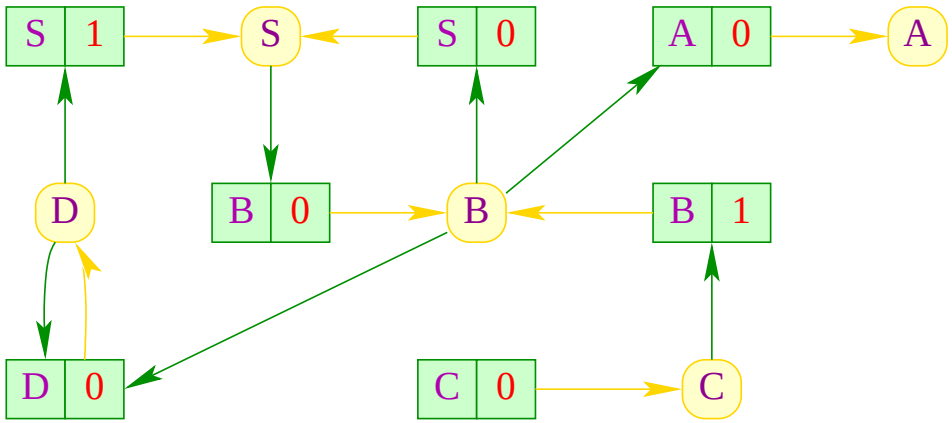
Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



And-Knoten: Regeln

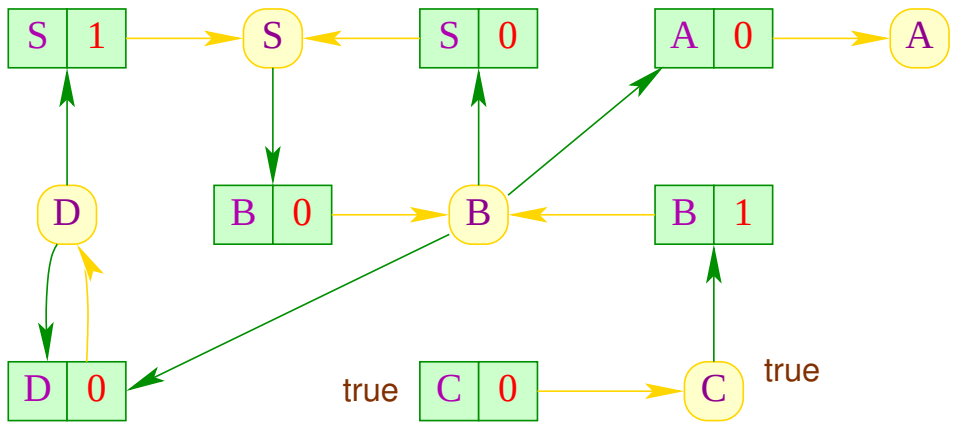
Or-Knoten: Nichtterminale

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

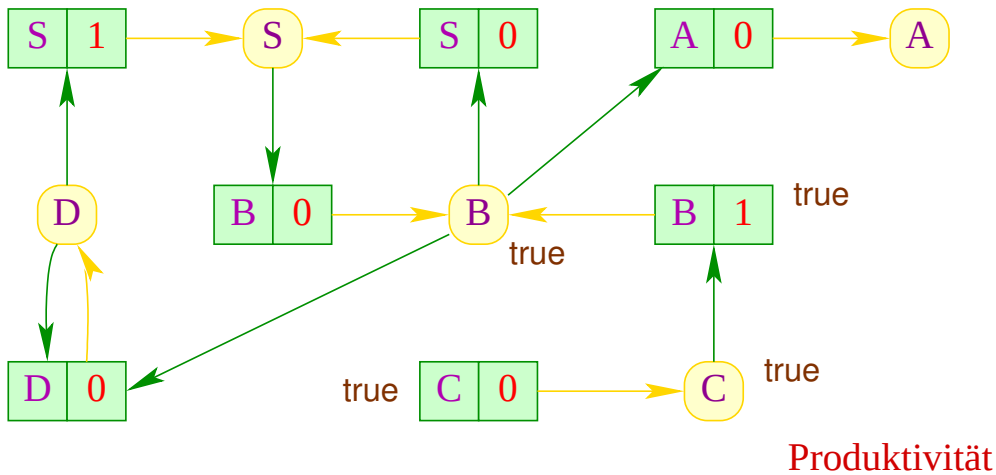
Or-Knoten: Nichtterminale

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

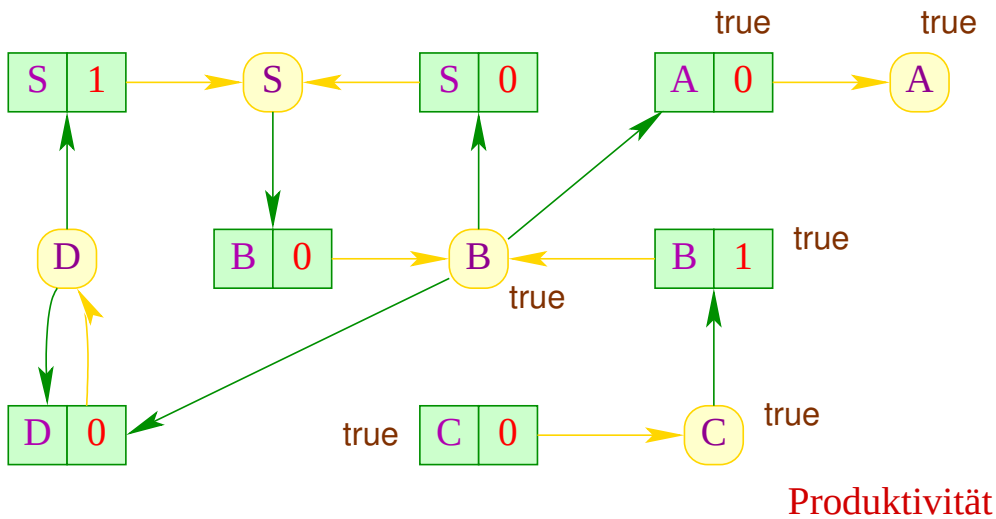
Or-Knoten: Nichtterminale

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

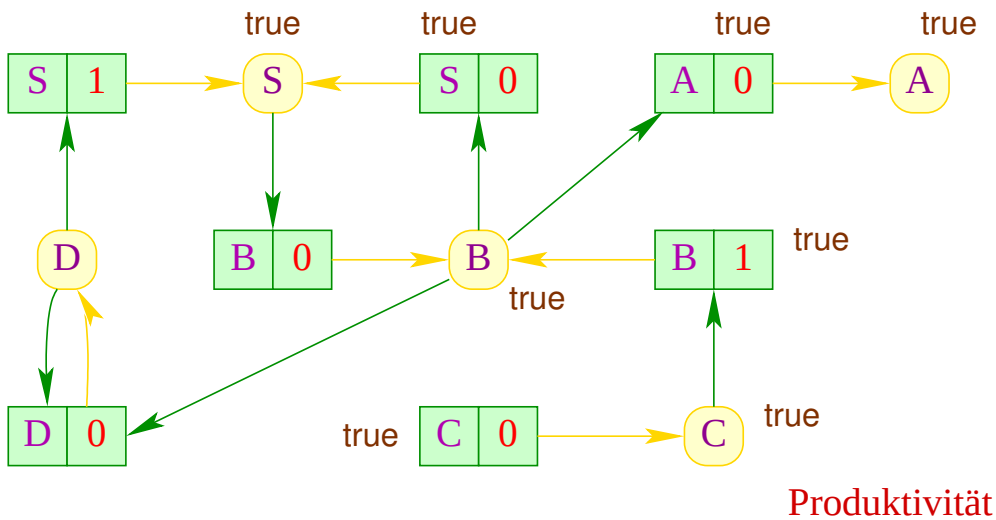
Or-Knoten: Nichtterminale

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

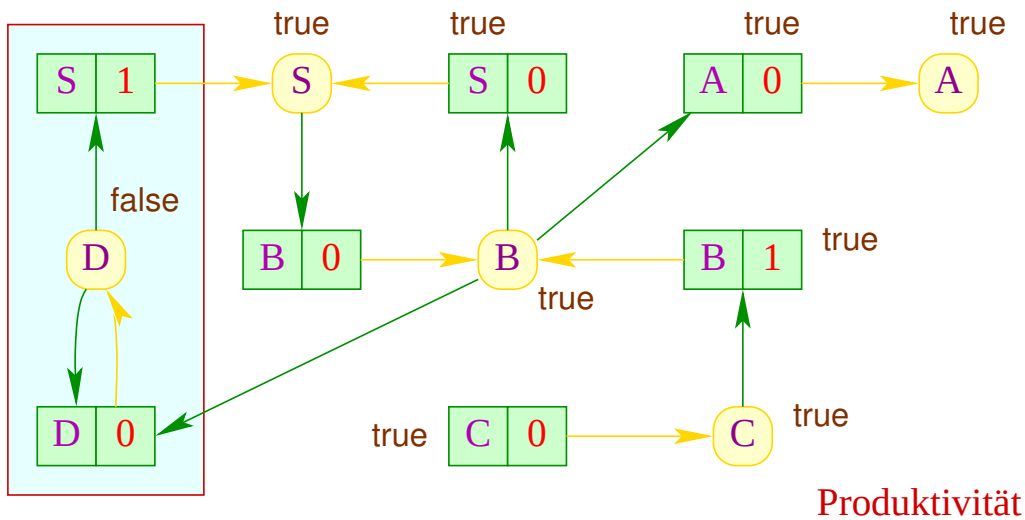
Or-Knoten: Nichtterminale

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:

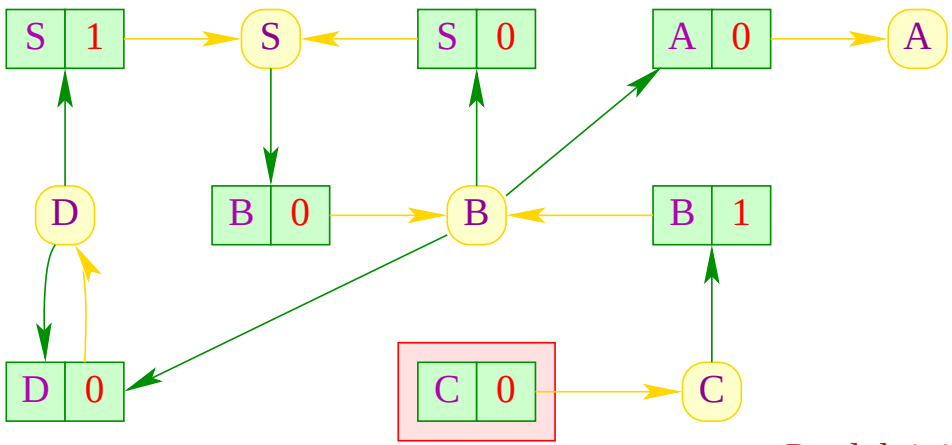
```
2^N  result = ∅;                          //  Ergebnis-Menge
int  count[P];                            //  Zähler für jede Regel
2^P  rhs[N];                              //  Vorkommen in rechten Seiten

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
```

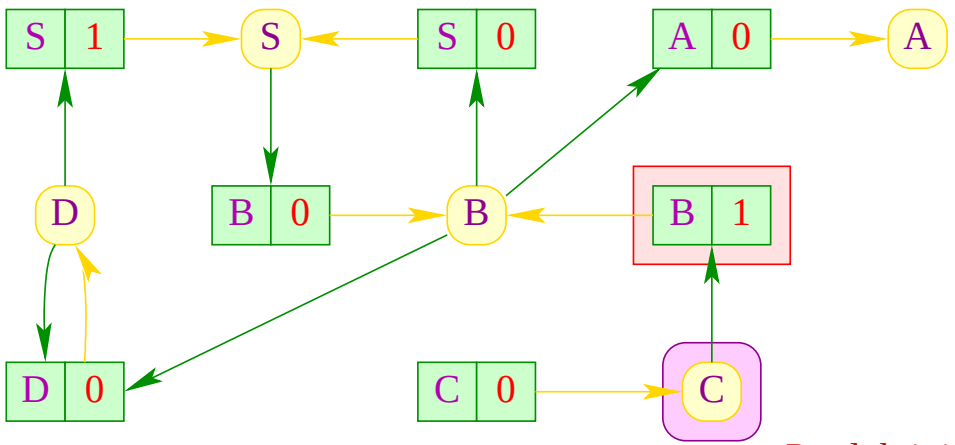
Die Menge W verwaltet die Regeln, deren rechte Seiten nur produktive Nichtterminale enthalten :-))

... im Beispiel:

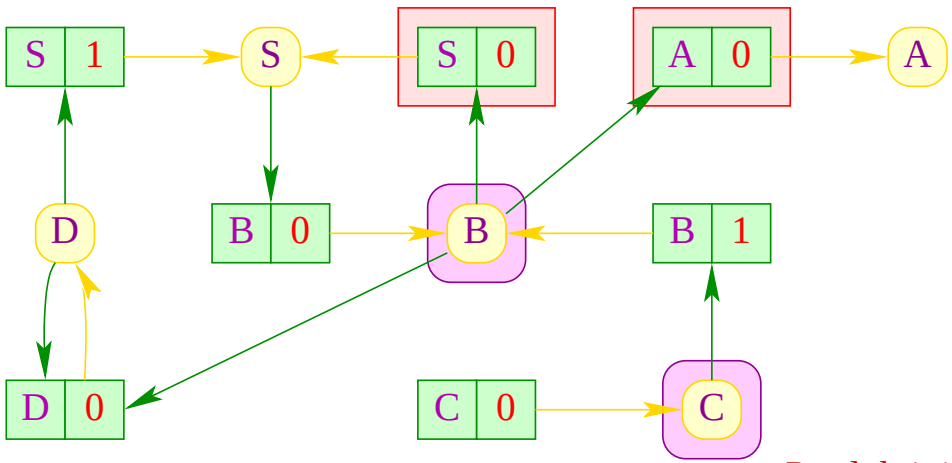


Produktivität

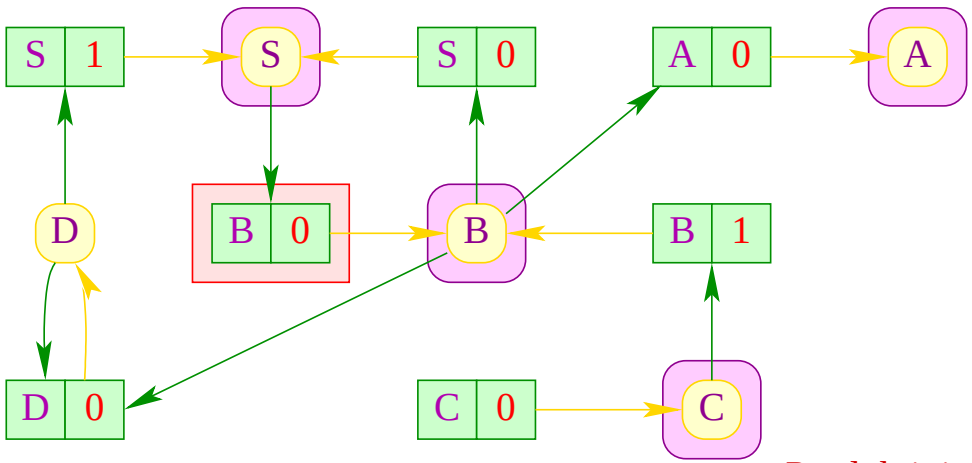
... im Beispiel:



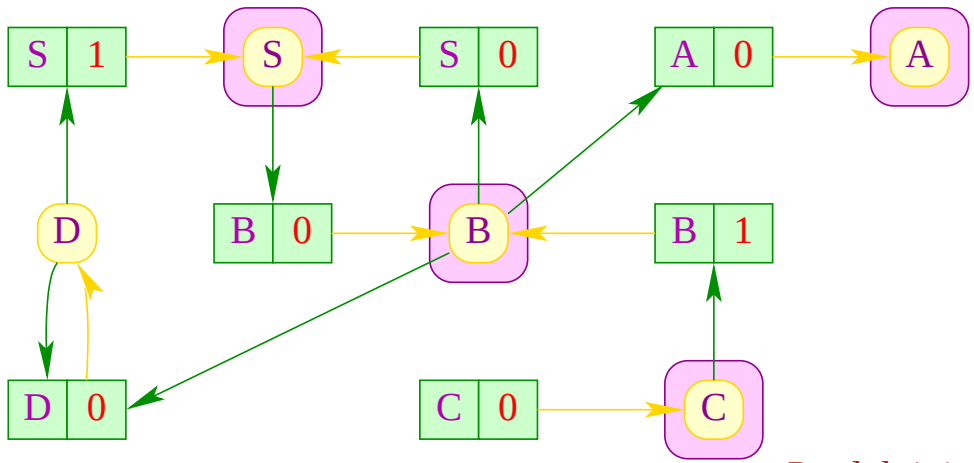
... im Beispiel:



... 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 :-)

Diskussion:

- Um den Test $(A \in \mathsf{result})$ einfach zu machen, repräsentiert man die Menge $\mathsf{result})$ durch ein Array.
- W wie auch die Mengen $\mathsf{rhs}[A]$ wird man dagegen als Listen repräsentieren :-)

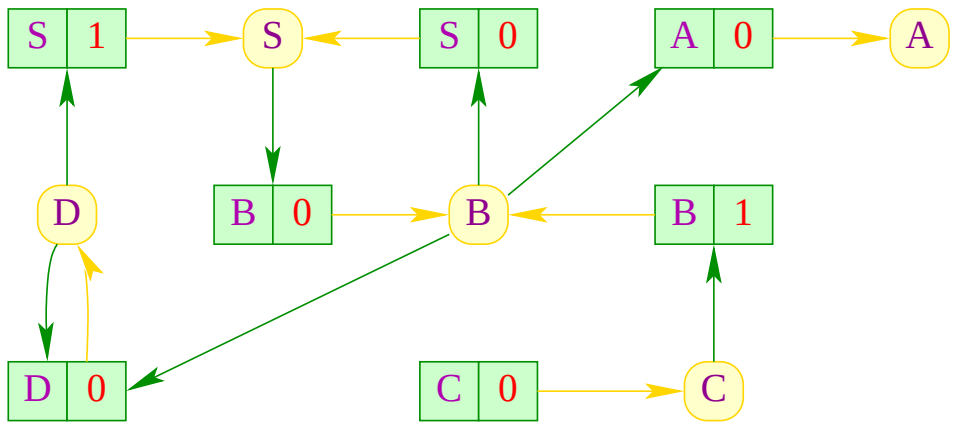
Diskussion:

- Um den Test $(A \in \mathsf{result})$ einfach zu machen, repräsentiert man die Menge result) durch ein Array.
- W wie auch die Mengen $\mathsf{rhs}[A]$ wird man dagegen als Listen repräsentieren :-)

- Der Algorithmus funktioniert auch, um kleinste Lösungen von Booleschen Ungleichungssystemen zu bestimmen :-)
- Die Ermittlung der produktiven Nichtterminale kann benutzt werden, um festzustellen, ob $\mathcal{L}(G) \neq \emptyset$ ist $(\rightarrow$ Leerheitsproblem)

Idee für Erreichbarkeit: Abhängigkeits-Graph

... hier:

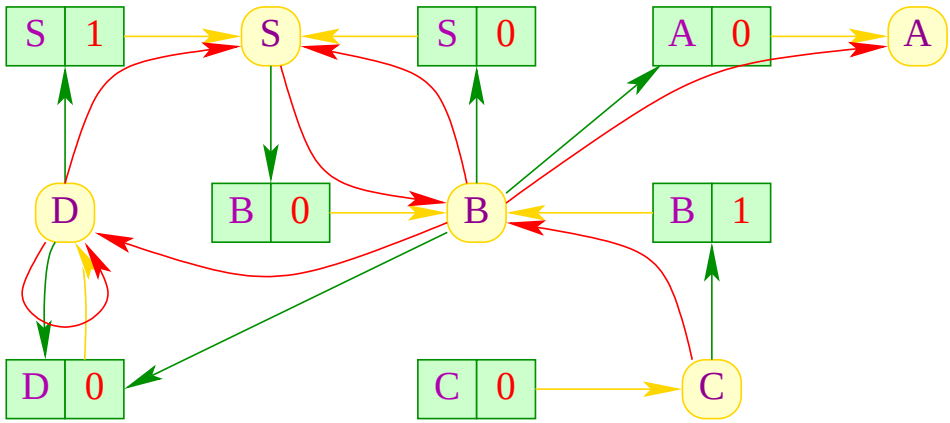


Knoten: Nichtterminale

Kanten: (A, B) falls $B \to \alpha_1\, A\, \alpha_2 \in P$

Idee für Erreichbarkeit: Abhängigkeits-Graph

... hier:

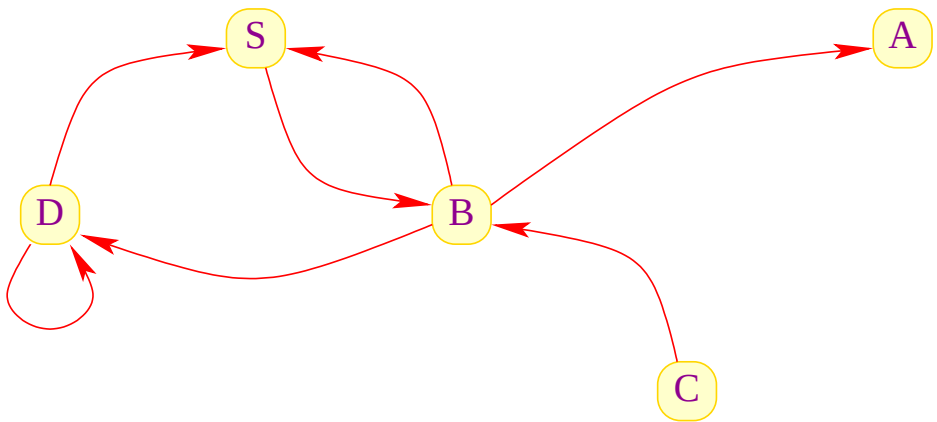


Knoten: Nichtterminale

Kanten: (A, B) falls $B \to \alpha_1\, A\, \alpha_2 \in P$

Idee für Erreichbarkeit: Abhängigkeits-Graph

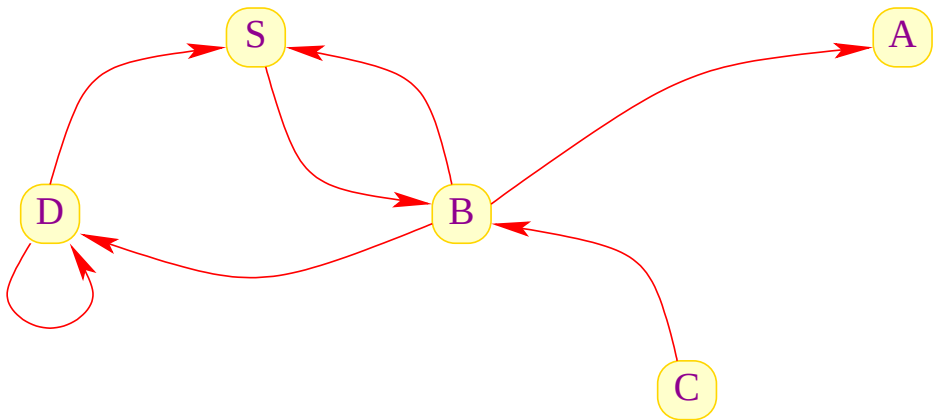
... hier:



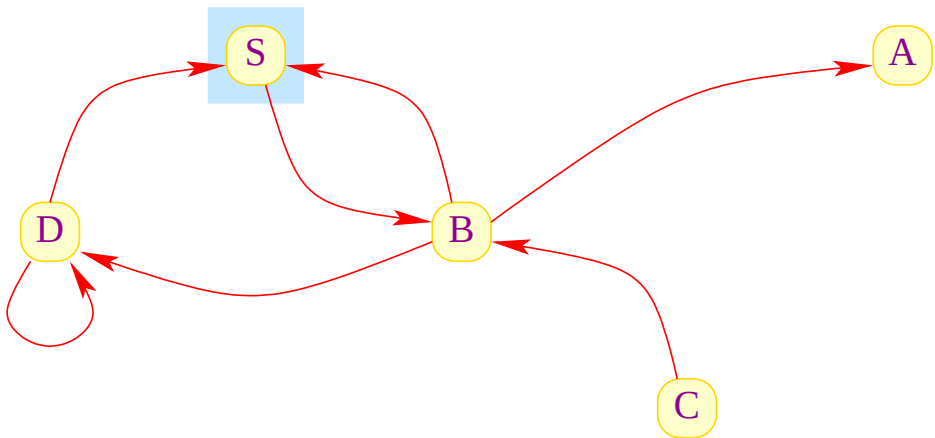
Knoten: Nichtterminale

Kanten: (A, B) falls $B \to \alpha_1\, A\, \alpha_2 \in P$

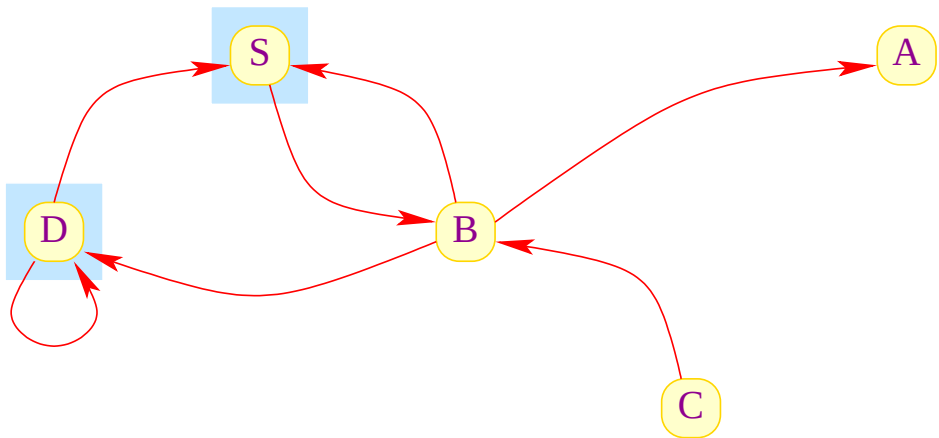
Das Nichtterminal A ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von A nach S gibt :-)



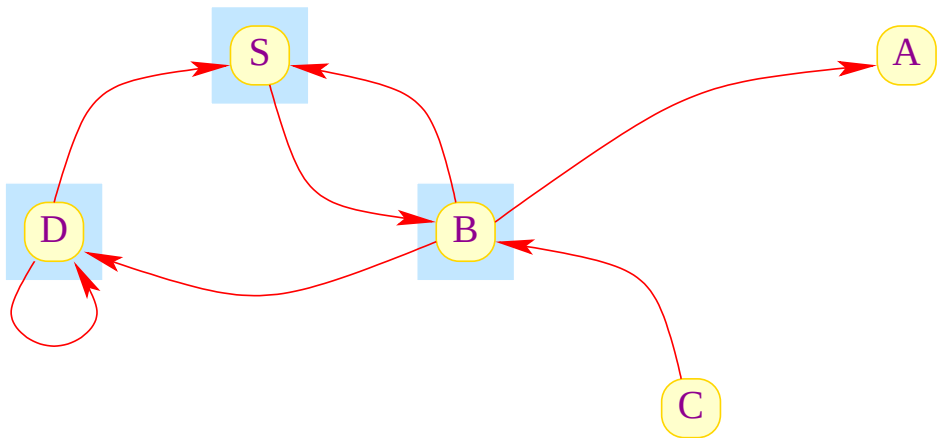
Das Nichtterminal A ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von A nach S gibt :-)



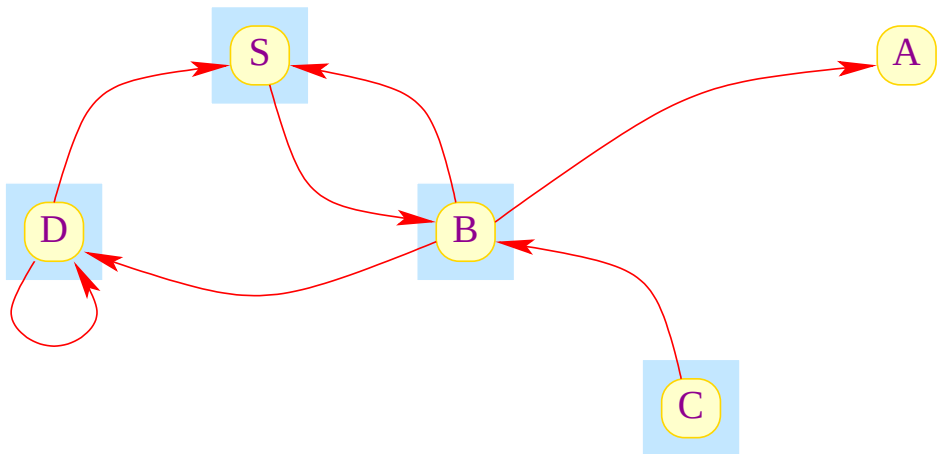
Das Nichtterminal A ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von A nach S gibt :-)



Das Nichtterminal A ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von A nach S gibt :-)



Das Nichtterminal A ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von A nach S gibt :-)



Fazit:

- Erreichbarkeit in gerichteten Graphen kann mithilfe von DFS in linearer Zeit berechnet werden.
- Damit kann die Menge aller erreichbaren und produktiven Nichtterminale in linearer Zeit berechnet werden :-)

Fazit:

- Erreichbarkeit in gerichteten Graphen kann mithilfe von DFS in linearer Zeit berechnet werden.
- Damit kann die Menge aller erreichbaren und produktiven Nichtterminale in linearer Zeit berechnet werden :-)

Eine Grammatik G heißt reduziert, wenn alle Nichtterminale von G sowohl produktiv wie erreichbar sind ...

Fazit:

- Erreichbarkeit in gerichteten Graphen kann mithilfe von DFS in linearer Zeit berechnet werden.
- Damit kann die Menge aller erreichbaren und produktiven Nichtterminale in linearer Zeit berechnet werden :-)

Eine Grammatik G heißt reduziert, wenn alle Nichtterminale von G sowohl produktiv wie erreichbar sind ...

Satz

Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ mit $\mathcal{L}(G) \neq \emptyset$ kann in linearer Zeit eine reduzierte Grammatik G' konstruiert werden mit

$$\mathcal{L}(G) = \mathcal{L}(G')$$

Konstruktion:

1. Schritt:

Berechne die Teilmenge $N_1 \subseteq N$ aller produktiven Nichtterminale von G.

Da $\mathcal{L}(G) \neq \emptyset$ ist insbesondere $S \in N_1$:-)

2. Schritt:

Konstruiere: $P_1 = \{A \to \alpha \in P \mid A \in N_1 \wedge \alpha \in (N_1 \cup T)^*\}$

Konstruktion (Forts.):

3. Schritt:

Berechne die Teilmenge $N_2 \subseteq N_1$ aller produktiven und erreichbaren Nichtterminale von G.

Da $\mathcal{L}(G) \neq \emptyset$ ist insbesondere $S \in N_2$:-))

4. Schritt:

Konstruiere: $P_2 = \{A \to \alpha \in P \mid A \in N_2 \wedge \alpha \in (N_2 \cup T)^*\}$

Konstruktion (Forts.):

3. Schritt:

Berechne die Teilmenge $N_2 \subseteq N$ aller produktiven und erreichbaren Nichtterminale von G.

Da $\mathcal{L}(G) \neq \emptyset$ ist insbesondere $S \in N_2$:-))

4. Schritt:

Konstruiere: $P_2 = \{A \to \alpha \in P \mid A \in N_2 \wedge \alpha \in (N_2 \cup T)^*\}$

Ergebnis: $G' = (N_2, T, P_2, S)$:-)

... im Beispiel:

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

... im Beispiel:

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

... im Beispiel:

$$
\begin{array}{lcl}
S & \rightarrow & a\,B\,B \\
A & \rightarrow & B\,c \\
B & \rightarrow & S\,d \;\mid\; C \\
C & \rightarrow & a
\end{array}
$$

... im Beispiel:

$$
\begin{array}{lcl}
S & \rightarrow & a\,B\,B \\
A & \rightarrow & B\,c \\
B & \rightarrow & S\,d \;\mid\; C \\
C & \rightarrow & a
\end{array}
$$

... im Beispiel:

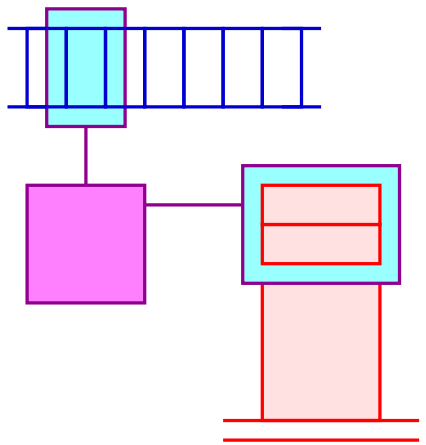
$$S \rightarrow a\,B\,B$$

$$B \rightarrow S\,d \mid C$$

$$C \rightarrow a$$

2.2 Grundlagen: Kellerautomaten

Durch kontextfreie Grammatiken spezifizierte Sprachen können durch Kellerautomaten (Pushdown Automata) akzeptiert werden:

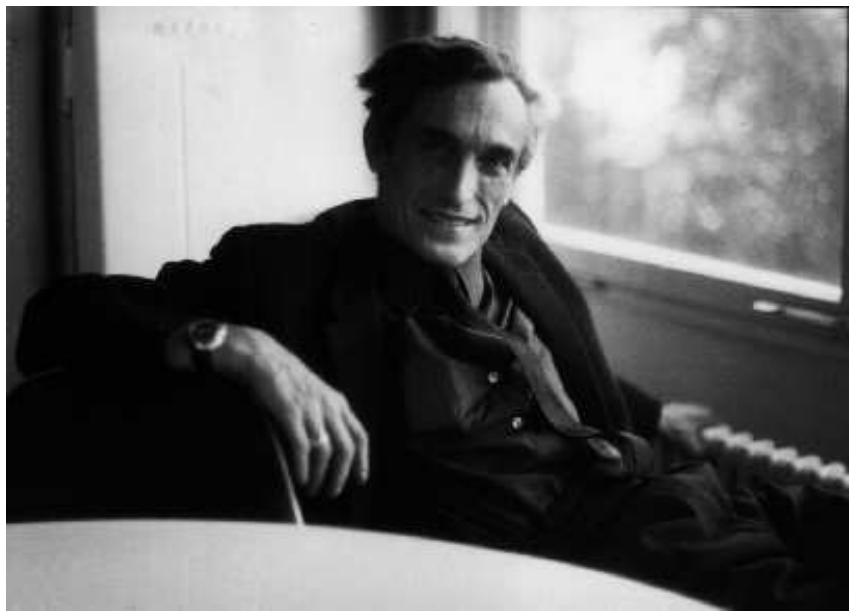


Der Keller wird z.B. benötigt, um korrekte Klammerung zu überprüfen :-)

Friedrich L. Bauer, TUM

Klaus Samelson, TUM

Kellerautomaten für kontextfreie Sprachen wurden erstmals vorgeschlagen von Michel Schützenberger und Antony G. Öttinger:



Marcel-Paul Schützenberger
(1920-1996), Paris

Antony G. Öttinger, Präsident der
ACM 1966-68

Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

Achtung:

- Wir unterscheiden nicht zwischen Kellersymbolen und Zuständen :-)
- Das rechteste / oberste Kellersymbol repräsentiert den Zustand :-)
- Jeder Übergang liest / modifiziert einen oberen Abschnitt des Kellers :-)

Formal definieren wir deshalb einen Kellerautomaten (PDA) als ein Tupel:
$M = (Q, T, \delta, q_0, F)$ wobei:

- Q eine endliche Menge von Zuständen;
- T das Eingabe-Alphabet;
- $q_0 \in Q$ der Anfangszustand;
- $F \subseteq Q$ die Menge der Endzustände und
- $\delta \subseteq Q^+ \times (T \cup \{\epsilon\}) \times Q^*$ eine endliche Menge von Übergängen ist (das Programm :-)

Formal definieren wir deshalb einen Kellerautomaten (PDA) als ein Tupel:
$M = (Q, T, \delta, q_0, F)$ wobei:

- Q eine endliche Menge von Zuständen;
- T das Eingabe-Alphabet;
- $q_0 \in Q$ der Anfangszustand;
- $F \subseteq Q$ die Menge der Endzustände und
- $\delta \subseteq Q^+ \times (T \cup \{\epsilon\}) \times Q^*$ eine endliche Menge von Übergängen ist (das Programm :-)

Mithilfe der Übergänge definieren wir Berechnungen von Kellerautomaten :-)

Der jeweilige Berechnungszustand (die aktuelle Konfiguration) ist ein Paar:

$$(\gamma, w) \in Q^* \times T^*$$

bestehend aus dem Kellerinhalt und dem noch zu lesenden Input.

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$(0, \quad a\,a\,a\,b\,b\,b)$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$(0,\ aaabbb) \vdash (11,\ aabbb)$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$(0, \ aaabbb) \vdash (11, \ aabbb)$$
$$\vdash (111, \ abbb)$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$
\begin{aligned}
(0,\ aaabbb) &\vdash (11,\ aabbb)\\
&\vdash (111,\ abbb)\\
&\vdash (1111,\ bbb)
\end{aligned}
$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$
\begin{aligned}
(0,\ aaabbb) &\vdash (11,\ aabbb)\\
&\vdash (111,\ abbb)\\
&\vdash (1111,\ bbb)\\
&\vdash (112,\ bb)
\end{aligned}
$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$
\begin{aligned}
(0,\ aaabbb) &\vdash (11,\ aabbb)\\
&\vdash (111,\ abbb)\\
&\vdash (1111,\ bbb)\\
&\vdash (112,\ bb)\\
&\vdash (12,\ b)
\end{aligned}
$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$
\begin{aligned}
(0,\ aaabbb) &\vdash (11,\ aabbb)\\
&\vdash (111,\ abbb)\\
&\vdash (1111,\ bbb)\\
&\vdash (112,\ bb)\\
&\vdash (12,\ b)\\
&\vdash (2,\ \epsilon)
\end{aligned}
$$

Ein Berechnungsschritt wird durch die Relation $\vdash \subseteq (Q^* \times T^*)^2$ beschrieben, wobei

$$(\alpha\,\gamma,\, x\,w) \vdash (\alpha\,\gamma',\, w) \quad \text{für} \quad (\gamma,\, x,\, \gamma') \in \delta$$

Ein Berechnungsschritt wird durch die Relation $\vdash \subseteq (Q^* \times T^*)^2$ beschrieben, wobei

$$(\alpha\,\gamma,\ x\,w) \vdash (\alpha\,\gamma',\ w) \quad \text{für} \quad (\gamma,\ x,\ \gamma') \in \delta$$

Bemerkungen:

- Die Relation $\vdash$ hängt natürlich vom Kellerautomaten M ab :-)
- Die reflexive und transitive Hülle von $\vdash$ bezeichnen wir mit $\vdash^*$.
- Dann ist die von M akzeptierte Sprache:

$$\mathcal{L}(M) = \{w \in T^* \mid \exists\, f \in F :\ (q_0, w) \vdash^* (f, \epsilon)\}$$

Ein Berechnungsschritt wird durch die Relation $\vdash \subseteq (Q^* \times T^*)^2$ beschrieben, wobei

$$(\alpha\,\gamma,\ x\,w) \vdash (\alpha\,\gamma',\ w) \quad \text{für} \quad (\gamma,\ x,\ \gamma') \in \delta$$

Bemerkungen:

- Die Relation $\vdash$ hängt natürlich vom Kellerautomaten M ab :-)
- Die reflexive und transitive Hülle von $\vdash$ bezeichnen wir mit $\vdash^*$.
- Dann ist die von M akzeptierte Sprache:

$$\mathcal{L}(M) = \{w \in T^* \mid \exists\, f \in F :\ (q_0, w) \vdash^* (f, \epsilon)\}$$

Wir akzeptieren also mit Endzustand und leerem Keller :-)

Der Kellerautomat M heißt deterministisch, falls jede Konfiguration maximal eine Nachfolge-Konfiguration hat.

Das ist genau dann der Fall wenn für verschiedene Übergänge
$(\gamma_1, x, \gamma_2)\,,\ (\gamma_1', x', \gamma_2') \in \delta$ gilt:

Ist γ_1 ein Suffix von γ_1', dann muss $x \neq x' \ \wedge\ x \neq \epsilon \neq x'$ sein.

Der Kellerautomat M heißt deterministisch, falls jede Konfiguration maximal eine Nachfolge-Konfiguration hat.

Das ist genau dann der Fall wenn für verschiedene Übergänge
$(\gamma_1, x, \gamma_2)\,,\ (\gamma_1', x', \gamma_2') \in \delta$ gilt:

Ist γ_1 ein Suffix von γ_1', dann muss $x \neq x' \ \wedge\ x \neq \epsilon \neq x'$ sein.

... im Beispiel:

0	a	11
1	a	11
11	b	2
12	b	2

ist das natürlich der Fall :-))

Satz

Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ kann ein PDA M konstruiert werden mit $\mathcal{L}(G) = \mathcal{L}(M)$.

Der Satz ist für uns so wichtig, dass wir zwei Konstruktionen angeben :-)

Satz

Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ kann ein PDA M konstruiert werden mit $\mathcal{L}(G) = \mathcal{L}(M)$.

Der Satz ist für uns so wichtig, dass wir zwei Konstruktionen angeben :-)

Konstruktion 1: Shift-Reduce-Parser

- Die Eingabe wird sukzessive auf den Keller geschiftet.
- Liegt oben auf dem Keller eine vollständige rechte Seite (ein Handle) vor, wird dieses durch die zugehörige linke Seite ersetzt (reduziert) :-)

Beispiel:

$$\begin{array}{lcl} S & \rightarrow & A\,B \\ A & \rightarrow & a \\ B & \rightarrow & b \end{array}$$

Der Kellerautomat:

Zustände: q_0, f, a, b, A, B, S;

Anfangszustand: q_0

Endzustand: f

q_0	a	$q_0\,a$
a	ϵ	A
A	b	$A\,b$
b	ϵ	B
$A\,B$	ϵ	S
$q_0\,S$	ϵ	f

Allgemein konstruieren wir einen Automaten $M_G^{(1)} = (Q, T, \delta, q_0, F)$ mit:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f neu);
- $F = \{f\}$;
- Übergänge:

$$
\begin{array}{lcll}
\delta & = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \cup & // \text{ Shift-Übergänge} \\
 & & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \cup & // \text{ Reduce-Übergänge} \\
 & & \{(q_0\,S, \epsilon, f)\} & // \text{ Abschluss :-)}
\end{array}
$$

Allgemein konstruieren wir einen Automaten $M_G^{(1)} = (Q, T, \delta, q_0, F)$ mit:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f neu);
- $F = \{f\}$;
- Übergänge:

$$\begin{array}{lll} \delta = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \cup & // \text{ Shift-Übergänge} \\ & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \cup & // \text{ Reduce-Übergänge} \\ & \{(q_0\,S, \epsilon, f)\} & // \text{ Abschluss :-)} \end{array}$$

Eine Beispiel-Berechnung:

$$\begin{array}{rcrcr} (q_0, & a\,b) & \vdash & (q_0\ a\ , & b) \\ & \vdash & (q_0\,A\ b\ , & \epsilon) & \vdash \\ & \vdash & (q_0\,S, & \epsilon) & \vdash \end{array} \quad \begin{array}{rcr} (q_0\,A, & b) & \\ (q_0\ A\,B\ , & \epsilon) & \\ (f, & \epsilon) & \end{array}$$

Allgemein konstruieren wir einen Automaten $M_G^{(1)} = (Q, T, \delta, q_0, F)$ mit:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f neu);
- $F = \{f\}$;
- Übergänge:

$$
\begin{array}{lll}
\delta = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \cup & // \text{ Shift-Übergänge} \\
& \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \cup & // \text{ Reduce-Übergänge} \\
& \{(q_0\,S, \epsilon, f)\} & // \text{ Abschluss :-)}
\end{array}
$$

Eine Beispiel-Berechnung:

$$
\begin{array}{rrrrrrr}
(q_0, & a\,b) & \vdash & (q_0\,\boxed{a}, & b) & \vdash & (q_0\,A, \quad b) \\
& & \vdash & (q_0\,A\,\boxed{b}, & \epsilon) & \vdash & (q_0\,\boxed{A\,B}, \quad \epsilon) \\
& & \vdash & (q_0\,S, & \epsilon) & \vdash & (f, \quad \epsilon)
\end{array}
$$