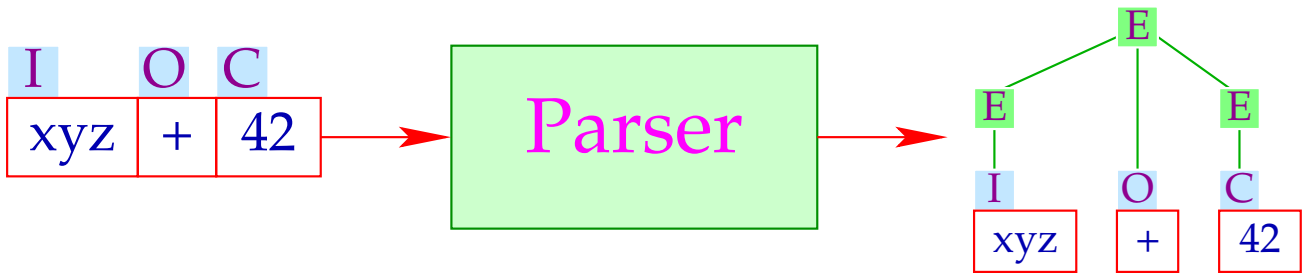
2 Die syntaktische Analyse

- 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:

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

Die Regeln kontextfreier Grammatiken sind von der Form:

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

Beispiel:

$$\begin{aligned} S &\to a\,S\,b \\ S &\to \epsilon \end{aligned}$$

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

Die Regeln kontextfreier Grammatiken sind von der Form:

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

Beispiel:

$$\begin{aligned} S &\to a\,S\,b \\ S &\to \epsilon \end{aligned}$$

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: $A, B, C, \ldots, \langle \mathsf{exp} \rangle, \langle \mathsf{stmt} \rangle, \ldots$;
  - → Terminale sind: $a, b, c, \ldots, \mathsf{int}, \mathsf{name}, \ldots$;

Weitere Beispiele:

$$
\begin{array}{lcl}
S & \rightarrow & \langle \text{stmt} \rangle \\
\langle \text{stmt} \rangle & \rightarrow & \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle ; \\
\langle \text{if} \rangle & \rightarrow & \text{if (} \langle \text{rexp} \rangle \text{) } \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle \\
\langle \text{while} \rangle & \rightarrow & \text{while (} \langle \text{rexp} \rangle \text{) } \langle \text{stmt} \rangle \\
\langle \text{rexp} \rangle & \rightarrow & \text{int} \mid \langle \text{lexp} \rangle \mid \langle \text{lexp} \rangle = \langle \text{rexp} \rangle \mid \ldots \\
\langle \text{lexp} \rangle & \rightarrow & \text{name} \mid \ldots
\end{array}
$$

Weitere Beispiele:

$$
\begin{array}{lcl}
S & \rightarrow & \langle \text{stmt} \rangle \\
\langle \text{stmt} \rangle & \rightarrow & \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle ; \\
\langle \text{if} \rangle & \rightarrow & \text{if } (\langle \text{rexp} \rangle) \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle \\
\langle \text{while} \rangle & \rightarrow & \text{while } (\langle \text{rexp} \rangle) \langle \text{stmt} \rangle \\
\langle \text{rexp} \rangle & \rightarrow & \text{int} \mid \langle \text{lexp} \rangle \mid \langle \text{lexp} \rangle = \langle \text{rexp} \rangle \mid \ldots \\
\langle \text{lexp} \rangle & \rightarrow & \text{name} \mid \ldots
\end{array}
$$

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	$\rightarrow$	$E+E$	$\mid$	$E*E$	$\mid$	$(\ E\ )$	$\mid$	name	$\mid$	int

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

Die beiden Grammatiken beschreiben die gleiche Sprache ;-)

Weitere Grammatiken:

$$E \rightarrow E+E\,^{0} \mid E*E\,^{1} \mid (\ E\ )\,^{2} \mid \mathsf{name}\,^{3} \mid \mathsf{int}\,^{4}$$

$$\begin{aligned} E &\rightarrow E+T\,^{0} \mid T\,^{1} \\ T &\rightarrow T*F\,^{0} \mid F\,^{1} \\ F &\rightarrow (\ E\ )\,^{0} \mid \mathsf{name}\,^{1} \mid \mathsf{int}\,^{2} \end{aligned}$$

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:

$$\underline{E}$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

... im letzten Beispiel:

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

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:

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

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

... im letzten Beispiel:

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

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

... im letzten Beispiel:

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

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

... im letzten Beispiel:

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

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

... im letzten Beispiel:

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

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

... im letzten Beispiel:

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

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

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

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

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

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

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

Bemerkungen:

- Die Relation $\rightarrow$ hängt von der Grammatik ab ;-)
- Eine Folge von Ersetzungsschritten: $\alpha_0 \rightarrow \ldots \rightarrow \alpha_m$ 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:

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

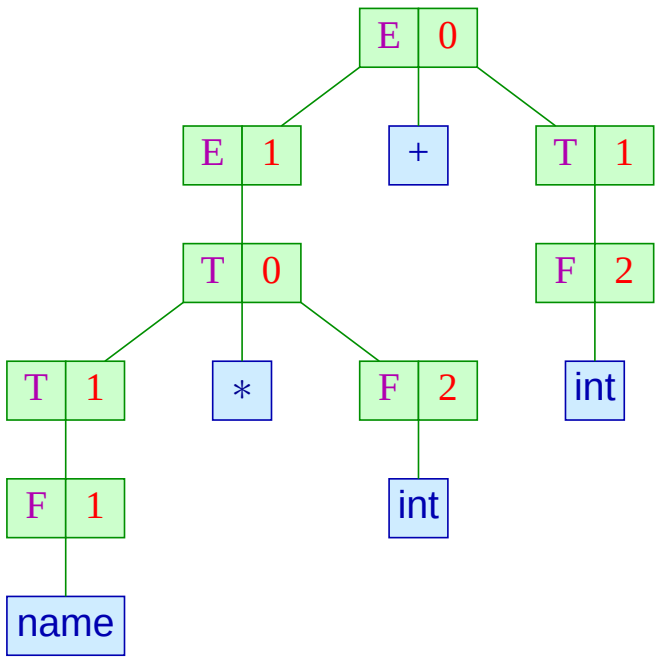
Achtung:

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

Ableitungen eines Symbols stellt man als Ableitungsbaum dar :-)

... im Beispiel:

$$
\begin{aligned}
\underline{E} &\rightarrow^{0} \underline{E} + T \\
&\rightarrow^{1} \underline{T} + T \\
&\rightarrow^{0} T * \underline{F} + T \\
&\rightarrow^{2} \underline{T} * \mathsf{int} + T \\
&\rightarrow^{1} \underline{F} * \mathsf{int} + T \\
&\rightarrow^{1} \mathsf{name} * \mathsf{int} + \underline{T} \\
&\rightarrow^{1} \mathsf{name} * \mathsf{int} + \underline{F} \\
&\rightarrow^{2} \mathsf{name} * \mathsf{int} + \mathsf{int}
\end{aligned}
$$



Ein Ableitungsbaum für $A \in N$:

innere Knoten:	Regel-Anwendungen;
Wurzel:	Regel-Anwendung für A;
Blätter:	Terminale oder ϵ;

Die Nachfolger von (B, i) entsprechen der rechten Seite der Regel :-)

Ein Ableitungsbaum für $A \in N$:

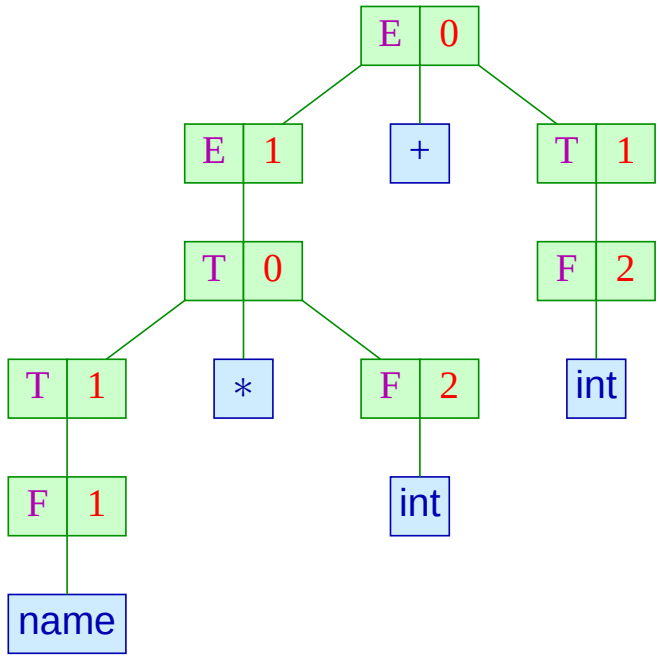
innere Knoten:	Regel-Anwendungen;
Wurzel:	Regel-Anwendung für A;
Blätter:	Terminale oder ϵ;

Die Nachfolger von (B, i) entsprechen der rechten Seite der Regel :-)

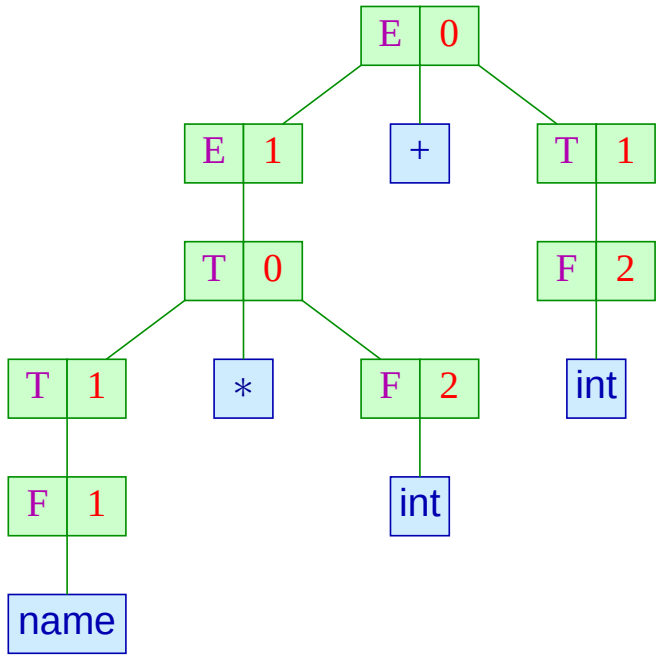
Beachte:

- Neben beliebiger Ableitungen betrachtet man solche, bei denen stets das linkste (bzw. rechteste) 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 :-)
- Reverse Rechts-Ableitungen entsprechen einem links-rechts postorder-DFS-Durchlauf durch den Ableitungsbaum :-))

... im Beispiel:

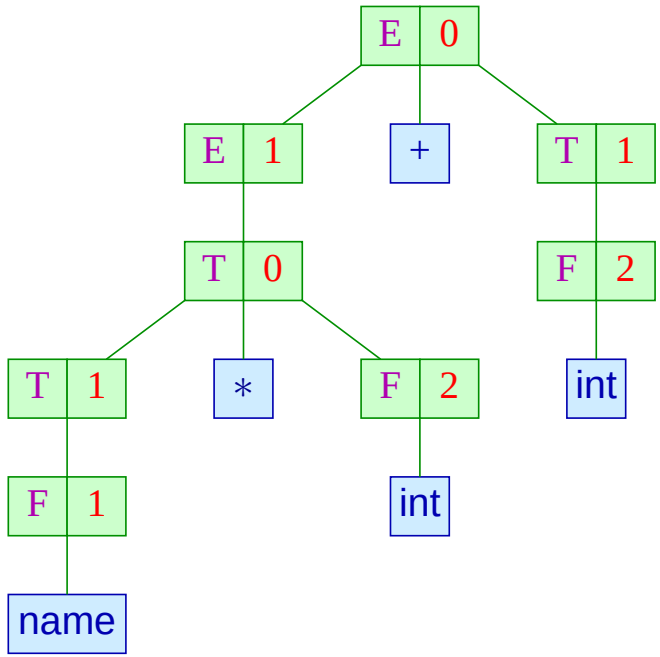


... im Beispiel:



Links-Ableitung: $(E,0)\,(E,1)\,(T,0)\,(T,1)\,(F,1)\,(F,2)\,(T,1)\,(F,2)$

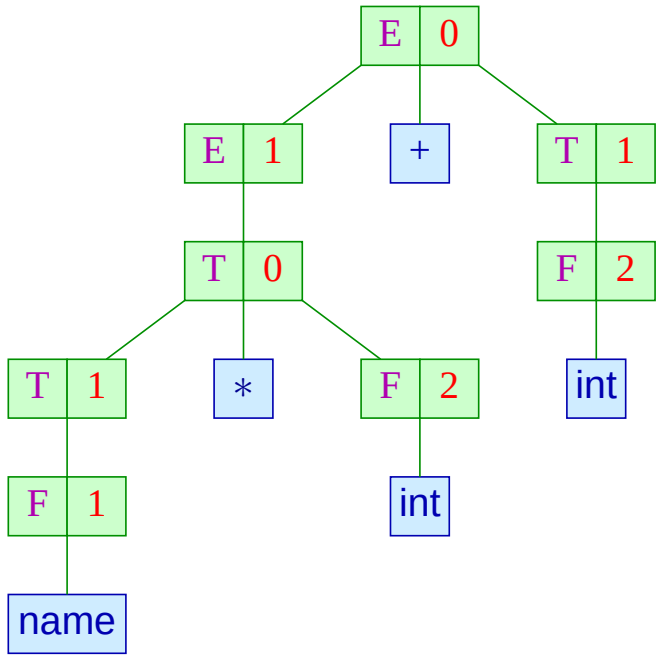
... im Beispiel:



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:



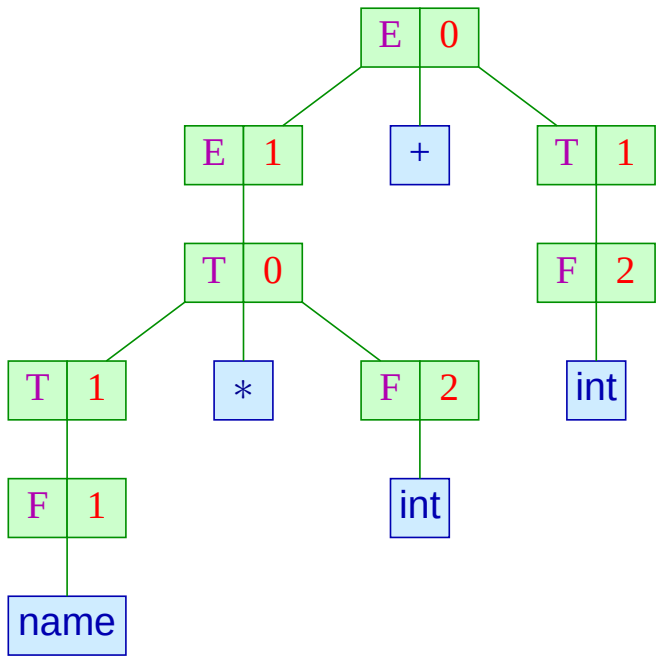
Links-Ableitung: $(E,0)\,(E,1)\,(T,0)\,(T,1)\,(F,1)\,(F,2)\,(T,1)\,(F,2)$

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 *brownt* bezeichnen wir auch mit $\mathsf{yield}(t)$.

... im Beispiel:



liefert die Konkatenation: $\mathsf{name} * \mathsf{int} + \mathsf{int}$.

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

... unsere beiden Grammatiken:

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

E	$\rightarrow$	$E+T$	^{0}	$\mid$	T	^{1}				
T	$\rightarrow$	$T*F$	^{0}	$\mid$	F	^{1}				
F	$\rightarrow$	$(\ E\ )$	^{0}	$\mid$	name	^{1}	$\mid$	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 $A \to^* w$ für ein $w \in T^*$.

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

Beispiel:

$$
\begin{array}{lcl}
S & \to & a\,B\,B \mid b\,D \\
A & \to & B\,c \\
B & \to & S\,d \mid C \\
C & \to & a \\
D & \to & B\,D
\end{array}
$$

Fingerübung: überflüssige Nichtterminale und Regeln

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

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

Beispiel:

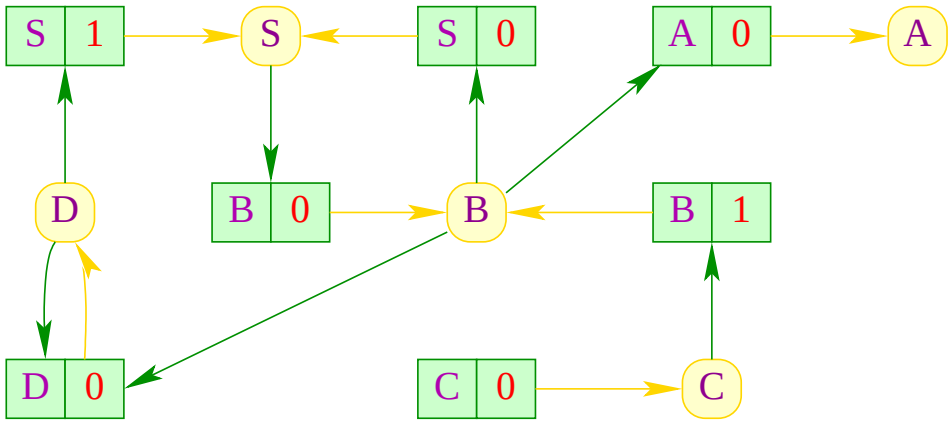
$$
\begin{array}{lcl}
S & \to & a\,B\,B \mid b\,D \\
A & \to & B\,c \\
B & \to & S\,d \mid C \\
C & \to & a \\
D & \to & B\,D
\end{array}
$$

Produktive Nichtterminale: S, A, B, C

Erreichbare Nichtterminale: S, B, C, D

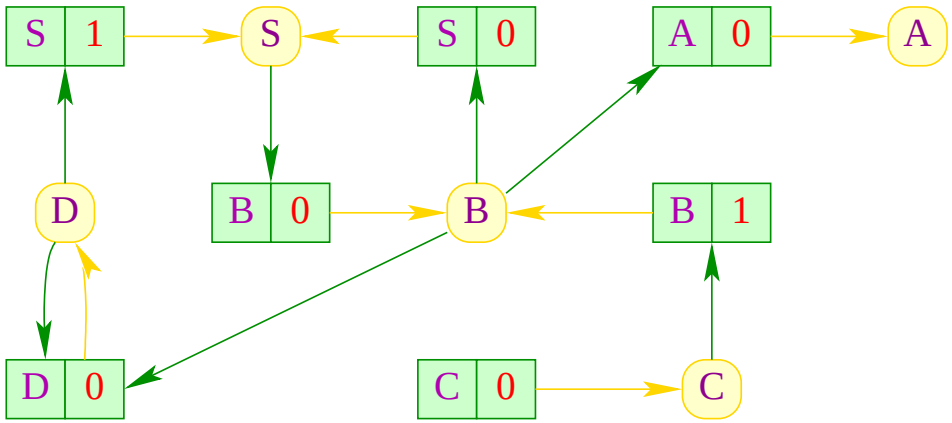
Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



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

... hier:



And-Knoten: Regeln

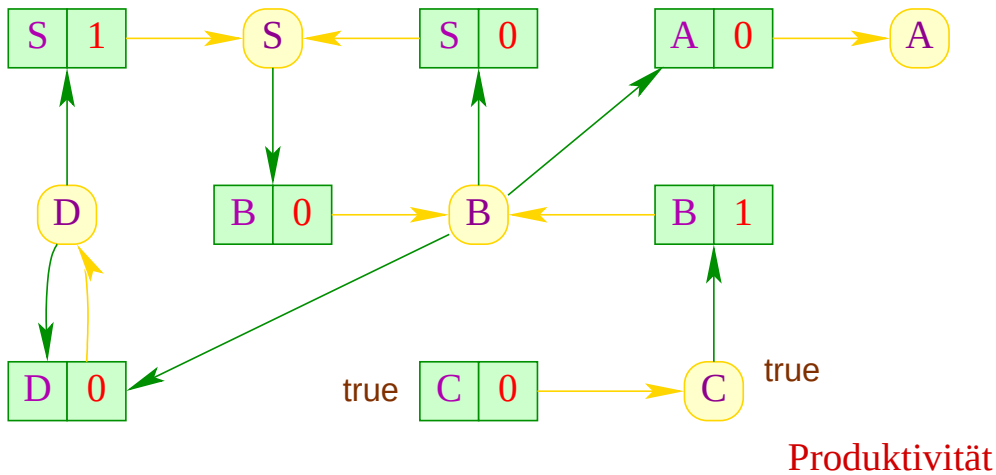
Or-Knoten: Nichtterminale

Kanten: $((B, i), B)$ für alle Regeln (B, i)

$(A, (B, i))$ falls $(B, i) \equiv B \to \alpha_1\, A\, \alpha_2$

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

... hier:



And-Knoten: Regeln

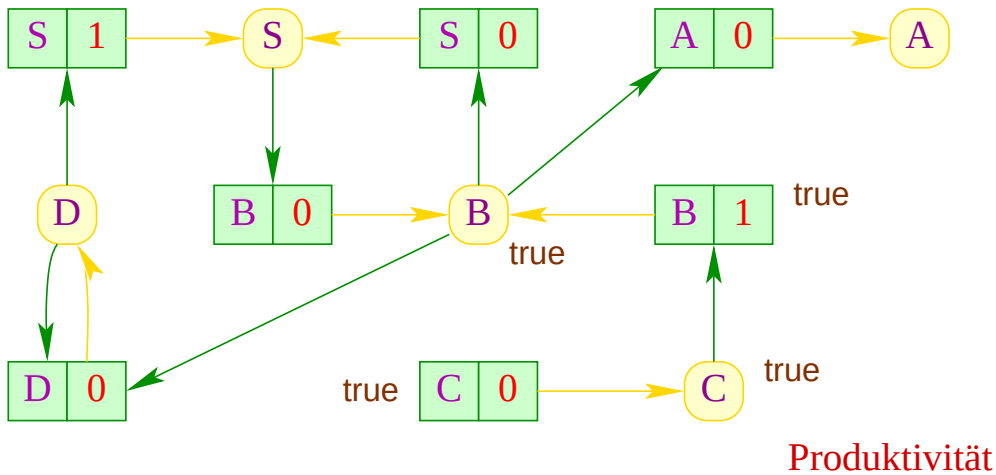
Or-Knoten: Nichtterminale

Kanten: $((B,i),B)$ für alle Regeln (B,i)

$(A,(B,i))$ falls $(B,i) \equiv B \to \alpha_1\, A\, \alpha_2$

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

... hier:



And-Knoten: Regeln

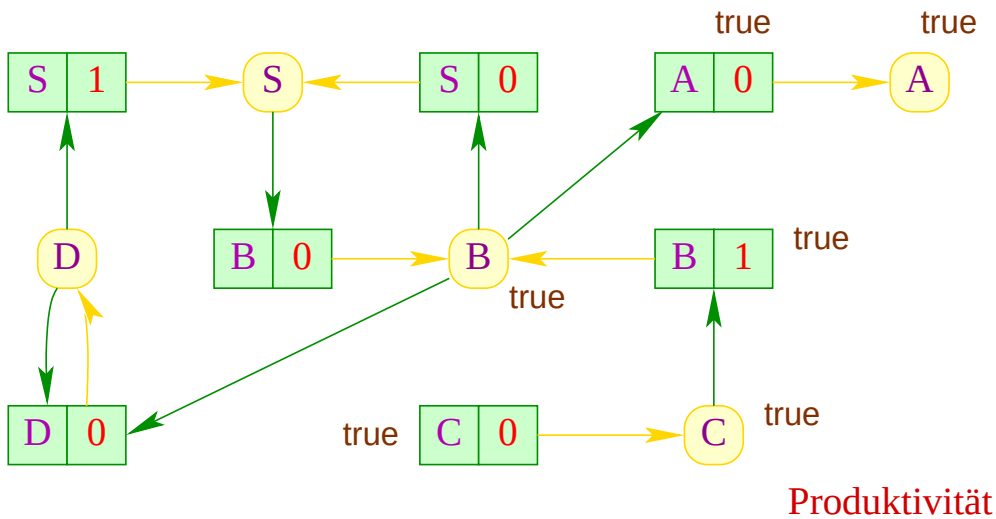
Or-Knoten: Nichtterminale

Kanten: $((B,i),B)$ für alle Regeln (B,i)

$(A,(B,i))$ falls $(B,i) \equiv B \to \alpha_1\, A\, \alpha_2$

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

... hier:



And-Knoten: Regeln

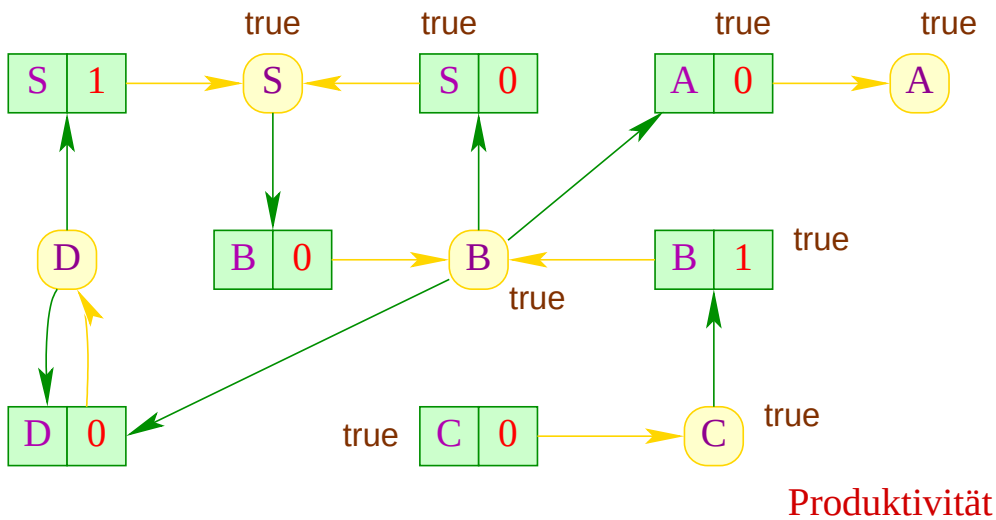
Or-Knoten: Nichtterminale

Kanten: $((B,i),B)$ für alle Regeln (B,i)

$(A,(B,i))$ falls $(B,i) \equiv B \to \alpha_1\, A\, \alpha_2$

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

... hier:



And-Knoten: Regeln

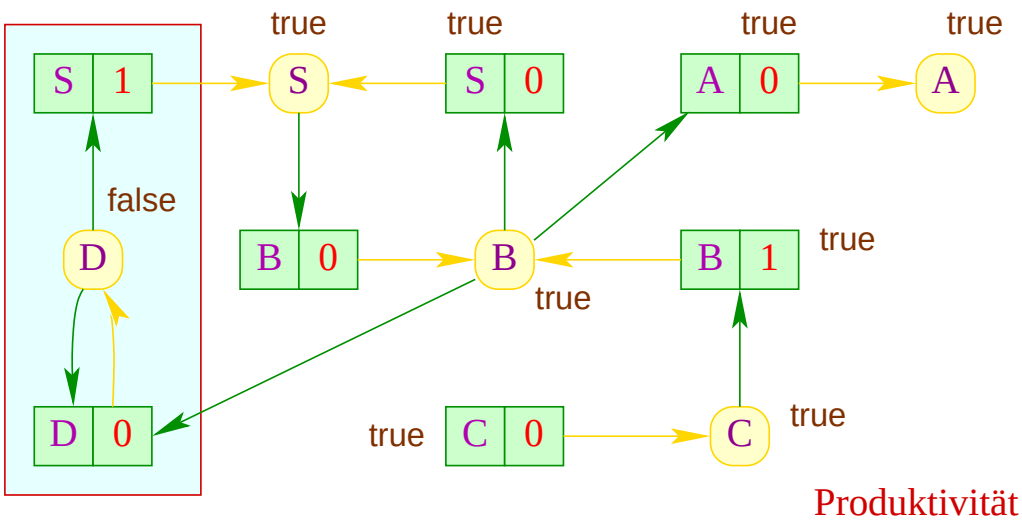
Or-Knoten: Nichtterminale

Kanten: $((B,i),B)$ für alle Regeln (B,i)

$(A,(B,i))$ falls $(B,i) \equiv B \to \alpha_1\, A\, \alpha_2$

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 $(B,i) \equiv B \to \alpha_1\, A\, \alpha_2$

Algorithmus:

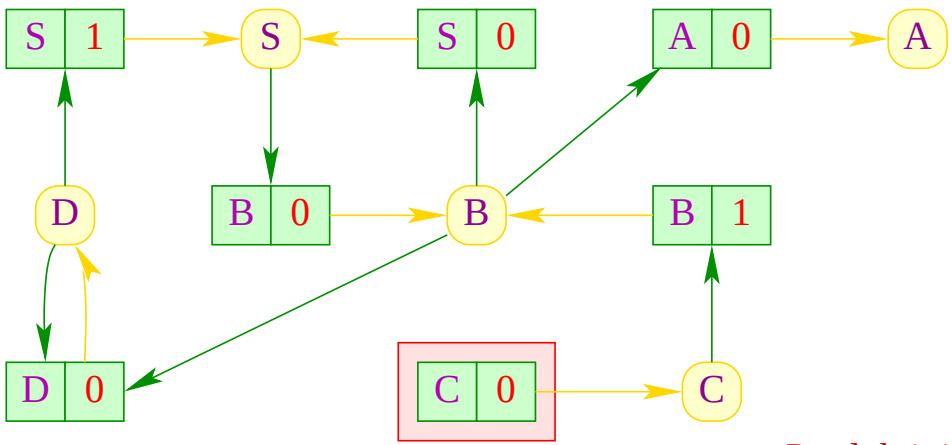
```
2^N  result = ∅;                    //  Ergebnis-Menge
int  count[P];                      //  Zähler für jede Regel
2^P  rhs[N];                        //  Vorkommen in rechten Seiten

forall (A ∈ N)   rhs[A] = ∅;        //  Initialisierung
forall ((A,i) ∈ P)   {              //
                 count[(A,i)] = 0;  //
                 init(A,i);         //  Initialisierung von rhs
                 }                  //
...                                 //
```

Die Hilfsfunktion init zählt die Nichtterminal-Vorkommen in der rechten Seite und vermerkt sie in der Datenstruktur rhs :-)

```
...                                                          //
2^P   W = {r | count[r] = 0};                                //   Workset
while (W ≠ ∅)   {                                            //
        (A, i) = extract(W);                                 //
        if (A ∉ result)   {                                  //
                result = result ∪ {A};                       //
                forall (r ∈ rhs[A])   {                      //
                        count[r]--;                          //
                        if (count[r] == 0)   W = W ∪ {r};    //
                        }                                    //   end of   forall
                }                                            //   end of   if
        }                                                    //   end of   while
```

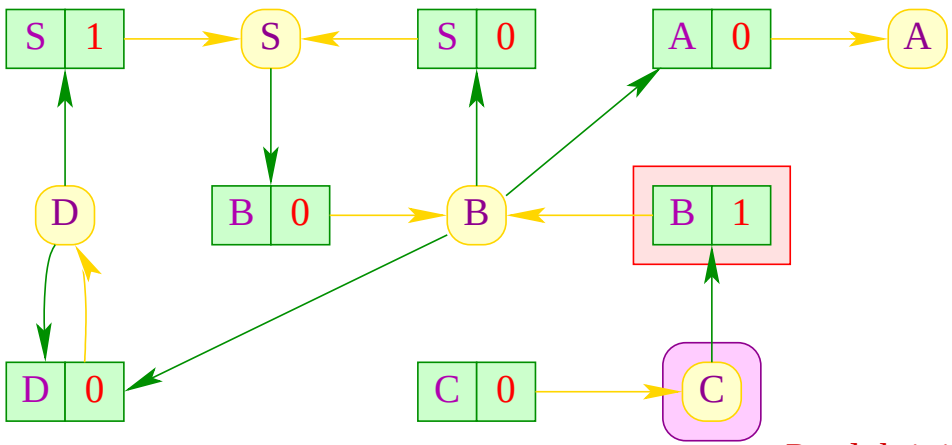
Die Menge W verwaltet die Regeln, deren rechte Seiten nur produktive Nichtterminale enthalten :-))

... im Beispiel:

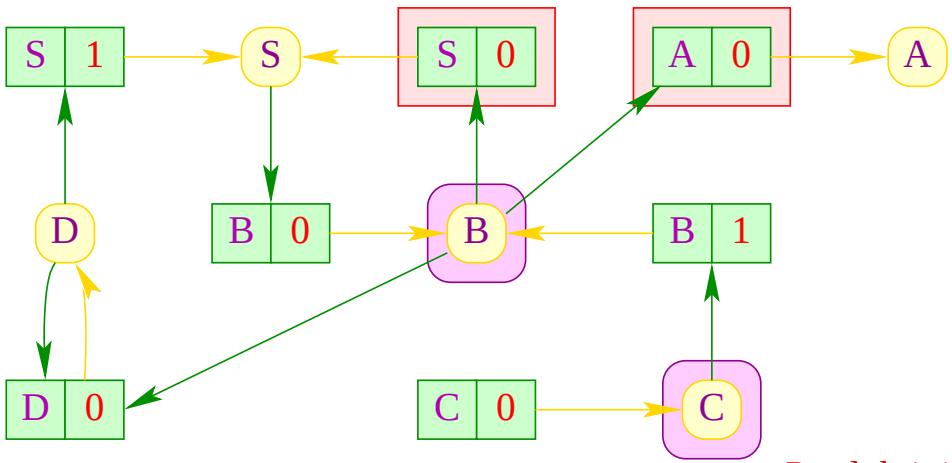


Produktivität

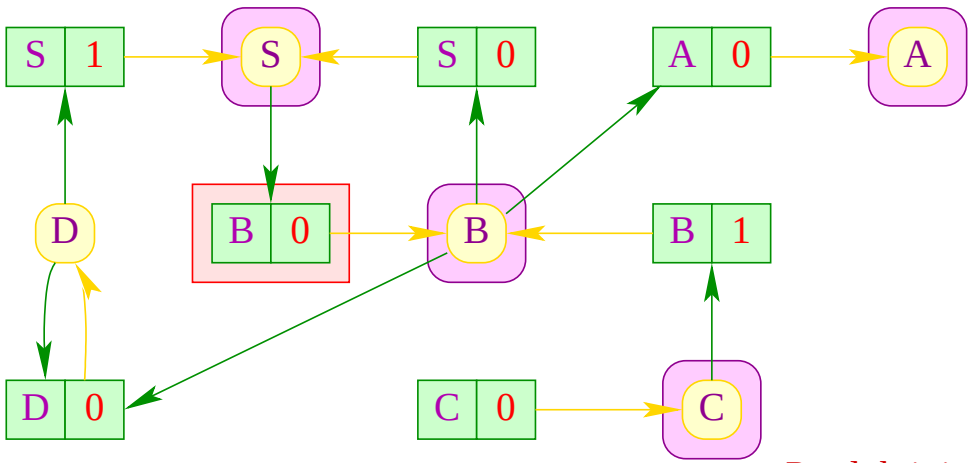
... im Beispiel:



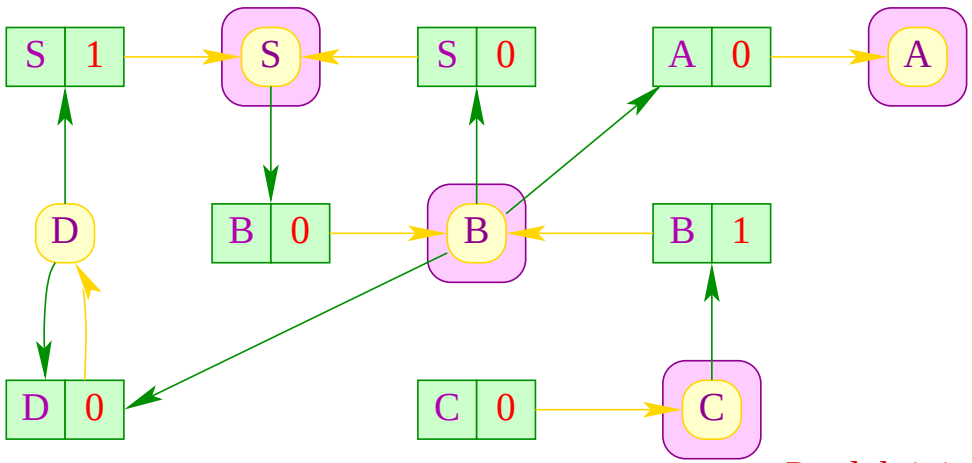
... im Beispiel:



... im Beispiel:



... im Beispiel:



Laufzeit:

- Die Initialisierung der Datenstrukturen erfordert lineare Laufzeit.
- Jede Regel wird maximal einmal in W eingefügt.
- Jedes A wird maximal einmal in result eingefügt.

  $\Longrightarrow$ Der Gesamtaufwand ist linear in der Größe der Grammatik :-)

Korrektheit:

- Falls A in der j-ten Iteration der **while**-Schleife in result eingefügt, gibt es einen Ableitungsbaum für A der Höhe maximal $j-1$:-)
- Für jeden Ableitungsbaum wird die Wurzel einmal in W eingefügt :-)