

... die zugehörige Tabelle (transponiert):

	0	1	2	3	4
<i>a</i>	1	4	4	4	4
<i>b</i>	4	2	4	4	4
<i>c</i>	4	4	3	4	4

### Beobachtung:

- Viele Einträge in der Tabelle sind **gleich** einem Wert **Default** (hier: **4**)
- Diesen Wert brauchen wir nicht zu repräsentieren **:-)**

... die zugehörige Tabelle (transponiert):

	0	1	2	3	4
<i>a</i>	1				
<i>b</i>		2			
<i>c</i>			3		

## Beobachtung:

- Viele Einträge in der Tabelle sind **gleich** einem Wert **Default** (hier: 4)
- Diesen Wert brauchen wir nicht zu repräsentieren :-)
- Dann legen wir einfach mehrere (transponierte) Spalten übereinander :-))

... im Beispiel:

	0	1	2
A	1	2	3
valid	a	b	c

- Feld **valid** teilt mit, für welches Element aus  $\Sigma$  der Eintrag gilt :-)
- **Achtung:** I.a. werden die Spalten nicht so perfekt übereinander passen!  
Dann verschieben wir sie so lange, bis die jeweils nächste in die bisherigen Löcher hineinpasst.
- Darum müssen wir ein zusätzliches Feld **displacement** verwalten, in dem wir uns die Verschiebung merken ;-)

Ein Feld-Zugriff  $\delta(j, a)$  wird dann so realisiert:

```
 $\delta(j, a) =$  let  $d = \text{displacement}[a]$   
in if ( $\text{valid}[d + j] \equiv a$ )  
    then  $A[d + j]$   
    else Default  
end
```

Ein Feld-Zugriff  $\delta(j, a)$  wird dann so realisiert:

```
 $\delta(j, a) =$  let  $d = \text{displacement}[a]$   
in if ( $\text{valid}[d + j] \equiv a$ )  
    then  $A[d + j]$   
    else Default  
end
```

## Diskussion:

- Die Tabellen werden i.a. erheblich kleiner.
- Dafür werden Tabellenzugriffe etwas teurer.

## 2 Die syntaktische Analyse



- Die syntaktische Analyse versucht, Tokens zu größeren Programmeinheiten zusammen zu fassen.

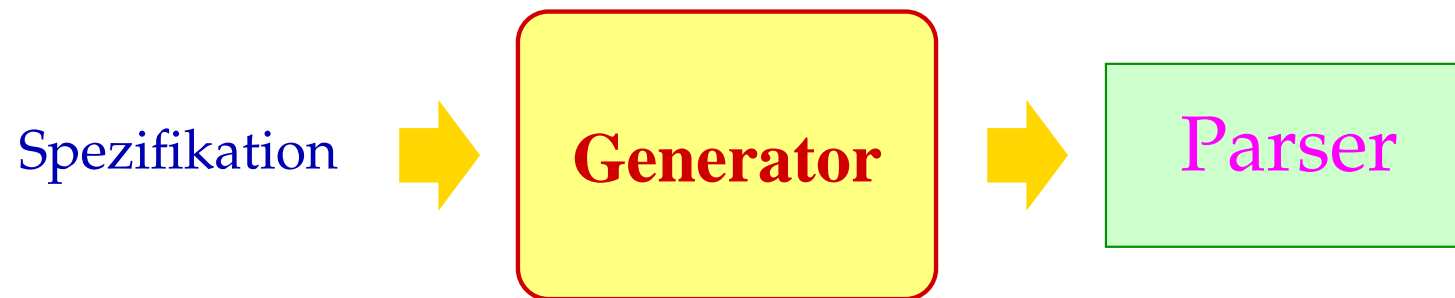
## 2 Die syntaktische Analyse



- Die syntaktische Analyse versucht, Tokens zu größeren Programmeinheiten zusammen zu fassen.
- Solche Einheiten können sein:
  - Ausdrücke;
  - Statements;
  - bedingte Verzweigungen;
  - Schleifen; ...

## Diskussion:

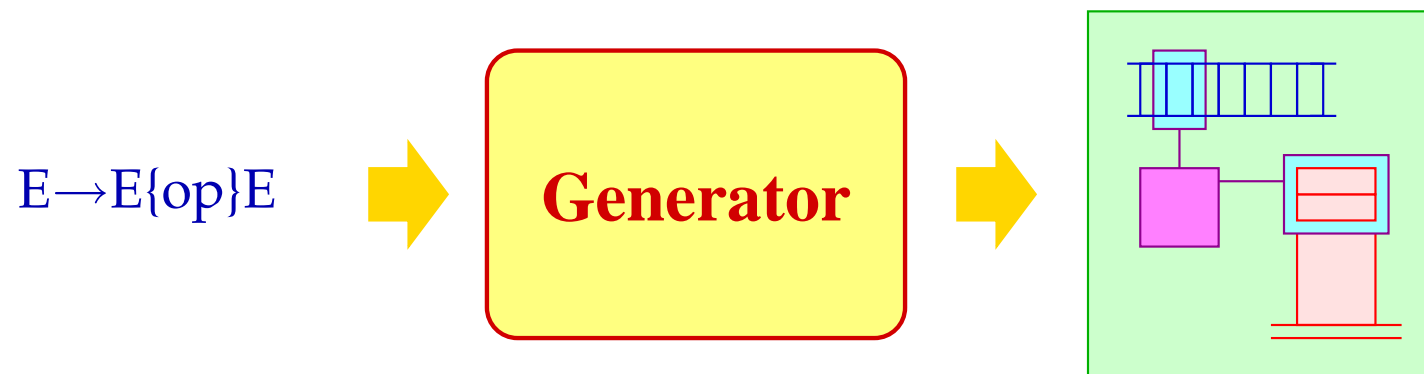
Auch Parser werden i.a. nicht von Hand programmiert, sondern aus einer Spezifikation **generiert**:





## Diskussion:

Auch Parser werden i.a. nicht von Hand programmiert, sondern aus einer Spezifikation **generiert**:



**Spezifikation der hierarchischen Struktur:**

kontextfreie Grammatiken;

**Generierte Implementierung:**

Kellerautomaten + X :-)

## 2.1 Grundlagen: Kontextfreie Grammatiken

- Programme einer Programmiersprache können unbeschränkt viele Tokens enthalten, aber nur endlich viele Token-Klassen :-)
- Als endliches Terminal-Alphabet  $T$  wählen wir darum die Menge der Token-Klassen.
- Die Schachtelung von Programm-Konstrukten lässt sich elegant mit Hilfe von **kontextfreien** Grammatiken beschreiben ...

## 2.1 Grundlagen: Kontextfreie Grammatiken

- Programme einer Programmiersprache können unbeschränkt viele Tokens enthalten, aber nur endlich viele Token-Klassen :-)
- Als endliches Terminal-Alphabet  $T$  wählen wir darum die Menge der Token-Klassen.
- Die Schachtelung von Programm-Konstrukten lässt sich elegant mit Hilfe von kontextfreien Grammatiken beschreiben ...

Eine kontextfreie Grammatik (CFG) ist ein 4-Tupel  $G = (N, T, P, S)$ , wobei:

- $N$  die Menge der Nichtterminale,
- $T$  die Menge der Terminale,
- $P$  die Menge der Produktionen oder Regeln, und
- $S \in N$  das Startsymbol ist.



Noam Chomsky, MIT (Guru)



John Backus, IBM (Erfinder von  
Fortran)

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \rightarrow \alpha \quad \text{mit} \quad A \in N, \quad \alpha \in (N \cup T)^*$$

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \rightarrow \alpha \quad \text{mit} \quad A \in N, \quad \alpha \in (N \cup T)^*$$

Beispiel:

$$S \rightarrow a S b$$

$$S \rightarrow \epsilon$$

Spezifizierte Sprache:  $\{a^n b^n \mid n \geq 0\}$

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \rightarrow \alpha \quad \text{mit} \quad A \in N, \quad \alpha \in (N \cup T)^*$$

Beispiel:

$$S \rightarrow a S b$$

$$S \rightarrow \epsilon$$

Spezifizierte Sprache:  $\{a^n b^n \mid n \geq 0\}$

Konventionen:

- In Beispielen ist die Spezifikation der Nichtterminale und Terminale i.a. **implizit**:
  - Nichtterminale sind:  $A, B, C, \dots, \langle \text{exp} \rangle, \langle \text{stmt} \rangle, \dots;$
  - Terminale sind:  $a, b, c, \dots, \text{int}, \text{name}, \dots;$

## Weitere Beispiele:

$S \rightarrow \langle \text{stmt} \rangle$   
 $\langle \text{stmt} \rangle \rightarrow \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle ;$   
 $\langle \text{if} \rangle \rightarrow \text{if} ( \langle \text{rexp} \rangle ) \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle$   
 $\langle \text{while} \rangle \rightarrow \text{while} ( \langle \text{rexp} \rangle ) \langle \text{stmt} \rangle$   
 $\langle \text{rexp} \rangle \rightarrow \text{int} \mid \langle \text{lexp} \rangle \mid \langle \text{lexp} \rangle = \langle \text{rexp} \rangle \mid \dots$   
 $\langle \text{lexp} \rangle \rightarrow \text{name} \mid \dots$



## Weitere Beispiele:

$S \rightarrow \langle \text{stmt} \rangle$   
 $\langle \text{stmt} \rangle \rightarrow \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle ;$   
 $\langle \text{if} \rangle \rightarrow \text{if} ( \langle \text{rexp} \rangle ) \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle$   
 $\langle \text{while} \rangle \rightarrow \text{while} ( \langle \text{rexp} \rangle ) \langle \text{stmt} \rangle$   
 $\langle \text{rexp} \rangle \rightarrow \text{int} \mid \langle \text{lexp} \rangle \mid \langle \text{lexp} \rangle = \langle \text{rexp} \rangle \mid \dots$   
 $\langle \text{lexp} \rangle \rightarrow \text{name} \mid \dots$

## Weitere Konventionen:

- Für jedes Nichtterminal sammeln wir die rechten Regelseiten und listen sie gemeinsam auf :-)
- Die  $j$ -te Regel für  $A$  können wir durch das Paar  $(A, j)$  bezeichnen ( $j \geq 0$ ).

## Weitere Grammatiken:

$E \rightarrow E+E \mid E * E \mid ( E ) \mid \text{name} \mid \text{int}$
$E \rightarrow E+T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow ( E ) \mid \text{name} \mid \text{int}$

Die beiden Grammatiken beschreiben die gleiche Sprache ;-)

## Weitere Grammatiken:

$E \rightarrow E + E^0 \mid E * E^1 \mid (E)^2 \mid \text{name}^3 \mid \text{int}^4$
$E \rightarrow E + T^0 \mid T^1$
$T \rightarrow T * F^0 \mid F^1$
$F \rightarrow (E)^0 \mid \text{name}^1 \mid \text{int}^2$

Die beiden Grammatiken beschreiben die gleiche Sprache ;-)

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

E

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\underline{E} \rightarrow \underline{E} + T$$

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \end{aligned}$$

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \\ &\rightarrow T * \underline{E} + T \end{aligned}$$



Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \\ &\rightarrow T * \underline{E} + T \\ &\rightarrow \underline{T} * \text{int} + T \end{aligned}$$

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \\ &\rightarrow T * \underline{E} + T \\ &\rightarrow \underline{T} * \text{int} + T \\ &\rightarrow \underline{E} * \text{int} + T \end{aligned}$$

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \\ &\rightarrow T * \underline{E} + T \\ &\rightarrow \underline{T} * \text{int} + T \\ &\rightarrow \underline{E} * \text{int} + T \\ &\rightarrow \text{name} * \text{int} + \underline{T} \end{aligned}$$

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \\ &\rightarrow T * \underline{F} + T \\ &\rightarrow \underline{T} * \text{int} + T \\ &\rightarrow \underline{F} * \text{int} + T \\ &\rightarrow \text{name} * \text{int} + \underline{T} \\ &\rightarrow \text{name} * \text{int} + \underline{F} \end{aligned}$$

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \\ &\rightarrow T * \underline{F} + T \\ &\rightarrow \underline{T} * \text{int} + T \\ &\rightarrow \underline{F} * \text{int} + T \\ &\rightarrow \text{name} * \text{int} + \underline{T} \\ &\rightarrow \text{name} * \text{int} + \underline{F} \\ &\rightarrow \text{name} * \text{int} + \text{int} \end{aligned}$$

Formal ist  $\rightarrow$  eine Relation auf Wörtern über  $V = N \cup T$ , wobei

$$\alpha \rightarrow \alpha' \text{ gdw. } \alpha = \alpha_1 A \alpha_2 \wedge \alpha' = \alpha_1 \beta \alpha_2 \text{ für ein } A \rightarrow \beta \in P$$

Den reflexiven und transitiven Abschluss von  $\rightarrow$  schreiben wir:  $\rightarrow^* \text{ :-)}$

Formal ist  $\rightarrow$  eine Relation auf Wörtern über  $V = N \cup T$ , wobei

$$\alpha \rightarrow \alpha' \text{ gdw. } \alpha = \alpha_1 A \alpha_2 \wedge \alpha' = \alpha_1 \beta \alpha_2 \text{ für ein } A \rightarrow \beta \in P$$

Den reflexiven und transitiven Abschluss von  $\rightarrow$  schreiben wir:  $\rightarrow^*$  :-)

## Bemerkungen:

- Die Relation  $\rightarrow$  hängt von der Grammatik ab ;-)
- Eine Folge von Ersetzungsschritten:  $\alpha_0 \rightarrow \dots \rightarrow \alpha_m$  heißt **Ableitung**.
- In jedem Schritt einer Ableitung können wir:
  - \* eine Stelle auswählen, **wo** wir ersetzen wollen, sowie
  - \* eine Regel, **wie** wir ersetzen wollen.
- Die von  $G$  spezifizierte Sprache ist:

$$\mathcal{L}(G) = \{w \in T^* \mid S \rightarrow^* w\}$$

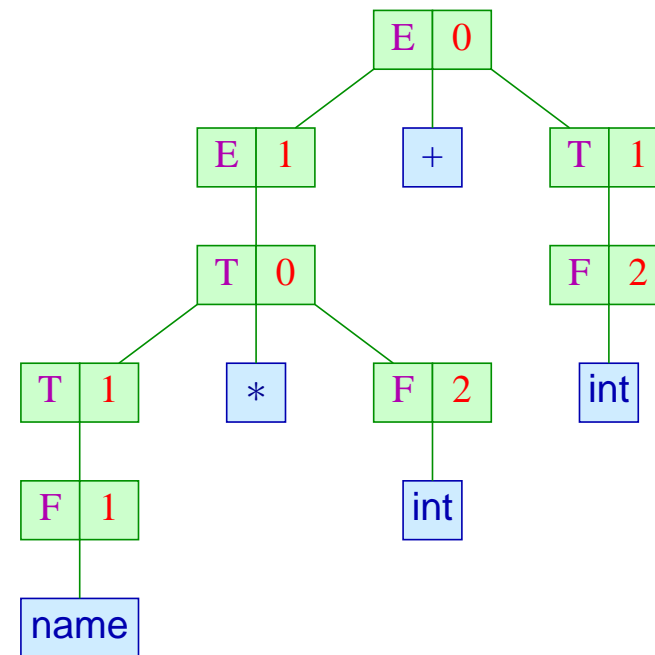
## Achtung:

Die Reihenfolge, in der disjunkte Teile abgeleitet werden, ist unerheblich :-)

Ableitungen eines Symbols stellt man als **Ableitungsbaum** dar :-)

... im Beispiel:

$\underline{E} \rightarrow^0 \underline{E} + T$   
 $\rightarrow^1 \underline{T} + T$   
 $\rightarrow^0 T * \underline{E} + T$   
 $\rightarrow^2 \underline{T} * \text{int} + T$   
 $\rightarrow^1 \underline{E} * \text{int} + T$   
 $\rightarrow^1 \text{name} * \text{int} + \underline{T}$   
 $\rightarrow^1 \text{name} * \text{int} + \underline{E}$   
 $\rightarrow^2 \text{name} * \text{int} + \text{int}$





Ein Ableitungsbaum für  $A \in N$ :

**innere Knoten:** Regel-Anwendungen;

**Wurzel:** Regel-Anwendung für  $A$ ;

**Blätter:** Terminale oder  $\epsilon$ ;

Die Nachfolger von  $(B, i)$  entsprechen der rechten Seite der Regel  $\text{: -)}$

Ein Ableitungsbaum für  $A \in N$ :

**innere Knoten:** Regel-Anwendungen;

**Wurzel:** Regel-Anwendung für  $A$ ;

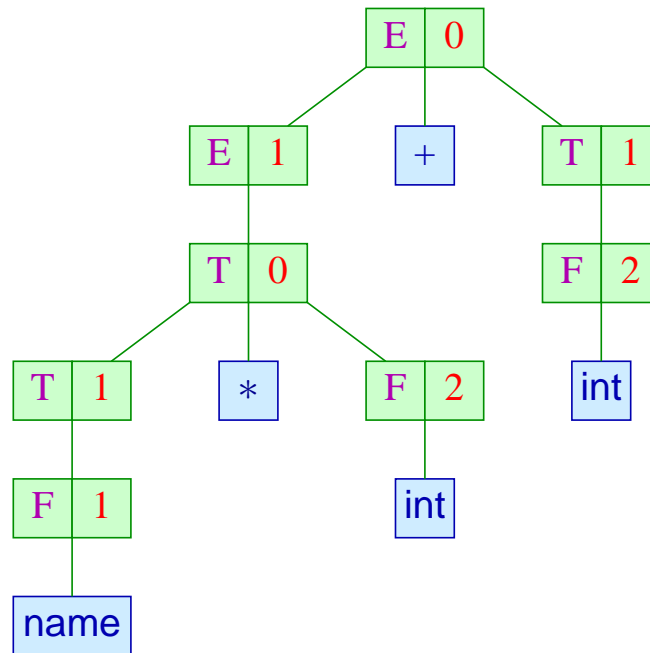
**Blätter:** Terminale oder  $\epsilon$ ;

Die Nachfolger von  $(B, i)$  entsprechen der rechten Seite der Regel  $\rightarrow$

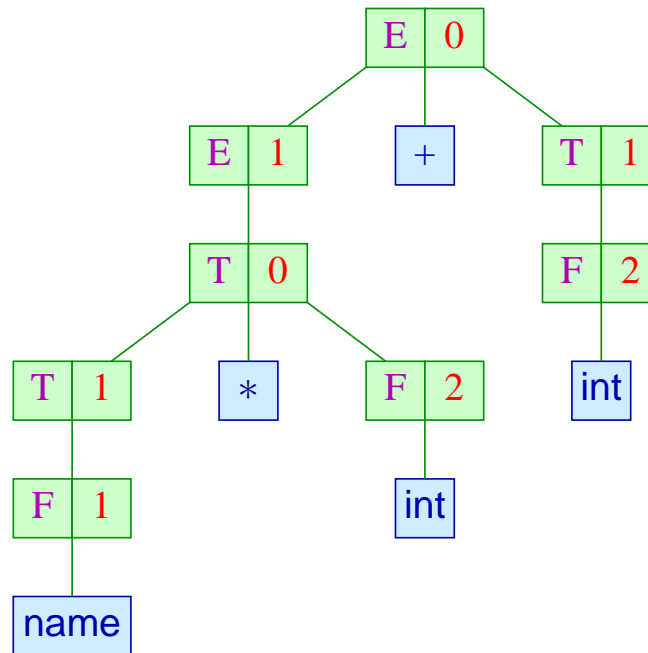
## Beachte:

- Neben beliebiger Ableitungen betrachtet man solche, bei denen stets das **linkste** (bzw. **rechtste**) Vorkommen eines Nichtterminals ersetzt wird.
- Diese heißen **Links-** (bzw. **Rechts-**) Ableitungen und werden durch Index  $L$  bzw.  $R$  gekennzeichnet.
- Links-(bzw. Rechts-) Ableitungen entsprechen einem links-rechts (bzw. rechts-links) **preorder**-DFS-Durchlauf durch den Ableitungsbaum  $\rightarrow$
- **Reverse** Rechts-Ableitungen entsprechen einem links-rechts **postorder**-DFS-Durchlauf durch den Ableitungsbaum  $\rightarrow$

... im Beispiel:



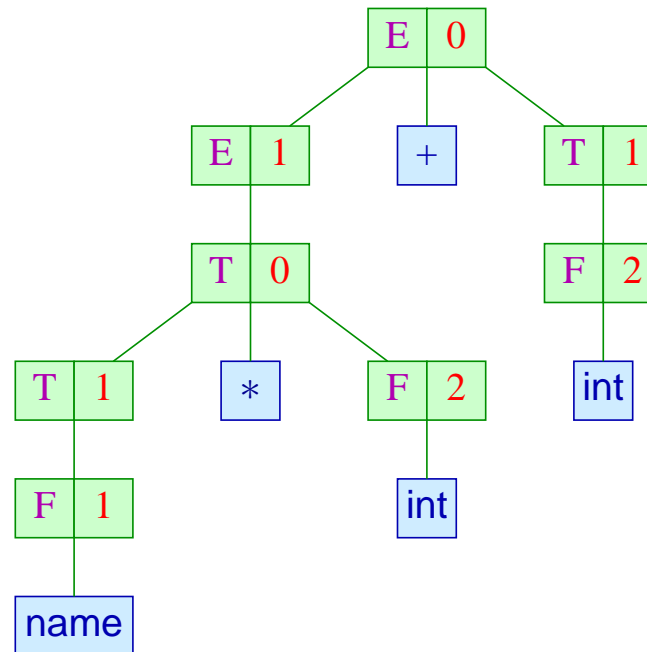
... im Beispiel:



Links-Ableitung:

$(E, 0) (E, 1) (T, 0) (T, 1) (F, 1) (F, 2) (T, 1) (F, 2)$

... im Beispiel:



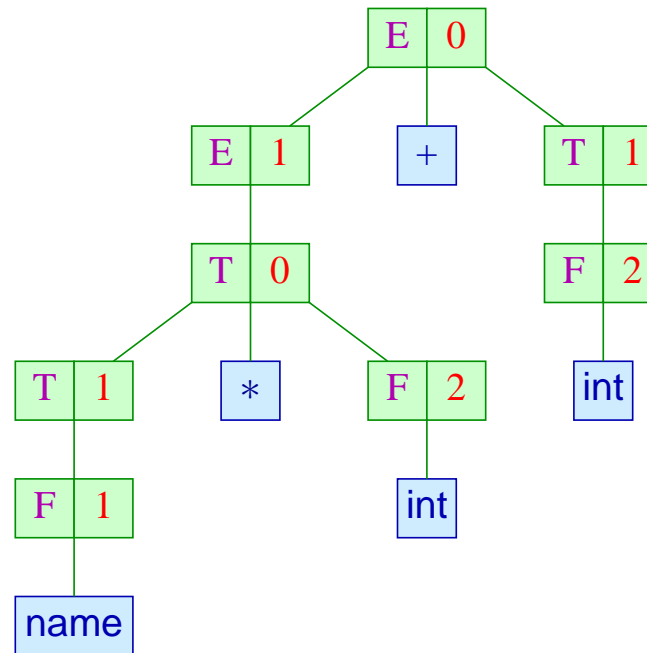
Links-Ableitung:

$(E, 0) (E, 1) (T, 0) (T, 1) (F, 1) (F, 2) (T, 1) (F, 2)$

Rechts-Ableitung:

$(E, 0) (T, 1) (F, 2) (E, 1) (T, 0) (F, 2) (T, 1) (F, 1)$

... im Beispiel:



Links-Ableitung:

$(E, 0) (E, 1) (T, 0) (T, 1) (F, 1) (F, 2) (T, 1) (F, 2)$

Rechts-Ableitung:

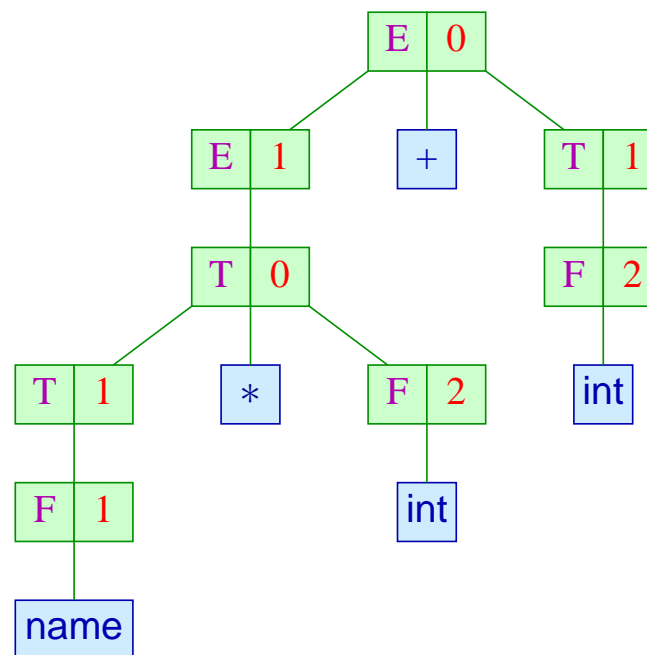
$(E, 0) (T, 1) (F, 2) (E, 1) (T, 0) (F, 2) (T, 1) (F, 1)$

Reverse Rechts-Ableitung:

$(F, 1) (T, 1) (F, 2) (T, 0) (E, 1) (F, 2) (T, 1) (E, 0)$

Die Konkatenation der Blätter des Ableitungsbaums  $t$  bezeichnen wir auch mit  $\text{yield}(t)$ .

... im Beispiel:



liefert die Konkatenation:

`name * int + int.`

Die Grammatik  $G$  heißt **eindeutig**, falls es zu jedem  $w \in T^*$  maximal einen Ableitungsbaum  $t$  von  $S$  gibt mit  $\text{yield}(t) = w$  :-)

... unsere beiden Grammatiken:

$E \rightarrow E + E^0$		$E * E^1$		$(E)^2$		$\text{name}^3$		$\text{int}^4$
$E \rightarrow E + T^0$		$T^1$						
$T \rightarrow T * F^0$		$F^1$						
$F \rightarrow (E)^0$		$\text{name}^1$		$\text{int}^2$				

Die zweite ist eindeutig, die erste nicht :-)



## Fazit:

- Ein Ableitungsbaum repräsentiert eine mögliche hierarchische Struktur eines Worts.
- Bei Programmiersprachen sind wir nur an Grammatiken interessiert, bei denen die Struktur stets eindeutig ist :-)
- Ableitungsbäume stehen in eins-zu-eins-Korrespondenz mit Links-Ableitungen wie auch (reversen) Rechts-Ableitungen.
- Links-Ableitungen entsprechen einem Topdown-Aufbau des Ableitungsbaums.
- Reverse Rechts-Ableitungen entsprechen einem Bottom-up-Aufbau des Ableitungsbaums.

## Fingerübung:

überflüssige Nichtterminale und Regeln

$A \in N$  heißt **produktiv**, falls  $A \rightarrow^* w$  für ein  $w \in T^*$ .

$A \in N$  heißt **erreichbar**, falls  $S \rightarrow^* \alpha A \beta$  für geeignete  $\alpha, \beta \in (T \cup N)^*$ .

## Beispiel:

$S \rightarrow a B B \mid b D$

$A \rightarrow B c$

$B \rightarrow S d \mid C$

$C \rightarrow a$

$D \rightarrow B D$

## Fingerübung:

überflüssige Nichtterminale und Regeln

$A \in N$  heißt **produktiv**, falls  $A \rightarrow^* w$  für ein  $w \in T^*$ .

$A \in N$  heißt **erreichbar**, falls  $S \rightarrow^* \alpha A \beta$  für geeignete  $\alpha, \beta \in (T \cup N)^*$ .

## Beispiel:

$S \rightarrow a B B \mid b D$

$A \rightarrow B c$

$B \rightarrow S d \mid C$

$C \rightarrow a$

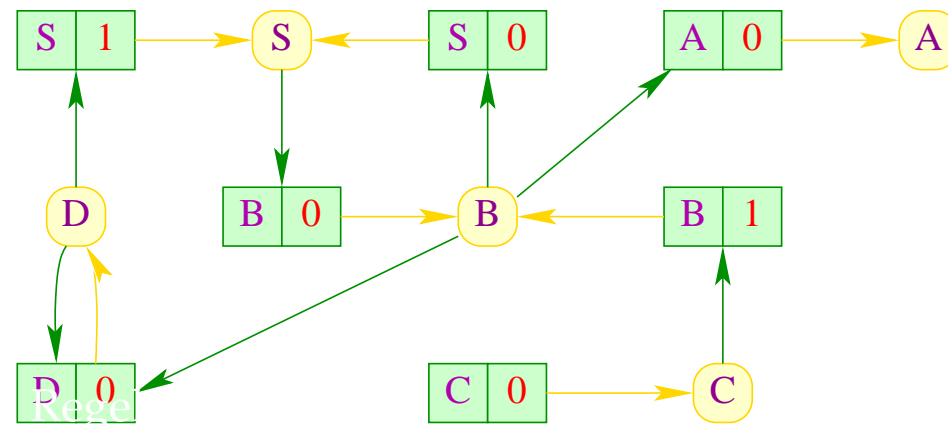
$D \rightarrow B D$

**Produktive Nichtterminale:**  $S, A, B, C$

**Erreichbare Nichtterminale:**  $S, B, C, D$

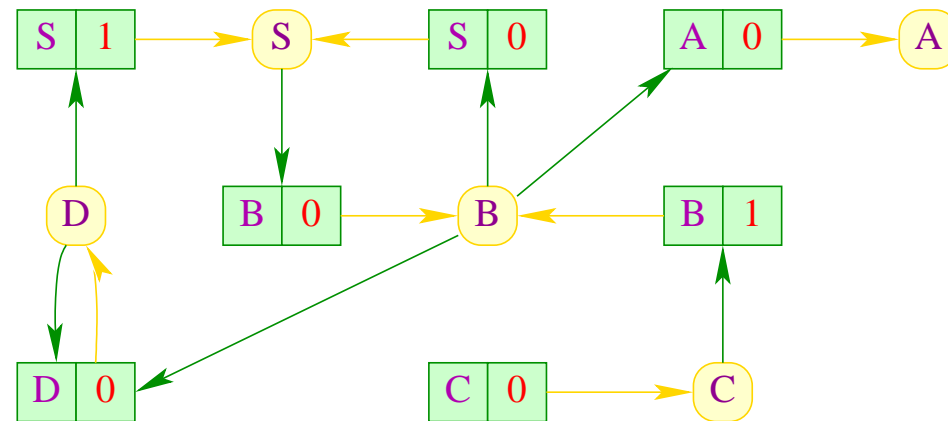
Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



# Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



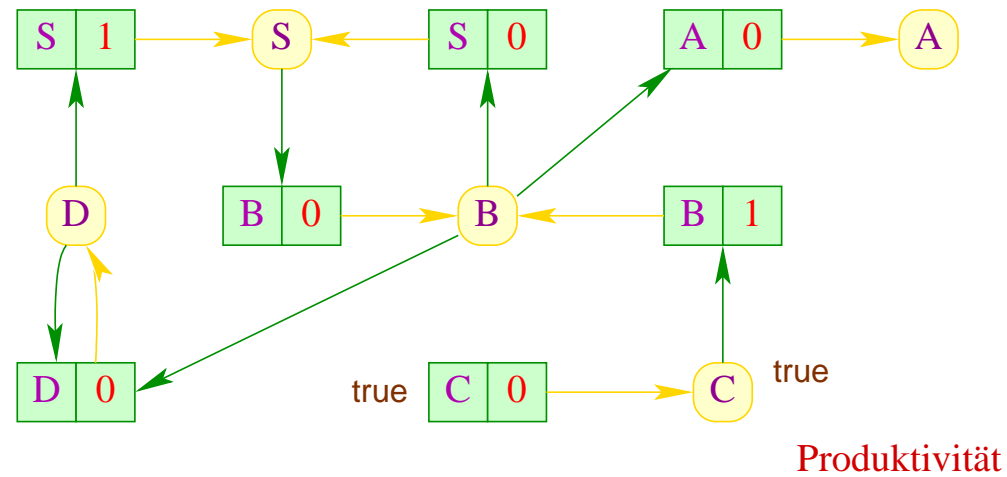
**And-Knoten:** Regeln

**Or-Knoten:** Nichtterminale

**Kanten:**  $((B, i), B)$  für alle Regeln  $(B, i)$   
 $(A, (B, i))$  falls  $(B, i) \equiv B \rightarrow \alpha_1 A \alpha_2$

# Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



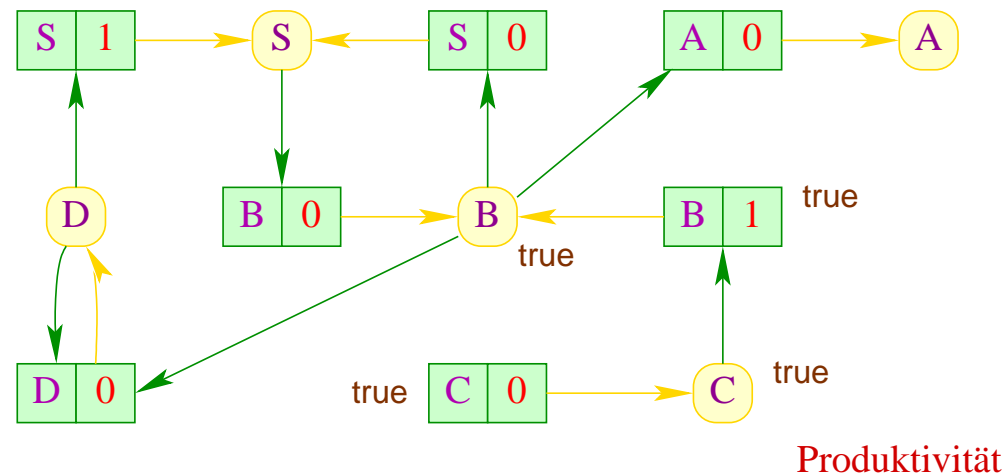
**And-Knoten:** Regeln

**Or-Knoten:** Nichtterminale

**Kanten:**  $((B, i), B)$  für alle Regeln  $(B, i)$   
 $(A, (B, i))$  falls  $(B, i) \equiv B \rightarrow \alpha_1 A \alpha_2$

# Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



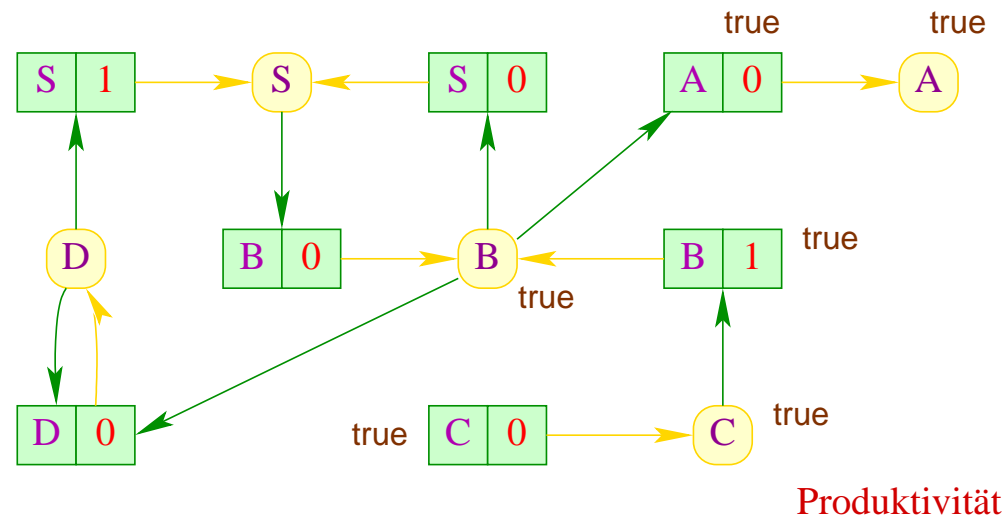
**And-Knoten:** Regeln

**Or-Knoten:** Nichtterminale

**Kanten:**  $((B, i), B)$  für alle Regeln  $(B, i)$   
 $(A, (B, i))$  falls  $(B, i) \equiv B \rightarrow \alpha_1 A \alpha_2$

# Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



**And-Knoten:** Regeln

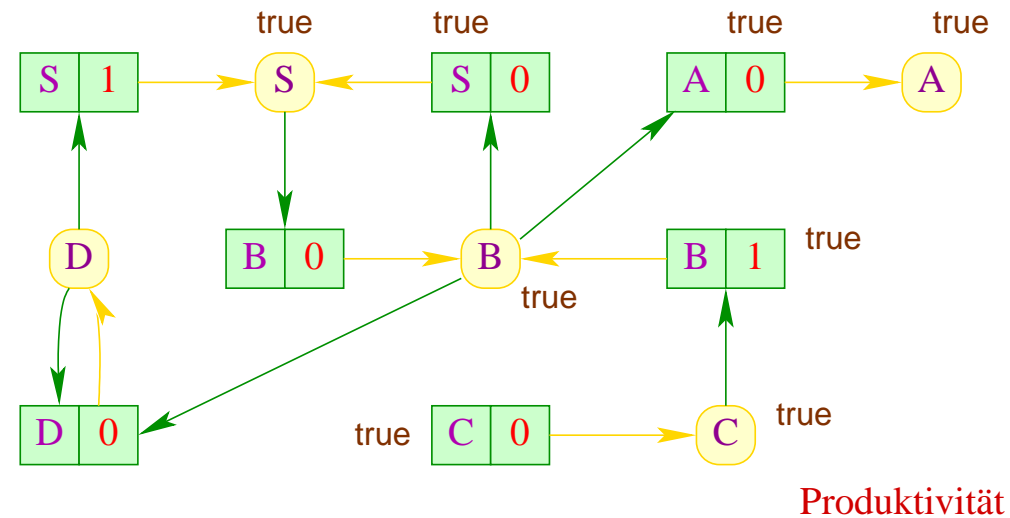
**Or-Knoten:** Nichtterminale

**Kanten:**  $((B, i), B)$  für alle Regeln  $(B, i)$   
 $(A, (B, i))$  falls  $(B, i) \equiv B \rightarrow \alpha_1 A \alpha_2$



# Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



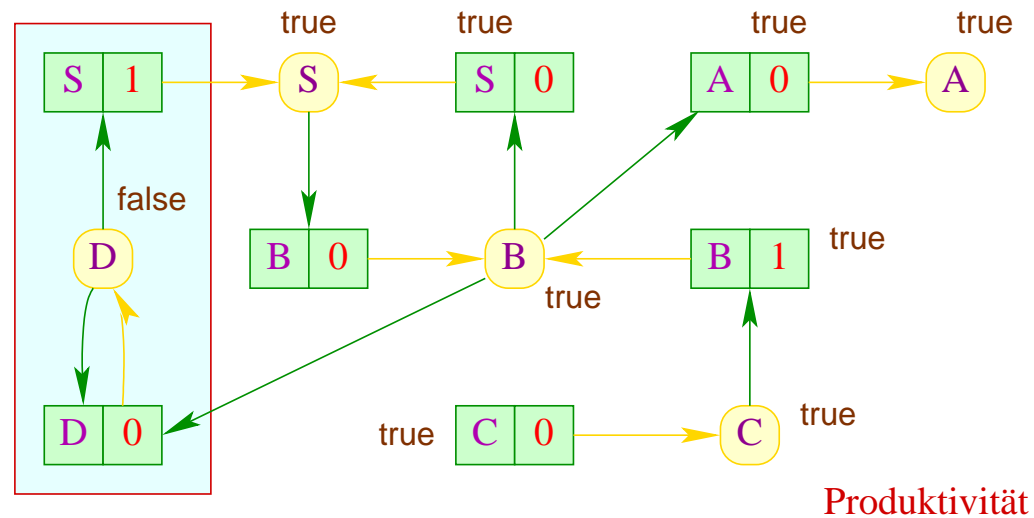
**And-Knoten:** Regeln

**Or-Knoten:** Nichtterminale

**Kanten:**  $((B, i), B)$  für alle Regeln  $(B, i)$   
 $(A, (B, i))$  falls  $(B, i) \equiv B \rightarrow \alpha_1 A \alpha_2$

# Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



**And-Knoten:** Regeln

**Or-Knoten:** Nichtterminale

**Kanten:**  $((B, i), B)$  für alle Regeln  $(B, i)$   
 $(A, (B, i))$  falls  $(B, i) \equiv B \rightarrow \alpha_1 A \alpha_2$