
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:

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

- **Semantik**

...in Form von *Zustandstransformationen*:

$$\llbracket \cdot \rrbracket : \mathbf{Prg} \rightarrow (\Sigma \rightarrow \Sigma)$$

über

- $\Sigma =_{df} \{ \sigma \mid \sigma : \mathbf{Var} \rightarrow D \}$ 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

$$z ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$$

$$n ::= z \mid n z$$

Arithmetische Ausdrücke

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

Boolesche Ausdrücke

$$b ::= true \mid false \mid$$

$$a_1 = a_2 \mid a_1 \neq a_2 \mid a_1 < a_2 \mid a_1 \leq a_2 \mid \dots \mid$$

$$b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b_1$$

Überblick über Syntax & Semantik (3)

In der Folge bezeichnen wir mit...

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

Überblick über Syntax & Semantik (4)

In der Folge werden wir im Detail betrachten...

- Operationelle Semantik
 - Natürliche Semantik: $\llbracket \cdot \rrbracket_{ns} : \mathbf{Prg} \rightarrow (\Sigma \rightarrow \Sigma)$
 - Strukturell operationelle Semantik:
 $\llbracket \cdot \rrbracket_{sos} : \mathbf{Prg} \rightarrow (\Sigma \rightarrow \Sigma)$
- Denotationelle Semantik: $\llbracket \cdot \rrbracket_{ds} : \mathbf{Prg} \rightarrow (\Sigma \rightarrow \Sigma)$
- Axiomatische Semantik: ...*abweichender Fokus*

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

$$\llbracket \cdot \rrbracket_{sos}, \llbracket \cdot \rrbracket_{ns} \text{ und } \llbracket \pi \rrbracket_{ds}$$

Semantik arithmetischer & Boolescher Ausdrücke

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

Semantik

- arithmetischer Ausdrücke: $\llbracket \cdot \rrbracket_A : \mathbf{Aexpr} \rightarrow (\Sigma \rightarrow \mathbb{Z})$
- Boolescher Ausdrücke: $\llbracket \cdot \rrbracket_B : \mathbf{Bexpr} \rightarrow (\Sigma \rightarrow \mathbb{B})$

Semantik arithmetischer Ausdrücke (1)

$\llbracket \cdot \rrbracket_A : \mathbf{Aexpr} \rightarrow (\Sigma \rightarrow \mathbb{Z})$ induktiv definiert durch

- $\llbracket n \rrbracket_A(\sigma) =_{df} \llbracket n \rrbracket_N$
- $\llbracket x \rrbracket_A(\sigma) =_{df} \sigma(x)$
- $\llbracket a_1 + a_2 \rrbracket_A(\sigma) =_{df} plus(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma))$
- $\llbracket a_1 * a_2 \rrbracket_A(\sigma) =_{df} mal(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma))$
- $\llbracket a_1 - a_2 \rrbracket_A(\sigma) =_{df} minus(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma))$
- $\llbracket a_1 / a_2 \rrbracket_A(\sigma) =_{df} durch(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma))$
- ... (andere Operatoren analog)

wobei

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

Semantik arithmetischer Ausdrücke (2)

$\llbracket \cdot \rrbracket_N : \mathbf{Num} \rightarrow \mathbb{Z}$ induktiv definiert durch

- $\llbracket 0 \rrbracket_N =_{df} \mathbf{0}, \dots, \llbracket 9 \rrbracket_N =_{df} \mathbf{9}$
- $\llbracket n i \rrbracket_N =_{df} plus(mal(\mathbf{10}, \llbracket n \rrbracket_A), \llbracket i \rrbracket_N), i \in \{0, \dots, 9\}$
- $\llbracket - n \rrbracket_N =_{df} minus(\llbracket n \rrbracket_N)$

Beachte: $0, 1, 2, \dots$ bezeichnen *syntaktische* Entitäten, $\mathbf{0}, \mathbf{1}, \mathbf{2}, \dots$ bezeichnen *semantische* Entitäten, in diesem Falle ganze Zahlen.

Semantik Boolescher Ausdrücke (1)

$\llbracket \cdot \rrbracket_B : \mathbf{Bexpr} \rightarrow (\Sigma \rightarrow \mathbf{B})$ induktiv definiert durch

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

Semantik Boolescher Ausdrücke (2)

...wobei

- tt und ff die Wahrheitswertkonstanten “wahr” und “falsch” sowie
- $conj, disj : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$ und $neg : \mathbb{B} \rightarrow \mathbb{B}$ die übliche zweistellige logische Konjunktion und Disjunktion und einstellige Negation auf der Menge der Wahrheitswerte und
- $equal : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$ 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:

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

...Boolescher Ausdrücke:

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

Eigenschaften von $\llbracket \cdot \rrbracket_A$ und $\llbracket \cdot \rrbracket_B$

Lemma 1.1

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

$$\llbracket a \rrbracket_A(\sigma) = \llbracket a \rrbracket_A(\sigma')$$

Lemma 1.2

Seien $b \in \mathbf{BExpr}$ und $\sigma, \sigma' \in \Sigma$ mit $\sigma(x) = \sigma'(x)$ für alle $x \in FV(b)$. Dann gilt:

$$\llbracket b \rrbracket_B(\sigma) = \llbracket b \rrbracket_B(\sigma')$$

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

$$\cdot[\cdot/\cdot] : \mathbf{Aexpr} \times \mathbf{Aexpr} \times \mathbf{Var} \rightarrow \mathbf{Aexpr}$$

die induktiv definiert ist durch

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

Semantische Substitution

Definition 1.4

Die *semantische Substitution* ist eine dreistellige Abbildung

$$\cdot[\cdot/\cdot] : \Sigma \times \mathbb{Z} \times \mathbf{Var} \rightarrow \Sigma$$

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)

$$\llbracket e[t/x] \rrbracket_A(\sigma) = \llbracket e \rrbracket_A(\sigma[\llbracket t \rrbracket_A(\sigma)/x])$$

wobei

- $[t/x]$ die *syntaktische Substitution* und
- $\llbracket t \rrbracket_A(\sigma)/x$ die *semantische Substitution*

bezeichnen.

Analog gilt ein entsprechendes Substitutionslemma für $\llbracket \cdot \rrbracket_B$.

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*

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

- *verallgemeinerten Induktion*

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

- *strukturellen Induktion*

$$(\forall s \in S. \forall s' \in \text{Komp}(s). A(s')) \succ A(s) \succ \forall s \in S. A(s)$$

Beachte: \succ bezeichnet hier die logische Implikation.

Beispiel: Beweis von Lemma 1.1 (1)

...durch strukturelle Induktion

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

Induktionsanfang:

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

Mit den Definitionen von $\llbracket \cdot \rrbracket_A$ und $\llbracket \cdot \rrbracket_N$ erhalten wir unmittelbar wie gewünscht:

$$\llbracket a \rrbracket_A(\sigma) = \llbracket n \rrbracket_A(\sigma) = \llbracket n \rrbracket_N = \llbracket n \rrbracket_A(\sigma') = \llbracket a \rrbracket_A(\sigma')$$

Beispiel: Beweis von Lemma 1.1 (2)

Fall 2: Sei $a \equiv x$, $x \in \mathbf{Var}$.

Mit der Definition von $\llbracket \cdot \rrbracket_A$ erhalten wir auch hier wie gewünscht:

$$\llbracket a \rrbracket_A(\sigma) = \llbracket x \rrbracket_A(\sigma) = \sigma(x) = \sigma'(x) = \llbracket x \rrbracket_A(\sigma') = \llbracket a \rrbracket_A(\sigma')$$

Beispiel: Beweis von Lemma 1.1 (3)

Induktionsschluss:

Fall 3: Sei $a \equiv a_1 + a_2$, $a_1, a_2 \in \mathbf{Aexpr}$

Dann erhalten wir:

$$\begin{aligned} & \llbracket a \rrbracket_A(\sigma) \\ &= \llbracket a_1 + a_2 \rrbracket_A(\sigma) \\ &= \llbracket a_1 \rrbracket_A(\sigma) + \llbracket a_2 \rrbracket_A(\sigma) \\ \text{(Induktionshypothese für } a_1, a_2) &= \llbracket a_1 \rrbracket_A(\sigma') + \llbracket a_2 \rrbracket_A(\sigma') \\ &= \llbracket a_1 + a_2 \rrbracket_A(\sigma') \\ &= \llbracket a \rrbracket_A(\sigma') \end{aligned}$$

Übrige Fälle: Analog.

q.e.d.

Zurück zur Semantik von WHILE

Vereinbarung:

Seien in der Folge die

- *Semantik arithmetischer Ausdrücke:*

$$\llbracket \cdot \rrbracket_A : \mathbf{Aexpr} \rightarrow (\Sigma \rightarrow \mathbb{Z})$$

- *Semantik Boolescher Ausdrücke:*

$$\llbracket \cdot \rrbracket_B : \mathbf{Bexpr} \rightarrow (\Sigma \rightarrow \mathbf{B})$$

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

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

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:

$$\llbracket \cdot \rrbracket_{sos} : \mathbf{Prg} \rightarrow (\Sigma \rightarrow \Sigma_\varepsilon)$$

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)

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

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

$$[\text{ass}_{\text{sos}}] \frac{\text{—}}{\langle x := t, \sigma \rangle \Rightarrow \sigma[\llbracket t \rrbracket_A(\sigma) / x]}$$

$$[\text{comp}^1_{\text{sos}}] \frac{\langle \pi_1, \sigma \rangle \Rightarrow \langle \pi'_1, \sigma' \rangle}{\langle \pi_1; \pi_2, \sigma \rangle \Rightarrow \langle \pi'_1; \pi_2, \sigma' \rangle}$$

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

SOS-Regeln von WHILE (2)

$$[\text{if}_{\text{SOS}}^{\text{tt}}] \quad \frac{\text{---}}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \Rightarrow \langle \pi_1, \sigma \rangle} \quad \llbracket b \rrbracket_B(\sigma) = \text{tt}$$

$$[\text{if}_{\text{SOS}}^{\text{ff}}] \quad \frac{\text{---}}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \Rightarrow \langle \pi_2, \sigma \rangle} \quad \llbracket b \rrbracket_B(\sigma) = \text{ff}$$

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

Sprechweisen (1)

Wir unterscheiden

- Prämissenlose *Axiome* der Form

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

- Prämissenbehaftete *Regeln* der Form

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

ggf. mit *Randbedingungen* (*Seitenbedingungen*) wie z.B. in Form von $\llbracket b \rrbracket_B(\sigma) = \text{ff}$ in der Regel $[\text{if}_{s\text{os}}^{ff}]$.

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 $\gamma_0, \dots, \gamma_k$ von Konfigurationen mit $\gamma_0 = \langle \pi, \sigma \rangle$ und $\gamma_i \Rightarrow \gamma_{i+1}$ für alle $i \in \{0, \dots, k-1\}$,
 - 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 $\sigma(x) = 3$
- $\pi \in \mathbf{Prg}$ mit
 $\pi \equiv y := 1; \text{ while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}$

Betrachte

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

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

induzierte Berechnungsfolge

Beispiel (2)

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

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

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

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)

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

$\Rightarrow \langle \text{skip}, (\sigma[\mathbf{6}/y])[\mathbf{1}/x] \rangle$

$\Rightarrow (\sigma[\mathbf{6}/y])[\mathbf{1}/x]$

Beispiel (Detailbetrachtung) (5)

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

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

steht vereinfachend für...

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

Beispiel (Detailbetrachtung) (6)

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

steht vereinfachend für...

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

Beispiel (Detailbetrachtung) (7)

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

steht vereinfachend für...

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

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 $\llbracket \cdot \rrbracket_{sos}$

Korollar 1.7 erlaubt uns jetzt festzulegen:

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

$$\llbracket \cdot \rrbracket_{sos} : \mathbf{Prg} \rightarrow (\Sigma \rightarrow \Sigma_\varepsilon)$$

welches definiert wird durch:

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

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

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

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

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