2 Die Sprache While

While ist eine einfache, imperative Programmiersprache, für die in dieser Vorlesung in allen drei Ansätzen eine Semantik ausgearbeitet wird. Obwohl eine Semantik erst auf einem abstrakten Syntaxbaum aufbaut, muss man sich auf eine konkrete Syntax festlegen, um überhaupt Programme textuell aufschreiben zu können.

2.1 Syntax

Programme werden üblicherweise `in Schreibmaschinenschrift` dargestellt. Deren Syntax wird durch eine BNF-Grammatik mit drei syntaktischen Kategorien beschrieben: arithmetische Ausdrücke Aexp, boolesche Ausdrücke Bexp und Anweisungen Com. Außerdem braucht man noch numerische Literale Num und einen unendlichen Vorrat Var an (Programm-)Variablen. Die Darstellung der numerischen Literale und (Programm-)Variablen ist nicht weiter relevant, im Folgenden werden die Dezimaldarstellung (z.B. `120`, `5`) bzw. einfache Zeichenketten wie `x`, `catch22` verwendet.

Variablenkonvention: Um nicht ständig die syntaktische Kategorie einer (Meta-)Variable angeben zu müssen, bezeichne a stets einen arithmetischen Ausdruck, b einen booleschen, c eine Anweisung, n ein numerisches Literal und x eine Programmvariable aus Var – entsprechende Dekorationen mit Indizes, Strichen, usw. eingeschlossen. Dabei muss man zwischen einer Meta-Variablen x, die für eine beliebige, aber feste Programmvariable aus Var steht, und den konkreten Programmvariablen wie `x`, `y` selbst unterscheiden.

Definition 1 (Syntax von While). Die Syntax von Ausdrücken und Anweisungen sei durch folgende kontext-freie BNF-Grammatik gegeben:

$$
\begin{array}{lcl}
\text{Aexp} \quad a & ::= & n \mid x \mid a_1 \texttt{ - } a_2 \mid a_1 \texttt{ * } a_2 \\
\text{Bexp} \quad b & ::= & \texttt{true} \mid a_1 \texttt{ <= } a_2 \mid \texttt{not } b \mid b_1 \texttt{ \&\& } b_2 \\
\text{Com} \quad c & ::= & \texttt{skip} \mid x \texttt{ := } a \mid c_1\texttt{; } c_2 \mid \texttt{if (}b\texttt{) then } c_1 \texttt{ else } c_2 \mid \texttt{while (}b\texttt{) do } c
\end{array}
$$

Obwohl diese Sprache minimalistisch ist, sind viele weitere Programmkonstrukte ausdrückbar: Beispielsweise sind a_1 `+` a_2, a_1 `==` a_2, `false` und b_1 `||` b_2 nur syntaktischer Zucker für

$$
a_1 \texttt{ - (0 - } a_2\texttt{)}, \qquad \texttt{(}a_1 \texttt{ <= } a_2\texttt{) \&\& (}a_2 \texttt{ <= } a_1\texttt{)},
$$
$$
\texttt{not true} \quad \text{und} \quad \texttt{not ((not } b_1\texttt{) \&\& (not } b_2\texttt{))}
$$

Durch diese Reduktion auf das Wesentliche wird es einfacher, eine Semantik anzugeben und Beweise zu führen, da weniger Fälle berücksichtigt werden müssen.

Obige Grammatik ist mehrdeutig, z. B. hat `while (true) do skip; x := y` zwei verschiedene Ableitungsbäume, d.h. zwei verschiedene Syntaxbäume. Da die Semantik aber erst auf dem Syntaxbaum aufbaut, ist das nicht wesentlich: Man kann immer mittels zusätzlicher Klammern den gewünschten Baum eindeutig beschreiben.

2.2 Zustand

While enthält Zuweisungen an (Programm-)Variablen x `:=` a und Zugriff auf Variablen in arithmetischen Ausdrücken. Ein *Zustand* modelliert den Speicher für diese Variablen, der sich während der Ausführung eines Programms von einem Anfangszustand in einen Endzustand entwickelt. Für das

formale Modell ist ein Zustand, hier üblicherweise mit σ bezeichnet, eine Abbildung von Var nach $\mathbb{Z}$, die jeder Variablen x einen Wert $\sigma(x)$ zuordnet. Die Menge aller dieser Zustände sei Σ.

Ein realer Computer muss natürlich viele weitere Informationen im Zustand speichern, die nicht in einem (abstrakten) Zustand aus Σ enthalten sind, z.B. die Zuordnung von Variablen zu Speicheradressen. Diese weiteren Informationen sind aber für die Beschreibung des Verhaltens von While nicht relevant, der abstrakte Zustand und das Modell abstrahieren also davon.

2.3 Semantik von Ausdrücken

Ausdrücke in While liefern einen Wert (Zahl oder Wahrheitswert) zurück, verändern aber den Zustand nicht. Da sie Variablen enthalten können, wird der Wert erst durch einen Zustand endgültig festgelegt.

Für eine Zahl n in Programmdarstellung, in unserem Fall Dezimaldarstellung, liefere $\mathcal{N}[\![n]\!]$ den Wert der Zahl als Element von $\mathbb{Z}$. Beispiel: $\mathcal{N}[\![\texttt{123}]\!] = 123$, wobei $\texttt{123} \in \mathsf{Aexp}$ ein arithmetischer Ausdruck und $123 \in \mathbb{Z}$ eine ganze Zahl ist.

Definition 2 (Semantik arithmetischer Ausdrücke). Für beliebige arithmetische Ausdrücke a definiert $\mathcal{A}[\![a]\!]\sigma$ rekursiv über den Syntaxbaum den Wert von a im Zustand σ:

$$\begin{array}{rcl} \mathcal{A}[\![n]\!]\sigma & = & \mathcal{N}[\![n]\!] \\ \mathcal{A}[\![x]\!]\sigma & = & \sigma(x) \\ \mathcal{A}[\![a_1 \texttt{ - } a_2]\!]\sigma & = & \mathcal{A}[\![a_1]\!]\sigma - \mathcal{A}[\![a_2]\!]\sigma \\ \mathcal{A}[\![a_1 \texttt{ * } a_2]\!]\sigma & = & \mathcal{A}[\![a_1]\!]\sigma \cdot \mathcal{A}[\![a_2]\!]\sigma \end{array}$$

$\mathcal{A}[\![\_]\!]$ ist also eine Funktion des Typs $\mathsf{Aexp} \Rightarrow \Sigma \Rightarrow \mathbb{Z}$, definiert über strukturelle (primitive) Rekursion über den Syntaxbaum arithmetischer Ausdrücke.

Beispiel 2. Was ist $\mathcal{A}[\![a_1 \texttt{ + } a_2]\!]\sigma$?
$a_1 \texttt{ + } a_2$ ist syntaktischer Zucker für $a_1 \texttt{ - (0 - } a_2\texttt{)}$. Damit gilt:

$$\begin{aligned} \mathcal{A}[\![a_1 \texttt{ + } a_2]\!]\sigma = \mathcal{A}[\![a_1 \texttt{ - (0 - } a_2\texttt{)}]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma - \mathcal{A}[\![\texttt{0 - } a_2]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma - (\mathcal{A}[\![\texttt{0}]\!]\sigma - \mathcal{A}[\![a_2]\!]\sigma) \\ = \mathcal{A}[\![a_1]\!]\sigma - (0 - \mathcal{A}[\![a_2]\!]\sigma) = \mathcal{A}[\![a_1]\!]\sigma + \mathcal{A}[\![a_2]\!]\sigma \end{aligned}$$

Die syntaktische Abkürzung $a_1 \texttt{ + } a_2$ ist also sinnvoll.

Analog zu arithmetischen Ausdrücken lässt sich der Wert von booleschen Ausdrücken definieren. Dabei bezeichnen **tt** und **ff** die beiden Wahrheitswerte in $\mathbb{B}$.

Definition 3 (Semantik boolescher Ausdrücke).
Die Semantik boolescher Ausdrücke $\mathcal{B}[\![\_]\!]$ ist definiert durch:

$$\begin{array}{rcl} \mathcal{B}[\![\texttt{true}]\!]\sigma & = & \mathbf{tt} \\ \mathcal{B}[\![a_1 \texttt{ <= } a_2]\!]\sigma & = & \mathcal{A}[\![a_1]\!]\sigma \leq \mathcal{A}[\![a_2]\!]\sigma \\ \mathcal{B}[\![\texttt{not } b]\!]\sigma & = & \neg\mathcal{B}[\![b]\!]\sigma \\ \mathcal{B}[\![b_1 \texttt{ \&\& } b_2]\!]\sigma & = & \mathcal{B}[\![b_1]\!]\sigma \wedge \mathcal{B}[\![b_2]\!]\sigma \end{array}$$

Übung: Was ist $\mathcal{B}[\![a_1 \texttt{ == } a_2]\!]\sigma$? Ist die Abkürzung sinnvoll? Was wäre, wenn auch Ausdrücke den Zustand ändern könnten?