Grundlagen der Informatik

Syntax, Semantik und formale Beschreibung von Programmiersprachen

Prof. Dr. Peter Jüttner

Syntaxbeschreibungen

- **Programmiersprachen sind *formale Sprachen*, d.h.:**
  - Syntax und
  - Semantik

  sind bis ins Detail definiert und müssen auch exakt verwendet werden.

  Im Gegensatz dazu ist die Syntax einer natürlichen Sprache (meist) ebenfalls definiert, wird aber selten ganz exakt verwendet.
  Die Semantik natürlicher Sprachen ist kontextabhängig

Syntaxbeschreibungen

- **Syntax**
  Struktureller Aufbau einer Programmiersprache, legt fest wie eine Programm formuliert werden darf bzw. muss
  (in der natürlichen Sprache, wie ein Wort oder Satz aufgebaut ist)

- **Semantik**
  Definiert die Bedeutung syntaktisch korrekter Programme
  (nicht korrekte Programme haben keine Bedeutung)

Syntaxbeschreibungen

- **Folgerung: Syntax muss eindeutig beschrieben werden.**

- zunächst: Definition der Struktur einer Programmiersprache ist hierarchisch aufgebaut, d.h. entweder
  - ein Syntaxelement wird durch andere (oder sich selbst) erklärt

  oder
  - ein Syntaxelement ist als festes Zeichen oder feste Zeichenfolge definiert

Syntaxbeschreibungen

- **Folgerung: Syntax muss eindeutig beschrieben werden.**

- Beispiele:
  - if-Anweisung besteht aus

    if (Bedingung) Anweisungsfolge

    oder aus

    if (Bedingung) Anweisungsfolge else Anweisungsfolge

Syntaxbeschreibungen

- **2 Arten der Syntaxdarstellung**
  - Backus-Naur-Form (BNF) (rein textuell)
  - Syntaxdiagramme (grafisch)

  Beide Darstellungsformen sind äquivalent, BNF ist maschinenlesbar (Compilerbau), Syntaxdiagramme sind schneller verständlich

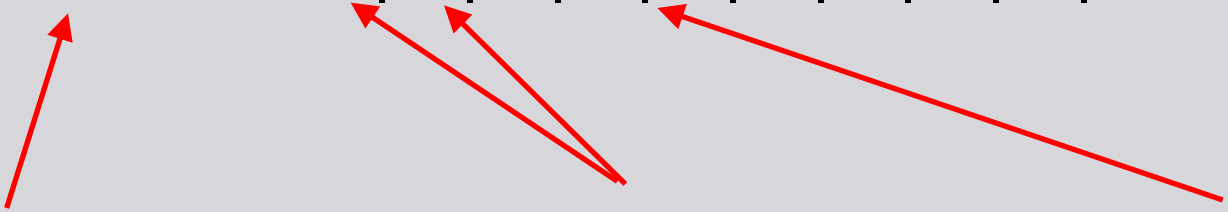
Syntaxbeschreibungen

- **Backus-Naur-Form**
  - nach John Backus und Peter Naur (~ 1960)
  - erste exakte, formale Darstellung der Syntax einer Programmiersprache (Algol 60)
  - BNF besitzt selbst eine Syntax (Metasyntax)
  - Ergänzung zur Erweiterten BNF ➔EBNF (N. Wirth)

Syntaxbeschreibungen

- **Backus-Naur-Form**
  - verwendet Terminalsymbole (terminals), i.e. Symbole, die im Programm in der angegebenen Form verwendet werden
  - verwendet Ableitungsregeln (Produktionen) für Nichtterminalsymbole (nonterminals), z.B.

    <Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9



Nichtterminalsymbol in <...>-Klammern

Terminalsymbole ohne <...>-Klammern

Alternative |

Syntaxbeschreibungen

- **Backus-Naur-Form**
  - <Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

    umgangssprachlich: Eine Ziffer ist entweder eine 0 oder 1 oder eine 2 oder eine 3 oder ... oder eine 9

  - kommt ein Nichtterminalsymbol auf der rechten Seite einer Produktion vor, muss es eine Produktion für dieses Symbol geben
    <A> ::= <B> | C
    <B> ::= ...

Syntaxbeschreibungen

- **Programmiersprachen sind *formale Sprachen***
- Eine Formale Sprachen wird gebildet über einem ***Alphabet***
- Eine ***Alphabet*** A ist eine (nichtleere) Menge von ***Zeichen*** (auch als ***Buchstaben*** oder ***Symbole*** bezeichnet)
- Ein ***Wort*** über dem Alphabet A ist eine endliche (ggf.auch leere) Folge von Zeichen aus A
- Die Menge aller Worte über A wird mit A^* bezeichnet

Syntaxbeschreibungen

- Eine ***(formale) Sprache L*** über dem Alphabet A ist eine Teilmenge von A^*

- Beispiel:

  Alphabet = {0, 1}

  A^* = Menge aller Binärzahlen =

  {0, 1, 00, 01, 000, 001, 010, … }

Syntaxbeschreibungen

- weitere Alphabete:

  Alphabet = {‘a‘, ‘b‘, ‘c‘, … ‘z‘, ‘A‘, ‘B‘, ‘C‘, … ‘Z‘, ‘ä‘, ‘ü‘, ‘ö‘, ‘Ä‘, ‘Ü‘, ‘Ö‘, ‘ß‘, ‘ ‘, ‘,‘, ‘.‘, ‘:‘, … }

  A^* = Menge aller über den deutschen Buchstaben und Satzzeichen erzeugbaren Texte (auch sinnlose!)

Syntaxbeschreibungen

- Eine ***Grammatik*** beschreibt, wie Worte einer formalen Sprache gebildet werden können bzw. welche Worte zu der Sprache gehören. Dies geschieht durch ***Regeln***.

  Die Menge der Worte ist eine Teilmenge von A*

Syntaxbeschreibungen

- Beispiel für Grammatikregeln: „Affe-Banane-Sätze“

1. <Satz> ::= <Subjekt> ‘ ‘ <Prädikat> ‘ ‘ <Objekt>
2. <Subjekt> ::= <Artikel_1> ‘ ‘ <Attribut_1> ‘ ‘ <Substantiv_1>
3. <Artikel_1> ::= der
4. <Attribut_1> ::= faule | alte | kluge | dumme
5. <Substantiv_1> ::= Affe
6. <Prädikat> ::= frisst | sucht | findet
7. <Objekt> ::= <Artikel_2> ‘ ‘ <Attribut_2> ‘ ‘ <Subjekt_2>
8. <Artikel_2> ::= eine | keine
9. <Attribut_2> ::= schöne | große | reife | gelbe | grüne | unreife
10. <Subjekt_2> ::= Banane

Alphabet z.B. = { der, faule, alte, kluge, dumme, Affe, frisst, sucht, findet, eine, keine, schöne, große, reife, gelbe, grüne, unreife, Banane, ‘ ‘ }

Syntaxbeschreibungen

- Beispiel für Grammatikregeln: „Affe-Banane-Sätze“

  folgende „Worte“ (hier eigentlich Sätze) können damit u.a. gebildet werden:

  - der alte Affe frisst eine gelbe Banane
  - der kluge Affe frisst keine unreife Banane
  - der dumme Affe findet keine reife Banane

Syntaxbeschreibungen

Eine ***Grammatik*** ist formal definiert durch ein Viertupel (N,T,P,S) mit

- N einer endlichen Menge von ***Nichtterminalsymbolen*** (Variablen, Symbole, die weiter erklärt werden). Diese sind
  - Symbole für syntaktische Abstraktionen, z. B. Satz, Subjekt, Prädikat, Objekt, . . .
  - Sind keine Wörter der Sprache
  - werden durch Anwendung der Grammatikregeln („Produktionen“ (s. u.) so lange ersetzt, bis nur noch Terminalsymbole übrig sind.

Syntaxbeschreibungen

Eine ***Grammatik*** ist formal definiert durch ein Viertupel (N,T,P,S) mit

- T einer endliche Menge von ***Terminalsymbolen*** (Symbole, die nicht weiter erklärt werden, wobei $T \cap N = \varnothing$, d.h. es gibt keine Symbole, die gleichzeitig terminal und nichtterminal sind.
  Die Elemente von T sind die Zeichen des Alphabets, aus denen die Wörter der Sprache bestehen.

  Die Vereinigungsmenge V von N und T ($N \cup T$) wird als ***Vokabular*** der Grammatik bezeichnet

Syntaxbeschreibungen

Eine ***Grammatik*** ist formal definiert durch ein Viertupel (N,T,P,S) mit

- P eine endliche Menge von ***Produktionsregeln*** **x ::= y**, wobei Regel x ::= y bedeutet, dass das Teilwort x durch das Teilwort y ersetzt werden kann. Zu beachten ist noch, dass das Teilwort x mindestens ein Nichtterminalsymbol enthalten muss, während y Terminal- oder Nichtterminalsymbole enthalten kann.

  Statt x ::= y wird auch x → y geschrieben

Syntaxbeschreibungen

Eine ***Grammatik*** ist formal definiert durch ein Viertupel (N,T,P,S) mit

- S das ***Startsymbol****, ein* spezielles Nichtterminalsymbol (S $\in$ N), aus dem alle Wörter der Sprache mit Hilfe der Produktionsregeln erzeugt werden.

  Im Affen-Bananenbeispiel ist S der Satz.

Syntaxbeschreibungen

Definition:
Ein Wort w heißt ***(direkt) ableitbar*** aus dem Wort z, (Schreibweise $z \Rightarrow w$), wenn es Worte $u,v \in V^*$ und eine Regel $(x::=y) \in P$ gibt mit z = uxv und w = uyv, d. h. w entsteht aus z, indem man eine Regel auf ein Teilwort x von z anwendet.

Beispiel:
<Subjekt> ‘ ‘ <Prädikat> ‘ ‘ <Objekt> $\Rightarrow$
<Subjekt> ‘ ‘ frisst ‘ ‘ <Objekt>

Syntaxbeschreibungen

Die von der Grammatik ***erzeugte Sprache*** ist definiert als

$$L(G) := \{w \in T^* \mid S \Rightarrow . . . \Rightarrow w\}$$

d.h. die Menge aller aus dem Startsymbol ableitbaren Wörter, bestehend nur aus Terminalsymbolen.

Syntaxbeschreibungen

Zwei Grammatiken G_1 und G_2 heißen ***äquivalent***, wenn sie dieselbe Sprache erzeugen, d.h.

$L(G_1) = L(G_2)$

Eine Grammatik heißt ***kontextfrei***, wenn jede Produktion die Form $X ::= y$ besitzt mit $X \in N$, $y \in V^*$, d. h. auf der linken Seite jeder Produktion steht genau ein nichtterminales Symbol (und sonst nichts).

Natürliche Sprachen sind nicht kontextfrei im Gegensatz zu Programmiersprachen.

Syntaxbeschreibungen

Beispiel einer nicht-kontextfreien Grammatik

G = (N,T,P,S) mit

N = {<A>,<B>} (Nichtterminale Symbole)

T = {a,b,c,d} (Terminale Symbole)

P = {<A> ::=a<B>cd, <B> ::=a<B>c, a<B>c ::=b}
(Produktionen)

S = <A> (Startsymbol)

wie schauen die Wörter von L(G) aus ?

Syntaxbeschreibungen

Beispiel einer nicht-kontextfreien Grammatik

Dann ist $L(G) = \{a^n b c^n d \mid n \geq 0\}$,

d.h. die Menge aller Wörter, die mit einer beliebigen Anzahl n von „a“s beginnen, gefolgt von genau einem „b“, dann gleich vielen „c“s wie „a“s zu Beginn, und schließlich genau einem „d“.

Syntaxbeschreibungen

Zum Schluss dieses Abschnitts …

Noch Fragen ??

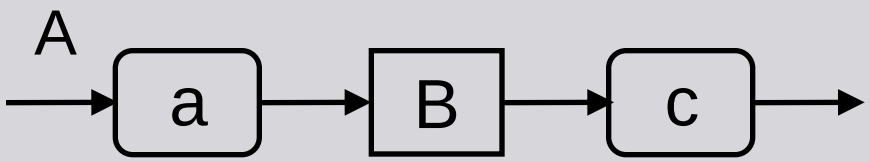
Syntaxbeschreibungen

Alternative Beschreibung von Programmiersprachen – Syntaxdiagramme:

Produktion in der Grammatik

<A> ::= a<B>c

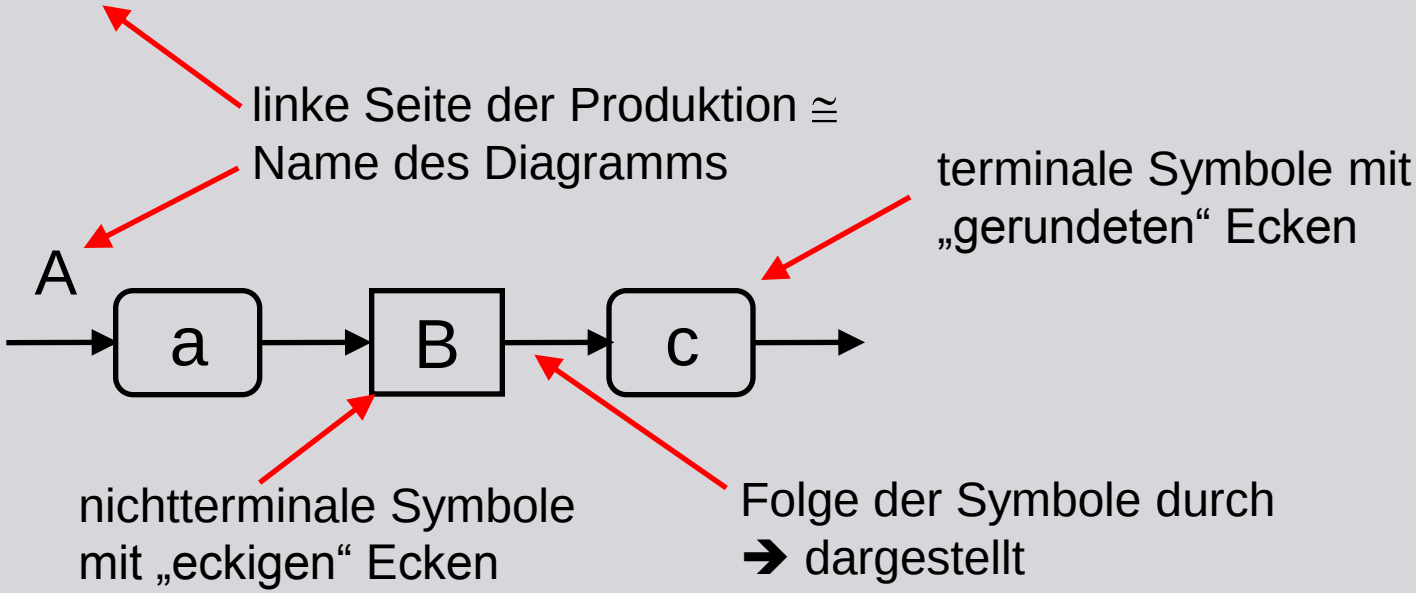
äquivalentes Syntaxdiagramm



Syntaxbeschreibungen

Alternative Beschreibung von Programmiersprachen – Syntaxdiagramme:

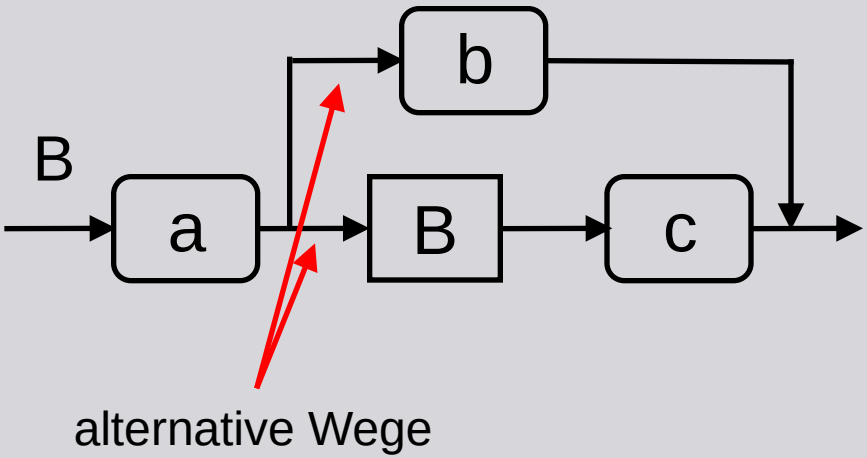
<A> ::= a<B>c



Syntaxbeschreibungen

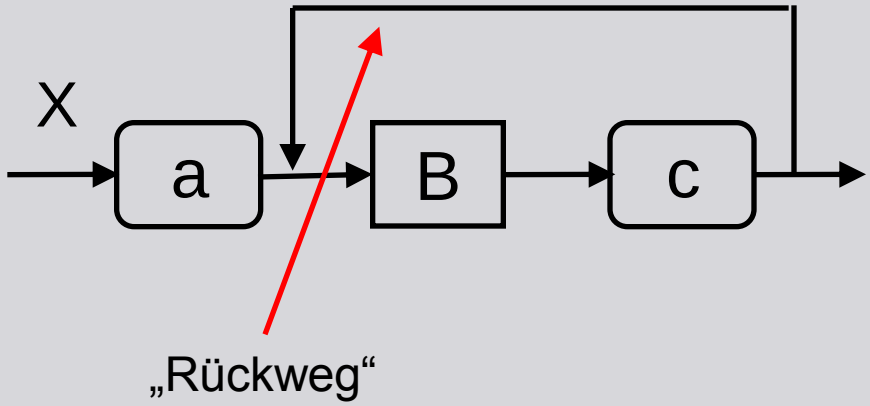
Alternative Beschreibung von Programmiersprachen – Syntaxdiagramme - Darstellung von Alternativen:

<B> ::= a<B>c | ab



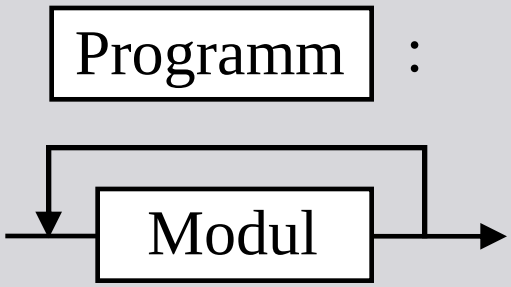
Syntaxbeschreibungen

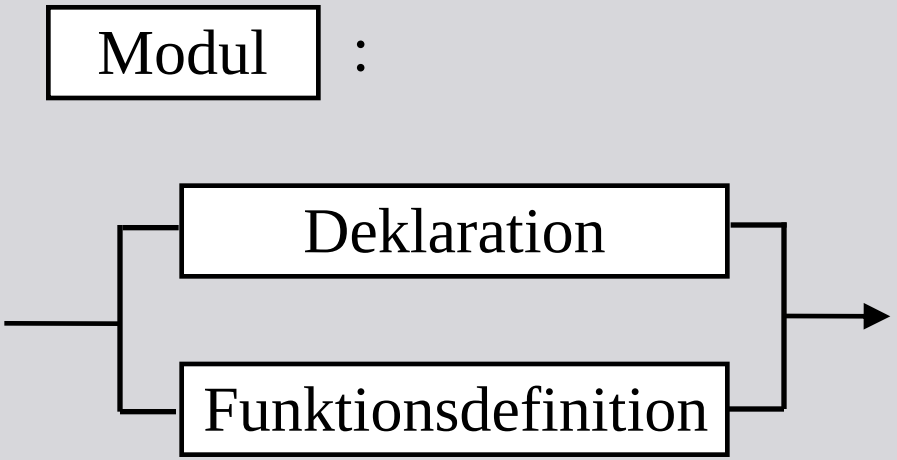
Alternative Beschreibung von Programmiersprachen – Syntaxdiagramme - Wiederholungen:



Syntaxbeschreibungen

Syntaxdiagramme der Sprache C - Beispiel:





Syntaxbeschreibungen

Zum Schluss dieses Abschnitts …

Noch Fragen ??