

4 Formale Semantik

4.2 Einfache Beispiele

Beispiel: Binärzahlen

Syntaktische Beschreibung:

$$BIN = \{w \in \{0, 1\}^* \mid w = 0 \text{ oder } w \text{ beginnt mit } 1\}$$

Semantik:

Sei $w \in BIN$. Ist $w = 0$, so ist der Wert von w die Zahl 0. Ist $w = 1$, so ist der Wert von w die Zahl 1.

(Hierbei sind in $w = 0$ und $w = 1$ die Ziffern 0 und 1 einfach Buchstaben, während die Zahl 0 bzw. 1 jeweils das Element von \mathbb{N} bezeichnet.)

Ist $w = w'0$ und die Länge von w' größer als 0, so ist der Wert von w das Doppelte des Wertes von w' .

Ist $w = w'1$ und die Länge von w' größer als 0, so ist der Wert von w gleich 2 mal Wert von w' plus 1.

4.1 Was ist Semantik

Programmiersprachen werden von drei Blickwinkeln betrachtet:

- a) Syntax (elementare lexikalische und grammatikalische Struktur; keine Beziehung zur Bedeutung oder Interpretation)
- b) Semantik (Bedeutung, Interpretation; z.B. eine Zahl, eine Funktion, . . .)
- c) Pragmatik (Brauchbarkeit, Lesbarkeit, Übersetzbarkeit, . . .)

Semantik von Programmiersprachen dient nicht nur dazu, den Sinn eines Programms festzulegen, sondern sie soll auch z.B. ein Hilfsmittel für Korrektheitsbeweise darstellen.

Beispiel: Reguläre Ausdrücke

Syntax: (Sei $\Gamma = \{a_1, \dots, a_k\}$.)

$\emptyset, \varepsilon, a_1, \dots, a_k$ sind Elemente von $RegExp_\Gamma$.

Wenn α und β Elemente von $RegExp_\Gamma$ sind, dann sind auch $(\alpha\beta)$, $(\alpha \cup \beta)$ und $(\alpha)^*$ in $RegExp_\Gamma$.

Semantik:

Sei $\gamma \in RegExp_\Gamma$. Ist $\gamma = \emptyset$, so ist der zugehörige Wert die leere Menge. Ist $\gamma = \varepsilon$, so ist die zugehörige Menge $\{\varepsilon\}$. Ist $\gamma = a_i$ für ein $i \in \{1, \dots, k\}$, so ist der Wert die Menge $\{a_i\}$.

Ist $\gamma = \alpha\beta$ für zwei Ausdrücke $\alpha, \beta \in RegExp_\Gamma$, so ist der Wert die Menge $\{uv \mid u \in W(\alpha), v \in W(\beta)\}$. Dabei sei $W(\alpha)$ der zu α gehörige Wert, entsprechend für β .

Ist $\gamma = \alpha \cup \beta$, so ist der Wert die Menge $W(\alpha) \cup W(\beta)$.

Ist $\gamma = (\alpha)^*$, so ist der Wert die Menge $\{u_1 \dots u_r \mid r \geq 0 \wedge \forall i : u_i \in W(\alpha)\}$.

Beispiel: Endliche Automaten

Syntax:

$(Q, \Gamma, \delta, q_0, F) \in EA_\Gamma$, falls

- Q endliche Menge
- Γ endl. Alphabet ($Q \cap \Gamma = \emptyset$)
- $\delta: Q \times \Gamma \rightarrow Q$
- $q_0 \in Q$
- $F \subseteq Q$

Semantik:

Der Wert von $A = (Q, \Gamma, \delta, q_0, F) \in EA_\Gamma$ ist die Menge der von dem Automat A akzeptierten Wörter aus Γ^* .

4.4 Die Sprache IMP

IMP steht für

Imperative Sprache

Grundbereiche:

Natürliche Zahlen \mathbb{N} , Wahrheitswerte $\mathbb{B} = \{\text{true}, \text{false}\}$.

Variablen:

$\mathbb{V} = \{X_1, X_2, X_3, \dots\}$ (abzählbare Menge von Bezeichnern)

$\mathbf{Loc} \subseteq \mathbb{V}$ (benutzte Variablen)

Speicherzustände:

$\Sigma = \mathbb{N}^{\mathbf{Loc}} = \{\sigma: \mathbf{Loc} \rightarrow \mathbb{N}\}$

4.3 Semantik-Funktionen

Als Semantik-Funktion bezeichnen wir die Abbildung, die zu jeder syntaktisch korrekten Konstruktion ihre Bedeutung als Resultat liefert:

$$\mathcal{B}: BIN \rightarrow \mathbb{N}$$

(z.B. $\mathcal{B}(10010) = 18$).

$$\mathcal{E}: RegExp_\Gamma \rightarrow 2^{\Gamma^*}$$

(z.B. $\mathcal{E}((a \cup b)^*a) = \{wa \mid w \in \{a, b\}^*\}$).

$$\mathcal{A}: EA_\Gamma \rightarrow 2^{\Gamma^*}$$

(z.B. $\mathcal{A}(\dots) = \{wa \mid w \in \{a, b\}^*\}$).

(Was kann man hier für \dots einsetzen??)

IMP verfügt über drei Arten syntaktischer Konstrukte:

- Arithmetische Ausdrücke
(Aexp)
- Boolesche Ausdrücke
(Bexp)
- Anweisungen bzw. Programme
(Cmd)

Aexp

Syntaktische Definition:

$$a ::= n \mid X \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 \cdot a_2)$$

Dabei ist $n \in \mathbb{N}$, $X \in \mathbb{V}$. Die Klammern können manchmal weggelassen werden, wenn dadurch keine Mehrdeutigkeit entsteht.

Das heißt also:

1. Natürliche Zahlen und Variablen bilden arithmetische Ausdrücke.
2. Wenn a_1 und a_2 zu **Aexp** gehören, dann auch $(a_1 + a_2)$, $(a_1 - a_2)$, $(a_1 \cdot a_2)$.
3. Sonst gehört nichts zu **Aexp**.

Beispiele

$$\begin{aligned} &(((X_1 \cdot 3) - 5) + X_2) - (X_3 \cdot 88) \in \mathbf{Aexp} \\ &5X_2 \notin \mathbf{Aexp} \end{aligned}$$

Auswertung

Arithmetische Ausdrücke werten sich je nach Speicherzustand zu einer natürlichen Zahl aus.

Formal haben wir also eine Funktion vom Typ

$$\mathbf{Aexp} \times \Sigma \rightarrow \mathbb{N}$$

Wir schreiben $(a, \sigma) \rightarrow n$, falls der Ausdruck a im Speicherzustand σ zur Zahl n ausgewertet wird.

Die Semantik-Funktion \mathcal{A} ordnet jedem Ausdruck $a \in \mathbf{Aexp}$ eine Funktion auf der Menge der Speicherzustände mit natürlichzahligen Werten zu.

Für $(a, \sigma) \rightarrow n$ schreiben wir daher auch

$$\mathcal{A}[[a]]\sigma = n$$

Bexp

Syntaktische Definition:

$$\begin{aligned} b ::= & \mathbf{true} \mid \mathbf{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \\ & a_1 > a_2 \mid a_1 \neq a_2 \mid \neg b \mid b_1 \wedge b_2 \mid \\ & b_1 \vee b_2 \mid b_1 \Rightarrow b_2 \mid b_1 \Leftrightarrow b_2 \end{aligned}$$

Dabei seien $a_1, a_2 \in \mathbf{Aexp}$, $b, b_1, b_2 \in \mathbf{Bexp}$.

Die Auswertung ist wie folgt definiert:

$$(n, \sigma) \rightarrow n \quad (n \in \mathbb{N}, \sigma \in \Sigma)$$

$$(X, \sigma) \rightarrow \sigma(X) \quad (X \in \mathbf{Loc}, \sigma \in \Sigma)$$

$$((a_1 + a_2), \sigma) \rightarrow (a_1, \sigma) + (a_2, \sigma)$$

$$((a_1 - a_2), \sigma) \rightarrow (a_1, \sigma) - (a_2, \sigma)$$

(bzw. 0, falls Wert negativ)

$$((a_1 \cdot a_2), \sigma) \rightarrow (a_1, \sigma) \cdot (a_2, \sigma)$$

Die Zeichen $+$, $-$, \cdot auf der rechten Seite bezeichnen die bekannten Operationen auf natürlichen Zahlen!

Achtung: In $(a_1, \sigma) + (a_2, \sigma)$ etc. setzen wir die Paare (a_1, σ) , ... bereits mit ihrer jeweiligen Auswertung gleich!!

Andere Schreibweise für diese Auswertung:

$$\mathcal{A}: \mathbf{Aexp} \rightarrow (\Sigma \rightarrow \mathbb{N})$$

Auswertung

$(b, \sigma) \rightarrow t$ für geeignetes $t \in \mathbb{B}$.

Wir schreiben hier auch

$$\mathcal{B}[[b]]\sigma = t$$

Also:

$$\mathcal{B}: \mathbf{Bexp} \rightarrow (\Sigma \rightarrow \mathbb{B})$$

Die Auswertung boolescher Ausdrücke geschieht nach folgenden Regeln:

$$(t, \sigma) \rightarrow t \quad (t \in \mathbb{B}, \sigma \in \Sigma)$$

$$(a_1 = a_2, \sigma) \rightarrow \mathbf{true}$$

(wenn $(a_1, \sigma) \rightarrow n$ und $(a_2, \sigma) \rightarrow m$ und $m = n$ gilt)

$$(a_1 = a_2, \sigma) \rightarrow \mathbf{false}$$

(wenn $(a_1, \sigma) \rightarrow n$ und $(a_2, \sigma) \rightarrow m$ und $m \neq n$ gilt)

$$(a_1 < a_2, \sigma) \text{ analog zu } (a_1 = a_2, \sigma).$$

$$(a_1 > a_2, \sigma) \text{ ebenfalls analog.}$$

$$(a_1 \neq a_2, \sigma) \text{ ebenfalls analog.}$$

$$(\neg b, \sigma) \rightarrow \neg(b, \sigma)$$

$(b_1 \vee b_2, \sigma) \rightarrow \mathbf{true}$
 (wenn $(b_1, \sigma) \rightarrow \mathbf{true}$ oder $(b_2, \sigma) \rightarrow \mathbf{true}$ gilt)

$(b_1 \vee b_2, \sigma) \rightarrow \mathbf{false}$
 (wenn $(b_1, \sigma) \rightarrow \mathbf{false}$ und $(b_2, \sigma) \rightarrow \mathbf{false}$ gilt)

$(b_1 \wedge b_2, \sigma)$ analog.

$(b_1 \Rightarrow b_2, \sigma) \rightarrow \mathbf{true}$
 (wenn $(b_1, \sigma) \rightarrow \mathbf{false}$ oder $(b_2, \sigma) \rightarrow \mathbf{true}$ gilt)

$(b_1 \Rightarrow b_2, \sigma) \rightarrow \mathbf{false}$
 (wenn $(b_1, \sigma) \rightarrow \mathbf{true}$ und $(b_2, \sigma) \rightarrow \mathbf{false}$ gilt)

$(b_1 \Leftrightarrow b_2, \sigma)$ analog.

Beispiel:

```

c ≡ X2 := 1;
  while X1 > 1 do
    X2 := X2 · X1;
    X1 := X1 - 1
  od
    
```

Σ besteht aus Zahlenpaaren, da nur X_1 und X_2 vorkommen:

$$\Sigma = \{(n_1, n_2) \mid n_1, n_2 \in \mathbb{N}\}$$

Dabei sei für $\sigma = (n_1, n_2) \in \Sigma$ erfüllt:

$$\sigma(X_1) = n_1 \quad \sigma(X_2) = n_2$$

Was ist $\mathcal{C}[[c]]\sigma$?

Offensichtlich gilt mit

$$\sigma = (n_1, n_2) \quad \text{und} \quad \sigma' = (n_1, n'_2)$$

immer $\mathcal{C}[[c]]\sigma = \mathcal{C}[[c]]\sigma'$, und für alle $\sigma = (n_1, n_2)$ mit $n_1 > 0$ gilt

$$\mathcal{C}[[c]]\sigma = \tau \quad \Rightarrow \quad \exists n : \tau = (1, n).$$

Cmd

Syntaktische Definition:

$$c ::= \mathbf{skip} \mid X := a \mid c_1; c_2 \mid$$

if b **then** c_1 **else** c_2 **fi** **|** **while** b **do** c **od**

($X \in \mathbb{V}$, $a \in \mathbf{Aexp}$, $b \in \mathbf{Bexp}$, $c, c_1, c_2 \in \mathbf{Cmd}$.)

Auswertung

Zu gegebenem Programm $c \in \mathbf{Cmd}$ und Speicherzustand $\sigma \in \Sigma$ wird ein neuer Speicherzustand $\sigma' \in \Sigma$ zugeordnet:

$$\mathbf{Cmd} \times \Sigma \rightarrow^p \Sigma$$

oder auch

$$\mathcal{C} : \mathbf{Cmd} \rightarrow (\Sigma \rightarrow^p \Sigma)$$

(Das p am Pfeil steht für „partiell“.)

(Warum?)

In $\mathcal{C}[[c]](n_1, n_2) = (1, n)$ hängt also eigentlich n nur von n_1 ab. Daher betrachten wir

$$\text{IN}(X_1) \ c \ \text{OUT}(X_2)$$

Was ist der Wert von X_2 nach Ausführung von c , wenn zuvor X_1 einen gegebenen Wert N hatte?

Diese Betrachtungen erfolgten auf einem intuitiven Begriff der Anweisungen unserer Programmiersprache IMP.

Um die Wirkung der Anweisung c formal fassen zu können und die Behauptung, c berechne die Fakultätsfunktion auch beweisbar zu machen, benötigen wir die formale Semantik!

4.5 Operationale Semantik der Sprache IMP (am Beispiel)

Wir denken uns eine

IMP-Maschine

die das Programm unserer naiven Vorstellung gemäß abarbeitet;

sei z.B. der Startzustand $\sigma = (3, 0)$ gegeben, d.h.

$$\sigma(X_1) = 3$$

$$\sigma(X_2) = 0$$

Dann erhalten wir:

4.6 Denotationale Semantik der Sprache IMP (am Beispiel)

Zu einem Programm der Form

$$p \equiv \text{IN}(X_1, \dots, X_m) \ c \ \text{OUT}(Y_1, \dots, Y_n)$$

betrachten wir die partielle Funktion

$$\mathcal{C}[[p]]: \mathbb{N}^m \rightarrow^p \mathbb{N}^n,$$

die wie folgt definiert ist:

Wird c auf den Speicherzustand, der durch Eingabe n_1, \dots, n_m für X_1, \dots, X_m gegeben ist (alle anderen Variablen 0), gestartet, so ist

$$\mathcal{C}[[p]](n_1, \dots, n_m)$$

genau dann definiert, wenn c terminiert, der Funktionswert ist das n -Tupel der Variablenwerte von Y_1, \dots, Y_n nach der Ausführung.

$$\begin{aligned} (c, (3, 0)) &\rightarrow (w, (3, 1)) && w = \text{gesamte} \\ &&& \text{While-Schleife} \\ &\rightarrow (c_1; c_2; w, (3, 1)) && c_1: X_2 := X_2 \cdot X_1 \\ &&& c_2: X_1 := X_1 - 1 \\ &\rightarrow (c_2; w, (3, 3)) \\ &\rightarrow (w, (2, 3)) \\ &\rightarrow (c_1; c_2; w, (2, 3)) \\ &\rightarrow (c_2; w, (2, 6)) \\ &\rightarrow (w, (1, 6)) \\ &\rightarrow (\text{skip}, (1, 6)) \\ &\rightarrow (1, 6) \end{aligned}$$

Der Endzustand $(1, 6)$ sagt, daß $3! = 6$ richtig berechnet wird.

Ohne Angabe von IN und OUT betrachten wir

$$\mathcal{C}[[c]]: \Sigma \rightarrow^p \Sigma,$$

d.h. alle verwendeten Variablen werden sowohl als Eingabe-, wie auch als Ausgabewerte angesehen.

Beispiel:

```
c ≡ while X1 ≠ X2 do
  if X1 > X2 then
    X1 := X1 - X2
  else
    X2 := X2 - X1
  fi
od
```

Betrachte $p \equiv \text{IN}(X_1, X_2) \ c \ \text{OUT}(X_2)$.

Welche Funktion wird berechnet?

Behauptung

Für $f = \mathcal{C}[[c]]$ und Speicherzustände $(\sigma(X_1), \sigma(X_2))$ aus \mathbb{N}^2 gilt:

$$f((0, 0)) = (0, 0)$$

$$f((m, 0)) = \text{undef} \quad (m > 0)$$

$$f((0, n)) = \text{undef} \quad (n > 0)$$

$$f((m, n)) = \text{ggT}(m, n) \quad (m > 0, n > 0)$$

Beweis

Mühsam „zu Fuß“.

Annahme: $X_1 > 0$

(sonst ist das Ziel trivialerweise erreicht!)

Wegen $c \equiv X_2 := 1; w$ mit

$$w \equiv \text{while } X_1 > 1 \text{ do } \dots \text{ od}$$

erhalten wir die Äquivalenz von

$$\{X_1 = N\} c \{X_2 = N!\}$$

mit

$$\{X_1 = N \wedge X_2 = 1\} w \{X_2 = N!\}.$$

Wegen $X_2 = 1$ gilt anfangs offenbar

$$X_2 \cdot (X_1!) = N!$$

Nach Ausführung von w ist $X_1 = 1$. Wenn immer noch

$$X_2 \cdot (X_1!) = N!$$

gilt, dann ist $X_2 = N!$, wie gewünscht.

4.7 Axiomatische Semantik der Sprache IMP (am Beispiel)

Verwende „Zusicherungen“ und „Invarianten“.

Betrachte noch einmal das Fakultätsprogramm:

```

c ≡ X2 := 1;
  while X1 > 1 do
    X2 := X2 · X1;
    X1 := X1 - 1
  od

```

Ziel:

$$\{X_1 = N\} c \{X_2 = N!\}$$

Das Ziel bedeutet:

Wenn c auf einen Speicherzustand mit $X_1 = N$ angewendet wird, dann erhält nach der Ausführung X_2 den Wert $N!$. Insbesondere terminiert das Programm.

Dies soll nun bewiesen werden.

Wir wollen also zeigen, daß $X_2 \cdot (X_1!) = N!$ eine Invariante von w ist.

Es genügt folgende Aussage:

$$\{X_1 > 1 \wedge X_2 \cdot (X_1!) = N!\}$$

$$X_2 := X_2 \cdot X_1; X_1 := X_1 - 1$$

$$\{X_2 \cdot (X_1!) = N! \wedge X_1 \geq 1\}.$$

Diese kann man in zwei Schritten herleiten.