Analyse und Verifikation

(SS 2010, 185.276, VU 2.0h, ECTS 3.0, MSE/W)

Jens Knoop
Institut für Computersprachen

knoop@complang.tuwien.ac.at
http://www.complang.tuwien.ac.at/knoop/

Dienstag, 13:30 Uhr bis 15:00 Uhr, Bibliothek E185.1,
Argentinierstr. 8, 4. Stock (Mitte), 1040 Wien)

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(2)

Als Textbücher...

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

  (Siehe http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html für eine frei verfügbare (überarbeitete) Version.)

Literaturhinweise 2(2)

Ergänzend und weiterführend...

- Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. 3. Auflage, Springer, 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, 1994.
- 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.

Konkret

- Modellsprache: WHILE
  Syntax und Semantik
- Drei Semantikdefinitionsstile
  1. Operationelle Semantik
     (a) Großschritt-Semantik: Natürliche Semantik
     (b) 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 Analyse, Verifikation und Transformation von Programmen

Unsere Modellsprache

- Die (Programmier-) Sprache WHILE
  - Syntax
  - Semantik
- Semantikdefinitionsstile
  (*...und wofür sie besonders geeignet sind und ihre Beziehungen zueinander*)
  - Operationelle Semantik
    - Natürliche Semantik
    - Strukturell operationelle Semantik
  - Denotationelle Semantik
  - Axiomatische Semantik
    - 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

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

WHILE-Syntax&Semantik im Überblick

- **Syntax**
  ...Programme der Form:

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

- **Semantik**
  ...in Form von *Zustandstransformationen*:

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

  über

  – $\Sigma =_{df} \{\sigma \mid \sigma : \textbf{Var} \to D\}$ 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

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

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)

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

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

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

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

Beachte:

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

Semantik arithmetischer und Wahrheitswertausdrücke

Semantik

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

Dabei bezeichnen

- $\mathbb{Z} =_{df} \{..., \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$ die Menge ganzer Zahlen
- $\mathbb{B} =_{df} \{\mathtt{tt}, \mathtt{ff}\}$ die Menge der Wahrheitswerte

Semantik arithmetischer Ausdrücke

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

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

wobei

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

Semantik Boolescher Ausdrücke (1)

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

- $[\![true]\!]_B(\sigma) =_{df}$ `tt`
- $[\![false]\!]_B(\sigma) =_{df}$ `ff`
- $[\![a_1 = a_2]\!]_B(\sigma) =_{df} \begin{cases} \texttt{tt} & \text{falls } equal([\![a_1]\!]_A(\sigma), [\![a_2]\!]_A(\sigma)) \\ \texttt{ff} & \text{sonst} \end{cases}$
- ... (andere Relatoren (z.B. $<$, $\leq$, ...) analog)
- $[\![\neg b]\!]_B(\sigma) =_{df} neg([\![b]\!]_B(\sigma))$
- $[\![b_1 \wedge b_2]\!]_B(\sigma) =_{df} conj([\![b_1]\!]_B(\sigma), [\![b_2]\!]_B(\sigma))$
- $[\![b_1 \vee b_2]\!]_B(\sigma) =_{df} disj([\![b_1]\!]_B(\sigma), [\![b_2]\!]_B(\sigma))$

Semantik Boolescher Ausdrücke (2)

...wobei

- `tt` und `ff` 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 `tt` und `ff`.

Vereinbarung

In der Folge seien die

- *Semantik arithmetischer Ausdrücke:*

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

- *Semantik Boolescher Ausdrücke:*

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

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

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

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...

- Substitutionen
  - *Syntaktische* Substitution
  - *Semantische* Substitution
  - *Substitutionslemma*

Syntaktische Substitution

Definition 1.4.3

Die *syntaktische Substitution* für arithmetische Terme ist eine dreistellige Abbildung

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

die induktiv definiert ist durch

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

Semantische Substitution

Definition 1.4.4

Die *semantische Substitution* ist eine dreistellige Abbildung

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

die definiert ist durch

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

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)

Zentral:

- Vollständige Induktion
- Verallgemeinerte Induktion
- Strukturelle Induktion

...zum Beweis einer Aussage A.

Induktive Beweisprinzipien (2)

Zur Erinnerung hier wiederholt:

Die Prinzipien der...

- *vollständigen Induktion*

  $(A(1) \wedge (\forall n \in I\!N.\, A(n) \succ A(n+1))) \quad \succ \quad \forall n \in I\!N.\, A(n)$

- *verallgemeinerten Induktion*

  $(\forall n \in I\!N.\, (\forall m < n.\, A(m)) \succ A(n)) \quad \succ \quad \forall n \in I\!N.\, A(n)$

- *strukturellen Induktion*

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

Beachte: $\succ$ bezeichnet die logische Implikation; A erinnert an *Aussage*, S an (induktiv definierte) *Struktur*.

Induktive Beweisprinzipien (3)

Beachte:

- 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 geeignetsten.

Beispiel: Beweis von Lemma 1.4.1 (1)

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

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

Induktionsanfang:

Fall 1: Sei $a \equiv n$, $n \in$ **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 $a \equiv x$, $x \in$ **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:

$$\begin{array}{ll} & [\![a]\!]_A(\sigma) \\ = & [\![a_1 + a_2]\!]_A(\sigma) \\ = & [\![a_1]\!]_A(\sigma) + [\![a_2]\!]_A(\sigma) \\ \text{(Induktionshypothese für } a_1, a_2) & [\![a_1]\!]_A(\sigma') + [\![a_2]\!]_A(\sigma') \\ = & [\![a_1 + a_2]\!]_A(\sigma') \\ = & [\![a]\!]_A(\sigma') \end{array}$$

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

Kapitel 2 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 (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.

In der Folge werden wir das Zustandstransformationsfunktional $[\![\,.\,]\!]_{sos}$ 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
- $\Gamma =_{df} (\mathbf{Prg} \times \Sigma) \cup \Sigma$ bezeichne die Menge aller Konfigurationen, $\gamma \in \Gamma$

SOS-Regeln von WHILE – Axiome (1)

[skip$_{sos}$] $\dfrac{}{\langle skip, \sigma \rangle \Rightarrow \sigma}$

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

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

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

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

SOS-Regeln von WHILE − Regeln (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

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

- Prämissenbehaftete *Regeln* der Form

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

ggf. mit *Randbedingungen* (*Seitenbedingungen*) wie z.B. in Form von $[\![b]\!]_B(\sigma) = \texttt{ff}$ 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

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

- 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 $i \in I\!N$.

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 **0**)

Beispiel (1)

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)

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

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

$$
\Rightarrow \quad \langle \text{while } x <> 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}, (\sigma[\mathbf{3}/y])[\mathbf{2}/x]\rangle
$$

Beispiel (3)

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

Beispiel (4)

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

Beispiel – Detailbetrachtung (5)

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

steht vereinfachend für...

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

Beispiel – Detailbetrachtung (6)

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

steht vereinfachend für...

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

Beispiel – Detailbetrachtung (7)

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

steht vereinfachend für...

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

Determinismus der SOS-Regeln

Lemma 2.1.1

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

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

Korollar 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:

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

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

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

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

Kap. 2.2 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.

In der Folge werden wir das Zustandstransformationsfunktional $[\![\, . \,]\!]_{ns}$ im Detail definieren.

Natürliche Semantik 2(2)

Intuitiv:

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

Zentral auch hier:

- Der von der SO-Semantik bekannte Begriff der *Konfigurationen*.

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)

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

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

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

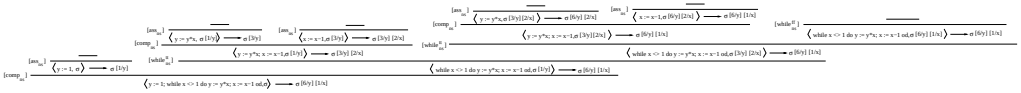
$$[\text{while}^{ff}_{ns}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma\rangle \to \sigma} \qquad [\![b]\!]_B(\sigma) = \mathtt{ff}$$

Beispiel (1)

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

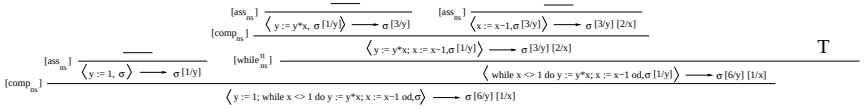
Dann gilt:

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

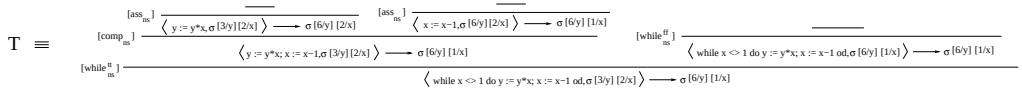


Beispiel (2)

Das gleiche Beispiel in etwas gefälligerer Darstellung:



T ≡



Determinismus der NS-Regeln

Lemma 2.2.1

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

Korollar 2.2.2
Die von den NS-Regeln für eine Konfiguration induzierte finale Konfiguration ist (sofern definiert) eindeutig bestimmt, d.h. *deterministisch*.

Salopper, wenn auch weniger präzise:
Die N-Semantik von WHILE ist deterministisch!

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

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:

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

Variante induktiver Beweisführung

Induktion über die Form von Ableitungsbäumen:

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

$\leadsto$ 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**!

Kap. 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' beschreibt den *ersten* Schritt der Berechnungsfolge von π angesetzt auf σ.
  Folgende Übergänge sind 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

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

  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.

Kap. 3 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

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

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

In der Folge werden wir das Zustandstransformationsfunktional $[\![\ ]\!]_{ds}$ 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.a. 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.

Denotationelle Semantik – Definierende Gleichungen

...der Modellsprache WHILE:

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

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

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

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

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

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

wobei

$Id : \Sigma \to \Sigma$ die identische Zustandstransformation bezeichnet:

$$\forall \sigma \in \Sigma.\ Id(\sigma) =_{df} \sigma$$

Zur Hilfsfunktion cond

Hilfsfunktion $cond$:

Funktionalität

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

Definiert durch

$$cond(p, g_1, g_2)\ \sigma =_{df} \begin{cases} g_1\ \sigma & \text{falls } p\ \sigma = \mathtt{tt} \\ g_2\ \sigma & \text{falls } p\ \sigma = \mathtt{ff} \\ undef & \text{sonst} \end{cases}$$

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

Somit ergibt sich

...für die Bedeutung der Fallunterscheidung:

$$
\begin{aligned}
&[\![\textbf{if } b \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi}]\!]_{ds}\ \sigma \\
&\quad = \quad cond([\![b]\!]_{\mathcal{B}}, [\![\pi_1]\!]_{ds}, [\![\pi_2]\!]_{ds})\ \sigma \\
&\quad = \quad \begin{cases}
\sigma' & \text{falls } ([\![b]\!]_{\mathcal{B}}\ \sigma = \mathtt{tt} \ \wedge\ [\![\pi_1]\!]_{\mathtt{ds}}\ \sigma = \sigma') \\
 & \quad \vee\ ([\![b]\!]_{\mathcal{B}}\ \sigma = \mathtt{ff} \ \wedge\ [\![\pi_2]\!]_{\mathtt{ds}}\ \sigma = \sigma') \\
undef & \text{falls } ([\![b]\!]_{\mathcal{B}}\ \sigma = undef) \\
 & \quad \vee\ ([\![b]\!]_{\mathcal{B}}\ \sigma = \mathtt{tt} \ \wedge\ [\![\pi_1]\!]_{\mathtt{ds}}\ \sigma = undef) \\
 & \quad \vee\ ([\![b]\!]_{\mathcal{B}}\ \sigma = \mathtt{ff} \ \wedge\ [\![\pi_2]\!]_{\mathtt{ds}}\ \sigma = undef)
\end{cases}
\end{aligned}
$$

Erinnerung:

- In unserem Szenario sind $[\![.]\!]_{\mathcal{A}}$ und $[\![.]\!]_{\mathcal{B}}$ partiell definiert (z.B. Division durch 0).

Zum Hilfsfunktional FIX

Hilfsfunktional FIX mit Funktionalität:

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

angewendet auf

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

definiert durch

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

bezüglich 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 (S; 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:

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

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

Hauptergebnisse / Äquivalenzresultate

Theorem

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

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:

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

definiert (z.B.) durch

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

Kap. 3.1 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:

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

angewendet auf

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

definiert durch

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

bezüglich 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 $[\![\text{while } b \text{ do } \pi \ od]\!]_{ds}$ und damit insgesamt von $[\![\ ]\!]_{ds}$.

Unser Arbeitsplan

Wir benötigen:

- Einige Resultate aus der *Fixpunkttheorie*.

Diese erlauben uns:

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

Dabei wird vorausgesetzt:

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

Dieser Hintergrund wird in Kapitel 3.2 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 dann 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 $[\Sigma \hookrightarrow \Sigma]$:

$$g_1 \sqsubseteq g_2 \iff \forall \sigma \in \Sigma.\ g_1\ \sigma\ definiert\ = \sigma' \Rightarrow g_2\ \sigma\ definiert\ = \sigma'$$
$$\text{mit } g_1, g_2 \in [\Sigma \hookrightarrow \Sigma]$$

Lemma 3.1.1

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

Ordnung auf Zustandstransformationen

Es gilt sogar:

Lemma 3.1.2
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:

- $\langle m, n\rangle \in graph(f)\ \wedge\ \langle m, n'\rangle \in graph(f) \Rightarrow n = n'$ (*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 3.1.3
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 3.1.4
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 3.1.5**
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:

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

Kap. 3.2 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. $\forall m,n,p \in M.\ m\,R\,n \ \wedge\ n\,R\,p \ \Rightarrow\ m\,R\,p$
- *antisymmetrisch* gdw. $\forall m,n \in M.\ m\,R\,n \wedge n\,R\,m \ \Rightarrow\ m = n$

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

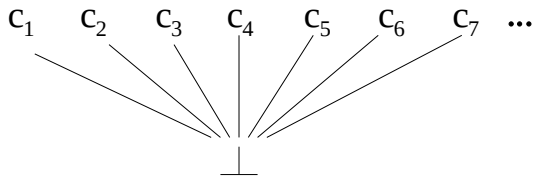
- *endlich* gdw. die Menge $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$ keine unendliche Kette enthält
- *endlich relativ zu* $r \in P$ gdw. die Menge $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$ keine unendliche Kette enthält

(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)

Produktkonstruktionen...

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 $\forall (p_1, p_2, \ldots, p_n), (q_1, q_2, \ldots, q_n) \in \times P_i.\ (p_1, p_2, \ldots, p_n) \sqsubseteq (q_1, q_2, \ldots, q_n) \Rightarrow \forall i \in \{1, \ldots, n\}.\ p_i \sqsubseteq_i q_i$
- und das *strikte* (*direkte*) *Produkt* (*smash Produkt*) mit
  - $(\otimes P_i, \sqsubseteq) = (P_1 \otimes P_2 \otimes \ldots \otimes P_n, \sqsubseteq)$, 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(4)

Summenkonstruktion...

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

- die *direkte Summe* $(\oplus P_i, \sqsubseteq)$ mit...
  - $(\oplus P_i, \sqsubseteq) = (P_1 \dot\cup P_2 \ \dot\cup \ \ldots \ \dot\cup \ P_n, \sqsubseteq)$ disjunkte Vereinigung der P_i, $i \in \{1, \ldots, n\}$ und $\forall p, q \in \oplus P_i.\ p \sqsubseteq q \Rightarrow \exists i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge \ p \sqsubseteq_i q$ 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(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
  - $\forall f, g \in [C \to D].\ f \sqsubseteq g \iff \forall c \in C.\ f(c) \sqsubseteq_D g(c)$

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. $\forall c, c' \in C.\ c \sqsubseteq_C c' \Rightarrow f(c) \sqsubseteq_D f(c')$ (*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*)

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

Korollar
Eine stetige Funktion ist stets monoton, d.h. f stetig $\Rightarrow$ 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. $\forall d \in C.\ f(d) = d \Rightarrow c \sqsubseteq d$
- *größter Fixpunkt* von f gdw. $\forall d \in C.\ f(d) = d \Rightarrow d \sqsubseteq c$

(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 3.2.1 (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 3.2.1 1(4)

Zu zeigen: μf:

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

Beweis des Fixpunktsatzes 3.2.1 2(4)

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

Beweis des Fixpunktsatzes 3.2.1 3(4)

2. *Fixpunkteigenschaft*

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

Beweis des Fixpunktsatzes 3.2.1 4(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 3.2.2 (Endliche 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 3.3.3 (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

Kapitel 4 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.

Intuitiv

"Totale Korrektheit = Partielle Korrektheit + Terminierung"

Partielle und totale Korrektheit

- Die Zustandsmenge

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

heißt *Charakterisierung von* $p \in$ **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.

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

Erinnerung

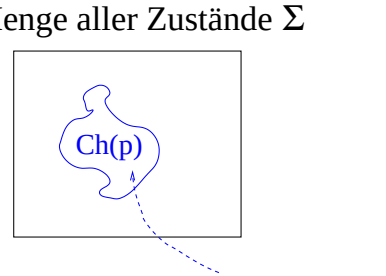
...an einige Sprechweisen:

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 **0**).
- Ein Programm π heißt *divergent* gdw. π terminiert für keinen Anfangszustand regulär.

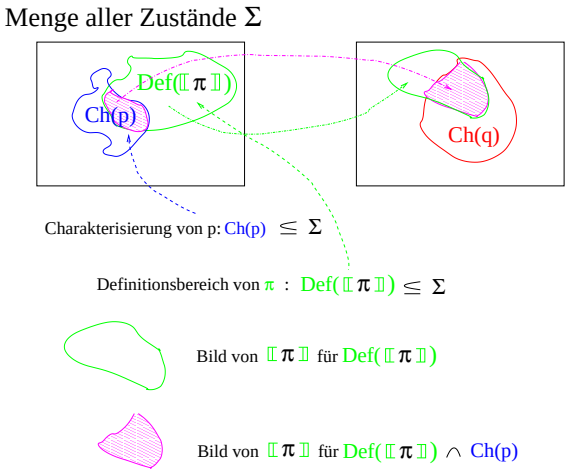
Veranschaulichung (1)

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



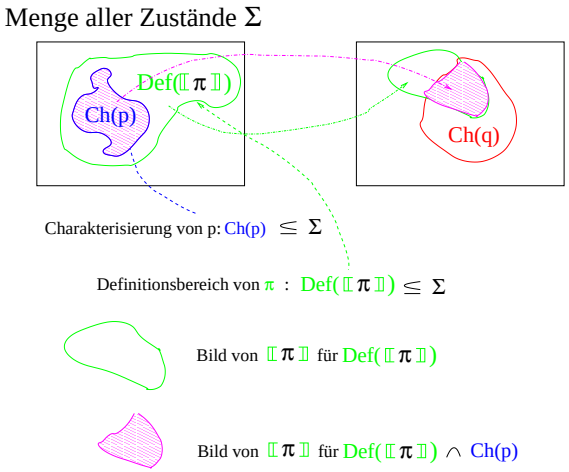
Veranschaulichung (2)

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



Veranschaulichung (3)

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



Stärkste Nachbedingungen, schwächste Vorbedingungen

In der Folge:

Präzisierung 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:

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

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

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

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

Schwächste Vorbedingungen

Definition

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)

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: $A \Rightarrow B$
- A_i heißt *stärkste* Formel in $\{A_1, A_2, \ldots\}$, wenn gilt: $A_i \Rightarrow A_j$ 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)

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

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

  partiell korrekt ist und $slpo(p, \pi)$ die stärkste Formel mit dieser Eigenschaft ist.

Stärkste Nachbedingungen (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] $\dfrac{p \Rightarrow p_1,\ \{p_1\}\ \pi\ \{q_1\},\ q_1 \Rightarrow q}{\{p\}\ \pi\ \{q\}}$

Diskussion von Vorwärtszuweisungsregel(n)

- Eine *Vorwärtsregel* für die Zuweisung wie

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

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

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

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

  Beweis: Übungsaufgabe

Kapitel 4.3 Beweiskalkül für totale Korrektheit

Hoare-Kalkül HK_{TK} für totale Korrektheit

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

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

wobei

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

  $\leadsto$ *Terminationsordnung!*

Zur Vollständigkeit

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

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

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

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

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

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

Bemerkung

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

Wohlfundierte oder Noethersche Ordnungen (1)

Definition

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 oder Noethersche Ordnungen (2)

Definition

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 oder Noethersche Ordnungen (3)

Konstruktionsprinzipien für wohlfundierte Ordnungen aus gegebenen wohlfundierten Ordnungen...

Lemma 4.3.1

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

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

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

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

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

wohlfundierte Ordnungen.

Anmerkungen zu...

...den der

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

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

Zur Konsequenzregel (1)

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

Intuitiv:

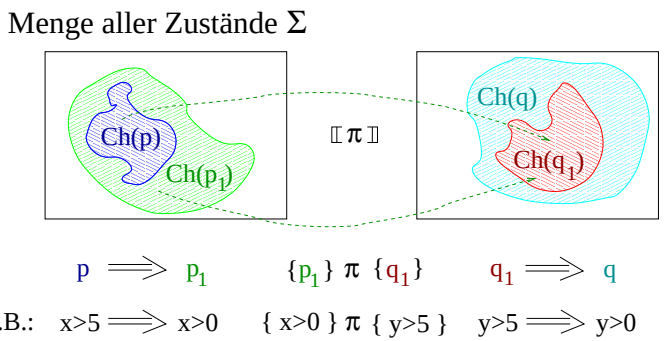
Die Konsequenzregel

- stellt die Schnittstelle zwischen Programmverifikation und den logischen Formeln der Zusicherungssprache dar
- erlaubt es,
  - Vorbedingungen zu *verstärken*
    (Übergang von p_1 zu p möglich, falls $p \Rightarrow p_1$ ($\Leftrightarrow\ Ch(p) \subseteq\ Ch(p_1)$)
  - Nachbedingungen *abzuschwächen*
    (Übergang von q_1 zu q möglich, falls $q_1 \Rightarrow q$ ($\Leftrightarrow\ Ch(q_1) \subseteq\ Ch(q)$)

  um so die Anwendung anderer Beweisregeln zu ermöglichen.

Zur Konsequenzregel (2)

Veranschaulichung von Verstärkung und Abschwächung:



Zur while-Regel in HK_{PK}

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

Intuitiv:

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

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

Erinnerung:

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

wobei

- u Boolescher Ausdruck über der Variablen v,
- t arithmetischer Term,
- w Variable, die in I, b, π und t nicht frei vorkommt,
- $M =_{df} \{\sigma(v) \mid \sigma \in \Sigma \wedge [\![u]\!]_B(\sigma) = \mathtt{tt}\}$ noethersch geordnete Menge (sog. noethersche Halbordnung).
  $\leadsto$ *Terminationsordnung*!

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

- Prämisse 1: $I \wedge b \Rightarrow u[t/v]$
  Wann immer der Schleifenrumpf noch einmal ausgeführt wird (d.h. $I \wedge b$ ist wahr), gilt, dass $u[t/v]$ wahr ist, woraus aufgrund der Definition von M folgt, dass der Wert von t Element einer noethersch geordneten Menge ist.
- 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.
- 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*
  ...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.
- $\{x = n\}\ y := 1;$ while $x \neq 1$ do $y := y{*}x; x := x{-}1$ od $\{y = x! \wedge x > 0\}$
  ...die Nachbedingung macht eine Aussage über den Zusammenhang der schließlichen Werte von x und y. (*Beachte*: nur mit Programmvariablen keine Aussage über die Fakultätsberechnung in diesem Bsp.!)

HK_{TK} versus HK_{PK}

Beachte:

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

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

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.

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

wobei

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

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:

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

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

- dem Nachweis ihrer partiellen Korrektheit
- dem Nachweis der Termination

Diese Trennung kann im Beweis explizit vollzogen werden. 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 4.5 Ergänzungen, Sprechweisen

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

Kapitel 4.3 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*?

Hauptresultate

Zur Korrektheit:

Theorem [Korrektheit von HK_{PK} und HK_{TK}]

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

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

Zur Vollständigkeit:
Für Korrektheitskalküle ist i.a. nur sog. *relative* Vollständigkeit möglich. Das gilt auch für HK_{PK} und HK_{TK}. Details dazu in der Folge.

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:

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

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

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

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

Theorem [Korrektheit von HK_{PK} und HK_{TK}]

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

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

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

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

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

Zur Vollständigkeit Hoarescher Beweiskalküle

Generell müssen wir unterscheiden zwischen Vollständigkeit

- *extensionaler* und
- *intensionaler*

Ansätze.

Extensionale vs. intensionale Ansätze

- *Extensional*

  $\leadsto$ Vor- und Nachbedingungen sind durch *Prädikate* beschrieben.

- *Intensional*

  $\leadsto$ Vor- und Nachbedingungen sind durch *Formeln einer Zusicherungssprache* beschrieben.

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

Für den extensionalen Ansatz gilt:

Theorem [Vollständigkeit von HK_{PK} und HK_{TK}]

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

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

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

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

Beweis ...durch strukturelle Induktion über den Aufbau von π.

Zur Vollständigkeit von HK_{PK} & 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 4.4 Beweis partieller Korrektheit: Zwei Beispiele

Die beiden Beispiele im Überblick 1(2)

...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(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

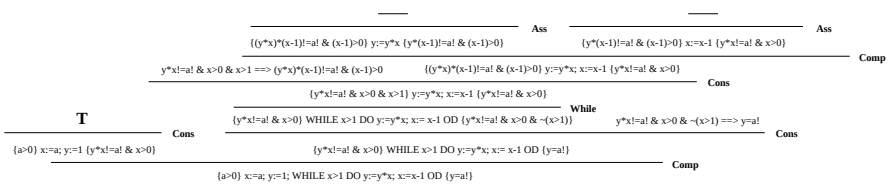
$$\{x \geq 0 \wedge y > 0\}$$
$$q := 0; r := x; \text{while } r \geq y \text{ do } q := q + 1; r := r - y \text{ od}$$
$$\{x = q * y + r \wedge 0 \leq r < y\}$$

gültig sind im Sinne partieller Korrektheit.

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

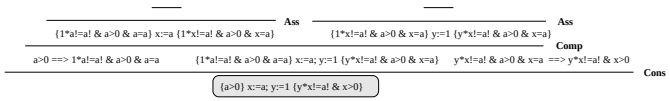
Bew. part. Korrektheit: Fakultät (1)

Erster Beweis



wobei

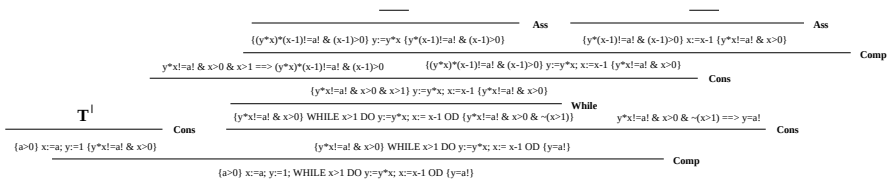
$\mathbf{T} \equiv$



& : Logisches und
~ : Logisches nicht

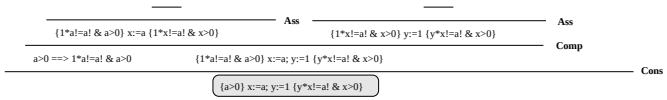
Bew. part. Korrektheit: Fakultät (2)

Zweiter Beweis



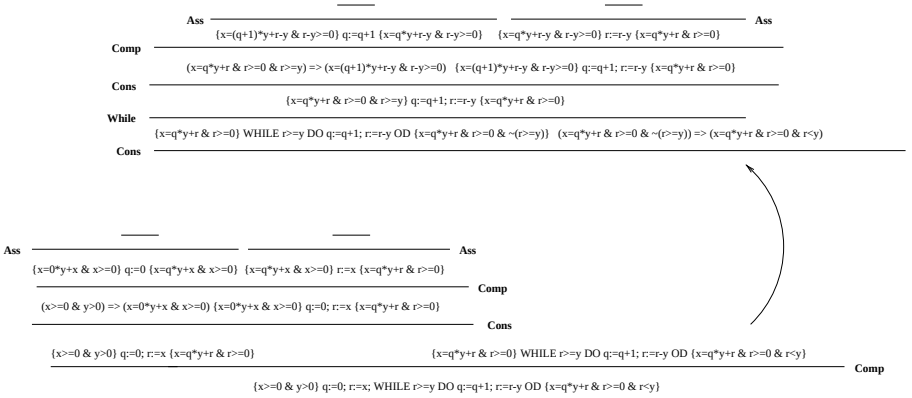
wobei

$\mathbf{T}' \equiv$



& : 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 *Rückwärts*zuweisungsregel 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:

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

Lin. Beweisskizze f. Fakultätsbsp. (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:

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

Überblick (16)

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

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 4.7 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: $I \wedge b \Rightarrow x \in M$ 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:

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

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

$$\begin{array}{c} y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ y := y * x; \\ x := x - 1; \\ [y * x! = a! \wedge x > 0 \wedge x < w] \end{array}$$

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

$$\begin{array}{c} [y * x! = a! \wedge x > 0] \\ \text{while } x > 1 \text{ do} \\ y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ y := y * x; \\ x := x - 1; \\ [y * x! = a! \wedge x > 0 \wedge x < w] \\ \text{od [while]} \\ [y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \end{array}$$

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 *Rückwärts*zuweisungsregel 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:

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

Bew. totaler Korrektheit: Fakultät (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:

$$
\begin{array}{c}
[a > 0] \\
\\
x := a; \\
y := 1; \\
[y * x! = a! \wedge x > 0] \\
\text{while } x > 1 \text{ do} \\
y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\
[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\
\Downarrow \text{ [cons]} \\
[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\
y := y * x; \text{ [ass]} \\
[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\
x := x - 1; \text{ [ass]} \\
[y * x! = a! \wedge x > 0 \wedge x < w] \\
\text{od [while]} \\
[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\
\Downarrow \text{ [cons]} \\
[y = a!]
\end{array}
$$

Bew. totaler Korrektheit: Fakultät (11)

Einmalige Anwendung der [ass]-Regel liefert:

$$
\begin{array}{c}
[a > 0] \\
\\
x := a; \\
[1 * x! = a! \wedge x > 0] \\
y := 1; \text{ [ass]} \\
[y * x! = a! \wedge x > 0] \\
\text{while } x > 1 \text{ do} \\
y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\
[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\
\Downarrow \text{ [cons]} \\
[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\
y := y * x; \text{ [ass]} \\
[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\
x := x - 1; \text{ [ass]} \\
[y * x! = a! \wedge x > 0 \wedge x < w] \\
\text{od [while]} \\
[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\
\Downarrow \text{ [cons]} \\
[y = a!]
\end{array}
$$

Bew. totaler Korrektheit: Fakultät (12)

Abermalige Anwendung der [ass]-Regel liefert:

$$
\begin{array}{c}
[a > 0] \\
\\
[1 * a! = a! \wedge a > 0] \\
x := a; \text{ [ass]} \\
[1 * x! = a! \wedge x > 0] \\
y := 1; \text{ [ass]} \\
[y * x! = a! \wedge x > 0] \\
\text{while } x > 1 \text{ do} \\
y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\
[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\
\Downarrow \text{ [cons]} \\
[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\
y := y * x; \text{ [ass]} \\
[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\
x := x - 1; \text{ [ass]} \\
[y * x! = a! \wedge x > 0 \wedge x < w] \\
\text{od [while]} \\
[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\
\Downarrow \text{ [cons]} \\
[y = a!]
\end{array}
$$

Bew. totaler Korrektheit: Fakultät (13)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

$$
\begin{array}{c}
[a > 0] \\
\Downarrow \text{ [cons]} \\
[1 * a! = a! \wedge a > 0] \\
x := a; \text{ [ass]} \\
[1 * x! = a! \wedge x > 0] \\
y := 1; \text{ [ass]} \\
[y * x! = a! \wedge x > 0] \\
\text{while } x > 1 \text{ do} \\
y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\
[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\
\Downarrow \text{ [cons]} \\
[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\
y := y * x; \text{ [ass]} \\
[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\
x := x - 1; \text{ [ass]} \\
[y * x! = a! \wedge x > 0 \wedge x < w] \\
\text{od [while]} \\
[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\
\Downarrow \text{ [cons]} \\
[y = a!]
\end{array}
$$

Überblick (14)

$[a > 0]$
$\Downarrow$ [cons]
$[1 * a! = a! \wedge a > 0]$
$x := a;$ [ass]
$[1 * x! = a! \wedge x > 0]$
$y := 1;$ [ass]
$[y * x! = a! \wedge x > 0]$
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 (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 4.8 Ausblick

Automatische Ansätze zur Programmverifikation (1)

...*Theorema*-Projekt am RISC, Linz: `http://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.

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`)

Automatische Ansätze zur Programmverifikation (2)

Einige Artikel zu Programmverifikation mit *Theorema*:

- 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`

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)

...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)

- *Fallunterscheidung*
  ...Ausführungszeit entspricht, bis auf einen konstanten Faktor, der gößeren 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 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

$[\![\, . \,]\!]_{TA}$: **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

- $[\![\, . \,]\!]_{TA}$ und $[\![\, . \,]\!]_{TB}$

...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

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

- $[\![\, true \,]\!]_{TB} =_{df} \mathbf{1}$
- $[\![\, false \,]\!]_{TB} =_{df} \mathbf{1}$
- $[\![\, a_1 = a_2 \,]\!]_{TB} =_{df} [\![\, a_1 \,]\!]_{TA} + [\![\, a_2 \,]\!]_{TA} + \mathbf{1}$
- $[\![\, a_1 < a_2 \,]\!]_{TB} =_{df} [\![\, a_1 \,]\!]_{TA} + [\![\, a_2 \,]\!]_{TA} + \mathbf{1}$
- ... (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:

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

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

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

Natürliche Semantik erweitert um den Ausführungszeitaspekt (2)

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

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

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

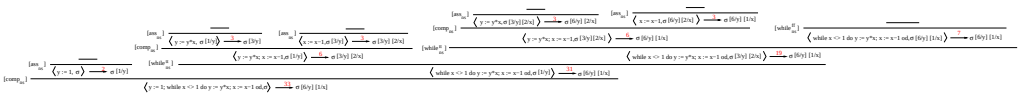
$$[\text{while}^{ff}_{tns}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow^{[\![b]\!]_{TB}+3} \sigma} \qquad [\![b]\!]_B(\sigma) = \texttt{ff}$$

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

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

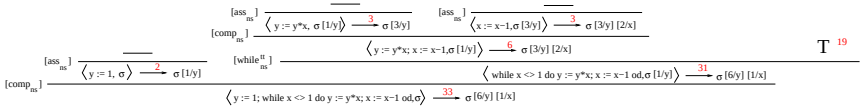
Dann gilt:

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

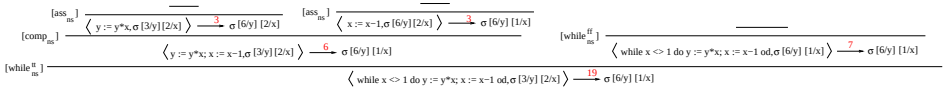


Beispiel zur nat. “Zeit”-Semantik (2)

Das gleiche Beispiel in etwas gefälligerer Darstellung:



T ≡



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
- $e \in$ **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 $\mathcal{O}(e)$.

Axiomatische Semantik zum Ausführungszeitaspekt (1)

[skip$_e$] $\dfrac{}{\{p\}\ skip\ \{1 \Downarrow p\}}$

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

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

wobei u frische logische Variable ist

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

[cons$_e$] $\dfrac{\{p'\}\ \pi\ \{e' \Downarrow q'\}}{\{p\}\ \pi\ \{e \Downarrow q\}}$

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

Axiomatische Semantik zum Ausführungszeitaspekt (2)

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

wobei $p(z+1) \Rightarrow b \wedge e \geq e_1 + e',\ p(0) \Rightarrow \neg\, b \wedge 1 \leq e$
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; while x/=1 do y:=y*x; x:=x-1 od {1 \/ 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; while x/=1 do y:=y*x; x:=x-1 od {x \/ 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.

Kapitel 6 Programmanalyse

Programmanalyse

...speziell *Datenflussanalyse*

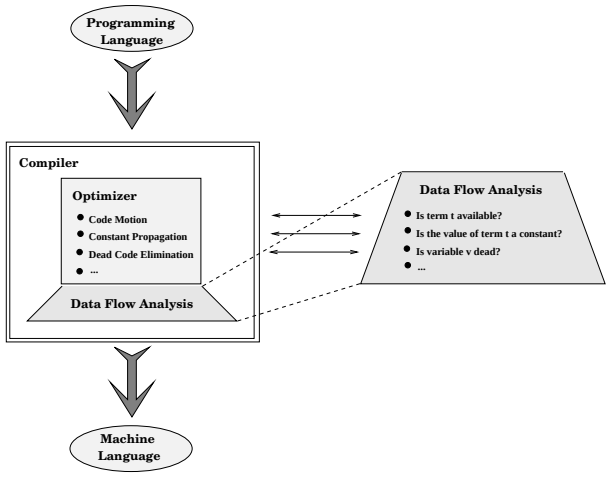
Typische Fragen sind...

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

Kapitel 6.1 Hintergrund und Motivation

Hintergrund und Motivation

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



In der Folge

Zentrale Fragen...

Grundlegendes ebenso...

- 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

Ingredienzien (*intraprozeduraler*) Datenflussanalyse:

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

Theorie intraprozeduraler DFA

Hauptresultate:

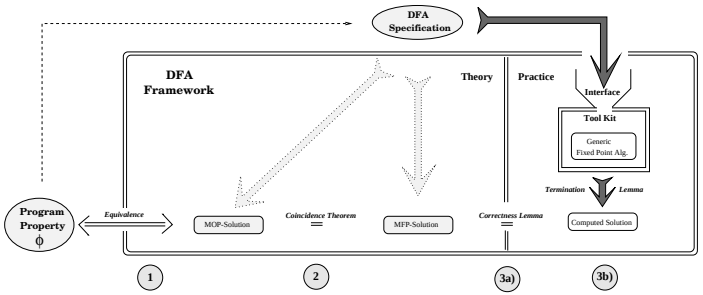
- *Sicherheits-* (*Korrektheits-*) Theorem
- *Koinzidenz-* (*Vollständigkeits-*) Theorem

Sowie:

- *Effektivitäts-* (*Terminierungs-*) Theorem

Praxis intraprozeduraler DFA

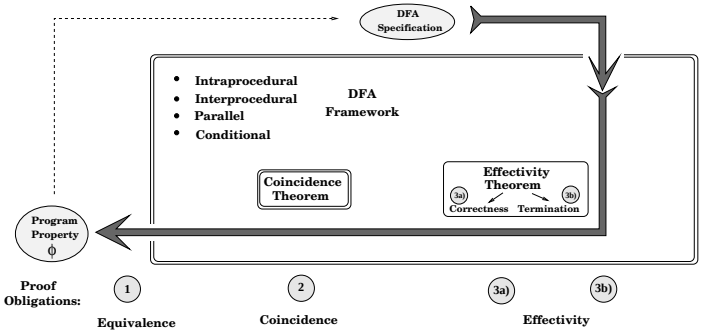
Die intraprozedurale DFA-Framework / DFA-Toolkit - Sicht:



Praxis DFA

Die "intraprozedurale" Einschränkung kann fallen...

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



Ziel

Optimale Programmoptimierung...

...*weiße 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

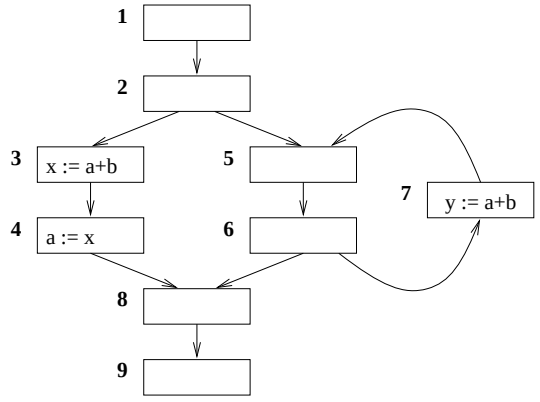
- Knotenmenge (engl. *Nodes*) N
- Kantenmenge (engl. *Edges*) $E \subseteq N \times N$
- ausgezeichnetem Startknoten s ohne Vorgänger und
- ausgezeichnetem Endknoten e ohne Nachfolger

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

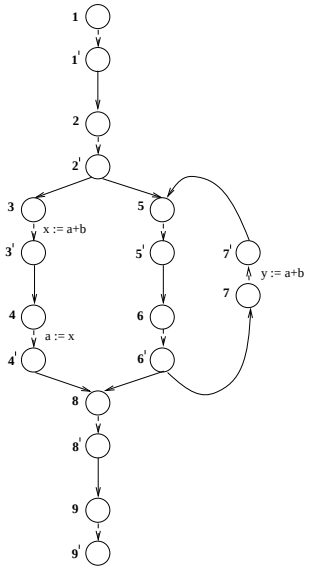
- Knoten ($\leadsto$ *knotenbenannter* Flussgraph)
- Kanten ($\leadsto$ *kantenbenannter* Flussgraph)

repräsentiert werden.

Bsp.: Knotenbenannter Flussgraph

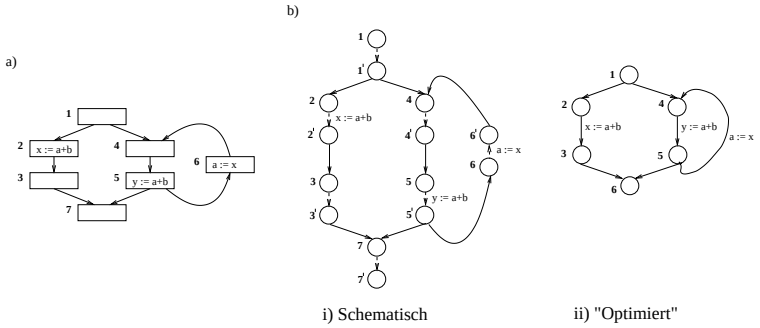


Bsp.: Kantenbenannter Flussgraph



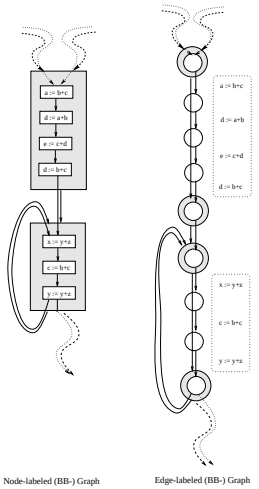
Flussgraphdarstellungsvarianten I

Knoten- vs. kantenbenannte Flussgraphen
(hier mit Einzelanweisungsbenennung)



Flussgraphdarstellungsvarianten II

Knoten- vs. kantenbenannte Flussgraphen
(hier mit Basisblockbenennung)



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: $c_{\mathbf{s}} \in \mathcal{C}$

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
   (a) Algorithmen zur Berechnung von Schnitt, Gleichheitstest und Anwendung von Funktionen auf Verbandselemente eines monotonen Datenflussanalyserahmens
   (b) 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

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

Die $MaxFP$ -Lösung

$\forall\, c_{\mathrm{s}} \in \mathcal{C}\ \forall\, n \in N.\ MaxFP_{([\![\ ]\!],c_{\mathrm{s}})}(n) =_{df} \textbf{inf}^{\,*}_{c_{\mathrm{s}}}(n)$

wobei $\textbf{inf}^{\,*}_{c_{\mathrm{s}}}$ die größte Lösung des $MaxFP$ -Gleichungssystems bezüglich $[\![\ \ ]\!]$ und c_{s} bezeichnet.

Die $MaxFP$-Lösung: die (unter geeigneten Voraussetzungen) *berechenbare* Lösung eines durch $\mathcal{C}$, $[\![\ \ ]\!]$ und c_{s} gegebenen intraprozeduralen DFA-Problems.

Generischer Fixpunktalgorithmus 1(2)

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

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_{s}.

(2) $[\![\ \ ]\!]$ ist *monoton*: Variable *inf* enthält für jeden Knoten eine sichere (d.h. untere) Approximation der stärksten Nachbedingung bezüglich der Anfangsinformation c_{s}.

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

Generischer Fixpunktalgorithmus 2(2)

(Prolog: Initialisierung von *inf* and *workset*)
FORALL $n \in N \backslash \{\mathrm{s}\}$ DO *inf*[n] := $\top$ OD;
inf[s] := c_{s};
workset := { 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 $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ 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 $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ 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 $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ 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

Sicherheitstheorem [Safety]

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

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

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

Vollständigkeit (und Korrektheit)

Koinzidenztheorem [Coincidence]

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

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

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

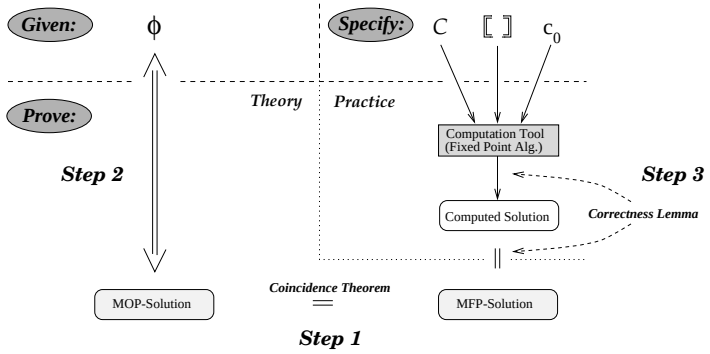
Effektivität/Terminierung

Effektivitätstheorem

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

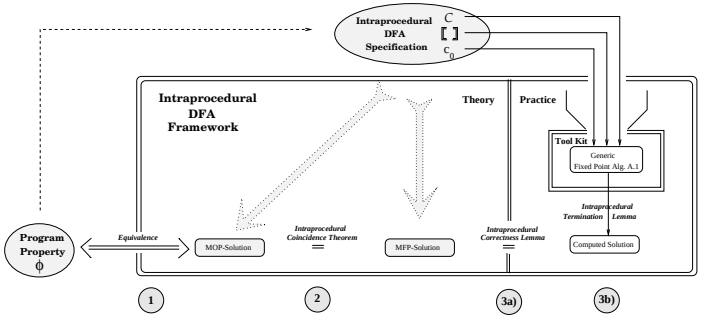
Intraprozedurale DFA im Überblick (1)

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**
  $\leadsto$ kanonisches Bsp. eines *distributiven* DFA-Problems
- **Einfache Konstanten**
  $\leadsto$ kanonisches Bsp. eines *monotonen* DFA-Problems

Verfügbare Ausdrücke

...ein typisches distributives DFA-Problem.

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

$$\forall\, e \in E.\ [\![\, e \,]\!]_{av} =_{df} \begin{cases} Cst_{\mathtt{tt}} & \text{falls } \mathit{Comp}_e \wedge \mathit{Transp}_e \\ \mathit{Id}_{\mathbb{B}} & \text{falls } \neg \mathit{Comp}_e \wedge \mathit{Transp}_e \\ Cst_{\mathtt{ff}} & \text{sonst} \end{cases}$$

Verfügbare Ausdrücke

Dabei bezeichnen:

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

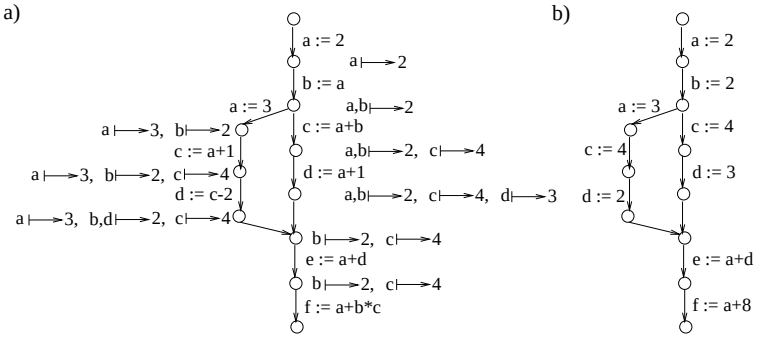
Verfügbare Ausdrücke

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

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

Einfache Konstanten

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



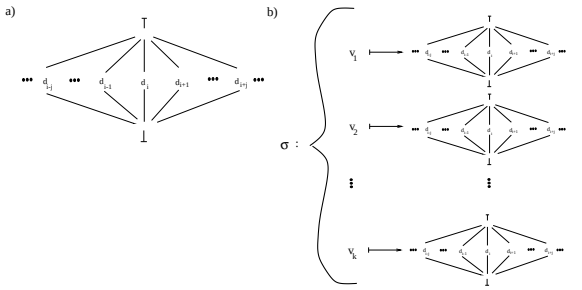
Abstrakte Semantik für einfache Konstanten

- **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:

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

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

Zu ergänzende Begriffe & Definitionen

...um die Definition der Termsemantik abzuschließen:

- Termsyntax
- Interpretation
- Zustand

Die Syntax von Termen (1)

Sei

- $\mathbf{V}$ eine Menge von Variablen und
- $\mathbf{Op}$ 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 $\mathbf{Op}$.

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

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

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

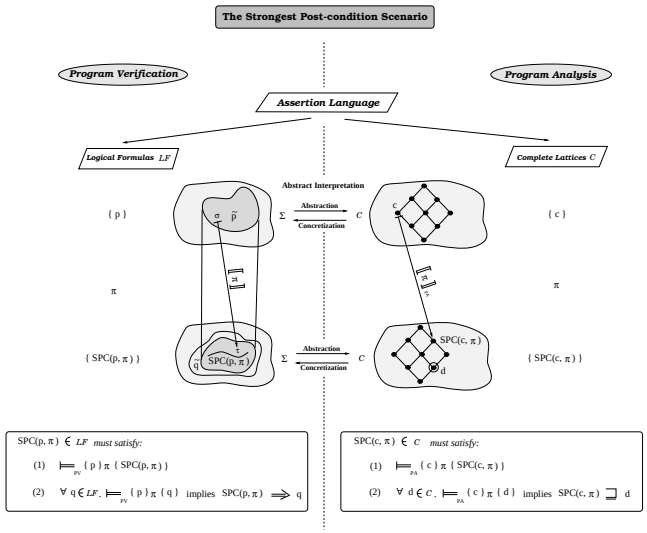
Lemma $[\![\ ]\!]_{sc}$ ist monoton.

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

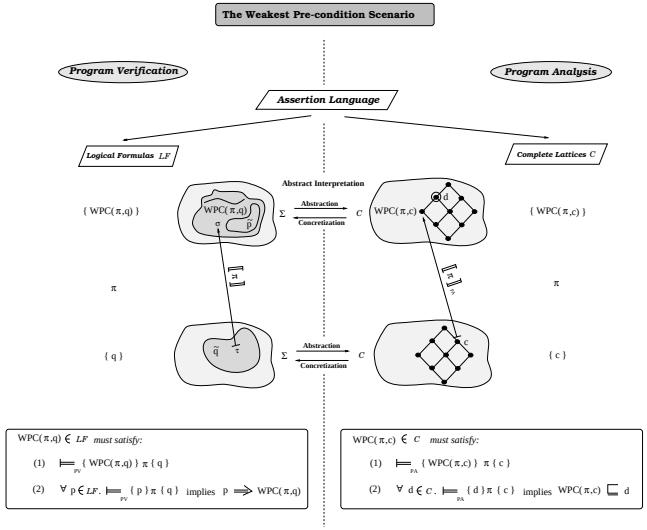
Korollar Für einfache Konstanten stimmen MOP- und $MaxFP$ -Lösung i.a. nicht überein. Die $MaxFP$ -Lösung ist aber stets eine sichere Approximation der MOP-Lösung für einfache Konstanten.

Kapitel 7 Programmverifikation vs. Programmanalyse: Ein Vergleich

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* $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$
2. *Reverses Datenflussanalysefunktional* $[\![\ ]\!]_R : E \to (\mathcal{C} \to \mathcal{C})$ definiert durch

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

Zusammenhang von $[\![\ ]\!]$ und $[\![\ ]\!]_R$ (1)

Lemma

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

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

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

Sprechweise in der Theorie "Abstrakter Interpretation":

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

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

Hilfssatz

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 **q** ("query node") eine Kopie von q ist mit $pred(\mathbf{q}) = pred(q)$ und $succ(\mathbf{q}) = \emptyset$.

Kapitel 8.1.1 R-JOP-Ansatz

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 8.1.2 R-$MinFP$-Ansatz

Der R-$MinFP$-Ansatz

Das R-$MinFP$-Gleichungssystem:

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

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

Die R-$MinFP$-Lösung:

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

Der generische R-$MinFP$-Alg. (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 xx), 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*[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.

(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 *reqInf*[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. (2)

(Prolog: Initialisierung von *reqInf* und *workset*)

FORALL $n \in N \backslash \{\mathbf{q}\}$ DO *reqInf*[n] := $\bot$ OD;
reqInf[$\mathbf{q}$] := c_q;
workset := $\{\mathbf{q}\}$;

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

(Hauptprozess: Iterative Fixpunktberechnung)

```
WHILE workset ≠ ∅ DO
  CHOOSE m ∈ workset;
    workset := workset\{m};
    ( Update the predecessor-environment of node 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 8.1.3 Reverses Koinzidenz- und Sicherheitstheorem

Reverses Sicherheitstheorem

Reverses Sicherheitstheorem

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

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

Reverses Koinzidenztheorem

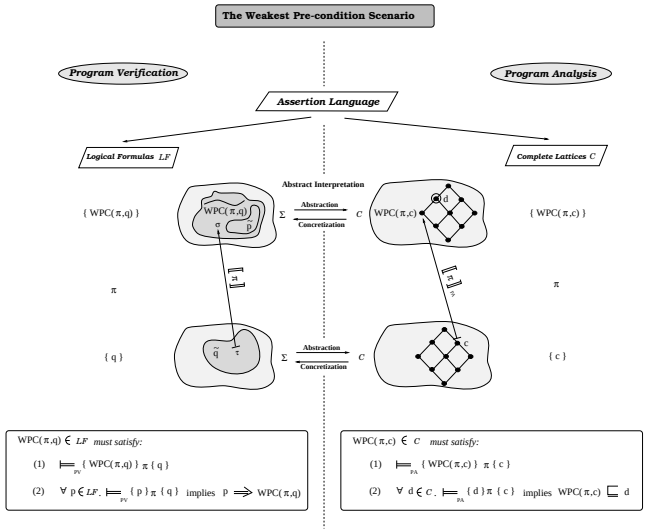
Reverses Koinzidenztheorem

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

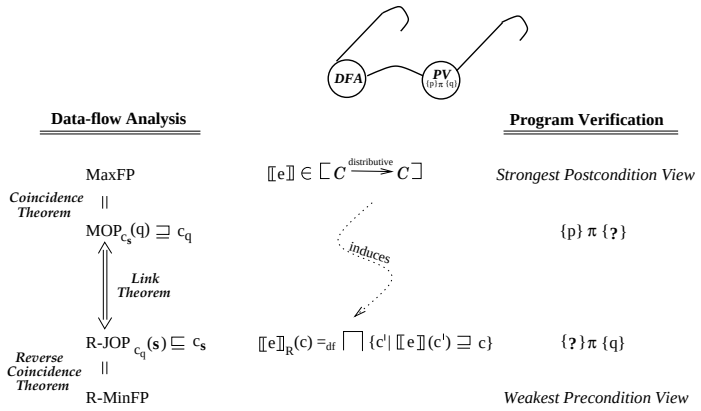
$$\forall\, c_q \in \mathcal{C}\ \forall\, n \in N.\ R\text{-}MinFP_{c_q}(n) = R\text{-}JOP_{c_q}(n)$$

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

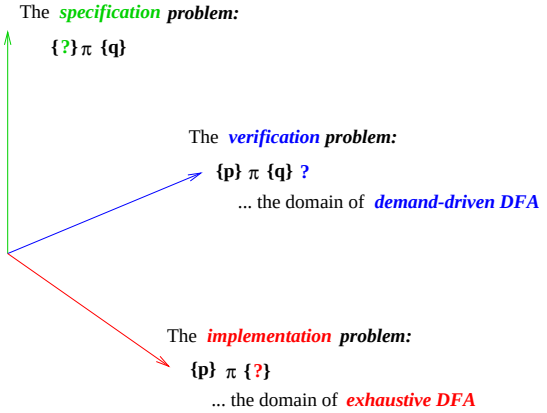
Somit die gewünschte WPC-Analogie



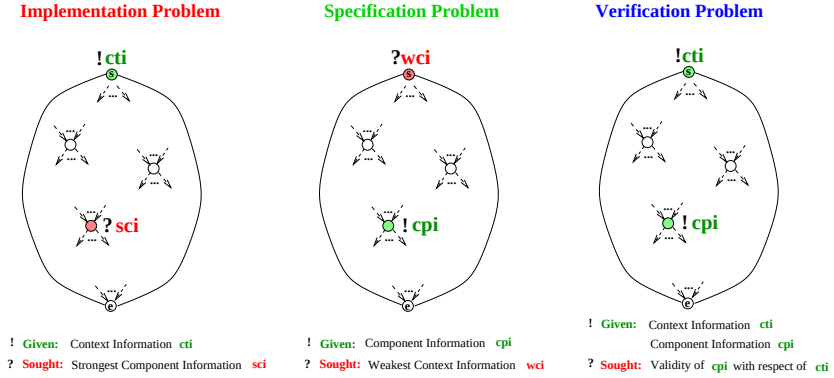
(R)DFA vs. Verifikation



Drei unterschiedliche Problemperspektiven (1)



Drei unterschiedliche Problemperspektiven (2)



Kapitel 8.1.4 Anwendungen reverser DFA

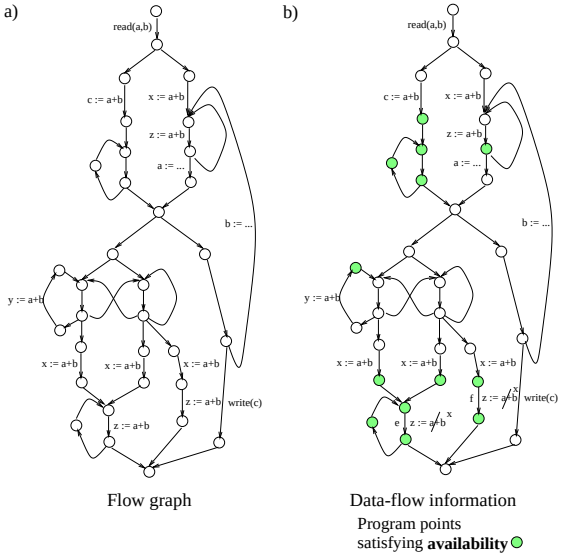
Anwendungen reverser DFA

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

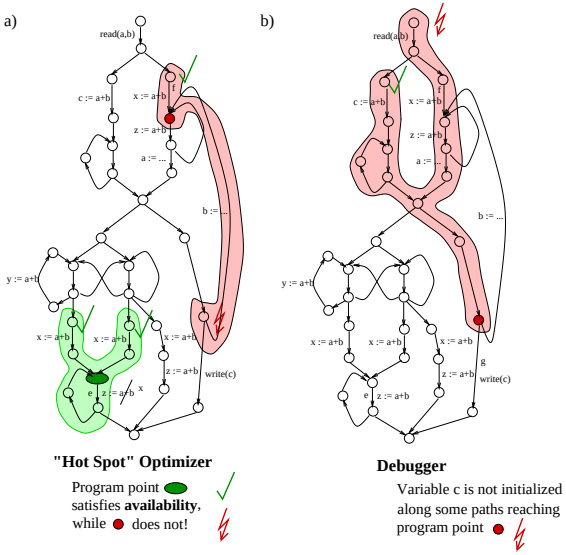
Insbesondere

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

Einfacher Analysator und Optimierer



"Hot Spot" Analysator&Optimierer, Debugger

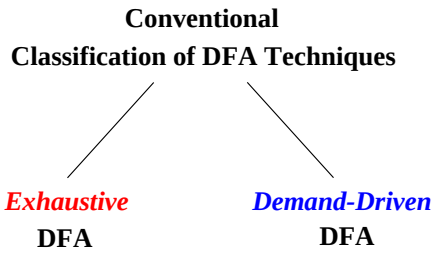


Hot-Spot Program Optimization

Siehe auch Ergänzungsfolien auf der Webpage zur LVA.

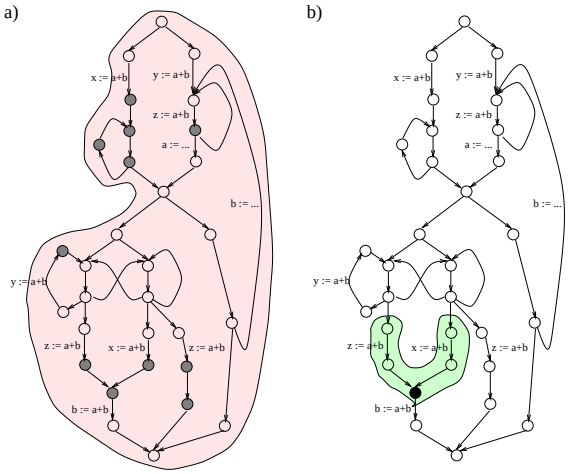
XDFA vs. DD-DFA

Erschöpfende (XDFA) vs. anforderungsgetriebene DFA (DD-DFA)



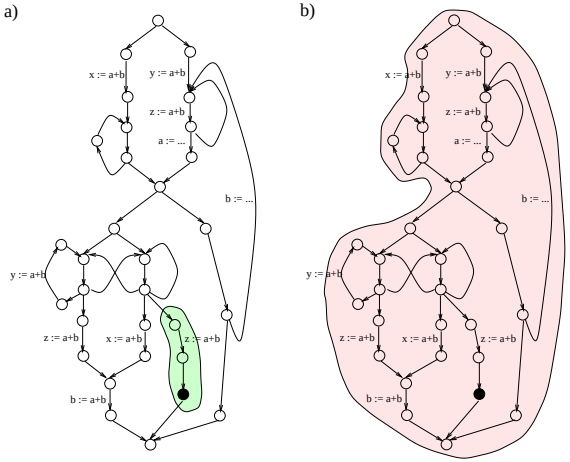
Beispiel 1: XDFA vs. DD-DFA

Verfügbarkeit an einem Punkt: Erschöpfend vs. anforderungsgetrieben

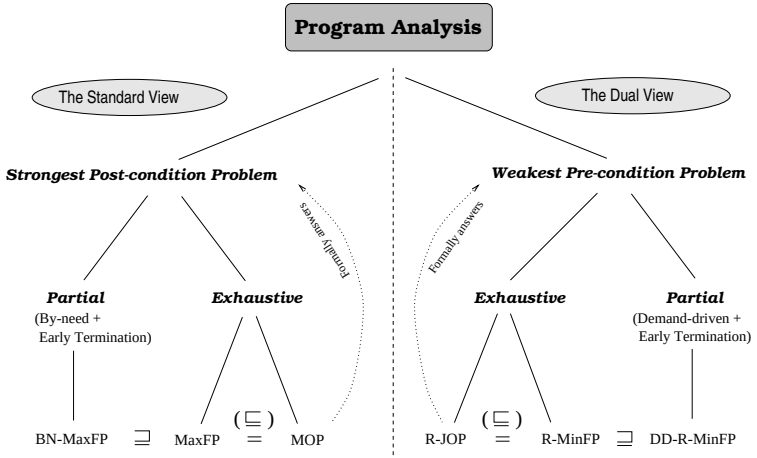


Beispiel 2: XDFA vs. DD-DFA

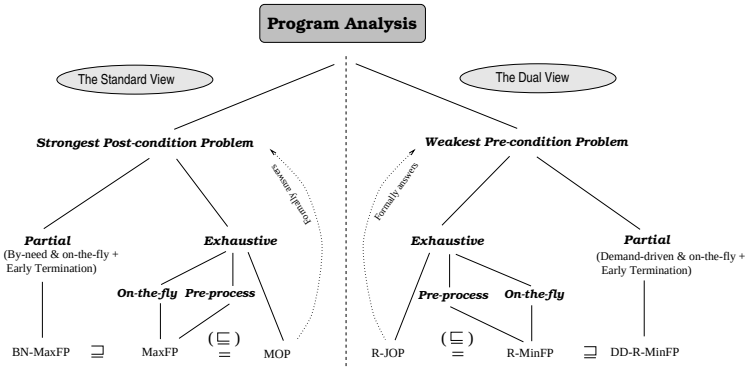
Verfügbarkeit an einem Punkt: Erschöpfend vs. anforderungsgetrieben



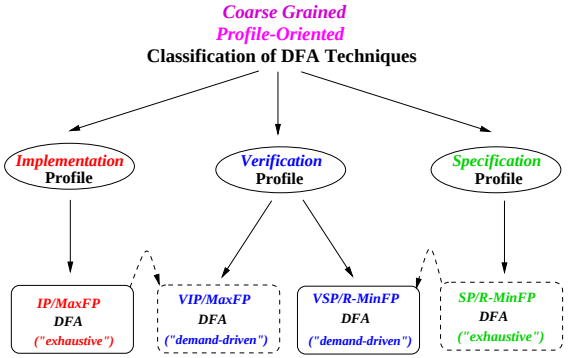
XDFA vs. DD-DFA (Fortsetzung)



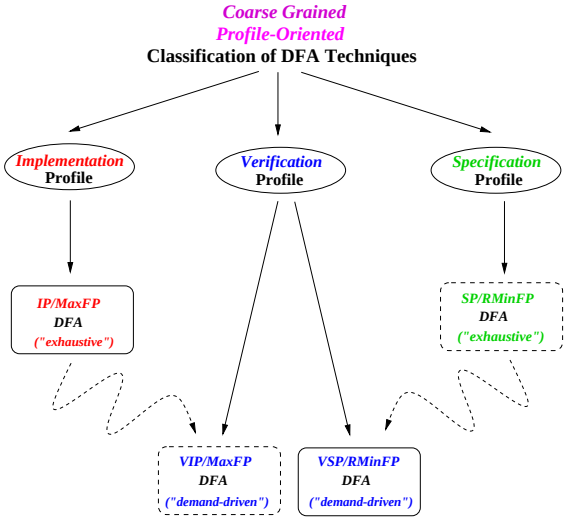
XDFA vs. DD-DFA (Fortsetzung)



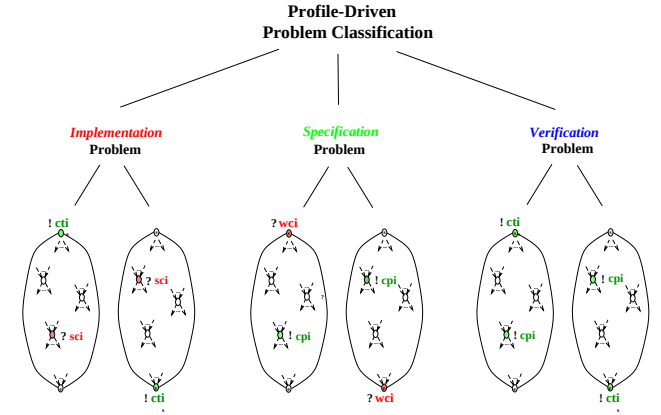
Eine andere Sicht (1)



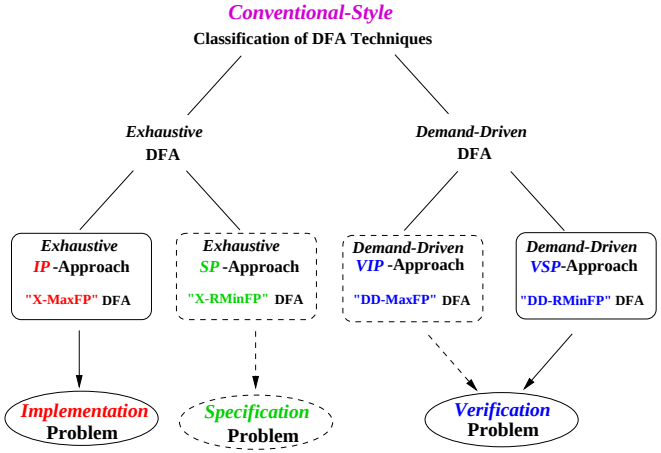
Eine andere Sicht (2)



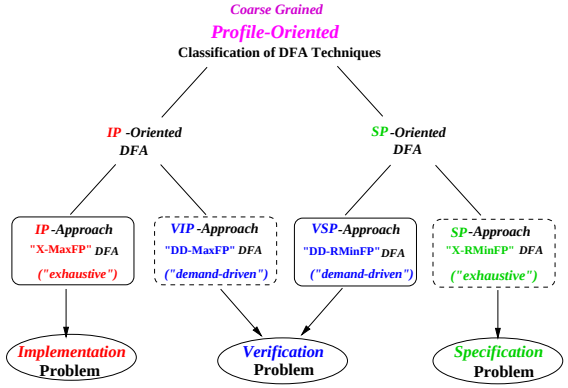
Im Überblick



Algorithmenorientierte Sicht



Problemorientierte Sicht



Kapitel 9 Pragmatik: Basisblöcke vs. Einzelanweisungen

Der Einfluss der Repräsentation

- Basisblöcke vs. Einzelanweisungen: Vor- und Nachteile
- Weitere Beispiele konkreter Datenflussanalysen/Datenflussanalyseprobleme

Einfach eine Geschmacksfrage?

Basisblöcke: Vermeintliche Vorteile

...und die ihnen gemeinhin aus ihrer Verwendung zugeschriebenen Vorteile ("Folk Knowledge"):

- *Performanz*: "...weil weniger Knoten in die teure iterative Fixpunktberechnung involviert sind".
- *Kompaktheit*: "...weil größere Programme in den Hauptspeicher passen".

Basisblöcke: Sichere Nachteile

...und die in der Folge aus ihrer Verwendung behaupteten Nachteile:

- *Höhere konzeptuelle Komplexität*: ... Basisblöcke führen zu einer unerwünschten *Hierarchisierung*, die sowohl theoretische Überlegungen wie Implementierungen erschwert.
- *Notwendigkeit von Prä- und Postprozessen*:: ... i.a. erforderlich, um die hierarchieinduzierten Zusatzprobleme zu behandeln (z.B. bei *dead code elimination*, *constant propagation*, ...); oder "trickbehaftete" Formulierungen nötig macht, um sie zu umgehen (z.B. bei *partial redundancy elimination*).
- *Eingeschränkte Allgemeinheit*: ... bestimmte praktisch relevante Analysen und Optimierungen sind nur schwer oder gar nicht auf Basisblockebene auszudrücken (z.B. *faint variable elimination*).

In der Folge

Zweierlei:

- Beispiele konkreter Datenflussanalyse (-probleme)
- Basisblöcke vs. Einzelinstruktionen: Vor- und Nachteile

MOP -Ansatz (Einzelanweisungen)

... für kantenbenannte Einzelanweisungsgraphen:

Die MOP-Lösung:

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

$MaxFP$ -Ansatz (Einzelanweisungen)

... für kantenbenannte Einzelanweisungsgraphen:

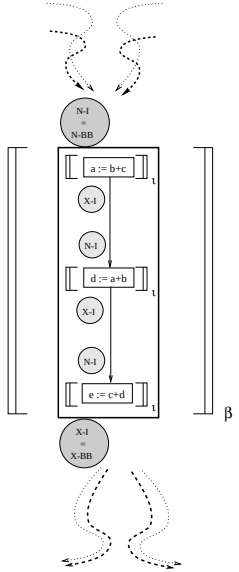
Die $MaxFP$-Lösung:

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

wobei $\mathsf{inf}^*_{c_s}$ die größte Lösung des $MaxFP$ -Gleichungssystems bezeichnet:

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

Hierarchisierung durch Basisblöcke



Vereinbarung

In der Folge werden

- Basisblockknoten mit fett gesetzten Buchstaben (**m**, **n**,...)
- Einzelanweisungsknoten mit normal gesetzten Buchstaben (m, n,...)

bezeichnet.

Weiters bezeichnen

- $\llbracket\ \rrbracket_\beta$ und
- $\llbracket\ \rrbracket_\iota$

(lokale) abstrakte Datenflussanalysefunktionale auf Basisblock- bzw. Einzelanweisungs- (Instruktions-) Ebene.

MOP -Ansatz (Basisblöcke) (1)

... für knotenbenannte Basisblockgraphen:

Die MOP-Lösung: (Basisblockebene)

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

mit

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

MOP -Ansatz (Basisblöcke) (2)

Die *MOP*-Lösung: (Anweisungsebene)

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

mit...

MOP -Ansatz (Basisblöcke) (3)

...mit

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

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

MaxFP -Ansatz (Basisblöcke) (1)

...für knotenbenannte Basisblockgraphen:

Die *MaxFP*-Lösung: (Basisblockebene)

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

mit

$$N\text{-}MFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \mathsf{pre}^{\beta}_{c_{\mathbf{s}}}(\mathbf{n}) \qquad \text{und}$$

$$X\text{-}MFP_{(\llbracket\ \rrbracket_\beta, c_{\mathbf{s}})}(\mathbf{n}) =_{df} \mathsf{post}^{\beta}_{c_{\mathbf{s}}}(\mathbf{n})$$

wobei...

MaxFP -Ansatz (Basisblöcke) (2)

...wobei $\mathsf{pre}^{\beta}_{c_{\mathbf{s}}}$ und $\mathsf{post}^{\beta}_{c_{\mathbf{s}}}$ die größten Lösungen des folgenden Gleichungssystems bezeichnen:

$$\mathbf{pre(n)} = \begin{cases} c_{\mathbf{s}} \text{ falls } \mathbf{n} = \mathbf{s} \\ \bigsqcap \{\, \mathbf{post(m)} \mid \mathbf{m} \in pred_{\mathbf{G}}(\mathbf{n}) \,\} \text{ sonst} \end{cases}$$

$$\mathbf{post(n)} = \llbracket\, \mathbf{n}\,\rrbracket_\beta(\mathbf{pre(n)})$$

MaxFP -Ansatz (Basisblöcke) (3)

Die *MaxFP*-Lösung: (Anweisungsebene)

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

mit

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

MaxFP -Ansatz (Basisblöcke) (4)

...wobei $\mathsf{pre}^{\iota}_{c_{\mathrm{s}}}$ und $\mathsf{post}^{\iota}_{c_{\mathrm{s}}}$ die größten Lösungen des folgenden Gleichungssystems bezeichnen:

$$\mathbf{pre}(n) = \begin{cases} \mathsf{pre}^{\beta}_{c_{\mathrm{s}}}(\mathtt{block}(n)) \\ \quad \text{falls } n = start(\mathtt{block}(n)) \\ \\ \mathbf{post}(m) \\ \quad \text{sonst } (m \text{ ist hier der eindeutig} \\ \quad \text{bestimmte Vorgänger von } n \\ \quad \text{in } \mathtt{block}(n)) \end{cases}$$

$$\mathbf{post}(n) = \llbracket\, n \,\rrbracket_\iota(\mathbf{pre}(n))$$

Verfügbarkeit von Ausdrücken (1)

...für knotenbenannte BB-Graphen:

Phase I: Die Basisblockebene

Lokale Prädikate: (assoziiert mit BB-Knoten)

- BB-XCOMP$_\beta(t)$: β enthält eine Anweisung ι, die t berechnet, und weder ι noch eine andere Anweisung von β nach ι modifiziert einen Operanden von t.
- BB-TRANSP$_\beta(t)$: β enthält keine Anweisung, die einen Operanden von t modifiziert.

Verfügbarkeit von Ausdrücken (2)

Das Gleichungssystem von Phase I:

$$\text{BB-N-AVAIL}_\beta = \begin{cases} \mathtt{ff} & \text{falls } \beta = \mathrm{s} \\ \prod\limits_{\hat{\beta} \in pred(\beta)} \text{BB-X-AVAIL}_{\hat{\beta}} & \text{sonst} \end{cases}$$

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

Verfügbarkeit von Ausdrücken (3)

Phase II: Die Anweisungsebene

Lokale Prädikate: (assoziiert mit EA-Knoten)

- **COMP**$_\iota(t)$: ι berechnet t.
- **TRANSP**$_\iota(t)$: ι modifiziert keinen Operanden von t.
- BB-N-AVAIL*, BB-X-AVAIL*: größte Lösung des Gleichungssystem von Phase I.

Das Gleichungssystem von Phase II:

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

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

Verfügbarkeit von Ausdrücken (4)

...für knotenbenannte EA-Graphen:

Lokale Prädikate: (assoziiert mit Knoten)

- **COMP**$_\iota(t)$: ι berechnet t.
- **TRANSP**$_\iota(t)$: ι modifiziert keinen Operanden von t.

Das Gleichungssystem:

$$\text{N-AVAIL}_\iota = \begin{cases} \texttt{ff} & \text{falls } \iota = s \\ \prod\limits_{\widehat{\iota} \in pred(\iota)} \text{X-AVAIL}_{\widehat{\iota}} & \text{sonst} \end{cases}$$

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

Verfügbarkeit von Ausdrücken (5)

...für kantenbenannte EA-Graphen:

Lokale Prädikate: (assoziiert mit EA-Kanten)

- **COMP**$_\varepsilon(t)$: Anweisung ι von Kante ε berechnet t.
- **TRANSP**$_\varepsilon(t)$: Anweisung ι von Kante ε ändert keinen Operanden von t.

Das Gleichungssystem:

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

Zwei weitere Beispiele

...zur Veranschaulichung des Einflusses der Flussgraphdarstellungsvariante:

- Konstantenfaltung (Constant folding)
- Geistervariablenelimination (Faint Variable Elimination)

In der Folge Formulierung dieser Probleme für die Varianten

- *knotenbenannte Basisblockgraphen* und
- *kantenbenannte Einzelanweisungsgraphen*

Konstantenfaltung am Bsp. "Einfacher Konstanten"

Zunächst zwei Hilfsfunktionen:

- Die Rückwärtssubstitution
- Die Zustandstransformation(sfunktion)

Rückwärtssubstitution und Zustandstransformation (1)

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 (als Erinnerung)

$$\theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

Zusammenhang von δ und θ

Bezeichne $\mathcal{I}$ die Menge aller Anweisungen.

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 (Einzelanweisungen) (1)

...für kantenbenannte Einzelanweisungsgraphen:

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

Das Gleichungssystem:

$\forall v \in \mathbf{V}.\ \mathrm{CP}_n =$

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

Rückwärtssubstitution & Zustandstransformation (2)

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.

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 (Basisblöcke) (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 (Basisblöcke) (2)

Das Gleichungssystem von Phase I:

$$\text{BB-N-CP}_\beta \;=\; \begin{cases} \sigma_0 \ \text{ falls } \ \beta = \mathbf{s} \\ \bigsqcap\{\text{BB-X-CP}_{\hat\beta} \mid \hat\beta \in pred(\beta)\} \\ \text{sonst} \end{cases}$$

$$\forall\, v \in \mathbf{V}.\ \text{BB-X-CP}_\beta(v) \;=\; \mathcal{E}(\Delta_\beta(v))(\text{BB-N-CP}_\beta)$$

Einfache Konstanten (Basisblöcke) (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 (Basisblöcke) (4)

Das Gleichungssystem von Phase II:

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

$$\forall\, v \in \mathbf{V}.\ \text{X-CP}_\iota(v) \;=\; \begin{cases} \text{BB-X-CP}^\star_{\texttt{block}(\iota)}(v) \\ \quad \text{falls } \iota = \mathit{end}(\texttt{block}(\iota)) \\ \mathcal{E}(\delta_\iota(v))(\text{N-CP}_\iota) \quad \text{sonst} \end{cases}$$

Geistervariablenelimination (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.
- *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 eine Ausgabeanweisung).
- *Ass-Used*$_\varepsilon(v)$: v ist eine in der Zuweisung ι von Kante ε rechtsseitig vorkommende Variable.

Geistervariablenelimination (2)

Das Gleichungssystem:

$$\begin{aligned} \text{FAINT}_n(v) = & \prod_{m \in \mathit{succ}(n)} \overline{\mathit{Rel\text{-}Used}_{(n,m)}(v)} \cdot \\ & (\text{FAINT}_m(v) \;+\; \text{MOD}_{(n,m)}(v)) \cdot \\ & (\text{FAINT}_m(\mathit{LhsVar}_{(n,m)}) + \overline{\mathit{Ass\text{-}Used}_{(n,m)}(v)}) \end{aligned}$$

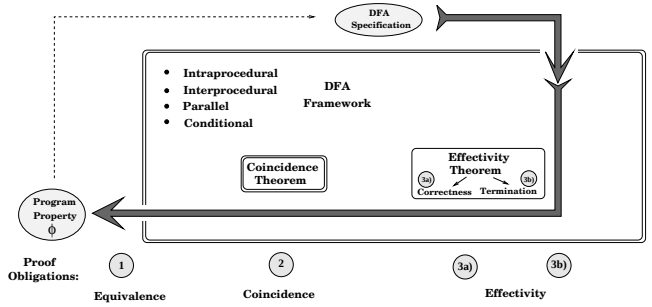
Geistervariablenelimination (3)

...ein typisches Beispiel für ein DFA-Problem, für das eine Formulierung

- auf (knoten- und kantenbenannten) Einzelanweisungsgraphen offensichtlich ist,
- auf (knoten- und kantenbenannten) Basisblockgraphen alles andere als ersichtlich ist.

Fazit

Für die Anwendungssicht reicht die allgemeine Rahmen- bzw. Werkzeugkistensicht und das Wissen, dass je nach Flussgraphrepräsentationsvariante unterschiedlich aufwändige Spezifikations-, Implementierungs- und Beweisverpflichtungen entstehen.



Kapitel 10 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

Beobachtung:

- Das Lösen eines Systems rekursiver Gleichungen ist äquivalent zur Berechnung eines Fixpunktes von $\mathcal{F}$, d.h.
  - eines *gemeinsamen Fixpunkts* $x = f_k(x)$ für alle f_k.

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 oft als *chaotische Iteration* bezeichnet.

Fixpunkttheoreme in der Literatur

- Fixpunkttheorem von Tarski [1955]
  - Garantiert Existenz kleinster Fixpunkte für monotone Funktionen über vollständigen partiellen Ordnungen
  - Iteration: $\vec{x}_0 = \vec{\perp}, \vec{x}_1 = \vec{f}(\vec{x}_0), \vec{x}_2 = \vec{f}(\vec{x}_1), \ldots$, wobei $\vec{x}_i$ den Wert von $\vec{x}$ nach der i-ten Iteration bezeichnet.
  - Erfolgreich, aber dennoch für viele Probleme 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 $\vec{x}_j$, $j \leq i$.

Literaturhinweise

- 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 Überblick 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

Ein weiteres Fixpunkttheorem, 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(2)

- 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 $(d_i)_{i \in \mathbb{N}}$ von Elementen $d_i \in D$ heißt *(aufsteigende) Kette*, falls $\forall i \in \mathbb{N}.\ d_i \sqsubseteq d_{i+1}$.
- Eine Kette $T =_{df} (d_i)_{i \in \mathbb{N}}$ heißt *stationär*, falls $\{d_i \mid i \in \mathbb{N}\}$ endlich ist.
- Eine partielle Ordnung $\sqsubseteq$ heißt *wohlfundiert*, falls jede Kette stationär ist.

Vorbereitung 2(2)

- Eine Funktion $f : D \to D$ auf D heißt
  - *vergrößernd*, falls $d \sqsubseteq f(d)$ für alle $d \in D$, und
  - *monoton*, falls $\forall d, d' \in D.\, d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$.
- Ist $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine Familie von Funktionen und $s = (s_1, \ldots, s_n) \in \mathbb{N}^*$, dann ist f_s definiert durch die Komposition

$$f_s =_{df} f_{s_n} \circ \cdots \circ f_{s_1}$$

Strategien, Iterationsfolgen, Fairness

Definition 10.1 [Strategie, Chaotische Iterationsfolge, Fairness]

Sei $\langle D; \sqsubseteq \rangle$ eine partielle Ordnung und $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$ eine Familie vergößernder 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 10.2 [Verzögert-Monotonie]

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

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

Lemma 10.3 $\mathcal{F}$ ist verzögert-monoton, wenn alle f_k im üblichen Sinn monoton sind.

Das neue Fixpunkttheorem

Theorem 10.4 [Chaotische Fixpunktiteration]

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

Dann gilt:

1. Der kleinste gemeinsame Fixpunkt $\mu\mathcal{F}$ von $\mathcal{F}$ existiert und ist gegeben durch $\bigsqcup f_\gamma(\bot)$.
2. $\mu\mathcal{F}$ wird stets in einer endlichen Zahl von Iterationsschritten erreicht.

Generischer Fixpunktalgorithmus 10.5

Der nichtdeterministische Rumpf-Algorithmus:

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

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 Indizesx i, um die i-te Komponente eines Vektors der Länge n zu bezeichnen.

Vektor-Iterationen

Definition 10.6 [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 Anwend'g von FP-Theorem 10.4

...zur Modellierung der Vektor-Iteration.

Vorbereitung: Verallgemeinerung des Strategiebegriffs auf einen Strategiebegriff für *Mengen*

Definition 10.7 [Mengenstrategie, faire Mengenstrategie]

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

  Intuition: $\gamma(i)$ liefert eine Menge J_i von Indizes aus $\{1, \ldots, n\}$, deren zugehörige Komponenten in Schritt i aktualisiert werden sollen.
- Eine Mengenstrategie heißt *fair* gdw

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

Modellierungsresultate 1(3)

Lemma 10.8 [Vektor-Iterationen]

Sei $\langle C; \sqsubseteq_C \rangle$ eine wohlfundierte partielle Ordnung mit kleinstem Element $\bot_C$, sei $n \in I\!N$ 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 $\gamma : I\!N \to \mathcal{P}(\{1, \ldots, n\})$ eine Mengenstrategie.

Dann gilt: Jede chaotische Iteration $f_\gamma(\bot)$ liefert eine Kette.

Modellierungsresultate 2(3)

Korollar 10.9 [Chaotische Vektor-Iterationen]

Sei $\langle C; \sqsubseteq_C \rangle$ eine wohlfundierte partielle Ordnung mit kleinstem Element $\bot_C$, sei $n \in I\!N$ 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(3)

- Korollar 10.9 ist ein Spezialfall von Fixpunkttheorem 10.4 für Vektor-Iterationen und folgt zusammen mit Lemma 10.8.
- Für $|\mathcal{F}| = 1$ reduziert sich Korollar 10.9 auf Tarskis Fixpunkttheorem im Fall wohlfundierter partieller Ordnungen.

Zweite Anw'g von FP-Theorem 10.4

...auf intraprozedurale DFA.

Erinnerung:
Das $MaxFP$-Gleichungssystem

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

Die $MaxFP$-Lösung:
Die größte Lösung des $MaxFP$-Gleichungssystems

Dual dazu: Das $MinFP$-Gleichungssystem 10.10

Das $MinFP$-Gleichungssystem

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

Die $MinFP$-Lösung:
Die kleinste Lösung des $MinFP$-Gleichungssystems

Der $MinFP$-Fixpunktalgorithmus 10.11

inf[s] := c_{s};
forall $n \in N \backslash \{\mathrm{s}\}$ **do** *inf*[n] := $\bot$ **od**;
workset := N;
while *workset* $\neq \emptyset$ **do**
  choose $n \in$ *workset* **in**
    workset := *workset*$\backslash\{\, n \,\}$;
    new := *inf*[n] $\sqcup$ $\bigsqcup\{[\![\, (m,n) \,]\!](\textbf{inf}[m]) \mid m \in pred_G(n)\}$;
    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 $G = (N, E, \mathrm{s}, \mathrm{e})$ der betrachtete Flussgraph.
- Sei $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ ein distributives DFA-Funktional, das die lokale abstrakte Semantik von G festlegt.
- Die Menge der Knoten N werde mit der Menge der natürlichen Zahlen $\{1, \ldots, \mathrm{n}\}$ identifiziert, wobei n die Anzahl der Knoten von N bezeichnet.

Zur Fixpunktcharakt. d. $MinFP$-Lösung

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 10.12

Für alle $d \in D$ gilt: d ist eine Lösung des $MinFP$-Gleichungssystems gdw d ist Fixpunkt von $f{=}_{df}(f^1, \ldots, f^{\mathrm{n}})$.

Theorem 10.13 [Korrektheit und Terminierung]

Jeder Lauf von $MinFP$-Algorithmus 10.11 terminiert mit der $MinFP$-Lösung.

Beweisanmerkungen

- Der $MinFP$-Fixpunktalgorithmus 10.11 folgt dem Muster von Rumpfalgorithmus 10.5 mit $\mathcal{F} = \{f_{\{k\}} \mid 1 \leq k \leq \mathrm{n}\}$.
- Die Verwendung von *workset*, die die Invariante *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 10.9 erfüllt, woraus Theorem 10.13 folgt.

Ausblick

Weitere Anwendungen des neuen Fixpunkttheorems 10.4 in Kapitel 11 und 12 zum Beweis der Optimalität von

- Partieller Dead-Code Elimination
- Partieller Redundant-Assignment Elimination

Andere Fixpunkttheoreme sind dafür nicht anwendbar.

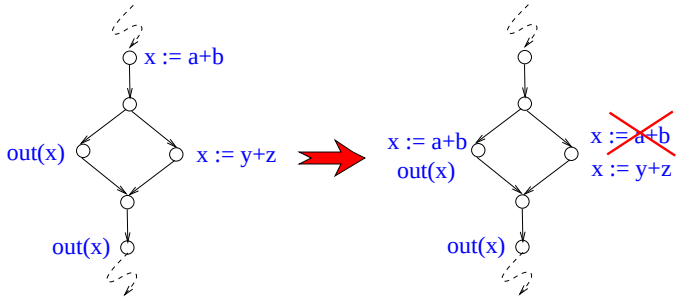
Kapitel 11 Partielle Dead/Faint-Code Elimination

Kapitel 11.1 Motivation

...und einleitende Beispiele.

Kanonisches Beispiel

Partial Dead-Code Elimination (PDCE)



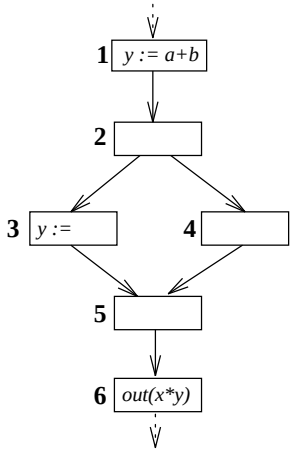
Elimination toten/geisterhaften Codes

Zwei unterschiedliche Optimierungstransformationen:

- Partial Dead-Code Elimination (PDCE)
- Partial Faint-Code Elimination (PFCE)

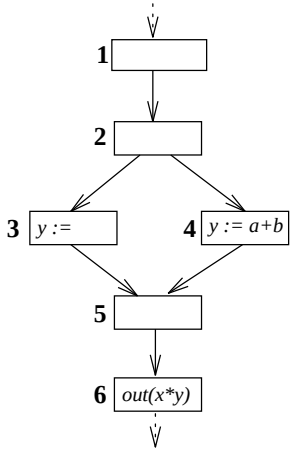
PDCE anhand eines Beispiels 1(2)

Das Ausgangsprogramm



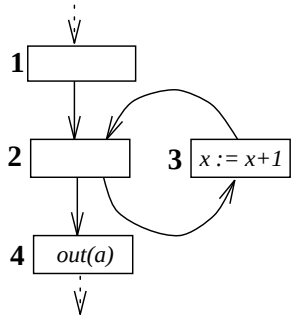
PDCE anhand eines Beispiels 2(2)

Das optimierte Programm



PFCE anhand eines Beispiels

Die Anweisung $x := x + 1$: *Geisterhaft*, aber nicht tot.



Kapitel 11.2 Theorie v. PDCE&PDFE

Theorie: Transformation(en) und Optimalität

Dazu führen wir zunächst notwendige Begriffe ein.

Anweisungsmuster

Bezeichne in der Folge...

- α ein sog. *Anweisungsmuster*:

$$\alpha \equiv x := t$$

- $\mathcal{AP}$ die Menge aller Anweisungsmuster

Verzögerbarkeit von Anweisungen

Definition [Assignment Sinking] An *assignment sinking* for α is a program transformation that

- eliminates some occurrences of α,
- inserts instances of α at the entry or the exit of some basic blocks being reachable from a basic block with an eliminated occurrence of α.

Lokale Blockaden

Definition [Local Barriers]
The sinking of an assignment pattern α is *blocked* by an instruction that

- modifies an operand of t or
- uses the variable x or
- modifies the variable x.

Zulässige Anweisungsverzögerungen

Definition [Admissible Assignment Sinking] An assignment sinking for α is *admissible*, iff it satisfies:

1. The removed assignments are *substituted*, i.e., on every program path leading from n to $\mathbf{e}$, where an occurrence of α has been eliminated at n, an instance of α has been inserted at a node m on the path such that α is not blocked by any instruction between n and m.
2. The inserted assignments are *justified*, i.e., on each program path from $\mathbf{s}$ to n, where an instance of α has been inserted at n, an occurrence of α has been eliminated at a node m on the path such that α is not blocked by any instruction between m and n.

Elimination von Anweisungen

Definition [Assignment Elimination] An *assignment elimination* for α is a program transformation that eliminates some original occurrences of α in the argument program.

Zulässige Elimination von Anweisungen

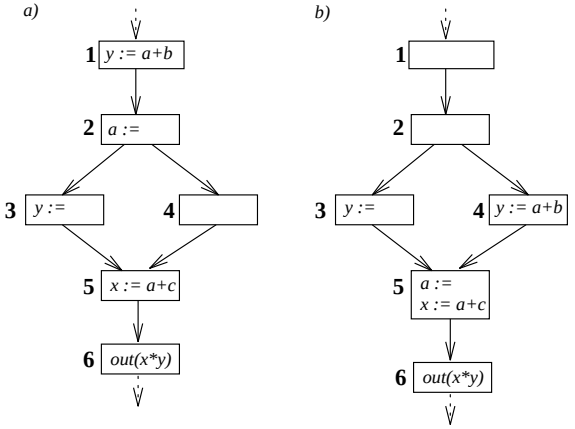
Definition [Dead (Faint) Code Elimination] A *dead* (*faint*) *code elimination* for an assignment pattern α is an assignment elimination for α, where some dead (faint) occurrences of α are eliminated.

Second-Order Effekte

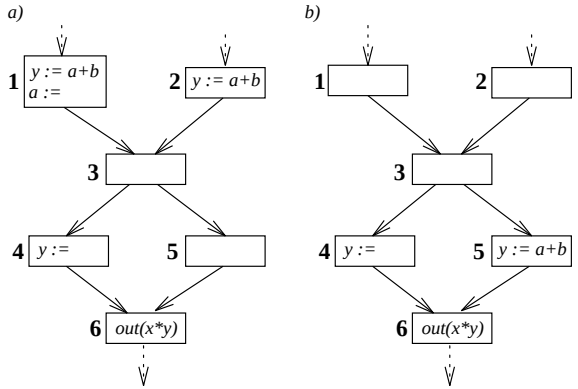
Sog. *second-order* Effekte im Fall von PDCE:

- *Sinking/Elimination*-Effekte (Primäreffekt)
- *Sinking/Sinking*-Effekte
- *Elimination/Sinking*-Effekte
- *Elimination/Elimination*-Effekte

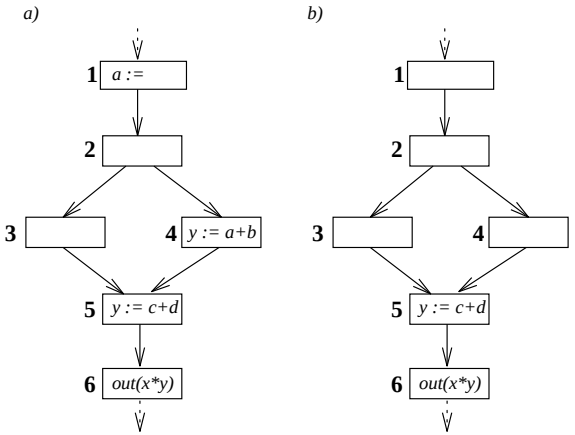
Bsp. f. Sinking/Sinking-Effekt



Bsp. f. Elimination/Sinking-Effekt

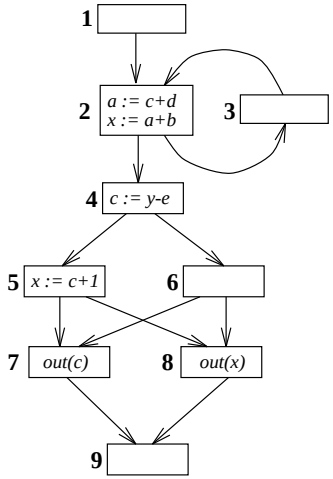


Bsp. f. Elimination/Elimination-Effekt



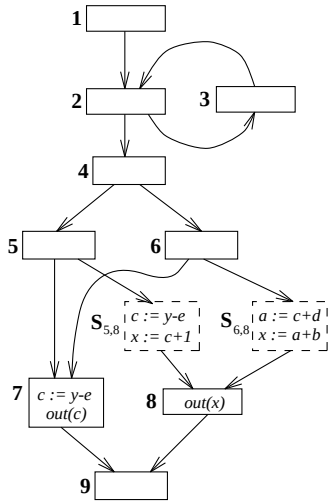
2nd-Order Effekte im Zusammenspiel 1

Das Ausgangsprogramm:



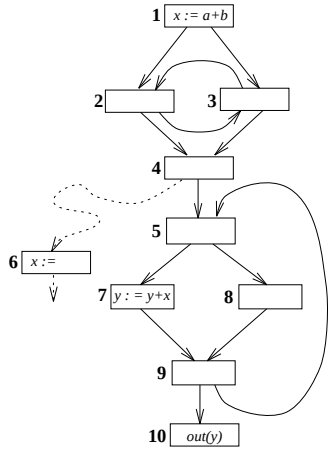
2nd-Order Effekte im Zusammenspiel 2

Das optimierte Programm



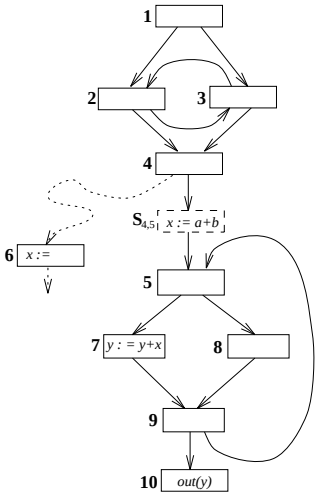
Sog. Kritische Kanten

...behindern die Optimierung.



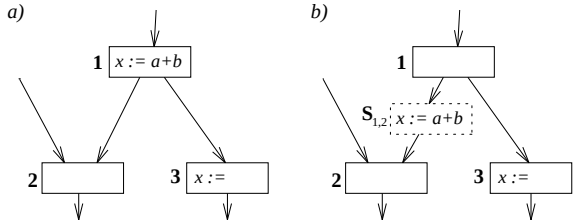
Für bestmögliche Resultate

Kritische Kanten spalten!



Kritische Kanten

Eine Kante heißt *kritisch*, wenn sie von einem Knoten mit mehr als einem Nachfolger zu einem Knoten mit mehr als einem Vorgänger führt.



Die PDCE/PDFE-Transformationen

Definition [Partial Dead (Faint) Code Elimination] *Partial dead (faint) code elimination* PDCE (PFCE) is an arbitrary sequence of admissible assignment sinkings and dead (faint) code eliminations.

Schreibweisen und Vereinbarungen

- $G \vdash_{PDCE} G'$ bzw. $G \vdash_{PFCE} G'$:
  ...G' resultiert aus G durch Anwendung einer zulässigen Verzögerungs- oder (dead/faint) Eliminationstransformation.
- $\tau \in \{PDCE, PFCE\}$:
  ...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-Transformationen entstehende *Universum*.

Der Optimalitätsbegriff

Definition [Optimality of *PDCE* (*PFCE*)]

1. Let $G', G'' \in \mathcal{G}_\tau$. Then G' is *better* than G'', in signs $G'' \sqsubseteq\!\!\!\!\sim\; G'$, if and only if

   $$\forall\, p \in \mathbf{P}[\mathbf{s},\mathbf{e}]\ \forall \alpha \in \mathcal{AP}.\ \alpha\#(p_{G'}) \leq \alpha\#(p_{G''})$$

   where $\alpha\#(p_{G'})$ and $\alpha\#(p_{G''})$ denote the number of occurrences of the assignment pattern α on p in G' and G'', respectively.

2. $G^* \in \mathcal{G}_\tau$ is *optimal* if and only if G^* is better than any other program in $\mathcal{G}_\tau$.

Zur Relation "besser"

Die Relation $\sqsubseteq\!\!\!\!\sim$ ist eine

- Quasiordnung (d.h. reflexiv, transitiv, aber nicht antisymmetrisch)

Monotonie und Dominanz

Sei

- $\vec{\sqsubseteq}_\tau =_{df} (\sqsubseteq\!\!\!\!\sim \cap \vdash_\tau)^*$
- $\mathcal{F}_\tau \subseteq \{f \mid f : \mathcal{G}_\tau \to \mathcal{G}_\tau\}$ 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:*
     $\forall\, G', G'' \in \mathcal{G}_\tau.\ G' \vdash_\tau G'' \Rightarrow \exists\, f \in \mathcal{F}_\tau.\ G'' \vec{\sqsubseteq}_\tau f(G')$

Optimalitätsresultat

Theorem [Existence of Optimal Program] $\mathcal{G}_\tau$ has an optimal element (wrt $\sqsubseteq\!\!\!\!\sim$) which can be computed by any sequence of function applications that contains all elements of $\mathcal{F}_\tau$ 'sufficiently' often.

Beweis ...mithilfe von Fixpunkttheorem 10.4 (Geser, Knoop, Lüttgen, Rüthing, Steffen [CC'96])

Anmerkungen zum Optimalitätsresultat

- PDCE und PFCE erfüllen die Anforderungen des vorstehenden Optimalitätstheorems
- Das optimale Programm ist eindeutig bestimmt bis auf irrelevante Umsortierungen von Anweisungen in Basisblöcken.

Kapitel 11.3 Praxis von PDCE&PDFE

Praxis: Implementierung / Spezifikation der DFAs

Dazu benötigen wir eine Reihe von Hilfsprädikaten:

Lokale Prädikate für die Spezifikation der lokalen abstrakten Interpretation:

- USED, *Rel-Used*, *Ass-Used* und MOD
- LOC-DELAYED und LOC-BLOCKED

und der darauf aufbauenden DFAs:

- Total Dead-Code Elimination (TDCE)
- Total Faint-Code Elimination (TFCE)
- Assignment Sinking (AS / Delayability)

Lokale Prädikate für TDCE und TDFE

Lokale Prädikate für *Total Dead-Code Elimination* (*TDCE*) und *Total Faint-Code Elimination* (*TFCE*):

- $\mathrm{USED}_\iota(x)$: x is a right-hand side variable of the instruction ι.
- *Rel-Used*$_\iota(x)$: x is a right-hand side variable of the relevant instruction ι.
- *Ass-Used*$_\iota(x)$: x is a right-hand side variable of the assignment statement ι.
- $\mathrm{MOD}_\iota(x)$: x is the left-hand side variable of the instruction ι.

Die TDCE-Analyse

The Dead Variable Analysis (TDCE):

$$\mathrm{N\text{-}DEAD}_\iota = \neg\mathrm{USED}_\iota * (\mathrm{X\text{-}DEAD}_\iota + \mathrm{MOD}_\iota)$$

$$\mathrm{X\text{-}DEAD}_\iota = \prod_{\widehat{\iota} \in succ(\iota)} \mathrm{N\text{-}DEAD}_{\widehat{\iota}}$$

Die TFCE-Analyse

The Faint Variable Analysis (TFCE):

(Slotwise simultaneously for all variables x)

$$\begin{aligned}\text{N-FAINT}_\iota(x) &= \neg \textit{Rel-Used}_\iota(x) * \\ &\quad (\text{X-FAINT}_\iota(x) + \text{MOD}_\iota(x)) * \\ &\quad (\text{X-FAINT}_\iota(LhsVar_\iota) + \neg \textit{Ass-Used}_\iota(x))\end{aligned}$$

$$\text{X-FAINT}_\iota(x) = \prod_{\hat{\iota} \in succ(\iota)} \text{N-FAINT}_{\hat{\iota}}(x)$$

Lokale Prädikate für AS

- LOC-DELAYED$_n(\alpha)$: There is a sinking candidate of α in n.
- LOC-BLOCKED$_n(\alpha)$: The sinking of α is blocked by some instruction of n.

Die AS-Analyse

AS, eine Verzögerbarkeitsanalyse (delayability):

Delayability Analysis:

$$\text{N-DELAYED}_n = \begin{cases} \texttt{ff} & \text{if } n = \mathrm{s} \\ \prod_{m \in pred(n)} \text{X-DELAYED}_m & \text{otherwise} \end{cases}$$

$$\begin{aligned}\text{X-DELAYED}_n &= \text{LOC-DELAYED}_n + \\ &\quad \text{N-DELAYED}_n * \neg \text{LOC-BLOCKED}_n\end{aligned}$$

Einsetzungspunkte (Keine DFA!)

Die aus der AS-Analyse resultierenden Einsetzungspunkte:

Insertion Points:

$$\text{N-INSERT}_n =_{df} \text{N-DELAYED}_n * \text{LOC-BLOCKED}_n$$

$$\text{X-INSERT}_n =_{df} \text{X-DELAYED}_n * \sum_{m \in succ(n)} \neg \text{N-DELAYED}_m$$

Die PDCE/PDFE-Transformationen 1

Bezeichnen

- *pdce* und
- *pfce*

die aus vorigen Analysen abgeleiteten Algorithmen für die

- *Elimination partiell toter Anweisungen* (PDCE) und
- *Elimination partiell geisterhafter Anweisungen* (PFCE)

Die PDCE/PDFE-Transformationen 2

Bezeichnen

- G_{pdce} und
- G_{pfce}

die aus G durch pdce und pfce resultierenden Programme, sowie

- $\mathcal{G}_{PDCE}$ und
- $\mathcal{G}_{PFCE}$

die von den Elementartransformation von *pdce* und *pfce* aufgespannten Universen für G.

Dann gilt...

PDCE/PDFE-Hauptresultate: Korrektheit

Theorem [Correctness]

1. $G_{pdce} \in \mathcal{G}_{PDCE}$
2. $G_{pfce} \in \mathcal{G}_{PFCE}$

PDCE/PDFE-Hauptresultate: Optimalität

Theorem [Optimality Theorem]

1. G_{pdce} *is optimal in* $\mathcal{G}_{PDCE}$
2. G_{pfce} *is optimal in* $\mathcal{G}_{PFCE}$

Veranschaulichung

Konzeptuell können wir PDCE wie folgt verstehen:

- PDCE = (AS + TDCE)*

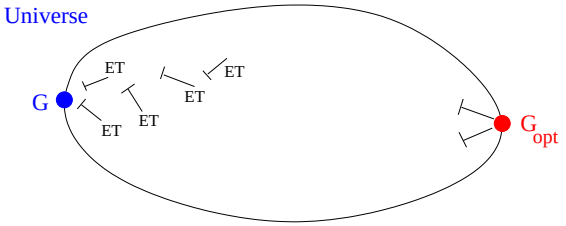
PDCE-Optimalitätsergebnis

Ableitungsrelation $\vdash$...

- PDCE... $G \vdash_{AS,TDCE} G'$ (ET={AS,TDCE})

Wir können beweisen:

Optimalitätstheorem
Für PDCE ist $\vdash_{ET}$ konfluent und terminierend



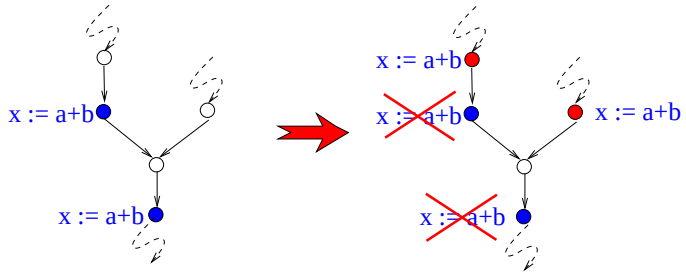
Kapitel 12 Partielle Redundant-Assignment Elimination

Kapitel 12.1 Motivation

...und einleitende Beispiele.

Kanonisches Beispiel

Partial Redundant-Assignment Elimination (PRAE)



Elimination partiell redundanter Anweisungen

Wir können unterscheiden:

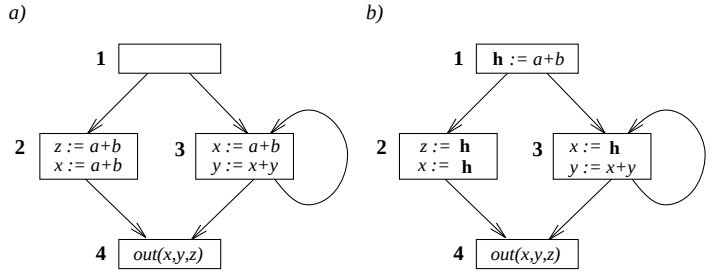
- (Pure) Expression Motion (PuEM) (bzw. PuPREE)
- (Pure) Assignment Motion (PuAM) (bzw. PuPRAE)

...sowie

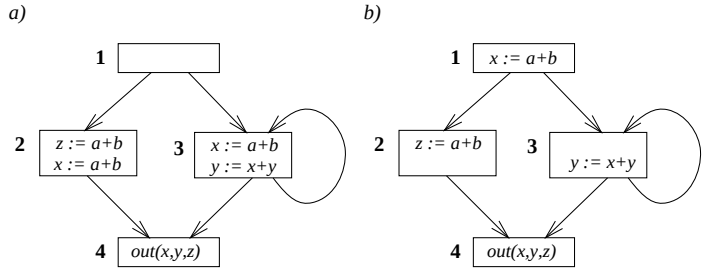
- Uniform Expression and Assignment Motion (AM)

Dazu einige Beispiele...

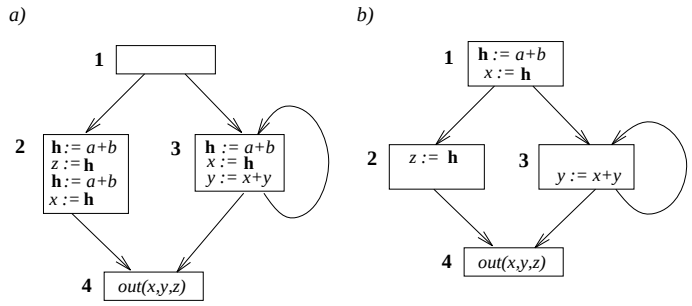
Pure Expression Motion (PuEM)



Pure Assignment Motion (PuAM)

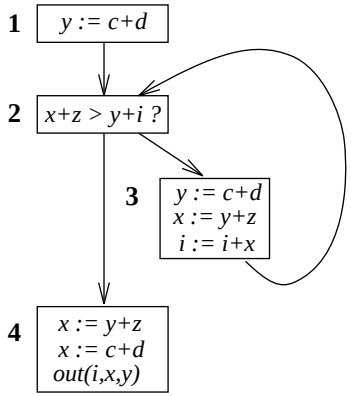


Uniform Expression&Assignment Motion



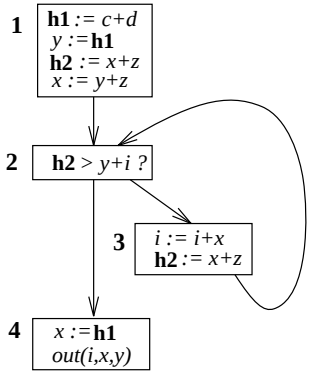
Ein größeres Beispiel 1(2)

Das Ausgangsprogramm



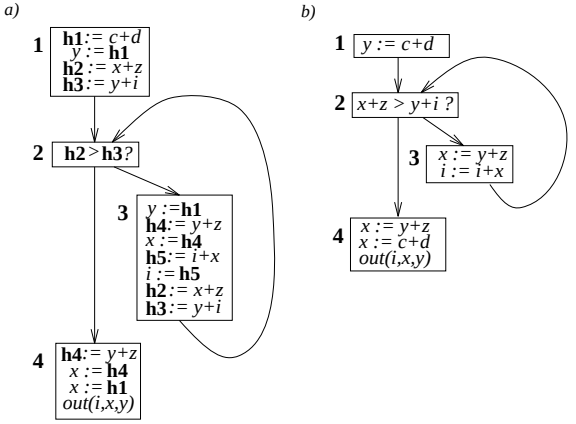
Ein größeres Beispiel 2(2)

Das optimierte Programm



Zum Vergleich

Die (schwächeren) Verbesserungseffekte von PuEM (a) und PuAM (b)



Kapitel 12.2 Theorie von AM

Theorie: Transformation und Optimalität

PRAE: Der einheitl. PuEM/PuAM-Alg.

Ein dreistufiger Algorithmus:

- *Präprozess*
  Ersetze jedes Vorkommen einer Anweisung $x := t$ durch die Sequenz $h_t := t;\ x := h_t$.
- *Hauptprozess*
  Wiederholte Anwendung von
  - Assignment Hoisting (AH) und
  - Totally Redundant Assignment Elimination (TRAE)

  bis Stabilität eintritt.
- *Postprozess*
  Aufräumen isolierter Initialisierungen

Beachte

...dank des Präprozesses wird PuEM durch PuAM abgedeckt und einheitlich erfasst!

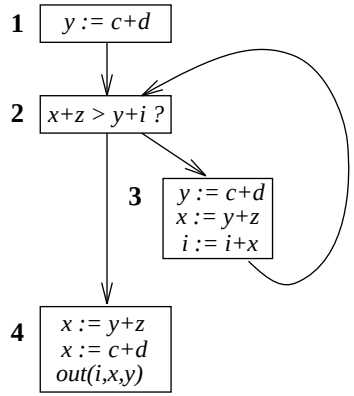
Second-Order Effekte auch für PRAE

Second order Effekte im Fall von PRAE:

- *Hoisting-Hoisting* Effekte
- *Hoisting-Elimination* Effekte
- *Elimination-Hoisting* Effekte
- *Elimination-Elimination* Effekte

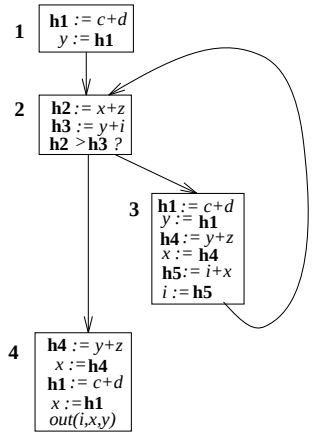
PRAE angesetzt auf das lfd. Bsp. 1(4)

Das Ausgangsprogramm



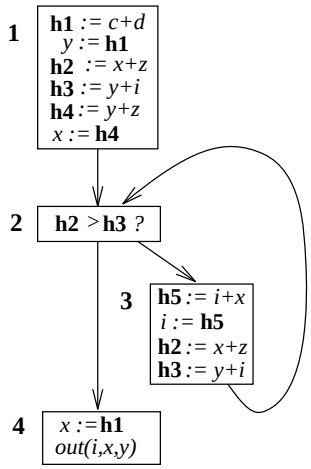
PRAE angesetzt auf das lfd. Bsp. 2(4)

Nach dem Präprozess



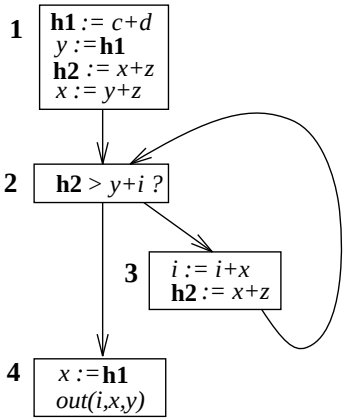
PRAE angesetzt auf das lfd. Bsp. 3(4)

Nach dem Hauptprozess



PRAE angesetzt auf das lfd. Bsp. 4(4)

Das Endresultat: Das optimierte Programm nach dem Postprozess



PRAE-Hauptresultate: Korrektheit

Analog zu PDCE/PFCE gilt:

Theorem [Correctness] $G_{GlobAlg} \in \mathcal{G}$

PRAE-Hauptresultate: Optimalität (1)

Theorem [Expression-Optimality] $G_{GlobAlg}$ is expression-optimal in $\mathcal{G}$, i.e., it requires at most as many expression evaluations at run-time than any other program that can be obtained via EM and AM transformations.

PRAE-Hauptresultate: Optimalität (2)

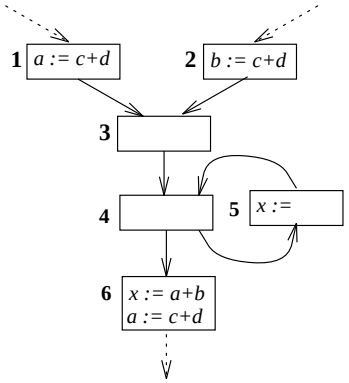
Theorem [Relative Assignment-Optimality] $G_{GlobAlg}$ is relatively assignment-optimal in $\mathcal{G}$, i.e., it is impossible to decrease the number of assignments required by $G_{GlobAlg}$ at run-time by means of EM and AM transformations.

PRAE-Hauptresultate: Optimalität (3)

Theorem [Relative Temporary-Optimality] $G_{GlobAlg}$ is relatively temporary-optimal in $\mathcal{G}$, i.e. it is impossible to decrease the number of assignments to temporaries or the length of temporary lifetimes in $G_{GlobAlg}$ by a corresponding assignment sinking.

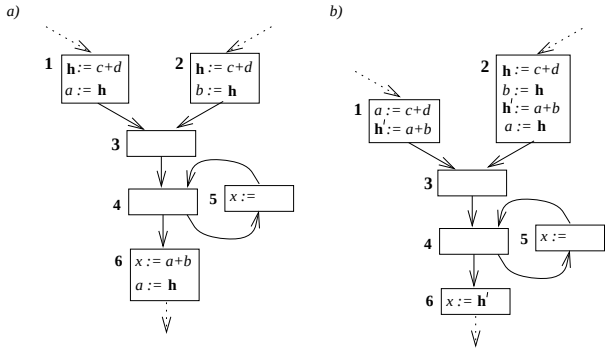
Warum nur “relative Optimalität” (1)

Betrachte dazu:



Warum nur “relative Optimalität” (2)

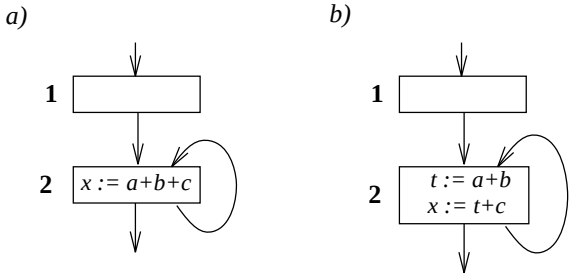
...und folgende zwei unvergleichbare Transformationsresultate:



⇒ Relative Optimalität ist das Beste, was wir erzielen können!

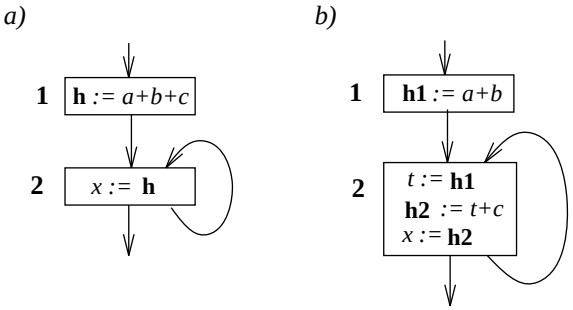
Weitere Phänomene 1(3)

Komplex- vs. 3-Adresscode



Weitere Phänomene 2(3)

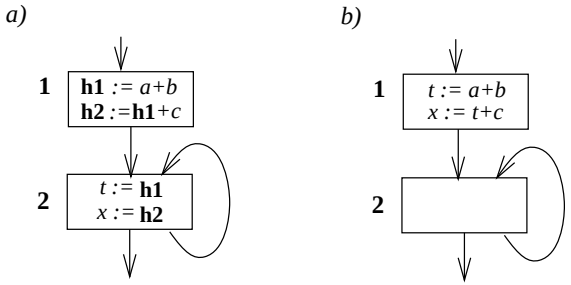
Der Effekt von PuEM:



Weitere Phänomene 3(3)

Die Effekte von

- PuEM + Copy Propagation: Fig. a)
- Uniform PuEM&PuAM = PRAE: Fig. b)



Veranschaulichung

Konzeptuell können wir PRAE wie folgt verstehen:

- PRAE = (AH + TRAE)*

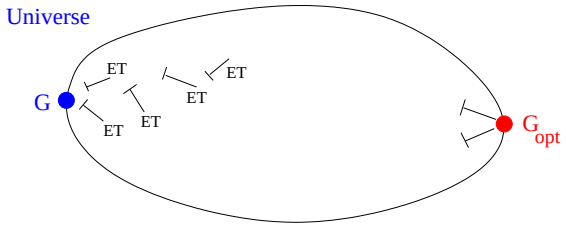
PRAE-Optimalitätsergebnis

Ableitungsrelation $\vdash$...

- PRAE... $G \vdash_{AH,TRAE} G'$ (ET={AH,TRAE})

Wir können beweisen:

Optimalitätstheorem
Für PRAE ist $\vdash_{ET}$ konfluent und terminierend



Kapitel 14 PRAE + PDCE

Erinnerung

Konzeptuell können wir PRAE und PDCE wie folgt verstehen:

- PRAE = (AH + TRAE)*
- PDCE = (AS + TDCE)*

Idee:

- AP = (AS + TDCE + AH + TRAE)*

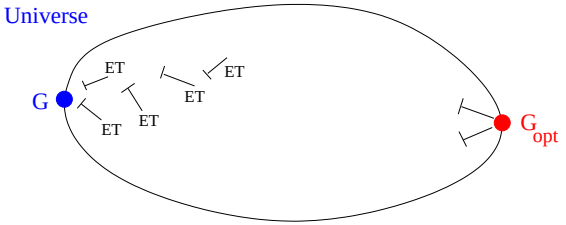
PRAE/PDCE-Optimalitätsergebnisse

Ableitungsrelation $\vdash$...

- PRAE... $G \vdash_{AH,TRAE} G'$ (ET={AH,TRAE})
- PDCE... $G \vdash_{AS,TDCE} G'$ (ET={AS,TDCE})

Wir können beweisen:

Optimalitätstheorem
Für PRAE und PDCE ist $\vdash_{ET}$ konfluent und terminierend



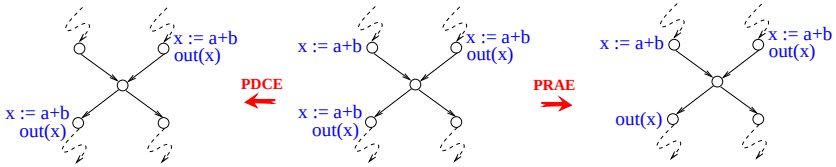
PRAE + PDCE = AP

Betrachte nun

- Assignment Placement AP
  AP = (AH + TRAE + AS + TDCE)*

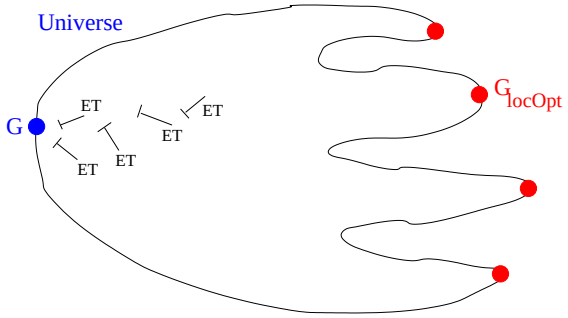
...sollte mächtiger sein als PRAE und PDCE separat!

In der Tat, aber:



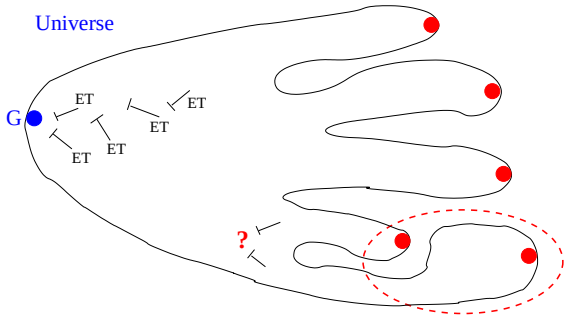
Verlust globaler Optimalität

Konfluenz und folglich globale Optimalität ist verloren!



Tatsächlich

...gibt es sogar Szenarien, in denen wir mit Universen wie dem Folgenden enden können:



- Knoop, J., and Mehofer, E. Distribution assignment placement: Effective optimization of redistribution costs. IEEE Transactions on Parallel and Distributed Systems 13, 6 (2002), 628 - 647.

Kapitel 15 Interprozedurale Datenflussanalyse

Interprozedurale Datenflussanalyse

In zwei Teilen...

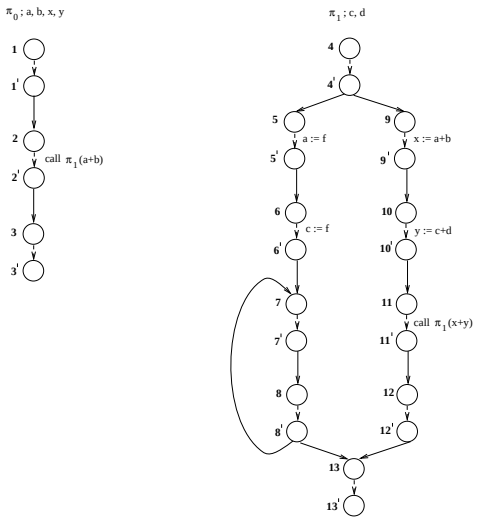
- Der Basisfall: Parameterlose Prozeduren, keine lokalen Variablen
- Erweiterungen
  - Wertparameter und lokale Variablen
  - Referenzparameter
  - Prozedurparameter

Interprozedurale Datenflussanalyse

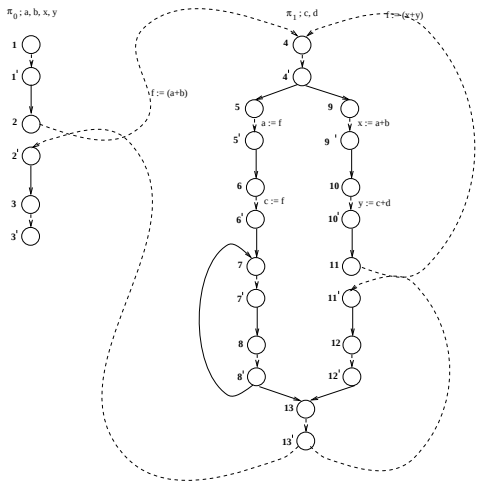
Darstellung von Programmen durch

- Flussgraphsysteme
- Interprozedurale Flussgraphen

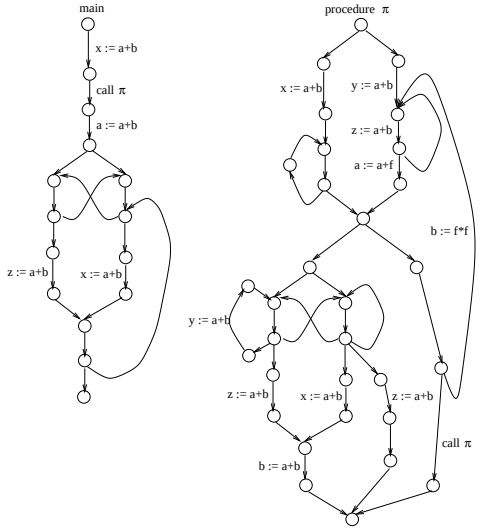
Bsp. 1: Flussgraphsystem



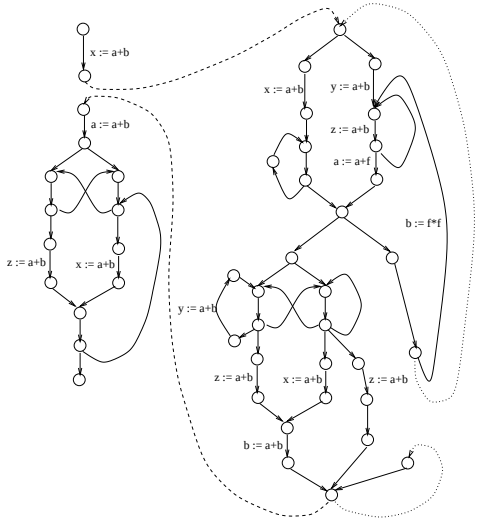
Bsp. 1: Interprozeduraler Flussgraph



Bsp. 2: Flussgraphsystem



Bsp. 2: Interprozeduraler Flussgraph



Vorbereitung interprozeduraler (paralleler) DFA

Der "funktionale" $MaxFP$-Ansatz und das $MaxFP$-Gleichungssystem:

$$[\![n]\!] = \begin{cases} Id_{\mathcal{C}} & \text{falls } n = \mathbf{s} \\ \bigsqcap\{[\![(n,m)]\!] \circ [\![m]\!] \mid m \in pred(n)\} & \text{sonst} \end{cases}$$

In der Folge bezeichne das Funktional

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

die größte Lösung des obigen Gleichungssytems.

$MaxFP$- vs. funktionalem $MaxFP$-Ansatz

Das Äquivalenztheorem beschreibt den Zusammenhang von $MaxFP$- und funktionalem $MaxFP$-Ansatz:

Theorem [Äquivalenz]

$\forall n \in N \ \forall c_{\mathbf{s}} \in \mathcal{C}.\ MaxFP_{(G,[\![\ ]\!])}(n)(c_s) = [\![n]\!](c_{\mathbf{s}})$

Ausblick

Die funktionale Perspektive auf den $MaxFP$ -Ansatz liefert den Schlüssel zu

- interprozeduraler (d.h. von Programmen mit Prozeduren)
- paralleler (d.h. von Programmen mit Parallelität)

Datenflussanalyse.

Kapitel 15.1 Interprozeduraler Basisfall

(Lokale) abstrakte Semantik

Zwei Komponenten:

- *Datenflussanalyseverband* $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$
- *Datenflussanalysefunktional* $[\![\ ]\!]' : E^* \to (\mathcal{C} \to \mathcal{C})$

Kapitel 15.1.1 *IMOP*-Ansatz

Der *IMOP*-Ansatz

Die *IMOP*-Lösung:

$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ IMOP_{c_{\mathbf{s}}}(n) =_{df} \bigsqcap \{\, [\![\, p \,]\!]'(c_{\mathbf{s}}) \mid p \in \mathbf{IP}[\mathbf{s}, n] \,\}$

wobei IP$[\mathbf{s}, n]$ die Menge der *interprozedural gültigen Pfade* von $\mathbf{s}$ nach n bezeichnet.

Kapitel 15.1.2 *IMaxFP*-Ansatz

Der *IMaxFP*-Ansatz 1(3)

Zweistufiger Ansatz

Stufe 1: Semantik von Prozeduren (2nd-Order Effects)

$[\![n]\!] =$

$$\begin{cases} Id_{\mathcal{C}} & \text{falls } n \in \{ \mathrm{s}_0, \ldots, \mathrm{s}_k \} \\ \bigsqcap \{ [\![(m,n)]\!] \circ [\![m]\!] \mid m \in pred_{flowGraph(n)}(n) \} & \text{sonst} \end{cases}$$

und

$$[\![e]\!] = \begin{cases} [\![e]\!]' & \text{falls } e \in E \backslash E_{call} \\ [\![end(caller(e))]\!] & \text{sonst} \end{cases}$$

Der *IMaxFP*-Ansatz 2(3)

Stufe 2: Die "eigentliche" interprozedurale DFA

Das *IMaxFP*-Gleichungssystem

inf $(n) =$

$$\begin{cases} c_{\mathrm{s}} & \text{falls } n = \mathrm{s}_0 \\ \bigsqcap \{ \textbf{inf}\ (src(e)) \mid e \in caller(flowGraph(n)) \} & \text{falls } n \in \{\mathrm{s}_1, \ldots, \mathrm{s}_k\} \\ \bigsqcap \{ [\![(m,n)]\!] (\textbf{inf}\ (m)) \mid m \in pred_{flowGraph(n)}(n) \} & \text{sonst} \end{cases}$$

Der *IMaxFP*-Ansatz 3(3)

Die *IMaxFP*-Lösung

$\forall\, c_{\mathrm{s}} \in \mathcal{C}\ \forall\, n \in N.\ IMaxFP_{c_{\mathrm{s}}}(n) =_{df} \textbf{inf}\ ^{*}_{c_{\mathrm{s}}}(n)$

Kapitel 15.1.3 Hauptresultate

Hauptergebnisse 1(2)

Sicherheit und Koinzidenz der ersten Stufe:

Theorem [2nd-Order] For all $e \in E_{call}$, we have:

1. $[\![e]\!] \sqsubseteq \bigsqcap\{[\![p]\!]' \mid p \in \mathbf{CIP}[src(e), dst(e)]\}$, if the data-flow functional $[\![\ ]\!]'$ is monotonic,
2. $[\![e]\!] = \bigsqcap\{[\![p]\!]' \mid p \in \mathbf{CIP}[src(e), dst(e)]\}$, if the data-flow functional $[\![\ ]\!]'$ is distributive.

Hauptergebnisse 2(2): Sicherheit und Koinzidenz

Theorem [Interprocedural Safety] The $IMaxFP$-solution is a lower approximation of the $IMOP$-solution, i.e.,

$$\forall\, c_{\mathbf{s}} \in \mathcal{C}\ \forall\, n \in N.\ IMaxFP_{c_{\mathbf{s}}}(n) \sqsubseteq IMOP_{c_{\mathbf{s}}}(n)$$

if the data-flow functional $[\![\ ]\!]'$ is monotonic.

Theorem [Interprocedural Coincidence] The $IMaxFP$-solution coincides with the $IMOP$-solution, i.e.,

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

if the data-flow functional $[\![\ ]\!]'$ is distributive.

Kapitel 15.2 Allgemeiner Interprozeduraler Fall

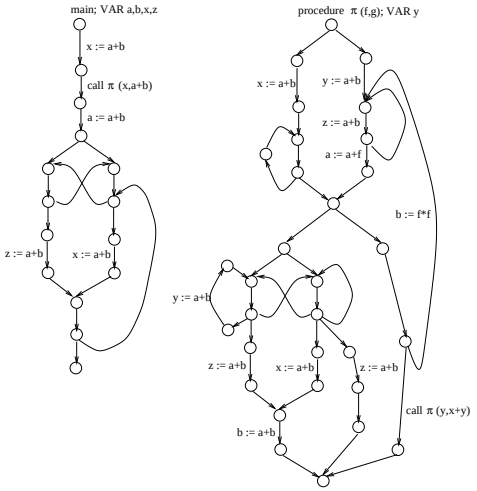
Interprozedurale Erweiterungen

Zunächst:

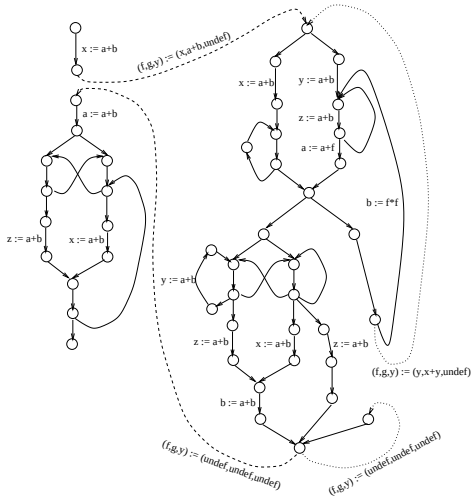
Hinzunahme von

- Wertparametern und lokalen Variablen

Interprozedurale Flussgraphsysteme

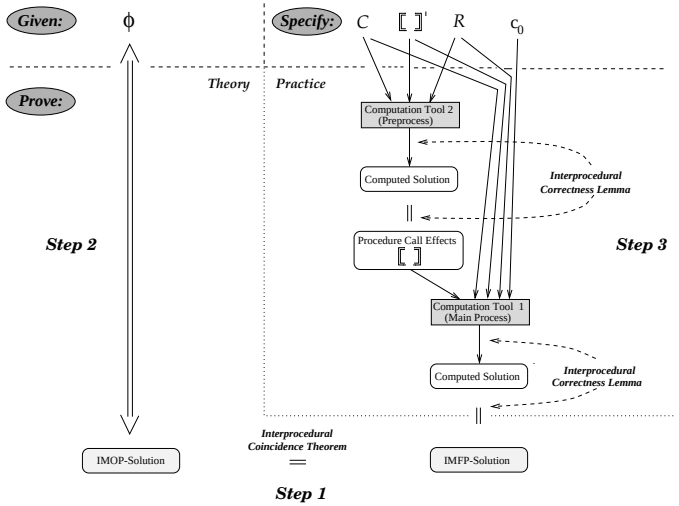


Interprozedurale Flussgraphen



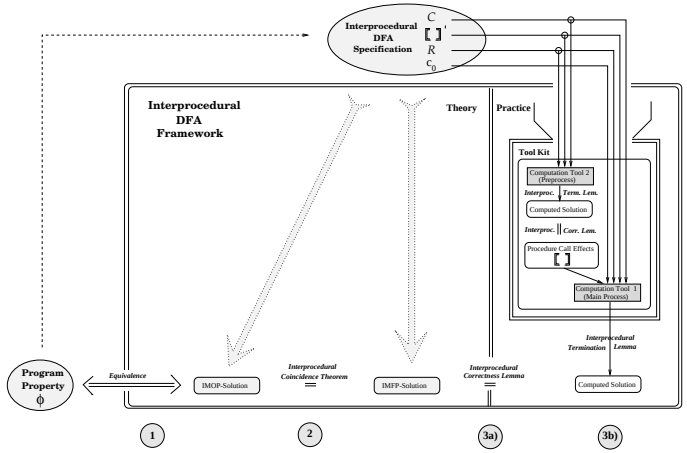
Interprozedurale Datenflussanalyse (1)

Der *interprozedurale* DFA-Rahmen im Überblick:



Interprozedurale Datenflussanalyse (2)

Der *interprozedurale* DFA-Rahmen in der Toolkit-Sicht:



Für Details

Siehe ausgeteilten Umdruck!

Weiter geht es mit Ausblick

...auf

- parallele,
- konditionale Datenflussanalyse

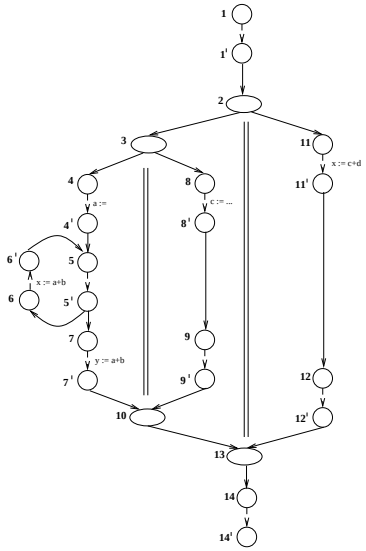
in...

- Rahmen- und
- Toolkit-Sicht.

Kapitel 16 Weitere Paradigmen

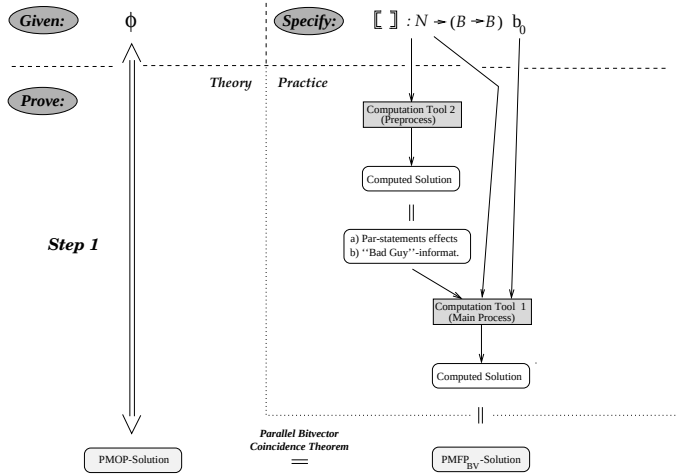
Kapitel 16.1 Parallele Datenflussanalyse

Parallele Flussgraphen



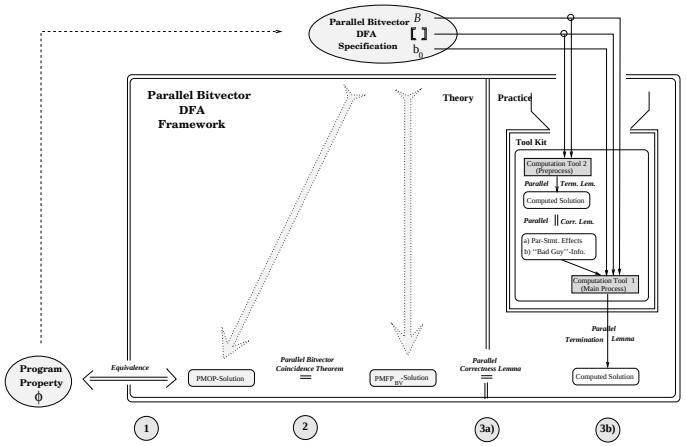
Parallele Datenflussanalyse (1)

Der *parallele* DFA-Rahmen im Überblick:



Parallele Datenflussanalyse (2)

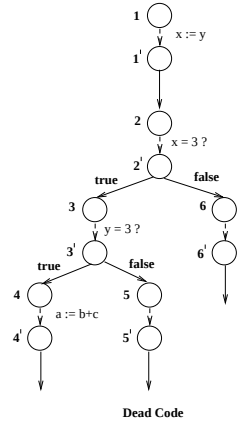
Der *parallele* DFA-Rahmen in der Toolkit-Sicht:



Kapitel 16.2 Konditionale Datenfluss-analyse

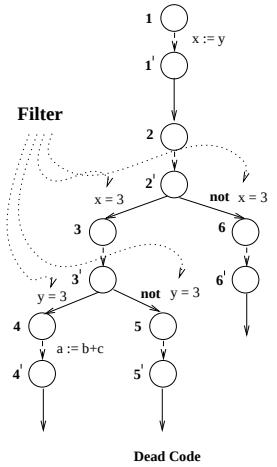
Konditionale Datenflussanalyse (1)

Ein Anwendungsbeispiel zur Motivation...



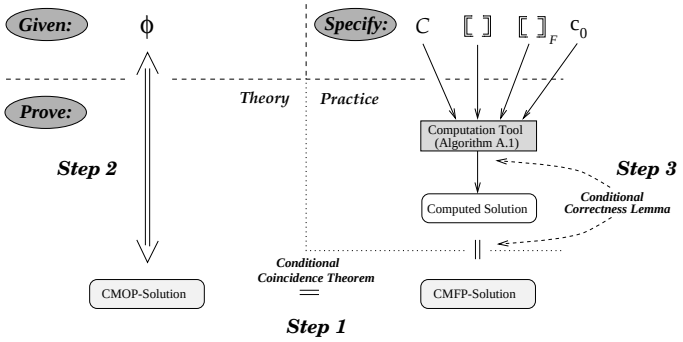
Konditionale Datenflussanalyse (2)

Technische Umsetzung: Mittels *Filterfunktionen*...



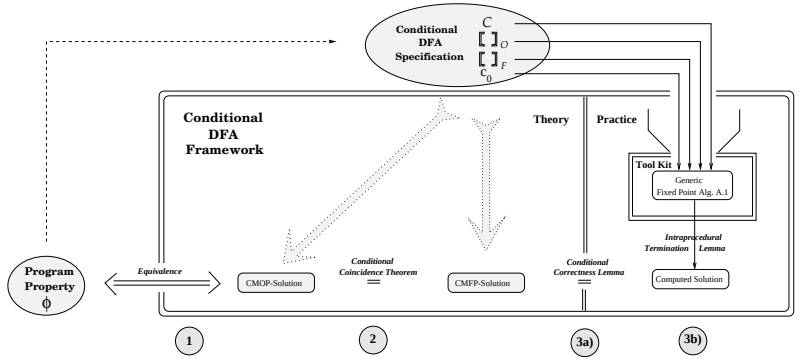
Konditionale Datenflussanalyse (3)

Der *konditionale* DFA-Rahmen im Überblick:



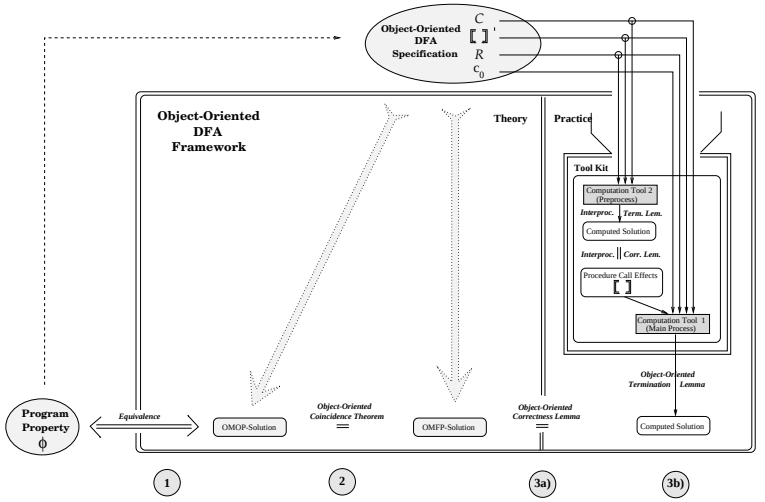
Konditionale Datenflussanalyse (4)

Der *konditionale* DFA-Rahmen in der Toolkit-Sicht:



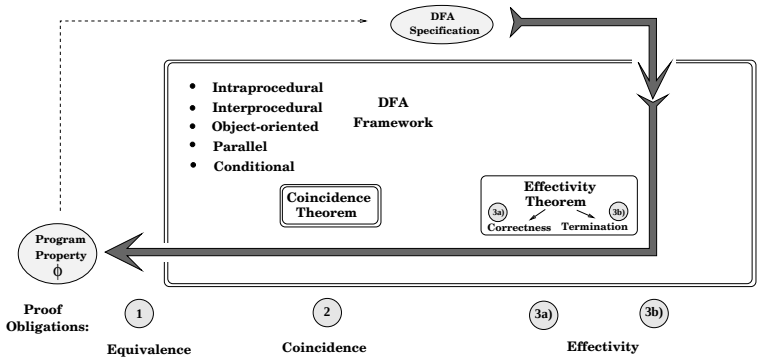
Kapitel 16.3 Objektorientierte Datenflussanalyse

Objektorientierte Datenflussanalyse



Kapitel 16.4 Fazit

Das paradigmenübergreifende generelle Muster



Für Details

Siehe ausgeteilte Umdrucke!

Kapitel 17 Abschließendes

Kapitel 17.1 Abstrakte Interpretation

Kapitel 17.2 Model-Checking

Für Details

Siehe ausgeteilte Umdrucke!