Diskussion:

- Die meisten Übergänge dienen dazu, im Ausdruck zu navigieren :-(
- Der Automat ist i.a. nichtdeterministisch :-(

Diskussion:

- Die meisten Übergänge dienen dazu, im Ausdruck zu navigieren :-(
- Der Automat ist i.a. nichtdeterministisch :-(

$$\Longrightarrow$$

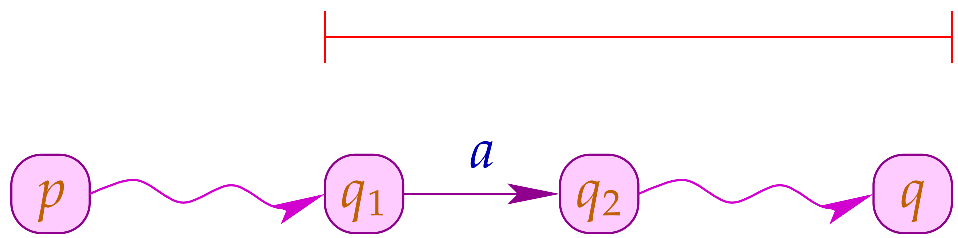
Strategie:

(1) Beseitigung der ϵ-Übergänge;

(2) Beseitigung des Nichtdeterminismus :-)

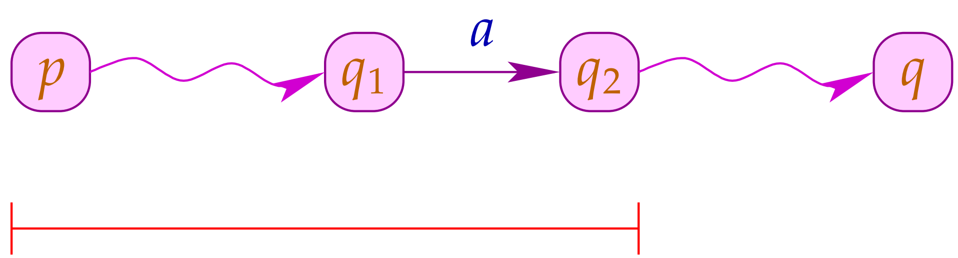
Beseitigung von ϵ-Übergängen:

Zwei einfache Ansätze:



Beseitigung von ϵ-Übergängen:

Zwei einfache Ansätze:

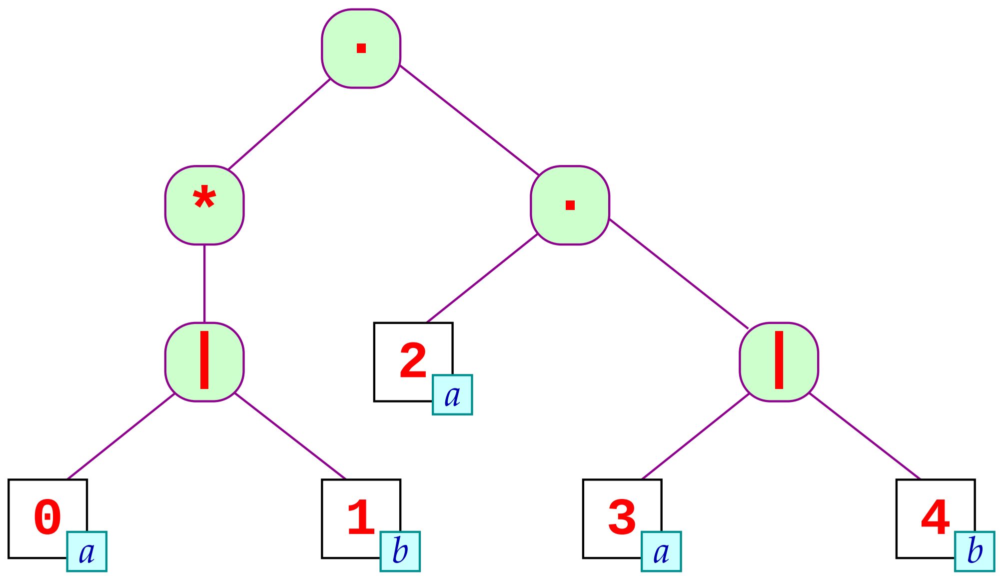


Wir benutzen hier den zweiten Ansatz.

Zur Konstruktion von Parsern werden wir später den ersten benutzen :-)

1. Schritt: $\mathsf{empty}[r] = t$ gdw. $\epsilon \in [\![r]\!]$

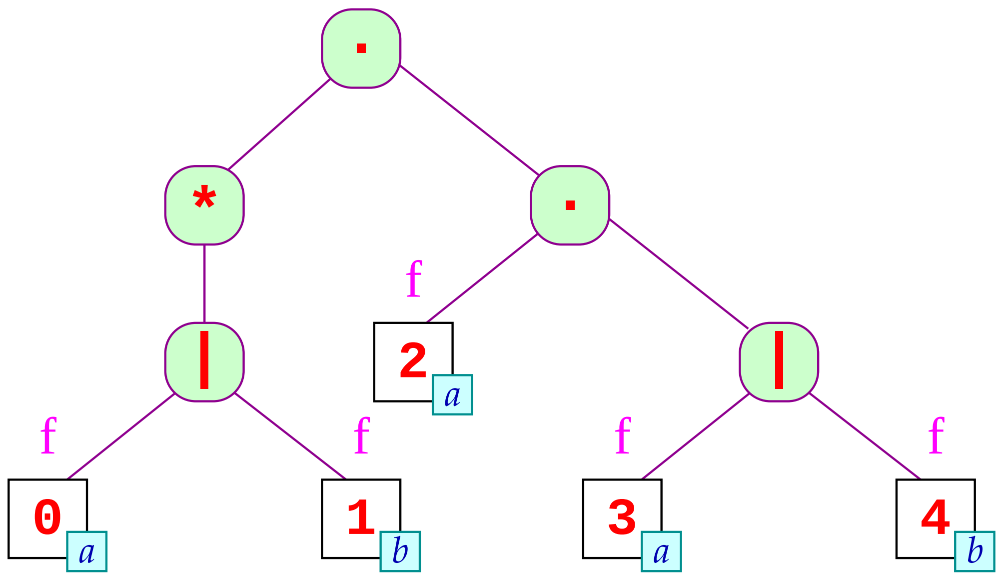
... im Beispiel:



1. Schritt:

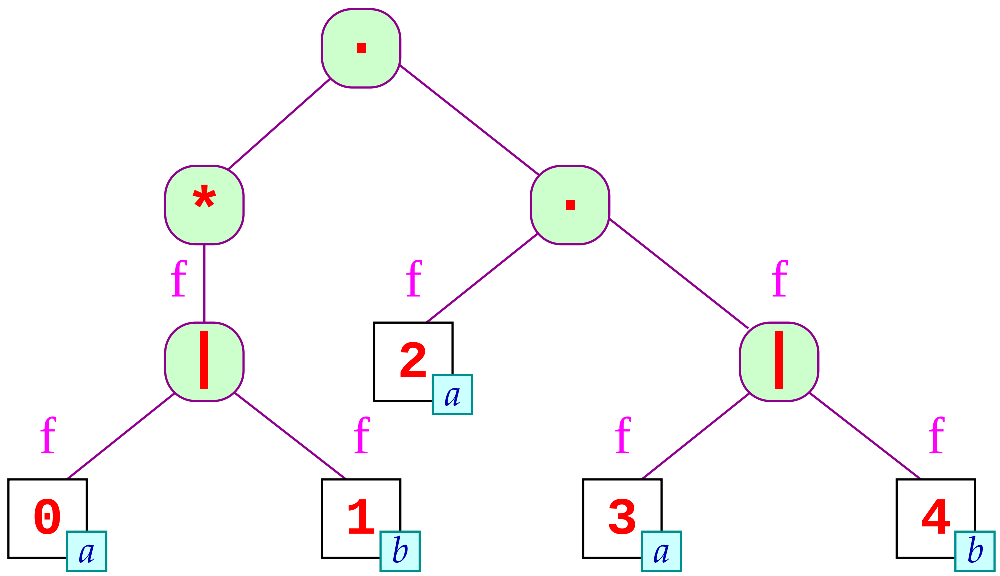
$$\mathsf{empty}[r] = t \quad \text{gdw.} \quad \epsilon \in [\![r]\!]$$

... im Beispiel:



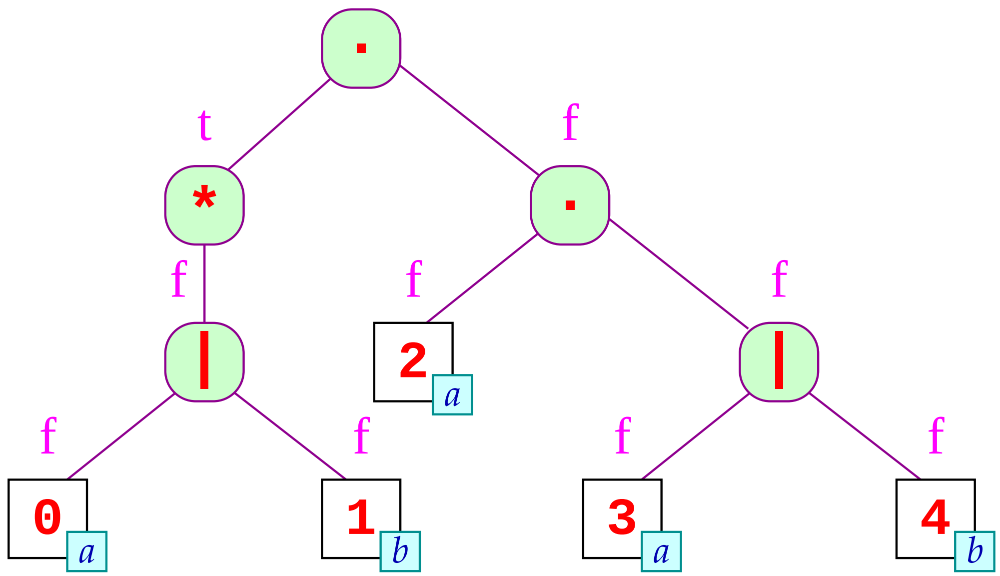
1. Schritt: $\quad \mathsf{empty}[r] = t \quad$ gdw. $\quad \epsilon \in [\![r]\!]$

... im Beispiel:



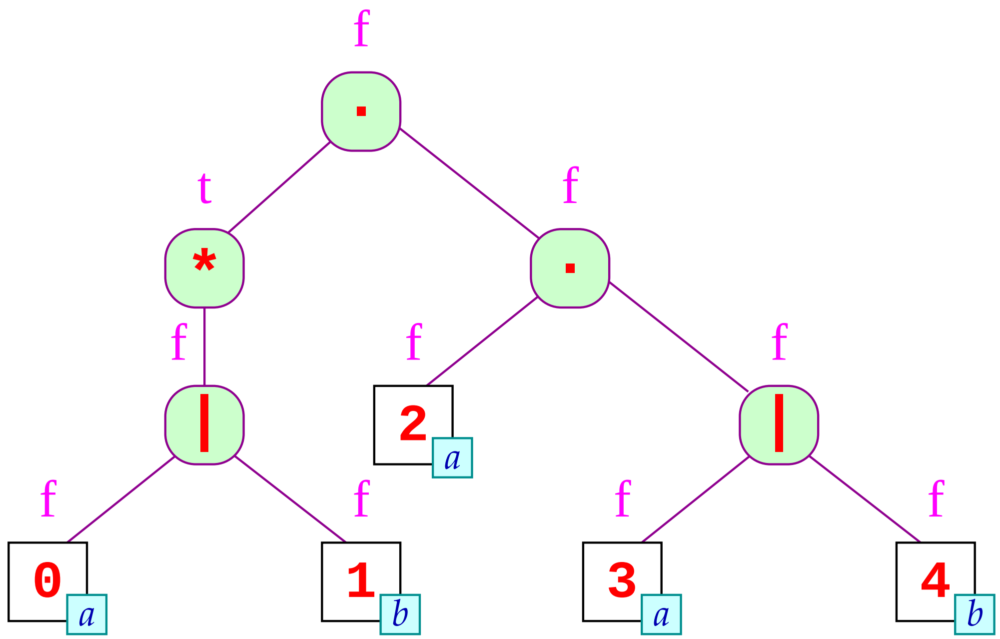
1. Schritt: $\mathsf{empty}[r] = t \quad \text{gdw.} \quad \epsilon \in [\![r]\!]$

... im Beispiel:



1. Schritt: $\mathsf{empty}[r] = t$ gdw. $\epsilon \in [\![r]\!]$

... im Beispiel:



Implementierung: DFS post-order Traversierung

Für Blätter $r \equiv \boxed{i \mid x}$ ist $\mathsf{empty}[r] = (x \equiv \epsilon)$.

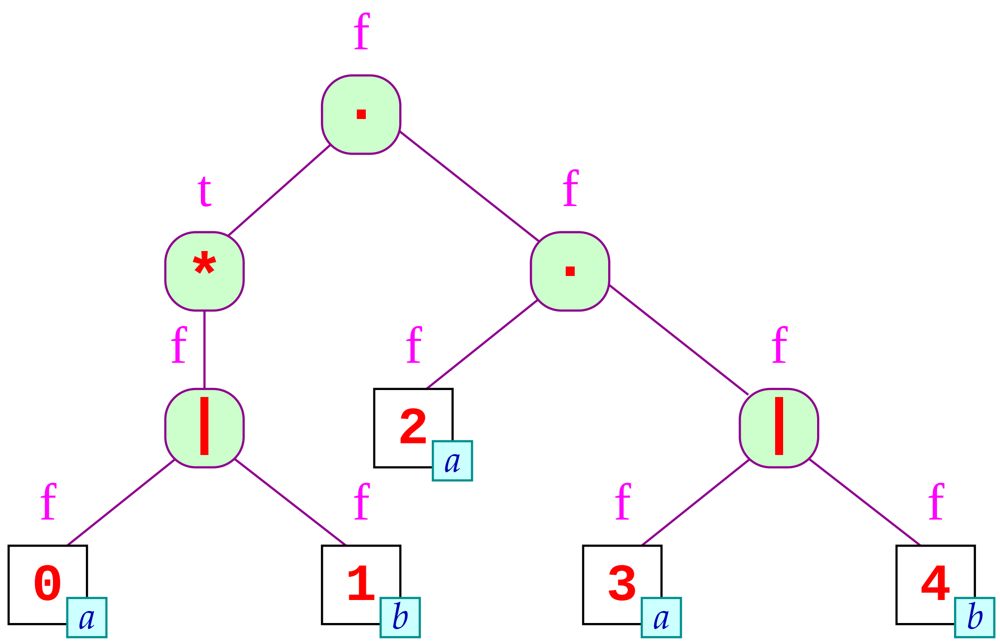
Andernfalls:

$$
\begin{array}{lll}
\mathsf{empty}[r_1 \mid r_2] & = & \mathsf{empty}[r_1] \vee \mathsf{empty}[r_2] \\
\mathsf{empty}[r_1 \cdot r_2] & = & \mathsf{empty}[r_1] \wedge \mathsf{empty}[r_2] \\
\mathsf{empty}[r_1^*] & = & t \\
\mathsf{empty}[r_1 ?] & = & t
\end{array}
$$

2. Schritt:

Die Menge erster Bätter: $\mathsf{first}[r] = \{i \text{ in } r \mid (\bullet r, \epsilon, \bullet \boxed{i \mid x}) \in \delta^*\}$

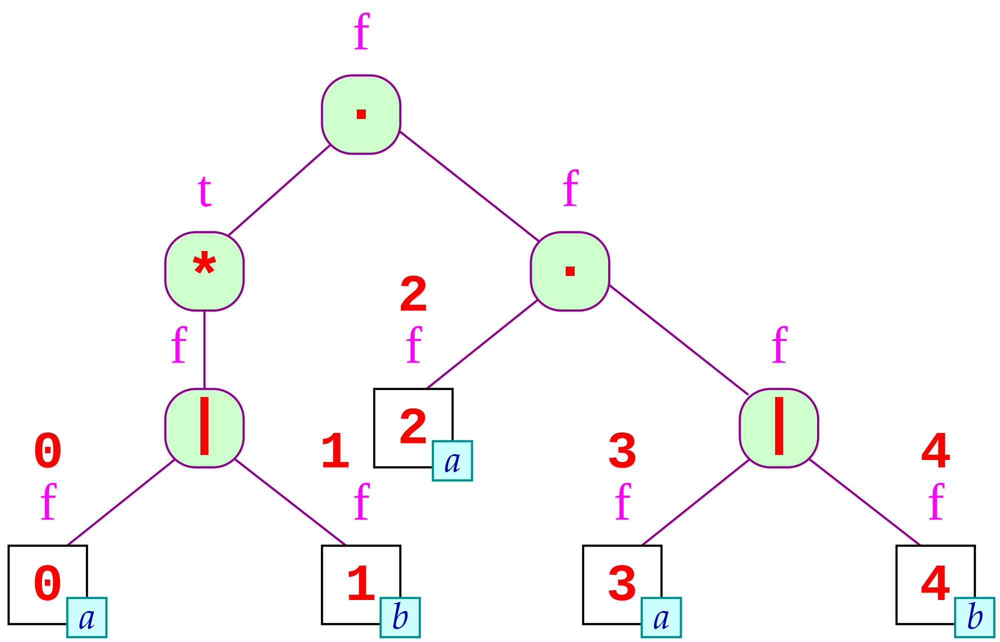
... im Beispiel:



2. Schritt:

Die Menge erster Bätter: $\mathsf{first}[r] = \{i \text{ in } r \mid (\bullet r, \epsilon, \bullet \boxed{i \mid x}) \in \delta^*\}$

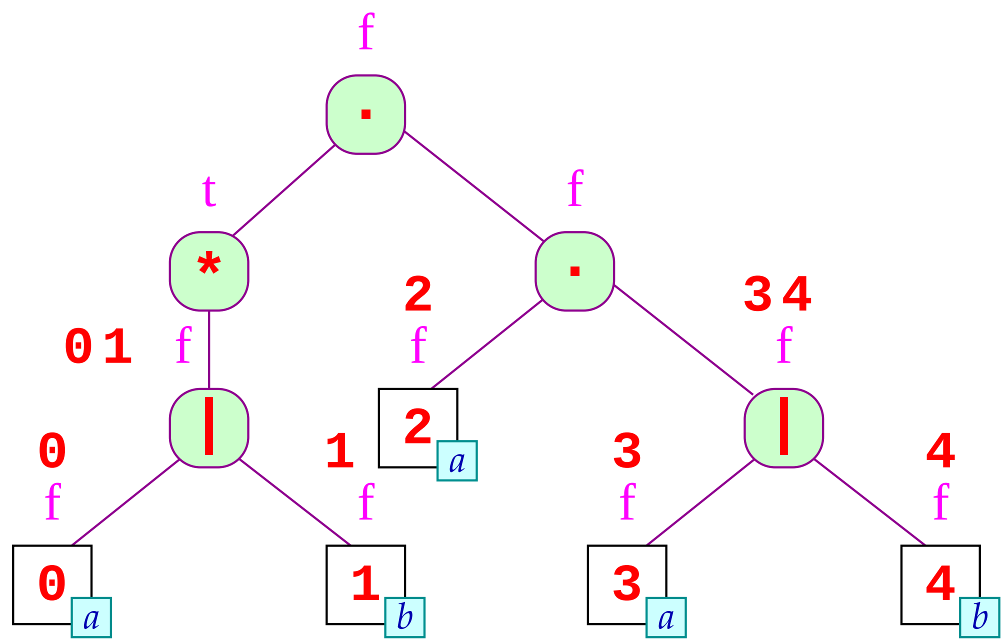
... im Beispiel:



2. Schritt:

Die Menge erster Bätter: $\mathsf{first}[r] = \{i \text{ in } r \mid (\bullet r, \epsilon, \bullet\boxed{i \mid x}) \in \delta^*\}$

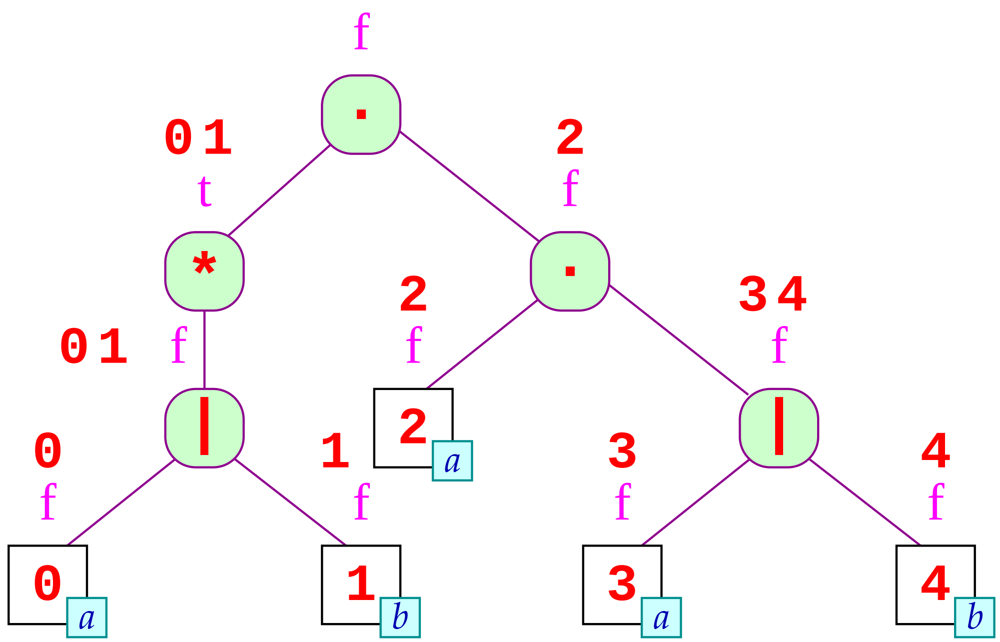
... im Beispiel:



2. Schritt:

Die Menge erster Bätter: $\mathsf{first}[r] = \{i \text{ in } r \mid (\bullet r, \epsilon, \bullet\boxed{i \mid x}) \in \delta^*\}$

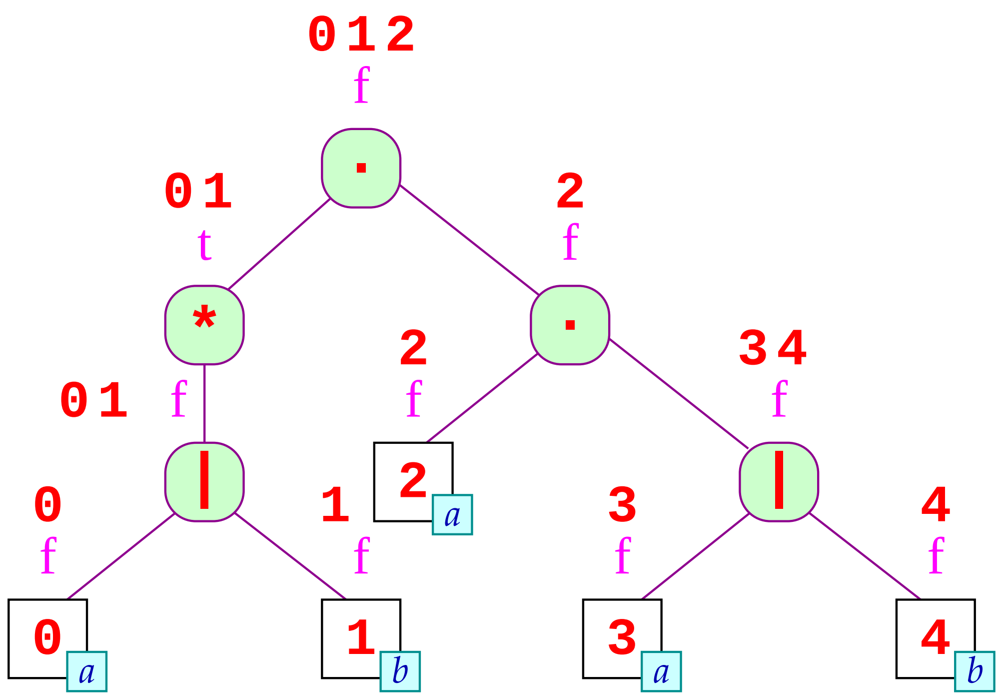
... im Beispiel:



2. Schritt:

Die Menge erster Bätter: $\text{first}[r] = \{i \text{ in } r \mid (\bullet r, \epsilon, \bullet\boxed{i \mid x}) \in \delta^*, x \neq \epsilon\}$

... im Beispiel:



Implementierung: DFS post-order Traversierung

Für Blätter $r \equiv \boxed{i \mid x}$ ist $\textsf{first}[r] = \{i \mid x \not\equiv \epsilon\}$.

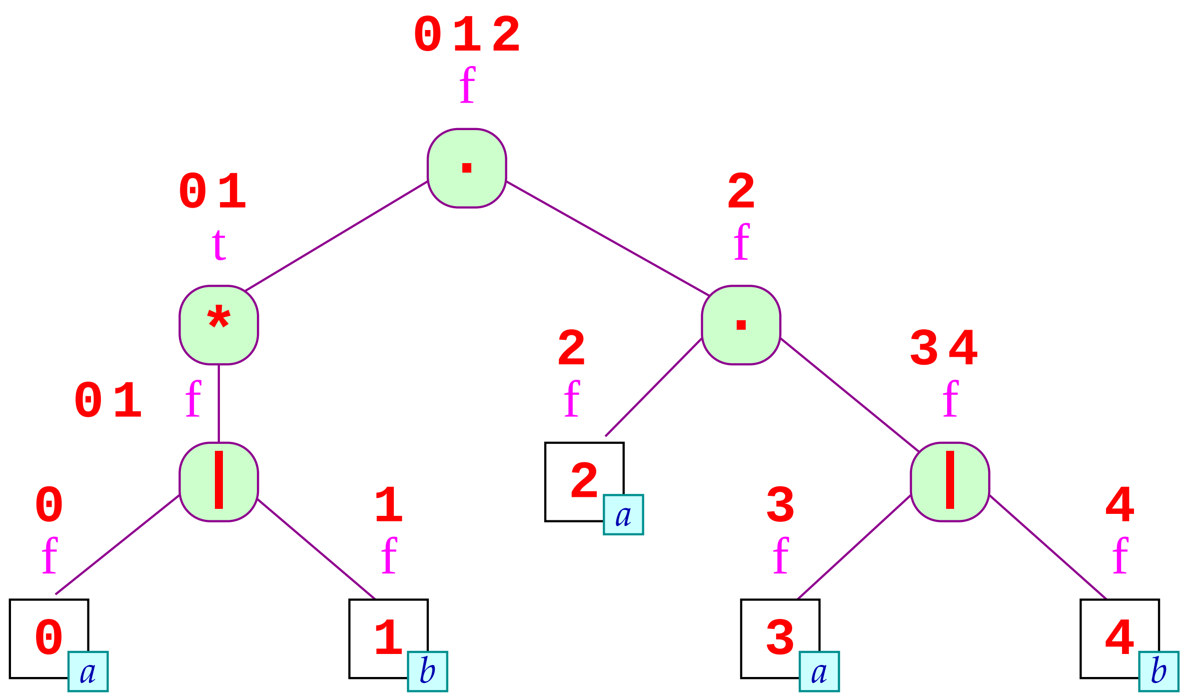
Andernfalls:

$$
\begin{array}{lll}
\textsf{first}[r_1 \mid r_2] & = & \textsf{first}[r_1] \cup \textsf{first}[r_2] \\
\textsf{first}[r_1 \cdot r_2] & = & \left\{ \begin{array}{lll} \textsf{first}[r_1] \cup \textsf{first}[r_2] & \text{falls} & \textsf{empty}[r_1] = t \\ \textsf{first}[r_1] & \text{falls} & \textsf{empty}[r_1] = f \end{array} \right. \\
\textsf{first}[r_1^*] & = & \textsf{first}[r_1] \\
\textsf{first}[r_1?] & = & \textsf{first}[r_1]
\end{array}
$$

3. Schritt:

Die Menge nächster Bätter: $\mathsf{next}[r] = \{ i \mid (r\bullet, \epsilon, \bullet\boxed{i \mid x}) \in \delta^* \}$

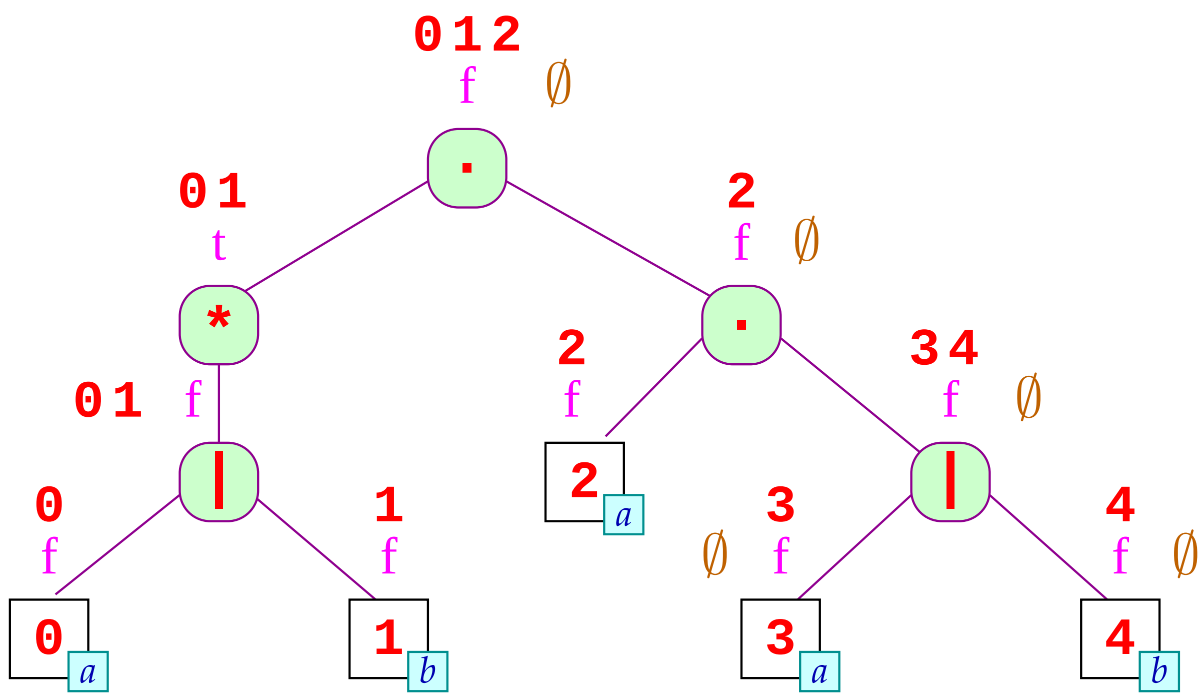
... im Beispiel:



3. Schritt:

Die Menge nächster Bätter: $\textsf{next}[r] = \{i \mid (r\bullet, \epsilon, \bullet\boxed{i \mid x}) \in \delta^*\}$

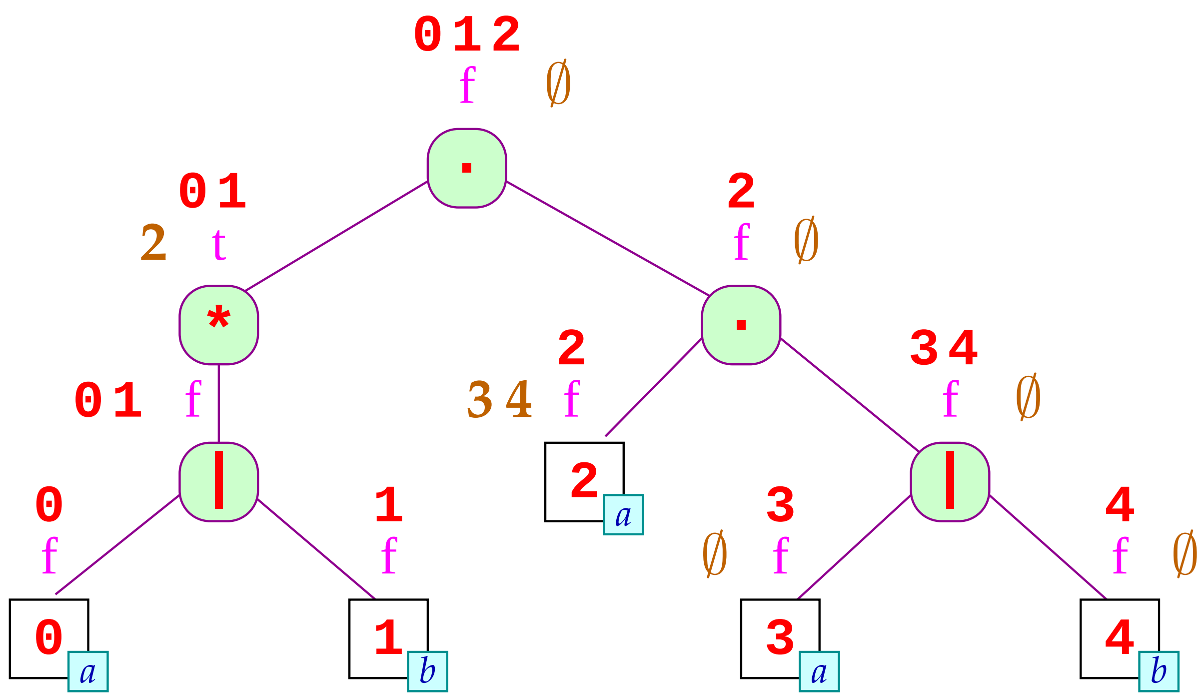
... im Beispiel:



3. Schritt:

Die Menge nächster Bätter: $\text{next}[r] = \{i \mid (r\bullet, \epsilon, \bullet\boxed{i \mid x}) \in \delta^*\}$

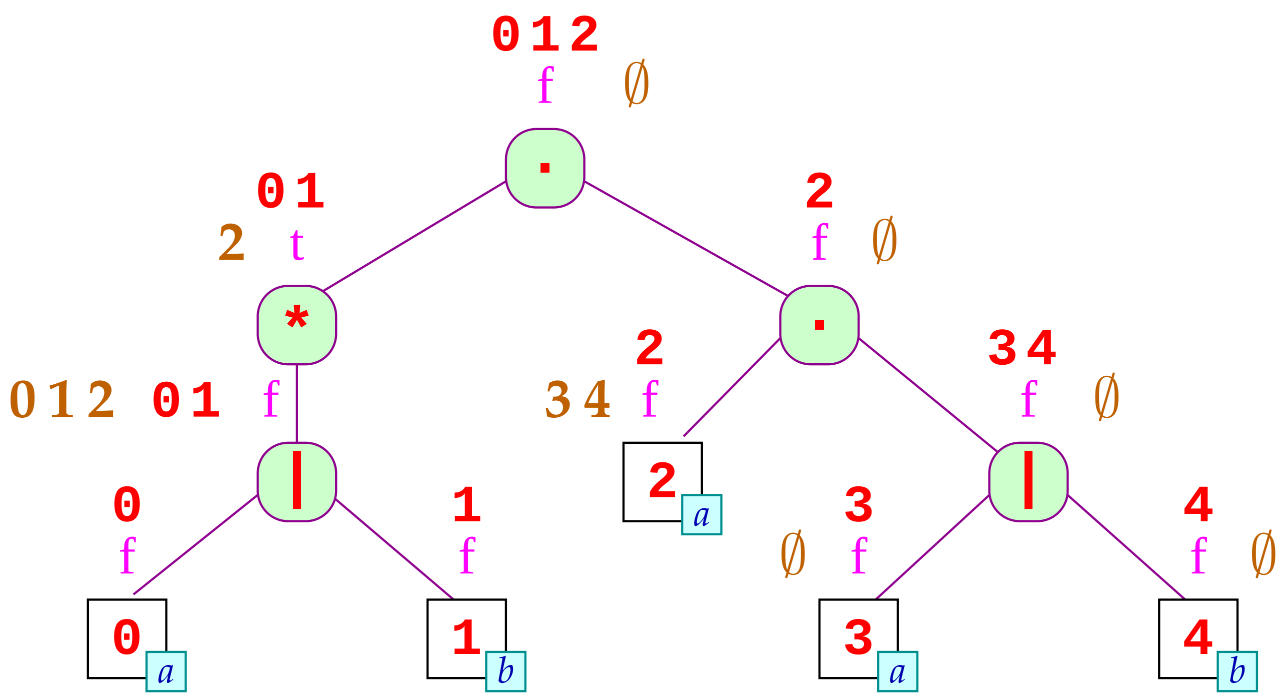
... im Beispiel:



3. Schritt:

Die Menge nächster Bätter: $\text{next}[r] = \{i \mid (r\bullet, \epsilon, \bullet\boxed{i \mid x}) \in \delta^*\}$

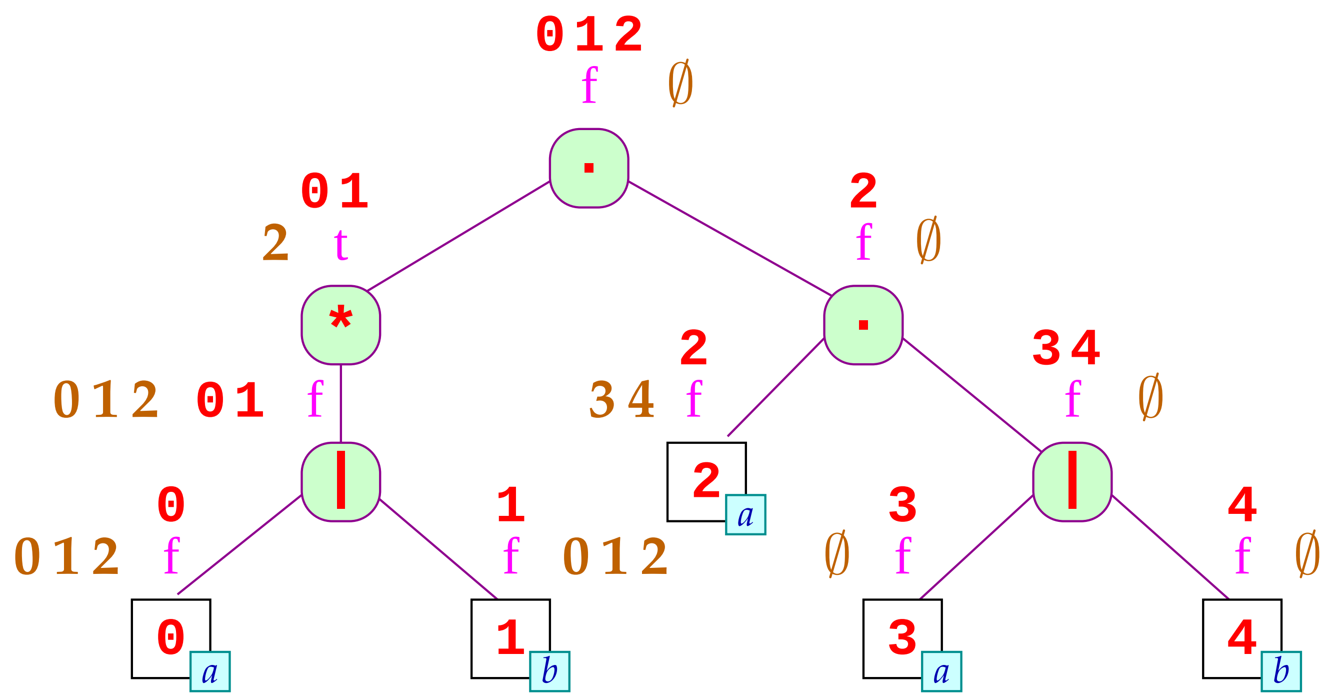
... im Beispiel:



3. Schritt:

Die Menge nächster Bätter: $\qquad \mathsf{next}[r] = \{i \mid (r\bullet, \epsilon, \bullet\boxed{i \mid x}) \in \delta^*\}$

... im Beispiel:



Implementierung: DFS pre-order Traversierung ;-)

Für die Wurzel haben wir:

$$\mathsf{next}[e] = \emptyset$$

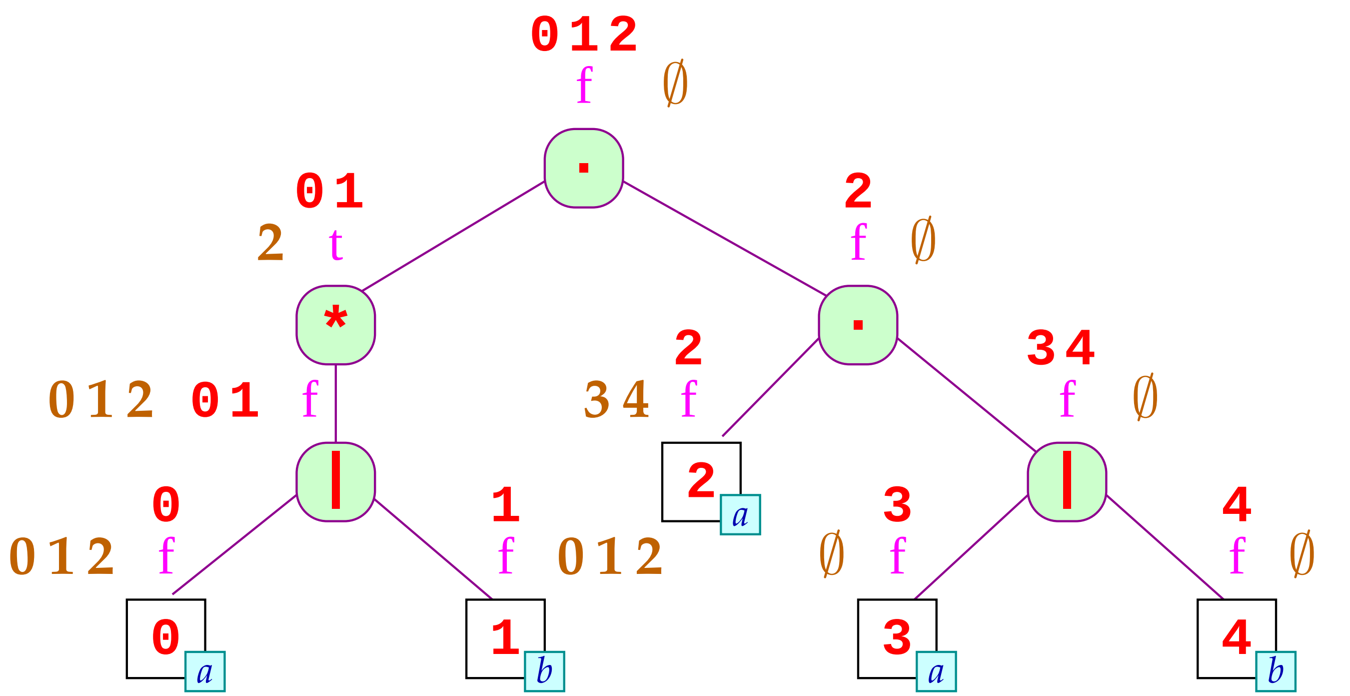
Ansonsten machen wir eine Fallunterscheidung über den Kontext:

r	Regeln
$r_1 \mid r_2$	$\mathsf{next}[r_1] = \mathsf{next}[r]$ $\mathsf{next}[r_2] = \mathsf{next}[r]$
$r_1 \cdot r_2$	$\mathsf{next}[r_1] = \begin{cases} \mathsf{first}[r_2] \cup \mathsf{next}[r] & \text{falls} \quad \mathsf{empty}[r_2] = t \\ \mathsf{first}[r_2] & \text{falls} \quad \mathsf{empty}[r_2] = f \end{cases}$ $\mathsf{next}[r_2] = \mathsf{next}[r]$
r_1^*	$\mathsf{next}[r_1] = \mathsf{first}[r_1] \cup \mathsf{next}[r]$
$r_1?$	$\mathsf{next}[r_1] = \mathsf{next}[r]$

4. Schritt:

Die Menge **letzter** Bätter: $\text{last}[r] = \{i \text{ in } r \mid (\boxed{i \mid x}\bullet, \epsilon, r\bullet) \in \delta^*\}$

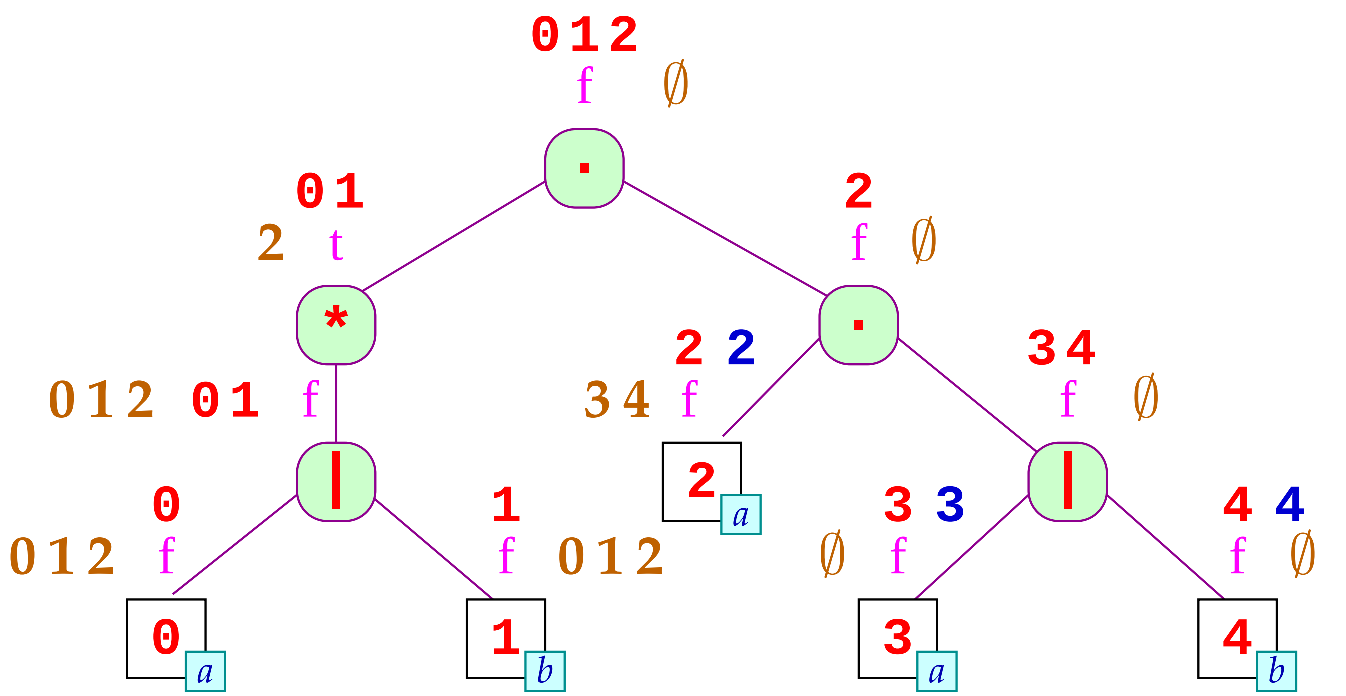
... im Beispiel:



4. Schritt:

Die Menge **letzter** Bätter: $\text{last}[r] = \{i \text{ in } r \mid (\boxed{i \mid x}\bullet, \epsilon, r\bullet) \in \delta^*\}$

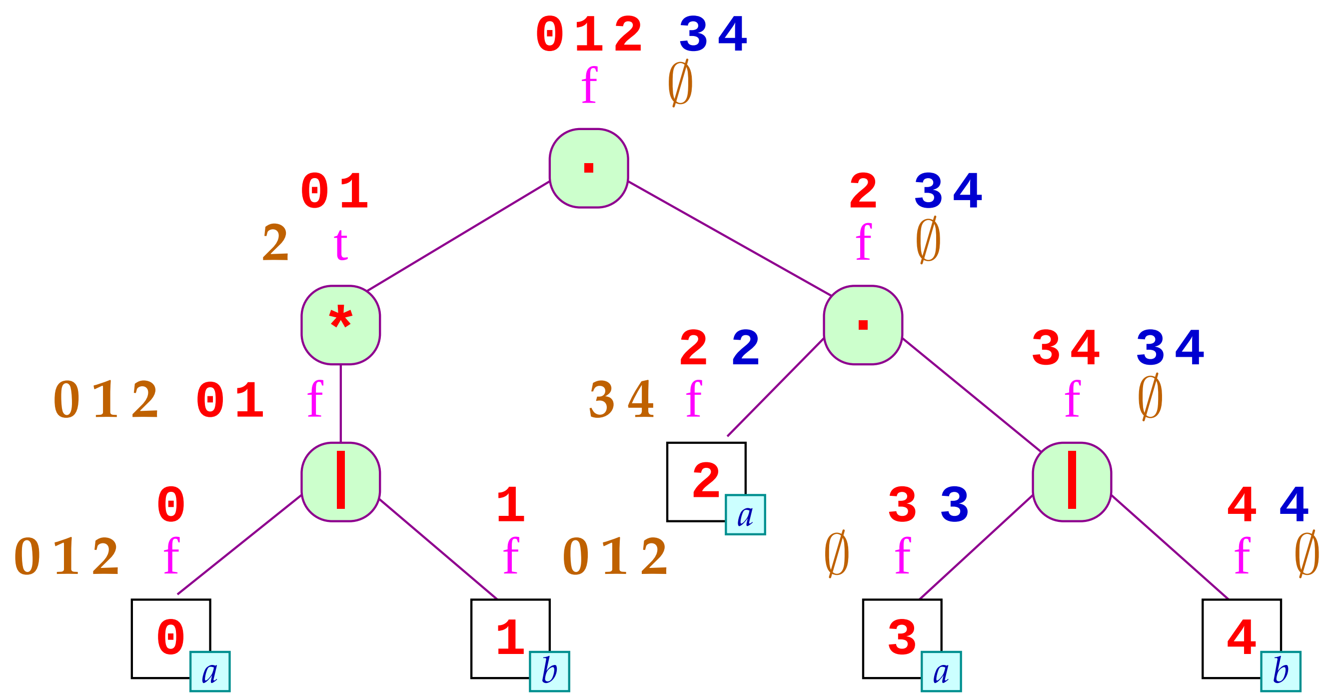
... im Beispiel:



4. Schritt:

Die Menge letzter Bätter: $\text{last}[r] = \{i \text{ in } r \mid (\boxed{i \mid x}\bullet, \epsilon, r\bullet) \in \delta^*, x \neq \epsilon\}$

... im Beispiel:



Implementierung: DFS post-order Traversierung :-)

Für Blätter $r \equiv \boxed{i \mid x}$ ist $\mathsf{last}[r] = \{i \mid x \not\equiv \epsilon\}$.

Andernfalls:

$$
\begin{array}{lll}
\mathsf{last}[r_1 \mid r_2] & = & \mathsf{last}[r_1] \cup \mathsf{last}[r_2] \\
\mathsf{last}[r_1 \cdot r_2] & = & \begin{cases} \mathsf{last}[r_1] \cup \mathsf{last}[r_2] & \text{falls} \quad \mathsf{empty}[r_2] = t \\ \mathsf{last}[r_2] & \text{falls} \quad \mathsf{empty}[r_2] = f \end{cases} \\
\mathsf{last}[r_1^*] & = & \mathsf{last}[r_1] \\
\mathsf{last}[r_1?] & = & \mathsf{last}[r_1]
\end{array}
$$

Integration:

Zustände: $\{\bullet e\} \cup \{i\bullet \mid i \text{ Blatt}\}$

Startzustand: $\bullet e$

Endzustände:

Falls $\mathsf{empty}[e] = f$, dann $\mathsf{last}[e]$. Andernfalls: $\{\bullet e\} \cup \mathsf{last}[e]$.

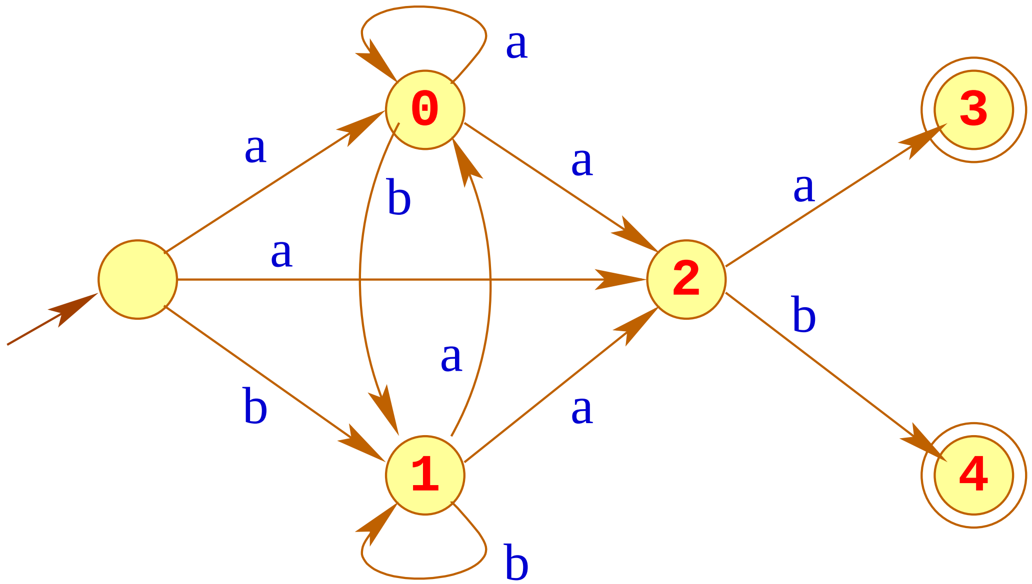
Übergänge:

$(\bullet e, a, i\bullet)$ falls $i \in \mathsf{first}[e]$ und i mit a beschriftet ist;

$(i\bullet, a, i'\bullet)$ falls $i' \in \mathsf{next}[i]$ und i' mit a beschriftet ist.

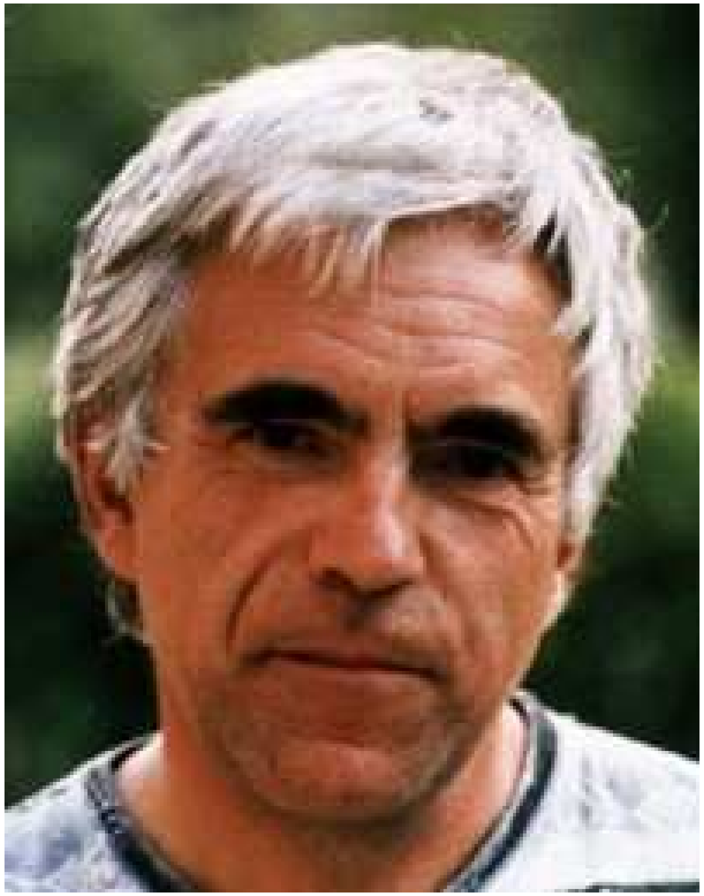
Den resultierenden Automaten bezeichnen wir mit A_e.

... im Beispiel:

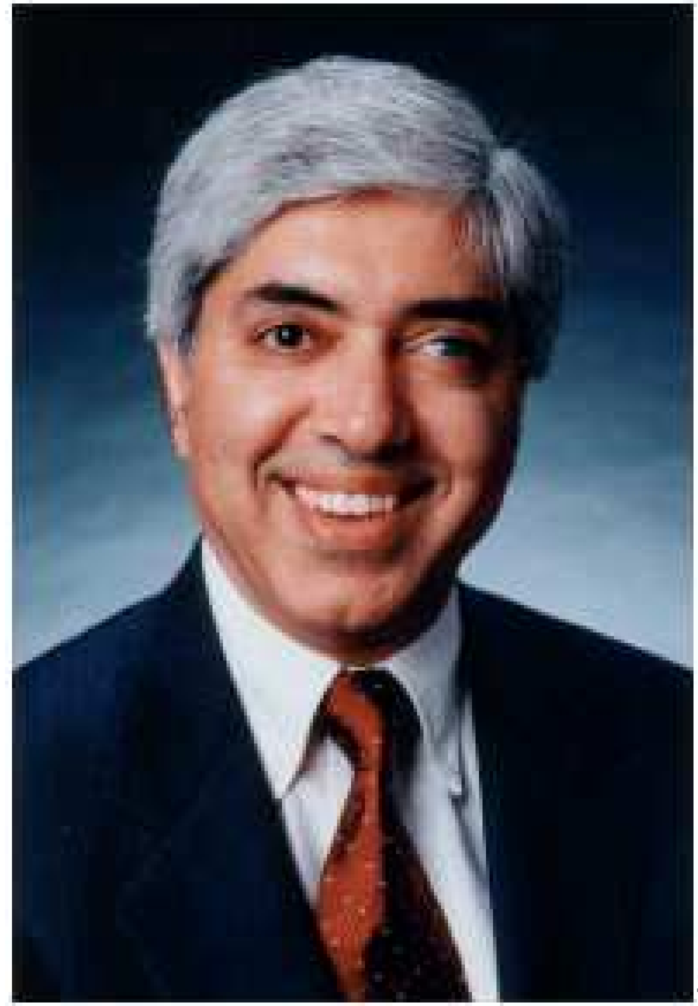


Bemerkung:

- Die Konstruktion heißt auch Berry-Sethi- oder Glushkow-Konstruktion.
- Sie wird in XML zur Definition von Content Models benutzt ;-)
- Das Ergebnis ist vielleicht nicht, was wir erwartet haben ...

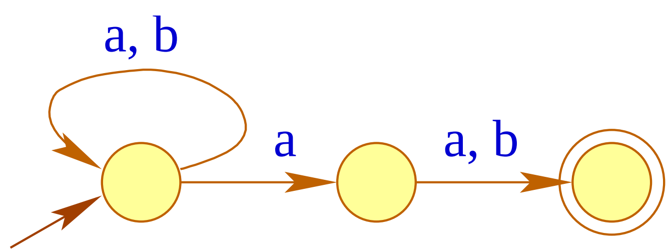


Gerard Berry, Esterel Technologies



Ravi Sethi, Research VR, Lucent Technologies

Der erwartete Automat:

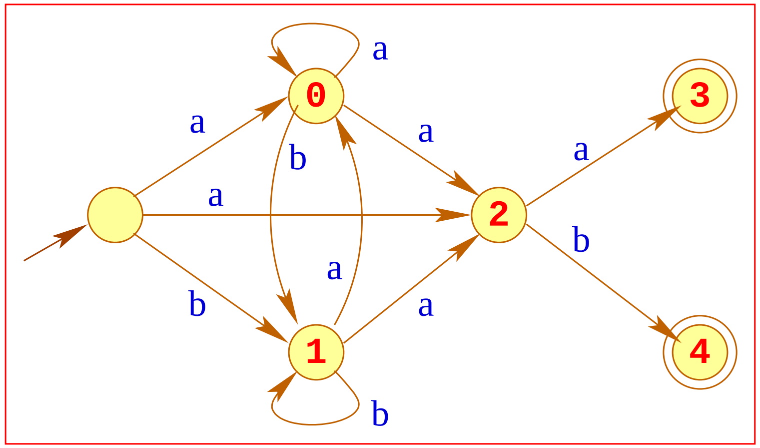


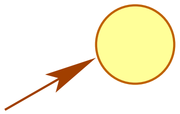
Bemerkung:

- in einen Zustand eingehende Kanten haben hier nicht unbedingt die gleiche Beschriftung :-)
- Dafür ist die Berry-Sethi-Konstruktion direkter ;-)
- In Wirklichkeit benötigen wir aber deterministische Automaten

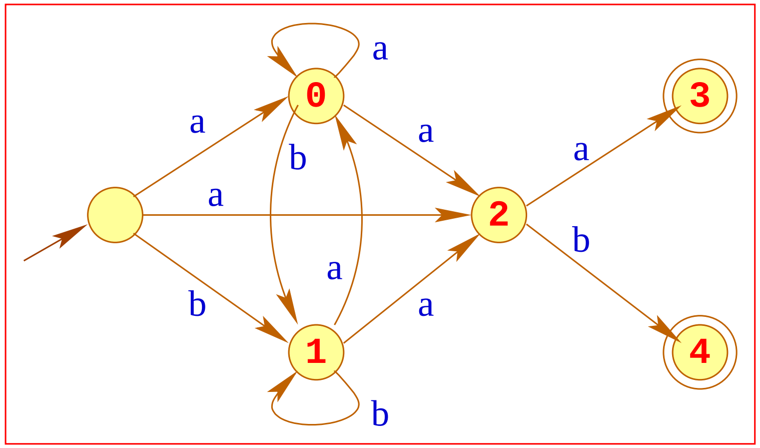
$\Longrightarrow$ Teilmengen-Konstruktion

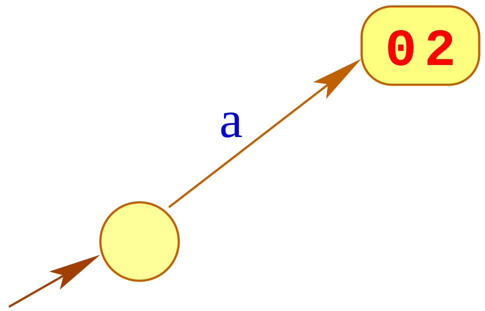
... im Beispiel:



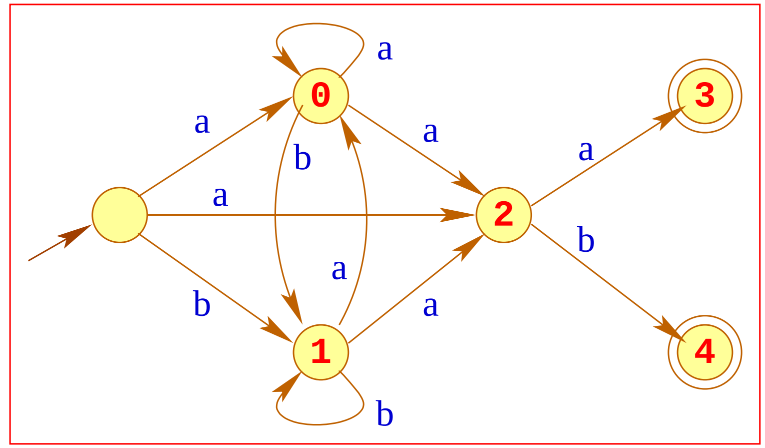


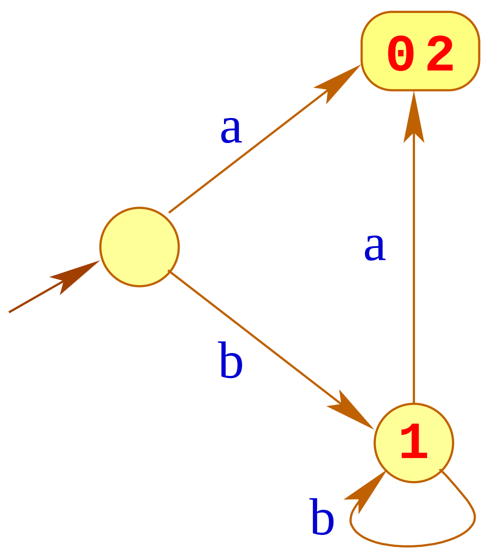
... im Beispiel:



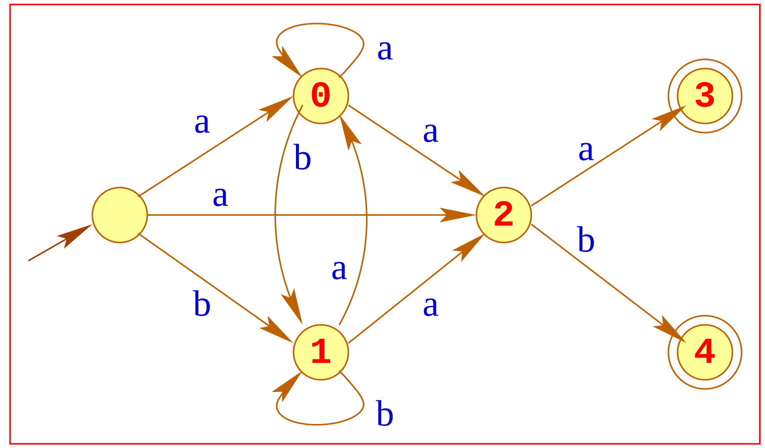


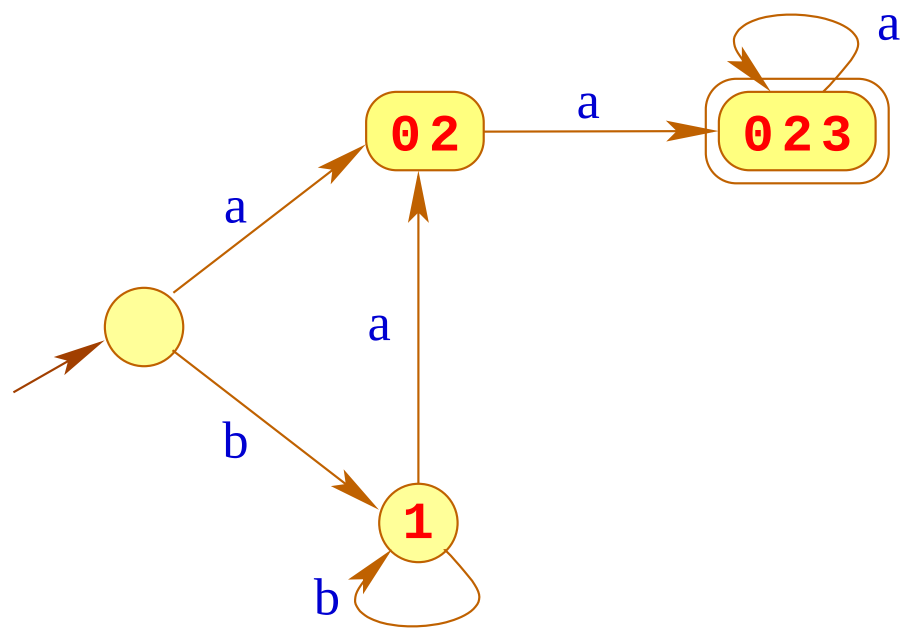
... im Beispiel:



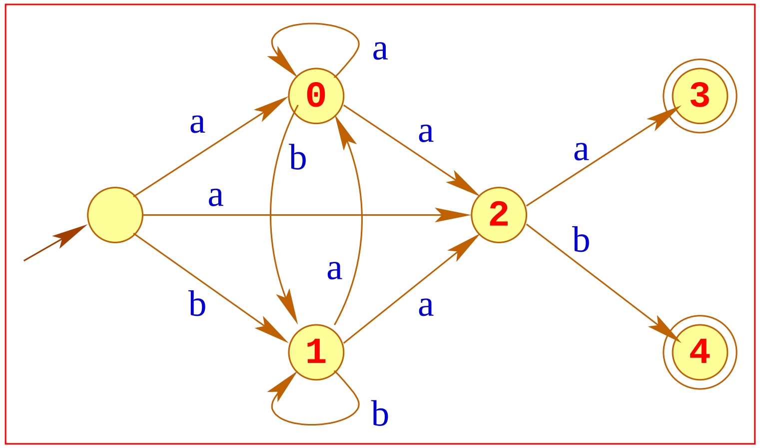


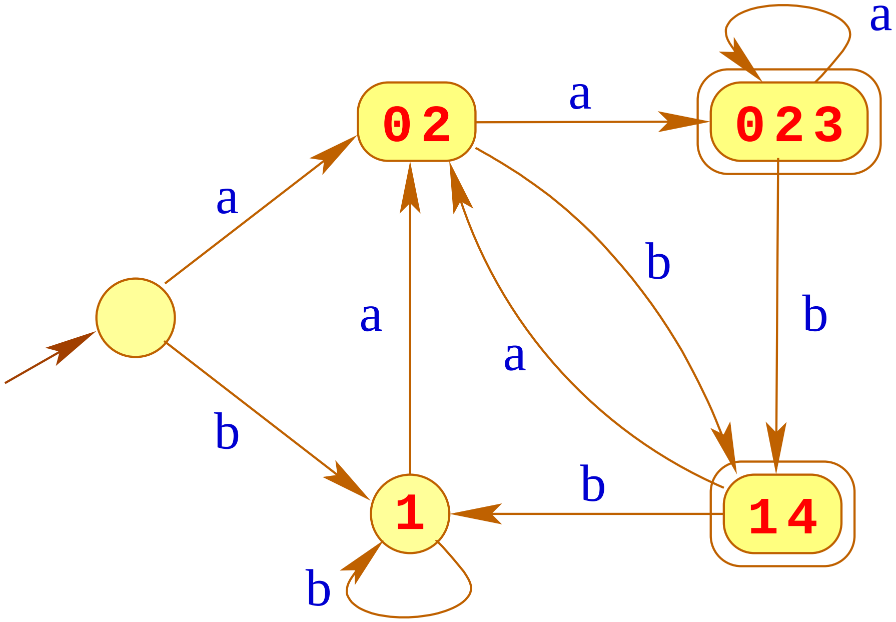
... im Beispiel:





... im Beispiel:





Satz:

Zu jedem nichtdeterministischen Automaten $A = (Q, \Sigma, \delta, I, F)$ kann ein deterministischer Automat $\mathcal{P}(A)$ konstruiert werden mit

$$\mathcal{L}(A) = \mathcal{L}(\mathcal{P}(A))$$

Satz:

Zu jedem nichtdeterministischen Automaten $A = (Q, \Sigma, \delta, I, F)$ kann ein deterministischer Automat $\mathcal{P}(A)$ konstruiert werden mit

$$\mathcal{L}(A) = \mathcal{L}(\mathcal{P}(A))$$

Konstruktion:

Zustände: Teilmengen von Q;

Anfangszustände: $\{I\}$;

Endzustände: $\{Q' \subseteq Q \mid Q' \cap F \neq \emptyset\}$;

Übergangsfunktion: $\delta_{\mathcal{P}}(Q', a) = \{q \in Q \mid \exists\, p \in Q' : \ (p, a, q) \in \delta\}$.

Achtung:

- Leider gibt es exponentiell viele Teilmengen von Q :-(
- Um nur nützliche Teilmengen zu betrachten, starten wir mit der Menge $Q_{\mathcal{P}} = \{I\}$ und fügen weitere Zustände nur nach Bedarf hinzu ...
- d.h., wenn wir sie von einem Zustand in $Q_{\mathcal{P}}$ aus erreichen können :-)
- Trotz dieser Optimierung kann der Ergebnisautomat riesig sein :-((

  ... was aber in der Praxis (so gut wie) nie auftritt :-))

Achtung:

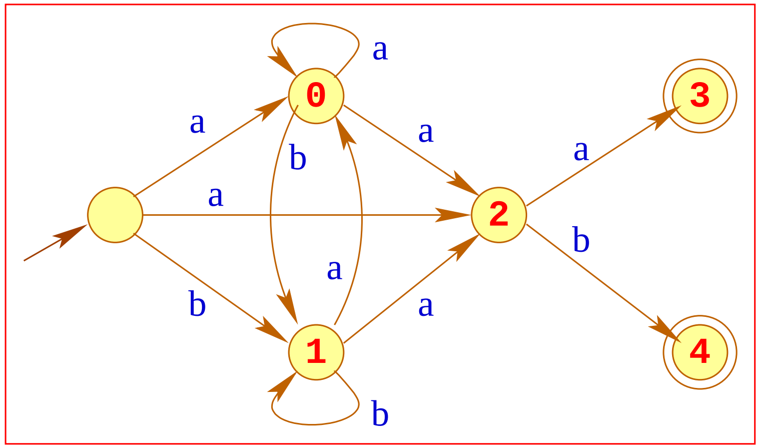
- Leider gibt es exponentiell viele Teilmengen von Q :-(
- Um nur nützliche Teilmengen zu betrachten, starten wir mit der Menge $Q_{\mathcal{P}} = \{I\}$ und fügen weitere Zustände nur nach Bedarf hinzu ...
- d.h., wenn wir sie von einem Zustand in $Q_{\mathcal{P}}$ aus erreichen können :-)
- Trotz dieser Optimierung kann der Ergebnisautomat riesig sein :-((

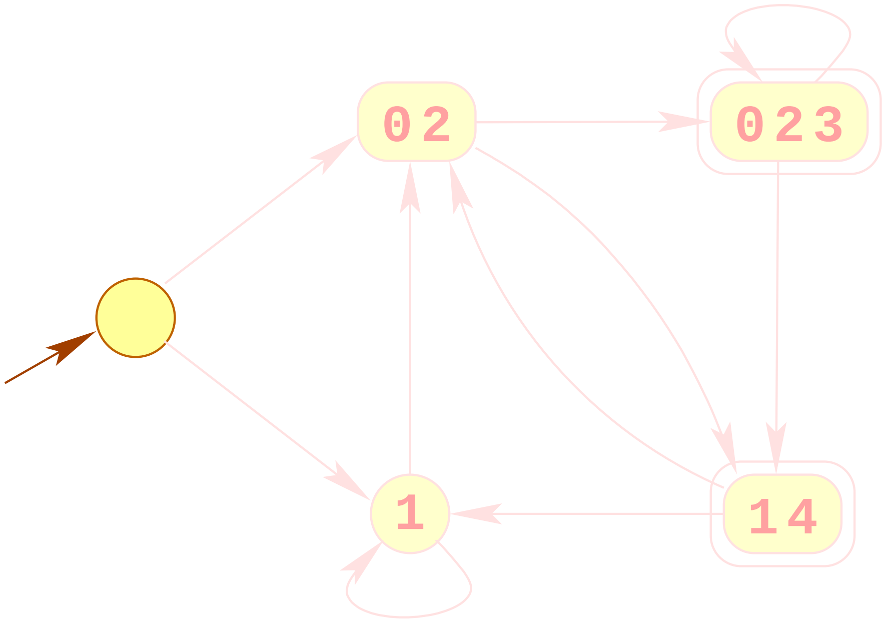
  ... was aber in der Praxis (so gut wie) nie auftritt :-))

- In Tools wie `grep` wird deshalb zu der DFA zu einem regulären Ausdruck nicht aufgebaut !!!
- Stattdessen werden während der Abbarbeitung der Eingabe genau die Mengen konstruiert, die für die Eingabe notwendig sind ...

... im Beispiel:

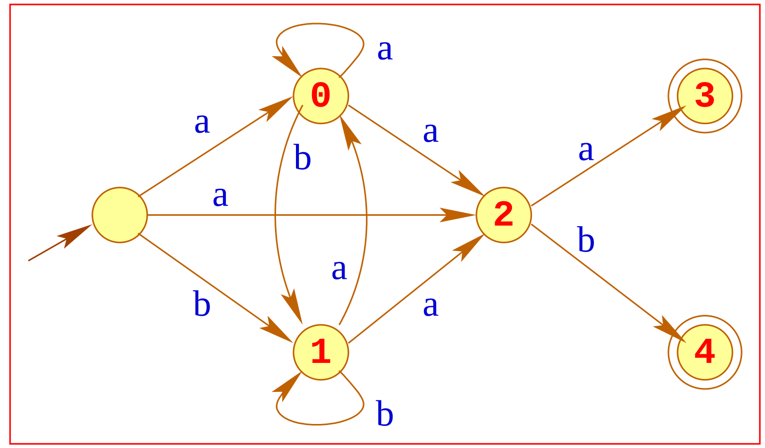
a	b	a	b

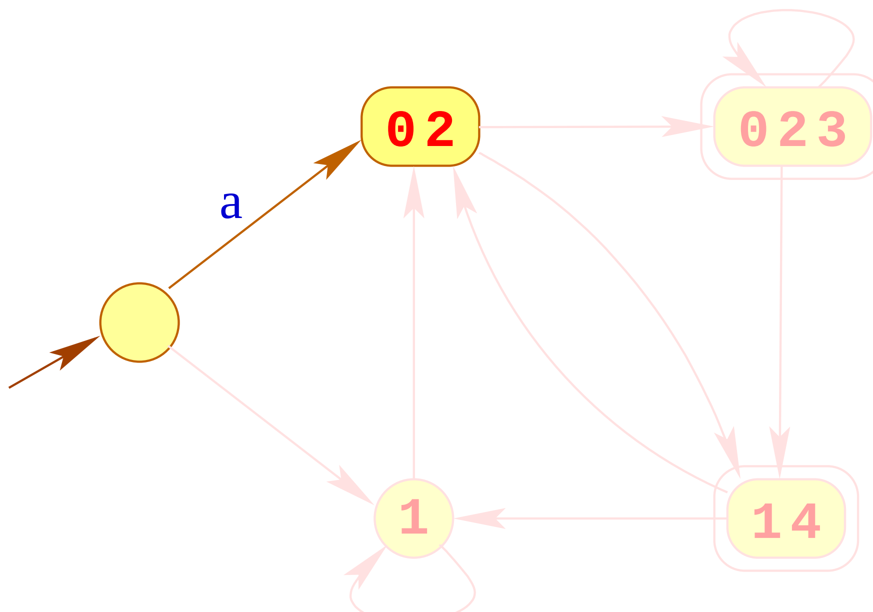




... im Beispiel:

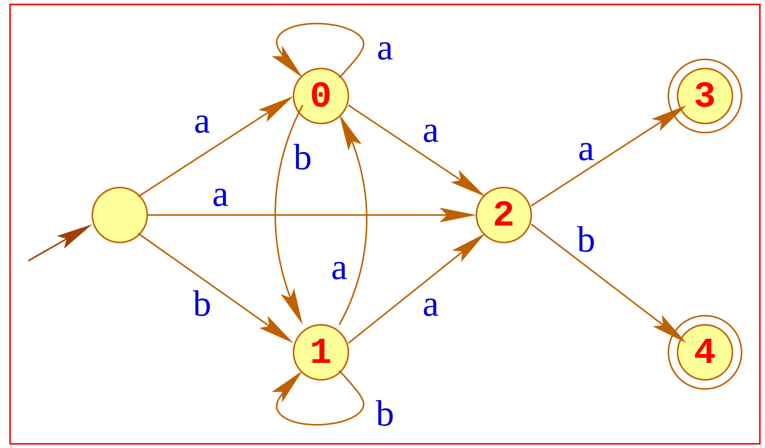
a	b	a	b

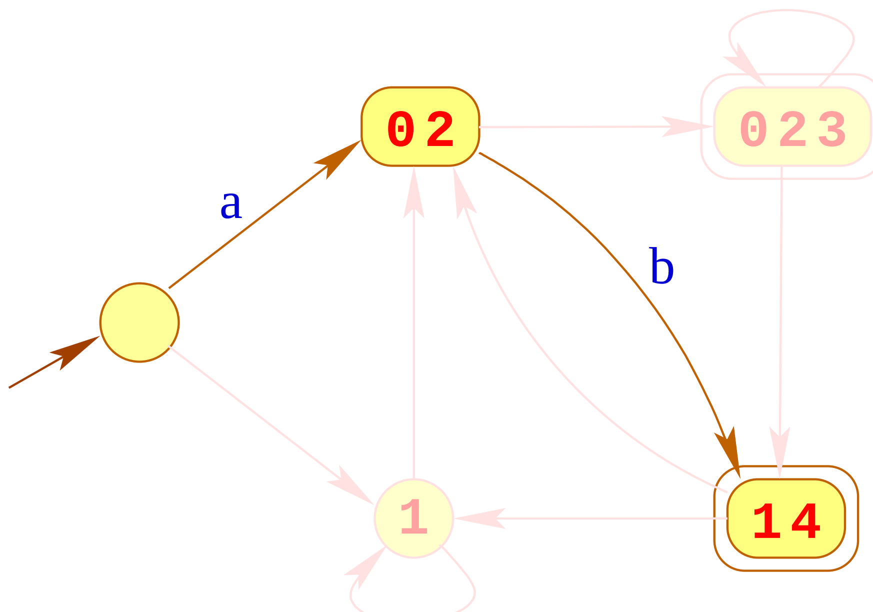




... im Beispiel:

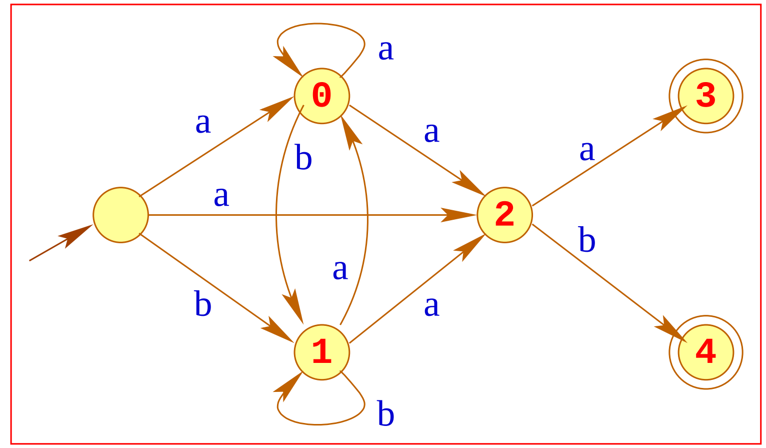
a	b	a	b

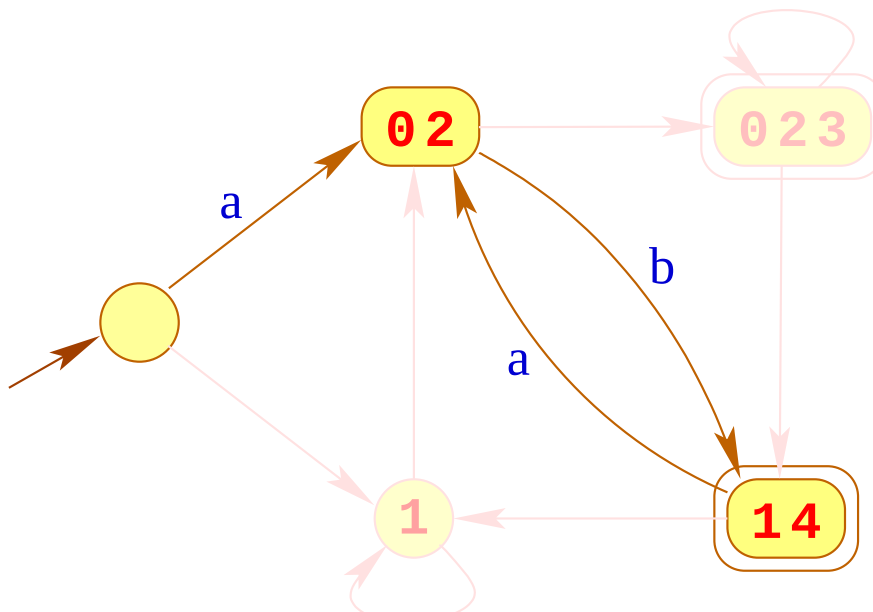




... im Beispiel:

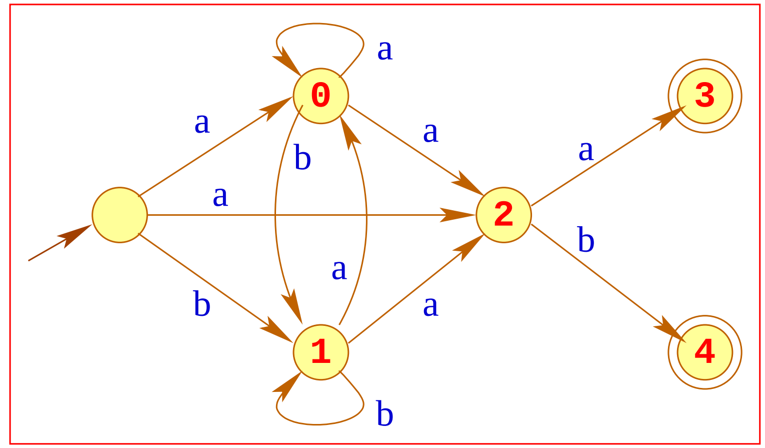
a	b	a	b

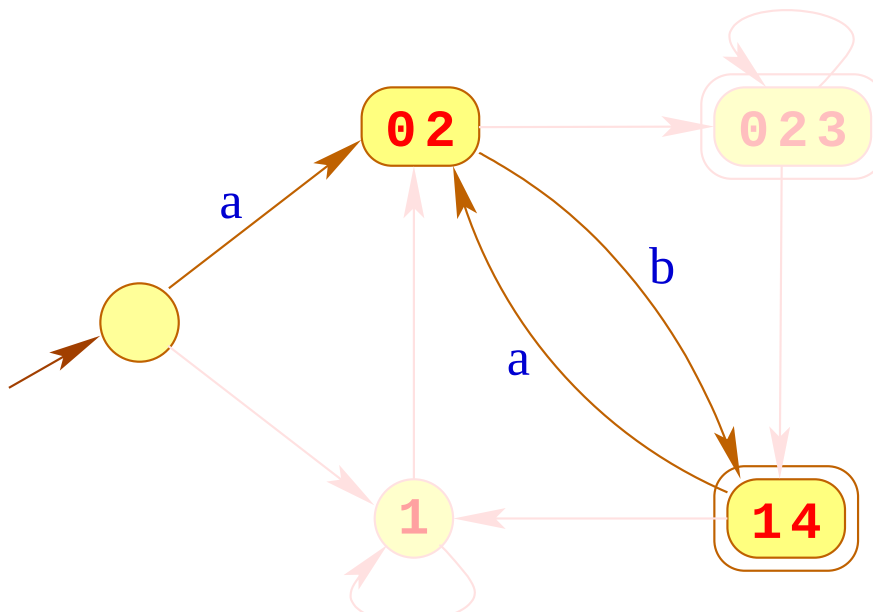




... im Beispiel:

a	b	a	b





Bemerkungen:

- Bei einem Eingabewort der Länge n werden maximal $\mathcal{O}(n)$ Mengen konstruiert :-)
- Ist eine Menge bzw. eine Kante des DFA einmal konstuiert, heben wir sie in einer Hash-Tabelle auf.
- Bevor wir einen neuen Übergang konstruieren, sehen wir erst nach, ob wir diesen nicht schon haben :-)

Bemerkungen:

- Bei einem Eingabewort der Länge n werden maximal $\mathcal{O}(n)$ Mengen konstruiert :-)
- Ist eine Menge bzw. eine Kante des DFA einmal konstuiert, heben wir sie in einer Hash-Tabelle auf.
- Bevor wir einen neuen Übergang konstruieren, sehen wir erst nach, ob wir diesen nicht schon haben :-)

Zusammen fassend finden wir:

Satz

Zu jedem regulären Ausdruck e kann ein deterministischer Automat $A = \mathcal{P}(A_e)$ konstruiert werden mit

$$\mathcal{L}(A) = [\![e]\!]$$