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)

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

In der Folge

- 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

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 1.2 Die Programmiersprache WHILE

Die Programmiersprache WHILE

...die Sprache 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 arithmetischer und Boolescher Ausdrücke

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

Semantik arithmetischer Ausdrücke (1)

$[\![.]\!]_A$: **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.

Semantik arithmetischer Ausdrücke (2)

$[\![.]\!]_N$: **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, $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, ... 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`.

Kapitel 1.4 Syntaktische und semantische Substitution

Freie Variablen

...arithmetischer Ausdrücke:

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

...Boolescher Ausdrücke:

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

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

Lemma 1.1
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 1.2
Seien $b \in$ **BExpr** und $\sigma, \sigma' \in \Sigma$ mit $\sigma(x) = \sigma'(x)$ für alle $x \in FV(b)$. Dann gilt:

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

Syntaktische/Semantische Substitution

Von zentraler Bedeutung...

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

Syntaktische Substitution

Definition 1.3

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

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 1.5 (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

Exkurs: Induktive Beweisprinzipien (1)

Zentral:

- Vollständige Induktion
- Verallgemeinerte Induktion
- Strukturelle Induktion

...zum Beweis einer Aussage A.

Exkurs: Induktive Beweisprinzipien (2)

Zur Erinnerung hier wiederholt:

Die Prinzipien der...

- *vollständigen Induktion*

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

- *verallgemeinerten Induktion*

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

- *strukturellen Induktion*

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

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

Beispiel: Beweis von Lemma 1.1 (1)

...durch strukturelle Induktion

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

Induktionsanfang:

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

Mit den Definitionen von $[\![\ ]\!]_A$ und $[\![\ ]\!]_N$ erhalten wir unmittelbar wie gewünscht:

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

Beispiel: Beweis von Lemma 1.1 (2)

Fall 2: Sei $a \equiv x$, $x \in$ **Var**.

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

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

Beispiel: Beweis von Lemma 1.1 (3)

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

Vereinbarung

Seien in der Folge 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 2 Operationelle Semantik von WHILE

Kapitel 2.1 Strukturelle Operationelle Semantik

Strukturell operationelle Semantik

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

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.

Strukturell operationelle Semantik

...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)

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

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

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

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

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

SOS-Regeln von WHILE (2)

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

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

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

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 $[\![b]\!]_B(\sigma) = \texttt{ff}$ 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.

Beispiel (1)

Sei

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

Betrachte

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

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

  induzierte Berechnungsfolge

Beispiel (2)

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

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

Beispiel (3)

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

Beispiel (4)

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

Beispiel (Detailbetrachtung) (5)

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

steht vereinfachend für...

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

Beispiel (Detailbetrachtung) (6)

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

steht vereinfachend für...

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

Beispiel (Detailbetrachtung) (7)

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

steht vereinfachend für...

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

Determinismus der SOS-Regeln

Lemma 1.6

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

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

Korollar 1.7
Die von den SOS-Regeln für eine Konfiguration induzierte Berechnungsfolge ist eindeutig bestimmt, d.h. *deterministisch*.

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

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

Korollar 1.7 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 1.8

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

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

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)

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

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

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

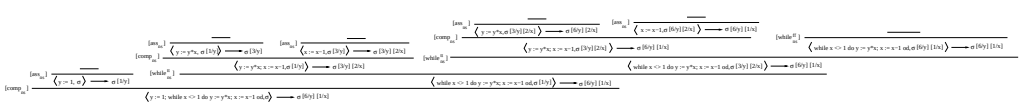
$$[\text{while}^{ff}_{ns}] \quad \frac{}{\langle \text{while } b \text{ do } \pi \text{ od},\sigma\rangle \to \sigma} \qquad [\![\, b \,]\!]_B(\sigma) = \texttt{ff}$$

Beispiel zur natürlichen Semantik (1)

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

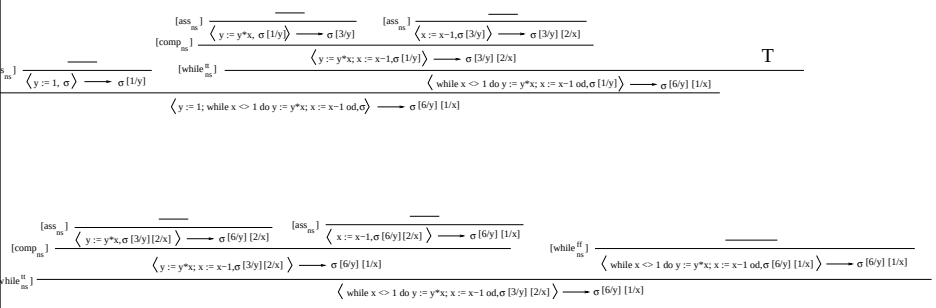
Dann gilt:

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



Beispiel zur natürlichen Semantik (2)

Das gleiche Beispiel in etwas gefälligerer Darstellung:



Determinismus der NS-Regeln

Lemma 2.1

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

Korollar 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 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 2.1**!

Kap. 2.3 Konzeptueller Vergleich strukturell operationeller und natürlicher Semantik

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...

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

  ...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

Denotationelle Semantik (1)

...auch für das Beispiel von WHILE:

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

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

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

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

$[\![$ if b then π_1 else π_2 fi $]\!]_{ds} = cond([\![b]\!]_{\mathcal{B}}, [\![\pi_1]\!]_{ds}, [\![\pi_2]\!]_{ds})$

$[\![$ while b do π 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 2.2
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.

Hauptergebnisse

Theorem

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

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:

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

definiert durch

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

WHILE – Denotationelle Semantik (1)

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

Denotationelle Semantik

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

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...

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

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!)

Schrittweise zur denotationellen Semantik der while-Schleife

Dazu folgende Beobachtung...

- $\textbf{while } b \textbf{ do } \pi \textbf{ od}$ muss dieselbe Bedeutung haben wie...
  $\textbf{if } b \textbf{ then } (\pi;\ \textbf{while } b \textbf{ do } \pi \textbf{ od}) \textbf{ else } skip \textbf{ fi}$

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.

Etwas formaler: Unser Arbeitsplan

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

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

Ordnung auf Zustandstransformationen

Bezeichne...

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

Wir definieren...

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

Lemma 1

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

Ordnung auf Zustandstransformationen

Sogar...

Lemma 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

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

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 $m \notin M_f$

Stetigkeitkeitsresultate (1)

Lemma 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*)

Stetigkeitkeitsresultate (2)

Lemma 4
Sei $g_0 \in [\Sigma \to \Sigma]$ und sei F definiert durch

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

Dann gilt: F ist stetig.

Zusammen mit...

Lemma 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!

Und somit wie anfangs angedeutet...

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!

Nachträge

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

Eindeutigkeit von Schranken

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

Verbände und vollständige Verbände

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

Dann heißt $(P, \sqsubseteq)$...

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

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

Vollständige partielle Ordnungen

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

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

Dann heißt $(P, \sqsubseteq)$...

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

Es gilt:

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

Ketten

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

Eine Teilmenge $K \subseteq P$ heißt...

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

Eine Kette K heißt...

- *endlich*, wenn K endlich ist, sonst *unendlich*.

Kettenendlichkeit, endliche Elemente

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

- *kettenendlich* gdw. P enthält keine unendlichen Ketten

Ein Element $p \in P$ heißt

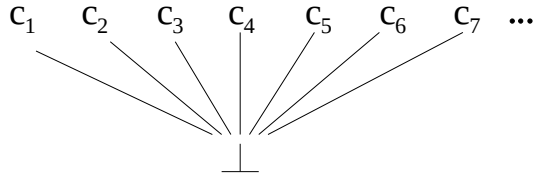
- *endlich* gdw. die Menge $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$ keine unendliche Kette enthält
- *endlich relativ zu* $r \in P$ gdw. die Menge $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$ keine unendliche Kette enthält

(Standard-) CPO-Konstruktionen 1(4)

Flache CPOs...

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

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



(Standard-) CPO-Konstruktionen 2(4)

Produktkonstruktionen...

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

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

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

  CPOs.

(Standard-) CPO-Konstruktionen 3(4)

Summenkonstruktion...

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

- die *direkte Summe* $(\oplus P_i, \sqsubseteq)$ mit...
  - $(\oplus P_i, \sqsubseteq) = (P_1 \dot\cup P_2 \dot\cup \ldots \dot\cup P_n, \sqsubseteq)$ disjunkte Vereinigung der P_i, $i \in \{1, \ldots, n\}$ und $\forall p, q \in \oplus P_i.\ p \sqsubseteq q \Rightarrow \exists i \in \{1, \ldots, n\}.\ p, q \in P_i \wedge p \sqsubseteq_i q$ und der Identifikation der kleinsten Elemente der $(P_i, \sqsubseteq_i)$, $i \in \{1, \ldots, n\}$, d.h. $\bot =_{df} \bot_i$, $i \in \{1, \ldots, n\}$

  eine CPO.

(Standard-) CPO-Konstruktionen 4(4)

Funktionenraum...

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

Dann ist auch...

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

Funktionen auf CPOs / Eigenschaften

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

Dann heißt f...

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

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

Dann heißt f...

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

Funktionen auf CPOs / Resultate

Mit den vorigen Bezeichnungen gilt...

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

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

(Kleinste und größte) Fixpunkte 1(2)

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

Dann heißt c ...

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

Ein Fixpunkt c von f heißt...

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

(Kleinste und größte) Fixpunkte 2(2)

Seien $d, c_d \in C$. Dann heißt c_d ...

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

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

Fixpunktsatz

Theorem (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 1(4)

Zu zeigen: μf...

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

Beweis des Fixpunktsatzes 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(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 4(4)

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

Bedingte Fixpunkte

Theorem (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 (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

Kap. 4 Axiomatische Semantik von WHILE

Kapitel 4.1 Partielle und totale Korrektheit

Axiomatische Semantik von WHILE

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

Definition partieller Korrektheit

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

Ein 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:

Ein 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

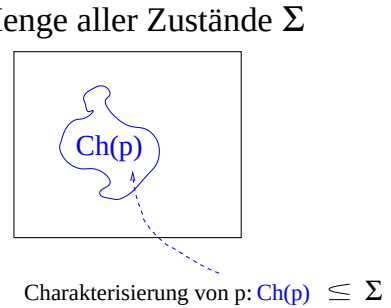
...an einige Sprechweisen:

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.

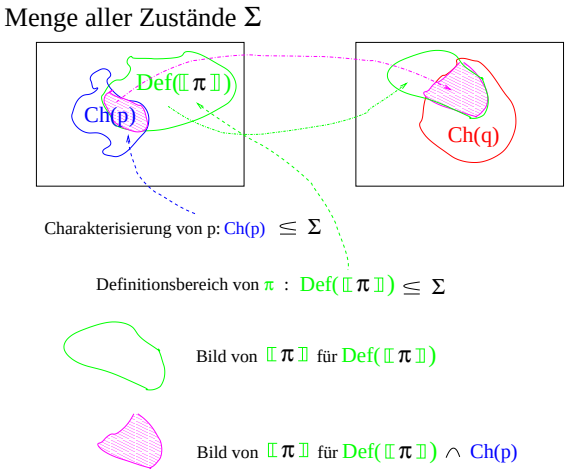
Veranschaulichung (1)

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



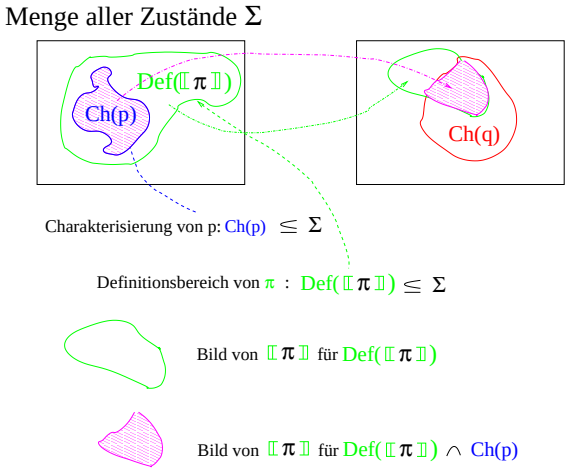
Veranschaulichung (2)

...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:



Kapitel 4.2 Kalküle für partielle und totale 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] $\dfrac{p \Rightarrow p_1,\ \{p_1\}\ \pi\ \{q_1\},\ q_1 \Rightarrow q}{\{p\}\ \pi\ \{q\}}$

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).

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 später.

Kapitel 4.4 Anwendungsbeispiele zum Nachweis partieller Korrektheit

Beispiele 1(2)

...Beweis partieller Korrektheit von Hoareschen Zusicherungen anhand zweier Programme zur Berechnung

- der Fakultät und
- der ganzzahligen Division mit Rest

Beispiele 2(2)

Im Detail:

Beweise, dass die beiden Hoareschen Zusicherungen

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

und

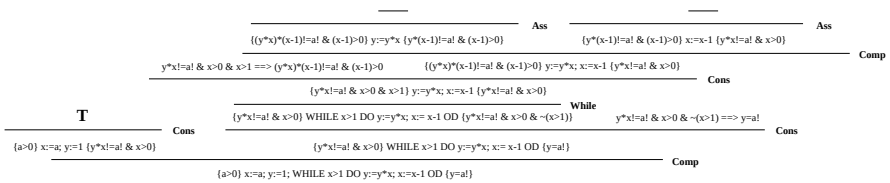
$$\{x \geq 0 \wedge y > 0\}$$
$$q := 0; r := x; \text{while } r \geq y \text{ do } q := q + 1; r := r - y \text{ od}$$
$$\{x = q * y + r \wedge 0 \leq r < y\}$$

gültig sind im Sinne partieller Korrektheit.

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

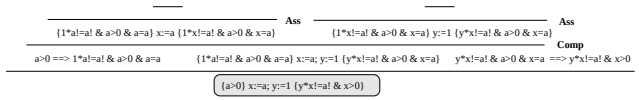
Bew. part. Korrektheit: Fakultät (1)

Erster Beweis



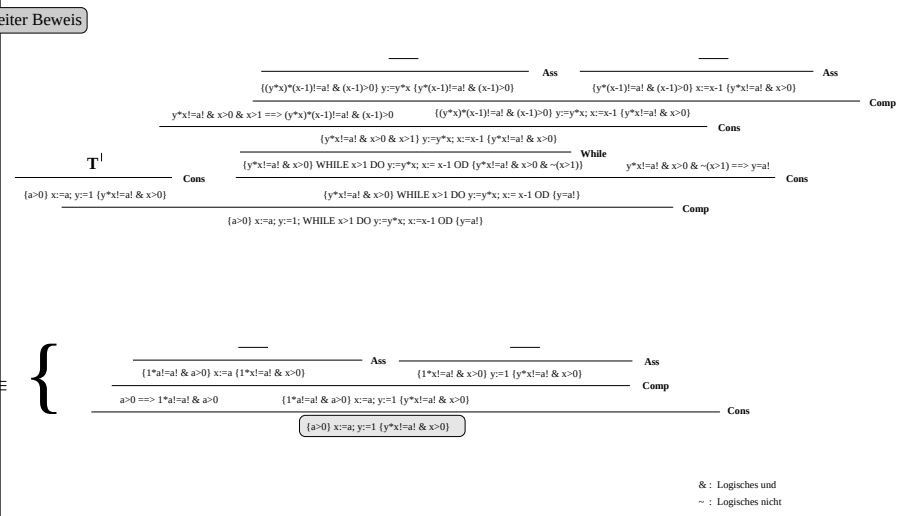
wobei

T ≡

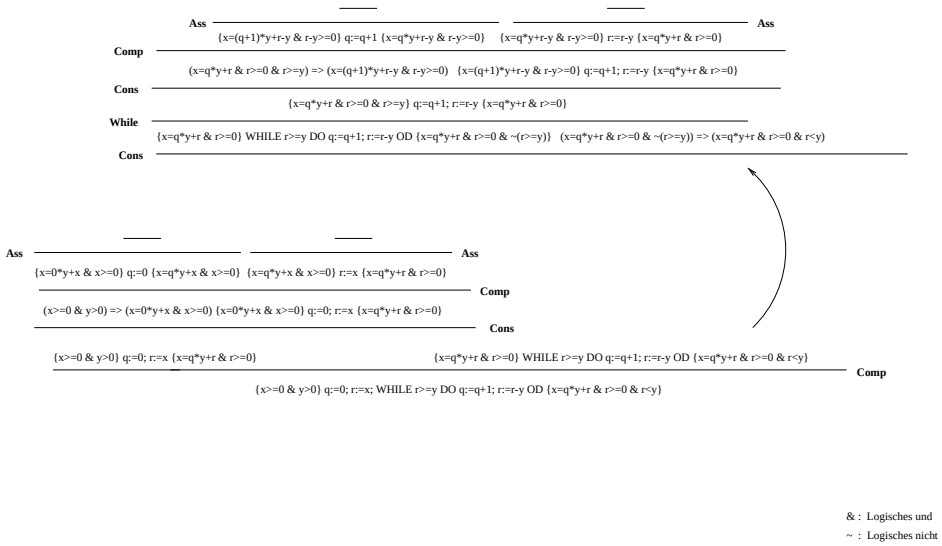


& : Logisches und
~ : Logisches nicht

Bew. part. Korrektheit: Fakultät (2)



Bew. partieller Korrektheit: Division



Bew. part. Korrektheit: Fakultät (1)

- 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*.

Bew. part. Korrektheit: Fakultät (2)

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!

Bew. part. Korrektheit: Fakultät (3)

Schritt 1

"Träumen" der Invariante...

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

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

Bew. part. Korrektheit: Fakultät (4)

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:

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

Bew. part. Korrektheit: Fakultät (5)

Behandlung des Rumpfs der while-Schleife im Detail:

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

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

Bew. part. Korrektheit: Fakultät (6)

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

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

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

Bew. part. Korrektheit: Fakultät (7)

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!

Bew. part. Korrektheit: Fakultät (8)

Schluss der "Beweislücke" in der zugrundeliegenden Theorie:

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

Bew. part. Korrektheit: Fakultät (9)

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

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

Bew. part. Korrektheit: Fakultät (10)

Schritt 3

Zur gewünschten Nachbedingung verbleibt offenbar ebenfalls eine Beweislücke:

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

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

Bew. part. Korrektheit: Fakultät (11)

Schluss der Beweislücke in der zugrundeliegenden Theorie:

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

Bew. part. Korrektheit: Fakultät (12)

Aus Platzgründen etwas verkürzt dargestellt:

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

Bew. part. Korrektheit: Fakultät (13)

Schritt 4

Es verbleibt, die Beweislücke zur gewünschten Vorbedingung zu schließen:

$$\{a > 0\}$$

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

Bew. part. Korrektheit: Fakultät (14)

Einmalige Anwendung der [ass]-Regel liefert:

$$\{a > 0\}$$

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

Bew. part. Korrektheit: Fakultät (15)

Abermalige Anwendung der [ass]-Regel liefert:

$$\{a > 0\}$$

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

Bew. part. Korrektheit: Fakultät (16)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

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

Überblick (17)

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

Bew. part. Korrektheit: Fakultät (18)

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 4.5 Ergänzungen, Sprechweisen

Linearer vs. baumartiger Beweisstil

Vorteil linearen gegenüber baumartigen Beweisnotationsstils:

- wenig Redundanz
- daher insgesamt knappere Beweise

Sprechweisen im Zshg. mit Hoare-Tripeln (1)

Hoaresche Zusicherungen sind von einer der zwei Formen

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

wobei

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

Sprechweisen im Zshg. mit Hoare-Tripeln (2)

In einer Hoareschen Zusicherung von einer der Formen

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

heißen

- p und q *Vor-* bzw. *Nachbedingung*.

Sprechweisen im Zshg. mit Hoare-Tripeln (3)

In einer Hoareschen Zusicherung werden üblicherweise

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

benutzt.

Sprechweisen im Zshg. mit Hoare-Tripeln (4)

Zwei Beispiele Hoarescher Zusicherungen:

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

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

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

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

Sprechweisen im Zshg. mit Hoare-Tripeln (5)

Die Wortwahl

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

betont jeweils die

- syntaktische bzw.
- semantische Sicht

auf

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

Kapitel 4.6 Rückblick, Vertiefungen

Stärkste Nachbedingungen, schwächste Vorbedingungen

In der Folge:

Präzisierung von...

- Stärkste Nachbedingungen
- Schwächste Vorbedingungen

Zunächst:

- Ein Rückblick (R.blick)

R.blick: Definition partieller Korrektheit

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

Ein 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.

R.blick: Definition totaler Korrektheit

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

Ein 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.

R.blick: Partielle und totale Korrektheit

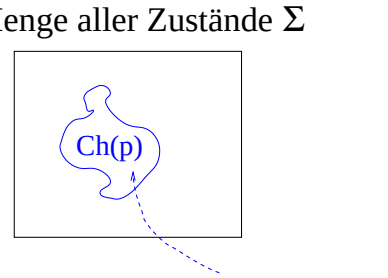
- Die Zustandsmenge
$$Ch(p) =_{df} \{\sigma \in \Sigma \mid [\![p]\!]_B(\sigma) = \mathtt{tt}\}$$
heißt *Charakterisierung von* $p \in$ **Bexp**.

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

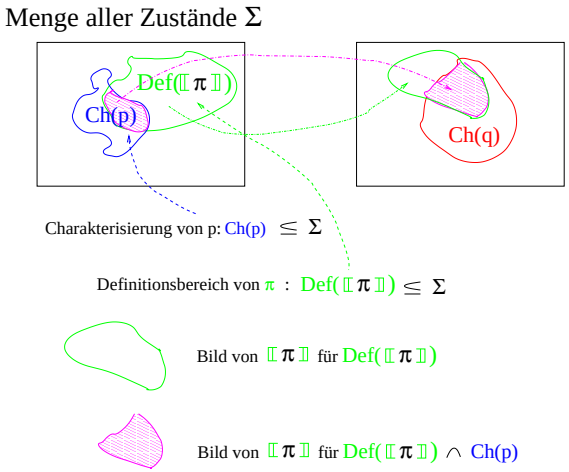
R.blick: Veranschaulichung (1)

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



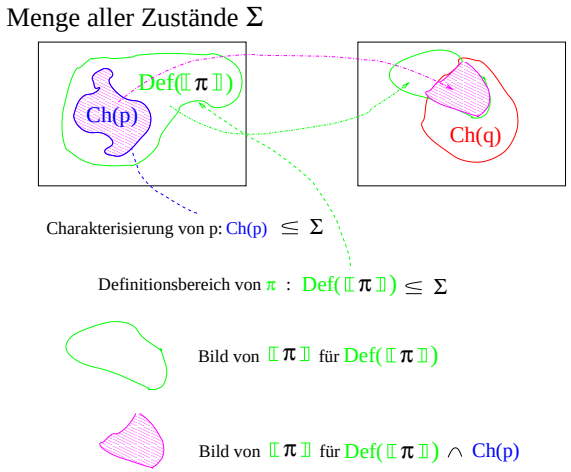
R.blick: Veranschaulichung (2)

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



R.blick: Veranschaulichung (3)

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



Stärkste Nach- und schwächste Vorbedingungen (1)

In der Situation der vorigen Abbildungen gilt:

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

Stärkste Nach- und schwächste Vorbedingungen (2)

Lemma

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

Für deterministische Programme π gilt:

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

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

Schwächste Vor- und stärkste Nachbedingungen

...noch einmal anders betrachtet:

Definition

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

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

Schwächste Vorbedingungen

Definition

Sei π ein Programm und q eine Formel.

Dann heißt

- $wp(\pi, q)$ *schwächste Vorbedingung* für totale Korrektheit von π bezüglich (der Nachbedingung) q, wenn
  $$[wp(\pi, q)]\ \pi\ [q]$$
  total korrekt ist und $wp(\pi, q)$ die schwächste Formel mit dieser Eigenschaft ist.
- $wlp(\pi, q)$ *schwächste liberale Vorbedingung* für partielle Korrektheit von π bezüglich (der Nachbedingung) q, wenn
  $$\{wlp(\pi, q)\}\ \pi\ \{q\}$$
  partiell korrekt ist und $wlp(\pi, q)$ die schwächste Formel mit dieser Eigenschaft ist.

Stärkste Nachbedingungen (1)

Analog zu A ist *schwächer* als B lässt sich definieren:

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

Zum Überlegen:

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

Stärkste Nachbedingungen (2)

Betrachte...

Definition(sversuch)

Sei π ein Programm und p eine Formel.

Dann heißt

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

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

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

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

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

Stärkste Nachbedingungen (3)

Fragen

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

  unterscheidbar, d.h. logisch nicht äquivalent sind?
- Wie passen die hier betrachteten Begriffe von schwächsten Vor- und stärksten Nachbedingungen mit denen auf Folie 13 von diesem Vorlesungsteil betrachteten zusammen?

Totale Korrektheit: HK_{TK} (1)

Zur Erinnerung sei hier der Hoare-Kalkül HK_{TK} für totale Korrektheit noch einmal wiederholt...

[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?

Totale Korrektheit: HK_{TK} (2)

[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).
  $\leadsto$ *Terminationsordnung*!

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

Wohlfundierte oder Noethersche Ordnungen (1)

Definition

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

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

Beispiele:

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

Wohlfundierte oder Noethersche Ordnungen (2)

Definition

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

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

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

von Elementen $w_i \in W$ gibt.

Das Paar $(W, <)$ heißt dann eine *wohlfundierte Struktur* oder auch eine *wohlfundierte* oder *Noethersche Ordnung*.

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

Beispiele:

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

Wohlfundierte oder Noethersche Ordnungen (3)

Konstruktionsprinzipien für wohlfundierte Ordnungen aus gegebenen wohlfundierten Ordnungen...

Lemma

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

Dann sind auch

- $(W_1 \times W_2, <_{com})$ mit *komponentenweiser* Ordnung definiert durch
  $$(m_1, m_2) <_{com} (n_1, n_2) \quad \text{gdw.} \quad m_1 <_1 n_1 \wedge m_2 <_2 n_2$$
- $(W_1 \times W_2, <_{lex})$ mit *lexikographischer* Ordnung def. durch
  $$(m_1, m_2) <_{lex} (n_1, n_2) \quad \text{gdw.}$$
  $$(m_1 <_1 n_1) \ \vee \ (m_1 = n_1 \wedge m_2 <_2 n_2)$$

wohlfundierte Ordnungen.

Anmerkungen zu...

...den der

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

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

Zur Konsequenzregel (1)

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

Intuitiv:

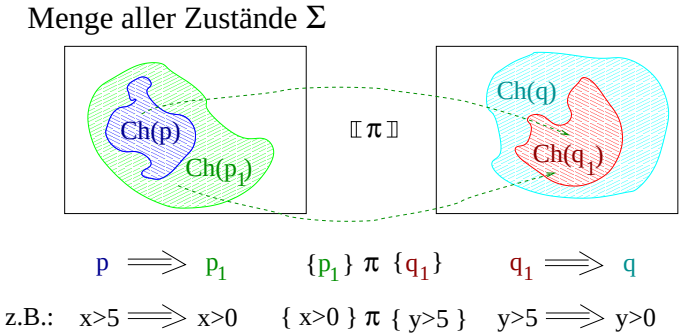
Die Konsequenzregel

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

  ...um so die Anwendung anderer Beweisregeln zu ermöglichen.

Zur Konsequenzregel (2)

Veranschaulichung von Verstärkung und Abschwächung:



Zur while-Regel in HK_{PK}

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

Intuitiv:

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

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

Erinnerung:

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

wobei

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

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

- Prämisse 1: $I \wedge b \Rightarrow u[t/v]$
  Wann immer der Schleifenrumpf noch einmal ausgeführt wird (d.h. $I \wedge b$ ist wahr), gilt, dass $u[t/v]$ wahr ist, woraus aufgrund der Definition von M folgt, dass der Wert von t Element einer noethersch geordneten Menge ist.
- Prämisse 2: $\{I \wedge b \wedge t = w\}\ \pi\ \{I \wedge t < w\}$
  - w speichert den initialen Wert von t (w ist sog. *logische Variable*), d.h. den Wert, den t vor Eintritt in die Schleife hat (gilt, da w als logische Variable insbesondere nicht in π vorkommt)
  - Zusammen damit, dass der Wert von w (als logische Variable) invariant unter der Ausführung des Schleifenrumpfs ist, garantiert $t < w$ in der Nachbedingung von Prämisse 2, dass der Wert von t nach jeder Ausführung des Schleifenrumpfs bzgl. der noetherschen Ordnung abgenommen hat.
- Zusammen implizieren die obigen beiden Punkte die Terminierung der while-Schleife, da es in einer noethersch geordneten Menge keine unendlich absteigenden Ketten gibt. Folglich kann die Bedingung $I \wedge b$ in Prämisse 1 nicht unendlich oft wahr sein, da dies zusammen mit Prämisse 2 ein unendliches Absteigen erforderte.)

Programm- vs. logische Variablen

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

- *Programmvariablen*
  ...Variablen, die in π vorkommen
- *logischen Variablen*
  ...Variablen, die in π nicht vorkommen

Logische Variablen erlauben...

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

Beispiel:

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

HK_{TK} versus HK_{PK}

Beachte:

HK_{TK} und HK_{PK} sind bis auf die Schleifenregel (und die Regel für *abort*) identisch...

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

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

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

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

Kapitel 4.7 Anwendungsbeispiel zum Nachweis totaler Korrektheit

Bew. totale Korrektheit: Fakultät (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 totaler Korrektheit.

Wir entwickeln den Beweis in der Folge Schritt für Schritt!

Wahl von Invariante und Terminierungsterm

Schritt 1

"Träumen"...

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

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

Beachte:

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

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

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

Wahl von Invariante und Terminierungsterm

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

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

Damit haben wir insbesondere:

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

Bew. totaler Korrektheit: Fakultät (4)

Schritt 2

Behandlung des Rumpfs der while-Schleife...

Der Nachweis der Gültigkeit von

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

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

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

Bew. totaler Korrektheit: Fakultät (5)

Behandlung des Rumpfs der while-Schleife im Detail:

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

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

Bew. totaler Korrektheit: Fakultät (6)

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

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

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

Bew. totaler Korrektheit: Fakultät (7)

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

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

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

...wobei noch eine "Beweislücke" verbleibt!

Bew. totaler Korrektheit: Fakultät (8)

Schluss der "Beweislücke" in der zugrundeliegenden Theorie:

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

Bew. totaler Korrektheit: Fakultät (9)

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

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

Bew. totaler Korrektheit: Fakultät (10)

Schritt 3

Zur gewünschten Nachbedingung verbleibt offenbar ebenfalls eine Beweislücke:

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

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

Bew. totaler Korrektheit: Fakultät (11)

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 (12)

Aus Platzgründen etwas verkürzt dargestellt:

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

Bew. totaler Korrektheit: Fakultät (13)

Schritt 4

Es verbleibt, die Beweislücke zur gewünschten Vorbedingung zu schließen:

$$[a > 0]$$

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

Bew. totaler Korrektheit: Fakultät (14)

Einmalige Anwendung der [ass]-Regel liefert:

$$[a > 0]$$

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

Bew. totaler Korrektheit: Fakultät (15)

Abermalige Anwendung der [ass]-Regel liefert:

$$[a > 0]$$

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

Bew. totaler Korrektheit: Fakultät (16)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

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

Überblick (17)

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

Bew. totaler Korrektheit: Fakultät (18)

Damit haben wir wie gewünscht insgesamt gezeigt:

Die Hoaresche Zusicherung

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

ist gültig im Sinne totaler Korrektheit.

Kapitel 4.8 Abschließendes, Ausblick

Nachtrag zur totalen Korrektheit (1)

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

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

wobei

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

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

Nachtrag zur totalen Korrektheit (2)

Beweistechnische Anmerkung:

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

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

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

- dem Nachweis ihrer partiellen Korrektheit
- dem Nachweis der Termination

Diese Trennung kann im Beweis explizit vollzogen werden. Der Gesamtbeweis wird dadurch modular. Oft gilt, dass der Terminationsnachweis einfach ist.

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

Zur Korrektheit und Vollständigkeit Hoarescher Beweiskalküle

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

Dann heißt K...

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

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

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

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

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

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

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

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

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

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

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

Zur Vollständigkeit Hoarescher Beweiskalküle

Generell müssen wir unterscheiden zwischen Vollständigkeit

- *extensionaler* und
- *intensionaler*

Ansätze.

Extensionale vs. intensionale Ansätze

- *Extensional*

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

- *Intensional*

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

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

Für den extensionalen Ansatz gilt:

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

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

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

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

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

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

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

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

Intuition

- *Entscheidbarkeit*

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

- *Ausdruckskraft*

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

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

Stichwort: *Relative Vollständigkeit im Sinne von Cook.*

Nachträge zu(r) Vorwärtszuweisungsregel(n)

- Eine *Vorwärtsregel* für die Zuweisung wie

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

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

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

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

  Beweis: Übungsaufgabe

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 Symposium of Mathematics and its Applications, Timisoara, Romania, November 6-9, 2003.

  `www.theorema.org/publication/2003/Laura/Poli_Timisoara_nov.pdf`