

Übungen zur Vorlesung
 “Grundlagen der Programm- und Systementwicklung”

Aufgabe 10.1 (H) Zustandsbegriff; denotationelle Semantik

Die denotationelle Semantik dient dazu, einem syntaktischen Konstrukt (ein Programm, ein Graph) seine Bedeutung (Interpretation) zuzuordnen. Mit ihr ist es beispielsweise möglich, einem Programm für die Fakultätsfunktion bei Vorgabe eines Eingabeparameters diejenige natürliche Zahl zuzuordnen, die der Fakultät der Eingabe entspricht.

Für die Definition eines Interpreters einer Variante der while-Sprache benötigt man zunächst die syntaktischen Kategorien der arithmetischen Ausdrücke AExp, der Booleschen Ausdrücke BExp und der Befehle CExp, die wie folgt definiert sind:

$AExp = \mathcal{N} \mid \mathcal{I} \mid AExp + AExp \mid AExp - AExp \mid (AExp)$, wobei \mathcal{N} für die natürlichen Zahlen und \mathcal{I} für eine endliche Menge von Variablennamen (z.B. $x, y, x_1, \dots, x_n, \dots$) steht (es ergibt sich aus den in einem Programm vorkommenden Variablen, wie \mathcal{I} konkret aussieht),

$BExp = true \mid false \mid AExp \Delta AExp \mid (BExp)$ für $\Delta \in \{<, >, \leq, \geq, =, \neq\}$,

$CExp = CExp; CExp \mid \mathcal{I} := AExp \mid while BExp do CExp \mid$
 $if BExp then CExp else CExp \mid (CExp)$,

wobei wiederum \mathcal{I} für die endliche Menge von Variablennamen steht. Die Syntax der while-Sprache ist dann durch die BNF-Definition von CExp festgelegt.

Ausdrücke der while-Sprache werden nun mit Hilfe dreier semantischer Funktionen $\mathcal{A} : AExp \rightarrow States \rightarrow \mathcal{N}$, $\mathcal{B} : BExp \rightarrow States \rightarrow \{true, false\}$ und $\mathcal{C} : CExp \rightarrow States \rightarrow States$ über einer Zustandsmenge $States = \mathcal{I} \rightarrow \mathcal{N}$ interpretiert, die Identifikatoren (Variablennamen) auf natürliche Zahlen abbildet. Dabei bezeichnet $A \rightarrow B$ eine partielle Funktion von A nach B. Üblicherweise wird das erste Argument dieser semantischen Funktionen, die zu interpretierende syntaktische Einheit, in doppelte eckige Klammern gesetzt:

$\mathcal{A}[[n]]\sigma = n$, falls $n \in \mathcal{N}$

$\mathcal{A}[[v]]\sigma = \sigma(v)$, falls $v \in \mathcal{I}$

$\mathcal{A}[[a \Delta b]]\sigma = \mathcal{A}[[a]]\sigma \Delta^{\mathcal{N}} \mathcal{A}[[b]]\sigma$ für $\Delta \in \{+, -\}$; das $\Delta^{\mathcal{N}}$ der rechten Seite ist die Interpretation von Δ in \mathcal{N} .

$\mathcal{B}[[x \Delta y]]\sigma = \mathcal{A}[[x]]\sigma \Delta \mathcal{A}[[y]]\sigma$ für $\Delta \in \{<, >, \leq, \geq, =, \neq\}$, und diese Vergleichsoperatoren werden wie üblich in \mathcal{N} mit *true* oder *false* als Resultat definiert.

$\mathcal{C}[[c; d]]\sigma = \mathcal{C}[[d]](\mathcal{C}[[c]]\sigma)$

$\mathcal{C}[[v := a]]\sigma = \sigma[v \mapsto \mathcal{A}[[a]]\sigma]$, d.h. nach Ausführung dieses Befehls ist $v \in \mathcal{I}$ in σ an die Semantik von a gebunden.

$\mathcal{C}[[if b then c else d]]\sigma = \begin{cases} \mathcal{C}[[c]]\sigma, & \text{falls } \mathcal{B}[[b]]\sigma = true \\ \mathcal{C}[[d]]\sigma, & \text{falls } \mathcal{B}[[b]]\sigma = false \end{cases}$

$$\mathcal{C}[\textit{while } b \textit{ do } c]\sigma = \begin{cases} \sigma, & \textit{falls } \mathcal{B}[b]\sigma = \textit{false} \\ \mathcal{C}[c; \textit{while } b \textit{ do } c]\sigma & \textit{falls } \mathcal{B}[b]\sigma = \textit{true} \end{cases}$$

Zum Beispiel ist $\mathcal{A}[x]\{x \mapsto 42\} = 42$, $\mathcal{B}[x \leq y]\{x \mapsto 42, y \mapsto 37\} = \textit{false}$ und $\mathcal{C}[\textit{while } x \leq 3 \textit{ do}(y := y + y; x := x + 1)]\{x \mapsto 1, y \mapsto 1\} = \{x \mapsto 4, y \mapsto 8\}$.

Es wird angenommen, daß die Eingabeparameter in Form von Variablenbindungen, d.h. Zuständen, übergeben werden, und daß die Ausgabe in bestimmten bekannten Variablen abgespeichert wird. Als Semantik eines while-Programms P ergibt sich dann mit den Eingabeparametern x_1, \dots, x_m , die über $\sigma_\iota = \{x_1 \mapsto n_1, \dots, x_m \mapsto n_m\}$ vorbelegt sind, und den Resultatvariablen $\{y_1, \dots, y_l\} \subseteq \mathcal{I}$: $(\mathcal{C}[P]\sigma_\iota)_{y_1}, \dots, (\mathcal{C}[P]\sigma_\iota)_{y_l}$, d.h. das Ergebnis der Berechnung von \mathcal{C} mit den Anfangsparametern σ_ι , ein Zustand, wird auf die *Namen* der Ausgabevariablen y_i angewendet und liefert so die *Werte* der einzelnen Ausgabevariablen. Offenbar stellen die semantischen Funktionen einen Interpreter dar, da einem Programm und einer Menge von Eingabeparametern eine Menge von Ausgabewerten (partiell) zugeordnet wird.

1. Schreiben Sie ein while-Programm, das für den Startzustand $\sigma = \{x \mapsto n\}$ für $n \in \mathcal{N}$ die n-te Fibonacci-Zahl berechnet!
2. *Beweisen* Sie, daß Ihr Programm die n-te Fibonacci-Zahl berechnet!
3. Wie könnte man die Definition der denotationellen Semantik erweitern, um Nichtdeterminismus zu modellieren?

Aufgabe 10.2 (H) Das Kaffeedosenproblem; Invarianten

Eine Kaffeedose enthält weiße und schwarze Bohnen. Der folgende Prozeß soll so lange wie möglich wiederholt werden: Wähle zwei Bohnen. Haben Sie dieselbe Farbe, ersetze die zwei Bohnen durch eine schwarze Bohne. Haben sie verschiedene Farben, so ersetze die zwei Bohnen durch eine weiße Bohne.

Geben Sie eine Invariante des Prozesses an, die garantiert, daß am Ende genau eine schwarze Bohne in der Dose bleibt. Für welche Ausgangskonfigurationen kann das garantiert werden? Beweis! (Hinweis: Konfluenz (Übung 5) oder Noethersche Induktion!)

Aufgabe 10.3 (H) Vor- und Nachbedingungen

Für ein Programm S gelte unter der Vorbedingung $x=y$ die Nachbedingung $x=y!$. Unter welchen Bedingungen bzgl. S berechnet S die Fakultätsfunktion?

Aufgabe 10.4 (H) Zusicherungskalkül

Ist die Ableitungsregel $\{\textit{true}\} x:=t \{x=t\}$ für Zuweisungen korrekt?

Aufgabe 10.5 (H) Wp-Kalkül

Berechnen Sie $w(1)_p(S, x=y)$ und $w(1)_p(S, x \neq y)$ für

$$S = \textit{do } x \neq y \textit{ then } x:=x+1 \textit{ od}.$$