Kapitel 3

Syntax und Semantik

In diesem Kapitel führen wir grundlegende Konzepte für logische Sprachen ein. Wir tun dies am Beispiel einer Sprache A, die arithmetische Ausdrücke als Formeln hat (zum Beispiel $x * (y + 8)$).

3.1 Syntax

Die Formeln von A sind Ausdrücke, die aus ganzen Zahlen, Variablen, Addition und Multiplikation gebildet werden. Hier ist ein Beispiel:

$$4 * (2 * x + y * z)$$

Man unterscheidet zwischen *syntaktischen* und *semantischen Aspekten* von logischen Sprachen. Entsprechend spricht man von der *Syntax* und der *Semantik* einer logischen Sprache. Die Formeln einer logischen Sprache zählen zur Syntax und werden als mathematische Objekte dargestellt.

Die Syntax von logischen Sprachen kann man durch *Grammatiken* definieren. Die Grammatik in Abbildung 3.1 definiert die Syntax der logischen Sprache A. A hat drei Arten von syntaktischen Objekten: Konstanten, Variablen und Ausdrücke. Für Ausdrücke gibt es vier Varianten: (1) Ausdrücke die nur aus einer Konstante bestehen, (2) Ausdrücke die nur aus einer Variablen bestehen, (3) Summen, und (4) Produkte.

Die Grammatik legt die Menge *Var* der Variablen nicht fest. Damit erreicht man eine nach *Var parametrisierte Darstellung* der Syntax von A. Wenn wir $Var = \emptyset$ setzen, haben wir Ausdrücke ohne Variablen. Wenn wir $Var = \mathbb{N}$ setzen, können wir Ausdrücke mit beliebig vielen verschiedenen Variablen bilden. Wenn Sie

c	$\in$	$Kon = \mathbb{Z}$	*Konstante*
x	$\in$	Var	*Variable*
a	$\in$	$Aus =$	*Ausdruck*
		c	*Konstante*
		$\mid x$	*Variable*
		$\mid a_1 + a_2$	*Summe*
		$\mid a_1 * a_2$	*Produkt*

Abbildung 3.1: Syntax von A

die Parametrisierung irritiert, nehmen Sie vorerst einfach an, dass die Grammatik $Var = \mathbb{N}$ festlegt.

Die Grammatik definiert die Menge der Ausdrücke durch strukturelle Rekursion gemäß der folgenden Gleichung:

$$Aus = Kon \uplus Var \uplus (Aus \times Aus) \uplus (Aus \times Aus)$$

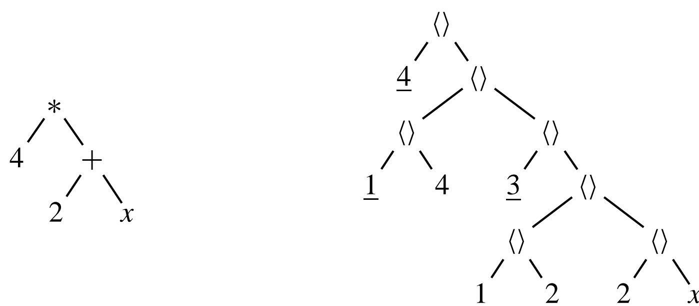
Damit können wir beispielsweise den durch

$$4 * (2 + x)$$

beschriebenen Ausdruck wie folgt darstellen (die Variantennummern sind unterstrichen):

$$\begin{aligned}&\langle \underline{4},\ \langle\langle \underline{1},\ 4\rangle,\\ &\quad \langle \underline{3},\ \langle\langle \underline{1},\ 2\rangle,\\ &\qquad \langle \underline{2},\ x\rangle\rangle\rangle\rangle\rangle\end{aligned}$$

Diese Darstellung folgt der Baumstruktur des Ausdrucks:



Da die Standardnotation für die Objekte in `Aus` offensichtlich viel zu schwerfällig ist, legt die Grammatik eine *Zusatznotation* fest, die sich an der üblichen Notation für arithmetische Ausdrücke orientiert. Für den gerade dargestellten Ausdruck liefert die Grammatik die Zusatznotation $4 * (2 + x)$.

Man unterscheidet zwischen *abstrakter und konkreter Syntax*. Bei einer abstrakten Syntax werden die syntaktischen Objekte wie gerade gezeigt durch geschachtelte Tupel mit Variantennummern dargestellt. Bei einer konkreten Syntax werden die syntaktische Objekte durch Zeichenreihen dargestellt. In dieser Vorlesung interessieren wir uns nur für abstrakte Syntax. Im Kontext von Programmiersprachen ist die Definition und Verarbeitung von konkreter Syntax eine gut verstandene Angelegenheit.

Wir legen großen Wert darauf, die syntaktischen Objekte präzise als mathematische Objekte zu definieren. Dagegen sind wir bei der Festlegung und Verwendung der Zusatznotation für A vergleichsweise nachlässig und vertrauen auf die mathematische Erfahrung des Lesers. Beispielsweise ist die Notation

$$1 + 2 + 3$$

für Ausdrücke in *Aus* unzulässig, da sie zwei Lesarten hat (solange wie keine weiteren Vorgaben machen):

$$(1 + 2) + 3 \quad \text{oder} \quad 1 + (2 + 3)$$

Generell ist der Gebrauch einer Notationen nur dann zulässig, wenn der Kontext dafür sorgt, dass es nur eine Lesart gibt.

Wenn man festlegt, wie fehlende Klammern ergänzt werden sollen, kann man Klammern einsparen und trotzdem eindeutig sein. Für die Zusatznotation für A vereinbaren wir die folgenden Regeln:

- Die Infixoperatoren $+$ und $*$ werden linksassoziativ gruppiert.
- Der Infixoperator $+$ steht auf höherer Rangstufe als $*$.

Mit diesen Regeln muss die Notation

$$3 * x + 4 + y$$

genauso wie

$$((3 * x) + 4) + y$$

interpretiert werden.

Die durch die Grammatik definierte Zusatznotation bringt noch eine weitere Komplikation mit sich. Beispielsweise können wir nur mit zusätzlicher Information entscheiden, ob mit der Notation

$$4$$

die Zahl 4 oder der Ausdruck $\langle 1, 4\rangle$, der nur aus der Zahl 4 besteht, gemeint ist. Diese Ambiguität wird im Folgenden keine Schwierigkeiten machen, da der Kontext stets eindeutig festlegen wird, ob eine Konstante oder ein Ausdruck gemeint ist.

Mithilfe von struktureller Rekursion definieren wir eine Funktion, die zu einem Ausdruck in *Aus* die Menge der in ihm *vorkommenden Variablen* liefert:

$$\begin{aligned} &VV \in Aus \to \mathcal{P}(Var) \\ VV(c) &= \emptyset \\ VV(x) &= \{x\} \\ VV(a_1 + a_2) &= VV(a_1) \cup VV(a_2) \\ VV(a_1 * a_2) &= VV(a_1) \cup VV(a_2) \end{aligned}$$

Beispiel. Seien $x, y \in Var$. Dann gilt $VV(x + 3 * y + 7)) = \{x, y\}$.

Die ersten zwei Regeln der Definition von VV sind in einer Kurzschreibweise aufgeschrieben. Eigentlich müssten wir

$$\begin{aligned} VV(\langle 1, c\rangle) &= \emptyset \\ VV(\langle 2, x\rangle) &= \{x\} \end{aligned}$$

schreiben, da VV auf Ausdrücken definiert ist und Konstanten und Variablen als solche keine Ausdrücke sind. Andererseits ist der Kontext der Definition von VV ausreichend, um die fehlenden Variantennummern automatisch zu ergänzen.

Es ist hilfreich, die Syntax von A mithilfe der vertrauten Notation von Standard ML zu definieren. Dies kann beispielsweise wie folgt geschehen:

```
type con = int      (* Konstanten *)
type var = int      (* Variablen  *)

datatype exp =      (* Ausdrücke  *)
    Con of con
  | Var of var
  | Add of exp * exp
  | Mul of exp * exp
```

 © G. Smolka

Dabei haben wir $Var = \mathbb{Z}$ gesetzt. Wenn wir annehmen, dass x die Variable 1 ist, kann der durch

$$5 + (-3 * x)$$

beschriebene Ausdruck in Standard ML wie folgt beschrieben werden:

```
Add(Con 5, Mul(Con ~3, Var 1))
```

Eine Prozedur, die die Anzahl der in einem Ausdruck vorkommenden Additionen liefert, kann wie folgt deklariert werden:

```
fun noa (Con c)       = 0
  | noa (Var y)       = 0
  | noa (Add(e1,e2)) = 1 + noa e1 + noa e2
  | noa (Mul(e1,e2)) = noa e1 + noa e2

val noa : exp -> int
```

Mit mathematischer Notation kann eine entsprechende Funktion wie folgt definiert werden:

$$noa \in Aus \to \mathbb{N}$$

$$\begin{aligned} noa(c) &= 0 \\ noa(x) &= 0 \\ noa(a_1 + a_2) &= 1 + noa(a_1) + noa(a_2) \\ noa(a_1 * a_2) &= noa(a_1) + noa(a_2) \end{aligned}$$

Bemerkung

Wahrscheinlich finden Sie die gerade gelesenen Ausführungen reichlich verwirrend. Versuchen Sie, die Vielfalt der Konzepte wie folgt zu ordnen.

Zunächst wurden die Menge *Kon* der Konstanten und die Menge *Aus* der Ausdrücke für die Sprache A wie folgt definiert:

$$Kon = \mathbb{Z}$$

$$Aus = Kon \uplus Var \uplus (Aus \times Aus) \uplus (Aus \times Aus)$$

Dabei kann die Menge *Var* der Variablen beliebig vorgegeben werden. Die Definition von Summen ($\uplus$) finden Sie in Kapitel 1.

Bis hierhin ist also alles ganz einfach. Diese Definitionen sind der technische Kern dieses Abschnitts.

Das zweite Anliegen des Abschnitts ist es, verschiedene Notationen für die Beschreibung von syntaktischen Objekten einzuführen. Das ist der komplizierte Teil der Übung.

Die einfachste dieser Notationen beruht auf der Programmiersprache Standard ML und wurde zum Schluß eingeführt. Diese Notation ist so einfach, dass Sie automatisch verarbeitet werden kann. Wenn Ihnen die Darstellung von Ausdrücken mit Standard ML noch nicht völlig klar ist, sollten Sie mit dem Interpreter experimentieren.

Die durch die Grammatik eingeführte mathematische Zusatznotation für Ausdrücke ist am kompaktesten, greift dafür aber tief in die notationale Trickkiste. Sie erfordert Leser mit hinreichender mathematischer Erfahrung. Sie ist nicht vollständig definiert und kann folglich auch nicht automatisch verarbeitet werden. Es ist also keineswegs überraschend, wenn Ihnen verschiedene Details unklar sind. Mathematische Notationen ähnelt in mancher Hinsicht natürlichen Sprachen (zum Beispiel Französisch): Man lernt sie am Besten durch den Gebrauch (also durch Beispiele), direkte Erklärungen (zum Beispiel der Grammatik) sind für den Anfänger eher verwirrend.

3.2 Denotation

Betrachten Sie den Ausdruck

$$(x + 3) * y$$

Wenn wir Werte für die Variablen x und y vorgeben, können wir den Wert des Ausdrucks bestimmen. Beispielsweise hat der Ausdruck den Wert 8, wenn wir $x = 1$ und $y = 2$ vorgeben.

Eine *Belegung* ist eine Funktion $Var \to \mathbb{Z}$, die jeder Variablen einen Wert zuordnet. Die Menge aller Belegungen bezeichnen wir mit Σ, und einzelne Belegungen mit σ:

$$\sigma \in \Sigma \stackrel{\text{def}}{=} Var \to \mathbb{Z}$$

Statt von Belegungen spricht man auch von *Valuationen*. Im Kontext von Programmiersprachen bezeichnet man Belegungen als Umgebungen.

Gegeben eine Belegung, können wir den Wert jedes Ausdrucks bestimmen, indem wir für die vorkommenden Variablen die durch die Belegung festgelegten Werte verwenden. Umgekehrt können wir jedem Ausdruck eine Funktion $\Sigma \to \mathbb{Z}$ zuordnen, die jeder Belegung einen Wert zuordnet. Diese Idee formalisieren wir mit einer Funktion

$$\mathcal{D} \in Aus \to (\Sigma \to \mathbb{Z})$$

 © G. Smolka

die wir durch strukturelle Rekursion über *Aus* definieren:

$$\mathcal{D} \in Aus \to \Sigma \to \mathbb{Z}$$

$$\begin{aligned}
\mathcal{D}(c)\sigma &= c \\
\mathcal{D}(x)\sigma &= \sigma(x) \\
\mathcal{D}(a_1 + a_2)\sigma &= \mathcal{D}(a_1)\sigma + \mathcal{D}(a_2)\sigma \\
\mathcal{D}(a_1 * a_2)\sigma &= \mathcal{D}(a_1)\sigma \cdot \mathcal{D}(a_2)\sigma
\end{aligned}$$

Die Funktion $\mathcal{D}$ wird als *Denotationsfunktion* bezeichnet. Sie ordnet jedem Ausdruck a ein mathematisches Objekt $\mathcal{D}(a)$ zu, dass als *Denotation des Ausdrucks* bezeichnet wird. Die Denotation $\mathcal{D}(a)$ eines Ausdrucks a ist die Funktion $\Sigma \to \mathbb{Z}$, die jeder Belegung σ den Wert des Ausdrucks für σ zuordnet.

Wir bezeichnen die Funktion $\mathcal{D}$ auch als die *denotationale Semantik* der Sprache A.

Proposition 3.2.1 *Sei $a \in Aus$ ein Ausdruck und seien $\sigma, \sigma' \in \Sigma$ Belegungen, die auf allen in a vorkommenden Variablen übereinstimmen. Dann: $\mathcal{D}(a)\sigma = \mathcal{D}(a)\sigma'$.*

Beweis Durch strukturelle Induktion über $a \in Aus$. □

3.3 Äquivalenz

Zwei Ausdrücke heißen *äquivalent* genau dann, wenn sie die gleiche Denotation haben. Wir schreiben

$$a_1 \models\mid a_2 \overset{\text{def}}{\iff} \mathcal{D}(a_1) = \mathcal{D}(a_2)$$

Machen Sie sich klar, dass

$$\{\, (a_1, a_2) \in Aus \times Aus \mid a_1 \models\mid a_2 \,\}$$

eine Äquivalenzrelation auf *Aus* ist. Wir bezeichnen diese Äquivalenzrelation mit $\models\mid$.

Offensichtlich gilt $a_1 \models\mid a_2$ genau dann, wenn die Ausdrücke a_1 und a_2 für alle Belegungen denselben Wert haben. Beispielsweise gilt für beliebige Ausdrücke $a_1, a_2, a_3 \in Aus$:

$$\begin{aligned}
a_1 + a_2 &\models\mid a_2 + a_1 \\
a_1 * (a_2 + a_3) &\models\mid a_1 * a_2 + a_1 * a_3
\end{aligned}$$

 © G. Smolka

Die Relation $\models\!\!\mid$ beschreibt die sogenannte *denotationale Gleichheit* für Ausdrücken von A. Diese ist uns vom Rechnen mit arithmetischen Ausdrücken wohlbekannt. Neben der denotationalen Gleichheit haben wir noch die *syntaktische Gleichheit*, die Ausdrücke von A als syntaktische Objekte vergleicht. Machen Sie sich klar, dass

$$1 + 2 \models\!\!\mid 2 + 1 \qquad \textit{denotationale Gleichheit}$$

kein Widerspruch ist zu

$$1 + 2 \neq 2 + 1 \qquad \textit{syntaktische Gleichheit}$$

Die Sprache A ist relativ zu einer vorgegebenen Variablenmenge *Var* definiert. Wenn wir zwei Mengen *Var* und Var' betrachten, bekommen wir zwei Sprachen A_{Var} und $A_{Var'}$. Die Ausdrücke und Äquivalenzen der beiden Sprachen bezeichnen wir mit Aus_{Var}, $Aus_{Var'}$, $\models\!\!\mid_{Var}$ und $\models\!\!\mid_{Var'}$. Erwartungsgemäß gilt:

Proposition 3.3.1 (Variablenunabhängigkeit) *Sei $Var' \subseteq Var$. Dann:*

1. $Aus_{Var'} \subseteq Aus_{Var}$.
2. $\forall a_1, a_2 \in Aus_{Var'}: \quad a_1 \models\!\!\mid_{Var'} a_2 \iff a_1 \models\!\!\mid_{Var} a_2$.

3.4 Substitution

Wenn man mit Ausdrücken rechnet, ersetzt man oft Teilausdrücke durch andere Ausdrücke:

$$\begin{aligned} ((4*6)+8)*x &\models\!\!\mid (24+8)*x && \text{da } 4*6 \models\!\!\mid 24 \\ &\models\!\!\mid 32*x && \text{da } 24+8 \models\!\!\mid 32 \end{aligned}$$

Statt ersetzen sagt man auch *substituieren*.

Substitution ist eine grundlegende syntaktische Operation für logische Sprachen. Besonders wichtig sind Substitutionsoperationen, die Variablen durch Ausdrücke ersetzen.

Betrachten Sie den Ausdruck

$$3 * x + x$$

Dieser enthält zwei Auftreten der Variablen x. Wenn wir beide Auftreten von x duch den Ausdruck $x + 4$ ersetzen, bekommen wir den Ausdruck

$$3 * (x + 4) + (x + 4)$$

Seien $a, a' \in Aus$ Ausdrücke und $x \in Var$ eine Variable. Dann bezeichnen wir mit

$$a[a'/x]$$

den Ausdruck, den wir aus a erhalten, wenn wir alle Auftreten der Variablen x durch den Ausdruck a' ersetzen. Beispielsweise gilt:

$$(3 * x + x)[(x + 4)/x] \;=\; 3 * (x + 4) + (x + 4)$$

Für Beweise ist es hilfreich, die Substitionsfunktion $a[a'/x]$ formal durch strukturelle Rekursion zu definieren:

$$\begin{aligned}
_[_/_] &\in Aus \times Aus \times Var \to Aus \\
c[a/x] &= c \\
x[a/x'] &= \text{if } x = x' \text{ then } a \text{ else } x \\
(a_1 + a_2)[a/x] &= (a_1[a/x]) + (a_2[a/x]) \\
(a_1 * a_2)[a/x] &= (a_1[a/x]) * (a_2[a/x])
\end{aligned}$$

Das nachfolgende Lemma ist als *Substitutionslemma* bekannt. Es formuliert eine wichtige Verträglichkeitseigenschaft für Denotation und Substitution. Beim ersten Lesen sollten Sie Sie das Substitutionslemma überspringen und zunächst die Aussage des nachfolgenden Kongruenzsatzes verstehen. Das Substitutionslemma wird für den Beweis des Kongruenzsatzes benötigt.

Lemma 3.4.1 (Substitution) *Seien $a, a' \in Aus$, $x \in Var$ und $\sigma \in \Sigma$. Dann:*

$$\mathcal{D}(a[a'/x])\sigma = \mathcal{D}(a)(\sigma[(\mathcal{D}(a')\sigma)/x])$$

Beweis Durch Induktion über a.

Sei $a = c$. Dann:

$$\begin{aligned}
\mathcal{D}(a[a'/x])\sigma &= \mathcal{D}(c[a'/x])\sigma \\
&= \mathcal{D}(c)\sigma && \text{Def. der Substitutionsfunktion} \\
&= c && \text{Definition von } \mathcal{D} \\
&= \mathcal{D}(c)(\sigma[(\mathcal{D}(a')\sigma)/x]) && \text{Definition von } \mathcal{D} \\
&= \mathcal{D}(a)(\sigma[(\mathcal{D}(a')\sigma)/x])
\end{aligned}$$

Sei $a = x'$. Wir unterscheiden zwei Fälle.

© G. Smolka

1. *Sei* $x' = x$. Dann:

$$\begin{aligned}
\mathcal{D}(a[a'/x])\sigma &= \mathcal{D}(x[a'/x])\sigma \\
&= \mathcal{D}(a')\sigma && \text{Def. der Substitutionsfunkt.} \\
&= \mathcal{D}(x)(\sigma[(\mathcal{D}(a')\sigma)/x]) && \text{Definition von } \mathcal{D} \\
&= \mathcal{D}(a)(\sigma[(\mathcal{D}(a')\sigma)/x])
\end{aligned}$$

2. *Sei* $x' \neq x$. Dann:

$$\begin{aligned}
\mathcal{D}(a[a'/x])\sigma &= \mathcal{D}(x'[a'/x])\sigma \\
&= \mathcal{D}(x')\sigma && \text{Def. der Substitutionsfunkt.} \\
&= \sigma(x') && \text{Definition von } \mathcal{D} \\
&= (\sigma[(\mathcal{D}(a')\sigma)/x])(x') \\
&= \mathcal{D}(x')(\sigma[(\mathcal{D}(a')\sigma)/x]) && \text{Definition von } \mathcal{D} \\
&= \mathcal{D}(a)(\sigma[(\mathcal{D}(a')\sigma)/x])
\end{aligned}$$

Sei $a = a_1 + a_2$. Dann:

$$\begin{aligned}
\mathcal{D}(a[a'/x])\sigma &= \mathcal{D}((a_1 + a_2)[a'/x])\sigma \\
&= \mathcal{D}((a_1[a'/x]) + (a_2[a'/x]))\sigma && \text{Def. der Substitutionsfunkt.} \\
&= \mathcal{D}(a_1[a'/x])\sigma + \mathcal{D}(a_2[a'/x])\sigma && \text{Definition von } \mathcal{D} \\
&= \mathcal{D}(a_1)(\sigma[(\mathcal{D}(a')\sigma)/x]) && \text{Induktionsannahme} \\
&\qquad + \mathcal{D}(a_2)(\sigma[(\mathcal{D}(a')\sigma)/x]) \\
&= \mathcal{D}((a_1 + a_2))(\sigma[(\mathcal{D}(a')\sigma)/x]) && \text{Definition von } \mathcal{D} \\
&= \mathcal{D}(a)(\sigma[(\mathcal{D}(a')\sigma)/x])
\end{aligned}$$

Sei $a = a_1 * a_2$. Analog zu $a = a_1 + a_2$. □

3.5 Kongruenzeigenschaften

Sei $a \models\!\mid a'$. Die Äquivalenz der zwei Ausdrücke bleibt erhalten, wenn wir auf beiden Seiten eine Variable x durch einen Ausdruck b ersetzen:

$$a \models\!\mid a' \quad \Rightarrow \quad a[b/x] \models\!\mid a'[b/x]$$

Man spricht von einer *Kongruenzeigenschaft* und sagt, dass Substitution Äquivalenzen erhält.

Sei a eine Ausdruck, der einen Teilausdruck a_1 enthält. Sei a' der Ausdruck, den man aus a erhält, wenn man den Teilausdruck a_1 durch a_2 ersetzt. Wenn $a_1 \models\!\mid a_2$ gilt, dann sollte auch $a \models\!\mid a'$ gelten. Hier ist ein Beispiel für diese zweite Kongruenzeigenschaft:

$$((4 * 6) + 8) * x \models\!\mid (24 + 8) * x \qquad \text{da } 4 * 6 \models\!\mid 24$$

Der folgende Satz formuliert eine allgemeine Kongruenzeigenschaft, die die gerade gezeigten Kongruenzeigenschaften als Spezialfälle enthält.

Satz 3.5.1 (Kongruenz) *Seien $a_1, a_2, b_1, b_2 \in Aus$ und $x \in Var$. Dann:*

$$a_1 \models\!\mid a_2 \;\wedge\; b_1 \models\!\mid b_2 \quad\Rightarrow\quad a_1[b_1/x] \models\!\mid a_2[b_2/x]$$

Beweis Sei $a_1 \models\!\mid a_2$ und $b_1 \models\!\mid b_2$. Es genügt zu zeigen:

$$\forall \sigma \in \Sigma: \; \mathcal{D}(a_1[b_1/x])\sigma = \mathcal{D}(a_2[b_2/x])\sigma$$

Sei $\sigma \in \Sigma$. Dann:

$$\begin{aligned}
\mathcal{D}(a_1[b_1/x])\sigma &\models\!\mid \mathcal{D}(a_1)(\sigma[\mathcal{D}(b_1)\sigma/x]) && \text{Substitutionslemma}\\
&\models\!\mid \mathcal{D}(a_2)(\sigma[\mathcal{D}(b_1)\sigma/x]) && \text{da } \mathcal{D}(a_1) = \mathcal{D}(a_2)\\
&\models\!\mid \mathcal{D}(a_2)(\sigma[\mathcal{D}(b_2)\sigma/x]) && \text{da } \mathcal{D}(b_1) = \mathcal{D}(b_2)\\
&\models\!\mid \mathcal{D}(a_2[b_2/x])\sigma && \text{Substitutionslemma}
\end{aligned}$$

□

3.6 Signifikante Variablen

Der Wert eines Ausdrucks kann nur von solchen Variablen abhängen, die in ihm vorkommen (Proposition 3.2.1). Es kann aber durchaus sein, dass der Wert eines Ausdrucks nicht von jeder Variablen abhängt, die in ihm vorkommt. Beispielsweise hängt der Wert des Ausdrucks

$$x + 0 * y$$

nur von der Variablen x ab. Die Variablen, von denen der Wert eines Ausdrucks abhängt, bezeichnen wir als die *signifikanten Variablen* des Ausdrucks.

Wir wollen den Begriff der signifikanten Variable formal definieren. Dazu definieren wir ihn zunächst für Funktionen $\Sigma \to \mathbb{Z}$. Eine Variable $x \in Var$ heißt *signifikante Variable einer Funktion* $f \in \Sigma \to \mathbb{Z}$, wenn

$$\exists \sigma \in \Sigma \; \exists z \in \mathbb{Z}: \; f(\sigma) \neq f(\sigma[z/x])$$

Eine Variable $x \in Var$ heißt *signifikante Variable eines Ausdrucks* $a \in Aus$, wenn x eine signifikante Variable der Denotation $\mathcal{D}(a)$ von a ist. Die Menge aller signifikanten Variablen eines Ausdrucks $a \in Aus$ bezeichnen wir mit $SV(a)$.

Proposition 3.6.1 *Sei $a \in Aus$ mit $SV(a) = \emptyset$. Dann gibt es ein $z_0 \in \mathbb{Z}$ mit $\mathcal{D}(a) = \lambda\,\sigma \in \Sigma\,.\; z_0$.*

Proposition 3.6.2 *Sei $a \in Aus$. Dann $SV(a) \subseteq VV(A)$.*

Beweis Durch Widerspruch. Sei $x \in SV(a) - VV(a)$. Dann existiert ein $\sigma \in \Sigma$ und ein $z \in \mathbb{Z}$ mit $\mathcal{D}(a)\sigma \neq \mathcal{D}(a)(\sigma[z/x])$. Das ist wegen Proposition 3.2.1 ein Widerspruch, da σ und $\sigma[z/x]$ auf $VV(a)$ übereinstimmen. □

3.7 Zusammenfassung

Am Beispiel einer einfachen Sprache A haben wir grundlegende Konzepte logischer Sprachen kennengelernt. Man unterscheidet zwischen syntaktischen und semantischen Aspekten einer Sprache. Die syntaktischen Objekte einer Sprache führt man mithilfe einer Grammatik ein. Dabei werden zusammengesetzte syntaktische Objekte (zum Beispiel Ausdrücke) durch geschachtelte Tupel mit Variantennummern dargestellt. Die Syntax einer logischen Sprache beinhaltet fast immer Konstanten und Variablen.

Durch die Grammatik wird auch eine sogenannte *Zusatznotation* für die syntaktischen Objekte der Sprache festgelegt. Diese Notation ist für mathematisch erfahrene Leser gedacht und eignet sich nicht direkt für eine automatische Verarbeitung.

Die Semantik einer logischen Sprache wir typischerweise mithilfe einer Denotationsfunktion definiert, die den syntaktischen Objekten semantische Objekte zuordnet. Die Denotationsfunktion sollte mit struktureller Rekursion bezüglich der Rekursionsstruktur der syntaktischen Objekte definiert sein. Diese Forderung garantiert wichtige mathematische Eigenschaften. Man spricht von einer denotationalen Semantik.

Für die Formulierung einer denotationalen Semantik benötigt man sogenannte Belegungen. Das sind Funktionen, die Variablen Werte zuordnen.

Ein zentrales semantisches Konzept ist die Äquivalenz von syntaktischen Objekten. Mithilfe einer denotationalen Semantik kann Äquivalenz besonders einfach charakterisiert werden: Zwei syntaktische Objekte sind äquivalent genau dann, wenn sie dieselbe Denotation haben. Die Äquivalenz von syntaktischen Objekten entspricht der Gleichheit von mathematischen Ausdrücken.

 © G. Smolka

Die Einhaltung eines wichtigen Designprinzips für logische Sprachen wird durch den Kongruenzsatz formuliert: Die Denotation eines syntaktischen Objektes bleibt unverändert, wenn ein Teilobjekt durch ein äquivalentes Teilobjekt ersetzt wird.

Übungen

Aufgabe 3.1 (Arithmetische Ausdrücke) Sei die für A definierte Syntax wie folgt in Standard ML implementiert:

```
type con = int
type var = int

datatype exp =
    Con of con
  | Var of var
  | Add of exp * exp
  | Mul of exp * exp
```

1. Schreiben Sie eine Prozedur

   ```
   show : exp -> string
   ```

   die zu einen Ausdruck seine Darstellung mit Tupeln und Variantennummern liefert. Beispielsweise soll gelten:

   ```
   show (Add(Con 5, Mul(Con ~3, Var 1))) = "<3,<1,5>,<4,<1,~3>,<2,1>>>"
   ```

   Verwenden Sie die Prozedur `Int.toString` um ganze Zahlen in Zeichenreihen zu konvertieren.

2. Schreiben Sie eine Prozedur

   ```
   subst : exp -> exp -> var -> exp
   ```

   die zu zwei Ausdrücken a, a' und einer Variablen x den durch Substitution erhaltenen Ausdruck $a[a'/x]$ liefert.

3. Schreiben Sie eine Prozedur

   ```
   eval : exp -> (var -> int) -> int
   ```

   die zu einem Ausdruck a und einer Belegung σ die Zahl $\mathcal{D}(a)\sigma$ liefert.

© G. Smolka