Algorithmus:

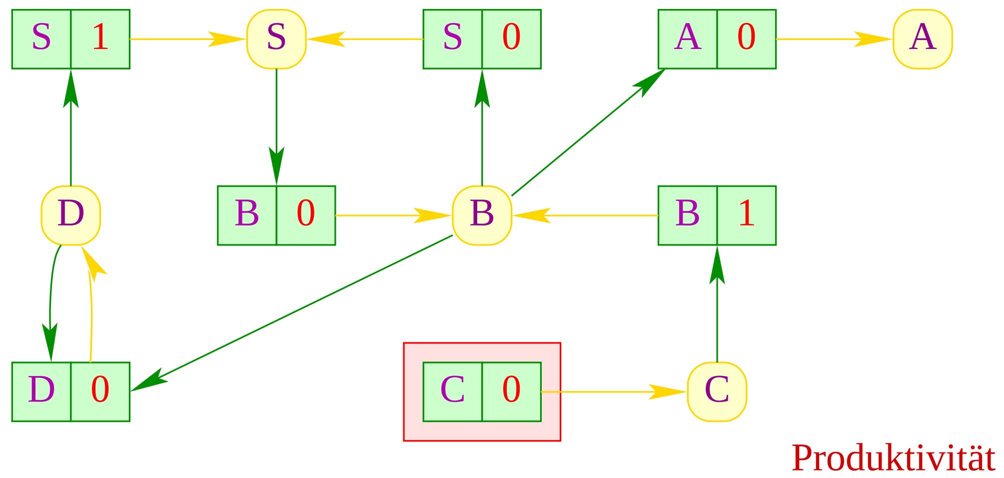
$$
\begin{array}{lll}
2^N \quad \mathsf{result} = \emptyset; & // & \text{Ergebnis-Menge} \\
\mathsf{int} \quad \mathsf{count}[P]; & // & \text{Zähler für jede Regel} \\
2^P \quad \mathsf{rhs}[N]; & // & \text{Vorkommen in rechten Seiten} \\
 & & \\
\mathbf{forall}\ (A \in N) \quad \mathsf{rhs}[A] = \emptyset; & // & \text{Initialisierung} \\
\mathbf{forall}\ ((A, i) \in P) \quad \{ & // & \\
\qquad\qquad \mathsf{count}[(A, i)] = 0; & // & \\
\qquad\qquad \mathsf{init}(A, i); & // & \text{Initialisierung von } \mathsf{rhs} \\
\qquad\qquad \} & // & \\
\ldots & // &
\end{array}
$$

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

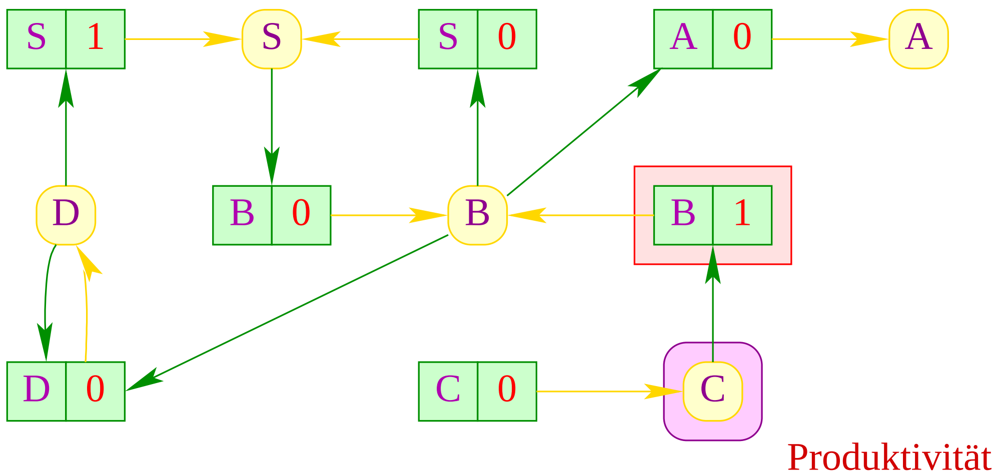
$$
\begin{array}{ll}
\ldots & // \\
2^P \quad W = \{r \mid \mathsf{count}[r] = 0\}; & // \quad \text{Workset} \\
\textbf{while}\ (W \neq \emptyset) \quad \{ & // \\
\qquad (A, i) = \mathsf{extract}(W); & // \\
\qquad \textbf{if}\ (A \notin \mathsf{result}) \quad \{ & // \\
\qquad\qquad \mathsf{result} = \mathsf{result} \cup \{A\}; & // \\
\qquad\qquad \textbf{forall}\ (r \in \mathsf{rhs}[A]) \quad \{ & // \\
\qquad\qquad\qquad \mathsf{count}[r]--; & // \\
\qquad\qquad\qquad \textbf{if}\ (\mathsf{count}[r] == 0) \quad W = W \cup \{r\}; & // \\
\qquad\qquad\qquad \} & // \quad \text{end of} \quad \textbf{forall} \\
\qquad\qquad \} & // \quad \text{end of} \quad \textbf{if} \\
\qquad \} & // \quad \text{end of} \quad \textbf{while}
\end{array}
$$

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

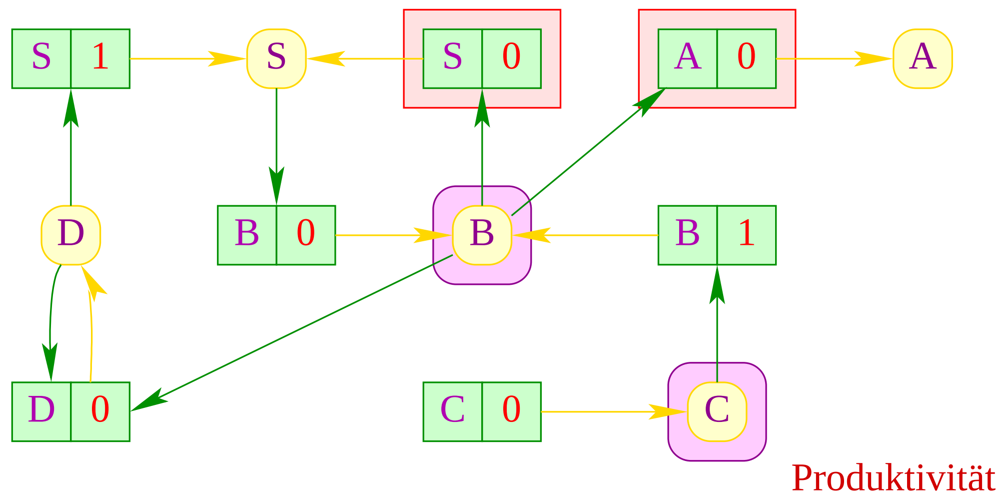
... im Beispiel:



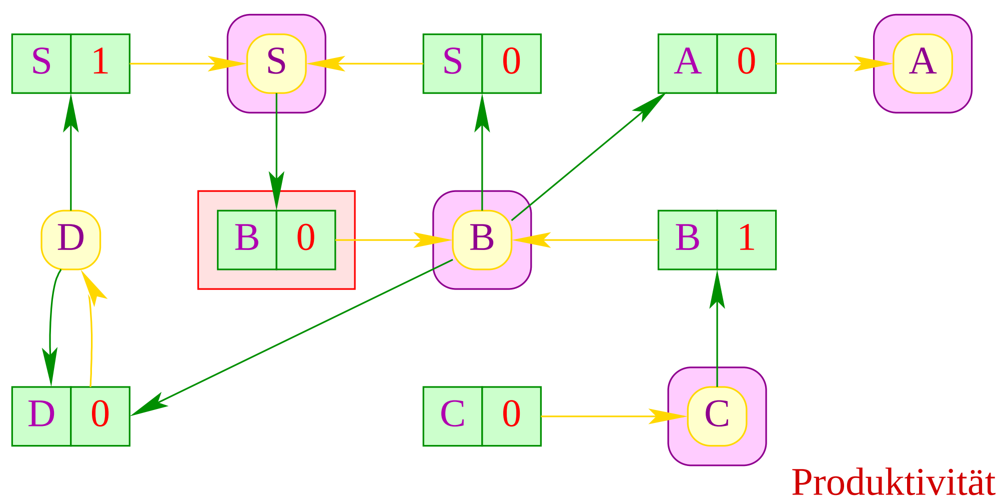
... im Beispiel:



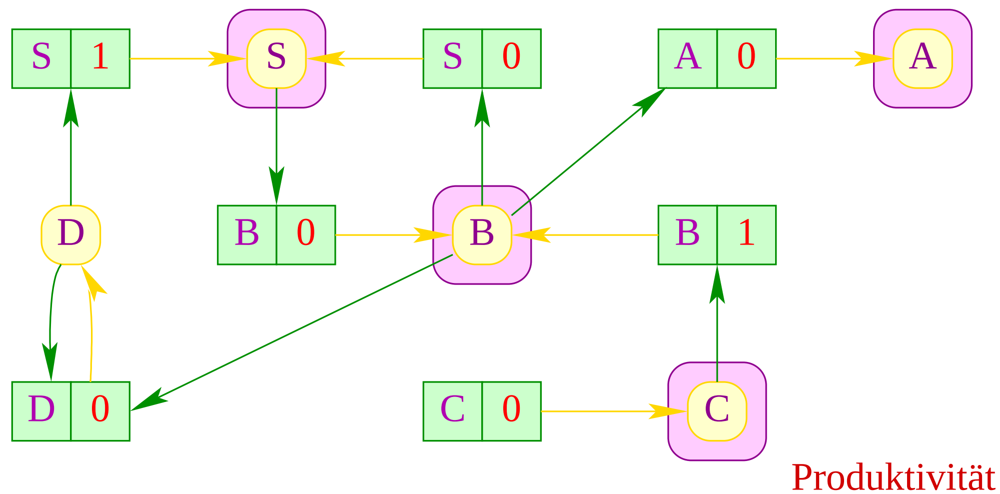
... im Beispiel:



... 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 :-)

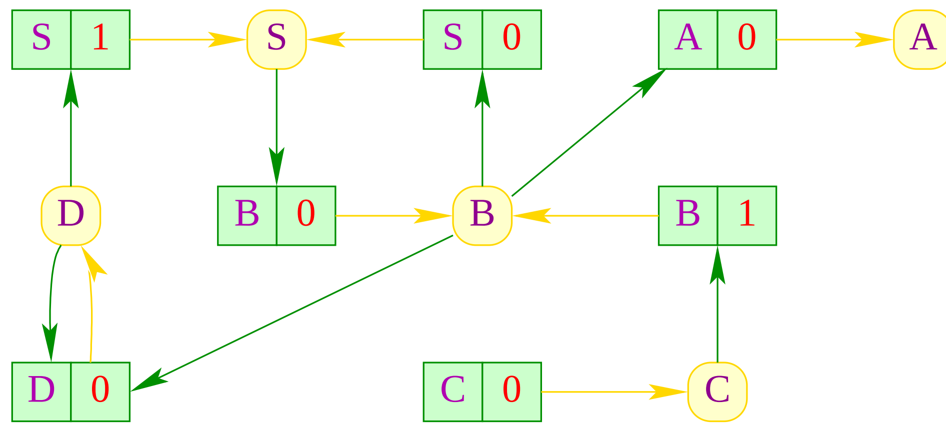
Diskussion:

- 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:

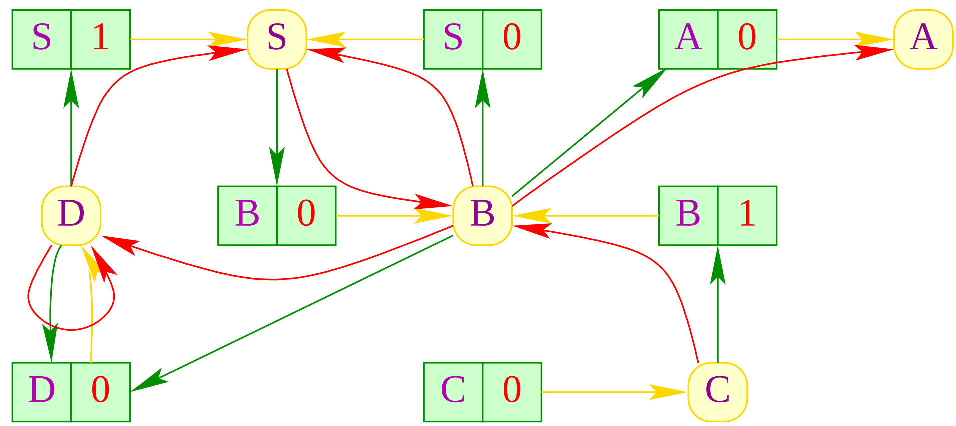


Knoten: Nichtterminale

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

Idee für Erreichbarkeit: Abhängigkeits-Graph

... hier:

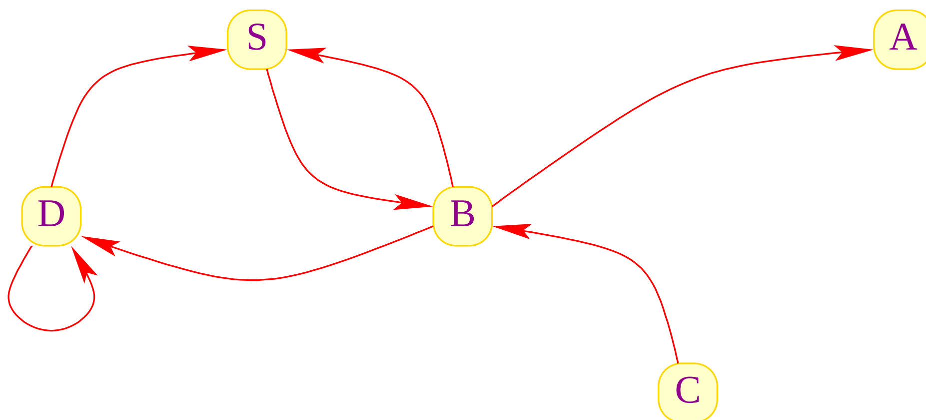


Knoten: Nichtterminale

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

Idee für Erreichbarkeit: Abhängigkeits-Graph

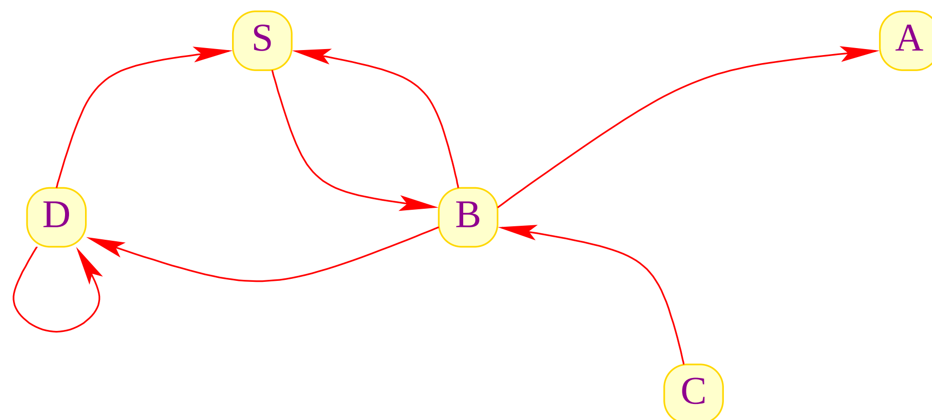
... hier:



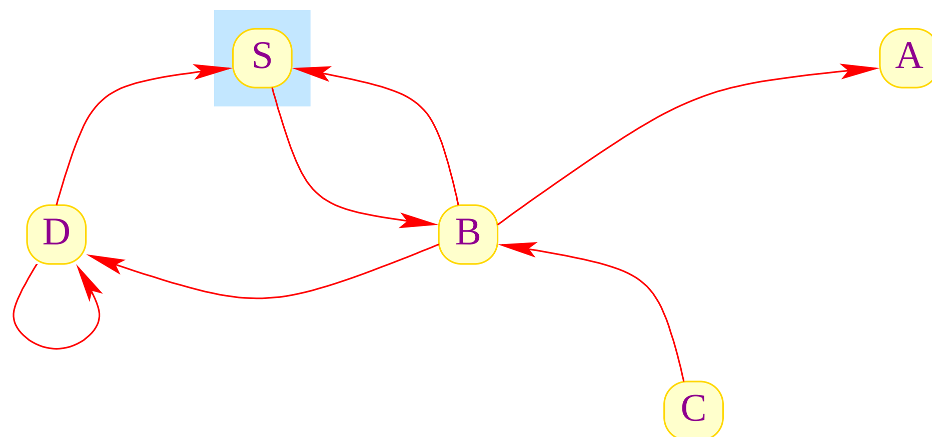
Knoten: Nichtterminale

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

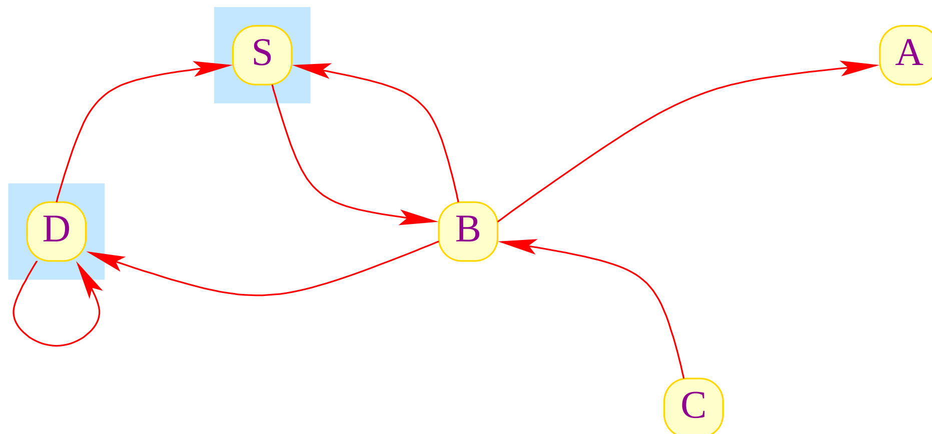
Das Nichtterminal A ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von A nach S gibt :-)



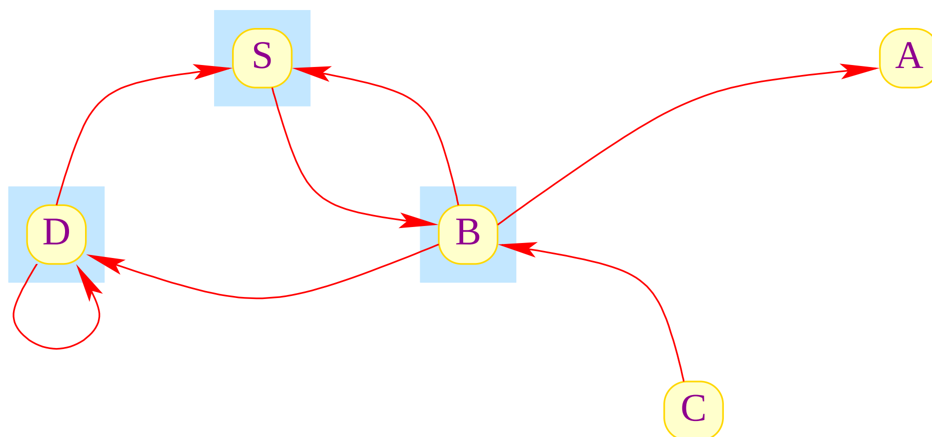
Das Nichtterminal A ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von A nach S gibt :-)



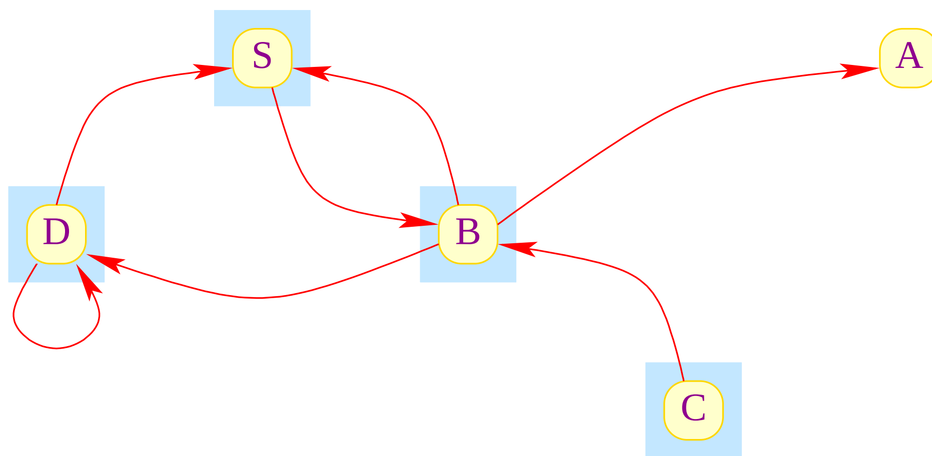
Das Nichtterminal A ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von A nach S gibt :-)



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 $N' \subseteq N$ aller produktiven und erreichbaren Nichtterminale von G .

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

Konstruktion:

1. Schritt:

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

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

2. Schritt:

Konstruiere: $P' = \{A \to \alpha \ \in \ P \mid A \in N' \wedge \alpha \in (N' \cup T)^*\}$

Konstruktion:

1. Schritt:

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

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

2. Schritt:

Konstruiere: $P' = \{A \to \alpha \in P \mid A \in N' \wedge \alpha \in (N' \cup T)^*\}$

Ergebnis: $G' = (N', T, P', S)$:-)

... im Beispiel:

$$
\begin{array}{lll}
S & \rightarrow & a\,B\,B \quad | \quad b\,D \\
A & \rightarrow & B\,c \\
B & \rightarrow & S\,d \quad | \quad C \\
C & \rightarrow & a \\
D & \rightarrow & B\,D
\end{array}
$$

... im Beispiel:

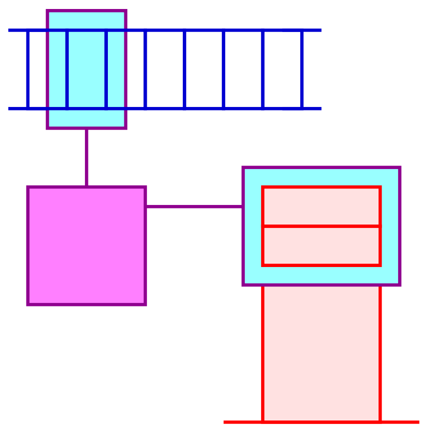
$$
\begin{array}{lll}
S & \rightarrow & a\,B\,B \quad | \quad b\,D \\
A & \rightarrow & B\,c \\
B & \rightarrow & S\,d \quad | \quad C \\
C & \rightarrow & a \\
D & \rightarrow & B\,D
\end{array}
$$

... im Beispiel:

$$\begin{array}{lll} S & \longrightarrow & a\,B\,B \\[1em] B & \longrightarrow & S\,d \quad \mid \quad C \\ C & \longrightarrow & a \end{array}$$

2.2 Grundlagen: Kellerautomaten

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

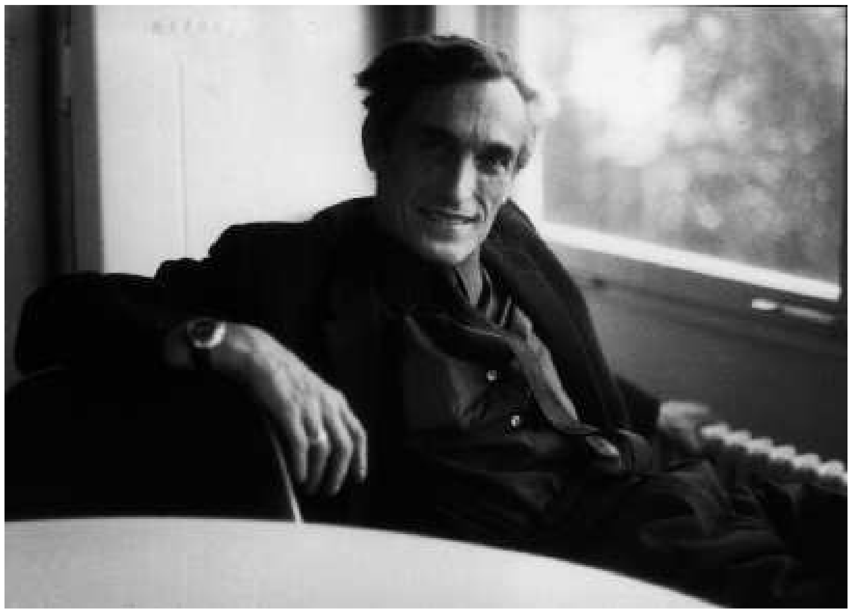


Der Keller wird z.B. benötigt, um korrekte Klammerung zu überprüfen :-)

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

$$(0\,,\quad a\,a\,a\,b\,b\,b) \quad \vdash \quad (1\,1\,,\quad a\,a\,b\,b\,b)$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

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

$$\begin{aligned}(0,\ \ a\,a\,a\,b\,b\,b) &\vdash (1\,1,\ \ a\,a\,b\,b\,b)\\ &\vdash (1\,1\,1,\ \ a\,b\,b\,b)\end{aligned}$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

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

$$\begin{aligned} (0\,, \quad a\,a\,a\,b\,b\,b) \quad &\vdash \quad (1\,1\,, \quad a\,a\,b\,b\,b) \\ &\vdash \quad (1\,1\,1\,, \quad a\,b\,b\,b) \\ &\vdash \quad (1\,1\,1\,1\,, \quad b\,b\,b) \end{aligned}$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

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

$$
\begin{array}{rcr}
(0\,, \quad a\,a\,a\,b\,b\,b) & \vdash & (1\,1\,, \quad a\,a\,b\,b\,b) \\
& \vdash & (1\,1\,1\,, \quad a\,b\,b\,b) \\
& \vdash & (1\,1\,1\,1\,, \quad b\,b\,b) \\
& \vdash & (1\,1\,2\,, \quad b\,b)
\end{array}
$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

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

$$
\begin{array}{rcr}
(0, \quad a\,a\,a\,b\,b\,b) & \vdash & (1\,1, \quad a\,a\,b\,b\,b) \\
& \vdash & (1\,1\,1, \quad a\,b\,b\,b) \\
& \vdash & (1\,1\,1\,1, \quad b\,b\,b) \\
& \vdash & (1\,1\,2, \quad b\,b) \\
& \vdash & (1\,2, \quad b)
\end{array}
$$

... im Beispiel:

Zustände: $0, 1, 2$

Anfangszustand: 0

Endzustände: $0, 2$

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

$$
\begin{array}{rcr}
(0\,, \quad a\,a\,a\,b\,b\,b) & \vdash & (1\,1\,, \quad a\,a\,b\,b\,b) \\
& \vdash & (1\,1\,1\,, \quad a\,b\,b\,b) \\
& \vdash & (1\,1\,1\,1\,, \quad b\,b\,b) \\
& \vdash & (1\,1\,2\,, \quad b\,b) \\
& \vdash & (1\,2\,, \quad b) \\
& \vdash & (2\,, \quad \epsilon)
\end{array}
$$

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:

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

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:

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

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

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

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

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

Eine Beispiel-Berechnung:

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

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:

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

Offenbar gilt:

- Die Folge der Reduktionen entspricht einer reversen Rechtsableitung für die Eingabe :-)
- Zur Korrektheit zeigt man, dass für jedes q gilt:

$$(q, w) \vdash^* (q\, A, \epsilon) \qquad \text{gdw.} \qquad A \rightarrow^* w$$

- Der Kellerautomat $M_G^{(1)}$ ist i.a. nicht-deterministisch :-(
- Um ein deterministisches Parse-Verfahren zu erhalten, muss man die Reduktionsstellen identifizieren $\Longrightarrow$ LR-Parsing