

## Übersicht

- Attributgrammatik
- Eigenschaften von Attributgrammatiken
- Semantik von Attributgrammatiken
- Beispiele von Attributgrammatiken
- Generierung von Attributauswertern
- Attributabhängigkeiten
- Attributauswertung
- Unterklassen von Attributgrammatiken
  - L-attributierte Grammatiken
  - LL-attributierte Grammatiken
  - LR-attributierte Grammatiken
  - Attributierung in YACC/Bison

## Lernziele

- Das Grundprinzip und die wichtigsten theoretischen Aspekte von Attributgrammatiken beherrschen
- Grundideen und wichtigste Algorithmen zur Bestimmung von Attributabhängigkeiten und zur Attributauswertung benennen können
- Das Prinzip der L-Attributierung und seine Integration in die Syntax-Analyse wiedergeben können

## Attributgrammatik intuitiv

- „Erweitert“ kfG um „semantische Information“
- Assoziiert „**Attribute**“ als Träger (statischer) semantischer Information mit den Symbolen der zugrunde liegenden kfG
- Gibt funktionale Abhängigkeiten zwischen den Werten von **Attributvorkommen** in den Produktionen der Grammatik an  
(mit Bedingungen für die Auswertbarkeit aller **Attributexemplare**)

„Fluss der Information  
durch die Produktionen“

## Beachte Begriffsbildungen

Präzise Definitionen, s.u.

- **Attribut** ➡ : allgemeiner Begriff
- **Attributvorkommen** ➡ : Vorkommen eines Attributs in einer Produktion
- **Attributexemplar** ➡ : Vorkommen eines Attributs im Syntaxbaum

## Attributgrammatik formal (Teil 1)

- Sei  $G = (V_N, V_T, P, S)$  eine kfG mit  $V =_{\text{def}} V_N \cup V_T$
- Für jedes Symbol  $X \in V$ 
  - $\text{Inh}(X)$ : Menge der **ererbten** (*inherited*) **Attribute** von  $X$
  - $\text{Syn}(X)$ : Menge der **abgeleiteten** (*synthesized*) **Attribute** von  $X$  mit  $\text{Inh}(X) \cap \text{Syn}(X) = \emptyset$
  - $\text{Attr}(X) =_{\text{def}} \text{Inh}(X) \cup \text{Syn}(X)$ : **Attribute** von  $X$
- Für kfG  $G$ 
  - $\text{Inh} =_{\text{def}} \bigcup_{X \in V} \text{Inh}(X)$ ,  $\text{Syn} =_{\text{def}} \bigcup_{X \in V} \text{Syn}(X)$ ,  $\text{Attr} =_{\text{def}} \text{Inh} \cup \text{Syn}$
- Für jedes Attribut  $a \in \text{Attr}$ 
  - $D_a$ : **Wertebereich** für  $a$

## Beispiel

- Geg.: kfG  $G_0$  mit Produktionen
  - $E \rightarrow E + T$ ,  $E \rightarrow T$ ,  $p:T \rightarrow T * F$ ,  $q:T \rightarrow F$ ,  $F \rightarrow (E)$ ,  $F \rightarrow \text{id}$
- Attribute
  - $\text{Syn}(X) =_{\text{def}} \{\text{val} \mid X \in V_N \cup \{\text{id}\}\}$
  - $\text{Inh}(X) =_{\text{def}} \{\text{depth} \mid X \in V_N \cup \{\text{id}\}\}$
- Wertebereiche von val, depth: nat

## Attributgrammatik formal (Teil 2)

- Für jede Produktion  $p: X_0 \rightarrow X_1 \dots X_{n_p}$ , mit  $X_i \in V$ ,  $1 \leq i \leq n_p$ ,  $X_0 \in V_N$ 
  - $a_i$ : **Vorkommen** des Attributs  $a$  in  $p$  (falls  $a \in \text{Attr}(X_i)$ )
  - $V(p)$ : Menge aller Attributvorkommen in  $p$   $\in \text{Attr}(X_{j_l})$
  - Semantische Regeln**  $a_i = f_{p,a,i}(b^1_{j_1}, \dots, b^k_{j_k})$  ( $0 \leq j_l \leq n_p$ ) ( $1 \leq l \leq k$ )  
für jedes Vorkommen  $a_i$  ( $0 \leq i \leq n_p$ ), wobei
    - $a \in \text{Inh}(X_i)$  (für  $1 \leq i \leq n_p$ )  $\longrightarrow$
    - $a \in \text{Syn}(X_0)$  (für  $i = 0$ )
    - $f_{p,a,i}: D_{b_1} \times \dots \times D_{b_k} \rightarrow D_a$

*D.h. definiert werden nur*  
 - abgeleitete Attributvorkommen für die linke Seite  
 - ererbte Attributvorkommen für die rechte Seite

## Beispiel (Fortsetzung)

- Geg.: kfG  $G_0$  mit  $E \rightarrow E + T$ ,  $E \rightarrow T$ ,  $p: T \rightarrow T * F$ ,  $q: T \rightarrow F$ ,  $F \rightarrow (E)$ ,  $F \rightarrow \text{id}$
- Attributvorkommen
  - $V(p) = \{\text{val}_0, \text{val}_1, \text{val}_3\} \cup \{\text{depth}_0, \text{depth}_1, \text{depth}_3\}$
  - $V(q) = \{\text{val}_0, \text{val}_1\} \cup \{\text{depth}_0, \text{depth}_1\}$
- Semantische Regeln
 

<ul style="list-style-type: none"> <li>Für <math>p</math>:               <math display="block">\begin{aligned} \text{val}_0 &amp;= \text{val}_1 \times \text{val}_3 \\ \text{depth}_1 &amp;= \text{depth}_0 + 1 \\ \text{depth}_3 &amp;= \text{depth}_0 + 1 \end{aligned}</math> </li> </ul>	<ul style="list-style-type: none"> <li>Für <math>q</math>:               <math display="block">\begin{aligned} \text{val}_0 &amp;= \text{val}_1 \\ \text{depth}_1 &amp;= \text{depth}_0 + 1 \end{aligned}</math> </li> </ul>
--	--

## Definierende und angewandte Attributvorkommen, Normalform

- Geg.: Attributgrammatik AG, Produktion  $p: X_0 \rightarrow X_1 \dots X_{n_p}$
- Dann
  - **Definierende Attributvorkommen:**  $a_i$  mit  $a \in \text{Inh}(X_i)$  ( $1 \leq i \leq n_p$ ) und  $a_0$  mit  $a \in \text{Syn}(X_0)$
  - **Angewandte Attributvorkommen:** alle anderen Vorkommen
  - AG **ist in Normalform:** alle Argumente semantischer Regeln sind angewandte Vorkommen

*D.h. z.B. die Attributvorkommen, die durch semantische Regeln definiert werden*

## Intuitiv

- Attributgrammatik ist in Normalform  $\Leftrightarrow$   
für jedes definierende Attributvorkommen in einer Produktion gibt es genau eine semantische Regel und deren Argumente sind ausschließlich angewandte Vorkommen

## Beispiel

- Geg.: Produktion  $p: T \rightarrow T * F$  mit semantischen Regeln

$$\begin{aligned} \text{val}_0 &= \text{val}_1 \times \text{val}_3 \\ \text{depth}_1 &= \text{depth}_0 + 1 \\ \text{depth}_3 &= \text{depth}_0 + 1 \end{aligned}$$

- Dann

- Definierend:  $\text{val}_0, \text{depth}_1, \text{depth}_3$
- Angewandt:  $\text{val}_1, \text{val}_3, \text{depth}_0$

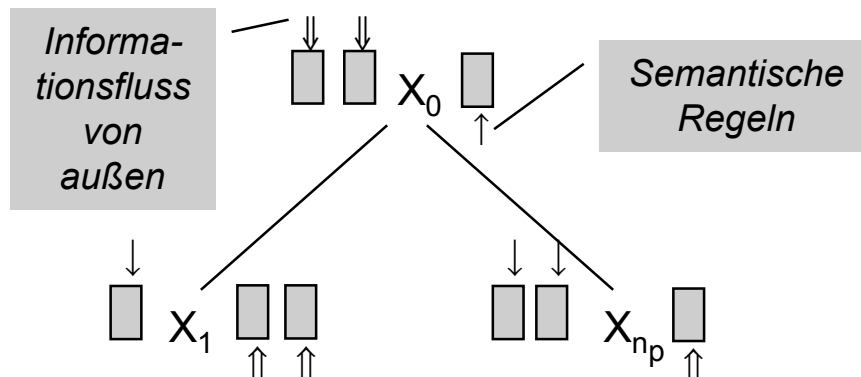
*Damit Normalform-Eigenschaft für p erfüllt*

## Abgeleitete Attribute bei Terminalen

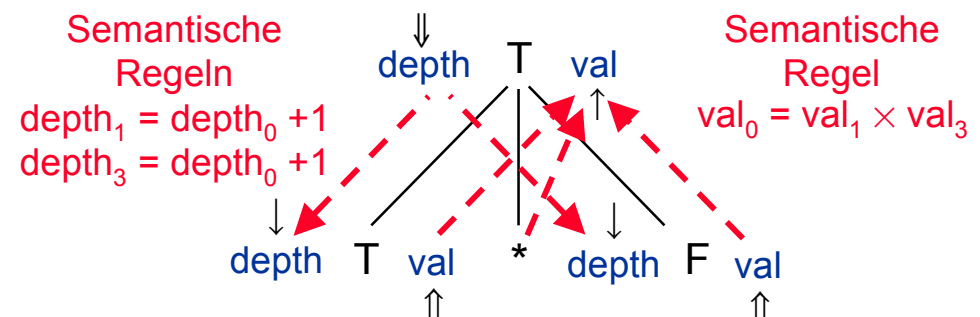
- Beispiele
  - Wert von Konstanten, Externdarstellung von Namen, Adressen von Stringkonstanten, ...
- Kommen nur als Argumente semantischer Regeln vor (werden nicht durch Regeln definiert)
- Wertzuordnung durch externe Regeln (in der Praxis: vom Scanner geliefert)

## Graphische Darstellung attributierter Produktionen

- Produktion als Baum dargestellt
- Ererbte Attributvorkommen: links (von den Knoten im Baum)
- Abgeleitete Attributvorkommen: rechts (von den Knoten im Baum)



## Schematischer Zusammenhang

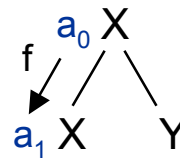


## Beispiel

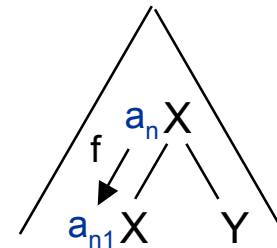
## Begriffe

- **Attribut** zu Symbol  $X$   
Element aus  $\text{Attr}(X)$
- **Attributvorkommen**  $a_i$  von  $a$  zu  $X$  in Produktion  $p$   
Eindeutige Zuordnung von  $a$  zu Vorkommen  $i$  von  $X$  in  $p$
- **Attributexemplar**  
Vorkommen  $a_{ni}$  von  $a$  in zu Knoten  $n$  gehöriger Produktion  $p$  im Syntaxbaum  $t$
- **Beispiel**
  - $a \in \text{Inh}(X)$
  - $p: X \rightarrow XY$
  - $a_1 = f(a_0)$ : semantische Regel zu  $p$

Attribut



Attributvorkommen



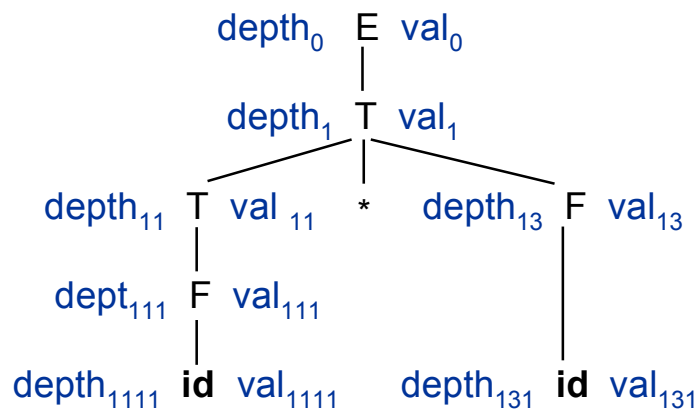
Attributexemplar

## Basis der Semantik einer Attributgrammatik (formal)

- Funktion value: Attributexemplare  $\rightarrow$  Wertebereich
  - Für Syntaxbaum  $t$  definiert durch Gleichungssystem, deren Unbekannte die Attributexemplare in  $t$  sind
  - Gleichungen  $\text{value}(a_{ni}) = f_{p,a,i}(\text{value}(b^1_{nj_1}), \dots, \text{value}(b^k_{nj_k}))$  für Knoten  $n$  und zu  $n$  gehöriger Produktion  $p$  sind durch semantische Regeln  $a_i = f_{p,a,i}(b^1_{j_1}, \dots, b^k_{j_k})$  induziert

## Beispiel

### Attributierter Baum



### Gleichungssystem

Für val

$$\begin{aligned}
 \text{val}_0 &= \text{val}_1 \\
 \text{val}_1 &= \text{val}_{11} \times \text{val}_{13} \\
 \text{val}_{11} &= \text{val}_{111} \\
 \text{val}_{111} &= \text{val}_{1111} & \text{val}_{1111} &= \dots \\
 \text{val}_{13} &= \text{val}_{131} & \text{val}_{131} &= \dots
 \end{aligned}$$

Analog für depth



## Eigenschaften des Gleichungssystems

- Zyklisch (oder rekursiv): keine oder mehrere Lösungen
- Azyklisch (oder nicht-rekursiv): genau eine Lösung (genau ein Wert pro Attributexemplar)

## Wohlgeformt

- Attributgrammatik heißt **wohlgeformt**, wenn für **alle** Syntaxbäume  $t$  das zugehörige Gleichungssystem azyklisch ist

## Semantik (einer wohlgeformten Attributgrammatik)

- Eindeutige Zuordnung von Attributwerten zu den Knoten der Syntaxbäume (der zugrunde liegenden kfG)

## Beispiele von Attributgrammatiken

- Varianten des Typberechnungsproblems für arithmetische Ausdrücke
- Konversion von Binärbrüchen in Dezimalzahlen

## Dabei

- Definition der funktionalen Beziehungen zwischen Attributvorkommen durch funktionale Sprache mit üblichen Eigenschaften, d.h.
  - Fallunterscheidung durch „pattern matching“
  - Unwesentliche Argumente durch „\_“ bezeichnet
  - Bei Überlappung verschiedener Fälle, Auswahl „speziellerer“ Fälle vor „allgemeineren“
  - Mehrfach vorkommende Variablen müssen beim matching auf die gleichen Werte treffen

**Attribute grammar**  $AG_1$ ;  
**Nonterminals** E, T, F, P, Aop, Mop;  
**Attributes** **syn** typ with E, T, F, P, c, v **domain** {int, real}  
**syn op with** Aop, Mop **domain** {'+', '-', '\*', '/', 'div'}

## Rules

1: $E \rightarrow E \text{ Aop } T$	$E_0.\text{typ} = f_1(E_1.\text{typ}, T.\text{typ})$
2: $E \rightarrow T$	$E.\text{typ} = T.\text{typ}$
3: $T \rightarrow T \text{ Mop } F$	$T_0.\text{typ} = f_3(\text{Mop.op}, T_1.\text{typ}, F.\text{typ})$
4: $T \rightarrow F$	$T.\text{typ} = F.\text{typ}$
5: $F \rightarrow P$	$F.\text{typ} = P.\text{typ}$
6: $P \rightarrow c$	$P.\text{typ} = c.\text{typ}$
7: $P \rightarrow v$	$P.\text{typ} = v.\text{typ}$
8: $P \rightarrow (E)$	$P.\text{typ} = E.\text{typ}$
9: $\text{Aop} \rightarrow +$	$\text{Aop.op} = '+'$
10: $\text{Aop} \rightarrow -$	$\text{Aop.op} = '-'$
11: $\text{Mop} \rightarrow *$	$\text{Mop.op} = '*'$
12: $\text{Mop} \rightarrow /$	$\text{Mop.op} = '/'$
13: $\text{Mop} \rightarrow \text{div}$	$\text{Mop.op} = \text{'div'}$

## Functions

$f_1 \text{ int int} = \text{int}$	$f_3 \text{ '*' int int} = \text{int}$
$f_1 \text{ -- } = \text{real}$	$f_3 \text{ 'div' int int} = \text{int}$
	$f_3 \text{ '*' -- } = \text{real}$
	$f_3 \text{ '/' -- } = \text{real}$

## Beobachtungen

- Hier nur abgeleitete Attribute
- Großteil der semantischen Regeln sind reine Gleichungen (identische Übergabe von Werten)

## Konvention (zur Schreibabkürzung)

- Fehlen einer semantischen Regel für (definierendes) Attributvorkommen bedeutet identische Übergabe von gleichnamigem Vorkommen:
  - Ererbt links  $\rightarrow$  ererbt rechts
  - Abgeleitet rechts  $\rightarrow$  abgeleitet links



**Attribute grammar  $AG_3$ :**

**Nonterminals** S, E, T, F, P, Aop, Mop;

**Attributes** syn typ with E, T, F, P, c, v domain {int, real}

syn op with Aop, Mop domain {'+', '-', '\*', '/', 'div'}

inh obltyp with E, T, F, P domain {int, unspec}

**Rules**

0:  $S \rightarrow E$  E.obltyp = unspec

1:  $E \rightarrow E \text{ Aop } T$   $E_0.\text{typ} = f_1(E_1.\text{typ}, T.\text{typ})$

2:  $E \rightarrow T$

3':  $T \rightarrow T \text{ Mop } F$   $T_0.\text{typ} = f_{31}(\text{Mop.op}, T_0.\text{obltyp}, T_1.\text{typ}, F.\text{typ}),$

$T_1.\text{obltyp} = f_{32}(\text{Mop.op}, T_0.\text{obltyp}), F.\text{obltyp} = f_{32}(\text{Mop.op}, T_0.\text{obltyp})$

4:  $T \rightarrow F$

5:  $F \rightarrow P$

6:  $P \rightarrow c$   $P.\text{typ} = f_6(P.\text{obltyp}, c.\text{typ})$

7:  $P \rightarrow v$   $P.\text{typ} = f_6(P.\text{obltyp}, v.\text{typ})$

8:  $P \rightarrow (E)$

9:  $\text{Aop} \rightarrow +$  Aop.op = '+'

10:  $\text{Aop} \rightarrow -$

Aop.op = '-'

11:  $\text{Mop} \rightarrow *$  Mop.op = '\*'

12:  $\text{Mop} \rightarrow /$

Mop.op = '/'

13:  $\text{Mop} \rightarrow \text{div}$  Mop.op = 'div'

**Functions**

$f_1 \text{ int int} = \text{int}$

$f_1 \text{ _ _} = \text{real}$

$f_{31} '*' \text{ int int int} = \text{int}$

$f_{31} 'div' \text{ _ int int} = \text{int}$

$f_{31} '*' \text{ _ _ _} = \text{real}$

$f_{31} '/' \text{ unspec _ _} = \text{real}$

$f_{32} 'div' \text{ _} = \text{int}$

$f_{32} \text{ _ int} = \text{int}$

$f_{32} \text{ _ _} = \text{unspec}$

$f_6 \text{ int int} = \text{int}$

$f_6 \text{ unspec t} = t$

## Attributgrammatik

**Attribute grammar** Bin\_to\_Dec;  
**Nonterminals** N, BIN, BIT;  
**Attributes** **syn** l with BIN domain {int}  
                   **syn** v with N, BIN, BIT domain {real}  
                   **inh** r with BIN, BIT domain {int}

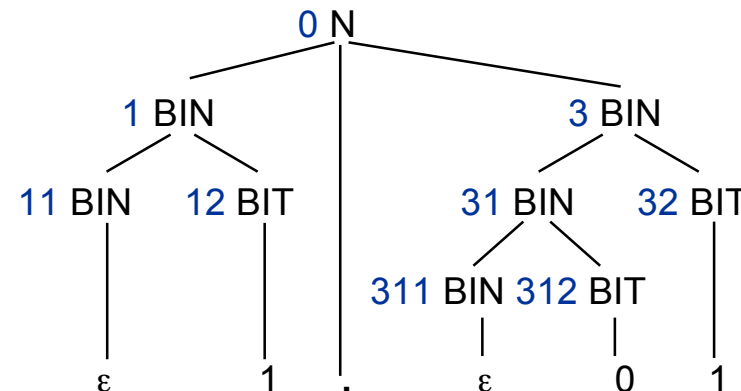
**Rules**

- 1:  $N \rightarrow \text{BIN.BIN}$        $N.v = \text{BIN}_1.v + \text{BIN}_3.v$   
                                   $\text{BIN}_1.r = 0$   
                                   $\text{BIN}_3.r = -\text{BIN}_3.l$
- 2:  $\text{BIN} \rightarrow \text{BIN BIT}$        $\text{BIN}_0.v = \text{BIN}_1.v + \text{BIT.v}$   
                                   $\text{BIN}_0.l = \text{BIN}_1.l + 1$   
                                   $\text{BIN}_1.r = \text{BIN}_0.r + 1$   
                                   $\text{BIT.r} = \text{BIN}_0.r$
- 3:  $\text{BIN} \rightarrow \varepsilon$                $\text{BIN.v} = 0$   
                                   $\text{BIN.l} = 0$
- 4:  $\text{BIT} \rightarrow 1$                $\text{BIT.v} = 2^{\text{BIT.r}}$
- 5:  $\text{BIT} \rightarrow 0$                $\text{BIT.v} = 0$

Intuition:

- v = „value“ (Dezimalwert)
- r = „rank“ (Stellenwert)
- l = „length“ (Länge)

## Syntaxbaum (Beispiel)



## Gleichungssystem (22 Gleichungen mit 22 Unbekannten)

<p>0    <math>v_0 = v_1 + v_3</math> 5/41  <math>r_1 = 0</math>  <math>r_3 = -l_3</math> 2</p>	<p>11    <math>v_{11} = 0</math>  <math>l_{11} = 0</math></p>	<p>12    <math>v_{12} = 2^{r_{12}}</math> 1</p>	<p>31    <math>v_{31} = v_{311} + v_{312}</math> 0  <math>l_{31} = l_{311} + 1</math> 1  <math>r_{311} = r_{31} + 1</math> 0  <math>r_{312} = r_{31} - 1</math></p>
<p>311    <math>v_{311} = 0</math>  <math>l_{311} = 0</math></p>	<p>32    <math>v_{32} = 2^{r_{32}}</math> 1/4</p>	<p>312    <math>v_{312} = 0</math></p>	

## Aufgabe

- Auswertung von Attributexemplaren in Syntaxbäumen t
  - Bereits betrachtet: Attributgrammatik definiert Gleichungssystem für t mit Attributexemplaren als Unbekannten
  - Damit: Auswertung von Attributexemplaren = Lösung dieses Gleichungssystems

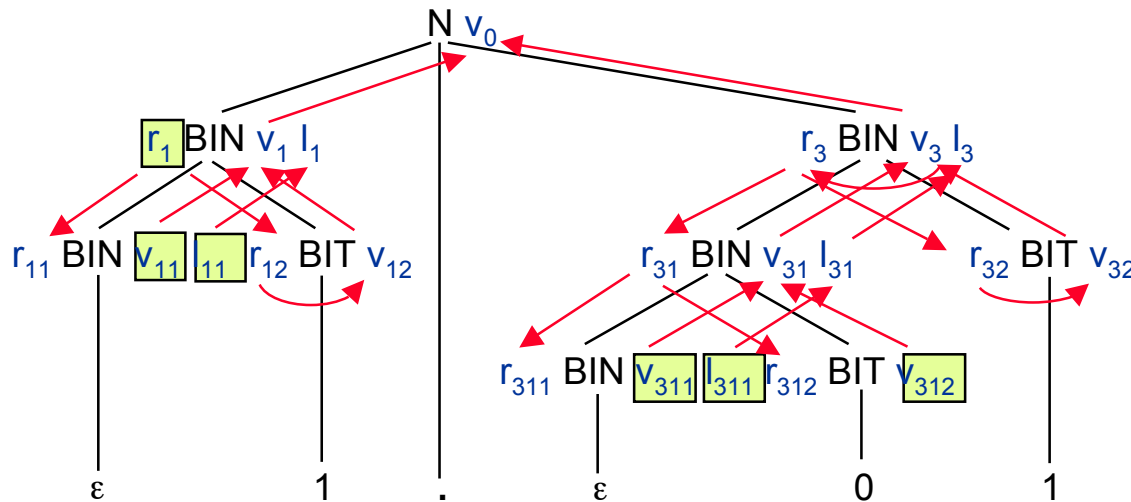
## Problem: Wie löst man ein solches Gleichungssystem?

- Annahme: Gleichungssystem azyklisch — *Ansonsten keine eindeutige Lösung, s.o.*

## Lösungsmöglichkeiten

- Dynamisch: Eliminationsverfahren — *Wie im letzten Beispiel*
  - Problem (in jedem Schritt):  
Suchen der nächsten zu eliminierenden Unbekannten
- Statisch
  - Analyse der *Abhängigkeiten zwischen Attributexemplaren* ➡ aufgrund der semantischen Regeln
  - Statisch bestimmte „Besuchsreihenfolge“
  - Attributauswerter, der diese Reihenfolge benutzt

## Beispiel (Abhängigkeiten zwischen Attributexemplaren)



## Unterschiedliche Abhängigkeiten

- Lokal: innerhalb von Produktionen
- Global: durch Anwendungskontext (Teilbaum oder oberes Baumfragment)

**Attribute grammar** Bin\_to\_Dec;  
**Nonterminals** N, BIN, BIT;  
**Attributes** **syn** l **with** BIN **domain** {int}  
                   **syn** v **with** N, BIN, BIT  
                           **domain** {real}  
                   **inh** r **with** BIN, BIT **domain** {int}

## Rules

1: $N \rightarrow \text{BIN.BIN}$	$N.v = \text{BIN}_1.v + \text{BIN}_3.v$ $\text{BIN}_1.r = 0$ $\text{BIN}_3.r = -\text{BIN}_3.l$
2: $\text{BIN} \rightarrow \text{BIN BIT}$	$\text{BIN}_0.v = \text{BIN}_1.v + \text{BIT.v}$ $\text{BIN}_0.l = \text{BIN}_1.l + 1$ $\text{BIN}_1.r = \text{BIN}_0.r + 1$ $\text{BIT.r} = \text{BIN}_0.r$
3: $\text{BIN} \rightarrow \epsilon$	$\text{BIN.v} = 0$ $\text{BIN.l} = 0$
4: $\text{BIT} \rightarrow 1$	$\text{BIT.v} = 2^{\text{BIT.r}}$
5: $\text{BIT} \rightarrow 0$	$\text{BIT.v} = 0$



## Produktionslokale Abhängigkeit

### Produktionslokale Abhängigkeitsrelation $Dp(p) \subseteq V(p) \times V(p)$

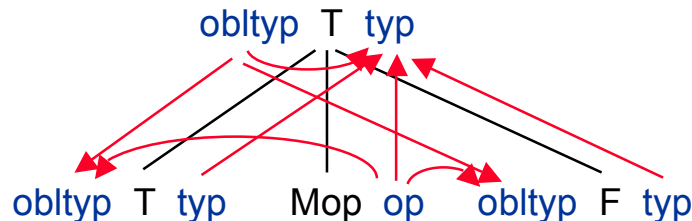
- Definiert durch  $b_j Dp(p) a_i \Leftrightarrow a_i = f_{p,a,i}(\dots, b_j, \dots)$
- Dargestellt durch produktionslokalen Abhängigkeitsgraphen

Menge aller  
Attributvorkommen in p

Knoten: Attributvorkommen in p  
(Gerichtete) Kanten: Abhängigkeiten

### Beispiel ( $AG_3$ )

- Geg.: Produktion 3':  $T \rightarrow T \text{ Mop } F$  mit
  - $T_0.\text{typ} = f_{31}(\text{Mop.op}, T_0.\text{obltyp}, T_1.\text{typ}, F.\text{typ})$
  - $T_1.\text{obltyp} = f_{32}(\text{Mop.op}, T_0.\text{obltyp})$
  - $F.\text{obltyp} = f_{32}(\text{Mop.op}, T_0.\text{obltyp})$
- $Dp(3')$



## Im folgenden vorausgesetzt: Attributgrammatiken in Normalform

- D.h. Argumente in den semantischen Regeln immer angewandte Vorkommen
  - $\Rightarrow Dp(p)$  nichtreflexiv (für alle p)
  - $\Rightarrow$  Graph ist nicht zyklisch und alle Wege haben Länge 1



## Beispiel (Attributgrammatik Bin\_to\_Dec)

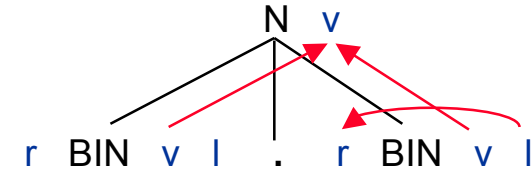
**Attribute grammar** Bin\_to\_Dec;  
**Nonterminals** N, BIN, BIT;  
**Attributes** **syn** l with BIN domain {int}  
                   **syn** v with N, BIN, BIT domain {real}  
                   **inh** r with BIN, BIT domain {int}

### Rules

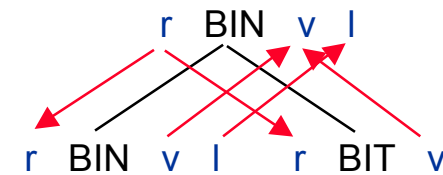
- |  |   |
|--|---|
| 1: $N \rightarrow \text{BIN}.\text{BIN}$           | $N.v = \text{BIN}_1.v + \text{BIN}_3.v$<br>$\text{BIN}_1.r = 0$<br>$\text{BIN}_3.r = -\text{BIN}_3.l$   |
| 2: $\text{BIN} \rightarrow \text{BIN} \text{ BIT}$ | $\text{BIN}_0.v = \text{BIN}_1.v + \text{BIT}.v$<br>$\text{BIN}_0.l = \text{BIN}_1.l + 1$<br>$\text{BIN}_1.r = \text{BIN}_0.r + 1$<br>$\text{BIT}.r = \text{BIN}_0.r$ |
| 3: $\text{BIN} \rightarrow \varepsilon$            | $\text{BIN}.v = 0$<br>$\text{BIN}.l = 0$  |
| 4: $\text{BIT} \rightarrow 1$                      | $\text{BIT}.v = 2^{\text{BIT}.r}$   |
| 5: $\text{BIT} \rightarrow 0$                      | $\text{BIT}.v = 0$  |

## Produktionslokale Abhängigkeitsgraphen Dp(i)

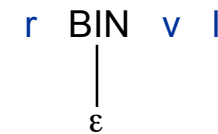
Dp(1)



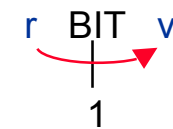
Dp(2)



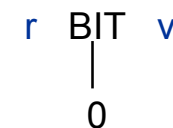
Dp(3)



Dp(4)



Dp(5)



## Es gilt

- Produktionslokale Abhängigkeiten induzieren Abhängigkeiten zwischen Attributexemplaren in Syntaxbäumen



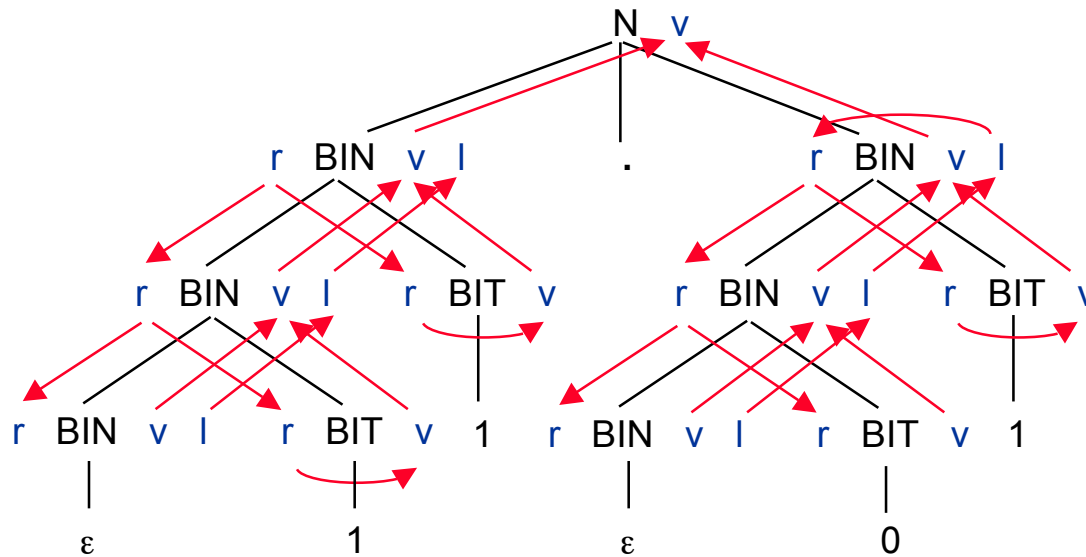
## Ziel

- Attributwerte für Syntaxbäume berechnen

## Individueller Abhängigkeitsgraph

- Geg.: Attributgrammatik AG mit zugrunde liegender kfG G und Syntaxbaum t gemäß G
- **Individueller Abhängigkeitsgraph** Dt(t) ergibt sich durch „Zusammenkleben“ der produktionslokalen Abhängigkeitsgraphen der angewandten Produktionen

## Beispiel (Attributgrammatik Bin\_to\_Dec)





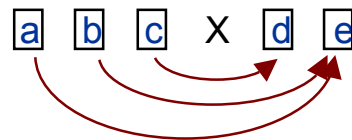
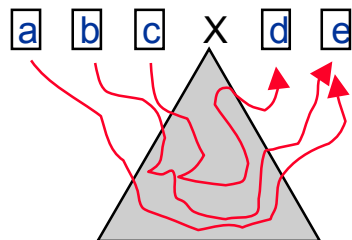
## Zyklenfrei

Individueller Abhängigkeitsgraph

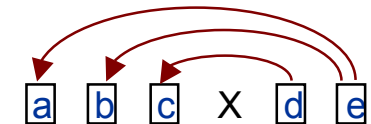
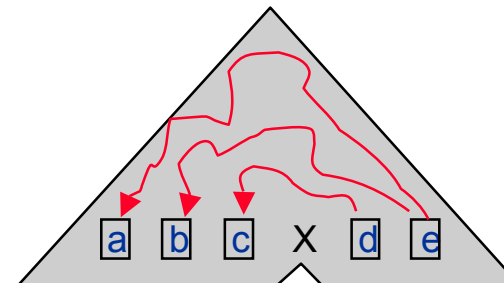
- Eine Attributgrammatik ist **zyklenfrei**, wenn  $Dt(t)$  für jeden Baum  $t$  der zugrunde liegenden kfG das zugehörige Gleichungssystem azyklisch ist
- D.h.: „wohlgeformt“ und „zyklenfrei“ sind quasi Synonyme

## Abhängigkeiten von Attributexemplaren

- Abgeleitete von den ererbten (in einem Teilbaum): *unterer charakteristischer Graph* ➡
- Ererbte von den abgeleiteten (in einem Baumfragment): *oberer charakteristischer Graph* ➡



unterer charakteristischer Graph



oberer charakteristischer Graph



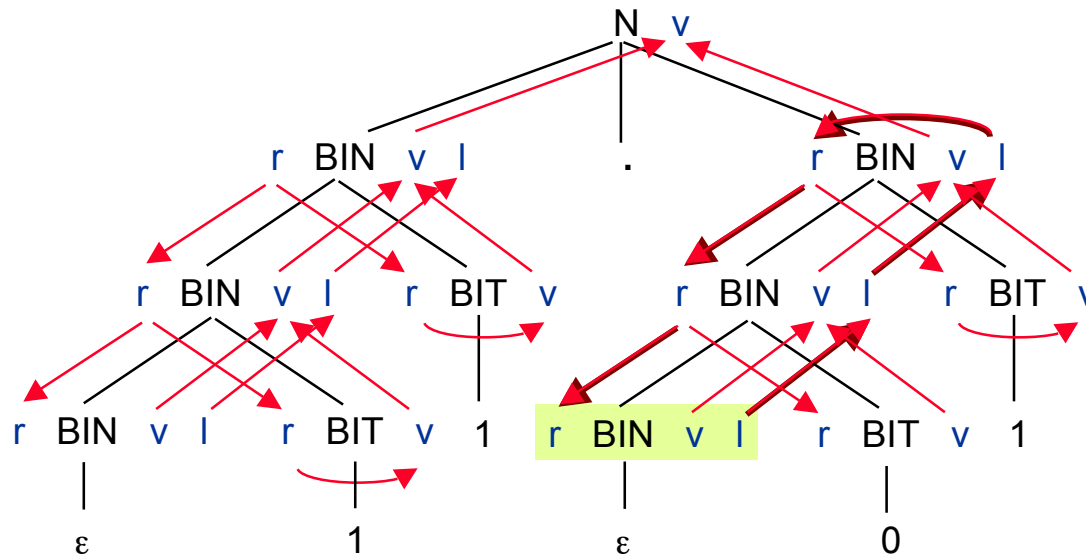


## Oberer charakteristischer Graph (für Nichtterminal X)

- Geg.: Baum  $t/n$  der zugrunde liegenden kfG mit Markierung  $X$  am Knoten  $n$ , bei dem der an  $n$  hängende Teilbaum entfernt wurde („oberes Baumfragment“)
- Von  $t$  induzierter **oberer charakteristischer Graph**
  - Graph der Relation  $Dt_{\downarrow t, n}(X) \subseteq \text{Syn}(X) \times \text{Inh}(X)$ 

Indirekter Zusammenhang abgeleitet  $\rightarrow$  ererbt
  - $Dt_{\downarrow t, n}(X) =_{\text{def}}$  Einschränkung der transitiven Hülle von  $Dt(t/n)$  auf die Attributexemplare von  $n$
- Effiziente Vorausberechnung (zur Generierungszeit) durch „Grammatikflussanalyse“

## Beispiel (Attributgrammatik Bin\_to\_Dec)



## Oberer charakteristischer Graph für BIN





## Abhängigkeiten zwischen Attributvorkommen

- Geg.: Knoten  $n$  (mit Produktion  $p$ ) in Baum  $t$
- **Abhängigkeitsrelation**  $ED_p(p)$  für  $p$  bei Vorkommen an  $n$  = Kombination von
  - Produktionslokaler Abhängigkeitsgraph  $D_p(p)$
  - Oberer charakteristischer Graph  $D_t \downarrow_{t,n}$  an  $n$
  - Untere charakteristische Graphen  $D_t \uparrow_t(X)$  an den Kindern von  $n$

## Es gilt

- Eine AG ist zyklensfrei  $\Leftrightarrow$   
alle Graphen  $ED_p(p)$  sind nichtzyklisch (für alle Produktionen  $p$ )
- Transitive Hülle jeder Abhängigkeitsrelation zu einer zyklensfreien Attributgrammatik ist eine partielle Ordnung — Folgt aus Azyklizität
- Diese partielle Ordnung kann zu einer totalen Ordnung gemacht werden (s.u.)

Z.B. durch „Topologisches Sortieren“



## Auswertungsordnung

*Legt Reihenfolge der Attributauswertung fest*

- **Auswertungsordnung einer Relation**  $R \subseteq V \times V =_{\text{def}} \text{totale Ordnung } T \text{ auf } V \text{ mit } R \subseteq T$
- **Auswertungsordnung für Baum  $t$**   $=_{\text{def}}$   
totale Ordnung  $T(t)$  auf der Menge  $V(t)$  aller Attributexemplare von  $t$  mit  $Dt(t) \subseteq T(t)$

## Beachte

*Individueller Abhängigkeitsgraph  
auf den Attributexemplaren von  $t$*

- Auswertungsordnung vom Baum  $t$  abhängig  
(da untere / obere charakteristische Graphen von  $t$  abhängig)
- Insgesamt aber nur endlich viele Ordnungen  
(da Zahl der Attribute zu jedem Symbol  $X$  endlich und fest)

*Allerdings „zu viele“, nämlich  
 $2^{nm}$  für jedes Symbol  $X$   
(wobei  $n = |\text{Inh}(X)|$  und  $m = |\text{Syn}(X)|$ )*

## Alternative: Approximative globale Abhängigkeiten

- „billigere“ Alternativen zu charakteristischen Graphen mit weniger oder mehr Kanten
- Probleme
  - Kanten weglassen: vergebliche Besuche von Unterbäumen
  - Kanten hinzufügen: evtl. Zyklen im Abhängigkeitsgraph (und damit ewiges Warten des Attributauswerter)

## Aufgabe der Attributauswertung

- Geg.: attributierter Baum = Syntaxbaum (der zugrunde liegenden kfG) erweitert um Speicherzellen für Attributexamplare  
(an jedem Knoten  $n$ : Verbund mit Komponenten für die Attribute von  $\text{symb}(n)$ )
- Ges.: Werte für Attributexamplare berechnen und (in den Verbunden an den Knoten) abspeichern

## Grundvoraussetzung

- Jeder Baum muss „Auswertungsordnung“ besitzen  $\Leftrightarrow$  AG ist zyklensfrei



(konzeptionelle) **Phasen der Attributauswertung** (in einem Syntaxbaum  $t$ )

- **Strategiephase**: Bestimmung einer Auswertungsordnung  $T(t)$ 
  - **Dynamisch** (alles zur Auswertungszeit)
  - **Statisch** (alles vorausberechnet)
  - **Selektion zwischen Besuchsfolgen** (teilweise statisch, teilweise dynamisch)
- **Auswertungsphase**: Attributauswertung gemäß  $T(t)$ 
  - **Bedarfsgetrieben** (demand-driven, „top-down“)
    - Auswerter wird mit Menge von Attributexemplaren aufgerufen, deren Werte erwünscht sind  
(maximale Elemente bezüglich Auswertungsordnung)
    - Dieser ruft dann (rekursiv) die Auswertung der benötigten Argumente auf
  - **Datengetrieben** (data-driven, „bottom-up“)
    - Attributauswertung beginnt mit der Auswertung der Attributexemplare, die von keinen anderen abhängen (minimale Elemente bezüglich Auswertungsordnung)
    - Exemplar wird immer erst dann ausgewertet, wenn alle Argumente bereits berechnet sind
- Für beide Phasen: Varianten (die spezielle Klassen von Auswertern festlegen)

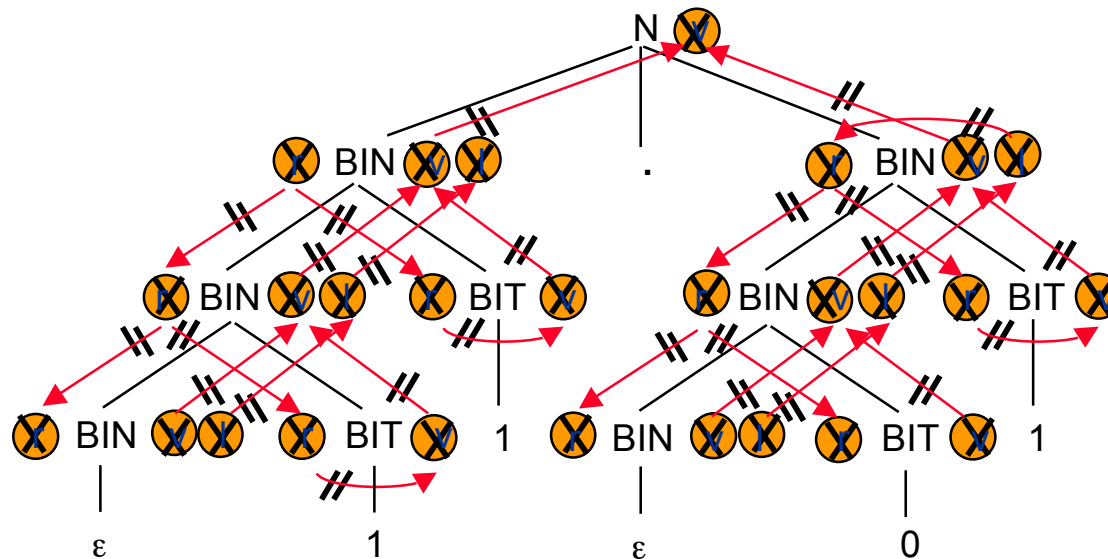
Können praktisch auch  
„verschmolzen“ sein

beginnt stets mit min./max. Elementen  
(= Quellen/Senken im Abhängigkeitsgraph)

## Dynamische Bestimmung der Auswertungsordnung

- **Topologisches Sortieren** des individuellen Abhängigkeitsgraphen  $Dt(t)$  ergibt totale Ordnung  $T(t)$  auf  $V(t)$
- Problem: platz- und zeitaufwendig

## Beispiel



$r_1, v_{111}, l_{111}, v_{311}, l_{311}, v_{312}, r_{11}, r_{12}, l_{11}, v_{31}, l_{31}, r_{111}, r_{112}, l_1, v_{12}, l_3, v_{111}, r_3, v_{11}, r_{31}, r_{32}, v_1, r_{311}, r_{312}, v_{32}, v_3, v_0$

## (statisch bestimmbare) **induzierte Auswertungsordnung**

- Bei spezieller Klasse von AGs (**l-geordnete Attributgrammatiken**) sind die individuellen Abhängigkeitsgraphen aller Bäume verträglich mit totalen Ordnungen auf den Attributmengen der Nichtterminale
- Ausgehend von diesen totalen Ordnungen auf den Attributmengen der Nichtterminale können „Besuchsfolgen“ (= totale Ordnungen auf den Mengen der Attributvorkommen für beliebige Produktionen) konstruiert werden
- Diese induzieren dann eindeutig eine Auswertungsordnung  $T(t)$  für jeden Baum  $t$  (d.h. zur Auswertungszeit: Strategiephase bereits erledigt)

## **Selektion zwischen Besuchsfolgen**

- Problem
  - Produktionen, denen in verschiedenen Baumkontexten verschiedene Besuchsfolgen zugeordnet werden müssen
- Lösung
  - Statisch: Berechnung der Menge aller Besuchsfolgen für jede Produktion
  - Dynamisch: Auswahl der richtigen Besuchsfolge für jede Anwendung einer Produktion

## Spezielle Attributgrammatikklassen

- Stark eingeschränkt
- Wegen Zeit- und Platzeffizienz trotzdem praktisch interessant

## Charakteristika

- Attributauswertung parallel zur Syntaxanalyse (parsergesteuert)
- Kellerartige Verwaltung der Attributwerte  
(damit Aufbau des Syntaxbaums zur späteren Attributauswertung überflüssig)

## L-attributierte Grammatiken

- Oberklasse aller Parser-gesteuert auswertbaren Attributgrammatiken  
(ererbte Attributvorkommen auf der rechten Seite einer Produktion hängen nur von Vorkommen ab, die „links von ihnen“ stehen)
- Enthält alle Grammatiken, in denen in jedem Syntaxbaum die Attribute in einem links-rechts-Tiefendurchlauf ausgewertet werden können
- Spezialfall: S-attribuiert = hat nur abgeleitete Attribute

## Beispiele

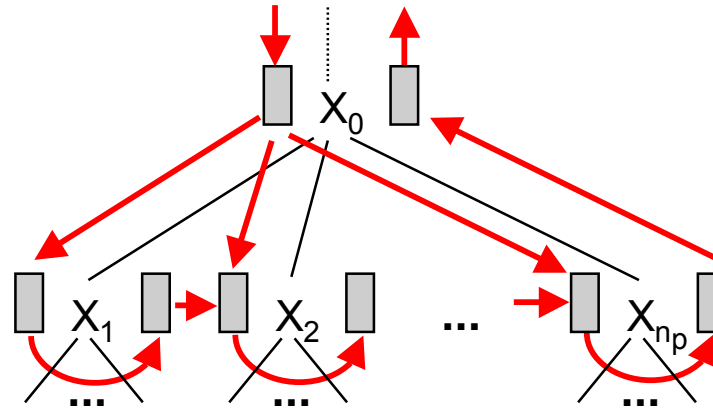
- $AG_1$  und  $AG_2$  ( $AG_1$  ist sogar S-attribuiert)

*Typberechnung für  
arithmetische Ausdrücke*

## Attributauswertung in einer L-attributierten Grammatik

```

program L-AA
proc visit (n: node)
begin
  case prod(n) of
    :
    p: begin
      eval(Inh(X1)); visit(n1);
      eval(Inh(X2)); visit(n2);
      :
      eval(Inh(Xnp)); visit(nnp);
      eval(Syn(X0));
    end;
  endcase
end;
begin
  visit(ε)
end.
  
```



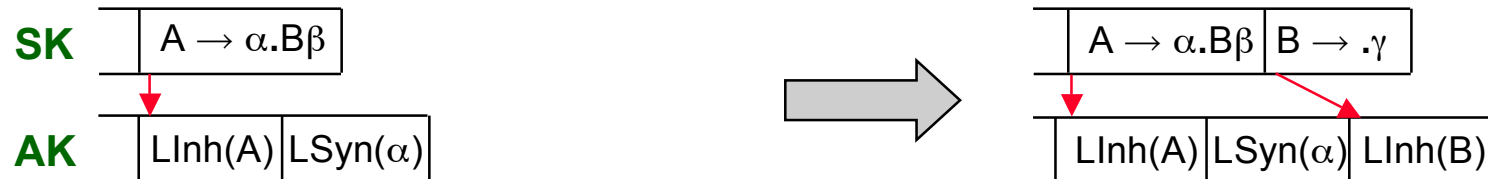
- Aktionen für Parser-gesteuerte Attributauswertung im LL-Parser
  - eval(Inh(X))     bei Beginn der Analyse eines Wortes für X (d.h. bei der Expansion)
  - eval(Syn(X))    bei Ende der Analyse für X (d.h. bei der Reduktion)
  - get(Syn(X))     beim Lesen eines Terminals X

## LL-attributierte Grammatik

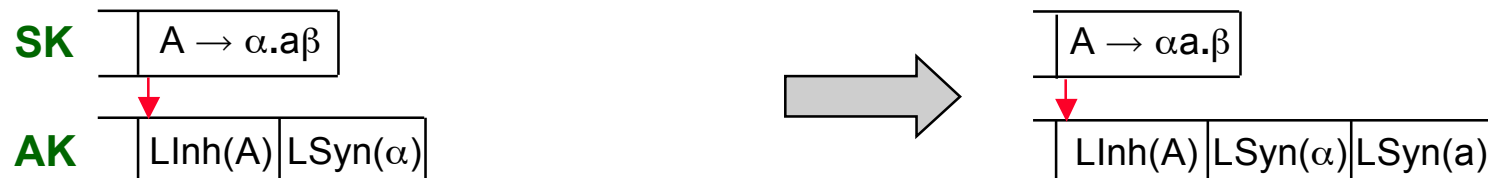
- AG ist **LL-attributiert**  $\Leftrightarrow$  AG ist L-attributiert + zugrunde liegende kfG ist LL-Grammatik

## Implementierung von LL-attributierten Grammatiken

- Expansion eines Nichtterminals B

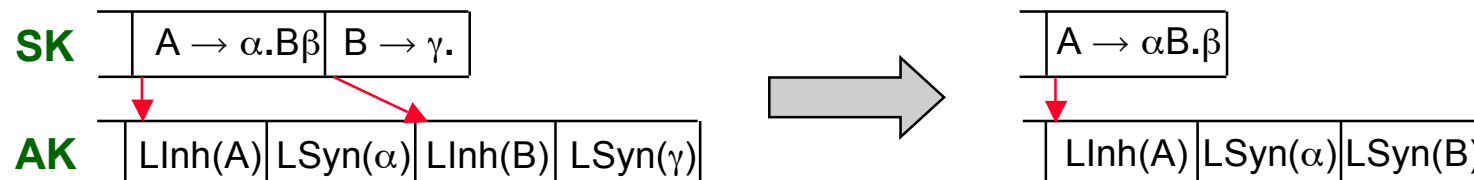


- Lesen eines Terminalsymbols a



*Linh/Lsyn bezeichnen die (geordnete) Liste der Inh/Syn-Attribute (mit relativer Adressierung) der einzelnen Attribute*

- Reduktion gemäß  $B \rightarrow \gamma$





## LR-attributierte Grammatiken

- Auswahl einer Produktion
  - LL: bei Expansion
  - LR: bei Reduktion
- Deshalb im LR-Parser
  - Attributauswertungsaktionen an Reduktionen + Leseaktionen des LR-Parsers koppeln
  - Damit verbunden: Kellern der Werte der abgeleiteten Attribute (im Attributkeller)
- AG ist **LR-attributiert**  $\Leftrightarrow$  AG ist L-attributiert + zugrunde liegende kfG ist LR-Grammatik

## Allgemeine Probleme (mit ererbten Attributen)

- Oberes Baumfragment, in dem die Transportwege für ererbte Attributwerte liegen, ist noch nicht bekannt (für L-attributierte Grammatik allerdings bekannt: alle Attribute, von denen ein ererbtes Attribut eines Nichtterminals abhängt, sind bereits berechnet)
- Bei Beginn der Analyse eines Wortes für ein Nichtterminal müssen die Werte seiner ererbten Attribute verfügbar gemacht werden; dieser Zeitpunkt kann vom LR-Parser evtl. nicht festgestellt werden — *Mehrere Items im betreffenden Zustand*
- Adressierung der Attribute positiv relativ zu einem Basiszeiger aus dem Syntaxkeller funktioniert nicht (aus demselben Grund) — *Parser weiß nicht, wann Basiszeiger gesetzt werden muss*



## Lösung: Grammatiktransformation

- Ziel: Berechnung ererbter Attribute an Reduktion koppeln
- Vorgehensweise
  - Vor Nichtterminale  $X_i$  (in den rechten Produktionsseiten) neues Nichtterminal  $N$  mit
    - Produktion  $N \rightarrow \varepsilon$
    - Attributregeln „ $LSyn(N) = Linh(X_i)$ “ einfügen
- Damit (bei Reduktion von  $\varepsilon$  zu  $N$ )
  - Berechnung der Attribute in  $Linh(X)$
  - Ablage der Werte oben auf dem Attributkeller

*Für jedes der jeweiligen Attribute*

## Verbleibende Probleme

- Zu viele neue Nichtterminale (Kellerplatzverbrauch zu hoch)
- Evtl. Verlust der LR-Eigenschaft

## (verbesserte) Lösung

- Nur dort  $\varepsilon$ -Nichtterminal einfügen, wo die Werte der ererbten Attribute eines Nichtterminals nicht das obere Ende des Attributkellers bilden
- Ist bereits ein Präfix der Liste der Werte (der ererbten Attribute) oben auf dem Attributkeller, dann nur soviel berechnen, dass Präfix zur ganzen Liste ergänzt wird



## In Bison

- Vordefinierte S-Attributierung (genau ein abgeleitetes Attribut zu jedem Grammatiksymbol)
- Grammatikregeln geben Produktionen und damit assoziierte semantische Aktionen an

**Beispiel** (Grammatik für variablenfreie, arithmetische Ausdrücke mit semantischen Regeln)

```
expr NUM          { $$ = $1 }
  | exp '+' exp    { $$ = $1 + $3; }
  | exp '-' exp    { $$ = $1 - $3; }
  | exp '*' exp    { $$ = $1 * $3; }
  | exp '/' exp    { $$ = $1 / $3; }
  | '-' exp %prec NEG { $$ = - $2; }
  | exp '^' exp    { $$ = pow($1, $3); }
  | '(' exp ')'    { $$ = $2; }
;
```

## Dabei

- $$$$ : Vorkommen des Attributs auf der linken Produktionsseite
- $$_i$ : Vorkommen des Attributs beim  $i$ -ten Symbol der rechten Seite