Analyse und Verifikation

(SS 2009, 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, FH Hörsaal 4 (Wiedner Hauptstr. 8, 1040 Wien)

Kapitel 1 Grundlagen

Grundlagen

Syntax und Semantik von Programmiersprachen...

- *Syntax*: Regelwerk zur Spezifikation wohlgeformter Programme
- *Semantik*: Regelwerk zur Spezifikation 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...

- 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.

Kapitel 1.1 Motivation

Motivation

...*formale* Semantik von Programmiersprachen einzuführen:

(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 Programmiersprache 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 Programmiersprache WHILE

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

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

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

Überblick über Syntax & Semantik (1)

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

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

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

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

  über

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

    (In der Folge werden wir für D oft die Menge der ganzen Zahlen $\mathbb{Z}$ betrachten.)

Überblick über Syntax & Semantik (2)

Zahldarstellungen

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

Arithmetische Ausdrücke

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

Boolesche Ausdrücke

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

Überblick über Syntax & Semantik (3)

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**

Überblick über Syntax & Semantik (4)

In der Folge werden wir im Detail betrachten...

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

Semantik arithmetischer & Boolescher Ausdrücke

Die Semantik von WHILE stützt sich ab auf die...

Semantik

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

$[\![.]\!]_A$: **Aexpr** $\to (\Sigma \to \mathbb{Z})$ 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, $minus$, $durch$: $\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ die übliche Addition, Multiplikation, Subtraktion und (ganzzahlige) Division auf den ganzen Zahlen $\mathbb{Z}$ bezeichnen.

$[\![.]\!]_N$: **Num** $\to \mathbb{Z}$ 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, **0**, **1**, **2**, ... bezeichnen *semantische* Entitäten, in diesem Falle ganze Zahlen.

Semantik Boolescher Ausdrücke (1)

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

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

- *Semantik Boolescher Ausdrücke:*

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

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

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

Kapitel 1.4 Syntaktische und semantische Substitution

Freie Variablen

...arithmetischer Ausdrücke:

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

...Boolescher Ausdrücke:

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

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*

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*
  $(A(1) \wedge (\forall n \in \mathbb{N}.\, A(n) \succ A(n+1))) \quad \succ \quad \forall n \in \mathbb{N}.\, A(n)$
- *verallgemeinerten Induktion*
  $((\forall n \in \mathbb{N}.\, (\forall m < n.\, A(m)) \succ A(n))) \quad \succ \quad \forall n \in \mathbb{N}.\, A(n)$
- *strukturellen Induktion*
  $((\forall s \in S. \forall s' \in \mathit{Komp}(s).\, A(s')) \succ A(s)) \quad \succ \quad \forall s \in S.\, A(s)$

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

Beispiel: Beweis von Lemma 1.4.1 (1)

...durch strukturelle Induktion

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

Induktionsanfang:

Fall 1: Sei $a \equiv n$, $n \in \textbf{Num}$.

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:

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

Induktionsschluss:

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

Dann erhalten wir:

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

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 Semantik...

Insbesondere unterscheiden wir den...

- *denotationellen*
- *operationellen*
- *axiomatischen*

Stil.

Semantikdefinitionsstile (2)

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

Kapitel 2 Operationelle Semantik von WHILE

Kapitel 2.1 Strukturell Operationelle 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, Elsevier, 1982.

...i.S.v. Gordon D. Plotkin.

- *Die SO-Semantik von WHILE* ist gegeben durch ein Funktional:

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

das in der Folge von uns zu definieren ist...

Dabei gilt:

- $\Sigma_\varepsilon =_{df} \Sigma \cup \{error\}$, wobei $error$ einen speziellen Fehlerzustand bezeichnet, $error \notin \Sigma$.

Strukturell operationelle Semantik

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 Formen σ oder $error$
    ...beschreiben das Resultat nach Ende der Berechnung, wobei Ende nach...
    - *regulärer* Terminierung: angezeigt durch gewöhnliche Zustände σ
    - *irregulärer* Terminierung: angezeigt durch $error$-behaftete Konfiguration
- Γ bezeichne die Menge aller Konfigurationen, $\gamma \in \Gamma$

SOS-Regeln von WHILE (1)

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

$[\text{abort}_{sos}]$ $\dfrac{}{\langle abort, \sigma \rangle \Rightarrow error}$

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

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

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

SOS-Regeln von WHILE (2)

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

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

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

Sprechweisen (1)

Wir unterscheiden

- Prämissenlose *Axiome* der Form

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

- Prämissenbehaftete *Regeln* der Form

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

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

Sprechweisen (2)

Im Fall der SO-Semantik von WHILE haben wir demnach

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

Berechnungsschritt, Berechnungsfolge

- Ein *Berechnungsschritt* ... ist von der Form

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

- 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 $i \in \mathbb{N}$.

Terminierende vs. divergierende Berechnungsfolgen

- Eine maximale (d.h. nicht mehr verlängerbare) Berechnungsfolge heißt
  - *regulär terminierend*, wenn sie endlich ist und die letzte Konfiguration aus Σ ist,
  - *irregulär terminierend*, wenn sie endlich ist und die letzte Konfiguration $error$-behaftet ist,
  - *divergierend*, falls sie unendlich ist.

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)

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

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

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

Beispiel (3)

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

Beispiel (4)

$\Rightarrow\ \langle \text{if } x <> 1$
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...

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

Beispiel (Detailbetrachtung) (6)

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

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

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...

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

Lemma 2.1.1

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

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!

Korollar 2.1.2 erlaubt uns jetzt festzulegen:

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

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

welches definiert wird durch:

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

Variante induktiver Beweisführung

Induktion über die Länge von Berechnungsfolgen:

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

Anwendung

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

Ein Beispiel 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

Natürliche Semantik (1)

...ebenfalls für das Beispiel von WHILE:

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

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

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

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

Natürliche Semantik (2)

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

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

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

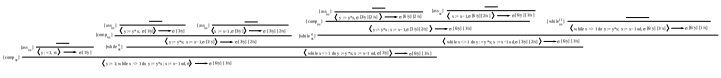
$$[\text{while}^{ff}_{ns}]\quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma\rangle \to \sigma} \qquad [\![\,b\,]\!]_B(\sigma) = \mathtt{ff}$$

Beispiel zur natürlichen Semantik (1)

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

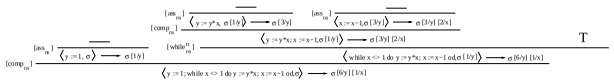
Dann gilt:

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

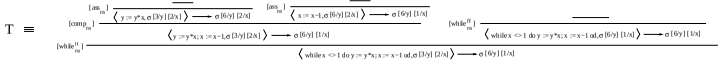


Beispiel zur natürlichen Semantik (2)

Das gleiche Beispiel in etwas gefälligerer Darstellung:



T ≡



Lemma 2.2.1

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

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

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

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

Korollar 2.2.2 erlaubt uns festzulegen:

- Die natürliche Semantik von WHILE ist gegeben durch das Funktional

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

  welches definiert wird durch:

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

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.

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' oder $error$ 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 nicht vollständig; das Restprogramm π' ist auf σ' anzusetzen
  - γ von der Form σ':
    Abarbeitung von π vollständig; π angesetzt auf σ terminiert in einem Schritt in σ'
  - γ von der Form $error$:
    Abarbeitung von π terminiert irregulär

Natürliche Semantik

Der Fokus liegt auf...

- Zusammenhang von *initialem* und *finalem* Zustand einer Berechnungsfolge

  Intuitive Bedeutung von...

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

  ...mit γ von der Form σ' oder $error$ ist: π angesetzt auf initialen Zustand σ terminiert schließlich im finalen Zustand σ' bzw. terminiert irregulär.

Kap. 3 Denotationelle Semantik von WHILE

...auch für das Beispiel von WHILE:

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

$[\![abort]\!]_{ds} = Error$

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

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

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

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

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

Denotationelle Semantik (2)

Es bezeichnen:

- $Id : \Sigma_\varepsilon \to \Sigma_\varepsilon$ die identische Zustandstransformation:

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

- $Error : \Sigma_\varepsilon \to \Sigma_\varepsilon$ die konstante Zustandstransformation mit:

$$\forall \sigma \in \Sigma_\varepsilon.\ Error(\sigma) =_{df} error$$

Denotationelle Semantik (3)

Zur Hilfsfunktion $cond$...

Funktionalität...

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

Definiert durch...

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

Zu den Argumenten und zum Resultat von $cond$...

- 1. Argument: Prädikat (in unserem Szenario total definiert; siehe Vorlesungsteil 1)
- 2.&3. Argument: Je eine partiell definierte Zustandstransformation
- Resultat: Wieder eine partiell definierte Zustandstransformation

Denotationelle Semantik (4)

Zur Hilfsfunktion FIX...

Funktionalität...

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

Definiert durch...

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

Daraus ergibt sich...

- FIX ist ein Funktional ("Zustandstransformationsfunktional")
- Die denotationelle Semantik der while-Schleife ist ein Fixpunkt des Funktionals F (und zwar der kleinste!)

Mehr Details zu FIX und Co. später!

Denotationelle Semantik (5)

- *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).

Denotationelle Semantik (6)

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 (7)

Lemma 3.1
Für alle $\pi \in$ **Prg** ist durch die Gleichungen von Folie "Denotationelle Semantik (1)" eine (partielle) Funktion $[\![\pi]\!]_{ds}$ definiert, die denotationelle Semantik von π.

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:

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

definiert durch

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

Kap. 3.1 Fixpunktfunktional

WHILE – Denotationelle Semantik (1)

- **Prg** ...bezeichne die Menge aller Programme der Sprache **WHILE**

Denotationelle Semantik

$$[\![\ ]\!]_{ds} : \textbf{Prg} \to (\Sigma \to \Sigma_\varepsilon)$$

Somit...

- Die *denotationelle Semantik* eines **WHILE**-Programms ist eine (partiell definierte) *Zustandstransformation*, wobei die Menge der *Zustände* gegeben ist durch

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

Beachte...

- Auch die operationelle (die strukturell operationelle wie auch die natürliche) Semantik eines **WHILE**-Programms ist eine (partiell definierte) *Zustandstransformation* auf Σ, nicht aber die axiomatische Semantik.

WHILE – Denotationelle Semantik (2)

Erinnerung:

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

$[\![\ abort\ ]\!]_{ds} = Error$

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

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

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

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

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

WHILE – Denotationelle Semantik (3)

Noch offen...

- Die Bedeutung von...
  - $cond$ und
  - $FIX\ F$

Diese Bedeutung wollen wir in der Folge aufklären...

Zur Bedeutung von cond

Hilfsfunktion $cond$...

Funktionalität...

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

Definiert durch...

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

Zu den Argumenten und zum Resultat von $cond$...

- 1. Argument: Prädikat (in unserem Szenario total definiert; siehe Vorlesungsteil 1)
- 2.&3. Argument: Je eine partiell definierte Zustandstransformation
- Resultat: Wieder eine partiell definierte Zustandstransformation

Damit erhalten wir

...für die Bedeutung der Fallunterscheidung

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

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

$$= \begin{cases} \sigma' & \text{falls } ([\![\ b\ ]\!]_{\mathcal{B}}\ \sigma = \mathtt{tt}\ \wedge\ [\![\ \pi_1\ ]\!]_{\mathtt{ds}}\ \sigma = \sigma') \\ & \quad \vee\ ([\![\ b\ ]\!]_{\mathcal{B}}\ \sigma = \mathtt{ff}\ \wedge\ [\![\ \pi_2\ ]\!]_{\mathtt{ds}}\ \sigma = \sigma') \\ error & \text{falls } ([\![\ b\ ]\!]_{\mathcal{B}}\ \sigma = \mathtt{tt}\ \wedge\ [\![\ \pi_1\ ]\!]_{\mathtt{ds}}\ \sigma = \mathtt{error}) \\ & \quad \vee\ ([\![\ b\ ]\!]_{\mathcal{B}}\ \sigma = \mathtt{ff}\ \wedge\ [\![\ \pi_2\ ]\!]_{\mathtt{ds}}\ \sigma = \mathtt{error}) \\ undef & \text{falls } ([\![\ b\ ]\!]_{\mathcal{B}}\ \sigma = \mathtt{tt}\ \wedge\ [\![\ \pi_1\ ]\!]_{\mathtt{ds}}\ \sigma = \mathtt{undef}) \\ & \quad \vee\ ([\![\ b\ ]\!]_{\mathcal{B}}\ \sigma = \mathtt{ff}\ \wedge\ [\![\ \pi_2\ ]\!]_{\mathtt{ds}}\ \sigma = \mathtt{undef}) \end{cases}$$

Erinnerung:

- $[\![\ b\ ]\!]_{\mathcal{B}}$ ist in unserem Szenario total definiert; $[\![\ b\ ]\!]_{\mathcal{B}}\ \sigma$ ist daher stets von $undef$ verschieden.

Zur Bedeutung von FIX F

Funktionalität...

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

Definiert durch...

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

Daraus ergibt sich...

- FIX ist ein Funktional ("Zustandstransformationsfunktional")
- Die denotationelle Semantik der while-Schleife ist ein Fixpunkt des Funktionals F (und zwar der kleinste!)

Dazu folgende Beobachtung...

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

Daraus folgt...

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

Und daraus schließlich...

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

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

Oder anders ausgedrückt, es muss gelten:

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

...was uns wie gewünscht zu einer *kompositionellen* Definition von $[\![\,\textbf{while}\ b\ \textbf{do}\ \pi\ \textbf{od}\,]\!]_{ds}$ und damit von $[\![\ \ ]\!]_{ds}$ insgesamt führen wird.

Erforderlich...

- Einige Resultate aus der *Fixpunkttheorie*

Zu tun...

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

Anschließend bleibt nachzuholen...

- Der mathematische Hintergrund (Ordnungen, CPOs, Stetigkeit von Funktionen) und die benötigten Resultate (Fixpunktsatz)

Folgende drei Argumente...

...werden dafür entscheidend sein

1. $[\Sigma \to \Sigma]$ 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

$$[\![\ \ ]\!]_{ds} : \textbf{Prg} \to (\Sigma \to \Sigma_{\varepsilon})$$

Ordnung auf Zustandstransformationen

Bezeichne...

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

Wir definieren...

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

Lemma 3.1.1

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

Ordnung auf Zustandstransformationen

Sogar...

Lemma 3.1.2
Das Paar $([\Sigma \to \Sigma], \sqsubseteq)$ ist eine vollständige partielle Ordnung (CPO) mit kleinstem Element $\bot$.

Weiter 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 $f : M \to N$ 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 $f : M \to N$ 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 $g_0 \in [\Sigma \to \Sigma]$, sei $p \in [\Sigma \to \mathbb{B}]$ und sei F definiert durch

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

Dann gilt: F ist stetig.

Zur 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 $g_0 \in [\Sigma \to \Sigma]$ und sei F definiert durch

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

Dann gilt: F ist stetig.

Lemma 3.1.5
Die Gleichungen zur Festlegung der denotationellen Semantik von WHILE (vgl. Folie 15 von heute) definieren eine totale Funktion

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

...sind wir durch! Wir können beweisen:

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

ist wohldefiniert!

Aus...

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

...ergibt sich wie gewünscht:

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

...ist wohldefiniert!

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 allerdings weniger wichtig)

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

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: $Q \sqsubseteq q$ ($q \sqsubseteq Q$)

- 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.

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

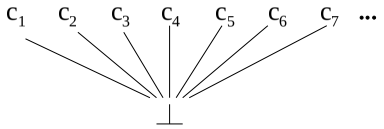
- *endlich* gdw. die Menge $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$ keine unendliche Kette enthält
- *endlich relativ zu* $r \in P$ gdw. die Menge $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$ keine unendliche Kette enthält

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

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

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

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 3.2.2 (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 3.2.2 (5)

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

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

Kapitel 4.1 Partielle und totale 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

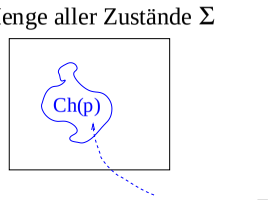
...an einige Sprechweisen:

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 zur Konfiguration $undef$ führt.
- Ein Programm π heißt *divergent* gdw. π terminiert für keinen Anfangszustand regulär.

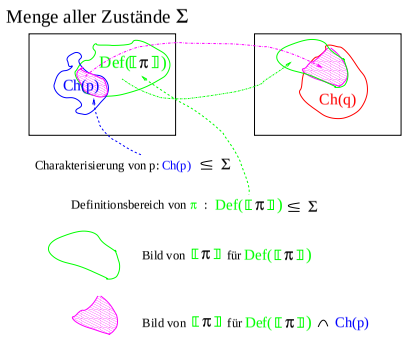
Veranschaulichung (1)

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



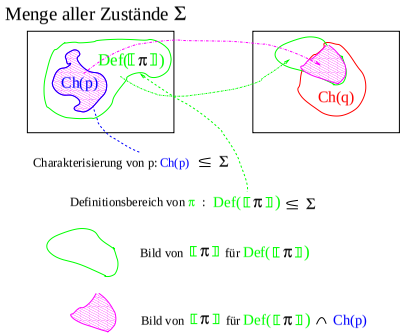
Veranschaulichung (2)

...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

dingungen (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.

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 denen auf Folie 13 von diesem Vorlesungsteil betrachteten zusammen?

Kapitel 4.2 Beweiskalkül für partielle Korrektheit

Hoare-Kalkül HK_{PK} für partielle Korrektheit

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

[abort] $\dfrac{}{\{p\}\ abort\ \{q\}}$

[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

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

  mag natürlich erscheinen, ist aber beweistechnisch unangenehm durch das Mitschleppen quantifizierter Formeln.
- *Beachte*: Folgende scheinbar naheliegende quantorfreie Realisierung der Vorwärtszuweisungsregel ist nicht korrekt:

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

  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:

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

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).

  $\rightsquigarrow$ *Terminationsordnung!*

Zur Vollständigkeit

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

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

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

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

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

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

Zum Überlegen: Warum fehlt eine Regel für abort?

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

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:

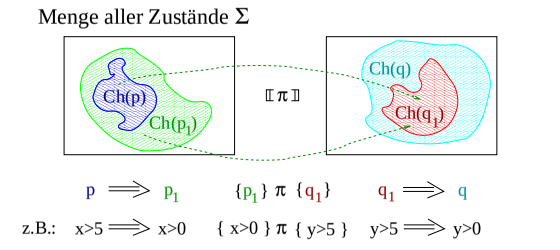
Die Konsequenzregel

- ...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:

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

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).
  $\rightsquigarrow$ *Terminationsordnung*!

- 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.)

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 (und die Regel für *abort*) identisch...

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

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.

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

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 $(\mathbf{N}, <)$.

Nachtrag zur totalen Korrektheit (2)

Beweistechnische Anmerkung:

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

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

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 [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:

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

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

$$\begin{array}{c}[a > 0]\\ x := a;\ y := 1;\ \text{while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}\\ [y = a!]\end{array}$$

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

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?

  $\leadsto$ *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

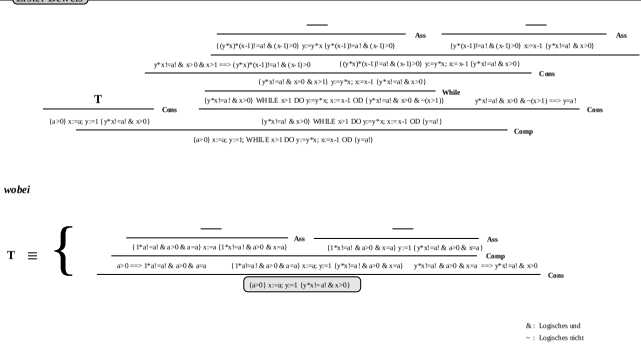
$$\{a > 0\}$$
$$x := a;\ y := 1;\ \text{while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}$$
$$\{y = a!\}$$

und

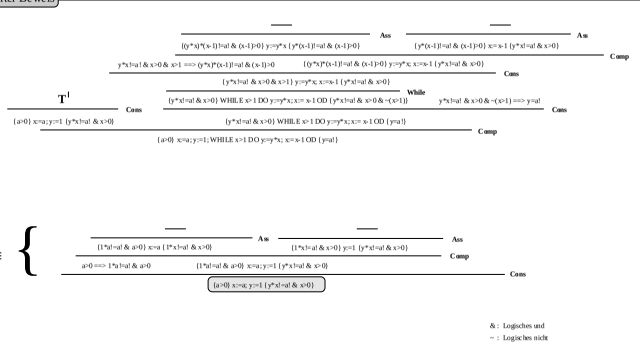
$$\{x \geq 0 \wedge y > 0\}$$
$$q := 0; r := x; \text{while } r \geq y \text{ do } q := q + 1; r := r - y \text{ od}$$
$$\{x = q * y + r \wedge 0 \leq r < y\}$$

gültig sind im Sinne partieller Korrektheit.

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

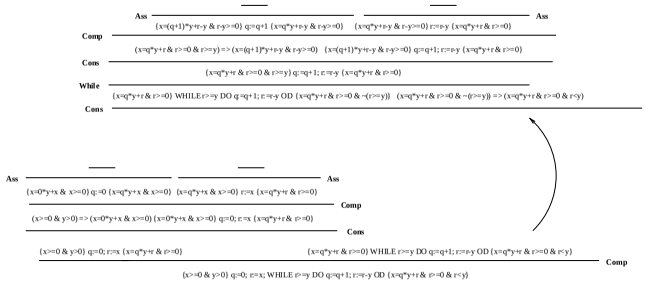


wobei



wobei

Bew. partieller Korrektheit: Division



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:

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

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

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:

$$\{a > 0\}$$

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

Einmalige Anwendung der [ass]-Regel liefert:

$$\{a > 0\}$$

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

Abermalige Anwendung der [ass]-Regel liefert:

$$\{a > 0\}$$

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

Lin. Beweisskizze f. Fakultätsbsp. (15)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

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

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*

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

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

Damit haben wir insbesondere:

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

Bemerkung

Der Beweis wird wieder in Form einer linearen Beweisskizze präsentiert...

Bew. totaler Korrektheit: Fakultät (1)

Schritt 2

Behandlung des Rumpfs der while-Schleife...

Der Nachweis der Gültigkeit von

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

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

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

Bew. totaler Korrektheit: Fakultät (2)

Behandlung des Rumpfs der while-Schleife im Detail:

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

Bew. totaler Korrektheit: Fakultät (3)

Wegen *Rückwärts*zuweisungsregel wird der Rumpf der while-Schleife von hinten nach vorne bearbeitet:

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

Bew. totaler Korrektheit: Fakultät (4)

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

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

...wobei noch eine "Beweislücke" verbleibt!

Bew. totaler Korrektheit: Fakultät (5)

Schluss der "Beweislücke" in der zugrundeliegenden Theorie:

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

Bew. totaler Korrektheit: Fakultät (6)

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

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

Schritt 3

Zur gewünschten Nachbedingung verbleibt offenbar ebenfalls eine Beweislücke:

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

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

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:

$$[a > 0]$$

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

Bew. totaler Korrektheit: Fakultät (11)

Einmalige Anwendung der [ass]-Regel liefert:

$$[a > 0]$$

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

Bew. totaler Korrektheit: Fakultät (12)

Abermalige Anwendung der [ass]-Regel liefert:

$$[a > 0]$$

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

Bew. totaler Korrektheit: Fakultät (13)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

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

Überblick (14)

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

Bew. totaler Korrektheit: Fakultät (15)

Damit haben wir wie gewünscht insgesamt gezeigt:

Die Hoaresche Zusicherung

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

ist gültig im Sinne totaler Korrektheit.

Kapitel 4.8 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 Ildico Kovacs 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 Ildico Kovacs 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.

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

Bemerkung: Verfeinerungen sind offenbar möglich.

...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

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

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

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

Die Semantikfunktionen

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

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

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

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

Ausführungszeitaspekt (1)

...für das Beispiel von WHILE:

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

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

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

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

Ausführungszeitaspekt (2)

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

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

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

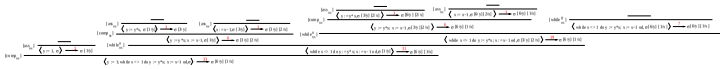
$$[\text{while}^{ff}_{tns}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od},\sigma\rangle \rightarrow^{[\![b]\!]_{TB}+3} \sigma} \qquad [\![b]\!]_B(\sigma) = \texttt{ff}$$

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

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

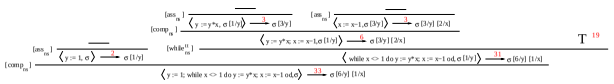
Dann gilt:

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

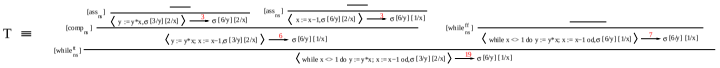


Beispiel zur nat. "Zeit"-Semantik (2)

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)

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

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

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

wobei u frische logische Variable ist

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

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

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

Axiomatische Semantik zum Ausführungszeitaspekt (2)

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

wobei $p(z+1) \Rightarrow b \wedge e \geq e_1 + e',\ p(0) \Rightarrow \neg b \wedge 1 \leq e$
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*

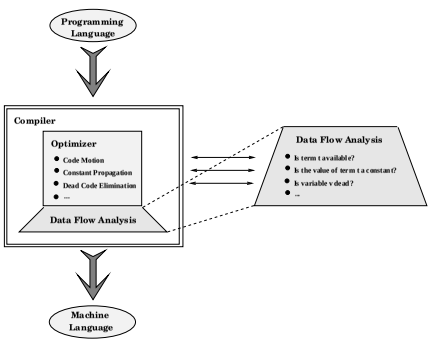
Typische Fragen sind...

- 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.

Ingredienzien *intraprozeduraler* Datenflussanalyse:

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

Ausblick

Hauptresultate:

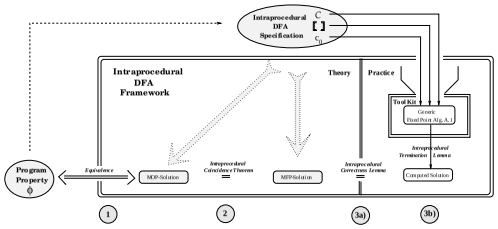
- *Sicherheits-* (*Korrektheits-*) Theorem
- *Koinzidenz-* (*Vollständigkeits-*) Theorem

Sowie:

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

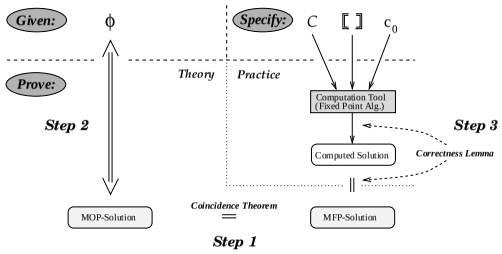
Ausblick: Intraprozedurale Datenflussanalyse (1)

...die (detaillierte) Werkzeugkistensicht:



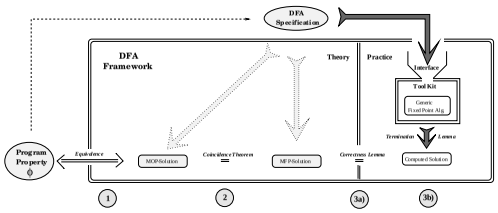
Ausblick: Intraprozedurale Datenflussanalyse (2)

...bei genauerem Hineinsehen:



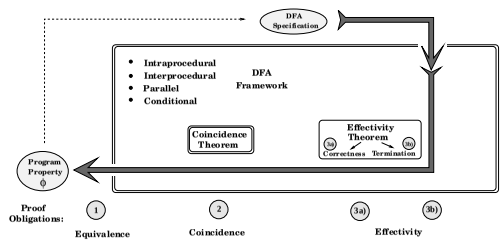
Ausblick: DFA-Frameworks / DFA-Toolkits (1)

...aus größerer Ferne und Konzentration auf das Wesentliche:



Ausblick: DFA-Frameworks / DFA-Toolkits (2)

...das generelle Muster, die Werkzeugkistensicht:



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

Zurück zum Anfang: Zur Programmanalyse

...speziell *Datenflussanalyse*

Üblich ist...

- die Repräsentation von Programmen durch (nichtdeterministische) *Flussgraphen*

Flussgraph

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

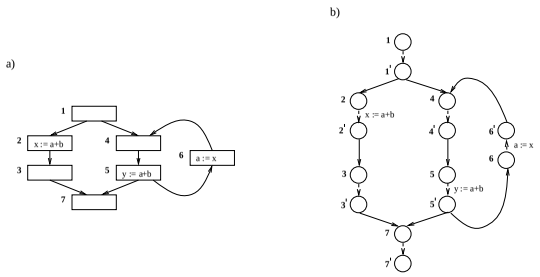
- Knotenmenge (engl. *Nodes*) N
- Kantenmenge (engl. *Edges*) $E \subseteq N \times N$
- ausgezeichnetem Startknoten s ohne Vorgänger und
- ausgezeichnetem Endknoten e ohne Nachfolger

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

$\rightsquigarrow$ Darstellungsvarianten: Knoten- vs. kantenbenannte Flussgraphen

Veranschaulichung

Knoten- vs. kantenbenannte Flussgraphen
(hier mit Einzelanweisungsbenennung)



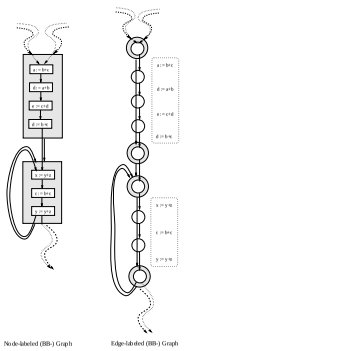
Flussgraphen

Darstellungsvarianten...

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

In der Folge werden wir bevorzugt kantenbenannte SI-Graphen betrachten.

Knoten- vs. kantenbenannte Flussgraphen



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)
  $\leadsto$ *spezifizierende* Lösung
- Maximaler Fixpunktansatz ($MaxFP$)
  $\leadsto$ *berechenbare* Lösung

Kapitel 6.3 MOP -Ansatz

Der MOP -Ansatz

Zentral: Ausdehnung der lokalen abstrakten Semantik auf Pfade

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

Die MOP -Lösung

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

Kapitel 6.4 $MaxFP$ -Ansatz

Der $MaxFP$ -Ansatz

Zentral: Das $MaxFP$ -Gleichungssystem:

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

Die $MaxFP$ -Lösung

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

wobei $\mathbf{inf}^{\,*}_{c_{\mathbf{s}}}$ die größte Lösung des $MaxFP$ -Gleichungssystems bezeichnet.

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

Ausgabe: Unter den Voraussetzungen des Effektivitätstheorems (später!) die $MaxFP$-solution. Abhängig von den Eigenschaften des Datenflussanalysefunktionals gilt dann:

(1) $[\![\ ]\!]$ ist *distributiv*: Variable *inf* enthält für jeden Knoten die stärkste Nachbedingung bezüglich der Anfangsinformation $c_{\mathbf{s}}$.

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

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

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

(Hauptprozess: Iterative Fixpunktberechnung)
WHILE *workset* $\neq \emptyset$ DO
  CHOOSE $m \in$ *workset*;
  workset := *workset*\{ 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.

Kapitel 6.5 Koinzidenz- und Sicherheitstheorem

Hauptresultate

Zusammenhang von...

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

Korrektheit: Sicherheitstheorem

Theorem [Sicherheit (Safety)]

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

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

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

Vollständigkeit (und Korrektheit): Koinzidenztheorem

Theorem [Koinzidenz (Coincidence)]

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

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

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

Terminierung: Effektivitätstheorem

Theorem [Effektivität]

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

Nachzutragende Definitionen

...sind:

- Absteigende (aufsteigende) Kettenbedingung
- Monotonie und Distributivität von Datenflussanalysefunktionalen

Definition [Ab-/aufsteigende Kettenbedingung]

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

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

Definition [Monotonie, Distributivität, Additivität]
Sei $\vec{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ 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)

Zur Erinnerung: Oft nützlich

...ist folgende äquivalente Charakterisierung der Monotonie:

Lemma

Sei $\vec{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ 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 6.6 Beispiele: Verfügbare Ausdrücke und Einfache Konstanten

Beispiel: 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

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

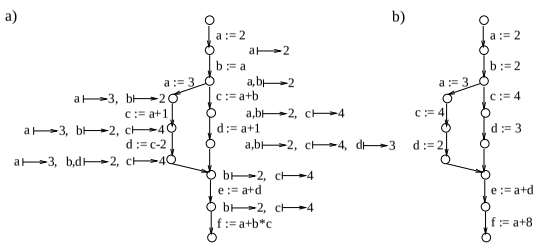
wobei

$$\hat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, \leq, \mathtt{ff}, \mathtt{tt})$$

den Verband der Wahrheitswerte bezeichnet mit $\mathtt{ff} \leq \mathtt{tt}$ und dem logischen "und" und "oder" als Schnitt- bzw. Vereinigungsoperation $\sqcap$ and $\sqcup$.

Beispiel: Einfache Konstanten

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



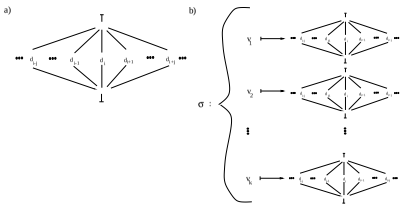
Abstrakte Semantik für einfache Konstanten

- **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

$$\forall\, e \in E.\ [\![e]\!]_{sc} =_{df} \theta_e$$

Konstanten

Der "kanonische" Verband für Konstantenausbreitung/-faltung:



Die Semantik von Termen

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

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

die induktiv definiert ist durch:

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

Nachzutragende Begriffe und 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:

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

Kapitel 6.7 Pragmatik: Basisblöcke vs. Einzelanweisungen

In der Folge

...zum Einfluss der Repräsentation:

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

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*).

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:

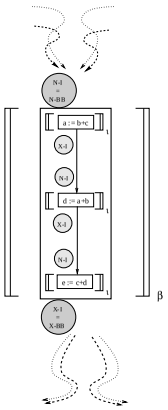
Die $MaxFP$-Lösung:

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

wobei $\mathtt{inf}^*_{c_s}$ 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)

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

mit

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

MOP -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

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

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

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

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

mit

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

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

$MaxFP$ -Ansatz (Basisblöcke) (4)

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

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

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

Verfügbarkeit von Ausdrücken (1)

...für knotenbenannte BB-Graphen:

Phase I: Die Basisblockebene

Lokale Prädikate: (assoziert 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:

$$\text{BB-N-AVAIL}_\beta = \begin{cases} \mathtt{ff} & \text{falls } \beta = \mathbf{s} \\ \prod_{\tilde{\beta} \in pred(\beta)} \text{BB-X-AVAIL}_{\tilde{\beta}} & \text{sonst} \end{cases}$$

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

Phase II: Die Anweisungsebene

Lokale Prädikate: (assoziiert mit EA-Knoten)

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

Das Gleichungssystem von Phase II:

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

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

Verfügbarkeit von Ausdrücken (4)

...für knotenbenannte EA-Graphen:

Lokale Prädikate: (assoziiert mit Knoten)

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

Das Gleichungssystem:

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

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

Verfügbarkeit von Ausdrücken (5)

...für kantenbenannte EA-Graphen:

Lokale Prädikate: (assoziiert mit EA-Kanten)

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

Das Gleichungssystem:

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

Weitere Beispiele

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

...für die Varianten *knotenbenannte Basisblockgraphen* und *kantenbenannte Einzelanweisungsgraphen*.

Konstantenfaltung: Einfache Konstanten

Zunächst zwei Hilfsfunktionen:

- Die Rückwärtssubstitution
- Die Zustandstransformation(sfunktion)

Rückwärtssubstitution & 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 (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.

...für kantenbenannte Einzelanweisungsgraphen:

Bemerkung: Falscher CP-Macro!!!!!

- $CalledProc_n \in \Sigma$
- $\sigma_0 \in \Sigma$ Anfangszusicherung

Das Gleichungssystem:

$\forall v \in \mathbf{V}.\ CalledProc_n =$

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

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:

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

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

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:

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

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

Geistervariablenelimination (1)

...für kantenbenannte Einzelanweisungsgraphen:

Lokale Prädikate: (assoziiert mit Einzelanweisungskanten)

- USED$_\varepsilon(v)$: Anweisung ι von Kante ε benutzt v.
- 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:

$$\text{FAINT}_n(v) =$$

$$\prod_{m \in succ(n)} \overline{\textit{Rel-Used}_{(n,m)}(v)} \cdot$$

$$(\text{FAINT}_m(v) \ + \ \text{MOD}_{(n,m)}(v)) \cdot$$

$$(\text{FAINT}_m(LhsVar_{(n,m)}) + \overline{\textit{Ass-Used}_{(n,m)}(v)})$$

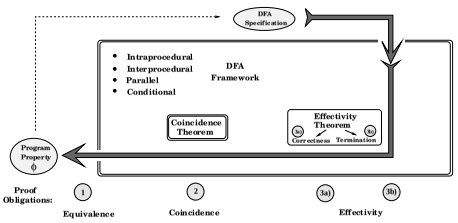
Geistervariablenelimination (3)

...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.

Für uns reicht...

...die allgemeine Werkzeugkistensicht und das Wissen, dass je nach Repräsentationsvariante unterschiedlich aufwändige Spezifikations-, Implementierungs- und Beweisverpflichtungen entstehen.



Kapitel 7 Optimierung

Kapitel 7.1 Motivation

Optimale Programmoptimierung

Zum Aufwärmen...

- Einige einfache Beispiele (siehe Tafel)

Anschließend zum echten Einstieg...

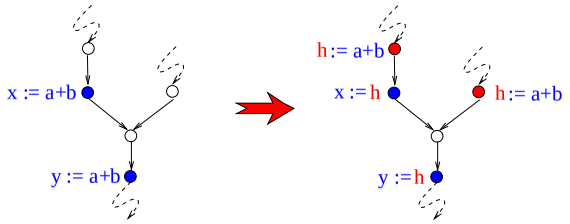
- *Partielle Redundanzelimination* (*PRE*) alias *Code Motion* (*CM*)

Kap. 7.2 Einfache Konstanten

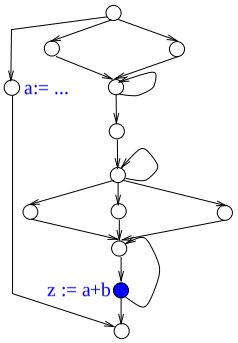
Kap. 7.3 Geistervariablenelimination

Kap. 7.4 Partielle Redundanzeliminati-on

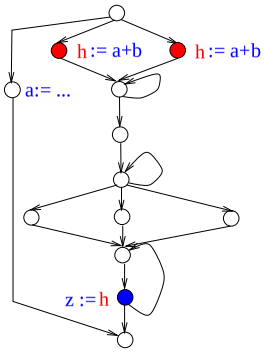
...im Kern um die Vermeidung von Mehrfachberechnungen von Werten



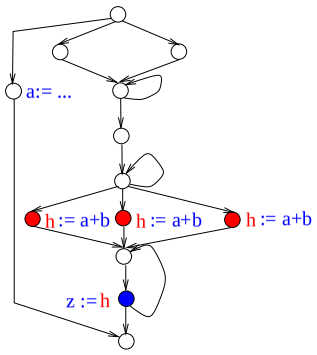
PRE – besonders wirksam im Zshg. mit Schleifen



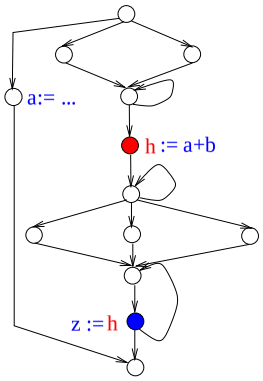
Ein redundanzfreies Programm



Aber welches soll es sein? Dieses? Das vorige?



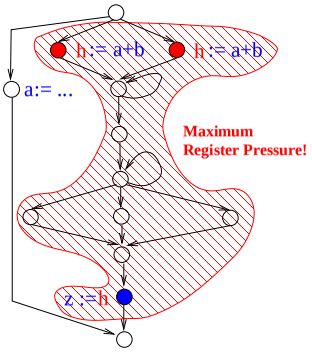
Oder vielleicht dieses?

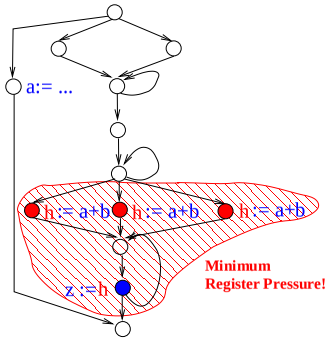


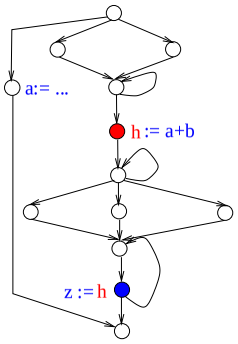
Auf die (Optimierungs-) Ziele kommt es an!

Die erste Transformation

...redundanzfrei, aber maximaler Registerdruck







Auf die (Optimierungs-) Ziele kommt es an!

In unseren Beispielen...

- *Performanz*: Vermeidung von Mehrfachberechnungen
  $\leadsto$ Berechnungsqualität/-optimalität
- *Registerdruck*: Vermeidung unnötigen Schiebens
  $\leadsto$ Lebenszeitqualität/-optimalität
- *Platz*: Vermeidung von Codereplikation
  $\leadsto$ Platzqualität/-optimalität

Zum Nachdenken

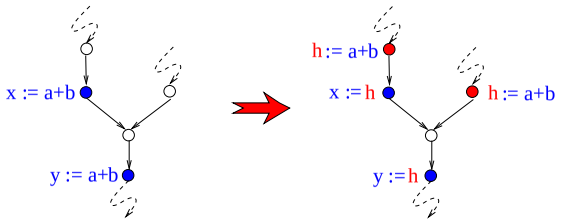
Sei P ein Programm mit Vorkommen partiell redundanter Berechnungen.

Ist es immer möglich, P so in ein Programm P' zu transformieren, dass P und P' bedeutungsgleich sind, P' aber frei von partiell redundanten Berechnungen ist?

Kap. 7.4.1 Nützliche Begriffe und Bezeichnungen

In Medias Res – PRE

Ziel: ...die Vermeidung von Mehrfachberechnungen von Werten



Bezeichnungen (1)

Sei $G = (N, E, \mathbf{s}, \mathbf{e})$ ein Flussgraph. Dann bezeichnen:

- $pred(n) =_{df} \{m \mid (m, n) \in E\}$: Menge aller *Vorgänger*
- $succ(n) =_{df} \{m \mid (n, m) \in E\}$: Menge aller *Nachfolger*
- $source(e)$, $dest(e)$: *Anfangs-* und *Endknoten* einer Kante
- *Endlicher Pfad*: Kantenfolge $(e_1, \ldots, e_k)$ mit $dest(e_i) = source(e_{i+1})$ für alle $1 \leq i < k$
- Statt Kantenfolgen betrachten wir entsprechend auch Knotenfolgen als Pfade, so zweckmäßig.

Bezeichnungen (2)

- $p = \langle e_1, \ldots, e_k \rangle$ *Pfad von m nach n*, falls $source(e_1) = m$ und $dest(e_k) = n$
- $\mathbf{P}[m, n]$: Menge aller Pfade von m nach n
- λ_p: *Länge* von p, d.h. die Anzahl der Kanten von p
- ε: Pfad der Länge 0
- $N_J \subseteq N$: Menge der *Join*-Knoten, d.h. Menge der Knoten mit mehr als einem Vorgänger
- $N_B \subseteq N$: Menge der *Branch*-Knoten, d.h. Menge der Knoten mit mehr als einem Nachfolger

Vereinbarung

Ohne Beschränkung der Allgemeinheit...

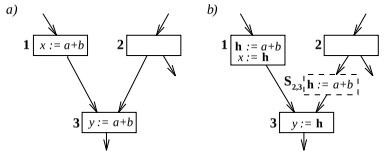
- Jeder Knoten in einem Flussgraphen liegt auf einem Pfad von s nach e

  Intuition: Es gibt keine unerreichbaren Teile in einem Flussgraphen.

...eine generell übliche Vereinbarung für Analyse und Optimierung!

Eine Kante heißt *kritisch*, wenn sie von einem branch- zu einem join-Knoten führt.

Zur Illustration: ...mit Knoten $\mathbf{S}_{2,3}$ als *künstlichem* (*synthetic*) *Knoten*, der die kritische Kante von Knoten **2** nach **3** spaltet.



Verfahrensspezifische Vereinbarung

Ohne Beschränkung der Allgemeinheit...

In der Folge betrachten wir Flussgraphen...

- in Form knotenbenannter EA-Graphen,
- bei denen alle Kanten, die in einem join-Knoten enden, durch Einfügen eines sog. *künstlichen* Knotens aufgespalten sind,

...eine PRE-spezifische Vereinbarung.

Hintergrund

...dieser Vereinbarung:

- Der PRE-Prozess vereinfacht sich dadurch.

  $\leadsto$ *Berechnungsoptimale* Ergebnisse können bereits erzielt werden, wenn erforderliche Hilfsvariableninitialisierungen einheitlich an Knotenanfängen erfolgen.

Bemerkung

Berechnungsoptimale Ergebnisse sind auch möglich, wenn ausschließlich kritische Kanten gespalten werden.

Dann aber muss ein PRE-Algorithmus in der Lage sein, sowohl N- als auch X-Initialisierungen (an Knoten) durchführen zu können.

Prinzipiell ist das kein Problem; mit vorstehender Vereinbarung ist die Präsentation des PRE-Algorithmus aber noch einfacher.

Arbeitsplan

In der Folge werden wir definieren...

- Die Menge der PRE-Transformationen
- Die Menge der *zulässigen* PRE-Transformationen
- Die Menge der *berechnungsoptimalen* PRE-Transformationen
- Die BCM-Transformation als spezielle berechnungsoptimale PRE-Transformation

Die Menge der PRE-Transformationen

Generelles (Transformations-) Muster für einen Term t...

- Deklariere eine neue Hilfvariable h für t in G
- Füge an einigen Knoten von G die Anweisung $h := t$ ein
- Ersetze einige der originalen Vorkommen von t in G durch h

Bem.: t wird oft auch als *Kandidatenausdruck* bezeichnet.

Beobachtung

Zwei (auf Knoten definierte) Prädikate

- $Insert_{CM}$
- $Repl_{CM}$

sind ausreichend, eine PRE- (bzw. CM-) Transformation vollständig zu beschreiben (beachte: die Deklaration der Hilfsvariablen h ist für jede CM-Transformation identisch und braucht deshalb nicht gesondert betrachtet zu werden).

CM-Transformationen

...bezeichne $\mathcal{CM}$ die Menge aller CM-Transformationen (für den Kandidatenausdruck t).

Beobachtung

Offenbar ist nicht jede Transformation in $\mathcal{CM}$ bedeutungserhaltend und damit akzeptabel.

Das führt uns auf den Begriff der *zulässigen* CM-Transformationen...

Zulässige CM-Transformationen

Sei $CM \in \mathcal{CM}$.

CM heißt *zulässig*, wenn CM *sicher* und *korrekt* ist.

Intuition:

- *Sicher*: ...es gibt keinen Pfad, auf dem durch Einfügen einer Initialisierung ein neuer Wert berechnet wird.
- *Korrekt*: ...die Hilfsvariable ist an jeder Benutzungsstelle "richtig" initialisiert, d.h. sie enthält denselben Wert, den eine Neuberechnung von t an dieser Stelle liefert.

Zur Formalisierung

...sind folgende (lokale) Prädikate erforderlich.

- *Comp*(n): n enthält ein Vorkommen des Kandidatenausdrucks t.
- *Transp*(n): n ist transparent für t, d.h., n weist keinem Operanden von t einen (neuen) Wert zu.

Ebenfalls nützlich:

- $\mathit{Comp}_{CM}(n) =_{df} \mathit{Insert}_{CM}(n) \vee \mathit{Comp}(n) \wedge \neg \mathit{Repl}_{CM}(n)$: Programmpunkte, an denen nach Anwendung von CM der Ausdruck t berechnet wird.

Globalisierung von Prädikaten auf Pfade

Sei p ein Pfad und bezeichne p_i den i-ten Knoten von p.

Mit diesen Bezeichnungen treffen wir die folgende Vereinbarung:

- $\mathit{Predicate}^{\forall}(p) \iff \forall\, 1 \leq i \leq \lambda_p.\ \mathit{Predicate}(p_i)$
- $\mathit{Predicate}^{\exists}(p) \iff \exists\, 1 \leq i \leq \lambda_p.\ \mathit{Predicate}(p_i)$

Sicherheit und Korrektheit

Definition [Sicherheit und Korrektheit]
Sei $n \in N$. Wir definieren:

1. $\mathit{Safe}(n) \iff_{df}$
   $\forall \langle n_1, \ldots, n_k \rangle \in \mathbf{P}[s, e]\ \forall i.\ (n_i = n) \Rightarrow$
   $i)\ \exists j < i.\ \mathit{Comp}(n_j) \wedge \mathit{Transp}^{\forall}(\langle n_j, \ldots, n_{i-1} \rangle)\ \vee$
   $ii)\ \exists j \geq i.\ \mathit{Comp}(n_j) \wedge \mathit{Transp}^{\forall}(\langle n_i, \ldots, n_{j-1} \rangle)$
2. Sei $CM \in \mathcal{CM}$. Dann:
   $\mathit{Correct}_{CM}(n) \iff_{df} \forall \langle n_1, \ldots, n_k \rangle \in \mathbf{P}[s, n]$
   $\exists i.\ \mathit{Insert}_{CM}(n_i) \wedge \mathit{Transp}^{\forall}(\langle n_i, \ldots, n_{k-1} \rangle)$

Aufwärts- und Abwärtssicherheit

Die Einschränkung der Definition für *Sicherheit* auf (i) bzw. (ii) führt auf die Begriffe

- *Aufwärtssicherheit*
- *Abwärtssicherheit*

Intuition

Eine Berechnung t ist an einer Programmstelle n

- *aufwärtssicher*, wenn t auf allen Pfaden von $\mathbf{s}$ nach n berechnet wird und auf die jeweils letzte Berechnung von t keine Modifikation eines Operanden von t mehr erfolgt.
- *abwärtssicher*, wenn t auf allen Pfaden von n nach $\mathbf{e}$ berechnet wird und der jeweils ersten Berechnung von t keine Modifikation eines Operanden von t vorausgeht.

Aufwärts- und Abwärtssicherheit

Definition [Aufwärts- und Abwärtssicherheit]

1. $\forall\, n \in N.$ *U-Safe*$(n) \iff_{df}$
   $\forall\, p \in \mathbf{P}[s,n]\ \exists\, i < \lambda_p.$ *Comp*$(p_i) \wedge$ *Transp*$^{\forall}(p[i, \lambda_p[)$

2. $\forall\, n \in N.$ *D-Safe*$(n) \iff_{df}$
   $\forall\, p \in \mathbf{P}[n,e]\ \exists\, i \leq \lambda_p.$ *Comp*$(p_i) \wedge$ *Transp*$^{\forall}(p[1, i[)$

Damit können wir jetzt genauer definieren...

Definition [Zulässige CM-Transformation]
Eine CM-Transformation $CM \in \mathcal{CM}$ heißt *zulässig* gdw für jeden Knoten $n \in N$ gelten folgende beide Eigenschaften:

1. $Insert_{CM}(n) \Rightarrow$ *Safe*(n)
2. $Repl_{CM}(n) \Rightarrow$ *Correct*$_{CM}(n)$

Die Menge aller zulässigen CM-Transformationen bezeichnen wir mit $\mathcal{CM}_{Adm}$.

Erste Aussagen... (1)

Korrektheitslemma

$$\forall\, CM \in \mathcal{CM}_{Adm}\ \forall\, n \in N.\ \textit{Correct}_{CM}(n) \Rightarrow \textit{Safe}(n)$$

Erste Aussagen... (2)

Sicherheitslemma

$$\forall\, n \in N.\ \textit{Safe}(n) \iff \textit{D-Safe}(n) \vee \textit{U-Safe}(n)$$

Berechnungsbesser, berechnungsoptimal

Eine CM-Transformation $CM \in \mathcal{CM}_{Adm}$ heißt *berechnungsbesser* als eine CM-Transformation $CM' \in \mathcal{CM}_{Adm}$ gdw

$$\forall\ p \in \mathbf{P}[s,e].\ |\,\{i \mid \textit{Comp}_{CM}(p_i)\}\,| \leq |\,\{i \mid \textit{Comp}_{CM'}(p_i)\}\,|$$

Bemerkung: Die Relation "berechnungsbesser" ist eine Quasiordnung, d.h. eine reflexive und transitive Relation.

Berechnungsoptimalität

Definition [Berechnungsoptimale CM-Transformation]
Eine zulässige CM-Transformation $CM \in \mathcal{CM}_{Adm}$ heißt *berechnungsoptimal* gdw CM ist berechnungsbesser als jede andere zulässige CM-Transformation.

Wir bezeichnen die Menge der berechnungsoptimalen CM-Transformationen mit $\mathcal{CM}_{CmpOpt}$.

Konzeptuell

...PRE kann als zweistufiger Prozess gesehen werden

1. Vorziehen von Ausdrücken (Expression hoisting)
   ...vorziehen von Ausdrücken an "frühere" sichere Berechnungspunkte
2. Beseitigung total redundanter Ausdrücke (Total redundancy elimination)
   ...beseitigen von Berechnungen, die durch das Vorziehen total redundant geworden sind

Kap. 7.4.2 Busy Code Motion

Extreme Strategie – Frühestheitprinzip

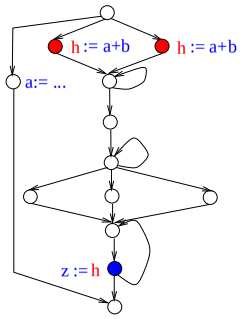
Platziere Berechnungen so früh wie möglich...

- Theorem [Berechnungsoptimalität]
  ...vorziehen von Ausdrücken zu ihren frühesten sicheren Berechnungspunkten liefert berechnungsoptimale Programme

  $\leadsto$ ...bekannt als Busy Code Motion

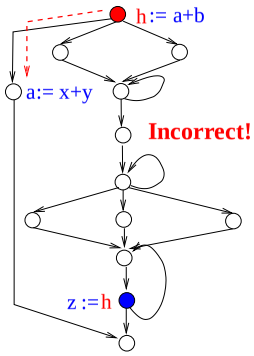
Platzieren von Berechnungen so früh wie möglich...

...liefert berechnungsoptimale Programme.



Beachte: Frühest heißt in der Tat...

...so früh wie möglich, aber nicht früher!



Busy Code Motion

Intuition:

Platziere Berechnungen *so früh wie möglich* im Programm, ohne Sicherheit und Korrektheit zu verletzen!

Beachte: Berechnungen werden dadurch so weit wie möglich entgegen des Kontrollflusses verschoben

$\leadsto$ ...liefert die Motivation für die Wahl der Bezeichnung *busy*.

Frühestheit

Definition [Frühestheit]
$\forall n \in N.\ \texttt{Earliest}(n) =_{df} \mathit{Safe}(n) \wedge$

$$\begin{cases} \texttt{tt} & \text{falls } n = s \\ \bigvee\limits_{m \in pred(n)} \neg \mathit{Transp}(m) \vee \neg \mathit{Safe}(m) & \text{sonst} \end{cases}$$

Die BCM-Transformation

- $Insert_{BCM}(n) =_{df} \texttt{Earliest}(n)$
- $Repl_{BCM}(n) =_{df} \mathit{Comp}(n)$

Das BCM-Theorem

BCM-Theorem

Die BCM-Transformation ist berechnungsoptimal, d.h., $BCM \in \mathcal{CM}_{CmpOpt}$.

Der Beweis für das BCM-Theorem stützt sich ab auf das *Frühestheit*- und das *BCM-Lemma*...

Das Frühestheitlemma

Frühestheitlemma

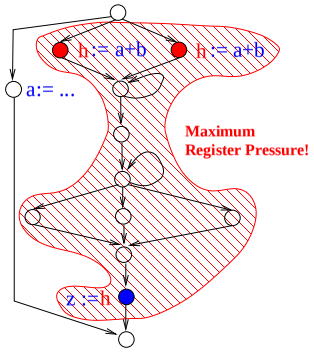
Sei $n \in N$. Dann gilt:

1. $\mathit{Safe}(n) \Rightarrow \forall p \in \mathbf{P}[s,n]\ \exists i \leq \lambda_p.$
   $\texttt{Earliest}(p_i) \wedge\ \mathit{Transp}^{\forall}(p[i, \lambda_p[)$
2. $\texttt{Earliest}(n) \iff$
   $\mathit{D\text{-}Safe}(n) \wedge \bigwedge\limits_{m \in pred(n)} (\neg \mathit{Transp}(m) \vee \neg \mathit{Safe}(m))$
3. $\texttt{Earliest}(n) \iff$
   $\mathit{Safe}(n) \wedge\ \forall\, CM \in \mathcal{CM}_{Adm}.\ \mathit{Correct}_{CM}(n) \Rightarrow \mathit{Insert}_{CM}(n)$

BCM-Lemma

Sei $p \in \mathbf{P}[s,e]$. Dann gilt:

1. $\forall i \leq \lambda_p.\ Insert_{BCM}(p_i) \iff$
   $\exists j \geq i.\ p[i,j] \in \mathit{FU\text{-}LtRg}(BCM)$
2. $\forall CM \in \mathcal{CM}_{Adm}\ \forall i,j \leq \lambda_p.\ p[i,j] \in \mathit{LtRg}(BCM) \Rightarrow$
   $\mathit{Comp}^{\exists}_{CM}(p[i,j])$
3. $\forall CM \in \mathcal{CM}_{CmpOpt}\ \forall i \leq \lambda_p.\ \mathit{Comp}_{CM}(p_i) \Rightarrow$
   $\exists j \leq i \leq l.\ p[j,l] \in \mathit{FU\text{-}LtRg}(BCM)$



Kap. 7.4.3 Lazy Code Motion

Duale extreme Strategie – Spätestheitprinzip

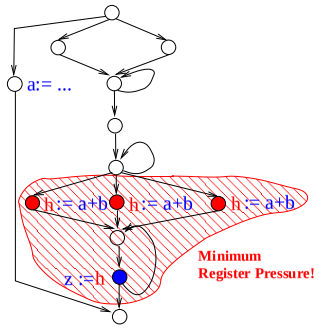
Platziere Berechnungen so spät wie möglich...

- Theorem [Optimalität]
  ...vorziehen von Ausdrücken so wenig wie möglich, aber so weit wie nötig (um berechnungsoptimal zu werden), liefert berechnungsoptimale Programme
  mit minimalem Registerdruck

  $\leadsto$ ...bekannt als Lazy Code Motion

Die LCM-Transformation

...ebenfalls berechnungsoptimal, aber mit minimalem Registerdruck!



Lazy Code Motion

Intuition:

Platziere Berechnungen *so spät wie möglich* im Programm, ohne Sicherheit, Korrektheit und Berechnungsoptimalität zu verletzen!

Beachte: Berechnungen werden dadurch so wenig wie möglich entgegen des Kontrollflusses verschoben

$\leadsto$ ...liefert die Motivation für die Wahl der Bezeichnung *lazy*.

Arbeitsplan

In der Folge werden wir definieren...

- Die Menge der *lebenszeitoptimalen* PRE-Transformationen
- Die LCM-Transformation als eindeutig bestimmte einzige lebenszeitoptimale PRE-Transformation

Zur Formalisierung

...ist der Begriff des *Lebenszeitbereichs* zentral.

Sei $CM \in \mathcal{CM}$.

- *Lebenszeitbereich*
  $\mathit{LtRg}(CM) =_{df}$
  $\{p \mid Insert_{CM}(p_1)\ \wedge Repl_{CM}(p_{\lambda_p}) \wedge \neg Insert^{\exists}_{CM}(p]1,\lambda_p])\}$
- *Erstbenutzungslebenszeitbereich*
  $\mathit{FU\text{-}LtRg}(CM) =_{df}$
  $\{p \in \mathit{LtRg}(CM) \mid \forall q \in \mathit{LtRg}(CM).\ (q \sqsubseteq p) \Rightarrow (q = p)\}$

Erste Aussagen

Erstbenutzungslebenszeitbereichslemma
Sei $CM \in \mathcal{CM}$, $p \in \mathbf{P}[s,e]$ und seien i_1, i_2, j_1, j_2 Indizes so dass $p[i_1,j_1] \in \mathit{FU\text{-}LtRg}(CM)$ und $p[i_2,j_2] \in \mathit{FU\text{-}LtRg}(CM)$. Dann gilt:

- entweder stimmen $p[i_1,j_1]$ und $p[i_2,j_2]$ überein, d.h. $i_1 = i_2$ und $j_1 = j_2$, oder
- $p[i_1,j_1]$ und $p[i_2,j_2]$ sind disjunkt, d.h., $j_1 < i_2$ oder $j_2 < i_1$.

Lebenszeitbesser, lebenszeitoptimal

Eine CM-Transformation $CM \in \mathcal{CM}$ heißt *lebenszeitbesser* als eine CM-Transformation $CM' \in \mathcal{CM}$ gdw

$$\forall\, p \in \mathit{LtRg}(CM)\ \exists q \in \mathit{LtRg}(CM').\ p \sqsubseteq q$$

Bemerkung: Die Relation "lebenszeitbesser" ist eine partielle Ordnung, d.h. eine reflexive, transitive und antisymmetrische Relation.

Lebenszeitoptimalität

Definition [Lebenszeitoptimale CM-Transformation]
Eine berechnungsoptimale CM-Transformation $CM \in \mathcal{CM}_{CmpOpt}$ heißt *lebenszeitoptimal* gdw CM ist lebenszeitbesser als jede andere berechnungsoptimale CM-Transformation.

Wir bezeichnen die Menge der lebenszeitoptimalen CM-Transformationen mit $\mathcal{CM}_{LtOpt}$.

Wdhg: 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. $\forall m,n \in M.\ m\,R\,n \wedge n\,R\,m \ \Rightarrow\ m = n$

Wo wir dabei sind...

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

Wdhg: 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 noch ergänzt...

- *Äquivalenzrelation* gdw. R ist reflexiv, transitiv und symmetrisch

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

Eindeutigkeit lebenszeitoptimaler PRE

Offensichtlich gilt:

$$\mathcal{CM}_{LtOpt} \subseteq \mathcal{CM}_{CmpOpt} \subseteq \mathcal{CM}_{Adm} \subset \mathcal{CM}$$

Es gilt sogar weitergehend:

Theorem [Eindeutigkeit lebenszeitoptimaler CM-Transformationen]

$$|\,\mathcal{CM}_{LtOpt}\,| \leq 1$$

Zur Entwicklung der LCM-Transformation

Zunächst folgende Beobachtung:

Lemma
$\forall\, CM \in \mathcal{CM}_{CmpOpt}\ \forall\, p \in \mathit{LtRg}(CM)\ \exists q \in \mathit{LtRg}(BCM).\ p \sqsubseteq q.$

Intuitiv:

- Keine berechnungsoptimale CM-Transformation platziert die Berechnungen früher als die BCM-Transformation
- Die BCM-Transformation ist diejenige berechnungsoptimale CM-Transformation mit maximalem Registerdruck

Verzögerbarkeit

Definition [Verzögerbarkeit]
$\forall n \in N.\ \mathit{Delayed}(n) \iff_{df}$
$\forall\, p \in \mathbf{P}[s,n]\ \exists\, i \leq \lambda_p.\ \mathtt{Earliest}(p_i) \wedge \neg \mathit{Comp}^{\exists}(\,p[i,\lambda_p[\,)$

Das Verzögerbarkeitslemma

Verzögerbarkeitslemma

1. $\forall\, n \in N.\ \mathit{Delayed}(n) \ \Rightarrow\ \mathit{D\text{-}Safe}(n)$
2. $\forall\, p \in \mathbf{P}[s,e]\ \forall i \leq \lambda_p.\ \mathit{Delayed}(p_i) \Rightarrow \exists\ j \leq i \leq l.\ p[j,l] \in \mathit{FU\text{-}LtRg}(BCM)$
3. $\forall\, CM \in \mathcal{CM}_{CmpOpt}\ \forall\, n \in N.\ \mathit{Comp}_{CM}(n) \Rightarrow \mathit{Delayed}(n)$

Spätestheit

Definition [Spätestheit]
$\forall\, n \in N.\ \mathit{Latest}(n) =_{df}\ \mathit{Delayed}(n) \wedge (\mathit{Comp}(n) \vee \bigvee_{m \in succ(n)} \neg \mathit{Delayed}(m))$

Das Spätestheitlemma

Spätestheitlemma

1. $\forall\, p \in \mathit{LtRg}(BCM)\ \exists\, i \leq \lambda_p.\ \mathit{Latest}(p_i)$
2. $\forall\, p \in \mathit{LtRg}(BCM)\ \forall\, i \leq \lambda_p.\ \mathit{Latest}(p_i) \Rightarrow \neg \mathit{Delayed}^{\exists}(\,p]i, \lambda_p]\,)$

Die ALCM-Transformation

Die "Almost Lazy Code Motion" Transformation...

- $\mathit{Insert}_{ALCM}(n) =_{df} \mathit{Latest}(n)$
- $\mathit{Repl}_{ALCM}(n) =_{df} \mathit{Comp}(n)$

Fast lebenszeitoptimal

Definition [Fast lebenszeitoptimale CM-Transformation]
Eine berechnungsoptimale CM-Transformation $CM \in \mathcal{CM}_{CmpOpt}$ heißt *fast lebenszeitoptimal* gdw

$\forall\, p \in \mathit{LtRg}(CM).\ \lambda_p \geq 2 \Rightarrow$

$\forall\, CM' \in \mathcal{CM}_{CmpOpt}\ \exists\, q \in \mathit{LtRg}(CM').\ p \sqsubseteq q$

Wir bezeichnen die Menge der fast lebenszeitoptimalen CM-Transformationen mit $\mathcal{CM}_{ALtOpt}$.

Das ALCM-Theorem

ALCM-Theorem

Die $ALCM$-Transformation ist fast lebenszeitoptimal, d.h., $ALCM \in \mathcal{CM}_{ALtOpt}$.

Isolierte Berechnungen

Definition [CM-Isolation]
$\forall\, CM \in \mathcal{CM}\ \forall\, n \in N.\ \mathit{Isolated}_{CM}(n) \iff_{df}$
$\forall\, p \in \mathbf{P}[n,e]\ \forall 1 < i \leq \lambda_p.\ \mathit{Repl}_{CM}(p_i) \Rightarrow \mathit{Insert}^{\exists}_{CM}(\,p]1, i]\,)$

Das Isolationslemma

Isolationslemma

1. $\forall\, CM \in \mathcal{CM}\ \forall\, n \in N.\ \mathit{Isolated}_{CM}(n) \iff$
   $\forall\, p \in \mathit{LtRg}(CM).\ \langle n \rangle \sqsubseteq p \Rightarrow \lambda_p = 1$
2. $\forall\, CM \in \mathcal{CM}_{CmpOpt}\ \forall\, n \in N.\ \mathit{Latest}(n) \Rightarrow$
   $(\mathit{Isolated}_{CM}(n) \iff \mathit{Isolated}_{BCM}(n))$

Die LCM-Transformation

- $Insert_{LCM}(n) =_{df} Latest(n) \wedge \neg Isolated_{BCM}(n)$
- $Repl_{LCM}(n) =_{df} Comp(n) \wedge \neg(Latest(n) \wedge Isolated_{BCM}(n))$

Das LCM-Theorem

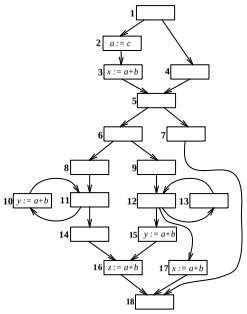
LCM-Theorem

Die LCM-Transformation ist lebenszeitoptimal, d.h., $LCM \in \mathcal{CM}_{LtOpt}$.

Kap. 7.4.4 Ein größeres Beispiel

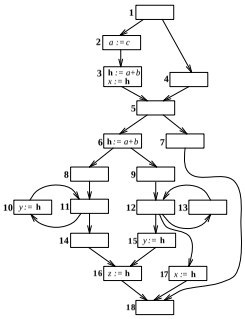
Ein größeres Beispiel zur Illustration (1)

Das Ausgangsprogramm...



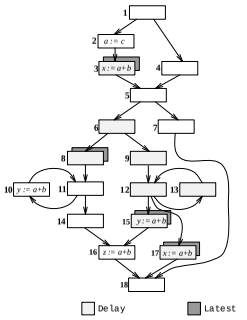
Ein größeres Beispiel zur Illustration (2)

Das Resultat der BCM-Transformation...



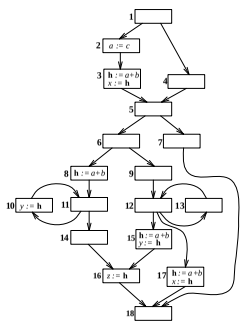
Ein größeres Beispiel zur Illustration (3)

Verzögerte und späteste Berechnungspunkte...



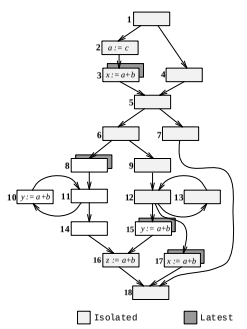
Ein größeres Beispiel zur Illustration (4)

Das Resultat der ALCM-Transformation...

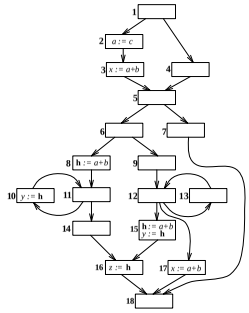


Ein größeres Beispiel zur Illustration (5)

Späteste und isolierte Berechnungspunkte...



Das Resultat der LCM-Transformation...



Kap. 7.4.5 Implementierungspragmatik

Zur Implementierung der BCM-Transformation auf EA-Graphen

...auf Einzelanweisungsniveau, hier für knotenbenannte EA-Graphen.

Beachte: ...wir nehmen für das folgende an, dass nur kritische Kanten gespalten sind (deshalb N- und X-Einsetzungen).

Busy Code Motion (EA-1)

1. Die Analysen für Aufwärts- und Abwärtssicherheit

Lokale Prädikate:

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

Busy Code Motion (EA-2)

Das Gleichungssystem für Aufwärtssicherheit:

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

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

Busy Code Motion (EA-3)

Das Gleichungssystem für Abwärtssicherheit:

$$\text{N-DSAFE}_\iota = \mathbf{COMP}_\iota + \text{X-DSAFE}_\iota \cdot \mathbf{TRANSP}_\iota$$

$$\text{X-DSAFE}_\iota = \begin{cases} \texttt{ff} & \text{falls } \iota = e \\ \prod_{\widehat{\iota} \in succ(\iota)} \text{N-DSAFE}_{\widehat{\iota}} & \text{sonst} \end{cases}$$

Busy Code Motion (EA-4)

2. Die Transformation: Einsetzungs- und Ersetzungspunkte

Lokale Prädikate:

- N-USAFE*, X-USAFE*, N-DSAFE*, X-DSAFE*: größte Lösungen der Gleichungssysteme für Aufwärts- und Abwärtssicherheit aus Schritt 1.

Busy Code Motion (EA-5)

$$\text{N-INSERT}_\iota^{\text{BCM}} =_{df} \text{N-DSAFE}^\star_\iota \cdot \prod_{\widehat{\iota} \in pred(\iota)} (\overline{\text{X-USAFE}^\star_{\widehat{\iota}} + \text{X-DSAFE}^\star_{\widehat{\iota}}})$$

$$\text{X-INSERT}_\iota^{\text{BCM}} =_{df} \text{X-DSAFE}^\star_\iota \cdot \overline{\mathbf{TRANSP}_\iota}$$

$$\text{REPLACE}_\iota^{\text{BCM}} =_{df} \mathbf{COMP}_\iota$$

Zur Implementierung der BCM-Transformation auf BB-Graphen (1)

...auf Basisblockniveau, hier für knotenbenannte BB-Graphen.

Beachte: ...wir nehmen für das folgende an, dass (1) nur kritische Kanten gespalten sind (deshalb N- und X-Einsetzungen), und (2) dass alle Redundanzen innerhalb eines Basisblocks schon durch einen Präprozess beseitigt sind.

t-verfeinerte Flussgraphen...

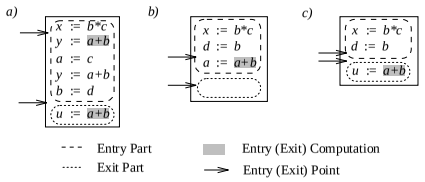
Bezüglich einer Berechnung t lässt sich ein Basisblock $\mathbf{n}$ in zwei disjunkte Teile unterteilen:

- ein *Eingangsteil* (*entry part*), der aus allen Anweisungen bis zu und einschließlich der letzten Modifikation von t besteht
- ein *Ausgangsteil* (*exit part*), der aus den verbleibenden Anweisungen von $\mathbf{n}$ besteht.

Beachte: ein nichtleerer Basisblock hat stets einen nichtleeren Eingangsteil; im Unterschied dazu kann der Ausgangsteil leer sein (zur Illustration siehe folgende Abbildung).

Zur Implementierung der BCM-Transformation auf BB-Graphen (3)

Zur Illustration von Eingangs- und Ausgangsteil eines Basisblocks...



Busy Code Motion (BB-1)

1. Die Analysen für Aufwärts- und Abwärtssicherheit

Lokale Prädikate:

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

Busy Code Motion (BB-2)

Das Gleichungssystem für Aufwärtssicherheit:

$$\text{BB-N-USAFE}_\beta = \begin{cases} \mathtt{ff} & \text{falls } \beta = \mathbf{s} \\ \prod_{\tilde{\beta} \in pred(\beta)} (\text{BB-XCOMP}_{\tilde{\beta}} + \text{BB-X-USAFE}_{\tilde{\beta}}) & \text{sonst} \end{cases}$$

$$\text{BB-X-USAFE}_\beta = (\text{BB-N-USAFE}_\beta + \text{BB-NCOMP}_\beta) \cdot \text{BB-TRANSP}_\beta$$

Busy Code Motion (BB-3)

Das Gleichungssystem für Abwärtssicherheit:

$$\text{BB-N-DSAFE}_\beta = \text{BB-NCOMP}_\beta + \text{BB-X-DSAFE}_\beta \cdot \text{BB-TRANSP}_\beta$$

$$\text{BB-X-DSAFE}_\beta = \text{BB-XCOMP}_\beta + \begin{cases} \mathtt{ff} & \text{falls } \beta = \mathbf{e} \\ \prod_{\tilde{\beta} \in succ(\beta)} \text{BB-N-DSAFE}_{\tilde{\beta}} & \text{sonst} \end{cases}$$

Busy Code Motion (BB-4)

2. Die Transformation: Einsetzungs- und Ersetzungspunkte

Lokale Prädikate:

- BB-N-USAFE*, BB-X-USAFE*, BB-N-DSAFE*, BB-X-DSAFE*: größte Lösungen der Gleichungssysteme für Aufwärts- und Abwärtssicherheit aus Schritt 1.

Busy Code Motion (BB-5)

$$\text{N-INSERT}^{\text{BCM}}_\beta =_{df} \text{BB-N-DSAFE}^*_\beta \cdot \prod_{\tilde{\beta} \in pred(\beta)} (\overline{\text{BB-X-USAFE}^*_{\tilde{\beta}} + \text{BB-X-DSAFE}^*_{\tilde{\beta}}})$$

$$\text{X-INSERT}^{\text{BCM}}_\beta =_{df} \text{BB-X-DSAFE}^*_\beta \cdot \overline{\text{BB-TRANSP}_\beta}$$

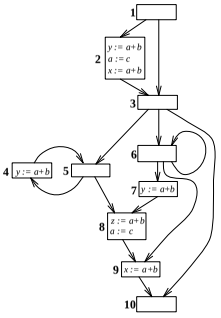
$$\text{N-REPLACE}^{\text{BCM}}_\beta =_{df} \text{BB-NCOMP}_\beta$$

$$\text{X-REPLACE}^{\text{BCM}}_\beta =_{df} \text{BB-XCOMP}_\beta$$

Die Gleichungssysteme für LCM

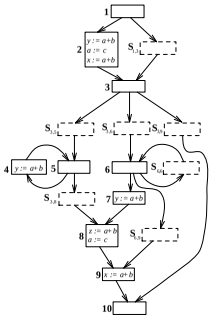
Ähnlich!

Das Ausgangsprogramm...



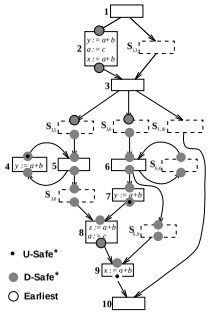
Ein größeres BB-Beispiel zur Illustration (2)

Das Ausgangsprogramm mit kritischen Kanten gespalten...



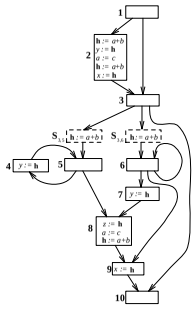
Ein größeres BB-Beispiel zur Illustration (3)

Die Berechnung der frühesten Berechnungspunkte...



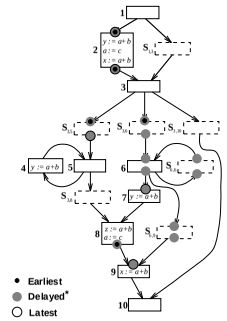
Ein größeres BB-Beispiel zur Illustration (4)

Das Ergebnis der BCM-Transformation...



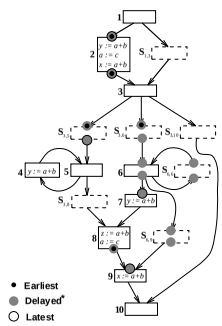
Ein größeres BB-Beispiel zur Illustration (5)

Die Berechnung der spätesten Berechnungspunkte...



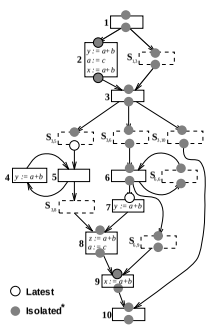
Ein größeres BB-Beispiel zur Illustration (6)

Das Ergebnis der ALCM-Transformation...

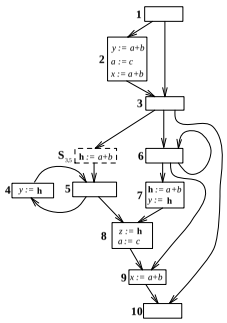


Ein größeres BB-Beispiel zur Illustration (7)

Die Berechnung isolierter Berechnungspunkte...



Das Ergebnis der LCM-Transformation...



Kapitel 7.4.5 PRE gemäß Morel/Renvoise

Die bahnbrechende PRE-Formulierung von Morel/Renvoise

PRE ist untrennbar mit den Namen von E. Morel und C. Renvoise verknüpft. Ihr 1979 vorgestelltes Verfahren kann als "Urvater" aller CM-Verfahren angesehen werden und war bis in die 90er-Jahre hinein das "state of the art"-Verfahren für PRE.

Kennzeichnend für dieses Verfahren sind:

- 3 unidirektionale Bitvektoranalysen (AV, ANT, PAV)
- 1 bidirektionales Bitvektoranalyse (PP)

PRE gemäß Morel/Renvoise (1)

- Verfügbarkeit (Availability):

$$\mathbf{AVIN}(n) = \begin{cases} \mathtt{ff} & \text{falls } n = \mathbf{s} \\ \prod\limits_{m \in pred(n)} \mathbf{AVOUT}(m) & \text{sonst} \end{cases}$$

$$\mathbf{AVOUT}(n) = \mathbf{TRANSP}(n) * (\mathbf{COMP}(n) + \mathbf{AVIN}(n))$$

PRE gemäß Morel/Renvoise (2)

- Vorziehbarkeit (Anticipability):

$$\mathbf{ANTIN}(n) = \mathbf{COMP}(n) + \mathbf{TRANSP}(n) * \mathbf{ANTOUT}(n)$$

$$\mathbf{ANTOUT}(n) = \begin{cases} \mathtt{ff} & \text{falls } n = \mathbf{e} \\ \prod\limits_{m \in succ(n)} \mathbf{ANTIN}(m) & \text{sonst} \end{cases}$$

PRE gemäß Morel/Renvoise (3)

- Partielle Verfügbarkeit (Partial Availability):

$$\mathbf{PAVIN}(n) = \begin{cases} \mathtt{ff} & \text{falls } n = \mathbf{s} \\ \sum\limits_{m \in pred(n)} \mathbf{PAVOUT}(m) & \text{sonst} \end{cases}$$

$$\mathbf{PAVOUT}(n) = \mathbf{TRANSP}(n) * (\mathbf{COMP}(n) + \mathbf{PAVIN}(n))$$

PRE gemäß Morel/Renvoise (4)

- Platzierung möglich (Placement Possible):

$$\mathbf{PPIN}(n) = \begin{cases} \mathtt{ff} \text{ falls } n = \mathbf{s} \\ \mathbf{CONST}(n) * \\ (\prod\limits_{m \in pred(n)} (\mathbf{PPOUT}(m) + \mathbf{AVOUT}(m)) * \\ (\mathbf{COMP}(n) + \mathbf{TRANSP}(n) * \mathbf{PPOUT}(n)) \quad \text{sonst} \end{cases}$$

$$\mathbf{PPOUT}(n) = \begin{cases} \mathtt{ff} & \text{falls } n = \mathbf{e} \\ \prod\limits_{m \in succ(n)} \mathbf{PPIN}(m) & \text{sonst} \end{cases}$$

wobei $\mathbf{CONST}(n) =_{df} \mathbf{ANTIN}(n) * (\mathbf{PAVIN}(n) + \neg\mathbf{COMP}(n) * \mathbf{TRANSP}(n))$

PRE gemäß Morel/Renvoise (5)

- Initialisierung:

$$\mathbf{INSIN}(n) =_{df} \mathtt{ff}$$

$$\mathbf{INSOUT}(n) =_{df} \mathbf{PPOUT}(n) * \neg\mathbf{AVOUT}(n) * (\neg\mathbf{PPIN}(n) + \neg\mathbf{TRANSP}(n))$$

- Ersetzung:

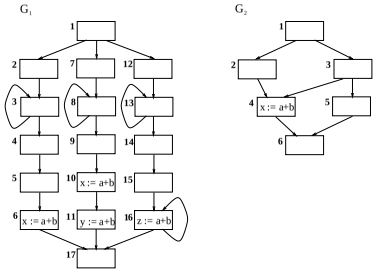
$$\mathbf{REPLACE}(n) =_{df} \mathbf{COMP}(n) * \mathbf{PPIN}(n)$$

...gemäß Morel/Renvoise:

- Konzeptuell
  - Fehlende Berechnungsoptimalität
    $\rightsquigarrow$ nur aufgrund nicht gespaltener kritischer Kanten
  - Fehlende Lebenszeitoptimalität
    $\rightsquigarrow$ Heuristische Behandlung
- Technisch
  - Bidirektionalität
    $\rightsquigarrow$ konzeptuell und berechnungsmäßig komplexer

...das Transformationsergebnis liegt (nicht vorhersagbar) zwischen denen von BCM- und LCM-Transformation.

...folgende zwei Beispiele mithilfe des PRE-Verfahrens von Morel/Renvoise zu optimieren:



Kap. 7.4.6 Sparse Code Motion

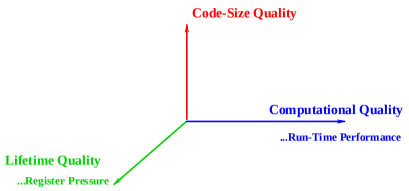
Heutzutage...

Lazy Code Motion ist...

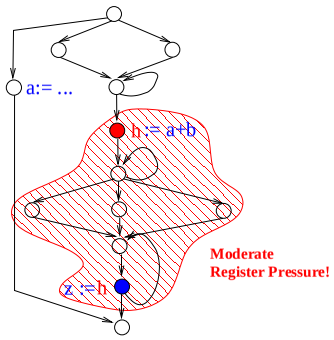
- ...der de-facto Standardalgorithmus für PRE, der in aktuellen state-of-the-art Übersetzern zum Einsatz kommt
  - Gnu compiler family
  - Sun Sparc compiler family
  - ...

In der Folge...

(Modulare) Erweiterung von LCM, um Anwenderprioritäten zu berücksichtigen!

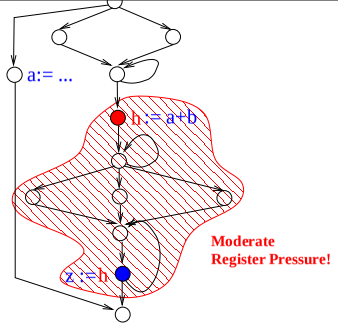


...um auch diese Transformation zu ermöglichen:



There is more than speed!

There is more than speed!



Chip-Kategorie	Verkaufte Anzahl
Eingebettet 4-bit	2000 Millionen
Eingebettet 8-bit	4700 Millionen
Eingebettet 16-bit	700 Millionen
Eingebettet 32-bit	400 Millionen
DSP	600 Millionen
Desktop 32/64-bit	150 Millionen

... David Tennenhouse (Intel Director of Research). Hauptvortrag auf dem *20th IEEE Real-Time Systems Symposium* (*RTSS'99*), Phoenix, Arizona, Dezember 1999.

Der Weltmarkt für Mikroprozessoren 1999

Chip-Kategorie	Verkaufte Anzahl
Eingebettet 4-bit	2000 Millionen
Eingebettet 8-bit	4700 Millionen
Eingebettet 16-bit	700 Millionen
Eingebettet 32-bit	400 Millionen
DSP	600 Millionen
Desktop 32/64-bit	150 Millionen

$\sim$ **2%**

... David Tennenhouse (Intel Director of Research). Hauptvortrag auf dem *20th IEEE Real-Time Systems Symposium* (*RTSS'99*), Phoenix, Arizona, Dezember 1999.

Man denke an...

... domänenspezifische Prozessoren eingesetzt in eingebetteten Systemen

- **Telekommunikation**
  - Mobiltelefone, Pager, ...
- **Heimelektronik**
  - MP3-Spieler, Kameras, Spielekonsolen, ...
- **Automobilbereich**
  - GPS-Navigation, Airbags, ...
- ...

Code für eingebettete Systeme

Anforderungen...

- Performanz (oft Echtzeitanforderungen)
- Codegröße (system-on-chip, on-chip RAM/ROM)
- ...

Für eingebettete Systeme...

...Codegröße ist oft eine kritischere Größe als Geschwindigkeit!

Code für eingebettete Systeme (Fortsetzung)

Anforderungen (und wie sie häufig adressiert werden...):

- Assemblerprogrammierung
- Händische Postoptimierung

Schwächen...

- Fehlerträchtig
- Verzögerte Marktreife/-eintritt

...die Probleme werden zunehmend größer mit wachsender Komplexität.

Generell beobachtet man...

...einen Trend hin zu Hochsprachenprogrammierung (C/C++)

Angesichts dieses Trends...

...wie unterstützen traditionelle Übersetzer- und Optimierertechnologien das spezielle Anforderungsprofil von Code für eingebettete Systeme?

Leider nur wenig.

Unbestritten...

Traditionelle Optimierungen...

- ...sind nahezu ausschließlich auf Performanzoptimierung getrimmt
- ...sind nicht codegrößensensitiv und bieten i.a. keinerlei Kontrolle über ihren Einfluss auf die Codegröße

...gilt dies für Optimierungen, die auf Codeverschiebung beruhen

Dazu gehören insbesondere

- Partial redundancy elimination
- Partial dead-code elimination
- Partial redundant-assignment elimination
- Strength reduction
- ...

Erinnerung am Beispiel von PRE

PRE kann konzeptuell als zweistufiger Prozess gesehen werden...

1. Ausdrucksvorziehen
   ...vorziehen von Ausdrücken an "frühere" sichere Berechnungspunkte
2. Totale Redundanzelimination
   ...eliminieren von Berechnungen, die durch das Vorziehen total redundant geworden sind

Erinnerung am Beispiel von LCM

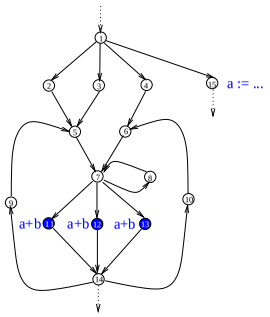
LCM kann konzeptuell als Ergebnis eines zweistufigen Prozesses gesehen werden...

1. Ausdrucksvorziehen (hoisting expressions)
   ...zu ihren "frühesten" sicheren Berechnungspunkten
2. Ausdrucksverzögern (sinking expressions)
   ...zu ihren "spätesten" sicheren und berechnungsoptimalen Berechnungspunkten

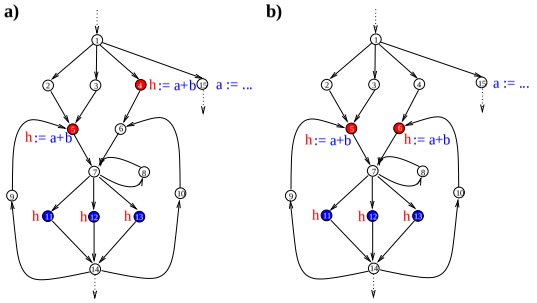
Auf dem Weg zu codegrößensensitiver PRE...

- **Hintergrund: Klassische PRE**
  - ⇝ Busy CM (BCM) / Lazy CM (LCM) (Knoop et al., PLDI'92)
  - – Ausgezeichnet mit dem *ACM SIGPLAN Most Influential PLDI Paper Award 2002* (*for 1992*)
  - – Ausgewählt für *"20 Years of the ACM SIGPLAN PLDI: A Selection"* (60 Artikel aus ca. 600 Artikeln)
- **Codegrößensensitive PRE** (Knoop et al., POPL'00)
  - ⇝ ...modulare Erweiterung von BCM/LCM
    - ∗ Problemmodellierung und -lösung
      ...basiert auf graphtheoretischen Hilfsmitteln
    - ∗ Hauptergebnisse
      ...**Korrektheit, Optimalität**

Laufendes Beispiel

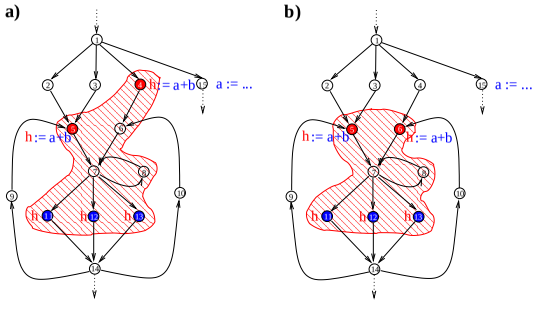


Laufendes Beispiel (Fortsetzung)



Two Code–size Optimal Programs

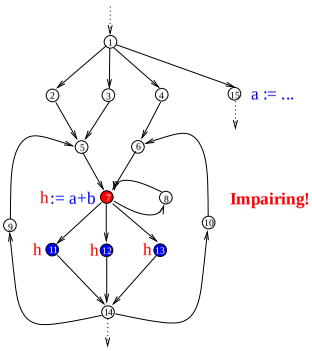
Laufendes Beispiel (Fortsetzung)



SQ > CQ> LQ **SQ > LQ > CQ**

Laufendes Beispiel (Fortsetzung)

Beachte: Folgende Transformation ist unerwünscht!



↝ **Das Problem**

...wir erhalten wir eine codegrößenminimale Platzierung der Berechnungen, d.h. eine Platzierung, die

- zulässig (semantik- & performanzerhaltend)
- codegrößenminimal ist?

↝ **Lösung: Eine neue Sicht auf PRE**

...betrachte PRE als ein Austauschproblem: Austauschen der ursprünglichen Berechnungen gegen neu eingesetzte!

↝ **Der Clou: Benutze Graphtheorie!**

...führe das Austauschproblem auf die Berechnung sog. tight sets in bipartiten Graphen zurück basierend auf maximalen matchings!

Wir müssen beantworten...

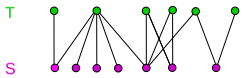
- Wo sind Initialisierungen vorzunehmen und wo sind ursprüngliche Berechnungen zu ersetzen?

...und beweisen

- Warum dies korrekt (semantikerhaltend) ist
- Wie sich dies auf die Codegröße auswirkt
- Warum dies "optimal" bezüglich einer vorgegebenen Priorisierung von Zielen ist?

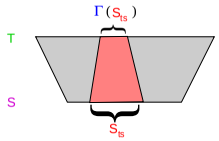
Für jede dieser Fragen werden wir ein spezielles Theorem angeben, das uns die entsprechende Antwort liefert!

Bipartite Graphen



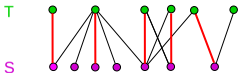
Tight Set
...eines bipartiten Graphen $(S \cup T, E)$: Teilmenge $S_{ts} \subseteq S$ mit

$$\forall\, S' \subseteq S.\ |S_{ts}| - |\Gamma(S_{ts})| \ \geq\ |S'| - |\Gamma(S')|$$



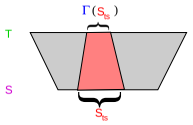
Zwei Varianten: (1) **Größte** Tight Sets (2) **Kleinste** Tight Sets

Bipartite Graphen



Tight Set
...eines bipartiten Graphen $(S \cup T, E)$: Teilmenge $S_{ts} \subseteq S$ mit

$$\forall\, S' \subseteq S.\ |S_{ts}| - |\Gamma(S_{ts})| \ \geq\ |S'| - |\Gamma(S')|$$



Zwei Varianten: (1) **Größte** Tight Sets (2) **Kleinste** Tight Sets

Offensichtlich

...können wir auf vorgefertigte Standardalgorithmen aus der Graphtheorie zurückgreifen, um

- **Maximale Matchings** und
- **Tight sets**

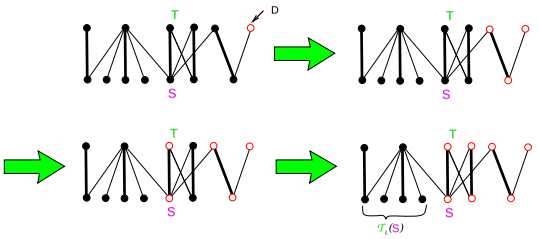
zu berechnen.

Damit reduziert sich unser PRE-Problem auf...

...die Konstruktion des bipartiten Graphen, der das Problem modelliert!

Zur Berechnung größter/kleinster Tight Sets

...auf Grundlage **maximaler Matchings**



Algorithmus LTS (Largest Tight Sets)

Eingabe: Bipartiter Graph (S∪̇T, E), maximales Matching M.

Ausgabe: Größte tight set $\mathcal{T}_{LaTS}(S) \subseteq S$.

```
S_M := S; D := {t ∈ T | t is unmatched};
WHILE D ≠ ∅ DO
  choose some x ∈ D; D := D \ {x};
  IF x ∈ S
    THEN S_M := S_M \ {x};
         D := D ∪ {y | {x,y} ∈ M}
    ELSE D := D ∪ (Γ(x) ∩ S_M)
  FI
OD;
T_LaTS(S) := S_M
```


Algorithmus STS (Smallest Tight Sets)

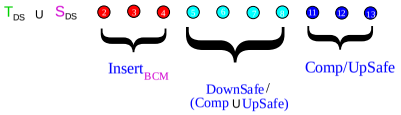
Eingabe: Bipartiter Graph (S∪̇T, E), maximales Matching M.

Ausgabe: Kleinste tight set $\mathcal{T}_{SmTS}(S) \subseteq S$.

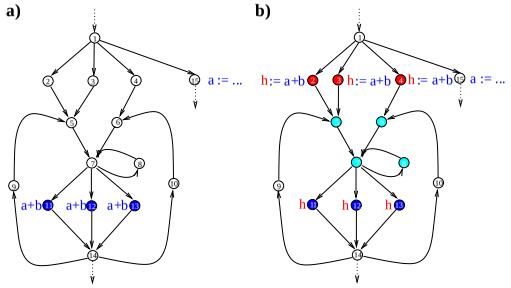
```
S_M := ∅; A := {s ∈ S | s is unmatched};
WHILE A ≠ ∅ DO
  choose some x ∈ A; A := A \ {x};
  IF x ∈ S
    THEN S_M := S_M ∪ {x};
         A := A ∪ (Γ(x) \ S_M)
    ELSE A := A ∪ {y | {x,y} ∈ M}
  FI
OD;
T_SmTS(S) := S_M
```

Modellierung des Austauschproblems

Die Menge der Knoten

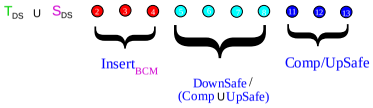


Die Menge der Kanten...

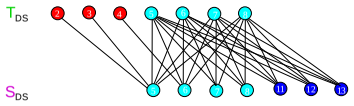


Modellierung des Austauschproblems

Die Menge der Knoten



Der bipartite Graph



Die Menge der Kanten ... $\forall n \in S_{DS}\ \forall m \in T_{DS}$.

$$\{n,m\} \in E_{DS} \Longleftrightarrow_{df} m \in \textbf{Closure}(pred(n))$$

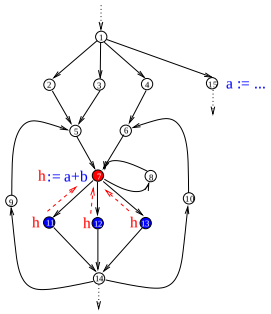
DownSafety-Hüllen

DownSafety-Hüllen

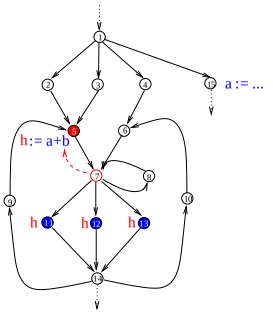
Für $n \in$ *DownSafe/Upsafe* ist die DownSafety-Hülle $Closure(n)$ die kleinste Menge von Knoten, so dass

1. $n \in Closure(n)$
2. $\forall m \in Closure(n) \setminus \mathit{Comp}.\ succ(m) \subseteq Closure(n)$
3. $\forall m \in Closure(n).\ pred(m) \cap Closure(n) \neq \emptyset \Rightarrow$
   $pred(m) \setminus \mathit{UpSafe} \subseteq Closure(n)$

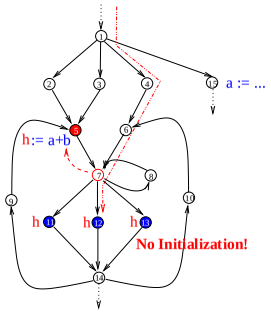
DownSafety-Hüllen – Die zentrale Idee 1(4)



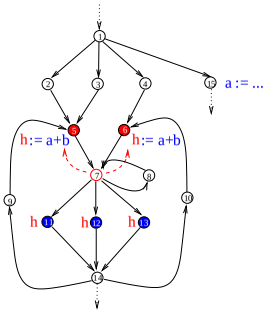
DownSafety-Hüllen – Die zentrale Idee 2(4)



DownSafety-Hüllen – Die zentrale Idee 3(4)



DownSafety-Hüllen – Die zentrale Idee 4(4)



DownSafety-Hüllen

DownSafety-Hüllen

Für $n \in$ *DownSafe/Upsafe* ist die DownSafety-Hülle $Closure(n)$ die kleinste Menge von Knoten, so dass

1. $n \in Closure(n)$
2. $\forall m \in Closure(n) \setminus Comp.\ succ(m) \subseteq Closure(n)$
3. $\forall m \in Closure(n).\ pred(m) \cap Closure(n) \neq \emptyset \Rightarrow pred(m) \setminus UpSafe \subseteq Closure(n)$

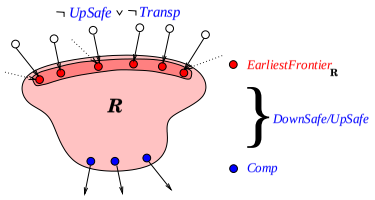
DownSafety-Regionen

Einige Teilmengen von Knoten sind in besonderer Weise ausgezeichnet. Wir nennen diese Mengen DownSafety-Regionen...

- Eine Menge $\mathcal{R} \subseteq N$ von Knoten heißt DownSafety-Region gdw
  1. $Comp \setminus UpSafe \subseteq \mathcal{R} \subseteq DownSafe \setminus UpSafe$
  2. $Closure(\mathcal{R}) = \mathcal{R}$

Fundamental...

Initialisierungstheorem
Initialisierungen zulässiger PRE-Transformationen erfolgen stets am **"frühesten Rand"** von **DownSafety-Regionen**.



...charakterisiert erstmals alle semantikerhaltenden PRE-Transformationen.

Die Schlüsselfragen

...bezüglich Korrektheit und Optimalität:

1. Wo Initialisierungen vornehmen, warum ist es korrekt?
2. Wie ist der Effekt auf die Codegröße?
3. Warum ist das Resultat optimal, d.h., codegrößenminimal?

...drei Theoreme werden jeweils eine dieser Fragen beantworten.

Hauptergebnisse / Erste Frage

1. Wo Initialisierungen vornehmen, warum ist es korrekt?

Intuitiv: am frühesten Rand der von der tight set induzierten DS-region...

Theorem 1 [Tight Sets: Initialisierungspunkte]
Sei $TS \subseteq S_{DS}$ eine tight set.
Dann ist $\mathcal{R}_{TS} =_{df} \Gamma(TS) \cup (Comp \setminus UpSafe)$
eine DownSafety-Region mit $Body_{\mathcal{R}_{TS}} = TS$

Korrektheit
...unmittelbares Korollar aus Theorem 1 und dem Initialisierungstheorem

Hauptergebnisse / Zweite Frage

2. Wie ist der Effekt auf die Codegröße?

Intuitiv: Die Differenz aus eingesetzten und ersetzten Berechnungen...

Theorem 2 [DownSafety-Regionen: Platzgewinn]
Sei $\mathcal{R}$ eine DownSafety-Region
mit $Body_{\mathcal{R}} =_{df} \mathcal{R} \setminus EarliestFrontier_{\mathcal{R}}$
Dann

- **Platzgewinn aufgrund Einsetzens an $EarliestFrontier_{\mathcal{R}}$:**
  $|Comp \setminus UpSafe| - |EarliestFrontier_{\mathcal{R}}| =$
  $|Body_{\mathcal{R}}| - |\Gamma(Body_{\mathcal{R}})| \quad_{df} = defic(Body_{\mathcal{R}})$

Hauptergebnisse / Dritte Frage

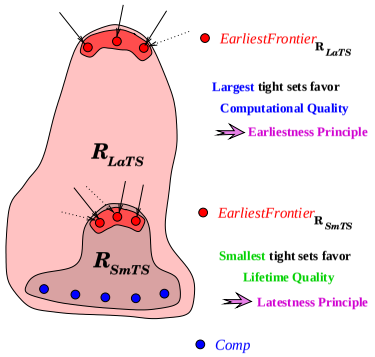
3. Warum ist das Resultat optimal, d.h., codegrößenminimal?

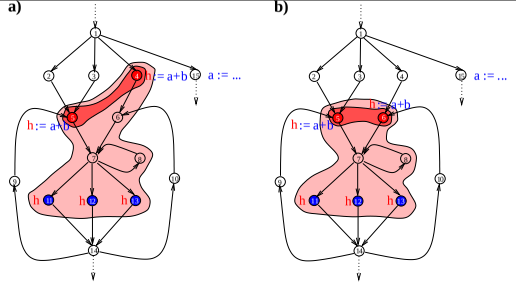
Aufgrund einer inhärenten Eigenschaft von tight sets (non-negative deficiency!)...

Optimalitätstheorem [Die Transformation]
Sei $TS \subseteq S_{DS}$ eine tight set.

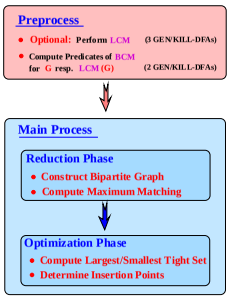
- **Initialisierungspunkte:**
  $Insert_{SpCM} =_{df} EarliestFrontier_{\mathcal{R}_{TS}} = \mathcal{R}_{TS} \setminus TS$
- **Platzgewinn:**
  $defic(TS) =_{df} |TS| - |\Gamma(TS)| \geq 0$ max.

Größte vs. kleinste Tight Sets: Der Einfluss





Blick 1(2)

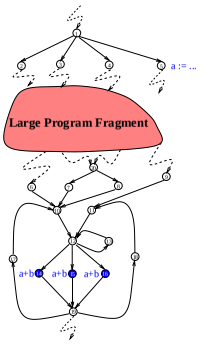


...auf einen Blick 2(2)

Choice of Priority	Apply	To	Using	Yields	Auxiliary Information Required
$\mathcal{LQ}$	Not meaningful: The identity, i.e., G itself is optimal!				
$\mathcal{SQ}$	Subsumed by $\mathcal{SQ} > \mathcal{CQ}$ and $\mathcal{SQ} > \mathcal{LQ}$!				
$\mathcal{CQ}$	BCM	G			UpSafe(G), DownSafe(G)
$\mathcal{CQ} > \mathcal{LQ}$	LCM	G		LCM(G)	UpSafe(G), DownSafe(G), Delay(G)
$\mathcal{SQ} > \mathcal{CQ}$	SpCM	G	Largest tight set	SpCM_{LTS}(G)	UpSafe(G), DownSafe(G)
$\mathcal{SQ} > \mathcal{LQ}$	SpCM	G	Smallest tight set		UpSafe(G), DownSafe(G)
$\mathcal{CQ} > \mathcal{SQ}$	SpCM	LCM(G)	Largest tight set		UpSafe(G), DownSafe(G), Delay(G) UpSafe(**LCM**(G)), DownSafe(**LCM**(G))
$\mathcal{CQ} > \mathcal{SQ} > \mathcal{LQ}$	SpCM	LCM(G)	Smallest tight set		UpSafe(G), DownSafe(G), Delay(G) UpSafe(**LCM**(G)), DownSafe(**LCM**(G))
$\mathcal{SQ} > \mathcal{CQ} > \mathcal{LQ}$	SpCM	DL(SpCM_{LTS}(G))	Smallest tight set		UpSafe(G), DownSafe(G), Delay(**SpCM**$_{LTS}$(G)), UpSafe(**DL**(**SpCM**$_{LTS}$(G))), DownSafe(**DL**(**SpCM**$_{LTS}$(G)))

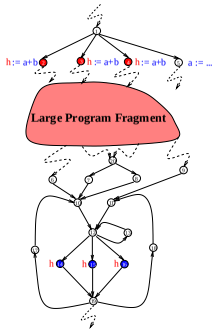
Flexibilität (1)

Das Ausgangsprogramm ...



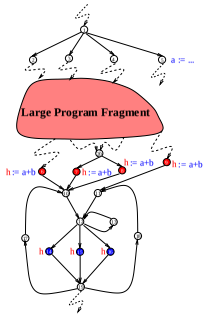
Flexibilität (2)

BCM ... Ein berechnungsoptimales Programm ($\mathcal{CQ}$)



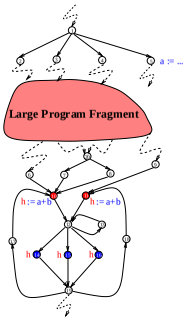
Flexibilität (3)

LCM ... A Computationally & Lifetime Opt. Program ($\mathcal{CQ} > \mathcal{LQ}$)



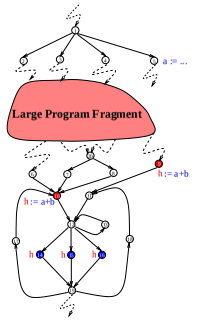
Flexibilität (4)

SpCM ... A Code-Size & Lifetime Optimal Program ($\mathcal{SQ} > \mathcal{LQ}$)

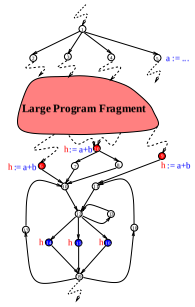


Flexibilität (5)

SpCM ... A Computationally and Lifetime Best Code-Size Optimal Program ($\mathcal{SQ} > \mathcal{CQ} > \mathcal{LQ}$)



SpCM ... A Code-Size and Lifetime Best Computationally Optimal Program ($\mathcal{CQ} > \mathcal{SQ} > \mathcal{LQ}$)



...auf die Entwicklung von PRE:

- 1958: ...*first glimpse of PRE*
  - ⇝ Ershov's work on *On Programming of Arithmetic Operations*.
- < **1979** ... Special Techniques
  - ⇝ Total Redundancy Elimination, Loop Invariant Code Motion
- **1979:** ...*origin of contemporary PRE*
  - ⇝ Morel/Renvoise's seminal work on PRE
- **1992:** ...*LCM* [Knoop et al., PLDI'92]
  - ⇝ ...first to achieve comp. optimality with minimum register pressure
  - ⇝ ...first to rigorously be proven correct and optimal

Ein Rückblick (fortgesetzt)

- **2000:** ...*origin of code-size sensitive PRE* [Knoop et al., POPL 2000]
  - ⇝ ...first to allow prioritization of goals
  - ⇝ ...rigorously be proven correct and optimal
  - ⇝ ...first to bridge the gap between traditional compilation and compilation for embedded systems
- ca. since 1997: ...*a new strand of research on PRE*
  - ⇝ Speculative PRE: Gupta, Horspool, Soffa, Xue, Scholz, Knoop,...
- **2005**: ...*another fresh look at PRE* (*as maximum flow problem*)
  - ⇝ Unifying PRE and Speculative PRE [Jingling Xue and J. Knoop]

Namen sind Nachrichten

Ein anderer Rückblick...

- < **1979** ... Special Techniques
  - ⇝ Total Redundancy Elimination, Loop Invariant Code Motion
- **1979** ... Partial Redundancy Elimination
  - ⇝ **Pioneering** ... Morel/Renvoise's bidirectional algorithm [1979]
  - ⇝ **Heuristic improvements** ... Dhamdhere [1988, 1991], Drechsler/Stadel [1988], Sorkin [1989], Dhamdhere/Rosen/Zadeck [1992], ...
- **1992** ... BCM & LCM [Knoop et al., PLDI'92]
  - ⇝ BCM ... first to achieve Computational Optimality: Earliestness Principle
  - ⇝ LCM ... first to achieve Comp. & Lifetime Optimality: Latestness Principle

  ... first to be purely unidirectional, however, not yet code-size sensitive.
- **2000/2004**: Code-Size Sensitive PRE [Knoop et al., POPL 2000, LCTES 2004]

Warum lohnt es sich, PRE zu betrachten? (1)

Es ist...

- Relevant ...weit verbreitet in der Praxis
- Generell ...eine Familie von Optimierungen denn eine einzelne Optimierung
- Wohlverstanden ...bewiesen korrekt und *optimal*
- Herausfordernd ...konzeptuell einfach, aber weist eine Reihe kopfnussaufgebender Phänomene auf

Warum lohnt es sich, PRE zu betrachten? (2)

Zu guter letzt, PRE ist...

- Wahrlich klassisch ...blickt auf eine lange Geschichte zurück
  - Morel, E. and Renvoise, C. *Global Optimization by Suppression of Partial Redundancies*. CACM 22 (2), 96 - 103, 1979.
  - Ershov, A. P. *On Programming of Arithmetic Operations*. CACM 1 (8), 3 - 6, 1958.

Kap. 7.5 Optimalitätsphänomene

Zurück zu CM im allgemeinen

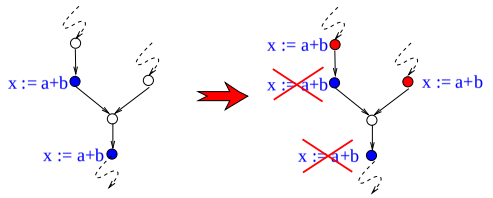
Traditionell

- Code (C) heißt Ausdrücke
- Motion (M) heißt vorziehen

Aber...

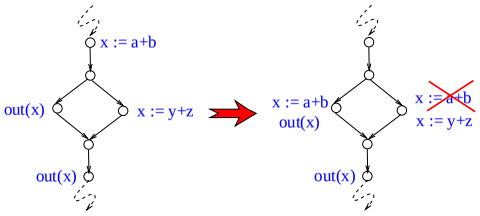
- CM ist mehr als vorziehen von Ausdrücken und PR(E)E!

...Anweisungen sind Code.



- Hier heißt CM Elimination partiell redundanter Anweisungen (PRAE)

...Anweisungen auch verzögert werden.



- CM heißt jetzt Elimination partiell toten Codes (PDCE)

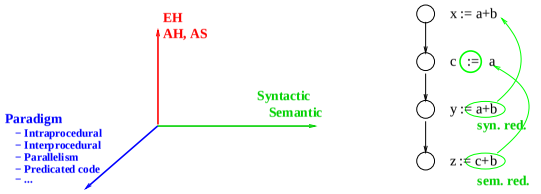
Über den Entwurfsraum von CM-Algorithmen...

Allgemeiner...

- Code heißt Ausdrücke/Anweisungen
- Motion heißt vorziehen/verzögern

Code / Motion	Hoisting	Sinking
Expressions	EH	·/·
Assignments	AH	AS

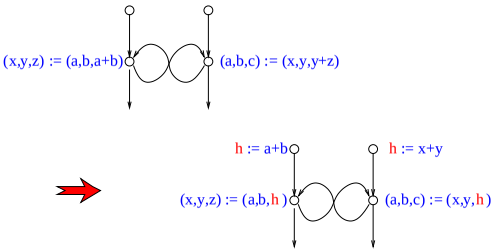
Weitere Verfeinerung des Entwurfsraums von CM-Algorithmen...



Introducing semantics... !

Semantisches Code Motion...

erlaubt mächtigere Optimierungen!



(Beispiel von B. Steffen, TAPSOFT'87)

Optimalitätsergebnisse sind sehr empfindlich!

Drei Beispiele sollen dies belegen...

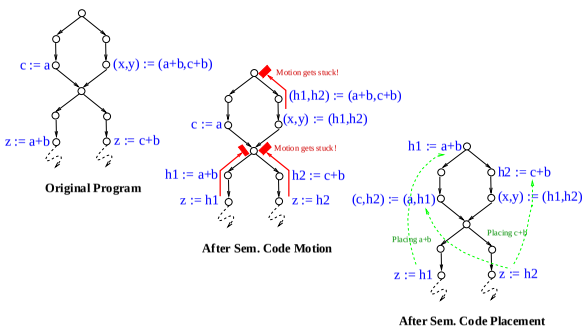
(I) Code motion vs. code placement

(II) Abhängigkeiten elementarer Transformationen

(III) Paradigmenabhängigkeiten

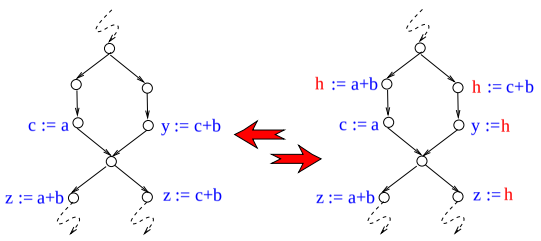
(I) Code Motion vs. Code Placement

...sind keine Synonyme!



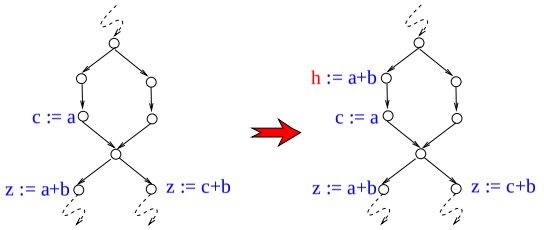
Schlechter noch...

Optimalität ist verloren!

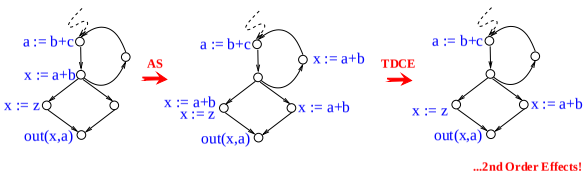


Incomparable!

Performanz kann verloren gehen, wenn naiv angewandt!



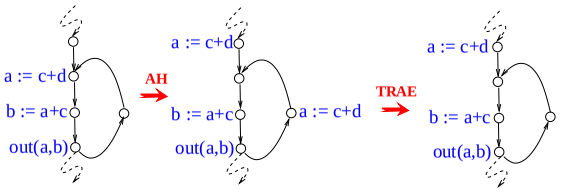
(II) Abhängigkeiten von Transformationen



...2nd Order Effects!

⇝ ...Partial Dead-Code Elimination (PDCE)

Abhängigkeiten von Transformationen



...2nd Order Effects!

⇝ ...Partially Redundant Assignment Elimination (PRAE)

Konzeptuell

...können wir PREE, PRAE und PDCE wie folgt verstehen:

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

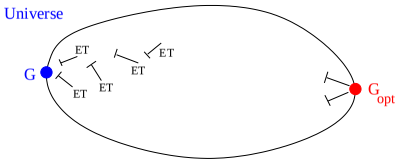
PRAE/PDCE – Optimalitätsergebnisse

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

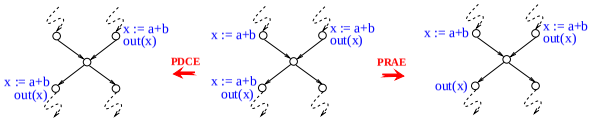


Betrachte jetzt...

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

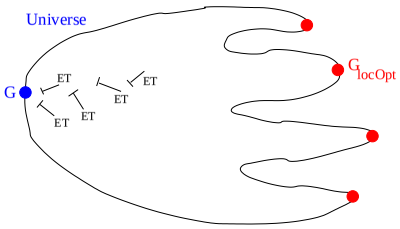
...sollte noch mächtiger sein!

In der Tat, aber...



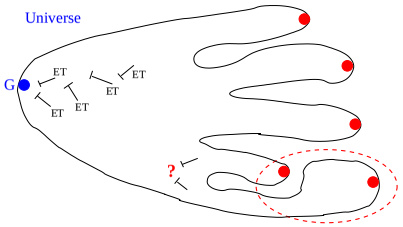
Konfluenz...

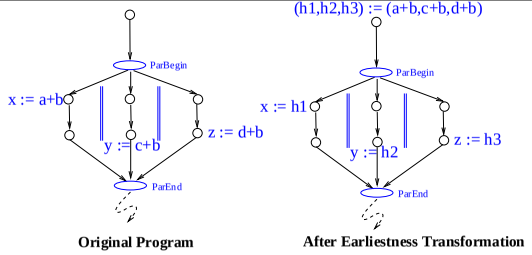
...und folglich (globale) Optimalität ist verloren!



Noch schlechter...

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





Original Program **After Earliestness Transformation**

...ein naiver Transfer der Transformationsstrategie führt hier zu einem im wesentlichen sequentiellen Programm!

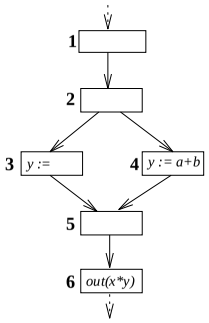
Kap. 7.5 Partial Dead-Code und Faint Code Elimination

Elimination partiell toten/geisterhaften Codes

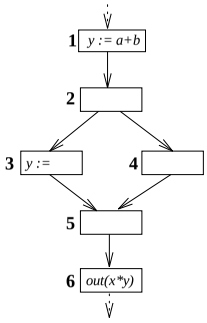
Wir unterscheiden zwei Spielarten...

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

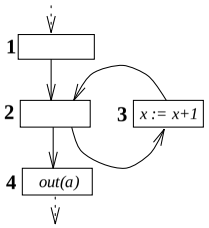
Ein Beispiel: Das Ausgangsprogramm



Ein Beispiel: Nach PDCE



Geisterhaft, aber nicht tot: Ein Beispiel



PDCE/PDFE: Anweisungsmuster

Bezeichne in der Folge...

- α ein sog. *Anweisungsmuster*:

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

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

PDCE/PDFE: Verzögerbarkeit von Anweisungen (1)

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.

PDCE/PDFE: Anweisungen (2)

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.

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.

PDCE/PDFE: Elimination von Anweisungen (1)

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

PDCE/PDFE: Zulässige Elimination von Anweisungen (2)

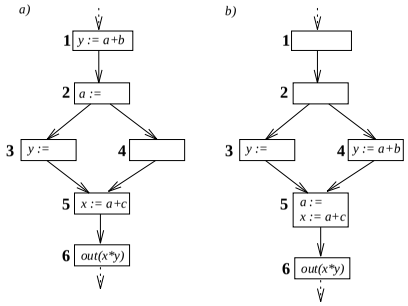
Definition [Dead (Faint) Code Elimination]
A *dead (faint) code elimination* for an assignment pattern α is an assignment elimination for α, where some dead (faint) occurrences of α are eliminated.

PDCE/PDFE: Ein neues Phänomen

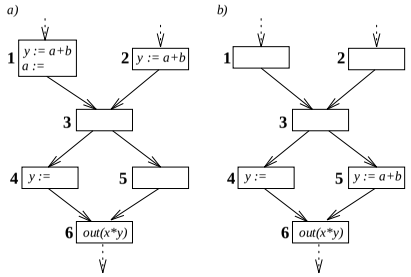
...sog. *second order* Effekte:

- *Sinking-Elimination* Effekte
- *Sinking-Sinking* Effekte
- *Elimination-Sinking* Effekte
- *Elimination-Elimination* Effekte

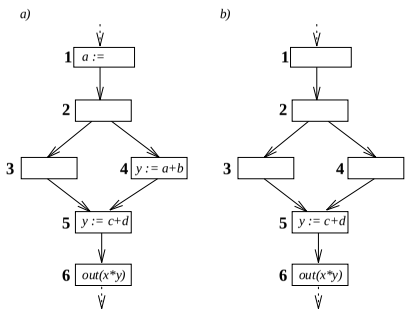
Ein Beispiel für "Sinking-Sinking"-Effekte

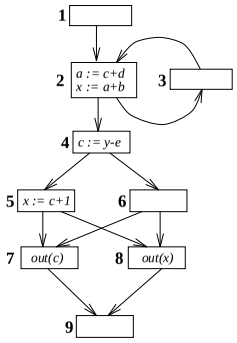


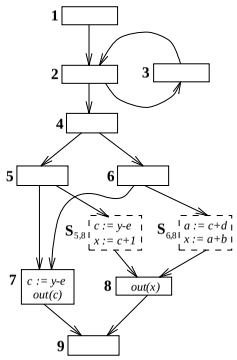
Ein Beispiel für "Elimination-Sinking"-Effekte



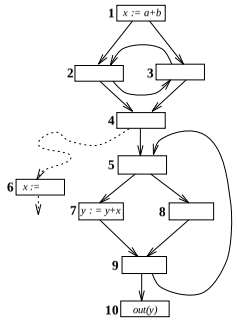
Ein Beispiel für "Elimination-Elimination"-Effekte



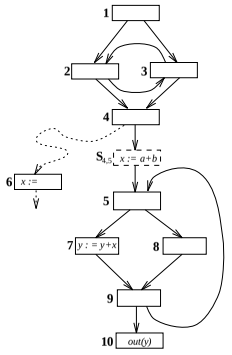




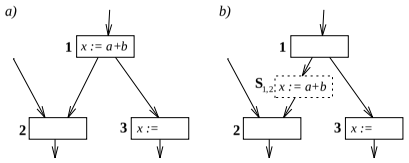
Auch für PDCE/PFCE: Spalten kritischer Kanten erforderlich (1)



Auch für PDCE/PFCE: Spalten kritischer Kanten erforderlich (2)



Erinnerung: Kritische Kanten



PDCE/PDFE: Die Definition

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

- $G \vdash_{PDE} G'$ bzw. $G \vdash_{PFE} G'$:
  ...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*.

PDCE/PDFE: Optimalität

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

1. Let $G', G'' \in \mathcal{G}_\tau$. Then G' is *better* than G'', in signs $G'' \sqsubseteq G'$, 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 $\mathcal{G}_\tau$.

PDCE/PDFE: Relation "besser"

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

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

Sei

- $\vec{\sqsubseteq}_\tau =_{df} (\sqsubseteq\!\!\!\!\sim \cap \vdash_\tau)^*$, und sei
- $\mathcal{F}_\tau \subseteq \{f \mid f : \mathcal{G}_\tau \to \mathcal{G}_\tau\}$ eine endliche Familie von Funktionen mit
  1. *Monotonicity:*
     $\forall G', G'' \in \mathcal{G}_\tau \; \forall f \in \mathcal{F}_\tau.$
     $G' \vec{\sqsubseteq}_\tau G'' \Rightarrow f(G') \vec{\sqsubseteq}_\tau f(G'')$
  2. *Dominance:*
     $\forall G', G'' \in \mathcal{G}_\tau.\; G' \vdash_\tau G'' \Rightarrow \exists f \in \mathcal{F}_\tau.\; G'' \vec{\sqsubseteq}_\tau f(G')$

PDCE/PDFE: Existenz optimaler Programme

Theorem [Existence of Optimal Programs]
$\mathcal{G}_\tau$ 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.

Proof ...by means of a generalized version of the fixed point theorem of Knaster/Tarski (cf. Geser, Knoop, Lüttgen et al. (CC'96))

PDCE/PDFE: Existenz optimaler Programme

Bemerkungen:

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

Die PDCE/PDFE-Analysen (1)

Lokale Prädikate für *Dead Code Elimination* (*DCE*) und *Faint Code Elimination* (*FCE*)...

- $\mathrm{USED}_\iota(x)$: x is a right-hand side variable of the instruction ι.*
- $\mathit{Rel\text{-}Used}_\iota(x)$: x is a right-hand side variable of the relevant instruction ι.
- $\mathit{Ass\text{-}Used}_\iota(x)$: 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 PDCE/PDFE-Analysen (2)

Die DCE-Analyse...

The Dead Variable Analysis:

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

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

Die PDCE/PDFE-Analysen (3)

Die FCE-Analyse...

The Faint Variable Analysis: (Slotwise simultaneously for all variables x)

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

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

Die PDCE/PDFE-Analysen (4)

Lokale Prädikate für *Assignment Sinking* (*AS*)...

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

Die AS-Analyse, im Kern eine Verzögerbarkeitsanalyse...

Delayability Analysis:

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

$$\text{X-DELAYED}_n = \text{LOC-DELAYED}_n + \text{N-DELAYED}_n * \neg\text{LOC-BLOCKED}_n$$

Die PDCE/PDFE-Analysen (6)

Die aus der AS-Analyse resultierenden Einsetzungspunkte...

Insertion Points:

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

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

PDCE/PDFE: Hauptresultate (1)

Bezeichnen

- *pdce* und
- *pfce*

die aus vorigen Analysen abgeleiteten Algorithmen für die

- *Elimination partiell toter Anweisungen* und
- *Elimination partiell geisterhafter Anweisungen*

PDCE/PDFE: Hauptresultate (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 (3)

Theorem [Correctness]

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

PDCE/PDFE: Hauptresultate (4)

Theorem [Optimality Theorem]

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

Kapitel 7.7 Assignment Motion

Elimination partiell redundanter Anweisungen

Auch hier neue Phänomene...

Wir können unterscheiden:

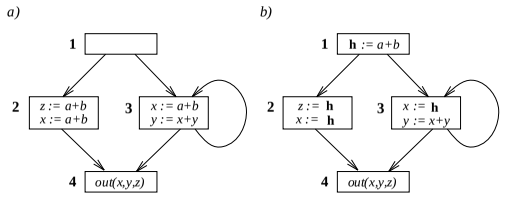
- (Pure) Expression Motion (PuEM) (bzw. PuPREE)
- (Pure) Assignment Motion (PuAM) (bzw. PuPRAE)

...sowie

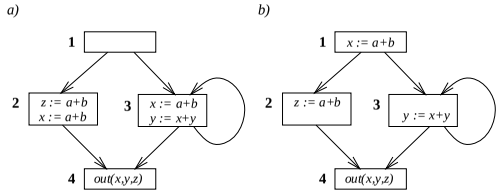
- Uniform Expression and Assignment Motion (AM)

Dazu einige Beispiele...

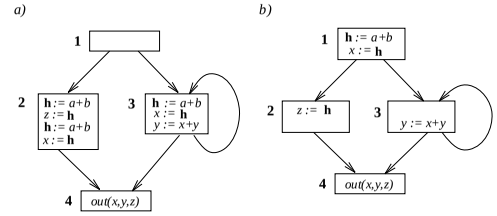
Expression Motion (PuEM)



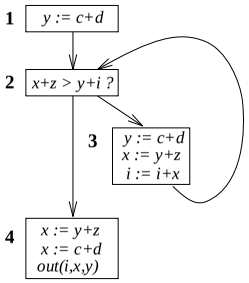
Assignment Motion (PuAM)



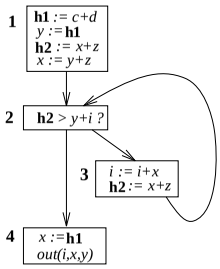
Uniform Expression and Assignment Motion (AM)



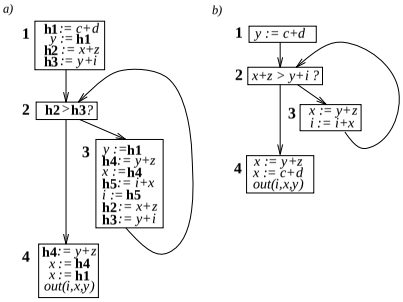
Ein komplexeres Beispiel: Ausgangsprogramm



Ein komplexeres Beispiel: Optimiertes Programm



Zum Vergleich: Separate Effekte von PuEM & PuAM



AM: Der einheitliche PuEM/PuAM-Algorithmus

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

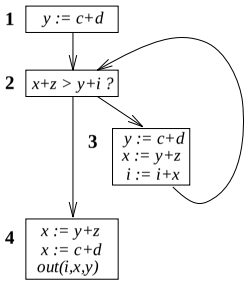
Bemerkung

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

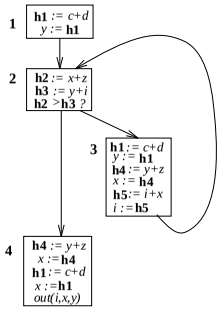
AM: Ein schon bekanntes Phänomen

...sog. *second order* Effekte:

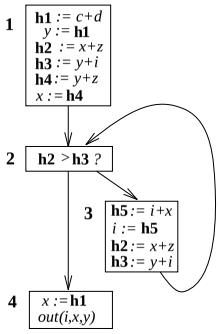
- *Hoisting-Hoisting* Effekte
- *Hoisting-Elimination* Effekte
- *Elimination-Elimination* Effekte



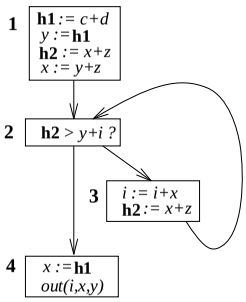
Nach dem Präprozess



Nach dem Hauptprozess



Das Endresultat: Nach dem Postprozess



Hauptergebnisse (1)

Analog zu PDCE/PFCE...

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

Hauptergebnisse (2)

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.

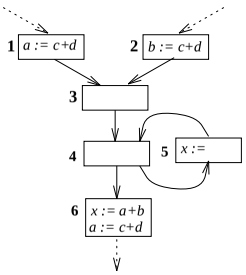
Hauptergebnisse (3)

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.

Hauptergebnisse (4)

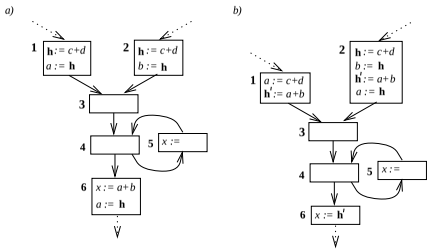
Theorem [Relative Temporary-Optimality]
$G_{GlobAlg}$ is relatively temporary-optimal in $\mathcal{G}$, i.e. it is impossible to decrease the number of assignments to temporaries or the length of temporary lifetimes in $G_{GlobAlg}$ by a corresponding assignment sinking.

Betrachte dazu...

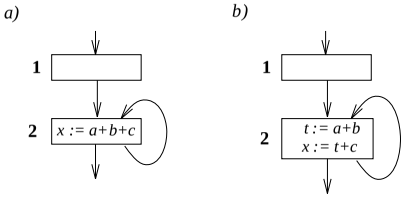


Warum nur “relative Optimalität” (2)

...und folgende zwei unvergleichbare Transformationsresultate:

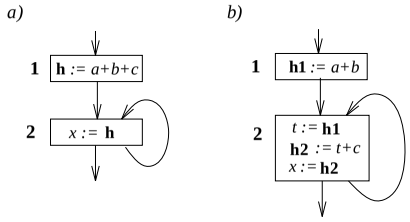


Andere Phänomene: Komplex- vs. 3-Adresscode (1)



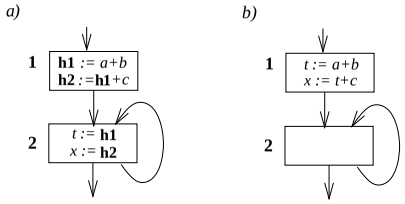
Andere Phänomene: Komplex- vs. 3-Adresscode (2)

Der Effekt von PuEM...



Andere Phänomene: Komplex- vs. 3-Adresscode (3)

Die Effekte von (PuEM + Copy Propagation) und von (Uniform PuEM&PuAM)...



Kap. 7.8 Fazit und Literaturhinweise

Ein Fazit bzw...

Die Frage nach dem Sinn des Lebens, was haben wir alles erreicht bzw...

- Was haben wir alles betrachtet?
  Das wenigste!

Oder umgekehrt...

- Was haben wir alles nicht betrachtet?
  Das meiste!

- *Erweiterungen syntaktischer PRE neben PDCE/PRAE*
  - Lazy Strength Reduction
  - ...
- *Semantische Erweiterungen*
  - Semantic Code Motion/Code Placement
  - Semantic Strength Reduction
  - ...
- *Sprachausweitungen*
  - Interprozeduralität
  - Parallelität
  - ...

Insbesondere nicht (oder nicht im Detail) (1)

- *Dynamische, profilgestützte Erweiterungen*
  - Spekulative PRE
  - ...
- ...

Literaturhinweise (1)

- *Syntaktische PRE*
  - Knoop, J., Rüthing, O., and Steffen, B. Retrospective: Lazy Code Motion. In "20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation (1979 - 1999): A Selection", ACM SIGPLAN Notices 39, 4 (2004), 460 - 461 & 462-472.
  - Knoop, J., Rüthing, O., and Steffen, B. Optimal code motion: Theory and practice. ACM Transactions on Programming Languages and Systems 16, 4 (1994), 1117 - 1155.
  - Rüthing, O., Knoop, J., and Steffen, B. Sparse code motion. In Conference Record of the 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000) (Boston, MA, Jan. 19 - 21, 2000), ACM New York, (2000), 170 - 183.

Literaturhinweise (2)

- *Elimination partiell toten Codes*
  - Knoop, J., Rüthing, O., and Steffen, B. Partial dead code elimination. In Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI'94) (Orlando, FL, USA, June 20 - 24, 1994), ACM SIGPLAN Notices 29, 6 (1994), 147 - 158.
- *Elimination partiell redundanter Anweisungen*
  - Knoop, J., Rüthing, O., and Steffen, B. The power of assignment motion. In Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI'95) (La Jolla, CA, USA, June 18 - 21, 1995), ACM SIGPLAN Notices 30, 6 (1995), 233 - 245.

Literaturhinweise (3)

- *BB- vs. EA-Graphen*
  - Knoop, J., Koschützki, D., and Steffen, B. Basic-block graphs: Living dinosaurs? In Proceedings of the 7th International Conference on Compiler Construction (CC'98) (Lisbon, Portugal, March 30 - April 3, 1998), Springer-Verlag, Heidelberg, LNCS 1383 (1998), 65 - 79.
- *Schieben vs. Platzieren*
  - Knoop, J., Rüthing, O., and Steffen, B. Code motion and code placement: Just synonyms? In Proceedings of the 7th European Symposium On Programming (ESOP'98) (Lisbon, Portugal, March 30 - April 3, 1998), Springer-Verlag, Heidelberg, LNCS 1381 (1998), 154 - 169.

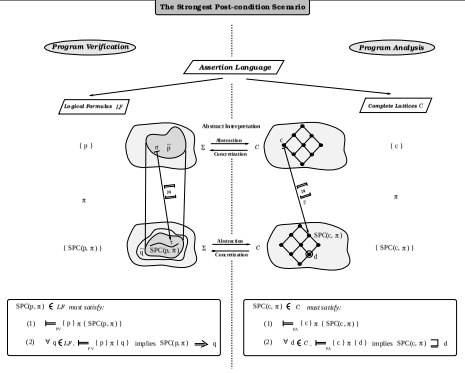
Literaturhinweise (4)

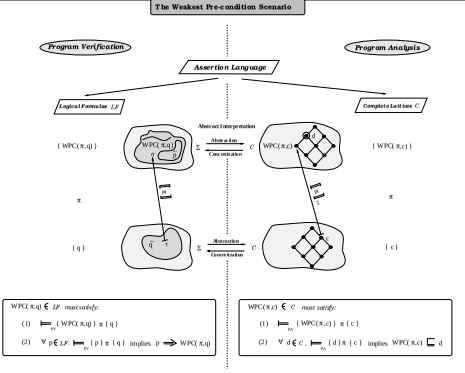
- *Spekulative vs. klassische PRE*
  - Scholz, B., Horspool, N. and Knoop, J. Optimizing for space and time usage with speculative partial redundancy elimination. In Proceedings of the ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2004) (Washington, DC, June 11 - 13, 2004), ACM SIGPLAN Notices 39, 7 (2004), 221 -230.
  - Xue, J., Knoop, J. A fresh look at PRE as a maximum flow problem. In Proceedings of the 15th International Conference on Compiler Construction (CC 2006) (Vienna, Austria, March 25 - April 2, 2006), Springer-Verlag, Heidelberg, LNCS 3923 (2006), 139 - 154.

Literaturhinweise (5)

- *Weitere Techniken und spezielle Verfahren*
  - Geser, A., Knoop, J., Lüttgen, G., Rüthing, O., and Steffen, B. Non-monotone fixpoint iterations to resolve second order effects. In Proceedings of the 6th International Conference on Compiler Construction (CC'96) (Linköping, Sweden, April 24 - 26, 1996), Springer-Verlag, Heidelberg, LNCS 1060 (1996), 106 - 120.
  - Knoop, J., and Mehofer, E. Optimal distribution assignment placement. In Proceedings of the 3rd European Conference on Parallel Processing (Euro-Par'97) (Passau, Germany, August 26 - 29, 1997), Springer-V., Heidelberg, LNCS 1300 (1997), 364 - 373.
  - Knoop, J., Rüthing, O., and Steffen, B. Lazy strength reduction. Journal of Programming Languages 1, 1 (1993), 71 - 91.
  - ...: siehe auch `www.complang.tuwien.ac.at/knoop`

Kapitel 8 Programmverifikation und -analyse im Vergleich





Kapitel 9 Reverse Datenflussanalyse

Kapitel 9.1 Grundlagen

Reverse abstrakte Semantik

Reverse abstrakte Semantik

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

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

wobei $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ eine abstrakte Semantik auf $\mathcal{C}$ ist.

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

Sprechweise in der Theorie "Abstrakter Interpretation":

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

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

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. $MOP_{(G, c_{\mathbf{s}})}(q) = MOP_{(G, c_{\mathbf{s}})}(\mathbf{q})$

Kapitel 9.1.1 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 9.1.2 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 **??**), 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)

(Prologue: Initialization of the annotation array *reqInf*, and the variable *workset*)

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

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

(Main process: Iterative fixed point computation)

WHILE *workset* $\neq \emptyset$ DO
  CHOOSE $m \in$ *workset*;
  workset := *workset*$\backslash\{m\}$;
  (Update the predecessor-environment of node m)
  FORALL $n \in pred(m)$ DO
    join := $[\![\ (n,m)\ ]\!]_R$(*reqInf*[m]) $\sqcup$ *reqInf*[n];
    IF *reqInf*[n] $\sqsubset$ *join*
      THEN
        reqInf[n] := *join*;
        workset := *workset* $\cup \{n\}$
    FI
  OD
  ESOOHC
OD.

Kapitel 9.1.3 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

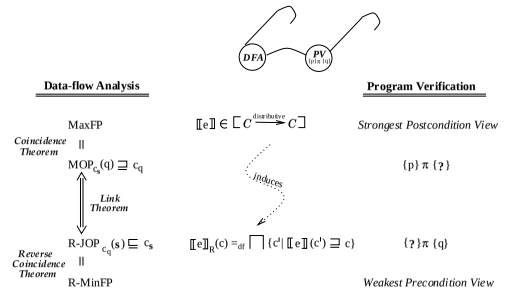
Diee R-*MinFP*-Lösung stimmt mit der R-*JOP*-Lösung überein, d.h.,

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

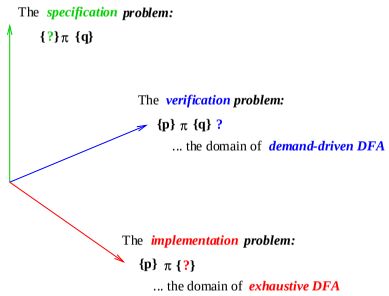
falls $[\![\ ]\!]$ distributiv ist.

Kapitel 9.2 DFA, RDFA und DD-DFA

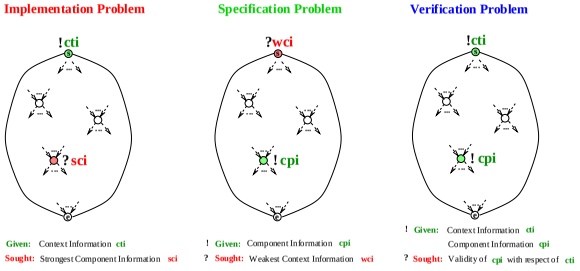
DFA vs. Verifikation: Überblick



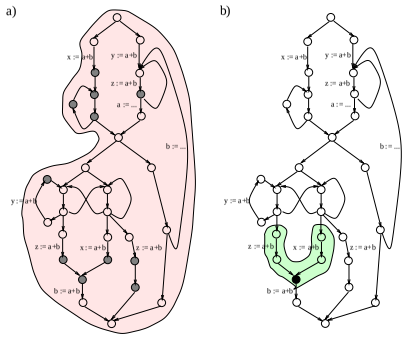
Drei unterschiedliche Problemperspektiven (1)



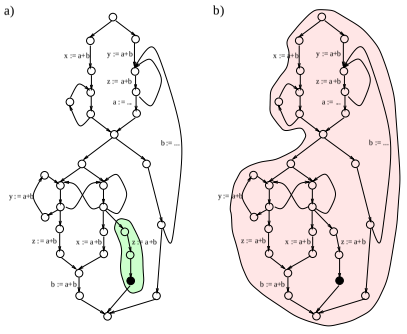
Drei unterschiedliche Problemperspektiven (2)



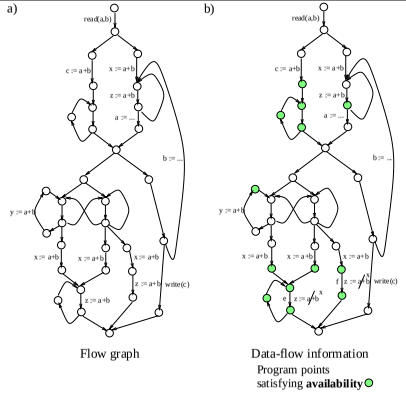
Bsp: Verfügbarkeit an einem Punkt (1)



Bsp: Verfügbarkeit an einem Punkt (2)



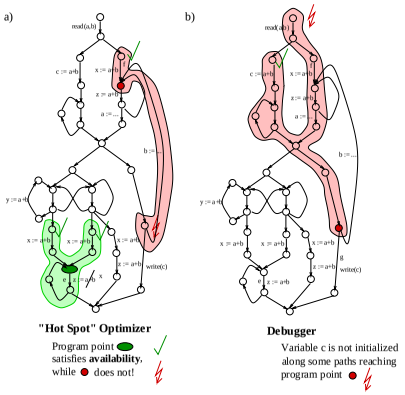
Kapitel 9.3 RDFA-Anwendungen



Flow graph

Data-flow information
Program points satisfying **availability**

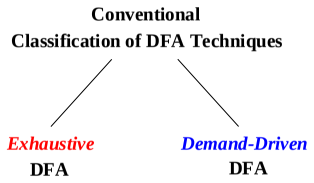
Anwendung: “Hot Spot” Optimierer und Debugger (2)



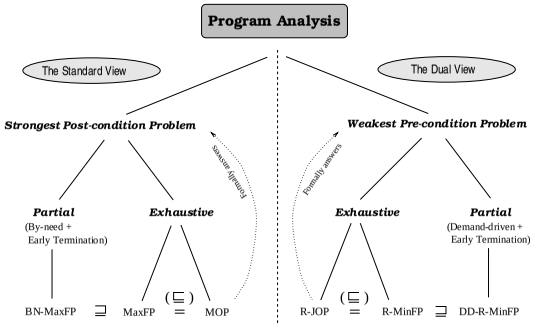
"Hot Spot" Optimizer
Program point satisfies **availability**, while does not!

Debugger
Variable c is not initialized along some paths reaching program point

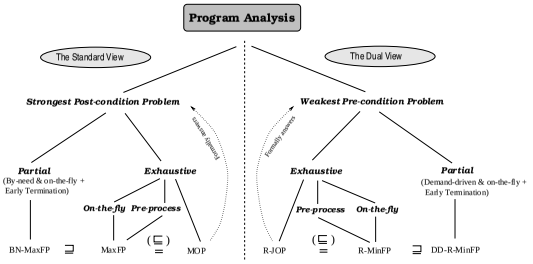
Erschöpfende vs. anforderungsgetriebene DFA (1)



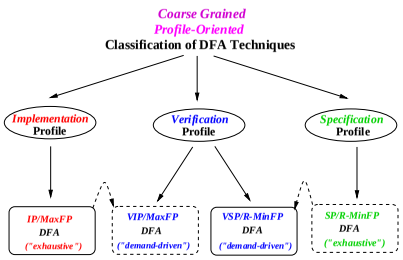
Erschöpfende vs. anforderungsgetriebene DFA (2)



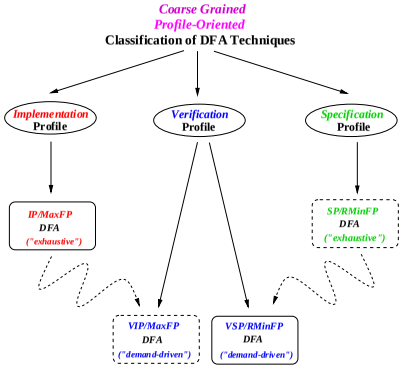
Erschöpfende vs. anforderungsgetriebene DFA (3)

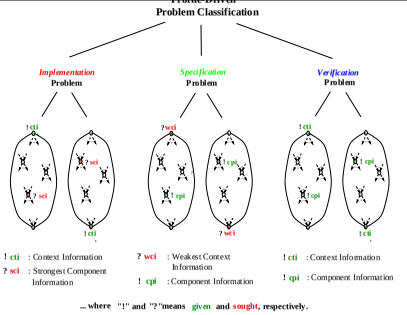


Eine andere Sicht (1)

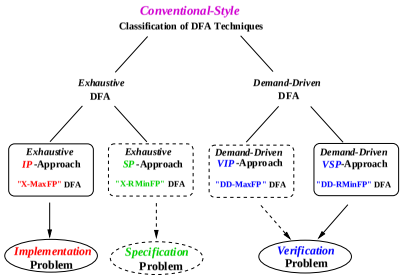


Eine andere Sicht (2)

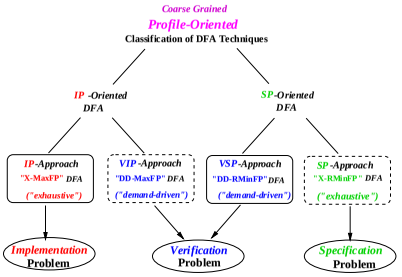




(1)



Zum Abschluss: Problemorientiert (2)



Hot-Spot Program Optimization

...in größerem Detail: Siehe Zusatzfolien!