Programmiersprachen-Grundlagen

- Syntax (und Semantik)
- Grammatik-Notationen
- Programmiersprachen Übersicht

Das erste M3-Programm

```
MODULE Willkommen EXPORTS Main;
(* Dieses Programm zeigt einen Willkommensgruss
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 16.08.98
   Letzte Aenderung: 20.08.98
*)

IMPORT SIO;
BEGIN
  SIO.PutText("Willkommen zum Studium in Aachen.");
END Willkommen.
```

Syntax und Semantik

Programmiersprachen - Definition

- **Programmiersprachen sind *künstliche Sprachen* (keine natürlichen Sprachen), deren Syntax und Semantik genau festgelegt ist.**
- **Syntax:**
  - Die ***Syntax*** einer Programmiersprache S ist die Definition aller in S ***zulässigen Aussagen***, die in einer Sprache formuliert werden können (Wörter, hier Programme).
- **Semantik:**
  - Die ***Semantik*** einer Sprache S ist die Definition der den zulässigen Aussagen zugeordneten ***Bedeutungen***.
  - Syntaktische ***falsche*** Aussagen haben ***keine*** Semantik
  - Aber auch syntaktisch korrekte Aussagen haben nicht immer eine Semantik (z.B. ein Programm, in dem durch 0 dividiert wird)
  - statische, dynamische Semantik
- **Pragmatik**
  - menschliche, ökonomische

Syntax und Semantik

Beispiel: Syntax, Semantik

- **Natürliche Zahlen**
  - (ohne die Null) dargestellt im Dezimalsystem in arabischen Ziffern bilden eine einfache künstliche Sprache
  - ***Syntax***:
    - jede Zahl ist eine Sequenz von Ziffern (0,1, .. , 9), wobei die erste Ziffer nicht 0 ist
  - ***Semantik***:
    - der Wert einer Zahl ist definiert als der Wert ihrer letzten Ziffer, vermehrt um den zehnfachen Wert der links davon stehenden Zahl, falls diese vorhanden ist (***rekursive*** Definition)
- **Beispiel**
  - syntaktisch ***korrekt*** ist: 367 Semantik: 7+10* (36)
7+10*(6+10*(3))
7+10*(6+10*3)
  - syntaktisch ***falsch*** ist: 007 keine Semantik

Syntax und Semantik

Alphabet

Alphabet

- Ein Alphabet (Zeichenvorrat) ist eine ***nichtleere endliche*** Menge von ***unterscheidbaren*** Zeichen ("Buchstaben", Symbolen)

  A = {a_1, a_2, a_3, …} mit einer ***Ordnungsrelation*** ≤ ($a_1 \le a_2 \le a_3$ ….)

- **Beispiel**:
  - das lateinische Alphabet (a, b, c,.. z)
  - der ASCII-Code (bestehend aus 128 Zeichen)
  - hier Latin-1-Zeichensatz ISO

Wort über einem Alphabet

- ***endliche Folge*** von Buchstaben, die auch ***leer*** sein kann (ε leere Wort)
- A* bezeichnet die ***Menge aller Wörter*** über dem Alphabet A (inkl. dem leeren Wort)

Syntax und Semantik

Formale Sprache

Definition:

- Sei A ein Alphabet. Eine (formale) Sprache (über A) ist ***eine beliebige Teilmenge von A****.

Beispiele:

- A_1 = { 0,1}, A_1* = {ε , 0, 1, 01, 10, 10, 000, 100 , … }
- L = {0, 1, 10, 11, 100, 101, .. } ⊂ A_1*, die Menge der Binärdarstellungen natürlicher Zahlen (mit Null, ohne führende Nullen)

- A_2 = {(,), +, -, *, /, a}, A_2* = {ε, (), (+-a), (a*a), … }
- die Sprache der korrekt geklammerten Ausdrücke EXPR ⊂ A_2* : EXPR = { (((a))), (a+ a), (a -a)*a+ a /(a+ a) -a, .. }

Da solche Sprachen i.d.R. unendlich sind, benötigt man eine endliche Beschreibungsvorschrift

- ***Grammatik***, die die Sprache erzeugt
- ***Automat***, der die Sprache erkennt

Grammatik-Notationen

Grammatik - informell - 1

- **Definiert *Regeln*, die festlegen, welche Wörter über einem *Alphabet* zur Sprache gehören und welche nicht.**
- **Beispiel: Grammatik für "Hund-Katze-Sätze"**

1	<Satz>	->	<Subjekt> <Prädikat> <Objekt>
2	<Subjekt>	->	<Artikel> <Attribut> <Substantiv>
3	<Artikel>	->	ε
4	<Artikel>	->	der
5	<Artikel>	->	die
6	<Artikel>	->	das
7	<Attribut>	->	ε
8	<Attribut>	->	<Adjektiv>
9	<Attribut>	->	<Adjektiv> <Attribut>
10	<Adjektiv>	->	kleine
11	<Adjektiv>	->	bissige
12	<Adjektiv>	->	große
13	<Substantiv>	->	Hund
14	<Substantiv>	->	Katze
15	<Prädikat>	->	jagt
16	<Objekt>	->	<Artikel> <Attribut> <Substantiv>

Grammatik-Notationen

Grammatik - informal - 2

- **Grammatik für "Hund-Katze-Sätze"**
  - durch diese Grammatik können z.B. die folgenden Sätze gebildet (abgeleitet) werden
    - "der kleine bissige Hund jagt die große Katze"
    - "die kleine Katze jagt der bissige Hund"
    - "das große Katze jagt der kleine große bissige kleine …. Katze"
  - folgende "Sätze" werden nicht durch diese Grammatik gebildet
    - "die Katze der Hund"
    - "Katze und Hund"
    - "der Hund jagt die Katze die jagt Hund"

Grammatik-Notationen

Grammatik - Definition - 1

- **Definition:**
  - Eine Grammatik G für eine Sprache L ist definiert durch
  - ein ***Viertupel (N, T, P, S)***
- **N: Menge der *Nichtterminalsymbole***
  - sind Zeichen oder Zeichenfolgen für syntaktische ***Abstraktionen***
  - Beispiel: <Subjekt>, <Objekt>
  - kommen nicht in den Wörtern der Sprache vor
  - werden durch Anwendung der ***Produktionsregeln*** solange ersetzt, bis nur noch Terminalsymbole übrig sind
- **T: Menge der *Terminalsymbole***
  - sind Zeichen oder Zeichenfolgen des Alphabets, aus denen die Wörter der Sprache bestehen
  - Beispiel: kleine , Katze

Grammatik-Notationen

Grammatik - Definition - 2

- **P: Menge von *Produktionsregeln***
  - definieren, wie aus bekannten Konstrukten ***neue Konstrukte*** geschaffen werden
  - Die Anwendung einer Regel bedeutet, daß in der bereits erzeugten Satzform der Teil, der der linken Seite der Regel entspricht, durch die rechte Seite ***ersetzt*** wird
  - **Beispiel**:
    - der kleine bissige Hund <Prädikat> <Objekt>
    - der kleine bissige Hund *jagt* <Objekt>

- **S: das *Startsymbol***
  - ist ein spezielles Nichtterminalsymbol, aus dem ***alle Wörter*** der Sprache mit Hilfe der Grammatik erzeugt werden
  - **Beispiel**: <Satz>
- **Sprache:** Jedes durch Anwendung der Regeln erzeugbare Wort, ***das nur aus Terminalsymbolen besteht***, gehört zu der von der Grammatik erzeugten Sprache L(G)

Grammatik-Notationen

Grammatik - Definition - 3

- **Es gilt:**
  - $N \cap T = \emptyset$
  - $V = N \cup T$ (Gesamtalphabet, Vokabular)
  - sei $p \in P$: $(\alpha \rightarrow \beta)$, $\alpha \in V^* N V^*$, $\beta \in V^*$

Terminale und Nichtterminale sind verschieden

Auf der linken Seite einer Produktion steht wenigstens ein Nichtterminal

- **Ableitung**
  - Ableitungsprozeß ist eine Relation " " auf V^*
  - Für $u,v, \beta \in V^*$ und $\alpha \in V^* N V^*$ gilt
    - $u\alpha v \quad u\beta v$ genau dann, wenn $(\alpha \rightarrow \beta) \in P$
- **Die von einer Grammatik *erzeugte Sprache* ist definiert als:**
  - $L(G) = \{ w \mid w \in T^*, S \overset{*}{} w\}$

w ist herleitbar aus dem Startsymbol

zwei Grammatiken heißen ***äquivalent***, wenn sie dieselbe Sprache erzeugen

Grammatik-Notationen

Typen von Produktionen

- **Je nach Gestalt der in P *zugelassenen Produktionen* definiert Chomsky 4 Typen von Grammatiken**

Produktion	Typ	Eigenschaften	CH-Typ
$(\alpha \rightarrow \beta)$	allgemein	$\alpha, \beta \in V^*$ beliebig	Typ-0
$(\alpha \rightarrow \varepsilon)$	ε -Produktion	$\alpha \in V^*$, $r = \varepsilon$	
$(\alpha \rightarrow \beta)$	beschränkt	$\alpha, \beta \in V^*$, $1 <= \lvert\alpha\rvert <= \lvert\beta\rvert$	Typ-1
$(uAv \rightarrow u\beta v)$	kontextsensitiv	$A \in N$, $u, v, \beta \in V^*$, $\beta \neq \varepsilon$	Typ-1
$(A \rightarrow \beta)$	kontextfrei	$A \in N$, $\beta \in V^*$	Typ-2
$(A \rightarrow Bx)$	linkslinear	$A, B \in N$, $x \in T$	Typ-3
$(A \rightarrow xB)$	rechtslinear		Typ-3
$(A \rightarrow x)$	terminierend	$A \in N$, $x \in T$	

Grammatik-Notationen

Chomsky-Grammatiken

- **Die Chomsky-Grammatiken bilden eine Hierarchie**
  - d.h. die Menge der von Typ-n-Grammatiken erzeugten Sprachen umfaßt die Menge der Sprachen, die von Typ n+1 Grammatiken erzeugt werden
- **Typ-0-Grammatik**
  - Gestalt der Produktionen ist nicht eingeschränkt
  - Alle Sprachen, die überhaupt mit endlichen Regelsystemen erzeugt werden können
- **Typ-1 oder kontextsensitive Grammatik**
  - Produktionen sind beschränkt oder kontextsensitiv
- **Typ-2 oder *kontextfreie* Grammatik**
  - Produktionen sind kontextfrei (d.h. die linke Seite einer Produktion ist immer ein Nichtterminal)
- **Typ-3 oder reguläre Grammatik**
  - Produktionen sind terminierend, links- , rechtslinear

Grammatik-Notationen

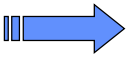
Kontextfreie Grammatik

- **Wichtigste Klasse zur formalen Beschreibung der Syntax von Programmiersprachen.**
- **Es ist möglich, Automaten zu bauen, die Wörter einer kontextfreien Sprache erkennen**
  - ***Wortproblem***
    - Kann für eine kf. Grammatik G und ein Wort w $\in$ T* festgestellt werden, ob w von G erzeugt wird oder nicht.
  - ***Analyseproblem***
    - Gibt es einen Algorithmus, der zu einer kf. Grammatik G und einem Wort w $\in$ T* die syntaktische Struktur von w bestimmt, oder aber feststellt, daß w nicht in L(G) liegt
    - ***Parser*** (Zerteilungsalgorithmus): Syntaxanalyse
- **Notationen zur Darstellung kontextfreier Grammatiken**
  - ***Syntaxdiagramme***
  - ***Extended Backus-Naur-Form* (EBNF)**

Grammatik-Notationen

Grammatik - Beispiel

- **kf. Grammatik, die korrekt geklammerte arithmetische Ausdrücke mit Operatoren *, + erzeugt**
- G1 = ({E,T,F}, {(,), a, +, *}, P, E) mit
  - P = { ❶ E -> T,
    ❷ E -> E + T,
    ❸ T -> F,
    ❹ T -> T * F,
    ❺ F -> a,
    ❻ F -> (E) }
- **a * (a + a) ∈ L(G1)**
  - denn a*(a+a) läßt sich aus E folgendermaßen ableiten
  - dabei wurde immer das am weitesten links stehende Nichtterminal ersetzt (***Linksableitung***)



		Angewandte Regel
E	=> T	❶
	=> T * F	❹
	=> F * F	❸
	=> a * F	❺
	=> a * (E)	❻
	=> a * (E + T)	❷
	=> a * (T + T)	❶
	=> a * (F + T)	❸
	=> a * (a + T)	❺
	=> a * (a + F)	❸
	=> a * (a + a)	❺

Grammatik-Notationen

Diskussion Beispiel - 1

- **Betrachtet man die Erzeugung von arithmetischen Ausdrücken genauer:**
  - Erzeugungsprozeß ist rückwärts betrachtet ein Prozeß der ***sukzessiven Zusammenfassung*** von Teilausdrücken: Reduktion
  - schließlich wird der gesamte Ausdruck auf das Startsymbol ***zurückgeführt***
- **Dabei wird z.B. folgendes beachtet**
  - Vorrang der Klammerstruktur
  - Vorrang der Multiplikation von der Addition
  - Zusammenfassen von links nach rechts bei gleichrangigen Operatoren
- **Beispiel zeigt**
  - daß die Syntax bereits auf grundlegende Eigenschaften der Semantik ***abgestimmt*** sein kann!

Grammatik-Notationen

Diskussion Beispiel - 2

- **Folgende kf. Grammatik erzeugt dieselbe Sprache wie G1**

  G2 = ({E}, {(,), a, +, *}, P, E) mit

  P = { ❶ E -> E + E,
  ❷ E -> E * E,
  ❸ E -> (E),
  ❹ E -> a }

- **Bemerkung**
  - bei G2 fehlt die bei G1 festgestellte Abstimmung der Syntax, d.h. des Erzeugungsprozesses auf die Regeln der Auswertung arithmetischer Ausdrücke
  - G1 kann als Modell für die Definition von Ausdrücken in höheren Programmiersprachen angesehen werden (keine Auswertungs-reihenfolge, Prioritätsfestlegung).

Grammatik-Notationen

Grammatik - Beispiel

- **Sei G = (N, T, P, S) mit**
  - N = {A, B}
  - T = {a, b, c, d}
  - P = { (A -> aBbc),
    (B -> aBb),
    (aBb -> d) }
  - S = A
  - G ist keine kontextfreie Grammatik, da die dritte Produktionsregel auf der linken Seite mehr als nur das Nichtterminalsymbol enthält.

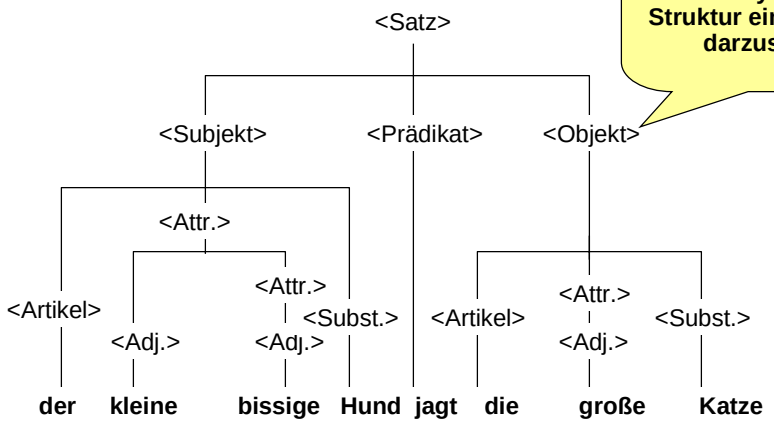
- **Ersetzt man in G die Produktionen P durch P', dann ist G' kontextfrei. Es gilt L(G) = L(G')**
  - P' = { (A -> Bc),
    (B -> aBb),
    (B -> d) }

Grammatik-Notationen

Ableitungsbaum -Strukturbaum

- **Vaterknoten = linke Seite einer Regel**
- **Söhne = rechte Seite einer Regel**

Mittel, um die syntaktische Struktur eines Wortes darzustellen



Grammatik-Notationen

Syntaxdarstellung - EBNF - 1

- **EBNF**
  - **E**xtended **B**ackus-**N**aur-**F**orm
  - ***Meta-Sprache*** zur Beschreibung der Syntax formaler Sprachen
  - BNF erstmals benutzt zur Definition der Sprache ***Algol-60***
  - ***Metasymbole*** von EBNF sind
    - = „definiert als“
    - (...|...) genau eine Alternative aus der Klammer muß stehen
    - [...] Inhalt der Klammer kann stehen oder nicht
    - { ... } Inhalt der Klammer kann n-fach stehen, n >= 0
    - . Ende der Produktion
    - Terminalsymbole werden in " " eingeschlossen

Grammatik-Notationen

Syntaxdarstellung - EBNF - 2

- **Unsere einfache Grammatik für "Hund-Katze-Sätze" sieht in EBNF folgendermaßen aus:**

Satz	=	Subjekt Prädikat Objekt.
Subjekt	=	Artikel Attribut Substantiv.
Artikel	=	[("**der**" | "**die**" | "**das**")].
Attribut	=	[(Adjektiv | Adjektiv Attribut)].
Adjektiv	=	("**kleine**" | "**bissige**" | "**große**").
Substantiv	=	("**Hund**" | "**Katze**").
Prädikat	=	"**jagt**".
Objekt	=	Artikel Attribut Substantiv.

Grammatik-Notationen

Syntaxdarstellung - EBNF - 2

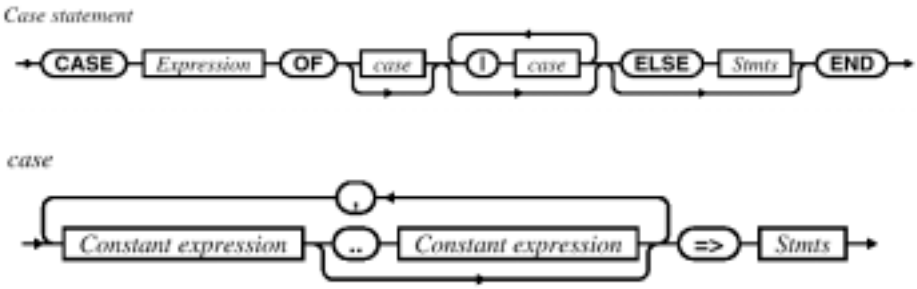
CaseStatement = „**CASE**“ Expression „**OF**“
[Case] { „**|**“ Case }
[„**ELSE**“ Stmts] „**END**“ .

```
CASE  operator OF
  '+'  => resultat := a + b;
 |'-'  => resultat := a - b;
 |'*'  => resultat := a * b;
 |'/'  => resultat := a / b;
 ELSE (* Fehler *)
END;
```

Grammatik-Notationen

Syntaxdiagramme - Beispiel

- **Syntaxdiagramme**
  - beschreiben Produktionen ***grafisch***
  - Nichtterminalsymbole sind Rechtecke
  - Terminalsymbole sind Langrunde



Programmier-sprachen: Allgemeines

Syntaxebenen

- **lexikalische Syntax**
- **kontextfreie Syntax**

} **Vgl. Modula-Syntax**

- **kontextsensitive Syntax (umgangssprachlich)**

```
CASE  operator OF
  '+'  => resultat := a + b;
 |'-'  => resultat := a - b;
 |'*'  => resultat := a * b;
 |'/'  => resultat := a / b;
 ELSE (* Fehler *)
END;
```

Programmier-sprachen: Allgemeines

Lexikalische Einheiten

- Bezeichner
- Begrenzer
- Wortsymbole
- Literale
- Kommentare

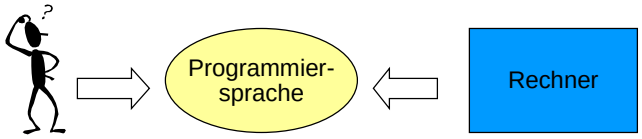
Eventl. Trennzeichen zwischen lexikalischen Einheiten

```
CASE  operator OF
  '+'  => resultat := a + b;
 |'-'  => resultat := a - b;
 |'*'  => resultat := a * b;
 |'/'  => resultat := a / b;
 ELSE (* Fehler *)
END;
```


Programmier-sprachen: Allgemeines

Programmiersprachen

- **Die Programmiersprache bildet die *Schnittstelle* zwischen Mensch und Rechner**



Beide haben unterschiedliche Anforderungen

- **Mensch**
  - Erlernbarkeit
  - Lesbarkeit
  - Ausdrucksstärke

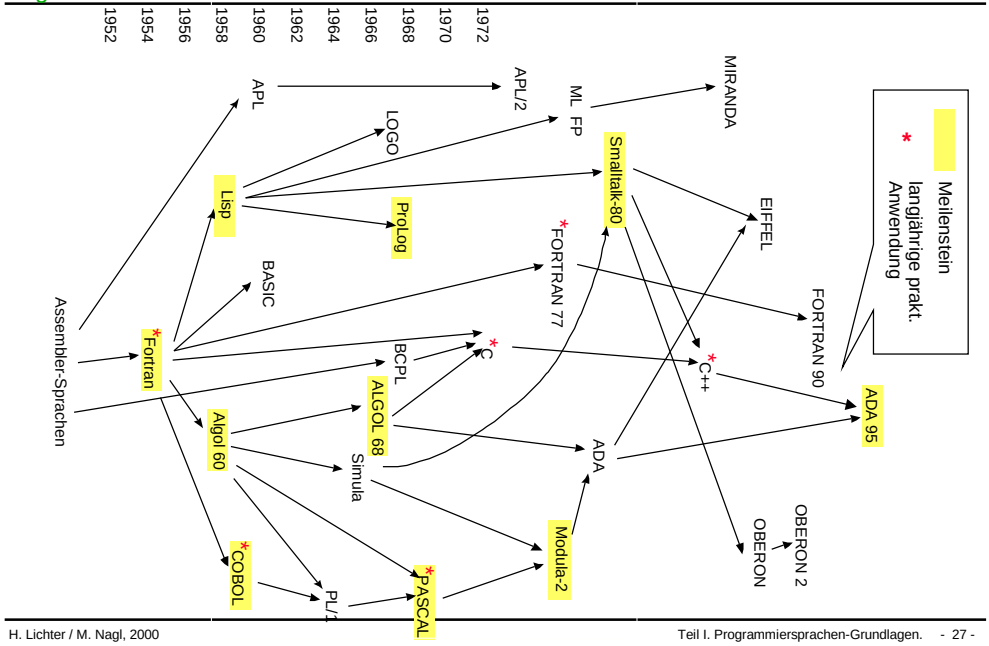
Problem-orientierte Sprachen

- **Rechner**
  - einfaches Übersetzen in Maschinensprache
  - effizienter Code soll generiert werden können

Maschinen-sprachen

Programmier-sprachen: Allgemeines

Entwicklung



Programmier-sprachen: Allgemeines

Werkstoff

- **Programmiersprachen sind der *Werkstoff* des Informatikers**
- **Das, was wir erzeugen (Programme)**
  - ist ***immateriell***
  - wir bauen es aus dem Werkstoff "Programmiersprache"
- **Ein Informatiker**
  - sollte die ***Qualität*** und ***Eignung*** verschiedener Werkstoffe (Programmiersprachen) kennen
  - Welche Sprache ist wofür geeignet?
- **Analogie!**
  - Ein Bauingenieur verwendet andere Werkstoffe, wenn ein
    - Hochhaus (Stahl, Beton etc.)
    - oder ein Einfamilienhaus (Ziegelsteine, Holz, Beton, etc.)
  - gebaut werden soll

Was haben wir gelernt

- Wie wissen was Programmiersprachen sind
- Wir kennen den Begriff "Formale Sprache"
- Wir haben gesehen, daß eine Grammatik eine Sprache erzeugt
- Wir kennen EBNF und Syntaxdiagramme
- Wir sehen Programmiersprachen als Werkstoff des Informatikers

Glossar

- Programmiersprache
- Syntax, Syntaxebenen
- Semantik
- Alphabet
- Formale Sprache
- Grammatik
  - kontextfreie Grammatik
- Syntaxdarstellung: EBNF, Syntaxdiagramm
- Ableitungsbaum, Syntaxbaum
- lexikalische Einheiten