar. *ABCD: Eliminating Array Bounds Check on Demand*. In Proceedings of the ACM SIGPLAN Conference on Porgramming Language Design and Implementation (PLDI'00), ACM SIGPLAN Notices 35(5):321-333, 2000.


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.

II Artikel, Dissertationen, Sammelbände (4)

- 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.
- 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.

II Artikel, Dissertationen, Sammelbände (5)


Patrick Cousot. *Verification by Abstract Interpretation*. In Proceedings of the 4th International Conference on Verification, Model-Checking, Abstract Interpretation (VMCAI 2003), Springer-V., LNCS 2575, 20-24, 2003.


Patrick Cousot. *Verification by Abstract Interpretation*. In Verification: Theory and Practice, Essays dedicated to Zohar Manna on the Occasion of His 64th Birthday. Springer-V., LNCS 2772, 243-268, 2003.


Patrick Cousot, Radhia Cousot. *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 (6)


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 (7)


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, Michael Monerau. *Probabilistic Abstract Interpretation*. In Proceedings 21st Symposium on Programming (ESOP 2012), Springer-V., LNCS 7211, 169-193, 2012.

II Artikel, Dissertationen, Sammelbände (8)


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 (TOPLAS) 13(4):451-490, 1991.


D. M. Dhamdhere. *Register Assignment using Code Placement Techniques*. Journal of Computer Languages 13(2):75-93, 1988.

II Artikel, Dissertationen, Sammelbände (9)


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.


Evelyn Duesterwald. *A Demand-driven Approach for Efficient Interprocedural Data-Flow Analysis*. PhD thesis, University of Pittsburgh, 1996.

II Artikel, Dissertationen, Sammelbände (10)


Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *Demand-driven Computation of Interprocedural Data Flow*. In Conference Record of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95), 37-48, 1995.


Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Demand-driven Analyzer for Data Flow Testing at the Integration Level*. In Proceedings of the IEEE Conference on Software Engineering (CoSE'96), 575-586, 1996.


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.

II Artikel, Dissertationen, Sammelbände (11)


L. Feigen, D. Klappholz, R. Casazza, X. Xue. *The Revival Transformation*. In Conference Record of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of of Programming Languages (POPL'94), 1994.


R.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.


Alfons Geser, Jens Knoop, Gerald Lüttgen, Oliver Rüthing, Bernhard Steffen. *Non-monotone Fixpoint Iterations to Resolve Second Order Effects*. In Proceedings CC'96, Springer-V., LNCS 1060, 106-120, 1996.

II Artikel, Dissertationen, Sammelbände (12)


Orna Grumberg, Helmut Veith. *25 Years of Model Checking: History, Achievements, Perspectives*. Springer-V., LNCS 5000, 2008.


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.


C.A.R. Hoare. *An Axiomatic Basis for Computer Programming*. Communications of the ACM 12:576-580, 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 (13)


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.


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.

II Artikel, Dissertationen, Sammelbände (14)

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, 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.

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.

II Artikel, Dissertationen, Sammelbände (15)


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.


Jens Knoop, Eduard Mehofer. *Distribution Assignment Placement: Effective Optimization of Redistribution Costs*. IEEE Transactions on Parallel and Distributed Systems 13(6):628-647, 2002.

II Artikel, Dissertationen, Sammelbände (16)


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Lazy Code Motion*. In Proceedings of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation (PLDI'92), ACM SIGPLAN Notices 27(7):224-234, 1992.


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Partial Dead Code Elimination*. In Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI'94), ACM SIGPLAN Notices 29(6):147-158, 1994.


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Optimal Code Motion: Theory and Practice*. ACM Transactions on Programming Languages and Systems 16(4):1117-1155, 1994.

II Artikel, Dissertationen, Sammelbände (17)


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *The Power of Assignment Motion*. In Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI'95), ACM SIGPLAN Notices 30(6):233-245, 1995.


Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Code Motion and Code Placement: Just Synonyms?* In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 154-169, 1998.


Jens Knoop, Oliver Rüthing. *Constant Propagation on the Value Graph: Simple Constants and Beyond*. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 94-109, 2000.

II Artikel, Dissertationen, Sammelbände (18)


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.


Laura Kovács and 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), Timisoara, Romania, October 1-4, 2003.
`apache.risc.uni-linz.ac.at/internals/ActivityDB/publications/download/risc_464/synasc03.pdf`

II Artikel, Dissertationen, Sammelbände (19)


Laura Kovács and Tudor Jebelean. *Generation of Invariants in Theorema*. In Proceedings of the 10th International Symposium of Mathematics and its Applications, Timisoara, Romania, November 6-9, 2003. `www.theorema.org/publication/2003/Laura/Poli_Timisoara_nov.pdf`


J.-L. Lassez, V.L. Nguyen, E.A. Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.


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, 1999.

II Artikel, Dissertationen, Sammelbände (20)


Kim Marriot. *Frameworks for Abstract Interpretation*. Acta Informatica 30:103-129, 1993.


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


R. J. Mintz, G. A. Fisher, M. Sharir. *The Design of a Global Optimizer*. In Proceedings of the ACM SIGPLAN'79 Symposium on Compiler Construction (SCC'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.

II Artikel, Dissertationen, Sammelbände (21)


Flemming Nielson. *A Bibliography on Abstract Interpretations*. ACM SIGPLAN Notices 21:31-38, 1986.


Ernst-Rüdiger Olderog, Bernhard Steffen. *Formale Semantik und Programmverifikation*. In Informatik-Handbuch, P. Rechenberg, G. Pomberger (Hrsg.), Carl Hanser Verlag, 129 - 148, 1997.


J.H. Reif, 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.


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.

II Artikel, Dissertationen, Sammelbände (22)


Thomas Reps. *Demand Interprocedural Program Analysis using Logic Databases*. In Applications of Logic Databases, R. Ramakrishnan (Ed.), Kluwer Academic Publishers, 1994.


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.


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.

II Artikel, Dissertationen, Sammelbände (23)


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.


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.

II Artikel, Dissertationen, Sammelbände (24)


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.


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.

II Artikel, Dissertationen, Sammelbände (25)

- Bernhard Steffen. *Data Flow Analysis as Model Checking*. In Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'91), Springer-V., LNCS 526, 346-365, 1991.
- Bernhard Steffen. *Generating Data Flow Analysis Algorithms from Modal Specifications*. International Journal on Science of Computer Programming 21:115-139, 1993.
- Bernhard Steffen. *Property-Oriented Expansion*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 22-41, 1996.
- Bernhard Steffen, Jens Knoop. *Finite Constants: Characterizations of a New Decidable Set of Constants*. Theoretical Computer Science 80(2):303-318, 1991.

II Artikel, Dissertationen, Sammelbände (26)


Munehiro Takimoto, Kenichi Harada. *Effective Partial Redundancy Elimination based on Extended Value Graph*. Information Processing Society of Japan 38(11):2237-2250, 1990.


Munehiro Takimoto, Kenichi Harada. *Partial Dead Code Elimination Using Extended Value Graph*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 179-193, 1999.


Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5:285-309, 1955.

II Artikel, Dissertationen, Sammelbände (27)


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.


Mark N. Wegman, F. Ken Zadeck. *Constant Propagation with Conditional Branches*. ACM Transactions on Programming Languages and Systems 13(2):181-201, 1991.


Daniel Weise. *Static Analysis of Mega-Programs (Invited Paper)*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 300-302, 1999.

II Artikel, Dissertationen, Sammelbände (28)

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):36.1-36.53, 2008.

X. Yuan, Rajiv Gupta, R. Melham. *Demand-driven Data Flow Analysis for Communication Optimization*. Parallel Processing Letters 7(4):359-370, 1997.

Anhang

A

Mathematische Grundlagen

Mathematische Grundlagen

...im Zusammenhang mit der Festlegung

- der denotationellen Semantik von WHILE,
- der abstrakten Semantiken für Programmanalysen,
- formaler Optimalitätskriterien von Programmtransformationen.

Insbesondere

- Mengen, Relationen und Verbände
- Partielle und vollständige partielle Ordnungen
- Fixpunkte und Fixpunkttheoreme

A.1

Mengen und Relationen

Mengen und Relationen

Sei M eine Menge, R eine Relation auf M, d.h. $R \subseteq M \times M$.

Dann heißt R

- reflexiv gdw $\forall\, m \in M.\ m\, R\, m$
- transitiv gdw $\forall\, m, n, p \in M.\ m\, R\, n\ \wedge\ n\, R\, p\ \succ\ m\, R\, p$
- antisymmetrisch gdw
  $\forall\, m, n \in M.\ m\, R\, n\ \wedge\ n\, R\, m\ \succ\ m = n$
- symmetrisch gdw $\forall\, m, n \in M.\ m\, R\, n\ \Longleftrightarrow\ n\, R\, m$
- total gdw $\forall\, m, n \in M.\ m\, R\, n\ \vee\ n\, R\, m$

A.2

Ordnungen

Ordnungen

Eine Relation R auf M heißt

- Quasiordnung gdw R ist reflexiv und transitiv
- partielle Ordnung gdw R ist reflexiv, transitiv und antisymmetrisch
- Äquivalenzrelation gdw R ist reflexiv, transitiv und symmetrisch

...eine partielle Ordnung ist also eine antisymmetrische Quasiordnung, eine Äquivalenzrelation eine symmetrische Quasiordnung.

Schranken und kleinste und größte Elemente

Sei $(Q, \sqsubseteq)$ eine Quasiordnung, sei $q \in Q$ und $Q' \subseteq Q$.

Dann heißt q

- obere (untere) Schranke von Q', in Zeichen: $Q' \sqsubseteq q$ ($q \sqsubseteq Q'$), wenn für alle $q' \in Q'$ gilt: $q' \sqsubseteq q$ ($q \sqsubseteq q'$).
- kleinste obere (größte untere) Schranke von Q', wenn q obere (untere) Schranke von Q' ist und für jede andere obere (untere) Schranke $\hat{q}$ von Q' gilt: $q \sqsubseteq \hat{q}$ ($\hat{q} \sqsubseteq q$).
- größtes (kleinstes) Element von Q, wenn gilt: $Q \sqsubseteq q$ ($q \sqsubseteq Q$).

Existenz und Eindeutigkeit von Schranken

...in partiellen Ordnungen.

- In partiellen Ordnungen sind kleinste obere und größte untere Schranken eindeutig bestimmt, wenn sie existieren.
- Existenz (und damit Eindeutigkeit) vorausgesetzt, wird die kleinste obere (größte untere) Schranke einer Menge $P' \subseteq P$ der Grundmenge einer partiellen Ordnung $(P, \sqsubseteq)$ mit $\bigsqcup P'$ ($\bigsqcap P'$) bezeichnet. Man spricht dann auch vom Supremum und Infimum von P'.
- Analog für kleinste und größte Elemente. Existenz vorausgesetzt, werden sie üblicherweise mit $\bot$ und $\top$ bezeichnet.

A.3

Verbände

Verbände und vollständige Verbände

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Dann heißt $(P, \sqsubseteq)$

- Verband, wenn jede endliche Teilmenge P' von P eine kleinste obere und eine größte untere Schranke in P besitzt.
- vollständiger Verband, wenn jede Teilmenge P' von P eine kleinste obere und eine größte untere Schranke in P besitzt.

...(vollständige) Verbände sind also spezielle partielle Ordnungen.

A.4
Vollständige partielle Ordnungen

Vollständige partielle Ordnungen

...ein etwas schwächerer als der Verbandsbegriff, aber für viele Problemstellungen in der Informatik ausreichend.

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Dann heißt $(P, \sqsubseteq)$

- vollständig, kurz CPO (engl. complete partial order), wenn jede aufsteigende Kette $K \subseteq P$ eine kleinste obere Schranke in P besitzt.

Es gilt:

- Eine CPO $(C, \sqsubseteq)$ (genauer wäre: "kettenvollständige partielle Ordnung (engl. chain complete partial order (CCPO)") besitzt stets ein kleinstes Element, eindeutig bestimmt als Supremum der leeren Kette und üblicherweise mit $\bot$ bezeichnet: $\bot =_{df} \bigsqcup \emptyset$.

Ketten

Sei $(P, \sqsubseteq)$ eine partielle Ordnung.

Eine Teilmenge $K \subseteq P$ heißt

- Kette in P, wenn die Elemente in K total geordnet sind.

  Für $K = \{k_0 \sqsubseteq k_1 \sqsubseteq k_2 \sqsubseteq \ldots\}$ $(\{k_0 \sqsupseteq k_1 \sqsupseteq k_2 \sqsupseteq \ldots\})$ spricht man auch genauer von einer aufsteigenden (absteigenden) Kette in P.

Eine Kette K heißt

- endlich, wenn K endlich ist, sonst unendlich.

Kettenendlichkeit, endliche Elemente

Eine partielle Ordnung $(P, \sqsubseteq)$ heißt

- kettenendlich gdw P enthält keine unendlichen Ketten.

Ein Element $p \in P$ heißt

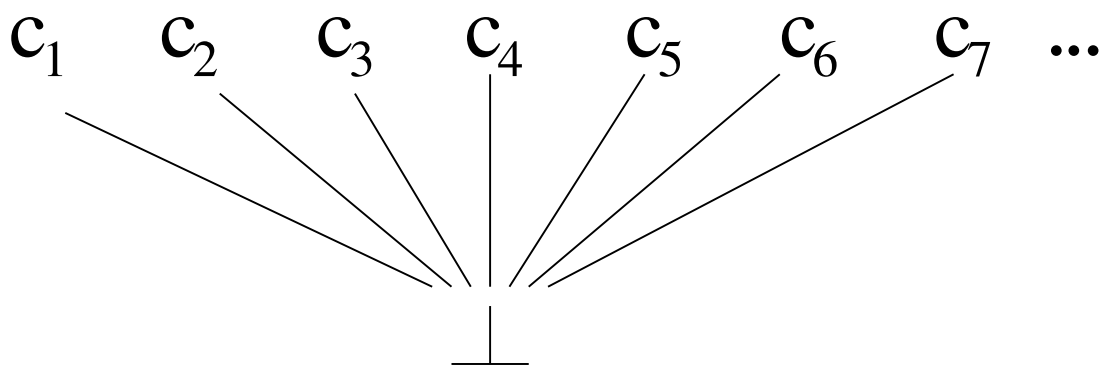
- endlich gdw die Menge $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$ keine unendliche Kette enthält.
- endlich relativ zu $r \in P$ gdw die Menge $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$ keine unendliche Kette enthält.

(Standard-) CPO-Konstruktionen (1)

Flache CPOs.

Sei $(C, \sqsubseteq)$ eine CPO. Dann heißt $(C, \sqsubseteq)$

- flach, wenn für alle $c, d \in C$ gilt:
  $c \sqsubseteq d \Leftrightarrow c = \bot \ \vee \ c = d$



(Standard-) CPO-Konstruktionen (2)

Produktkonstruktion:

Seien $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ CPOs.
Dann sind auch

- das nichtstrikte (direkte) Produkt $(\times P_i, \sqsubseteq)$ mit
  - $(\times P_i, \sqsubseteq) = (P_1 \times P_2 \times \ldots \times P_n, \sqsubseteq)$ mit
    $\forall (p_1, p_2, \ldots, p_n),$
    $(q_1, q_2, \ldots, q_n) \in \times P_i.\ (p_1, p_2, \ldots, p_n) \sqsubseteq$
    $(q_1, q_2, \ldots, q_n) \succ \forall\, i \in \{1, \ldots, n\}.\ p_i \sqsubseteq_i q_i$
- das strikte (direkte) Produkt (smash Produkt) mit
  - $(\bigotimes P_i, \sqsubseteq) = (P_1 \otimes P_2 \otimes \ldots \otimes P_n, \sqsubseteq)$, wobei $\sqsubseteq$ wie oben definiert ist, jedoch zusätzlich gesetzt wird:

$$(p_1, p_2, \ldots, p_n) = \bot \Rightarrow \exists\, i \in \{1, \ldots, n\}.\ p_i = \bot_i$$

CPOs.

(Standard-) CPO-Konstruktionen (3)

Summenkonstruktion:

Seien $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ CPOs.
Dann ist auch

- die direkte Summe $(\bigoplus P_i, \sqsubseteq)$ mit...
  - $(\bigoplus P_i, \sqsubseteq) = (P_1 \dot\cup P_2 \dot\cup \ldots \dot\cup P_n, \sqsubseteq)$ disjunkte Vereinigung der P_i, $i \in \{1, \ldots, n\}$ und $\forall\, p, q \in \bigoplus P_i.\ p \sqsubseteq q \succ \exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge\ p \sqsubseteq_i q$ und der Identifikation der kleinsten Elemente der $(P_i, \sqsubseteq_i)$, $i \in \{1, \ldots, n\}$, d.h. $\bot =_{df} \bot_i$, $i \in \{1, \ldots, n\}$

eine CPO.

(Standard-) CPO-Konstruktionen (4)

Funktionenraum:

Seien $(C, \sqsubseteq_C)$ und $(D, \sqsubseteq_D)$ zwei CPOs und $[C \to D] =_{df}$ $\{f : C \to D \mid f \text{ stetig}\}$ die Menge der stetigen Funktionen von C nach D.
Dann ist auch

- der stetige Funktionenraum $([C \to D], \sqsubseteq)$ mit

$$\forall f, g \in [C \to D].\ f \sqsubseteq g \iff \forall c \in C.\ f(c) \sqsubseteq_D g(c)$$

eine CPO.

Monotonie und Stetigkeit

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

Dann heißt f

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

Sei $(C, \sqsubseteq)$ eine CPO, sei $f : C \to C$ eine Funktion auf C.

Dann heißt f

- inflationär (vergrößernd) gdw $\forall\, c \in C.\ c \sqsubseteq f(c)$

Eigenschaften

Mit den vorigen Bezeichnungen gilt:

Lemma (A.4.1)

f ist monoton gdw. $\forall\, C' \subseteq C.\ f(\bigsqcup_C C') \sqsupseteq_D \bigsqcup_D f(C')$

Korollar (A.4.2)

Eine stetige Funktion ist stets monoton,
d.h. f stetig $\succ$ f monoton.

A.5

Fixpunkte und Fixpunkttheoreme

Fixpunkte

Sei $(C, \sqsubseteq)$ eine CPO, $f : C \to C$ eine Funktion auf C und sei c ein Element von C, also $c \in C$.

Dann heißt c

- Fixpunkt von f gdw $f(c) = c$

Ein Fixpunkt c von f heißt

- kleinster Fixpunkt von f gdw $\forall\, d \in C.\ f(d) = d \succ c \sqsubseteq d$
- größter Fixpunkt von f gdw $\forall\, d \in C.\ f(d) = d \succ d \sqsubseteq c$

Bezeichnungen:

- Der kleinste bzw. größte Fixpunkt einer Funktion f wird meist mit μf bzw. νf bezeichnet.

Bedingte Fixpunkte

Seien $d, c_d \in C$. Dann heißt c_d

- bedingter kleinster Fixpunkt von f bezüglich d gdw c_d ist der kleinste Fixpunkt von C mit $d \sqsubseteq c_d$, d.h. für alle anderen Fixpunkte x von f mit $d \sqsubseteq x$ gilt: $c_d \sqsubseteq x$.

Fixpunkttheorem

Theorem (A.5.1, Knaster/Tarski, Kleene)

Sei $(C, \sqsubseteq)$ eine CPO und $f : C \to C$ eine stetige Funktion auf C.

Dann hat f einen kleinsten Fixpunkt μf und dieser Fixpunkt ergibt sich als kleinste obere Schranke der Kette (sog. Kleene-Kette) $\{\bot, f(\bot), f^2(\bot), \ldots\}$, d.h.

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

Beweis von Fixpunkttheorem A.5.1 (1)

Zu zeigen:

μf

1. existiert
2. ist Fixpunkt
3. ist kleinster Fixpunkt

Beweis von Fixpunkttheorem A.5.1 (2)

1. Existenz

- Es gilt $f^0 \perp = \perp$ und $\perp \sqsubseteq c$ für alle $c \in C$.
- Durch vollständige Induktion lässt sich damit zeigen: $f^n \perp \sqsubseteq f^n c$ für alle $c \in C$.
- Somit gilt $f^n \perp \sqsubseteq f^m \perp$ für alle n, m mit $n \leq m$. Somit ist $\{f^n \perp \mid n \geq 0\}$ eine (nichtleere) Kette in C.
- Damit folgt die Existenz von $\bigsqcup_{i \in \mathbb{N}_0} f^i(\perp)$ aus der CPO-Eigenschaft von $(C, \sqsubseteq)$.

Beweis von Fixpunkttheorem A.5.1 (3)

2. Fixpunkteigenschaft

$$
\begin{aligned}
& & f\left(\bigsqcup_{i\in\mathbb{N}_0} f^i(\bot)\right) \\
(f \text{ stetig}) & = & \bigsqcup_{i\in\mathbb{N}_0} f(f^n\bot) \\
& = & \bigsqcup_{i\in\mathbb{N}_1} f^n\bot \\
(K \text{ Kette} \succ \bigsqcup K = \bot \sqcup \bigsqcup K) & = & \bigsqcup_{i\in\mathbb{N}_1} f^n\bot \ \sqcup \ \bot \\
(f^0 = \bot) & = & \bigsqcup_{i\in\mathbb{N}_0} f^n\bot \\
& = & \bigsqcup_{i\in\mathbb{N}_0} f^i(\bot)
\end{aligned}
$$

Beweis von Fixpunkttheorem A.5.1 (4)

3. Kleinster Fixpunkt

- Sei c beliebig gewählter Fixpunkt von f. Dann gilt $\bot \sqsubseteq c$ und somit auch $f^n \bot \sqsubseteq f^n c$ für alle $n \geq 0$.
- Folglich gilt $f^n \bot \sqsubseteq c$ wg. der Wahl von c als Fixpunkt von f.
- Somit gilt auch, dass c eine obere Schranke von $\{f^i(\bot) \mid i \in \mathbb{N}_0\}$ ist.
- Da $\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)$ nach Definition die kleinste obere Schranke dieser Kette ist, gilt wie gewünscht $\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \sqsubseteq c$.

Bedingte Fixpunkte

Theorem (A.5.2, Endliche Fixpunkte)

Sei $(C, \sqsubseteq)$ eine CPO und $f : C \to C$ eine stetige, inflationäre Funktion auf C und sei $d \in C$.

Dann hat f einen kleinsten bedingten Fixpunkt μf_d und dieser Fixpunkt ergibt sich als kleinste obere Schranke der Kette $\{d, f(d), f^2(d), \ldots\}$, d.h.

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

Endliche Fixpunkte

Theorem (A.5.3, Endliche Fixpunkte)

Sei $(C, \sqsubseteq)$ eine CPO und $f : C \to C$ eine stetige Funktion auf C.

Dann gilt: Sind in der Kleene-Kette von f zwei aufeinanderfolgende Glieder gleich, etwa $f^i(\bot) = f^{i+1}(\bot)$, so gilt $\mu f = f^i(\bot)$.

Existenz endlicher Fixpunkte

Hinreichende Bedingungen für die Existenz endlicher Fixpunkte sind:

- Endlichkeit von Definitions- und Wertebereich von f.
- f ist von der Form $f(c) = c \sqcup g(c)$ für monotones g über kettenendlichem Wertebereich.