Algorithmen und Programmierung

5. Kapitel
Syntax

Prof. Matthias Werner
Professur Betriebssysteme

5.1 Einführung

- Woher weiß der Compiler, dass ein Programm nicht korrekt ist?
- Unterscheiden: **Grammatik** und **Semantik**

Grammatik (...wie wir es aus der Schule kennen.)

Formenlehre von Wörtern (**Morphologie**) und von Sätzen (**Syntax**)

- Bei Programmiersprachen spielt die Morphologie (praktisch) keine Rolle ➡ Grammatik und Syntax sind weitgehend synonym
- Die Bedeutung bzw. der Sinn (genannt: **Semantik**) des Satzes spielt keine Rolle

Syntax

Definition 5.1 (Syntax, informal)

Die Syntax einer Sprache ist eine Menge von Regeln, nach der Ausdrücke gebildet werden können.

- Die Aussage gilt für natürliche Sprachen (Deutsch, Englisch,...) wie für formale (C, Python, Java, aber auch mathematische Logik, Beschreibungssprachen, etc.)
  - Die Syntax natürlicher Sprachen ist aber weitaus komplizierter als die (der meisten) formalen Sprachen
- Ausdrücke im Sinne der Definition können z.B. Sätze, Formeln oder Programmstatements sein

Beispiele

Quelle: Foto von Guillaume Blanchard, GNU FDL; aus Till Tantau: Beamer-Beispiel-Lecture

- Wir wissen nicht, was der Text bedeutet[1]
- Es gibt aber offensichtlich Regeln, die eingehalten werden, z.B. „Hieroglyphen stehen in Zeilen“

[1] Dies gilt zumindest für den Autor dieser Folien. 😜

Beispiele (Forts.)

In Sprachen wie Deutsch und Englisch kann man Sätze nach der S-P-O-Regel bilden

- Die Informatiker lieben die mathematische Logik. ✔
- Ein Tisch beißt die anaphylaktische Algebra. ✔
- Der Tiger jagt heute eine Antilope. ✔
- Dieser Satz kein Verb. ✗

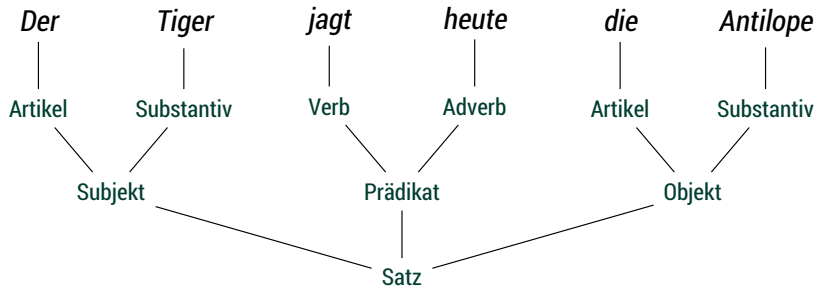
Achtung

Je besser man die Semantik versteht, um so schwerer fällt es oft, die Syntax zu erkennen.

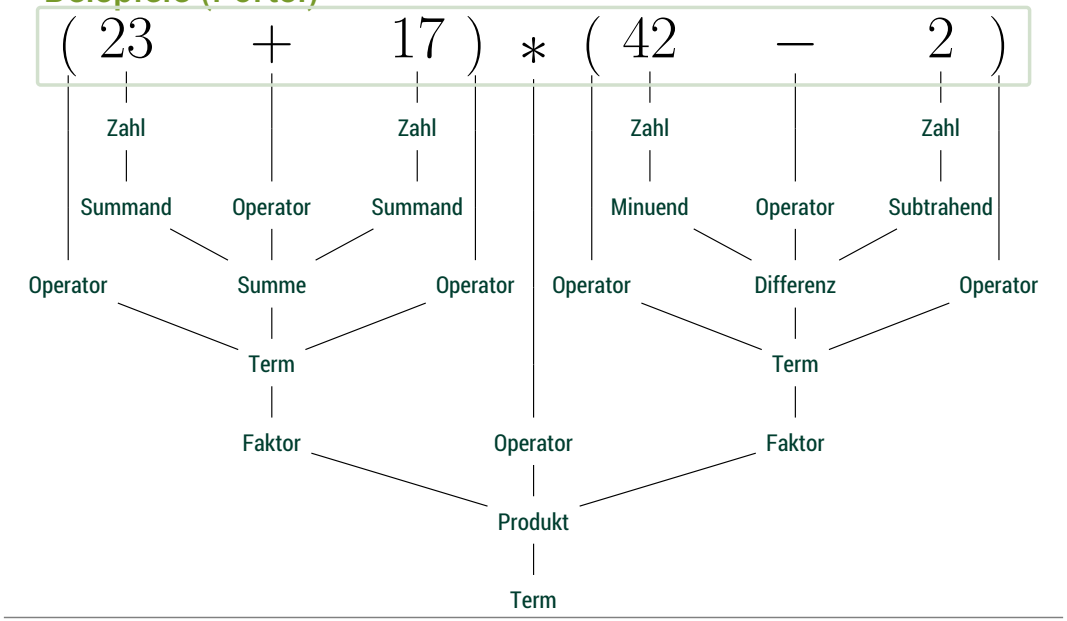
Auch mathematische Ausdrücke unterliegen syntaktischen Regeln:

- $(a + b) \cdot (c - d)$ ✔
- $(((a/b))$ ✗
- $a \cdot b + (c \cdot \frac{a+d}{e})$ ✔
- $:-)$ ✗

Beispiele (Forts.)



Beispiele (Forts.)



5.2 Formalisierung

Alphabet

Zur Formalisierung definieren wir eine Reihe von Begriffen:

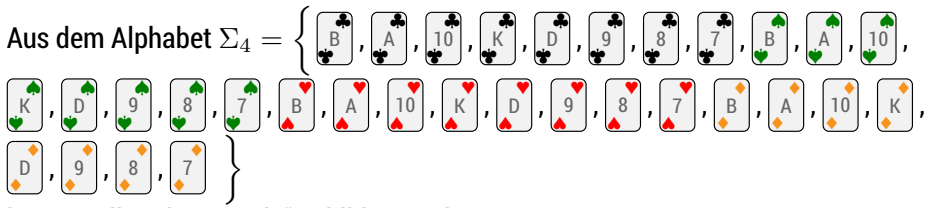
- Jede formale Sprache besteht aus **Symbolen** (Zeichen,*token*)
  - Ein Symbol ist die **kleinste** Einheit der Betrachtung **innerhalb der Sprache**
  - Symbole können z.B. einzelne Zeichen, Kombinationen davon oder auch ganze Wörter sein

Definition 5.2 (Alphabet)

*Jede endliche, nichtleere Menge Σ von Symbolen wird **Alphabet** genannt.*

- Aus den Elementen in Σ können **Ausdrücke** (auch: „Zeichenketten“, „Worte“, „Sätze“) gebildet werden
- Der leere Ausdruck wird mit ε bezeichnet

Alphabet (Forts.)

- Das Menge $\Sigma_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist hinreichend, um alle natürlichen Zahlen in Dezimalsystem darzustellen
- Aus dem Alphabet $\Sigma_2 = \{\bullet, \blacksquare\}$ können alle **Buchstaben** des Morsealphabets gebildet werden
- Aus dem Alphabet $\Sigma_3 = \{\bullet, \blacksquare, \sqcup\}$ können alle **Morse-Nachrichten** gebildet werden
- Aus dem Alphabet $\Sigma_4 = \{$  $\}$ können alle „Skat-Hände“ gebildet werden
- Das Alphabet $\Sigma = \{A, T, G, C\}$ wird zur Beschreibung von Gensequenzen genutzt

Formale Sprachen

Definition 5.3 (KLEENEsche Hülle)

*Die Menge aller aus den Symbolen von Σ bildbaren endlichen Zeichenketten und der leeren Zeichenkette ε heißt **Kleenesche Hülle** Σ^* von Σ.*

- Beispiel:
  $\Sigma = \{a, b\} \Rightarrow \Sigma^* = \{\varepsilon, a, b, aa, bb, ab, ba, aaa, aab, aba, baa, abb, bab, \ldots\}$
- Die Menge aller Zeichenketten ohne die leere Kette wird mit Σ^+ bezeichnet

Definition 5.4 (Formale Sprache)

*Jede Teilmenge $L \subseteq \Sigma^*$ ist eine **formale Sprache**.*

- Im Folgenden ist mit „Sprache“ immer „formale Sprache“ gemeint
- Man spricht von einer Sprache L **über** Σ

Formale Sprachen (Forts.)

- **Beispiele für Sprachen**:
  - Die Menge $\{AAA, AAC, GAA\}$ ➡ endliche Sprache
  - Die Menge aller Kartenpaare, die sich beim Skat im Skat befinden können ➡ endliche Sprache
  - Die Menge aller möglichen Schachspiele ➡ sehr groß, aber endlich
  - Menge aller C-Programmtexte ➡ unendliche Sprache
  - Die Menge aller Basensequenzen, die $GTAC$ enthalten ➡ unendlich

Formale Sprachen (Forts.)

- Es stellt sich die Frage, wie man die (Syntax der) Sprache beschreiben soll
- **Anders ausgedrückt**:
  Welche Ausdrücke/Zeichenketten sind in einer Sprache zulässig und welche nicht?
- Aufzählung ist unpraktisch ➡ oft ist die Anzahl der Ausdrücke sehr groß oder unendlich

- **Lösung**:
  Es wird beschrieben, wie man legale Ausdrücke **generieren** kann
  - Man spricht von einer **generativen Grammatik**

Generative Grammatik

- Bei einer generativen Grammatik der Sprache L wird neben dem Alphabet Σ von L ein zweites Alphabet V von Variablen benutzt.
- Zur Unterscheidung nennt man
  - ... die Elemente aus Σ ➡ **Terminalsymbole**
  - ... die Elemente aus V ➡ **Nichtterminalsymbole**
  - Wenn man abstrakt von diesen Symbolen spricht, werden die i.d.R. die Elemente aus Σ mit Kleinbuchstaben (a, b, c, ...) und die Elemente aus V mit Großbuchstaben bezeichnet
- Außerdem braucht man eine Menge von **Regeln** P, die beschreiben, wie man aus Elementen aus $(\Sigma \cup V)^+$ Elemente aus $(\Sigma \cup V)^*$ bildet:

$$P : (\Sigma \cup V)^+ \to (\Sigma \cup V)^*$$

  - Diese Regeln werden auch **Produktionsregeln** genannt
- Um ein Ausdruck zu generieren, beginnt man mit einem **Startsymbol** $S \in V$

Grammatiken

Nach der Art der Regeln P unterscheidet man verschiedene Arten von Grammatiken

Linksregulär: In jeder Regel aus P darf **links** nur ein Nichtterminalsymbol und **rechts** entweder ein Terminalsymbol oder ein Nichtterminalsymbol, gefolgt von einem Terminalsymbol stehen:

$$A \to a$$
$$A \to Ba$$

Rechtsregulär: In jeder Regel aus P darf **links** nur ein Nichtterminalsymbol und **rechts** ein Terminalsymbol, das von maximal einem Nichtterminalsymbol gefolgt wird, stehen:

$$A \to a$$
$$A \to aB$$

Grammatiken (Forts.)

Regulär: Entweder **alle** Regeln aus P entsprechen einer linksregulären Grammatik oder **alle** Regeln aus P entsprechen einer rechtsregulären Grammatik

Kontextfrei: In jeder Regel aus P darf links nur ein Nichtterminalsymbol stehen:

$$A \to \ldots$$

Kontextbehaftet: In jeder Regel aus P muss links ein Nichtterminalsymbol stehen, dass von (ggf. leeren) Zeichenketten eingerahmt wird; rechts müssen die gleichen Zeichenketten eine nichtleere Zeichenkette einrahmen:

$$\alpha A \beta \to \alpha \gamma \beta$$

Allgemein: Beliebige Regeln; keine Einschränkung

Grammatiken (Forts.)

- Alternativ nennt man ...
  - ... allgemeine Grammatiken auch **Typ-0-Grammatiken**
  - ... kontextbehaftete Grammatiken auch **Typ-1-Grammatiken**
  - ... kontextfreie Grammatiken auch **Typ-2-Grammatiken**
  - ... reguläre Grammatiken auch **Typ-3-Grammatiken**
- Die Grammatiken bilden die sogenannte **Chomsky[2]-Hierarchie**

Bildquelle: Wikipedia, Ministerio de Cultura de la Nación Argentina

[2] Noam Chomsky, *1928, US-amerikanischer Linguist und Bürgerrechtler

Grammatiken (Forts.)

- Mit jeder Typ-$(n-1)$-Grammatik kann man alles (und mehr) „machen“, was man mit Typ-n kann
- Die Verarbeitung allgemeinerer Sprachen verlangen komplexere Automaten
  - Für Typ-0 ist eine **Turing-Maschine** (allgemeines Modell eines Computers) notwendig[3]
  - Es existieren auch formale Sprachen, die nicht auf diese Weise generiert werden können
- In der theoretischen Informatik relevant

[3] ... aber nicht in allen Beziehungen hinreichend

Syntaxdiagramme

Wie können die Regeln einer Grammatik beschrieben werden?

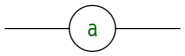
- Beschränken uns auf (maximal) kontextfreie Grammatiken
- Ansätze:
  - Syntaxdiagramme
  - (Erweiterte) Backus-Naur-Form

Syntaxdiagramme

- **Bestandteile**
  - abgerundete Kästchen ➡ Terminalsymbole
  - eckige Kästchen ➡ Nichtterminalsymbole
  - Verbindungen aus Linien und Pfeilen
- Jeder (in Pfeilrichtung) begehbare Weg ist ein valider Ausdruck

Syntaxdiagramme (Forts.)

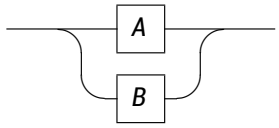
Terminalsymbol:



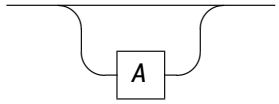
Nichtterminalsymbol:

Folge:

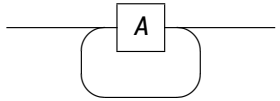
Alternative:



Option:



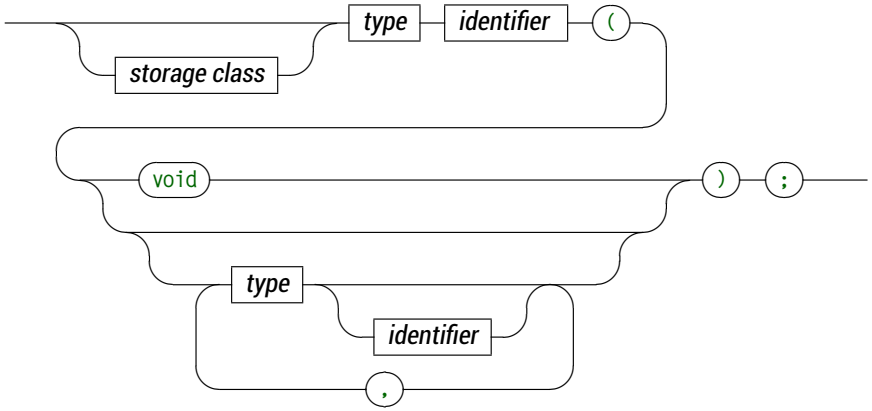
Wiederholung:



Syntaxdiagramme (Forts.)

- Beispiel Funktionsdeklaration in C (vereinfacht)

function declaration



Backus-Naur-Form

- Syntaxdiagramme sind zwar leicht zu lesen, nehmen aber viel Platz weg und sind nicht leicht zu zeichnen
- Kompaktere Alternative: **Backus-Naur-Form** (**BNF**)
- BNF benutzt **Meta-Zeichen**
  - `::=` Definitionszeichen
  - `|` Alternativzeichen
  - `< >` Nichtterminalklammern ➡ verwandeln eine beliebige Folge von Buchstaben, Ziffern und Leerzeichen in ein Nichtterminalsymbol
- Alle Symbole, die weder Metazeichen noch Nichtterminalsymbole sind, sind Terminalsymbole

Erweiterte Backus-Naur-Form

Erweiterte Backus-Naur-Form: EBNF

- Die BNF lässt sich direkt in kontextfreie Grammatik übersetzt
  - Benötigt aber (z.B. bei Wiederholungen) syntaktische Hilfsvariablen
- Erweiterung ➡ Erweiterte Backus-Naur-Form
- Wie BNF aber mit
  - ... Beschreibung **optionaler Teile** `[ ... ]`
  - ... Beschreibung von **Wiederholungen** `{ ... }`
- Syntaktische Unterschiede:
  - Beliebige Klammerung
  - Definitionszeichen ist „`=`“
  - Terminalzeichen werden in Anführungszeichen „`"`“ oder „`'`“ gesetzt
  - Nichtterminalzeichen nicht besonders gekennzeichnet
  - Ausdrücke enden mit Semikolon „`;`“

Erweiterte Backus-Naur-Form (Forts.)

Weitere Erweiterungen in der ISO/IEC 14977:1996(E), z.B.:

- Leerzeichen in Nichtterminalbezeichnern
  ➡ Sequenz dann mit Komma „`,`“ zwischen Folgeausdrücken

```
signed integer = sign, integer;
```

- Bestimmte Anzahl von Wiederholungen, z.B. „`4 * (...)`“
- Kommentare: „`(* Dies ist ein Kommentar *)`“

Definition 5.5 (Vollständigkeit einer (kontextfreien) Grammatik)

*Eine (E)BNF-Definition oder ein Syntaxdiagramm-Beschreibung ist **vollständig**, wenn es für jedes Nichtterminalsymbol, das auf einer rechten Regelseite auftaucht, auch eine Regel gibt, wo es auf linken Seite ist.*

5.3 Anwendung

Reguläre Grammatik in Aktion

- Es ist eine häufige Aufgabe, Daten zu finden, die eine bestimmte **Struktur** haben
- Dazu gibt es Werkzeuge, die mit Hilfe von **regulären Ausdrücken** suchen können
- Reguläre Ausdrücke beschreiben formale Sprachen, die durch eine reguläre Grammatik generiert werden
- Beispiele für Tools: `grep`, `sed`, `awk`, `perl`, `Python`, `C#`, ...
- Syntax weicht bei den einzelnen Tools voneinander ab
- Standardisierung in POSIX

Reguläre Grammatik in Aktion (Forts.)

- Da nur ein begrenzter Vorrat an Zeichen zur Verfügung steht, wird zwischen „normalen“ (Terminal-)Zeichen und **Metazeichen** unterschieden, die ein besondere Bedeutung haben
- Typisch sind folgende Metazeichen
  - `.` (Punkt): steht für (nahezu) jedes andere Zeichen
  - `[ ... ]` (eckige Klammern): steht für **eines** der Zeichen in der Klammer
  - `*` (Stern): der vorherige Ausdruck kann beliebig oft (auch überhaupt nicht) auftreten
  - `+` (Plus): der vorherige Ausdruck kann beliebig oft auftreten, aber wenigstens einmal
  - `{`n`,`m`}` (zwei Zahlen in geschweiften Klammern): der vorstehende Ausdruck tritt mindestens n-mal und maximal m-mal auf
  - `\` (Rückstrich): Die Wirkung des folgenden Metazeichens wird aufgehoben

Reguläre Grammatik in Aktion (Forts.)

Zeichenklassen

- Um verschiedene Zeichen zusammenzufassen, gibt es sogenannte Zeichenklassen, z.B.:
  - `[:alpha:]` Steht für alle Buchstaben *a, b, ..., z, A, B, ... C*
  - `[:digit:]` Steht für die Ziffern 0 ... 9
  - `[:alnum:]` Vereinigung von [:alpha:] und [:digit:]
  - `[:blank:]` Leerzeichen und Tabulator
  - `[:print:]` Steht für jedes druckbare Zeichen

Reguläre Grammatik in Aktion (Forts.)

Beispiel `grep`

- `grep` wird u.a. genutzt, um Zeilen in Dateien zu finden, die bestimmte Inhalte haben.
- Beispielsweise findet
  `grep -E "typedef ([_[:alpha:]][_[:alnum:]]*[[:blank:]]+)+ time_t ;" *.h`
  alle Definitionen von `time_t` in Header-Files.

```
> grep -E "typedef ([_[:alpha:]][_[:alnum:]]*[[:blank:]]+)+time_t;" *.h
utime.h:typedef __darwin_time_t        time_t;
utmp.h:typedef __darwin_time_t        time_t;
```

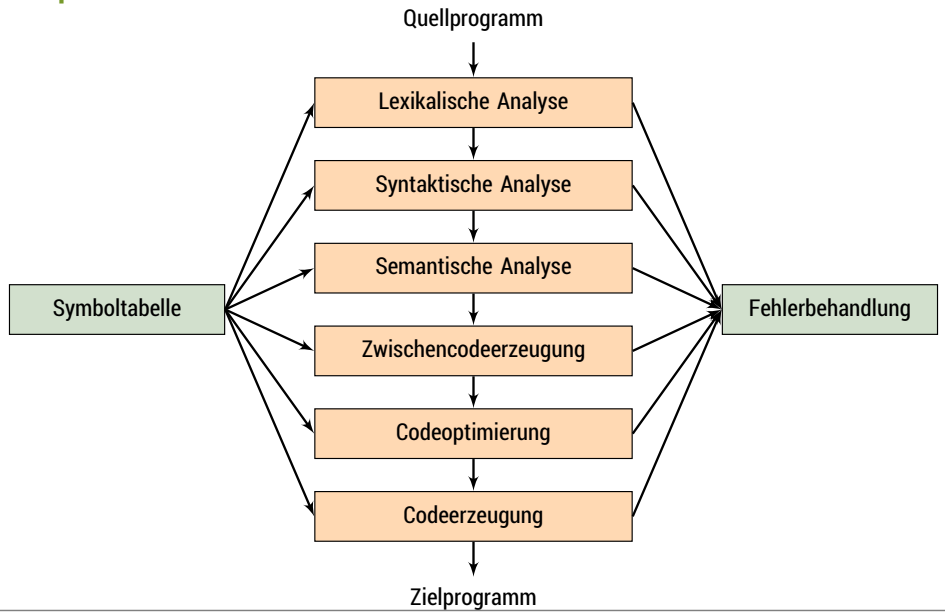

Reguläre Grammatik in Aktion (Forts.)

Beispiel `grep`

- Dagegen findet
  `grep -E "time_t [[:blank:]]+[_[:alpha:]][_[:alnum:]]*[[:blank:]]*\(" *.h`
  alle Funktionsdeklarationen, die `time_t` als Rückkgabetyp haben

```
> grep -E "time_t[[:blank:]]+[_[:alpha:]][_[:alnum:]]*[[:blank:]]*\(" *.h
time.h:time_t mktime(struct tm *) __DARWIN_ALIAS(mktime);
time.h:time_t time(time_t *);
time.h:time_t posix2time(time_t);
timeconv.h:time_t _long_to_time(long tlong);
timeconv.h:time_t _int_to_time(int tint);
```

Compilier



Formale Sprachen im Compiler

In einem modernen Compiler werden mehrmals formale Grammatiken verwendet:

- **Lexikalische Analyse** ➡ Identifizierung lexikalischer Elemente
  - Typisch: Reguläre Grammatik
- **Syntaktische Analyse**
  - Wurden die Syntaxregeln der Sprache eingehalten?
  - Typisch: Kontextfreie Sprache
- Für die **Semantische Analyse** werden in der Regel kontextsensitive Sprachen verwendet
  - ➡ Bei Interesse mehr in VL „Compilerbau“ (Prof. Rünger)

Token

- Die erste Stufe ist die Identifikation von lexikalischen Elementen
- Unterteilung des Eingabetexts in Schlüsselwörter, Bezeichner, Literale, ... ➡ **Token**
- Evtl. Verwerfen von Elementen
  - Z.B. werden in C Leerzeichen u.ä. ignoriert[4] (➡ *Whitespaces*)
  - Dagegen gehören Leerzeichen in Python zur Syntax ➡ Einrückungen haben Bedeutung

```c
/* token.c — greedy token parsing */

int main()
{
  int x=2,y=42, *px=&x;
  y=y/*px;
  /* use return value for result */;
  return y;
}
```

```
> gcc token.c -o token
> ./token
> echo $?
42
```

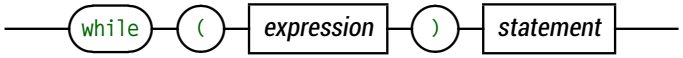
[4] Sie können aber zur Trennung anderer lexikalischer Elemente dienen.

Syntax am Beispiel von Schleifen

Betrachten als Beispiel Schleifen in C (vgl. Kapitel 4)

- Betrachten Regel für `while`-Schleife als Syntax-Diagramm:

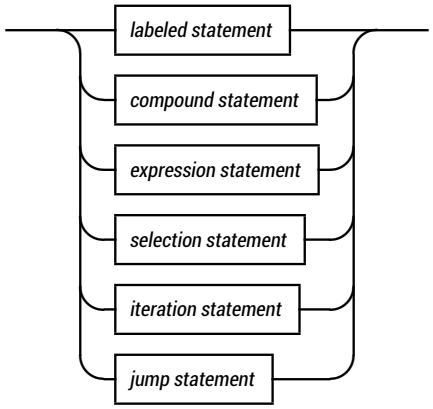
while loop



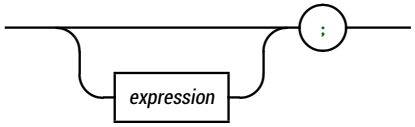
- Da „expression“ und „statement“ Nichterminalsymbole sind, müssen sie (bis zur Vollständigkeit) definiert werden
- Verfolgen am Beispiel Regeln für „statement“

Syntax am Beispiel von Schleifen (Forts.)

statement



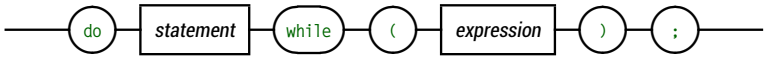
expression statement



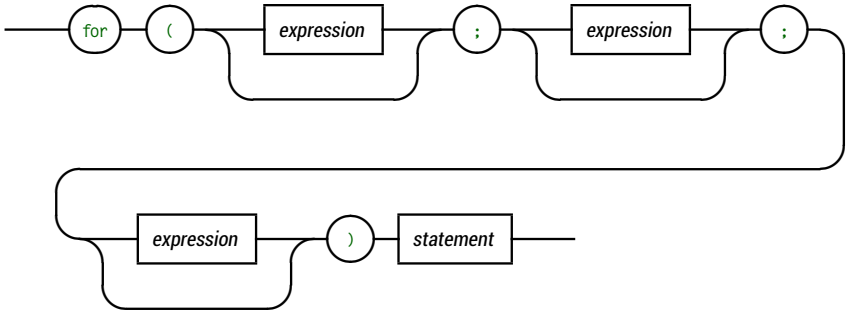
compound statement

Syntax am Beispiel von Schleifen (Forts.)

do while loop



for loop

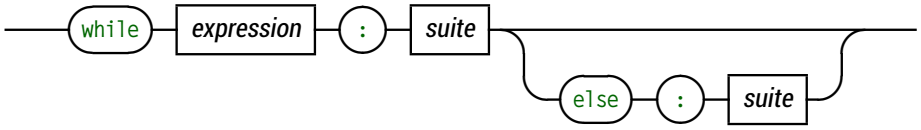


- Das Skript enthält im Anhang B.2 Regeln für den vollständigen Syntax von C in Syntaxdiagrammen

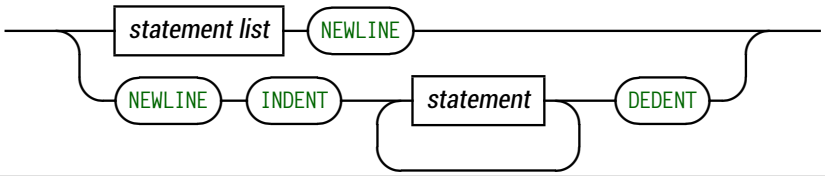
Beispiel Python-Schleifen

- Bei Python-Schleifen sieht man, dass für die lexikalische Analyse mitunter keine regulären Ausdrücke ausreichen
- In Python ist die Einrückung und das Zeilenende lexikalisches Elemente
- Abhilfe: Einführung von Token für Einrückungs**änderung**: „INDENT“ und „DEDENT“

while statement



suite



Formale Semantik

Merke

Während wir die Syntax (auch) formal beschrieben haben, geschah dies bei der Semantik stets informal.

Dies hat pragmatische Gründe: Während formale Syntax (Syntaxdiagramme, reguläre Ausdrücke) breite praktische Anwendungen haben, benötigt man formale Semantiken nur in engen Gebieten wie der (semi-)automatischen Programmverifikation.

Es gibt jedoch verschiedene Ansätze für formale Semantiken, z.B.

- **Denotationelle Semantik**
- **Axiomatische Semantik**
- **Operationelle Semantik**

Aufgaben

Aufgabe 5.1

Fünf Häuser stehen an einer Straße nebeneinander. In ihnen wohnen Menschen von fünf unterschiedlichen Nationalitäten, die fünf unterschiedliche Getränke trinken, fünf unterschiedliche Zigarettenmarken rauchen und fünf unterschiedliche Haustiere haben.

1. Der Brite lebt im roten Haus.
2. Der Schwede hält sich einen Hund.
3. Der Däne trinkt gern Tee.
4. Das grüne Haus steht (direkt) links neben dem weißen Haus.
5. Der Besitzer des grünen Hauses trinkt Kaffee.
6. Die Person, die Pall Mall raucht, hat einen Vogel.
7. Der Mann im mittleren Haus trinkt Milch.
8. Der Bewohner des gelben Hauses raucht Dunhill.
9. Der Norweger lebt im ersten Haus.
10. Der Marlboro-Raucher wohnt neben der Person mit der Katze.
11. Der Mann mit dem Pferd lebt neben der Person, die Dunhill raucht.
12. Der Winfield-Raucher trinkt gern Bier.
13. Der Norweger wohnt neben dem blauen Haus.
14. Der Deutsche raucht Rothmanns.
15. Der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt.

Wem gehört der Fisch?

Aufgaben (Forts.)

Aufgabe 5.2

Beschreiben Sie mit Hilfe von möglichst wenig Regeln in Syntaxdiagrammen oder E(BNF) eine Syntax, in der folgende Ausdrücke zulässig sind:

- „Die schwarze Katze jagt heute die graue Maus.“
- „Die graue Katze sieht die Maus.“
- „Die Maus sieht manchmal die weiße Katze.“

... aber folgende Ausdrücke **nicht** zulässig sind:

- „Die schwarze Katze sieht heute manchmal die graue Katze.“
- „Die Maus jagt morgen die graue Katze.“