GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Programmierparadigmen und Compilerbau

Syntax und Semantik
Teil 1

Woche 2
Sommersemester 2021
Arne Nägel

Syntax und Semantik

...definieren Programmiersprachen:

Syntax: Welche Zeichenfolgen sind *gültige* Programme?
(~ Korrektheit im Sinne einer Grammatik)

Semantik: Welche Bedeutung hat ein (gültiges) Programm?
(~ Funktion der Programmiersprache)

Beispiel: Syntaktisch korrekt (Subjekt – Prädikat – Objekt) sind:

```
Der Fuchs hat Hunger.
Das Auto schmilzt den Sonnenschein.
```

Die Semantik (=Bedeutung) erschließt sich jedoch nur im ersten Satz unmittelbar!

Semantiken

Verschiedene Typen

- Denotationale Semantik (mathematisch exakt)
- Transformationsbasierte Semantik (syntaktischer Zucker)
- Operationelle Semantik ()

Ausführung von Programmen via

1. Compiler (z.B. C, Haskell)
2. Interpreter (z.B. Python)
3. in Mischformen (z.B. Just-In-Time-Compiler oder `ghci`)

Syntax von Haskell (vereinfacht)

Betrachten folgende Syntax in *Backus-Naur-Form*

$$
\begin{array}{lll}
\langle \text{definition} \rangle & ::= & \langle \text{name} \rangle \langle \text{name} \rangle^* = \langle \text{ausdruck} \rangle \\
\langle \text{ausdruck} \rangle & ::= & \langle \text{wert} \rangle \mid \langle \text{name} \rangle \\
 & & \mid (\langle \text{ausdruck} \rangle \ \langle \text{ausdruck} \rangle) \\
 & & \mid (\langle \text{ausdruck} \rangle \langle \text{infix} \rangle \langle \text{ausdruck} \rangle) \\
 & & \mid \texttt{if} \ \langle \text{ausdruck} \rangle \ \texttt{then} \ \langle \text{ausdruck} \rangle \ \texttt{else} \ \langle \text{ausdruck} \rangle \\
\langle \text{infix} \rangle & ::= & * \mid + \mid - \mid /
\end{array}
$$

Funktionsdefinition/-rumpf

Argumente einer Funktion: *formale Parameter*.
Anzahl der Argumente: *Stelligkeit* der Funktion: $(ar(f))$

Elementare Datentypen

Als `<wert>` sind z.B. Daten folgender Typen möglich:

• **ganze Zahlen**	0,1,-3	Typ: `Int`
	n mit $\lvert n\rvert \leq 2^{31} - 1 = 2147483647$	
• **beliebig lange ganze Zahlen**	11122399387141	Typ: `Integer`,
• **rationale Zahlen**	3%7	Typ: `Ratio`
• **Gleitkommazahlen**	`3.456e+10`	Typ: `Floating`
• **Zeichen**	`'a'`	Typ: `Char`
• **Datenkonstruktoren**	`True, False`	Typ: `Bool`

Diese nennen wir auch *Basiswerte* (bis auf `Floating`)

Auch komplexere Datentypen sind möglich. Listen von Char erzeugen z.B. Zeichenketten "`Hallo`".

Funktions- und Variablennamen

- Die Bezeichner `<name>` repräsentieren Namen (z.B. x) von Variablen und Funktionen
- Im Sinne der formalen Sprachen handelt es sich um sog. Nichtterminale
- Diese können im Verlauf der Rechnung ersetzt werden.
- Bezeichner beginnen mit Kleinbuchstaben; nachfolgende Sonderzeichen sind möglich.

Syntax von Haskell (vereinfacht)

Betrachten folgende Syntax in *Backus-Naur-Form*

$$
\begin{aligned}
\langle \text{definition} \rangle &::= \langle \text{name} \rangle \langle \text{name} \rangle^* = \langle \text{ausdruck} \rangle \\
\langle \text{ausdruck} \rangle &::= \langle \text{wert} \rangle \mid \langle \text{name} \rangle \\
&\quad \mid (\langle \text{ausdruck} \rangle \ \langle \text{ausdruck} \rangle) && \text{Funktionsaufruf} \\
&\quad \mid (\langle \text{ausdruck} \rangle \langle \text{infix} \rangle \langle \text{ausdruck} \rangle) && \text{Operatoranwendung} \\
&\quad \mid \texttt{if} \ \langle \text{ausdruck} \rangle \ \texttt{then} \ \langle \text{ausdruck} \rangle \ \texttt{else} \ \langle \text{ausdruck} \rangle \\
\langle \text{infix} \rangle &::= * \mid + \mid - \mid / && \text{Bedingte Anweisung}
\end{aligned}
$$

Argumente einer Funktion: *formale Parameter*.
Anzahl der Argumente: *Stelligkeit* der Funktion: $(ar(f))$

Bedingte Anweisung (Fallunterscheidung)

Syntax: `if` ⟨Ausdruck⟩ `then` ⟨Ausdruck⟩ `else` ⟨Ausdruck⟩

„if“, „then“, „else“ sind reservierte Schlüsselworte

Der erste Ausdruck ist eine Bedingung (Typ `Bool`)

Typisierung: `if Bool ... then` typ `else` typ

```
(if x > y then x else y)
```

Klammerung

- Klammerung ist häufig optional, z.B:
- Sei f eine n-stellige Funktion. Dann gilt:
  $f\ x_1 \ldots x_n \equiv (\ldots (f\ x_1) \ldots x_n)$
- Achtung: $f\ x_1$ wird als (n-1) stellige Funktion interpretiert.
  (Dies bezeichnet man als *currying!)*
- Es gelten übliche „Punkt-vor-Strich-Regeln“:
  `1-2-3 ≡ ((1-2)-3)`
  `1-2*3 ≡ (1 - (2*3))`
- Funktionsanwendung vor bedingter Anweisung sowie bedingte Anweisung vor Operatoranwendung:
  `1 + if x<4 then g 1 * g x else 5`
  `≡ (1 + (if x<4 then ((g 1) * (g x)) else 5))`