3 Syntax von Programmiersprachen

Syntax ("Lehre vom Satzbau"):

- formale Beschreibung des Aufbaus der "Worte" und "Sätze", die zu einer Sprache gehören;
- im Falle einer Programmier-Sprache Festlegung, wie Programme aussehen müssen.

Hilfsmittel bei natürlicher Sprache:

- Wörterbücher;
- Rechtschreibregeln, Trennungsregeln, Grammatikregeln;
- Ausnahme-Listen;
- Sprach-“Gefühl”.

Hilfsmittel bei Programmiersprachen:

- Listen von Schlüsselworten wie `if`, `int`, `else`, `while` ...
- Regeln, wie einzelne Worte (Tokens) z.B. Namen gebildet werden.

  Frage:
  Ist `x10` ein zulässiger Name für eine Variable?
  oder `_ab$` oder `A#B` oder `0A?B` ...
- Grammatikregeln, die angeben, wie größere Komponenten aus kleineren aufgebaut werden.

  Frage:
  Ist ein `while`-Statement im `else`-Teil erlaubt?

- Kontextbedingungen.

  Beispiel:
  Eine Variable muss erst deklariert sein, bevor sie verwendet wird.

$\Longrightarrow$ formalisierter als natürliche Sprache

$\Longrightarrow$ besser für maschinelle Verarbeitung geeignet

Semantik (“Lehre von der Bedeutung”):

- Ein Satz einer (natürlichen) Sprache verfügt zusätzlich über eine Bedeutung, d.h. teilt einem Hörer/Leser einen Sachverhalt mit (↑Information)

- Ein Satz einer Programmiersprache, d.h. ein Programm verfügt ebenfalls über eine Bedeutung :-)

Die Bedeutung eines Programms ist

- alle möglichen Ausführungen der beschriebenen Berechnung (↑operationelle Semantik); oder
- die definierte Abbildung der Eingaben auf die Ausgaben (↑denotationelle Semantik).

Die Bedeutung eines Programms ist

- alle möglichen Ausführungen der beschriebenen Berechnung (↑operationelle Semantik); oder
- die definierte Abbildung der Eingaben auf die Ausgaben (↑denotationelle Semantik).

Achtung!

Ist ein Programm syntaktisch korrekt, heißt das noch lange nicht, dass es auch das "richtige" tut, d.h. semantisch korrekt ist !!!

3.1 Reservierte Wörter

- `int`
  → Bezeichner für Basis-Typen;
- `if`, `else`, `while`
  → Schlüsselwörter aus Programm-Konstrukten;
- `(`,`)`, `"`,`'`, `{`,`}`, `,`,`;`
  → Sonderzeichen.

3.2 Was ist ein erlaubter Name?

Schritt 1: Angabe der erlaubten Zeichen:

letter ::= `$` | `_` | `a` | ... | `z` | `A` | ... | `Z`

digit ::= `0` | ... | `9`

3.2 Was ist ein erlaubter Name?

Schritt 1: Angabe der erlaubten Zeichen:

letter ::= `$` | `_` | `a` | ... | `z` | `A` | ... | `Z`

digit ::= `0` | ... | `9`

- letter und digit bezeichnen Zeichenklassen, d.h. Mengen von Zeichen, die gleich behandelt werden.
- Das Symbol "|" trennt zulässige Alternativen.
- Das Symbol "..." repräsentiert die Faulheit, alle Alternativen wirklich aufzuzählen :-)

Schritt 2: Angabe der Anordnung der Zeichen:

name ::= letter (letter | digit)*

- Erst kommt ein Zeichen der Klasse letter, dann eine (eventuell auch leere) Folge von Zeichen entweder aus letter oder aus digit.
- Der Operator "*" bedeutet "beliebig ofte Wiederholung" ("weglassen" ist 0-malige Wiederholung :-).
- Der Operator "*" ist ein Postfix-Operator. Das heißt, er steht hinter seinem Argument.

Beispiele:

- ```
  _178
  Das_ist_kein_Name
  x
  _
  $Password$
  ```

... sind legale Namen :-)
  ```

- `5ABC`
  `!Hallo!`
  `x'`
  `-178`

... sind keine legalen Namen :-)

- `5ABC`
  `!Hallo!`
  `x'`
  `-178`

... sind keine legalen Namen :-)

Achtung:

Reservierte Wörter sind als Namen verboten !!!

3.3 Ganze Zahlen

Werte, die direkt im Programm stehen, heißen Konstanten.
Ganze Zahlen bestehen aus einem Vorzeichen (das evt. auch fehlt) und einer nichtleeren Folge von Ziffern:

$$\begin{array}{lll} \text{sign} & ::= & + \mid - \\ \text{number} & ::= & \text{sign ? digit digit*} \end{array}$$

3.3 Ganze Zahlen

Werte, die direkt im Programm stehen, heißen Konstanten.
Ganze Zahlen bestehen aus einem Vorzeichen (das evt. auch fehlt) und einer nichtleeren Folge von Ziffern:

$$\begin{array}{lll} \text{sign} & ::= & + \mid - \\ \text{number} & ::= & \text{sign ? digit digit*} \end{array}$$

- Der Postfix-Operator "?" besagt, dass das Argument eventuell auch fehlen darf, d.h. einmal oder keinmal vorkommt.
- Wie sähe die Regel aus, wenn wir führende Nullen verbieten wollen?

Beispiele:

- `-17`
  `+12490`
  `42`
  `0`
  `-00070`
  ... sind alles legale `int`-Konstanten.
- `"Hello World!"`
  `-0.5e+128`
  ... sind keine `int`-Konstanten.

Ausdrücke, die aus Zeichen (-klassen) mithilfe von

| (Alternative)

* (Iteration)

(Konkatenation) sowie

? (Option)

... aufgebaut sind, heißen reguläre Ausdrücke[a] (↑Automatentheorie).

[a] Gelegentlich sind auch ϵ, d.h. das "leere Wort" sowie $\emptyset$, d.h. die leere Menge zugelassen.

Reguläre Ausdrücke reichen zur Beschreibung einfacher Mengen von Worten aus.

- (letter letter)*
  – alle Wörter gerader Länge (über `a,...,z,A,...,Z`);
- letter* `test` letter*
  – alle Wörter, die das Teilwort `test` enthalten;
- `_` digit* `17`
  – alle Wörter, die mit _ anfangen, dann eine beliebige Folge von Ziffern aufweisen, die mit 17 aufhört;
- exp ::= (`e`|`E`)sign? digit digit*
  float ::= sign? digit digit* exp |
  sign? digit* (digit . | . digit) digit* exp?
  – alle Gleitkomma-Zahlen ...

Identifizierung von

- reservierten Wörtern,
- Namen,
- Konstanten

Ignorierung von

- White Space,
- Kommentaren

... erfolgt in einer ersten Phase (↑Scanner)

$\Longrightarrow$ Input wird mit regulären Ausdrücken verglichen und dabei in Wörter (“Tokens”) aufgeteilt.

In einer zweiten Phase wird die Struktur des Programms analysiert (↑Parser).