TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

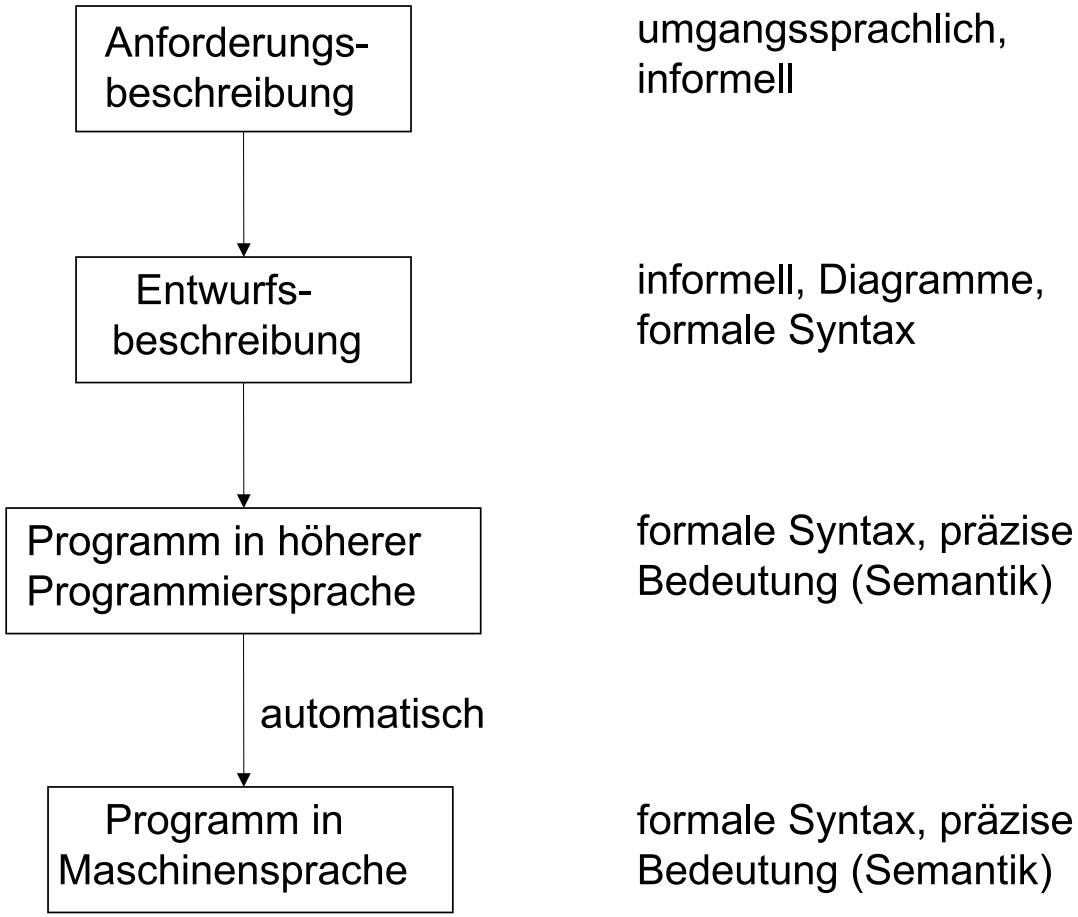
Annette Bieniusa / Arnd Poetzsch-Heffter

AG Softech
FB Informatik
TU Kaiserslautern

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Softwaresysteme und Programmierung

Beschreibungsebenen in der SE I



Beschreibungsebenen in der SE II

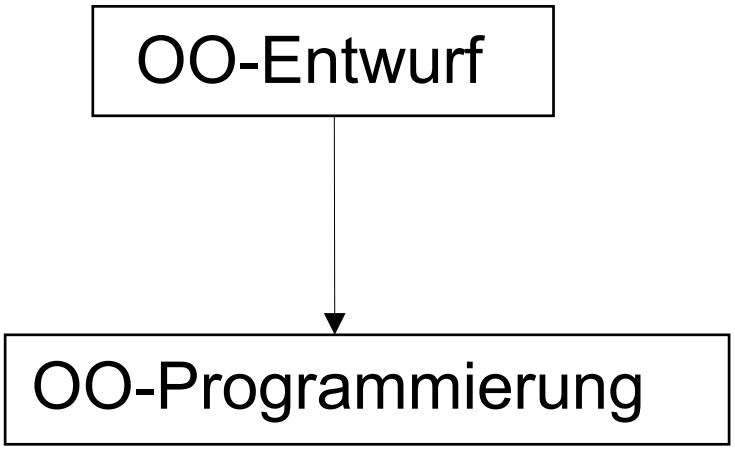
Das Verhalten von Softwaresystemen wird durch Programme beschrieben.

Die Programmiersprache ist das Medium, mit dem der Mensch der Maschine beschreibt, was sie tun soll.

Letztendlich muss der Entwurf eines SW-Systems also so weit verfeinert werden, dass er mit den Mitteln einer Programmiersprache ausgedrückt werden kann.

Beziehung: Entwurf/Programmierung

- ähnliche Modelle
- ähnliche Konzepte



OO-Modellierungssprachen
(UML, VDM++, Object-Z)

OO-Programmiersprachen
(SmallTalk, C++, Java, C#)

Vorgehensweise in der Vorlesung

Entwurfskonzepte und -modelle ⇒ SE2 und SE3
Konzepte und Modelle der Programmierung ⇒ SE1 – also hier!
Ziel ist insbesondere das Erlernen der folgenden Konzepte/Begriffe:

- Werte und Typen
- Ausdruck
- Anweisung
- Funktion und Prozedur
- Programmvariable
- Rekursion
- Objekt, Klasse
- Spezialisierung und Abstraktion
- Kapselung und Information Hiding (Geheimnisprinzip)
- Programmschnittstelle

Diese Konzepte sind *programmiersprachenunabhängig*.
Wir erläutern sie anhand ihrer Ausprägungen in den Sprachen Java.

Programmbeispiel: Haskell

```
qsort1 :: Ord a => [a] -> [a]
qsort1 []     = []
qsort1 (p:xs) = qsort1 lesser ++ [p] ++ qsort1 greater
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

Programmbeispiel: Python

```
def qs(l):
    if l==[]: return []
    m = l[0]
    return qs([x for x in l if x<m]) + \
           [x for x in l if x==m] + \
           qs([x for x in l if x>m])
```

Programmbeispiel: Java

```
void quicksort (int[] a, int lo, int hi) {
    int i=lo, j=hi, h;
    int x=a[(lo+hi)/2];
    do {
        while (a[i]<x) i++;
        while (a[j]>x) j--;
        if (i<=j)
        {
            h=a[i]; a[i]=a[j]; a[j]=h;
            i++; j--;
        }
    } while (i<=j);
    if (lo<j) quicksort(a, lo, j);
    if (i<hi) quicksort(a, i, hi);
}
```

Programmbeispiel: FORTRAN

```
C USING HERONS FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +      (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

Wie kann man Programmiersprachen (im Vergleich zu natürlichen Sprachen) charakterisieren?
Welche Elemente / Strukturen können Sie erkennen?

Programmiersprachen

- strukturiert; recht gut lesbar
- Modellierungs- und Beschreibungsebene abstrakter als bei Assembler- oder Maschinensprachen
- rechner-/plattformunabhängig ($\rightarrow$ portabel)
- nicht direkt ausführbar vom Rechner

Beispiel einer Maschinensprache, eines Assemblers

```
*C600L

C600-   A2 20     LDX   #$20
C602-   A0 00     LDY   #$00
C604-   A2 03     LDX   #$03
C606-   86 3C     STX   $3C
C608-   8A        TXA
C609-   0A        ASL
C60A-   24 3C     BIT   $3C
C60C-   F0 10     BEQ   $C61E
C60E-   05 3C     ORA   $3C
C610-   49 FF     EOR   #FF
C612-   29 7E     AND   #7E
C614-   B0 08     BCS   $C61E
```

Maschinensprache, Assembler

- Maschinenprogramme sind Byte- bzw. Zahlenfolgen
- Sprachelemente entsprechen den Befehlen und Speichermechanismen des zugehörigen Rechners
- rechnerspezifisch, nicht portabel
- direkt ausführbar vom Rechner

Zwischen Programmiersprachen und Maschinensprache gibt es oft weitere Sprachebenen:

- Assembler
- Abstrakte Maschinensprachen, Bytecode

Sprachliche Grundlagen

Beschreibungstechniken spielen eine besondere Rolle in der Informatik.

Sie sind

- Hilfsmittel und
- *Untersuchungsgegenstand*

der Informatik.

Im Mittelpunkt des Interesses stehen formale Sprachen
(vgl. Modul "Formale Grundlagen der Programmierung").

Einführung in formale Sprachen I

Ein **Zeichen** oder **Symbol** ist ein abstraktes Element, das eine *syntaktisch unmissverständliche Repräsentation* besitzt.

Typische Repräsentationen von Zeichen/Symbolen sind Ziffern, Buchstaben, das Euro-Zeichen, Klammern, Komma, Punkt, Plus-, Minus-Zeichen, Leerzeichen, Tabulatorzeichen, Zeilenumbruchzeichen, ...

Die Repräsentation kann aber auch zusammengesetzt sein (z.B. `>=` für größer gleich).

Einführung in formale Sprachen II

Eine **Zeichenreihe** (**Wort**, **String**, **Text**) ist eine endliche Folge/Sequenz von Zeichen/Symbolen.

Die syntaktische Repräsentation einer Zeichenreihe ergibt sich durch Aneinanderreihen ihrer Zeichen.

- ohne Zwischenraum
- oft eingeschlossen in hochgestellte Anführungszeichen

Die leere Zeichenreihe wird als ε oder “” notiert.

Einführung in formale Sprachen III

Ein **Alphabet** A ist ein endliche Menge von Zeichen.

Die *Menge der Zeichenreihen über* A (mit Zeichen aus A) wird mit A^* bezeichnet.

Eine **formale Sprache** *über* A ist eine Menge von Zeichenreihen über A, also eine Teilmenge von A^*.

Bemerkungen

- Teilweise betrachtet man auch formale Sprachen mit unendlichen Zeichenfolgen.
- Die *Spezifikation* einer formalen Sprache S sollte es ermöglichen zu prüfen, ob eine Zeichenreihe w Element von S ist.
- Elemente formaler Sprachen haben zunächst einmal keine Bedeutung.

Beispiele: Alphabete

- $\{ a, b, c, d \}$
- $\{ 0, 1, 2, 3, 4, 5, 6, 7, 8, +, ^*, (,), \pi, = \}$
- $\{ \boxed{0}, \boxed{1}, \boxed{2}, \ldots, \boxed{126}, \boxed{127} \}$
- 8-Bit ASCII-Zeichensatz bestehend aus
  - Steuerzeichen
  - Groß- und Kleinbuchstaben
  - Ziffern
  - Satzzeichen, Klammern, arithm. Operatoren
  - Leerzeichen, Prozentzeichen, ...
- Erweiterte Zeichensätze
- $\{$ if, $\langle$ Ausdruck $\rangle$, then, else, end $\}$, wobei die Symbole aus anderen Zeichen zusammengesetzt sind

Beispiele: Formale Sprachen

- { “aa“, “abba“, “babbab“, “c“}
- die Menge aller Zeichenreihen aus großen Buchstaben, die ein **Palindrom** bilden; z.B.:
  “RELIEFPFEILER “
- die Menge aller Zeichenreichen bestehend aus Ziffern, die eine Primzahl repräsentieren; z.B.:
  “3“, “007“, “255“, “23245678899444333343“
- die Menge der Java-Programme (Alphabet?)

Spezifikation formaler Sprachen

Endliche formale Sprachen kann man durch Aufzählung/Aufschreiben *spezifizieren*.

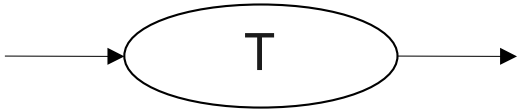
Zur Spezifikation unendlicher formaler Sprachen betrachten wir im Folgenden

- Syntaxdiagramme
- kontextfreie Grammatiken

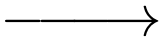
Syntaxdiagramme

Ein **Syntaxdiagramm** ist ein Flussgraph bestehend aus **Nichtterminalknoten**:

Terminalknoten:



Extrakanten:



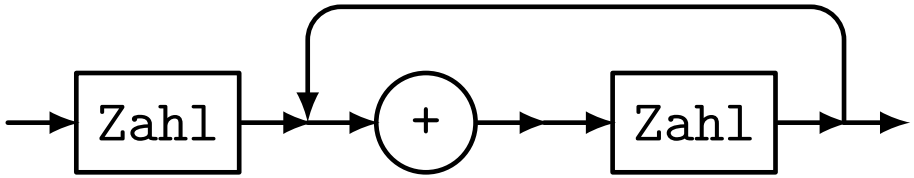
mit definiertem Eingang und Ausgängen.

Die Knoten sind mit *Symbolen* markiert.

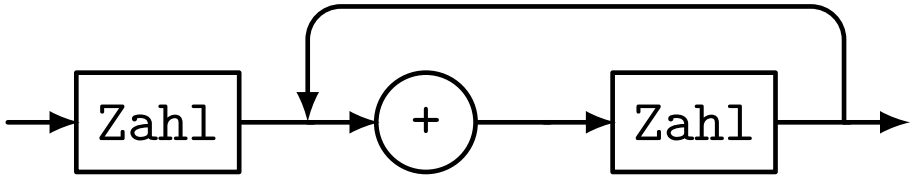
Jedes Syntaxdiagramm hat ein Symbol als **Namen**.

Beispiele: Syntaxdiagramme I

Summe:

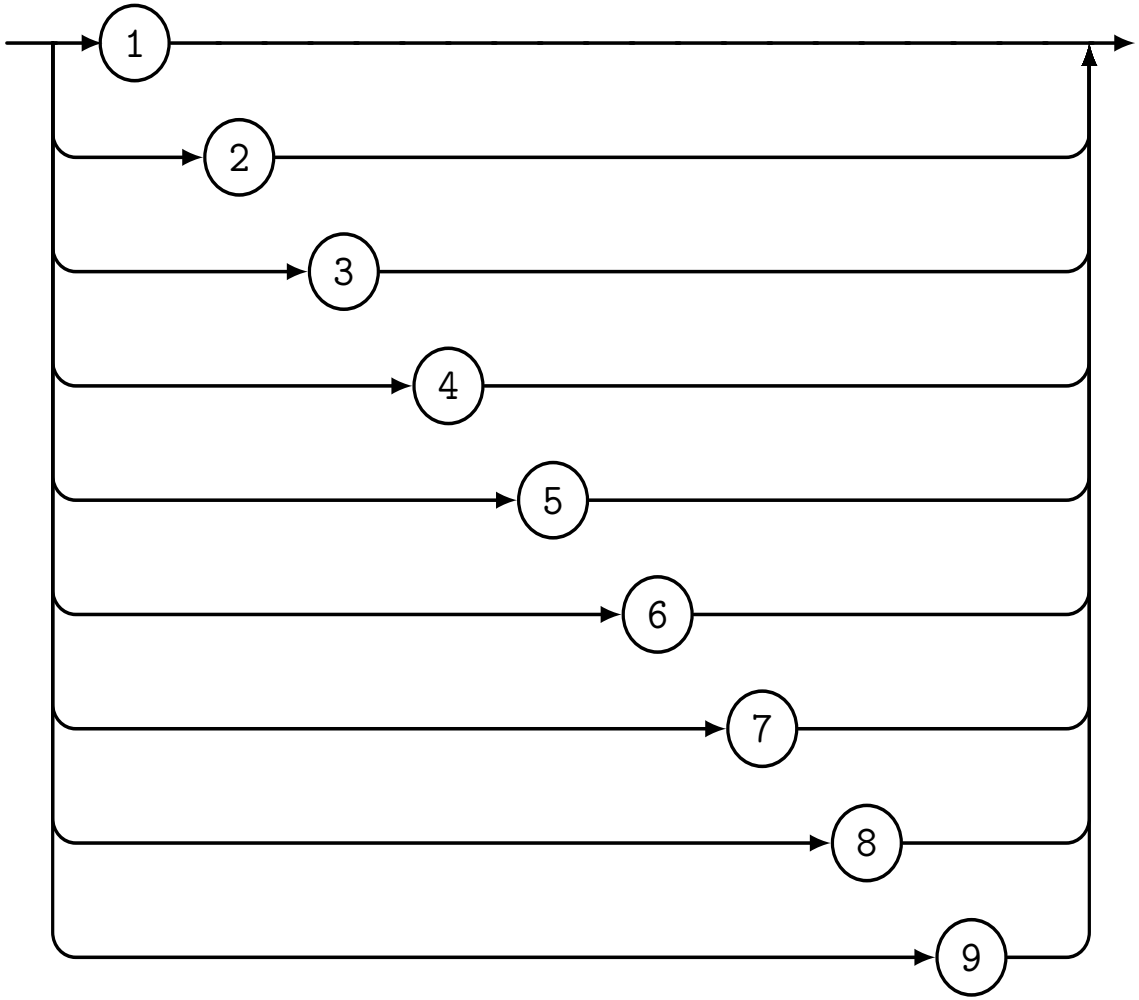


Die Kantenführung wird meist freier gehandhabt:



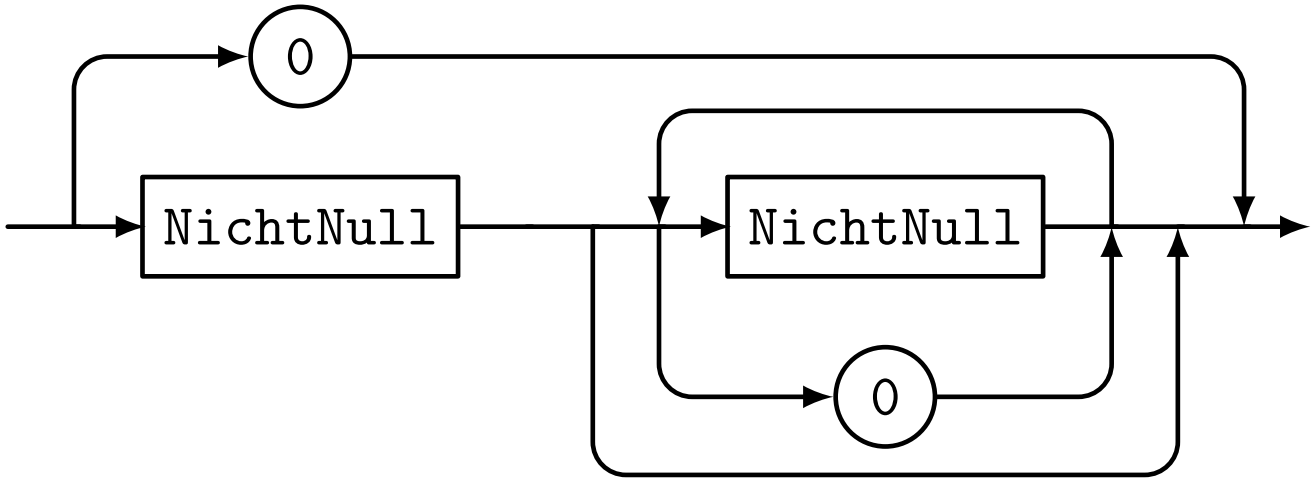
Beispiele: Syntaxdiagramme II

NichtNull:



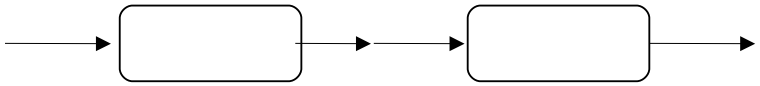
Beispiele: Syntaxdiagramme III

Zahl:

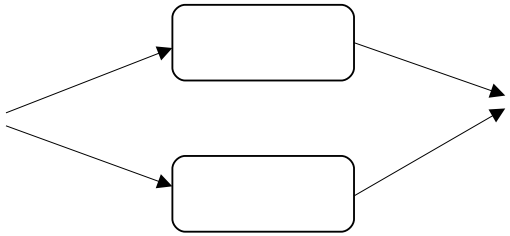


Grundtypen von Syntaxdiagrammen

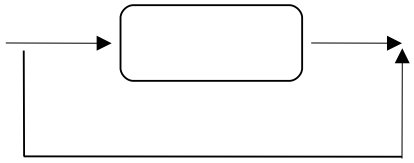
Sequenz:



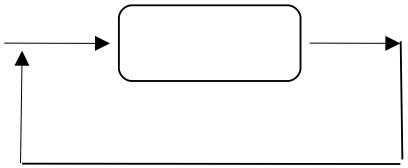
Alternative:



Option:



Wiederholung:



wobei  ein Nichtterminal- oder Terminalknoten ist.

Definition: Sprachdefinition mit Syntaxdiagrammen

Seien T und N zwei disjunkte Alphabete.
Die Elemente von T nennen wir **Terminalsymbole**, die von N **Nichtterminalsymbole**.

Sei Δ eine endliche Menge von Syntaxdiagrammen mit Namen aus N, in denen Terminalknoten mit Terminalsymbolen und Nichtterminalknoten mit Nichtterminalsymbolen markiert sind.

Sei $S \in N$; S heißt **Startsymbol** oder **Axiom**.

Dann heißt $\Gamma = (N, T, \Delta, S)$ eine **Sprachdefinition mit Syntaxdiagrammen**, kurz SDmSD.

Begriffsklärung: durch SDmSD definierte Sprache

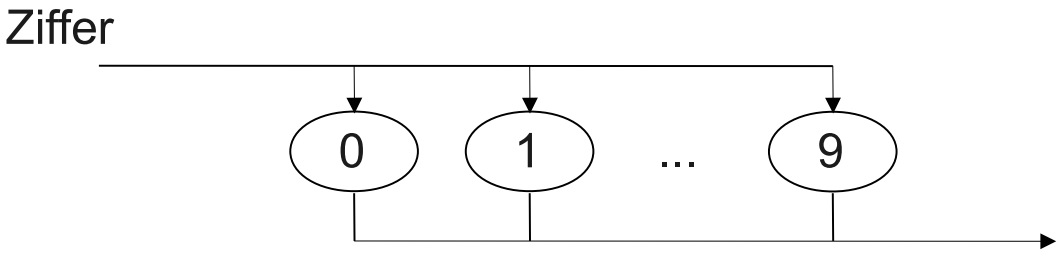
Die von einer **Sprachdefinition mit Syntaxdiagrammen definierte Sprache** ist die Menge der Zeichenreihen über T, die man durch "Ablaufen" der Syntaxdiagramme ausgehend vom Syntaxdiagramm mit Namen S erhält.

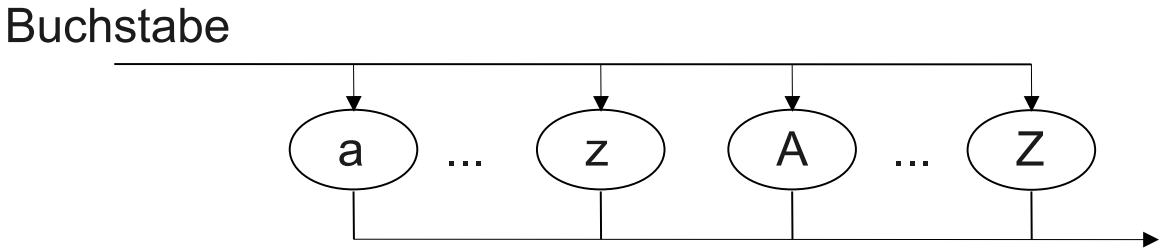
Beispiel: SDmSD I

$T = \{a, b, ..., z, A, ..., Z, 0, 1, ..., 9, \_\}$

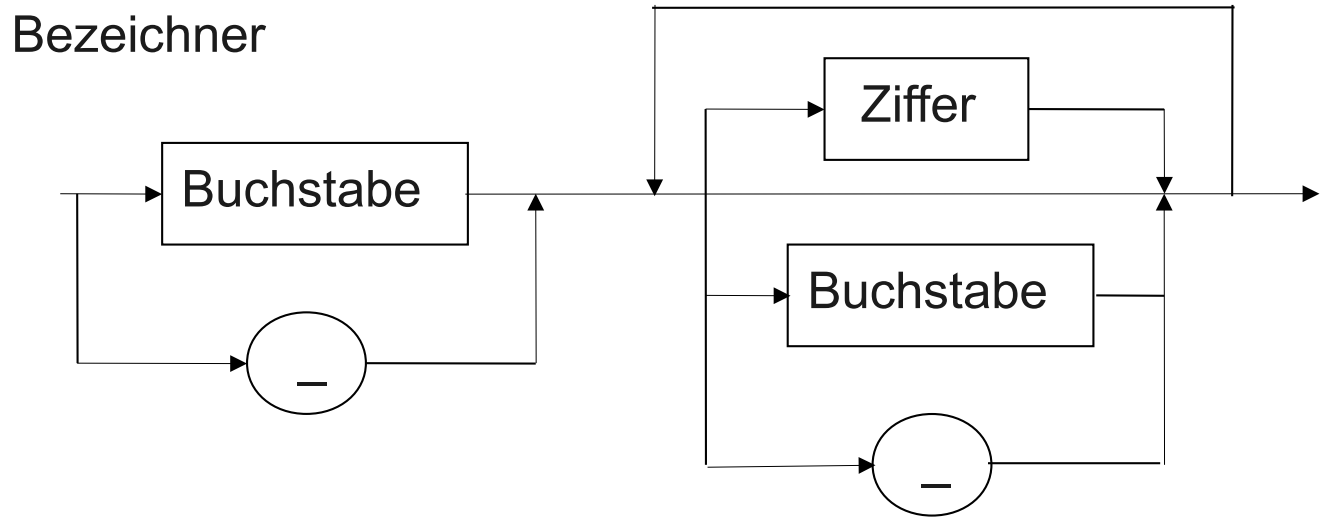
$N = \{\text{Bezeichner}, \text{ Buchstabe}, \text{ Ziffer}\}$

Sei Δ die Menge der folgenden drei Syntaxdiagramme:





Beispiel: SDmSD II



Dann definiert (N, T, Δ, Bezeichner) eine Menge von Worten, die in Programmiersprachen häufig als Menge der zulässigen Namen/Bezeichner verwendet wird.

Beispiel: Programmiersprache "Femto" I

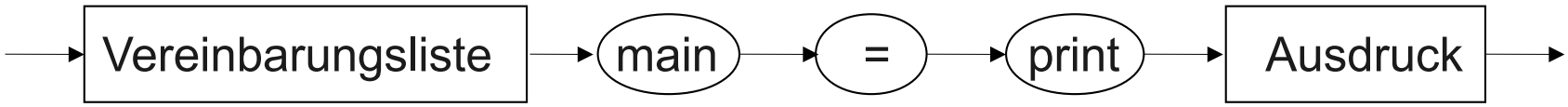
Als realistischeres Beispiel für eine Sprachdefinition betrachteten wir Vereinbarungslisten mit nachfolgender Ausgabevereinbarung.

Hier ein Beispiel für ein Femto-Programm:

```
a    = 73 ;
b    = ( 8 * a ) ;
main = print ( a + b )
```

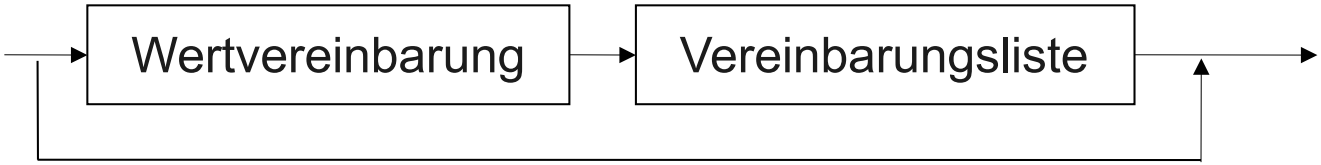
Definition mit Syntaxdiagrammen

Programm:

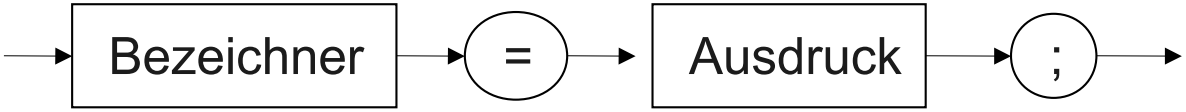


Beispiel: Programmiersprache “Femto” II

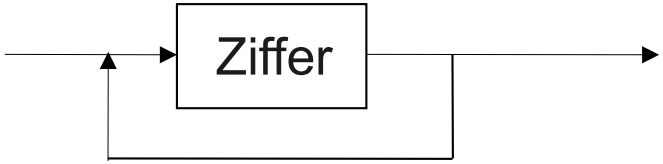
Vereinbarungsliste:



Wertvereinbarung:

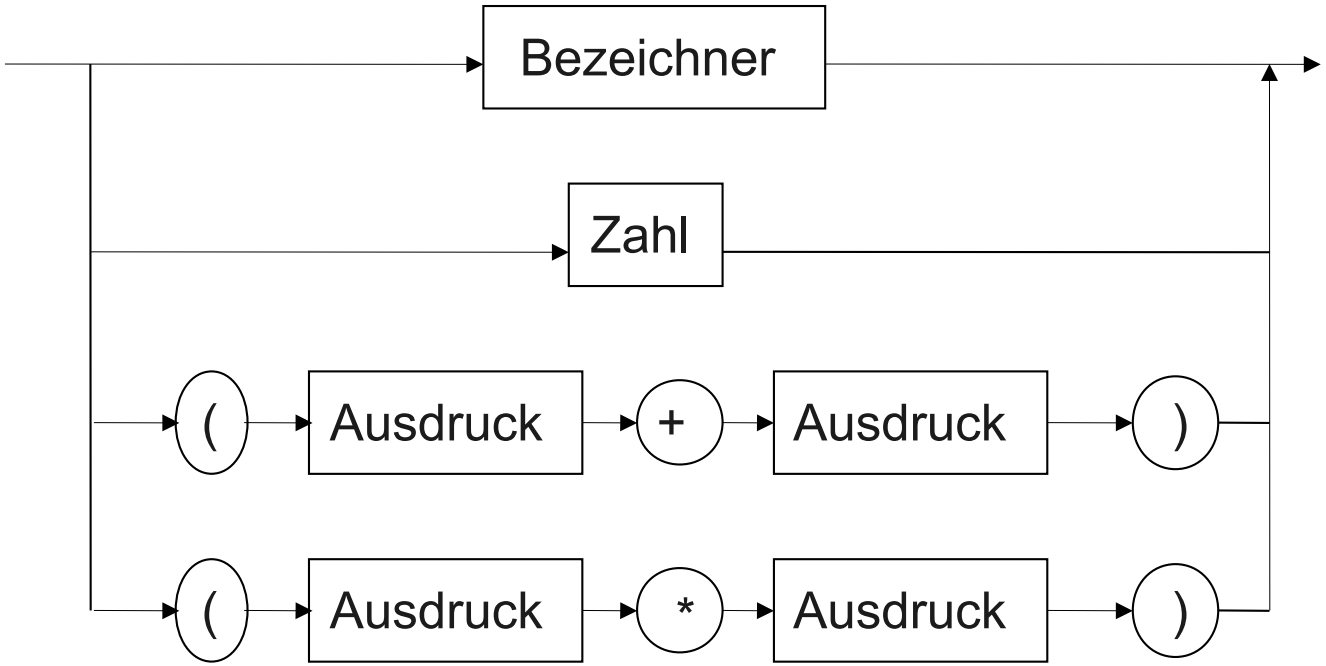


Zahl:



Beispiel: Programmiersprache “Femto” III

Ausdruck:



Beispiel: Programmiersprache “Femto” IV

Seien T , N, Δ wie im vorherigen Beispiel und

- $TF = T \cup \{\text{main}, \text{print}, (,), *, +, =, ;\}$
- $NF = N \cup \{$ Programm, Vereinbarungsliste, Wertvereinbarung, Ausdruck, Zahl $\}$
- Φ die Menge der obigen Syntaxdiagramme

Dann beschreibt $(TF, NF, \Delta \cup \Phi,$ Programm) die Syntax der Femto-Programme.

Definition: Kontextfreie Grammatik

Seien T und N zwei disjunkte Alphabete.

Die Elemente von T heißen *Terminalsymbole*, die von N *Nichtterminalsymbole*.

Sei Π eine endliche Teilmenge von $N \times (N \cup T)^*$.
Die Elemente von Π heißen *Produktionen* oder *Regeln*.

Sei $S \in N$; S heißt *Startsymbol* oder *Axiom*.

Dann heißt $\Gamma = (N, T, \Pi, S)$ eine **kontextfreie Grammatik**, kurz **kfG**.

Notationen

- Produktionen werden in der Form $A \to \alpha$ notiert.
- Abkürzend steht $A \to \alpha \mid \beta \mid \gamma \mid \ldots$ für die Regeln $A \to \alpha$, $A \to \beta$, $A \to \gamma$, $A \to \ldots$

Beispiele: Kontextfreie Grammatik I

1. Grammatik für balancierte Klammerausdrücke:

$$
\begin{aligned}
T &= \{(,)\} \\
N &= \{S\} \\
\Pi &= \{S \to SS \mid (S) \mid \varepsilon\}
\end{aligned}
$$

Nichtterminal S ist Startsymbol.

Beispiele: Kontextfreie Grammatik II

2 Grammatik für Bezeichner in Programmiersprachen:

$$
\begin{array}{lcll}
T & = & \{ & a, b, ..., z, A, ..., Z, 0, 1, ..., 9, \_ \} \\
N & = & \{ & \text{Bezeichner, Buchstabe, Ziffer, BezeichnerRest,} \\
 & & & \qquad\qquad\qquad\qquad\qquad\qquad \text{BustaUnstri } \} \\
\Pi & = & \{ & \text{Ziffer} \quad\quad\quad \rightarrow \quad 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \ , \\
 & & & \text{Buchstabe} \quad\quad \rightarrow \quad a \mid b \mid \ \ldots \ \mid z \mid A \mid \ \ldots \ \mid Z \ , \\
 & & & \text{Bezeichner} \quad\quad \rightarrow \quad \text{BustaUnstri BezeichnerRest} \ , \\
 & & & \text{BustaUnstri} \quad\quad \rightarrow \quad \text{Buchstabe} \mid \ '\_' \ , \\
 & & & \text{BezeichnerRest} \quad \rightarrow \quad \varepsilon \\
 & & & \qquad\qquad\qquad\quad \mid \quad \text{Ziffer BezeicherRest} \\
 & & & \qquad\qquad\qquad\quad \mid \quad \text{BustaUnstri BezeichnerRest} \\
 & & \} &
\end{array}
$$

Definitionen: Begriffe zur Ableitbarkeit I

Sei eine kontextfreie Grammatik $\Gamma = (N, T, \Pi, S)$ gegeben. Im Folgenden seien

- $A, B, C, \ldots$ aus N,
- $a, b, c, \ldots$ aus T,
- $x, y, z, \ldots$ aus T^* und
- $\alpha, \beta, \gamma, \psi, \varphi, \sigma, \tau, \ldots$ aus $(N \cup T)^*$.

Beispiel: Balancierte Klammerausdrücke

$$\begin{aligned} T &= \{(,)\} \\ N &= \{S\} \\ \Pi &= \{S \to SS \mid (S) \mid \varepsilon\} \end{aligned}$$

Für diese Grammatik kann man z.B. $\varphi = ((SS)S)$ betrachten.

Definitionen: Begriffe zur Ableitbarkeit II

- ψ ist mit Γ aus φ **direkt ableitbar**, in Zeichen $\varphi \Rightarrow \psi$, wenn es Zerlegungen $\sigma A \tau$ von φ und $\sigma \alpha \tau$ von ψ gibt und $A \to \alpha$ in Π.

  Beispiel: $\psi = ((SS)(S))$ ist aus $\varphi = ((SS)S)$ direkt ableitbar.

  Betrachte z.B. folgende Zerlegung:

Definitionen: Begriffe zur Ableitbarkeit III

- ψ ist mit Γ aus φ **ableitbar**, in Zeichen $\varphi \Rightarrow^* \psi$, wenn es $\varphi_0, \ldots, \varphi_n$ gibt mit $\varphi = \varphi_0$, $\varphi_n = \psi$ und für alle $i \in \{0, \ldots, n-1\} : \varphi_i \Rightarrow \varphi_{i+1}$

  $\varphi_0 \ldots \varphi_n$ heißt dann **Ableitung** von ψ aus φ.

  Beispiel: $\psi = (S()((S)))$ ist aus $\varphi = (S)$ ableitbar.

  Betrachte z.B. folgende Ableitung:

Definitionen: Begriffe zur Ableitbarkeit IV

- Eine Ableitung $\varphi_0 \ldots \varphi_n$ heißt **Linksableitung** (bzw. **Rechtsableitung**), wenn in φ_i jeweils nur das am weitesten links (bzw. rechts) stehende Nichtterminal ersetzt wird.

  Linksableitungsschritte werden als $\varphi \underset{lm}{\Rightarrow} \psi$,

  Rechtsableitungsschritte als $\varphi \underset{rm}{\Rightarrow} \psi$ notiert.

  Beispiel:
  Eine Linksableitung von $\psi = (()(()))$ aus $\varphi = (S)$ ist:

  Eine Rechtsableitung von $\psi = (()(()))$ aus $\varphi = (S)$ ist:

Definitionen: Begriffe zur Ableitbarkeit V

- $L(\Gamma) = \{z \in T^* \mid S \Rightarrow^* z\}$ heißt die von Γ *erzeugte* **Sprache**.
- $x \in L(\Gamma)$ heißt ein **Satz** von Γ.
- $\psi \in (N \cup T)^*$ mit $S \Rightarrow^* \psi$ heißt eine **Satzform** von Γ.

Syntaxbäume

Die baumartige Darstellung einer Ableitung nennt man **Syntaxbaum**.
Beispiel: Syntaxbaum für die Ableitung

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$

Frage: Was ist der Syntaxbaum für die folgende Ableitung?

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()$$

Mehrdeutige Grammatik I

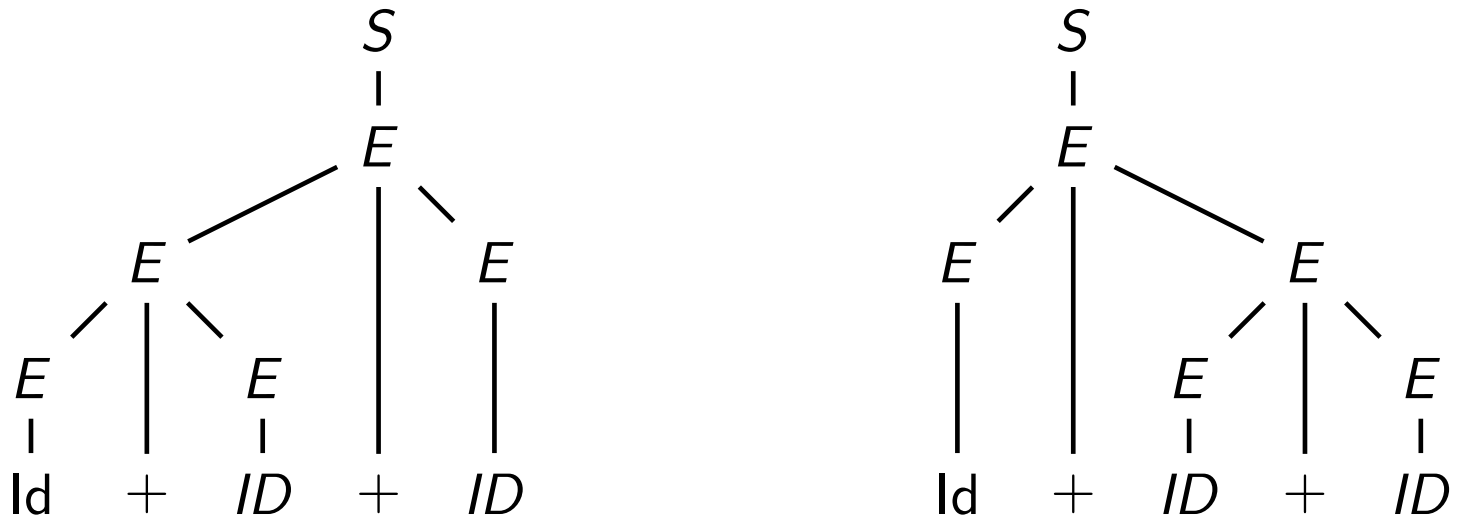
- Ein Satz heißt **mehrdeutig**, wenn er mehr als einen Syntaxbaum besitzt.
- Eine Grammatik heißt **mehrdeutig**, wenn sie einen mehrdeutigen Satz besitzt; andernfalls **eindeutig**.

Mehrdeutige Grammatik II

Die Grammatik Γ_0 für Ausdrücke mit den Produktionen

$$\begin{aligned} S &\rightarrow E, \\ E &\rightarrow E + E \mid E * E \mid (E) \mid ID \end{aligned}$$

ist ein Beispiel für eine mehrdeutige Grammatik, denn für $ID + ID + ID$ gibt es zwei verschiedene Syntaxbäume:



Bemerkungen I

- Jeder Ableitung entspricht genau ein Syntaxbaum. Umgekehrt kann es zu einem Syntaxbaum mehrere Ableitungen geben.
- Anstatt von Syntaxbaum spricht man häufig auch von **Struktur-** oder **Ableitungsbaum**.
- Zusammenhang zwischen Sprache und Grammatik:
  Die Abbildung L : Grammatik $\rightarrow$ Sprache ist im Allg. nicht injektiv; d.h. zu einer Sprache gibt es im Allg. mehrere erzeugende Grammatiken.
- $\Rightarrow^*$ ist die reflexive und transitive Hülle von $\Rightarrow$.

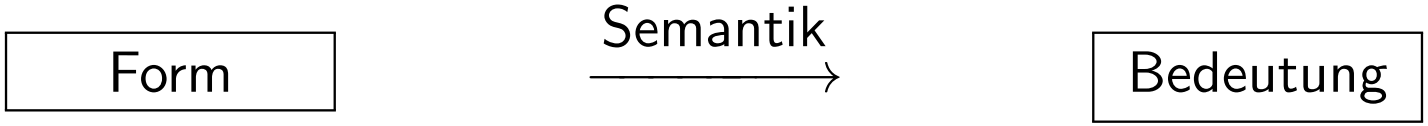
Bemerkungen II

- **Fakt**: Ein Satz ist genau dann eindeutig, wenn er genau eine Linksableitung (bzw. Rechtsableitung) besitzt.
- Bei Programmiersprachen spielen die *eindeutigen* Grammatiken die zentrale Rolle, da die Semantik (und Übersetzung) der Sprache über die syntaktische Struktur definiert wird.
- Mit kfGn lassen sich genau die Sprachen definieren, die man auch mit Syntaxdiagrammen definieren kann.

Begriffsklärung: Syntax vs. Semantik

Die **Syntax** einer formalen Sprache gibt an, wie die Elemente der Sprache (Sätze, Worte, Diagramme) zusammengesetzt sind, d.h. welche *Form* sie haben.

Die **Semantik** gibt den Elementen einer formalen Sprache eine *Bedeutung*.



Syntaktische Gebilde:
- Zeichenreihen
- Diagramme
- Programme
- Terme
- Formeln
- Spezifikationen
- ...

Semantische Gebilde:
- Zahlen
- Mengen
- Listen
- Funktionen
- Datenstrukturen
- Algebren
- ...

Bemerkungen

- Es gibt formale Sprachen ohne Semantik (z.B. die Sprache der balancierten Klammerausdrücke).
- Es gibt Sprachen mit formaler Syntax und informeller Semantik.
- Moderne Programmier- und Spezifikationssprachen besitzen formale Syntax und wohldefinierte Semantik.
- Syntax und Semantik zu unterscheiden bedeutet zwischen Beschreibung und Beschriebenem zu unterscheiden.
- Unterschiedliche Beschreibungen können die gleiche Semantik haben.

Beispiel: Informelle Semantik von Femto I

- Auf den Folien 31ff wurde die kontextfreie Syntax der Sprache Femto festgelegt.
- Zur Klärung der Semantik von Femto sind weitere syntaktische Einschränkungen notwendig (**Kontextbedingungen**):
  - Ein Bezeichner darf nur dann in einem Ausdruck angewendet werden, wenn er vorher vereinbart wurde.
  - Die Zahlen müssen Werte im Bereich von -2^{31} bis $2^{31} - 1$ beschreiben.

Beispiel: Informelle Semantik von Femto II

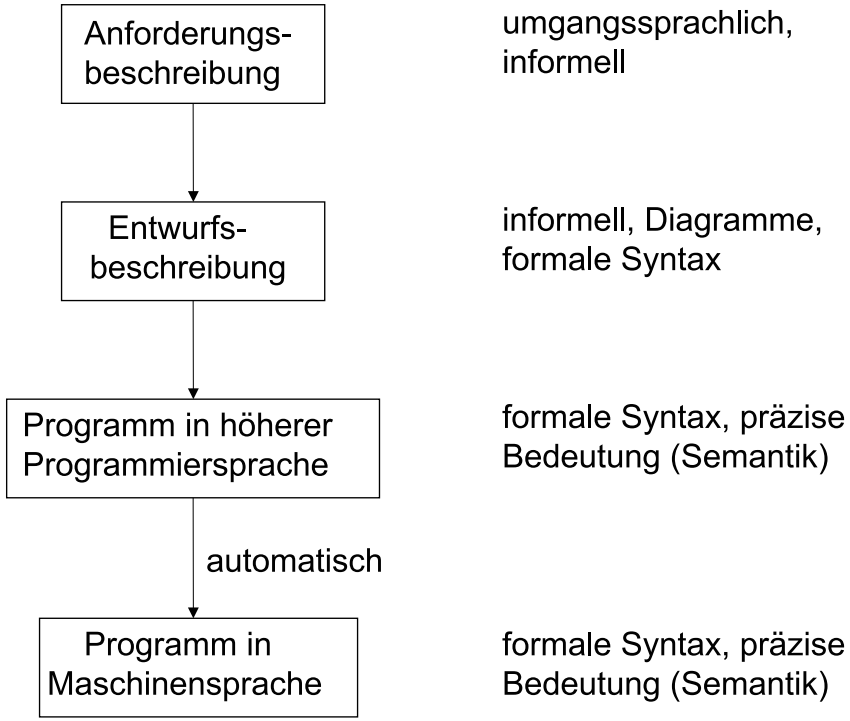
- Die Semantik weist jedem Ausdruck und dem ganzen Programm einen Wert im Bereich $[-2^{31}, 2^{31} - 1]$ zu:
  - angewandter Bezeichner: Liefert den an den Bezeichner gebundenen Wert.
  - Zahl: Liefert den von der Zahl beschriebenen Wert.
  - zusammengesetzter Ausdruck:
    - $\rightarrow$ Werte die Teilausdrücke aus; die Ergebnisse seien *lw* und *rw*.
    - $\rightarrow$ Ist das Operatorzeichen $'+'$, addiere *lw* und *rw*; ist das Operatorzeichen $'*'$, multipliziere *lw* und *rw*; das Ergebnis sei mit *w* bezeichnet.
    - $\rightarrow$ Liegt *w* im Bereich -2^{31} bis $2^{31} - 1$, ist *w* das Ergebnis der Ausdrucksauswertung; andernfalls ist das Ergebnis unbestimmt.
  - Wertvereinbarung: Werte den Ausdruck aus und binde das Ergebnis an den Bezeichner.
  - Programm: Drucke den Wert des Ausdrucks in der Ausgabevereinbarung.

Bemerkungen

- Syntaktisch inkorrekte Programme haben keine Semantik:
  z.B. : main $a == a; a;$
- Syntaxfehler werden vom Übersetzer bzw. Interpreter entdeckt.
- Eine formale Semantik definiert die Bedeutung mit mathematischen Mitteln.
- Die Implementierung einer Sprache in Form von Übersetzern/Interpretern liefert auch eine Semantik, die allerdings oft rechnerabhängig ist.

Beschreibung von SW-Systemen

Bei der Entwicklung von SW-Systemen werden auf vielen Ebenen Beschreibungen eingesetzt (vgl. Folie 3):



Dabei werden informelle und formale Sprachen kombiniert.

Frage: Was sind die Vor- und Nachteile von informeller gegenüber formaler Beschreibung?

Diskussion: Informelle vs. formale Beschreibungen

- Informelle Beschreibungen:
  - besser lesbar und besser vermittelbar
  - oft kompakter
  - flexibler und mächtiger, d.h. sie können sich auch auf Bereiche beziehen, die sich formal nicht oder nur schwer ausdrücken lassen
  - weniger Vorkenntnisse
- Formale Beschreibungen:
  - präziser
  - maschinell auswertbar (!!!!!)
  - zugänglich für mathematische Methoden
- Kombination formaler und informeller Sprachen:
  - Anforderungen anfangs informell
  - schrittweiser Übergang möglich und wichtig (!)

Zusammenfassung

- Formale Sprachen und ihre Spezifikation
  - Syntaxdiagramme
  - Kontextfreie Grammatiken
- Syntax beschreibt die Form, Semantik die Bedeutung von Sprachelementen.
- Programmiersprache "Femto": Syntax und Semantik

Ausblick auf nächste Vorlesung:

- Unser erstes Java-Programm!
- Integrierte Datentypen in Java
- Typkonvertierungen
- Ein- und Ausgabe von Daten