Kapitel 3

Syntax und Semantik von arithmetischen Ausdrücken

In diesem Kapitel führen wir einige grundlegende Konzepte für logische Sprachen ein. Wir tun dies am Beispiel einer einfachen Sprache AA, deren Formeln arithmetische Ausdrücke über den ganzen Zahlen sind.

3.1 Syntax

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

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

Die Formel einer logischen Sprache müssen als mathematische Objekte dargestellt werden.

Man unterscheidet zwischen *syntaktischen* und *semantischen Aspekten* von logischen Sprachen. Entsprechend spricht man von *Syntax* und *Semantik*.

Die Syntax von logischen Sprachen kann man durch *Grammatiken* definieren. Die Grammatik in Abbildung 3.1 definiert die Syntax der logischen Sprache AA. AA 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 AA. Wenn wir $Var = \emptyset$ setzen, haben wir Ausdrücke ohne Variablen. Wenn wir $Var = \mathbb{N}$ setzen, können

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 AA

wir Ausdrücke mit beliebig vielen verschiedenen Variablen bilden. Wenn Sie die Parametrisierung irritiert, nehmen Sie einfach an, dass die Grammatik $Var = \mathbb{N}$ festlegt.

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

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

Da die Standardnotation für die Objekte in `Aus` viel zu schwerfällig ist, legt die Grammatik eine zusätzliche Notation für die Objekte in `Aus` fest, die sich an der üblichen Notation für arithmetische Ausdrücke orientiert. Sei $x \in Var$. Dann ist

$$\langle 4,\ \langle\langle 1, 4\rangle,\ \langle 3,\ \langle\langle 1, 2\rangle,\ \langle 2, x\rangle\rangle\rangle\rangle\rangle$$

die Standardnotation für einen Ausdruck in *Aus*. Mit der durch die Grammatik beschriebenen *Zusatznotation* können wir diesen Ausdruck lesbar wie folgt beschreiben:

$$4 * (2 + x)$$

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.

Wir legen großen Wert darauf, die syntaktischen Objekte präzise als mathematische Objekte zu definieren. Dagegen sind wir bei der Festlegung und Verwendung unserer Notationen 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 AA 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 zusätzlich 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 vorgeben 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:

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

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

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

$$V(\langle 1, c\rangle) = \emptyset$$
$$V(\langle 2, x\rangle) = \{x\}$$

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

Es ist hilfreich, die Syntax von AA 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
```

Dabei haben wir $\mathit{Var} = \mathbb{Z}$ 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
```

Die entsprechende Funktion kann mit mathematischer Notation wie folgt definiert

werden:

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

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 AA wie folgt definiert:

$$
\begin{aligned}
Kon &= \mathbb{Z} \\
Aus &= Kon \uplus Var \uplus Aus \times Aus \uplus Aus \times Aus
\end{aligned}
$$

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 Reife. 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 Denotationale Semantik

Eine *Belegung* ist eine Funktion, die Variablen Werte zuordnet. Statt von Belegungen spricht man auch von *Valuationen*. Im Kontext von Programmiersprachen bezeichnet man Belegungen als Umgebungen.

Im Kontext von AA verstehen wir unter einer Belegung eine totale Funktion $Var \to \mathbb{Z}$. Die Menge aller Belegungen bezeichnen wir mit Σ, und einzelne Belegungen mit σ:

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

Gegeben eine Belegung, können wir den Wert eines Ausdrucks berechnen, indem wir für die vorkommenden Variablen die durch die Belegung festgelegten Werte verwenden. Sei beispielsweise eine Belegung σ mit $\sigma(x) = 7$ gegeben. Dann hat der Ausdruck

$$3 * x + 4$$

den Wert 25.

Diese Idee realisieren wir mit einer Funktion

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

die wir durch strukturelle Rekursion über *Aus* definieren. Für $\mathcal{D}$ vereinbaren wir zwei Notationen:

$$\begin{aligned} \mathcal{D}[\![a]\!] &= \mathcal{D}(a) \\ \mathcal{D}[\![a]\!]\sigma &= (\mathcal{D}(a))(\sigma) \end{aligned}$$

Damit bekommen wir die folgende Definition:

$$\begin{aligned} \mathcal{D} &\in Aus \to (\Sigma \to \mathbb{Z}) \\ \mathcal{D}[\![c]\!]\sigma &= c \\ \mathcal{D}[\![x]\!]\sigma &= \sigma(x) \\ \mathcal{D}[\![a_1 + a_2]\!]\sigma &= \mathcal{D}[\![a_1]\!]\sigma + \mathcal{D}[\![a_2]\!]\sigma \\ \mathcal{D}[\![a_1 * a_2]\!]\sigma &= \mathcal{D}[\![a_1]\!]\sigma \cdot \mathcal{D}[\![a_2]\!]\sigma \end{aligned}$$

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. Man bezeichnet $\mathcal{D}$ auch als *denotationale Semantik* von AA.

3.3 Äquivalenz

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

$$a_1 \models\!\models a_2 \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \mathcal{D}[\![a_1]\!] = \mathcal{D}[\![a_2]\!]$$

Machen Sie sich klar, dass

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

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

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

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

Die Relation $\models\!\models$ formalisiert die *denotationale Gleichheit* arithmentischen Ausdrücken. Machen Sie sich klar, dass

$$1 + 2 \models\!\models 2 + 1$$

kein Widerspruch ist zu

$$1 + 2 \neq 2 + 1$$

Die Sprache AA ist relativ zu einer vorgegebenen Variablenmenge *Var* definiert. Wenn wir zwei Mengen *Var* und Var' betrachten, bekommen wir zwei Sprachen $\mathrm{AA}_{\mathit{Var}}$ und $\mathrm{AA}_{\mathit{Var}'}$. Die Ausdrücke und Äquivalenzen der beiden Sprachen bezeichnen wir mit $\mathit{Aus}_{\mathit{Var}}$, $\mathit{Aus}_{\mathit{Var}'}$, $\models\!\models_{\mathit{Var}}$ und $\models\!\models_{\mathit{Var}'}$. Erwartungsgemäß gilt:

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

1. $\mathit{Aus}_{\mathit{Var}'} \subseteq \mathit{Aus}_{\mathit{Var}}$.
2. $\forall a_1, a_2 \in \mathit{Aus}_{\mathit{Var}'}: \quad a_1 \models\!\models_{\mathit{Var}'} a_2 \iff a_1 \models\!\models_{\mathit{Var}} a_2$.

3.4 Substitution

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

$$((4 * 6) + 8) * x \models\!\!\models (24 + 8) * x \models\!\!\models 32 * x$$

Statt von Ersetzen spricht man auch von *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

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

Sei $a, a' \in \mathit{Aus}$ Ausdrücke und $x \in \mathit{Var}$ 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:

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

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 Ersetzungssatzes verstehen. Das Substitutionslemma wird für den Beweis des Ersetzungssatzes 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.

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 $a = a_1 + a_2$. □

Zwei Ersetzungseigenschaften

Sei $a \models\!\models a'$. Die Äquivalenz bleibt erhalten, wenn wir in a und a' eine Variable X durch einen Ausdruck b ersetzen:

$$a \models\!\models a' \;\Rightarrow\; a[b/X] \models\!\models a'[b/X]$$

Man spricht von einer Ersetzungseigenschaft 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 $a_1 \models\!\models a_2$ gilt, dann sollte auch $a \models\!\models a'$ gelten. Hier ist ein Beispiel für diese zweite Ersetzungseigenschaft:

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

Der folgende Satz formuliert beide Ersetzungseigenschaften.

Satz 3.4.2 (Ersetzung) *Seien* $a_1, a_2, b_1, b_2 \in$ *Aus und* $x \in$ *Var. Dann:*

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

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

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

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

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

3.5 Zusammenfassung

Am Beispiel einer sehr einfachen Sprache AA haben wir grundlegende logische Konzepte kennengelernt. Diese Konzepte werden wir bei allen weiteren Sprachen wiederfinden, die wir in dieser Vorlesung behandeln.

Zunächst muss die Syntax einer logischen Sprache definiert werden. Dies erfolgt mithilfe einer Grammatik. Dabei werden zusammengesetzte syntaktische Objekte (zum Beispiel Ausdrücke) durch geschachtelte Tupel mit Variantennummern dargestellt. Die Syntax von logischen Sprachen beinhaltet fast immer Konstanten und Variablen.

Durch die Grammatik wird auch eine mathematische Notation 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 kann oft mithilfe einer Denotationsfunktion definiert werden, die den syntaktischen Objekten semantische Objekte zuordnet. Die Denotationsfunktion muss mit struktureller Rekursion bezüglich der Rekursionsstruktur der syntaktischen Objekte definiert werden. 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 heißen äquivalent genau dann, wenn ihre Denotation gleich ist. Die Äquivalenz von syntaktischen Objekten entspricht der Gleichheit von mathematischen Ausdrücken.

Die Einhaltung eines wichtigen Designprinzips für logische Sprachen wird durch den Ersetzungssatz 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 AA 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.