Teil 1: 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)

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: Partielle Korrektheit, 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.

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 *Turingmächtig*!

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: $[\![\ ]\!]_{ns} : \textbf{Prg} \to (\Sigma \to \Sigma)$
  - Strukturell operationelle Semantik: $[\![\ ]\!]_{sos} : \textbf{Prg} \to (\Sigma \to \Sigma)$
- Denotationelle Semantik: $[\![\ ]\!]_{ds} : \textbf{Prg} \to (\Sigma \to \Sigma)$
- Axiomatische Semantik: ...*abweichender Fokus*

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

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

Semantik arithmetischer & Boolescher Ausdrücke

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

Semantik

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

Semantik arithmetischer Ausdrücke (1)

$[\![\ .\ ]\!]_A : \textbf{Aexpr} \to (\Sigma \to \mathbb{Z})$ 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, **0**, **1**, **2**, ... bezeichnen *semantische* Entitäten, in diesem Falle ganze Zahlen.

Semantik Boolescher Ausdrücke (1)

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

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

Semantik Boolescher Ausdrücke (2)

...wobei

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

bezeichnen.

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

Freie Variablen

...arithmetischer Ausdrücke:

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

...Boolescher Ausdrücke:

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

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

Lemma 1.1
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 $[\![\ ]\!]_B$.

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

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

Beispiel: Beweis von Lemma 1.1 (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:

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

Übrige Fälle: Analog. q.e.d.

Zurück zur Semantik von WHILE

Vereinbarung:

Seien in der Folge die

- *Semantik arithmetischer Ausdrücke:*

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

- *Semantik Boolescher Ausdrücke:*

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

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

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

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)

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

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

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

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

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

SOS-Regeln von WHILE (2)

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

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

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

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 $i \in I\!N$.

Terminierende vs. divergierende Berechnungsfolgen

- Eine maximale (d.h. nicht mehr verlängerbare) Berechnungsfolge heißt
  - *regulär terminierend*, wenn sie endlich ist und die letzte Konfiguration aus Σ ist,
  - *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
  $\pi \equiv y := 1;\ \text{while } x <> 1 \text{ do } y := y * x;\ x := x - 1 \text{ od}$

Betrachte

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

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

  induzierte Berechnungsfolge

Beispiel (2)

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

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

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

Beispiel (3)

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

Beispiel (4)

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

Beispiel (Detailbetrachtung) (5)

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

steht vereinfachend für...

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

Beispiel (Detailbetrachtung) (6)

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

steht vereinfachend für...

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

Beispiel (Detailbetrachtung) (7)

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

steht vereinfachend für...

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

Determinismus der SOS-Regeln

Lemma 1.6

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

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

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

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

Nächste Vorlesungstermine...

- Mo, 08.10.2007, Vorlesung von 16:15 Uhr bis 17:45 Uhr im Hörsaal 14, TU-Hauptgebäude
- Mo, 15.10.2007, Vorlesung wird verschoben zugunsten des WIT-Kolloquiums: ab 17:00 Uhr s.t. im Hörsaal EI 7, Elektrotechnik (Neubau)
- Mo, 22.10.2007: Keine Vorlesung
- Mo, 29.10.2007: Vorlesung von 16:15 Uhr bis 17:45 Uhr im Hörsaal 14, TU-Hauptgebäude