2.2.7 Auswertung von Ausdrücken in Haskell

Auswertung durch Einsetzungs- (Reduktions-) verfahren:

Ersetze in einem Ausdruck f solange Teilausdrücke durch ihre Definitionen oder berechne den Wert von Teilausdrücken, in denen nur primitive Funktionen vorkommen, bis keine Reduktion mehr möglich. Auswertung erfolgt von außen nach innen und von links nach rechts gemäß Operatorpräferenz

Reduzierbarer Teilausdruck heißt Redex (reducible expression)

Auswertung durch Reduktion: einfaches Beispiel

Beispiel:

```
quadrat  n = n * n
hochVier n = quadrat (quadrat n)
```

Reduziere `hochVier 3`

```
hochVier 3  → quadrat (quadrat 3)
            → (quadrat 3)*(quadrat 3)
            → (3*3)*(quadrat 3)
            → 9*(quadrat 3)
            → 9*(3*3) → 9*9 → 81
```

2 Grundlagen

Syntax Definition von Programmiersprachen

Lernziele

- ´Begriffe 'Syntax' und 'Semantik' kennen
- Syntaktische Definitionen
  - BNF, EBNF
  - Syntaxdiagramme
  - Formale Sprachen (elementar)
- Syntaxbäume
- Zusammenhang mit Grammatiken / Formalen Sprachen verstehen

Syntax

- Syntax beschreibt die lexikalische und grammatikalische Struktur einer Sprache

  Welche Ausdrücke sind *formal* korrekt ?

  Umgangssprache, Programmiersprache, ...

  ```
  f x = f x
  „Der Hase jagt den Hund"
  ```

  Wie beschreibt man unendlich viele Ausdrücke?
  z.B. alle gültigen arithmetischen Ausdrücke, alle syntaktisch korrekten Haskell-Programme….
- Semantik beschreibt die Bedeutung von Ausdrücken einer Sprache

Syntax

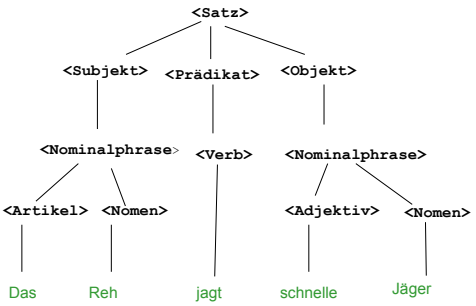
- Aussagen über Sprachen erfordern **Metasprache**

  „Der Satz „*Morgen ist Weihnachten*" ist syntaktisch korrekt"

  Umgangssprache ist Metasprache der Umgangssprache!
- Grammatik

  Regelsystem, mit Hilfe dessen syntaktische Korrektheit überprüft werden kann

Syntaxbaum: grammatikalische Zerlegung



Syntax von Programmiersprachen

- Programmiersprachen werden auch durch formale Grammatik beschrieben
  - (Erweiterte) Backus-Naur-Form (EBNF) : eher für maschinelle Prüfung der syntaktischen Korrektheit geeignet
  - Syntaxdiagramme : gut geeignet für menschliche Leser
  - Formale Grammatiken: kontextfreie Grammatiken im wesentlichen äquivalent zu BNF

Syntax

Terminalsymbole: kommen in Ausdrücken der Sprache vor
z.B. `[ , ] , ( , ) , a,b,c...`

Nichtterminale Symbole : bezeichnen syntaktische Begriffe,
die weiter expandiert werden müssen
z.B. : **`bezeichner, expression,...`**

Metasprachliche Symbole: Elemente der Definitionssprache

`< bezeichner> ::=` **`.....`**

Metasprachliche Symbole

Backus-Naur-Form

John Backus, Peter Naur (1959, zur Def. der Sprache **ALGOL 60**)

- BNF: einfache Metasprache zur Definition der Syntax einer (Programmier-)Sprache
- Elemente:
  - Regeln:
    <nicht-terminal-Symbol> ::= <Definition>
  - Zur Definiton werden verwendet:
    Sequenz
    z.B. <2-stellige Zahl> ::= <ziffer><ziffer>,
    oder: <agent> ::= 007
    Alternative
    z.B. <ziffer> ::= 0|1|2|3|4|5|6|7|8|9
    Rekursion
    z.B. <zahl> = <ziffer><zahl> | <ziffer>

BNF

- Beispiel

<bool > ::= True |False|<bool_Variable>|
¬<bool> | <bool> ∨ <bool> | <bool>∧ <bool> |(<bool>)

<bool_Variable> ::= <zeichen> | <zeichen><bool_Variable>
<zeichen> ::= a| b | c |d

Extended Backus-Naur Form

Vereinfachte Metasprache zur kompakteren metasprachlichen Beschreibung der Syntax

- (Nichtterminale in spitzen Klammern: **`<variable>`**)
- (Terminale in Hochkommata (heute meist durch Schriftart unterschieden))
- Optionale Teile in eckigen Klammern
  `<variable> ::= <klein > [<klein>]..`
- nachgestellter **`*`** : 0 oder mehrfache Wiederholung
- nachgestelltes **`+`** : 1 oder mehrfache Wiederholung (oder: geschweifte Klammern statt eckige)
- nachgestelltes **`n`** : n-malige Wiederholung
- Gruppierung von Elementen: **`(...)`**

Syntax: EBNF

```
<variable> ::= <klein>
               [<klein> |<ziffer>|<groß> | _ | ´]*
```

Verschiedenste Varianten im Umlauf
Im Haskell - Report werden z.B. Prioritäten von Ausdrücken durch hochgestellten Index bezeichnet.

Wichtig: alle Regeln sind kontextfrei, d.h. auf der linken Seite steht ausschließlich ein nichtterminales Symbol, das durch die rechte Seite definiert wird.

Syntaxanalyse

Folge: Effiziente Syntaxanalyse („ parsen“) von Ausdrücken möglich

Übersetzer (Compiler) erzeugt in der ersten Übersetzungsphase einen *Syntaxbaum*, der die Struktur des Ausdrucks widerspiegelt

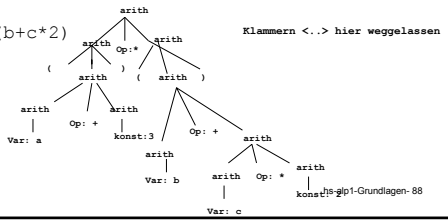
(„Parsetree“)

Syntax: EBNF

- Beispiel: einfache arithmetische Ausdrücke

```
<arith> ::= <variable> | <konstante> |
  (<arith>) |
            <arith> <op> <arith> | -<arith>
<op>    ::=  + | - | * | /
```

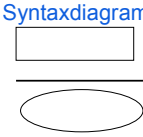
(a+3)*(b+c*2)



Klammern <..> hier weggelassen

Syntax

- Syntaxdiagramme

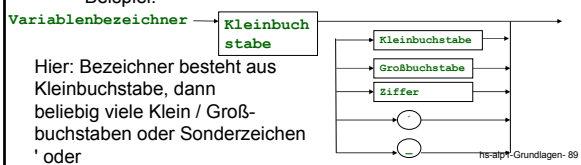


bezeichnet nichtterminales Symbol

gerichtete Verbindung

terminales Symbol

Beispiel:



Hier: Bezeichner besteht aus Kleinbuchstabe, dann beliebig viele Klein / Groß-buchstaben oder Sonderzeichen ' oder _

Syntaxdiagramme

Nichtterminalsymbol T

Terminales Symbol S

Option [T]

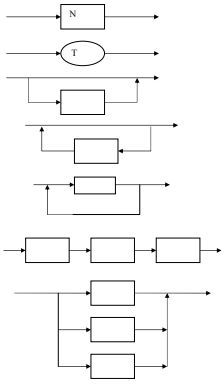
Wiederholung {T}

Mindestens einmalige Wiederholung (T)

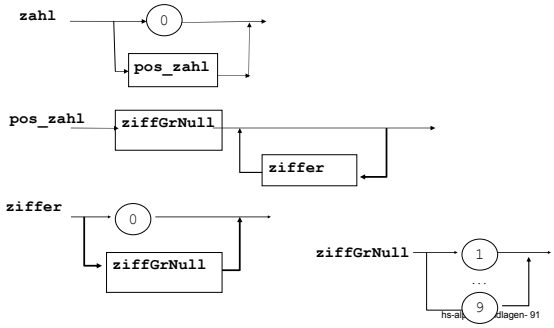
Sequenz T1 T2 T3

Alternative
T1 | T2 | T3

In den graphischen Syntaxregeln können selbstverständlich statt terminaler auch nicht-terminale Symbole stehen



Syntaxdiagramme



Syntax

- Überprüfung mit Syntaxdiagramm:

"Algorithmus":

- durchlaufe Diagramm in Pfeilrichtung
- wähle an Verzweigungspunkten beliebige Richtung
- wenn Nicht-Terminalsymbol `t` : suche das mit Beschriftung `t` auf und durchlaufe es
- Wenn Terminalsymbol, prüfe ob gleich nächstem Zeichen in w
- Wenn ja: streiche Zeichen und setze fort
  sonst wähle an letztem Verzweigungspunkt andere Alternative
  wenn es die gibt, sonst vorletzter Verzweigungspunkt … (rekursiv!)
- Wenn Ausgangspunkt erreicht und alle Zeichen von w konsumiert: syntaktisch korrekt, sonst Syntaxfehler

Grammatiken: Theorie

- Frage: Wie kann man die Kompliziertheit einer Sprache erfassen?
- Theoretische Informatik / formale Linguistik:
  Generative Grammatiken
  (Noam Chomsky ~ 1965)
- Formale Sprache =
  Menge von Wörtern über einem endlichen Alphabet A
- Generative Grammatik:
  Regelsystem, aus dem sich alle Wörter erzeugen lassen

Grammatiken

- Regelsystem
  - Terminale Symbole A = Alphabet
  - Nichtterminale Symbole Σ: endliche Menge von Symbolen, die nicht in A vorkommen
  - Startsymbol $\sigma \in N$
  - Endliche Menge R von Regeln der Form
    $(A \cup \Sigma)^* \rightarrow (A \cup \Sigma)^*$,
    wobei X^* = Menge aller Zeichenketten endlicher Länge über der endlichen Menge X
    und die linke Seite einer Regel darf nicht leer (Zeichenkette der Länge 0) sein.

Beispiel: $A = \{ a,b,c\}$, $\Sigma = \{\sigma,\beta)$, Startsymbol σ
$R=\{\sigma \rightarrow abc,\ \sigma \rightarrow a\beta\sigma c,\ \beta a \rightarrow a\beta,\ \beta b \rightarrow bb\}$

Beispielableitung: $\sigma \rightarrow a\beta\sigma c \rightarrow a\beta a\beta\sigma cc \rightarrow aa\beta\beta bcc \rightarrow aabbcc$

Gammatiken

- Kontextfreie Grammatiken
  Linke Seite einer Regel besteht nur aus *einem nichtterminalen* Zeichen.

Beispiel:
A = {t,f, $\wedge$, $\vee$, $\neg$, (,) }, Σ = {σ}, Startsymbol σ,
R = { $\sigma \rightarrow t$, $\sigma \rightarrow f$, $\sigma \rightarrow \neg\sigma$, $\sigma \rightarrow \sigma \wedge \sigma$, $\sigma \rightarrow \sigma \vee \sigma$, $\sigma \rightarrow (\sigma)$

$\sigma \rightarrow \sigma \vee \sigma \rightarrow \neg\sigma \vee \sigma \rightarrow \neg(\sigma) \vee \sigma \rightarrow \neg(\sigma \wedge \sigma) \vee \sigma$
$\rightarrow \neg(t \wedge \sigma) \vee \sigma \rightarrow \neg(t \wedge f) \vee \sigma \rightarrow \neg(t \wedge f) \vee f$

Grammatik

- Programmiersprachen: immer kontextfrei definiert
- Man kann zeigen: nicht jede Sprache kann mit einer kontextfreien Grammatik erzeugt werden
  (z.B. $a^n b^n c^n$ nicht)
  -> Grundlagen der theoretischen Informatik
- Ferner: Untersuchung von Grammatiken, die die gleiche Sprache erzeugen -> Normalformen.
- Umkehrung des Erzeugungsprozesses: Zerlegung.
  Genügt ein "Wort" den grammatischen Regeln?
  D.h. gehört es zur Sprache?
- Parser: Programm, das ein Eingabeprogramm auf syntaktische Korrektheit überprüft.