

Formale Aspekte von Programmiersprachen

Überblick:

- Operationale Semantik
 - Natürliche (operationale) Semantik
 - Strukturelle operationale Semantik
- Denotationale Semantik
 - Fixpunkttheorie
 - Denotationale Semantikdefinition
- Zusammenhänge
- Spracherweiterungen

1

Semantik in Theoretische Informatik 1/2

Theoretische Informatik 1 behandelt:

- Semantik von Termen unter Umgebungen
- rekursive Semantik-Definition für AL
- axiomatische Semantik für AL (*Hoare-Kalkül*)
- verschiedene (andere) Formalismen für Semantikdefinition
- konkrete vs. abstrakte Syntax, Partialität von Funktionen
- Zusammenhänge zwischen verschiedenen Formalismen

3

Schwerpunkt hier: Formale Semantik

warum (formale) Semantik:

- präzise Spezifikation von Soft- und Hardware
- informelle Beschreibungen oft unklar, mehrdeutig, widersprüchlich
- genaue (formale) Aussagen und Schlüsse über Systemverhalten sonst nicht möglich
- Testen kann Fehler aufzeigen, aber nicht Korrektheit nachweisen
- Grundlage für: Prototyp-Implementierungen (Interpreter, Compiler), Sprachdesigner; Analyse, Transformation, Implementierung und Verifikation von Programmen

2

Operationale Semantik

$y := 0$; while $\neg(x = 0)$ do $(y := y + x; x := x - 1)$

Wie arbeitet das Programm?

Zunächst bekommt y den Wert 0 zugewiesen.
Dann wird getestet, ob x gleich 0 ist oder nicht. Falls nicht, sind wir fertig.
Andernfalls erhält y einen neuen Wert, nämlich die Summe des alten Wertes von y plus x , und x wird um eins erniedrigt. Sodann wird getestet, ob der neue Wert von x gleich 0 ist. Falls ja, ...

Zwei Versionen operationaler Semantik:

- natürliche Semantik
- strukturelle operationale Semantik

4

Denotationale Semantik

$:= 0$; while $\neg(x = 0)$ do $(y := y + x; x := x - 1)$

Welche (partielle) Funktion berechnet das Programm?

Das Programm berechnet eine partielle Funktion von Zuständen nach Zuständen. Der Endzustand ist dabei gleich dem Anfangszustand, nur der Wert von x ist im Endzustand 0 und der Wert von y gleich der Summe der natürlichen Zahlen bis x (falls der Wert von x zu Beginn nicht negativ war(!)).

Zwei Versionen denotationaler Semantik:

- *direct style* denotationale Semantik
- *continuation style* denotationale Semantik (hier nicht)

5

Mögliche Ansätze und Formalismen

- operationale Semantik
 - natürliche Semantik (siehe rekursive Formulierung in *Theoretische Informatik 1*)
 - strukturelle operationale Semantik
- denotationale Semantik
 - direct style denotationale Semantik
 - continuation style denotationale Semantik
- axiomatische Semantik
 - axiomatische Semantik für partielle Korrektheit
 - axiomatische Semantik für totale Korrektheit
- ...

7

Axiomatische Semantik (hier nicht (!))

$y := 0$; while $\neg(x = 0)$ do $(y := y + x; x := x - 1)$

Welche (relevanten) logischen Eigenschaften besitzt das Programm?

Falls zu Anfang x den Wert n besitzt, wird y am Ende den Wert $\sum_{i=1}^n i$ haben, sofern das Programm terminiert.

Zwei Arten axiomatischer Semantik:

- partielle Korrektheit (relativ zur Termination)
- totale Korrektheit (incl. Termination)

6

Welche Semantik wofür?

Sprachkonstrukte, -eigenschaften:

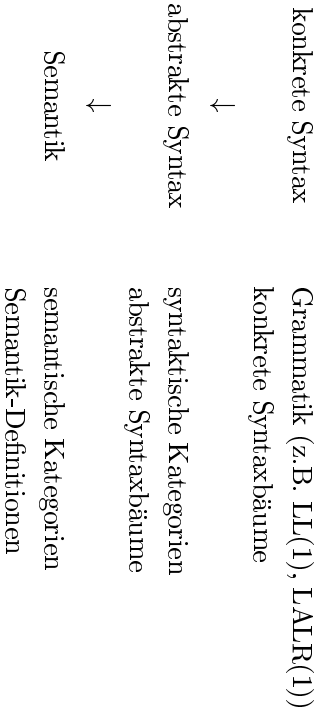
- imperativ
- funktional
- nebenläufig (concurrent) / parallel
- objektorientiert
- nichtdeterministisch ...

Zielsetzung:

- Verständnis der Sprache
- Programmanalyse
- Transformation / Verifikation von Programmen
- Prototyping / Übersetzerbau ...

8

Syntax — Semantik



9

WHILE: Semantische Kategorien

- Zahlen (hier: ganze Zahlen): $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$
- Wahrheitswerte: $\mathbf{T} = \{\mathbf{t}, \mathbf{f}\}$
- Zustände (Umgebungen):
 - State = $\mathbf{Var} \rightarrow \mathbb{Z}$
- Alternativen:
 - State' = $(\mathbf{Var} \times \mathbb{Z})^*$
 - State'' = $\mathbf{Var}^* \times \mathbb{Z}^*$
- Zugriff auf Wert einer Variablen: sx (statt $s(x)$)
- Zustandsänderung (update): $s[y \mapsto z]$, wobei

$$(s[y \mapsto z])x = \begin{cases} z & \text{falls } x = y \\ sx & \text{falls } x \neq y \end{cases}$$

11

WHILE: Syntaktische Kategorien

- Numerale: $n \in \mathbf{Num}$
- Variablen: $x \in \mathbf{Var}$
- arithmetische Ausdrücke: $a \in \mathbf{Aexp}$

$a ::= n \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2$
- boole'sche Ausdrücke: $b \in \mathbf{Bexp}$

$b ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2$
- Programme: $P \in \mathbf{Prog}$

$P ::= x := a \mid \text{skip} \mid S_1; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S$

10

WHILE: Bedeutungszuordnung

Semantik für syntaktische Kategorien (Signatur):

- Numerale:
 - $\mathcal{N} : \mathbf{Num} \rightarrow \mathbb{Z}$
- Variablen:
 - $s \in \text{State} = \mathbf{Var} \rightarrow \mathbb{Z}$
- arithmetische Ausdrücke:
 - $\mathcal{A} : \mathbf{Aexp} \rightarrow (\text{State} \rightarrow \mathbb{Z})$
- boole'sche Ausdrücke:
 - $\mathcal{B} : \mathbf{Bexp} \rightarrow (\text{State} \rightarrow \mathbf{T})$
- Programme:
 - $\mathcal{S} : \mathbf{Prog} \rightarrow (\text{State} \hookrightarrow \text{State})$

12

Semantik arithmetischer Ausdrücke

- $\mathcal{A} : \text{Aexp} \rightarrow (\text{State} \rightarrow \mathbb{Z})$
- Schreibweise: $\mathcal{A}[[a]]s$ statt $\mathcal{A}(s, a)$
- Definition:

$$\begin{aligned} \mathcal{A}[[n]]s &= \mathcal{N}[[n]] \\ \mathcal{A}[[x]]s &= s[x] \\ \mathcal{A}[[a_1 + a_2]]s &= \mathcal{A}[[a_1]]s + \mathcal{A}[[a_2]]s \\ \mathcal{A}[[a_1 * a_2]]s &= \mathcal{A}[[a_1]]s * \mathcal{A}[[a_2]]s \\ \mathcal{A}[[a_1 - a_2]]s &= \mathcal{A}[[a_1]]s - \mathcal{A}[[a_2]]s \end{aligned}$$
- Beispiel: Sei $sx = 2$ und $a = (x - 1) * (x + 1)$. Dann gilt:

$$\begin{aligned} \mathcal{A}[[a]]s &= (\mathcal{A}[[x - 1]]s) * (\mathcal{A}[[x + 1]]s) \\ &= (\mathcal{A}[[x]]s - \mathcal{A}[[1]]s) * (\mathcal{A}[[x]]s + \mathcal{A}[[1]]s) \\ &= (sx - \mathcal{N}([1]) * (sx + \mathcal{N}([1])) = (2 - 1) * (2 + 1) = 3 \end{aligned}$$

13

Compositionale Definitionen

- Die syntaktischen Kategorien werden durch abstrakte Syntax spezifiziert, so daß jedes Element eindeutig (in elementare Bestandteile) zerlegt werden kann.
- Die Semantik wird definiert durch kompositionale Funktionsdefinitionen.
- Semantik wohldefiniert, da Zerlegung eindeutig.

15

Semantik boole'scher Ausdrücke

- $\mathcal{B} : \text{Bexp} \rightarrow (\text{State} \rightarrow \mathbb{T})$
- $$\begin{aligned} \mathcal{B}[[\text{true}]]s &= \mathbf{t} \\ \mathcal{B}[[\text{false}]]s &= \mathbf{f} \\ \mathcal{B}[[a_1 = a_2]]s &= \begin{cases} \mathbf{t} & \text{falls } \mathcal{A}[[a_1]]s = \mathcal{A}[[a_2]]s \\ \mathbf{f} & \text{falls } \mathcal{A}[[a_1]]s \neq \mathcal{A}[[a_2]]s \end{cases} \\ \mathcal{B}[[a_1 \leq a_2]]s &= \begin{cases} \mathbf{t} & \text{falls } \mathcal{A}[[a_1]]s \leq \mathcal{A}[[a_2]]s \\ \mathbf{f} & \text{falls } \mathcal{A}[[a_1]]s \not\leq \mathcal{A}[[a_2]]s \end{cases} \\ \mathcal{B}[[\neg b]]s &= \begin{cases} \mathbf{t} & \text{falls } \mathcal{B}[[b]]s = \mathbf{f} \\ \mathbf{f} & \text{falls } \mathcal{B}[[b]]s = \mathbf{t} \end{cases} \\ \mathcal{B}[[b_1 \wedge b_2]]s &= \begin{cases} \mathbf{t} & \text{falls } \mathcal{B}[[b_1]]s = \mathbf{t} \text{ und } \mathcal{B}[[b_2]]s = \mathbf{t} \\ \mathbf{f} & \text{falls } \mathcal{B}[[b_1]]s = \mathbf{f} \text{ oder } \mathcal{B}[[b_2]]s = \mathbf{f} \end{cases} \end{aligned}$$

14

(Operationale) Semantik von Programmen

- syntaktische Kategorie:

$$P ::= x := a \mid \text{skip} \mid S_1; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S$$
- semantische Funktion:

$$\mathcal{S} : \text{Prog} \rightarrow (\text{State} \hookrightarrow \text{State})$$
- Definition von \mathcal{S} durch (*Zustands-Übergangssystem* $(\Gamma, E, \triangleright)$):
 - Γ : Menge von Konfigurationen (Zuständen)
 - E : Menge von Endkonfigurationen; $E \subseteq \Gamma$
 - \triangleright : (Zustands-)Übergangsrelation; $\triangleright \subseteq \Gamma \times \Gamma$
- *natürliche* und *strukturelle operationale* Semantik werden durch zwei verschiedene Zustandsübergangssysteme definiert

16

Natürliche Semantik

- Grundidee: Spezifiziere, wie das Gesamtergebnis der Berechnung zustandekommt.
- Übergangssystem: $(\Gamma, E, \triangleright)$ mit
 - $\Gamma = \{(P, s) \mid P \in \mathbf{WHILE}, s \in \text{State}\} \cup \text{State}$
 - $E = \text{State}$
 - $\triangleright \subseteq \{(P, s) \mid P \in \mathbf{WHILE}, s \in \text{State}\} \times \text{State}$
- Typischer Übergang: $(P, s) \rightarrow s'$, wobei
 - P Programm
 - s Anfangszustand
 - s' Endzustand

17

Inferenzregelsysteme

- Inferenzregelsystem I besteht aus
 - Axiomen(schemata) der Form

$$A$$
 - Regeln (Regelschemata) der Form

$$\frac{V_1 \quad V_2 \quad \dots \quad V_n}{K}$$
- mit Konklusion K und Voraussetzungen V_1, \dots, V_n
- Ableitung von Aussagen (Formeln) F in I :
 - **Prinzip:** Wende Regeln *rückwärts* an, bis alle entstehenden Voraussetzungen (*Blätter*) **Instanzen** von Axiomen sind.
 - **Beweis:** Falls dieser *Ableitungsbaum* vollständig ist (in obigem Sinn), ist die Aussage F in I bewiesen (abgeleitet).

19

Natürliche Semantik: Inferenzregelsystem

$$\begin{array}{l}
 [\text{ass}_{\text{ns}}] \quad \langle x := a, s \rangle \rightarrow s[x \mapsto \mathcal{A}[[a]]s] \\
 [\text{skip}_{\text{ns}}] \quad \langle \text{skip}, s \rangle \rightarrow s \\
 [\text{comp}_{\text{ns}}] \quad \frac{\langle P_1, s \rangle \rightarrow s', \quad \langle P_2, s' \rangle \rightarrow s''}{\langle P_1; P_2, s \rangle \rightarrow s''} \\
 [\text{if}_{\text{ns}}^{\text{f}}] \quad \frac{\langle P_1, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } P_1 \text{ else } P_2, s \rangle \rightarrow s'} \quad \text{falls } \mathcal{B}[[b]]s = \text{t} \\
 [\text{if}_{\text{ns}}^{\text{t}}] \quad \frac{\langle P_2, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } P_1 \text{ else } P_2, s \rangle \rightarrow s'} \quad \text{falls } \mathcal{B}[[b]]s = \text{f} \\
 [\text{while}_{\text{ns}}^{\text{f}}] \quad \frac{\langle P, s \rangle \rightarrow s', \quad \langle \text{while } b \text{ do } P, s' \rangle \rightarrow s''}{\langle \text{while } b \text{ do } P, s \rangle \rightarrow s''} \quad \text{falls } \mathcal{B}[[b]]s = \text{t} \\
 [\text{while}_{\text{ns}}^{\text{t}}] \quad \langle \text{while } b \text{ do } P, s \rangle \rightarrow s, \text{ falls } \mathcal{B}[[b]]s = \text{f}
 \end{array}$$

18

$$\text{Prog. } P: y := 0; \text{ while } \underbrace{\neg(x=0)}_b \text{ do } \underbrace{(y := y + x; x := x - 1)}_S$$

Greg.: s mit $sx = 1, sy$ beliebig (aber fest)

Ges.: s' mit $\langle P, s \rangle \rightarrow s'$

Not.: $\$m, n: \$m, n, x = m, \$m, n, y = n, \$m, n, z = sz$ für $z \notin \{x, y\}$

$$\begin{array}{l}
 \frac{\langle y := y + x, s_{1,0} \rangle \rightarrow s_{1,1} \quad \langle x := x - 1, s_{1,1} \rangle \rightarrow s_{0,1}}{[\text{ass}_{\text{ns}}] \quad \langle y := y + x; x := x - 1, s_{1,0} \rangle \rightarrow s_{0,1}} \quad [\text{comp}_{\text{ns}}] \\
 \frac{\frac{\langle S_1; S_2, s_{1,0} \rangle \rightarrow s_{0,1}}{[\text{while}_{\text{ns}}^{\text{f}}] \quad \langle \text{while } b \text{ do } S, s_{0,1} \rangle \rightarrow s_{0,1}} \quad [\text{while}_{\text{ns}}^{\text{t}}] \quad \langle \text{while } b \text{ do } S, s_{1,0} \rangle \rightarrow s_{0,1}}{[\text{while}_{\text{ns}}] \quad \langle \text{while } b \text{ do } S, s_{1,0} \rangle \rightarrow s_{0,1}} \\
 \frac{\langle y := 0, s \rangle \rightarrow s_{1,0} \quad [\text{while } b \text{ do } S, s_{1,0} \rangle \rightarrow s_{0,1}}{[\text{ass}_{\text{ns}}] \quad \langle y := 0, s \rangle \rightarrow s_{1,0}} \quad [\text{comp}_{\text{ns}}] \quad \langle P, s \rangle \rightarrow s_{0,1}
 \end{array}$$

20