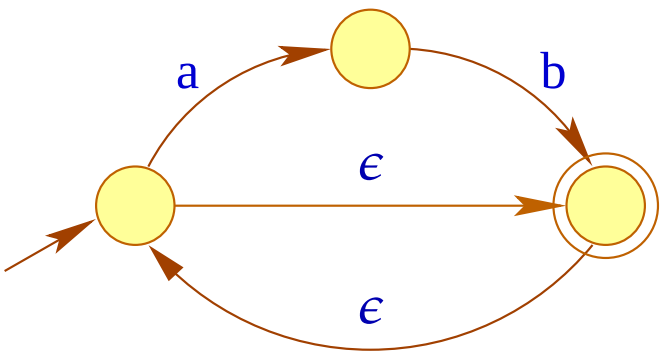
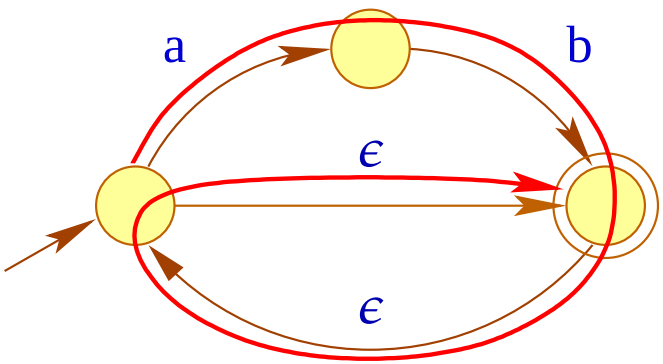
Akzeptierung

- Berechnungen sind Pfade im Graphen.
- akzeptierende Berechnungen führen von I nach F.
- Ein akzeptiertes Wort ist die Beschriftung eines akzeptierenden Pfades ...



Akzeptierung

- Berechnungen sind Pfade im Graphen.
- akzeptierende Berechnungen führen von I nach F.
- Ein akzeptiertes Wort ist die Beschriftung eines akzeptierenden Pfades ...



- Dazu definieren wir den transitiven Abschluss δ^* von δ als kleinste Menge δ' mit:

  $(p, \epsilon, p) \in \delta'$ und
  $(p, xw, q) \in \delta'$ sofern $(p, x, p_1) \in \delta$ und $(p_1, w, q) \in \delta'$.

  δ^* beschreibt für je zwei Zustände, mit welchen Wörtern man vom einen zum andern kommt :-)

- Die Menge aller akzeptierten Worte, d.h. die von A akzeptierte Sprache können wir kurz beschreiben als:

$$\mathcal{L}(A) = \{w \in \Sigma^* \mid \exists\, i \in I, f \in F : \quad (i, w, f) \in \delta^*\}$$

Satz:

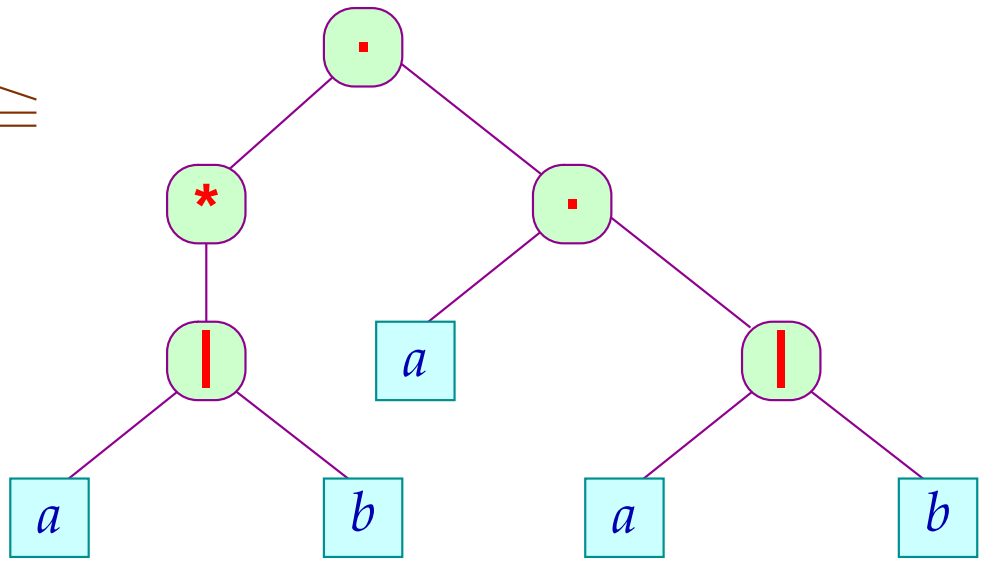
Für jeden regulären Ausdruck e kann (in linearer Zeit :-) ein ϵ-NFA konstruiert werden, der die Sprache $[\![e]\!]$ akzeptiert.

Idee:

Der Automat verfolgt (konzepionell mithilfe einer Marke "•"), wohin man in e mit der Eingabe w gelangen kann.

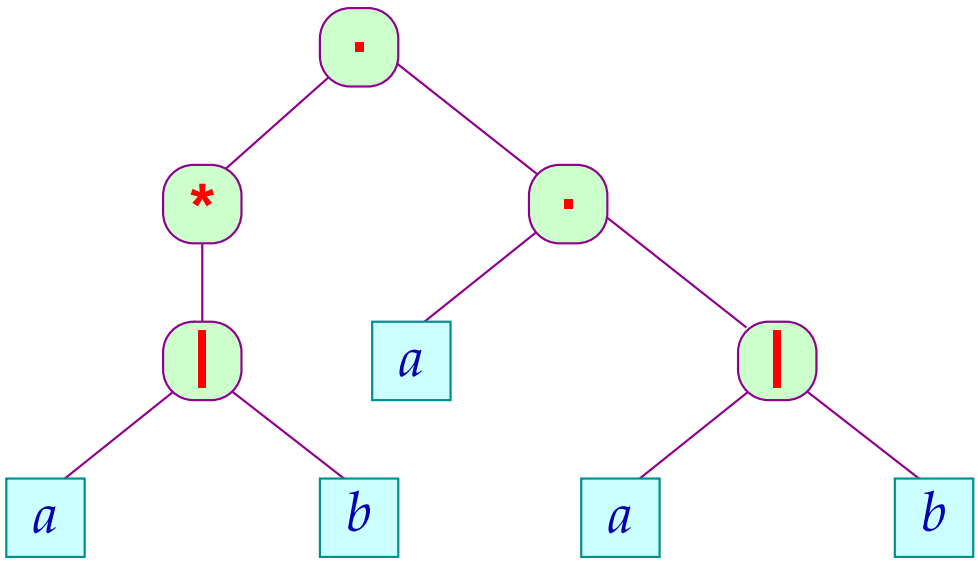
Beispiel:

$(a|b)^*a(a|b)$ $\hat{=}$



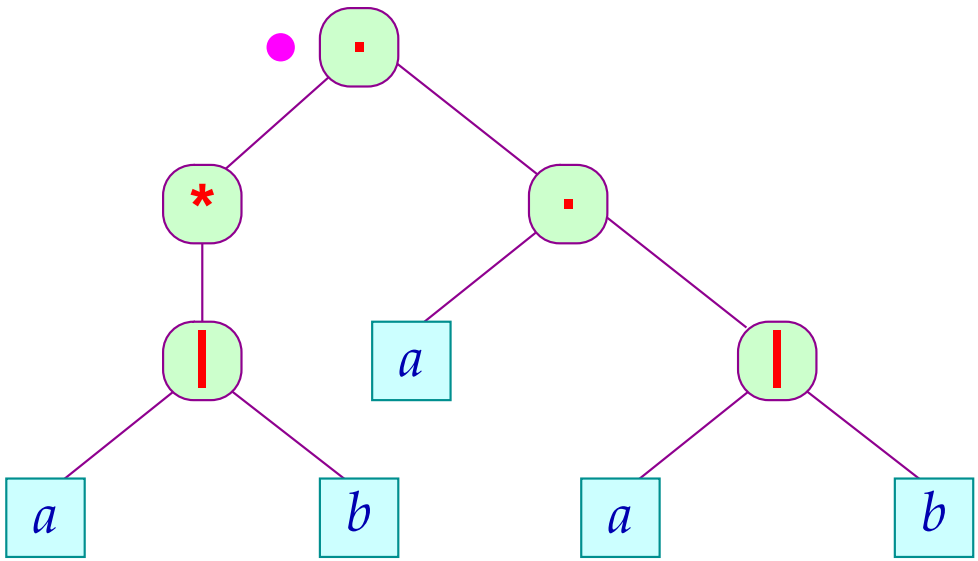
Beispiel:

$w = bbaa$:



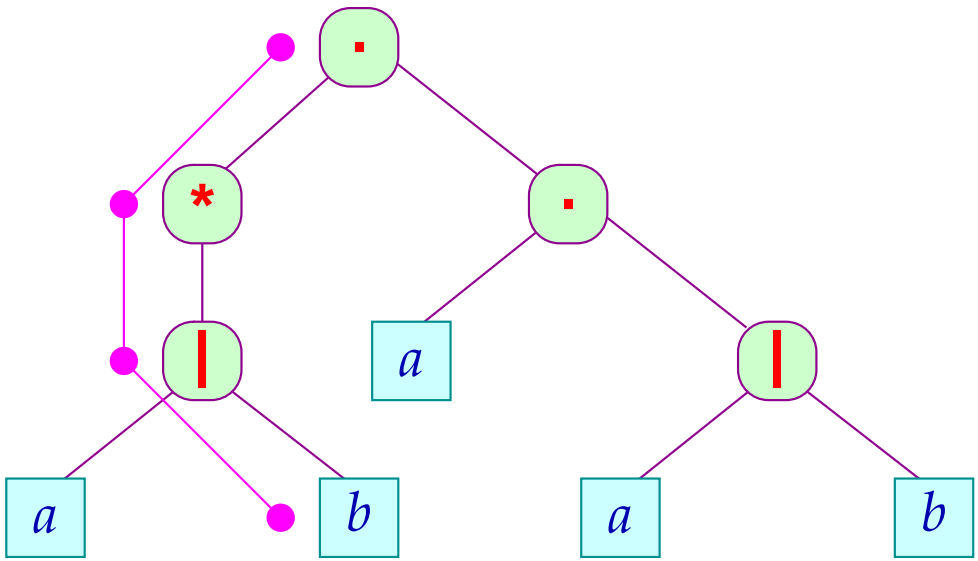
Beispiel:

$w = bbaa$:



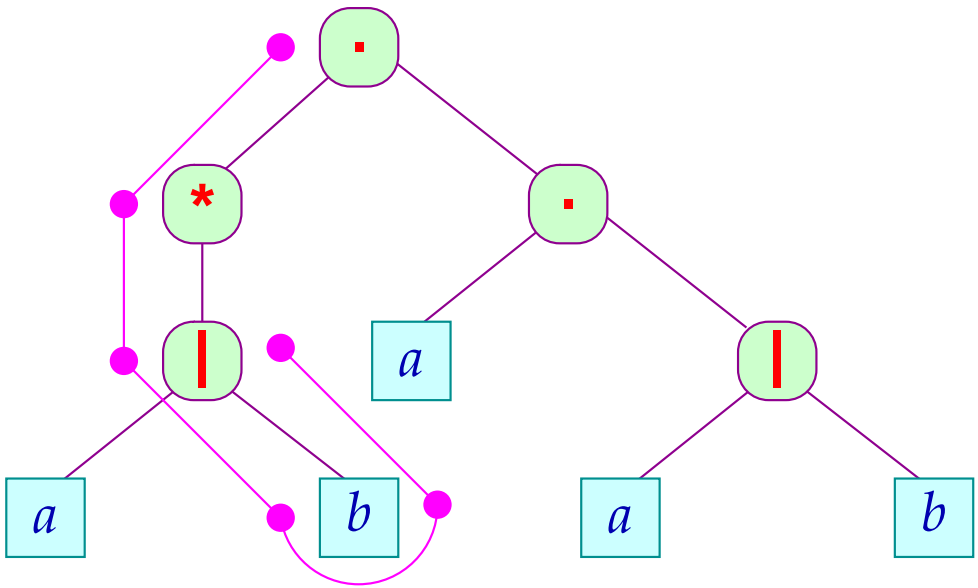
Beispiel:

$w = bbaa$:



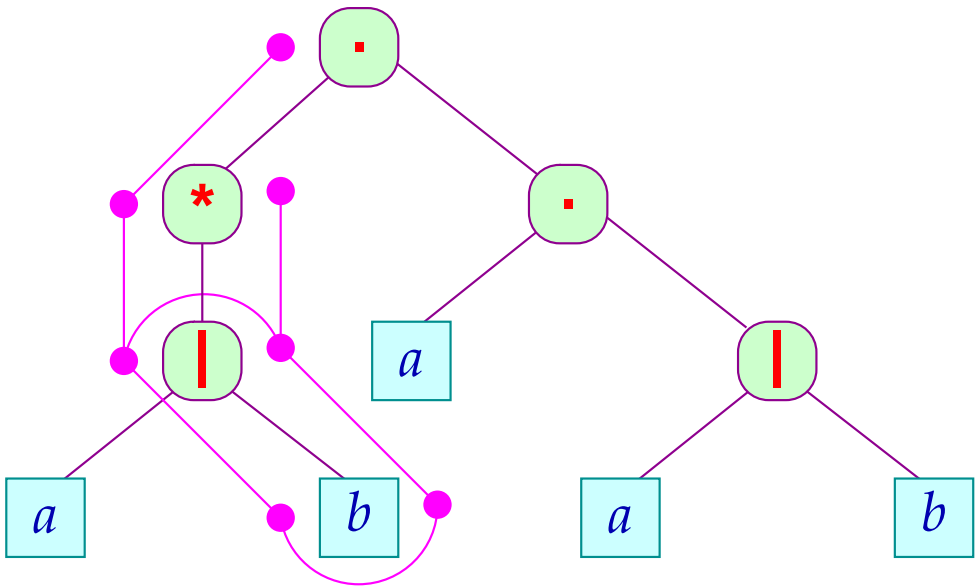
Beispiel:

$w = bbaa$:



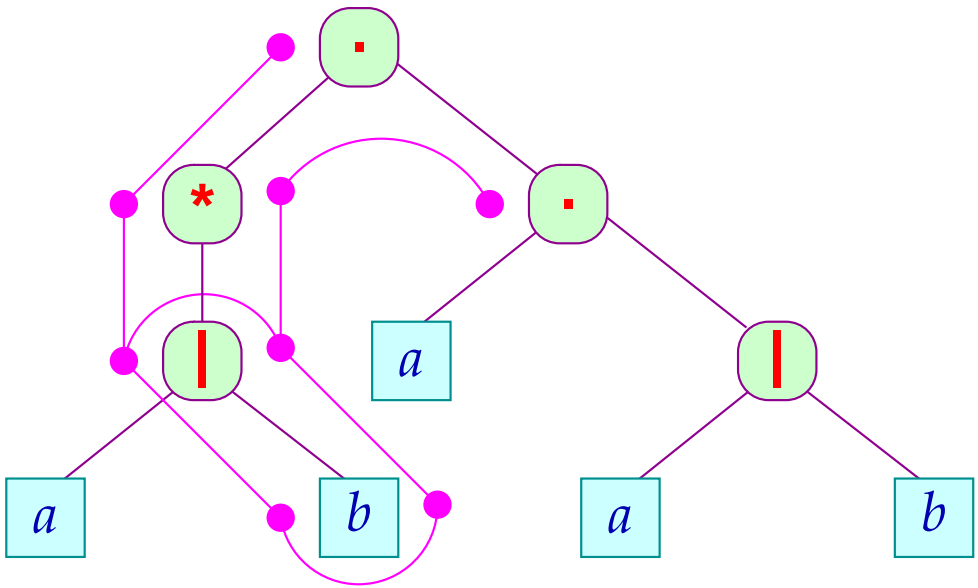
Beispiel:

$w = bbaa$:



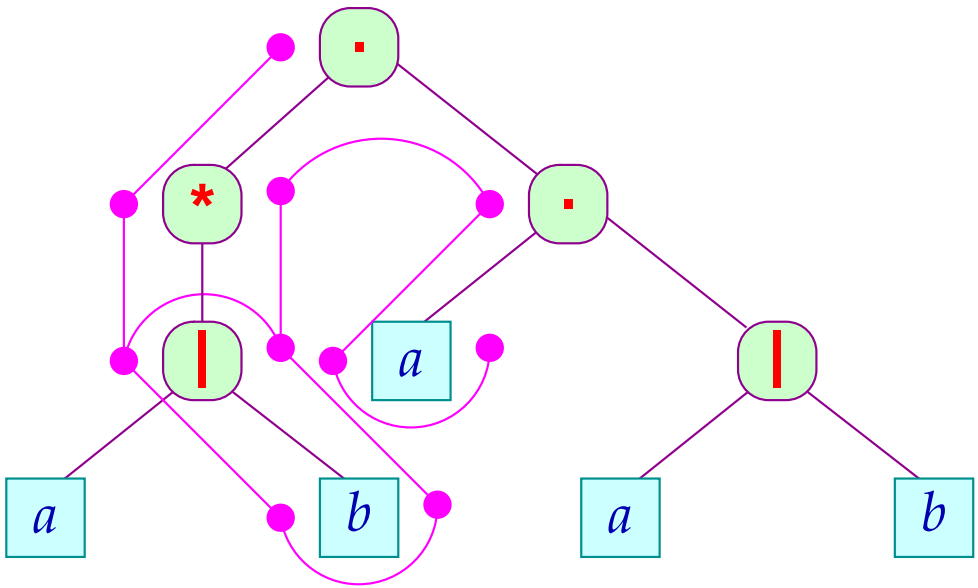
Beispiel:

$w = bbaa$:



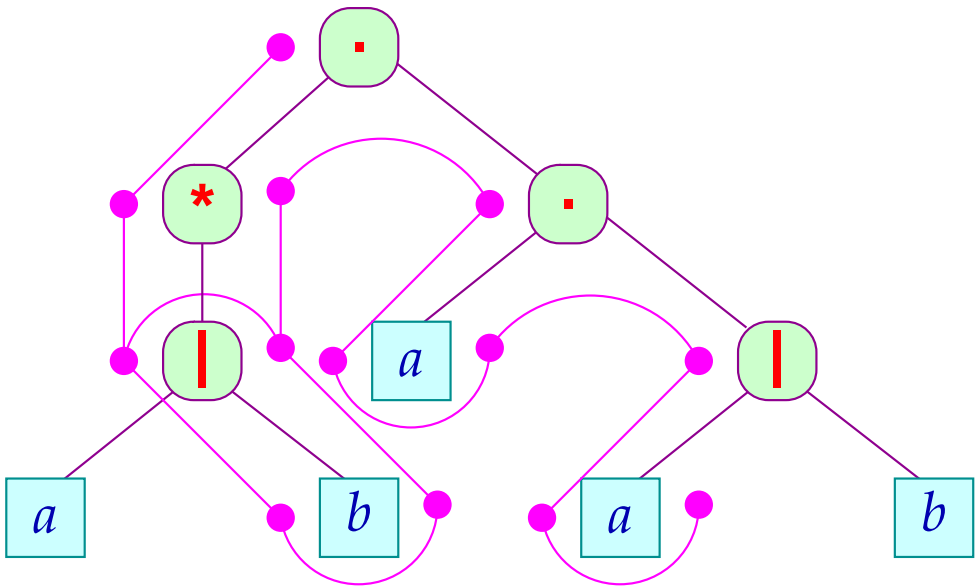
Beispiel:

$w = bbaa$:



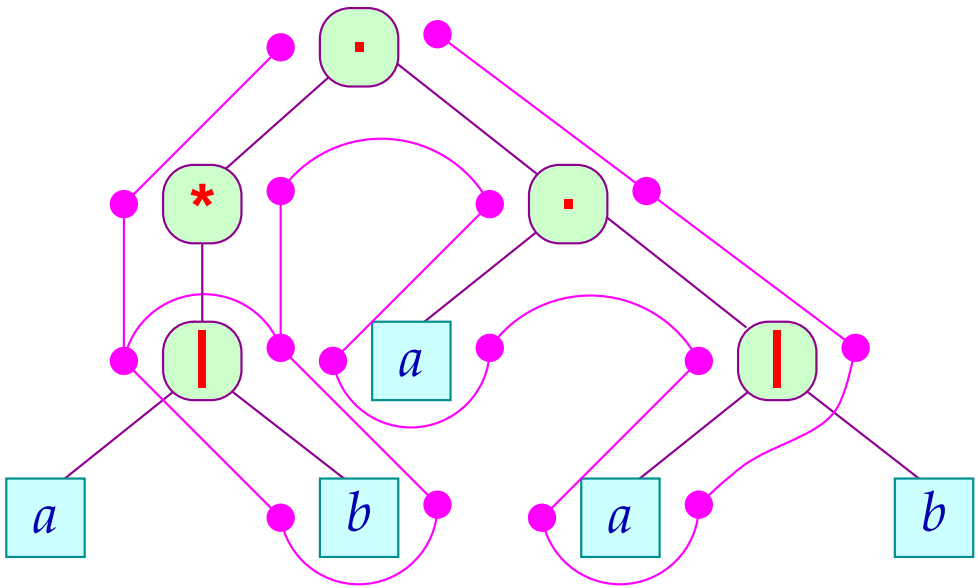
Beispiel:

$w = bbaa$:



Beispiel:

$w = bbaa$:

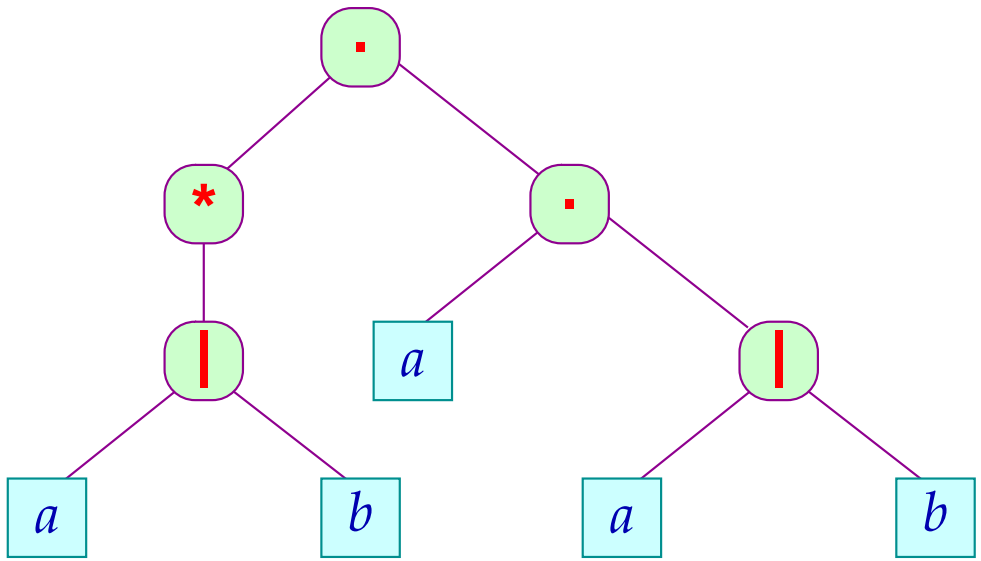


Beachte:

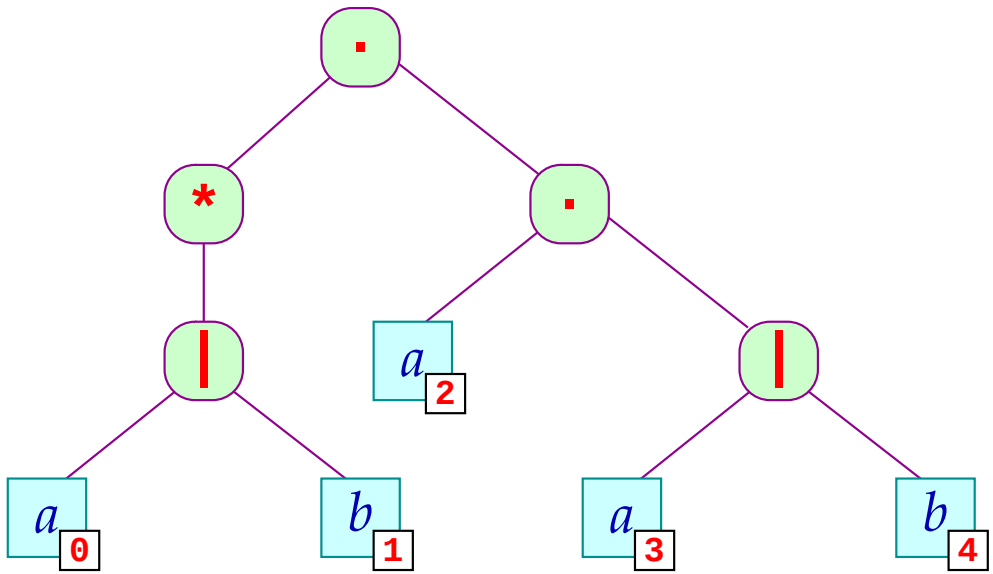
- Gelesen wird nur an den Blättern.
- Die Navigation im Baum erfolgt ohne Lesen, d.h. mit ϵ-Übergängen.
- Für eine formale Konstruktion müssen wir die Knoten im Baum bezeichnen.
- Dazu benutzen wir (hier) einfach den dargestellten Teilausdruck :-)
- Leider gibt es eventuell mehrere gleiche Teilausdrücke :-(

$\Longrightarrow$ Wir numerieren die Blätter durch ...

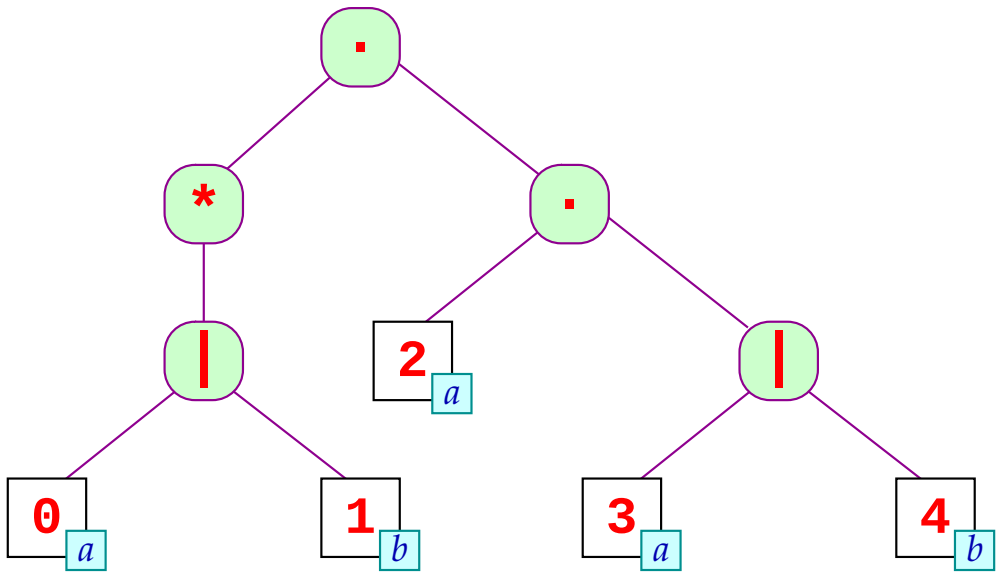
... im Beispiel:



... im Beispiel:



... im Beispiel:



Die Konstruktion:

Zustände: $\bullet r$, $r\bullet$ r Knoten von e;

Anfangszustand: $\bullet e$;

Endzustand: $e\bullet$;

Übergangsrelation:

Für Blätter $r \equiv \boxed{i \mid x}$ benötigen wir: $(\bullet r, x, r\bullet)$.

Die übrigen Übergänge sind:

r	Übergänge
$r_1 \mid r_2$	$(\bullet r, \epsilon, \bullet r_1)$ $(\bullet r, \epsilon, \bullet r_2)$ $(r_1 \bullet, \epsilon, r \bullet)$ $(r_2 \bullet, \epsilon, r \bullet)$
$r_1 \cdot r_2$	$(\bullet r, \epsilon, \bullet r_1)$ $(r_1 \bullet, \epsilon, \bullet r_2)$ $(r_2 \bullet, \epsilon, r \bullet)$

r	Übergänge
r_1^*	$(\bullet r, \epsilon, r \bullet)$ $(\bullet r, \epsilon, \bullet r_1)$ $(r_1 \bullet, \epsilon, \bullet r_1)$ $(r_1 \bullet, \epsilon, r \bullet)$
$r_1?$	$(\bullet r, \epsilon, r \bullet)$ $(\bullet r, \epsilon, \bullet r_1)$ $(r_1 \bullet, \epsilon, r \bullet)$

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 :-(

$\Longrightarrow$

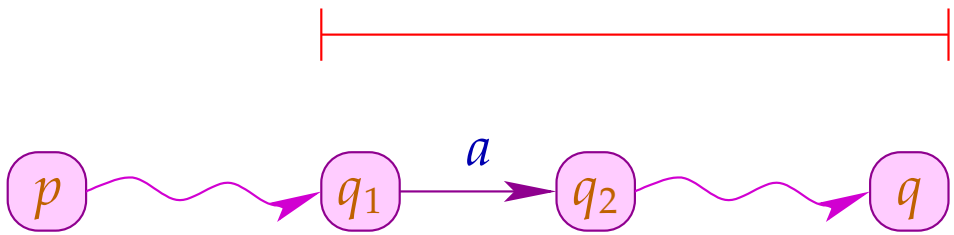
Strategie:

(1) Beseitigung der ϵ-Übergänge;

(2) Beseitigung des Nichtdeterminismus :-)

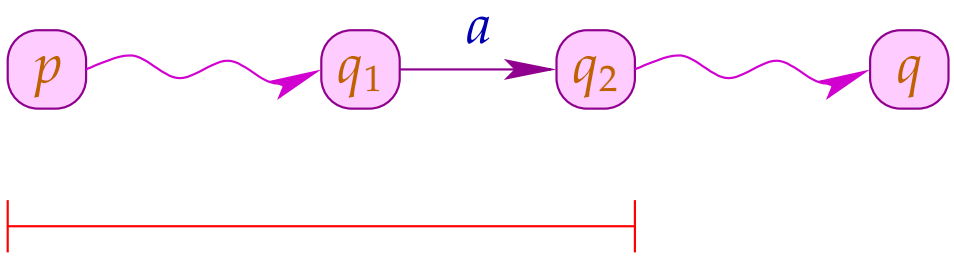
Beseitigung von ϵ-Übergängen:

Zwei einfache Ansätze:



Beseitigung von ϵ-Übergängen:

Zwei einfache Ansätze:



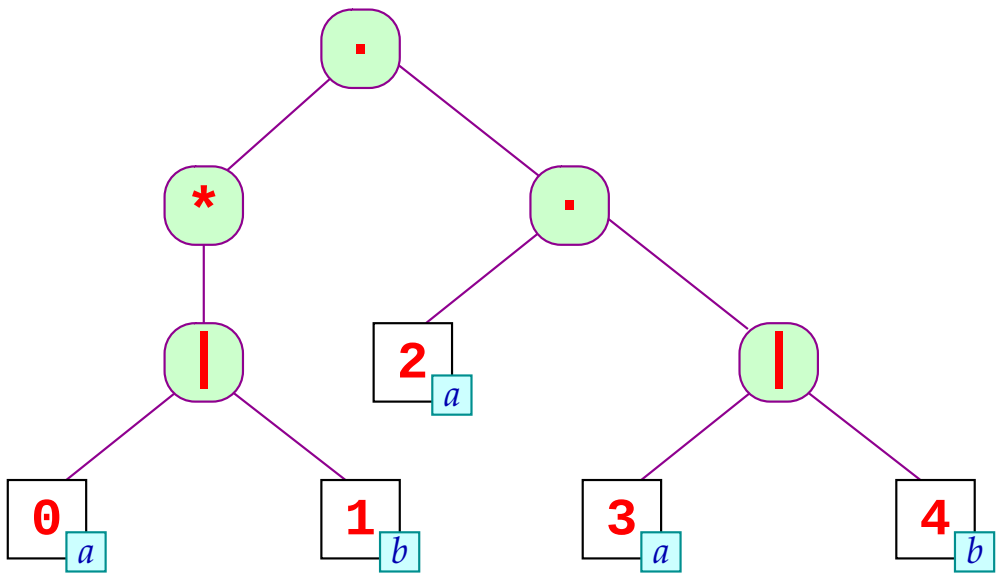
Wir benutzen hier den zweiten Ansatz.

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

1. Schritt:

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

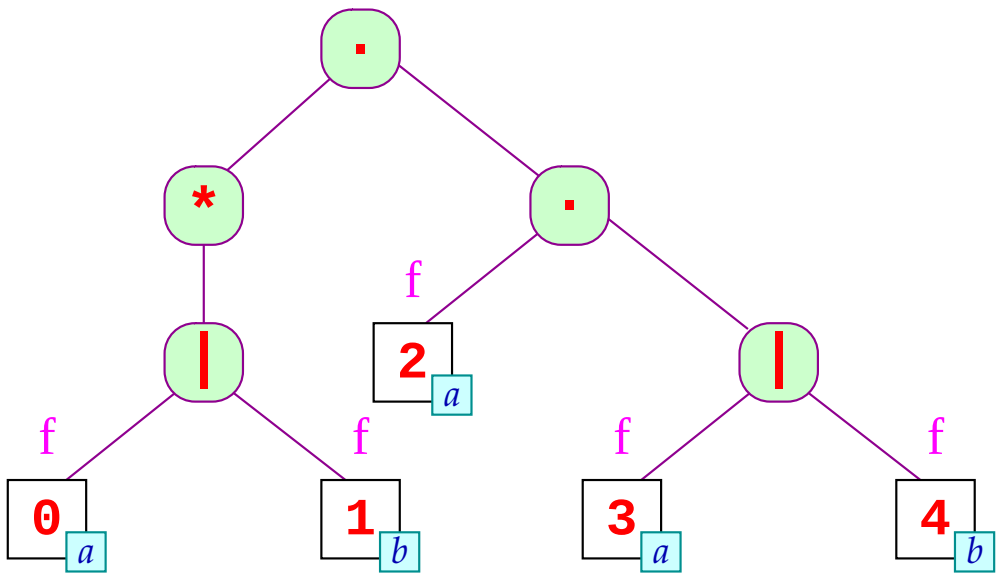
... im Beispiel:



1. Schritt:

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

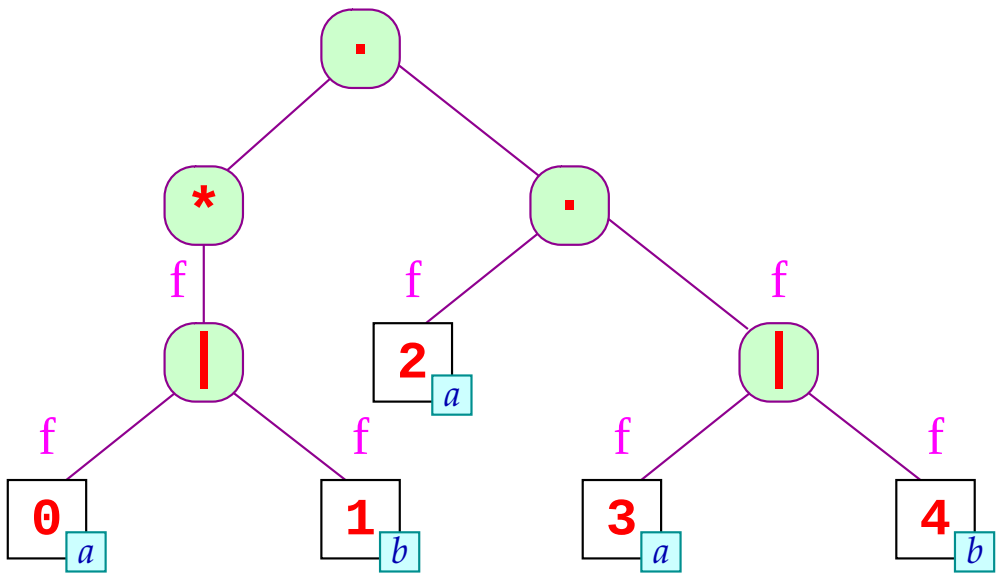
... im Beispiel:



1. Schritt:

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

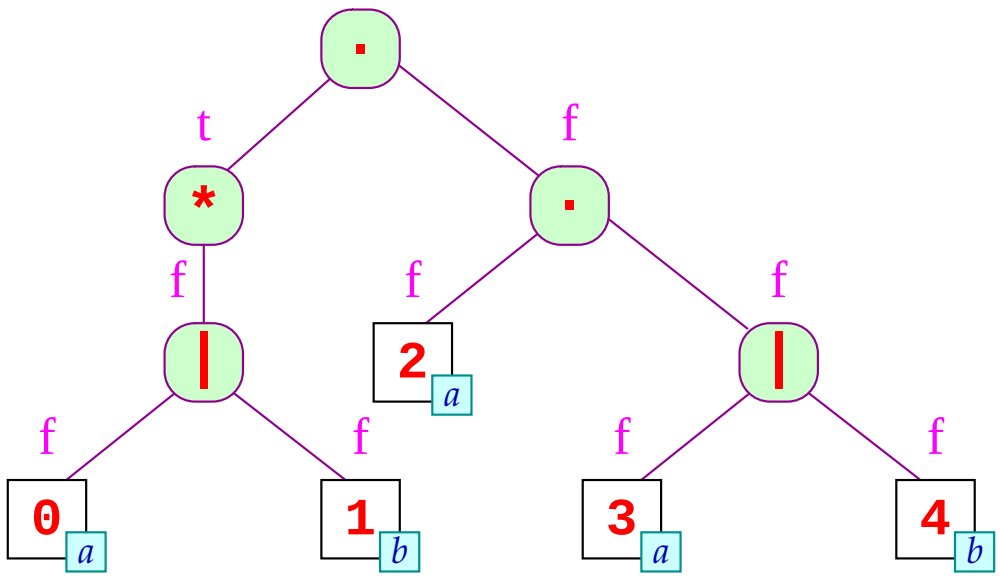
... im Beispiel:



1. Schritt:

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

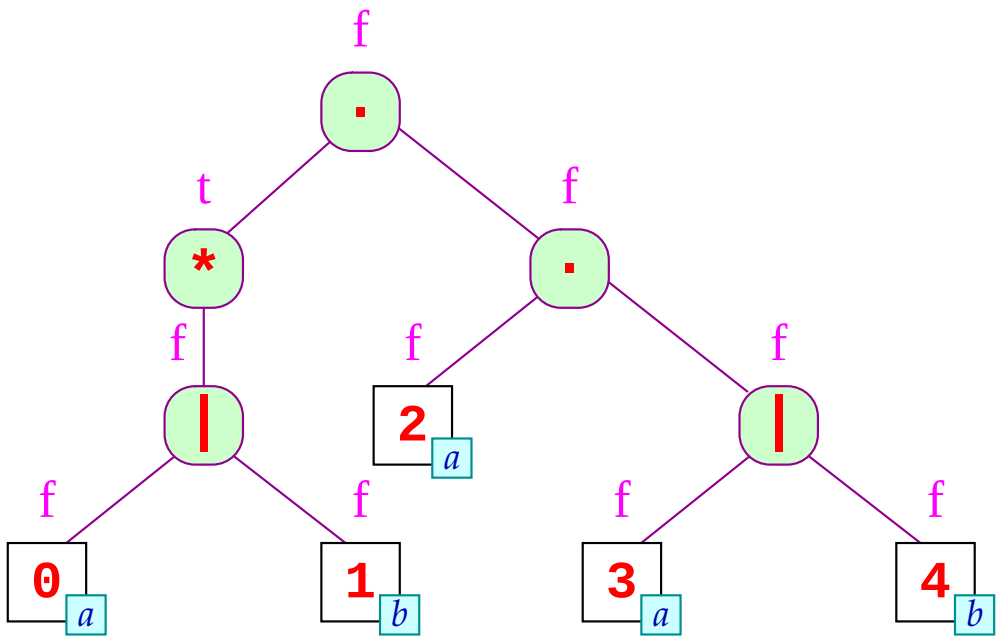
... im Beispiel:



1. Schritt:

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

... im Beispiel:



Implementierung: DFS post-order Traversierung

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

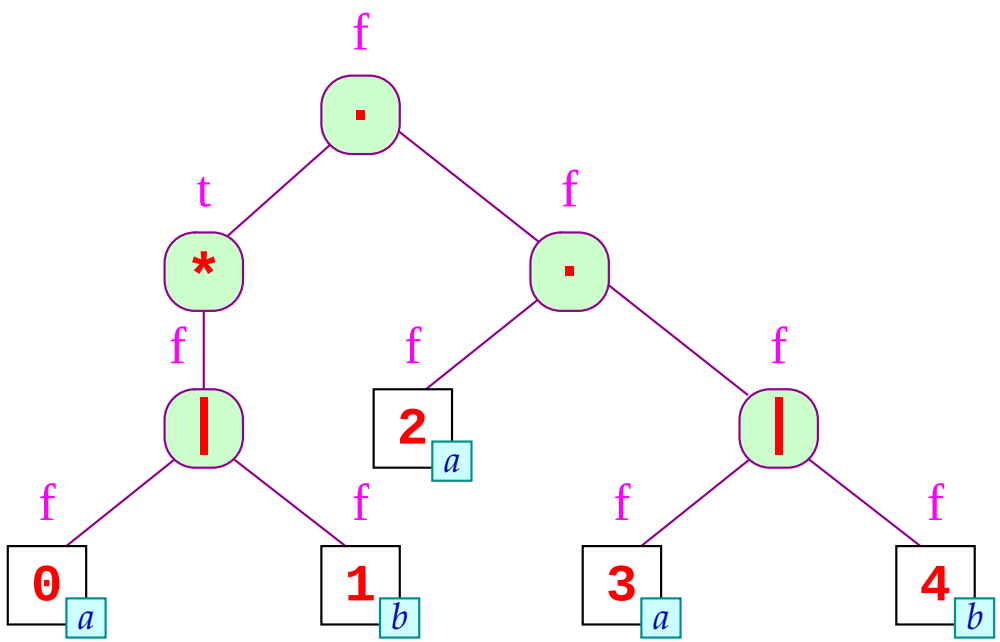
Andernfalls:

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

2. Schritt:

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

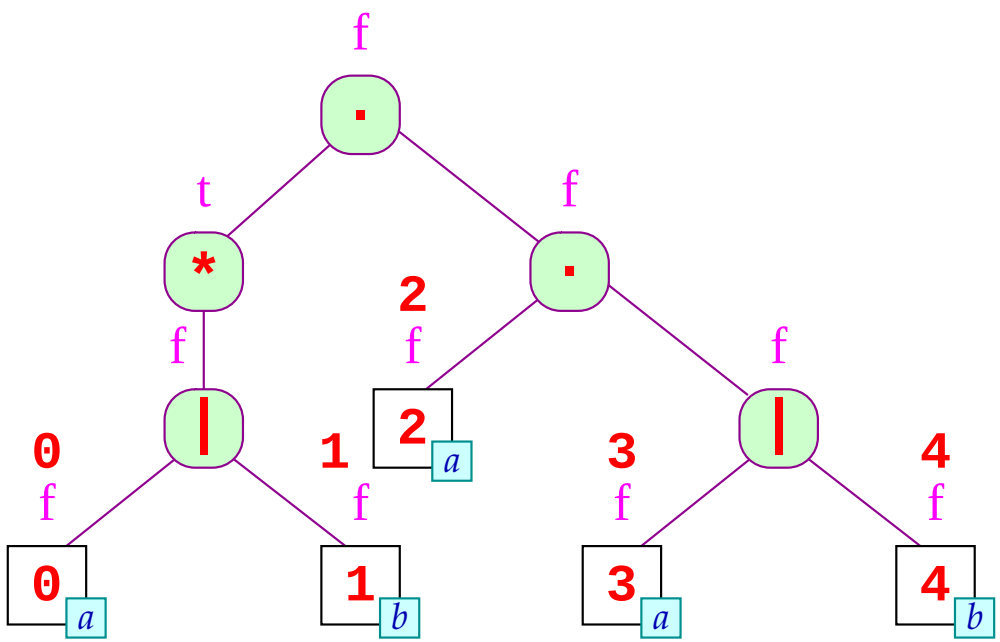
... im Beispiel:



2. Schritt:

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

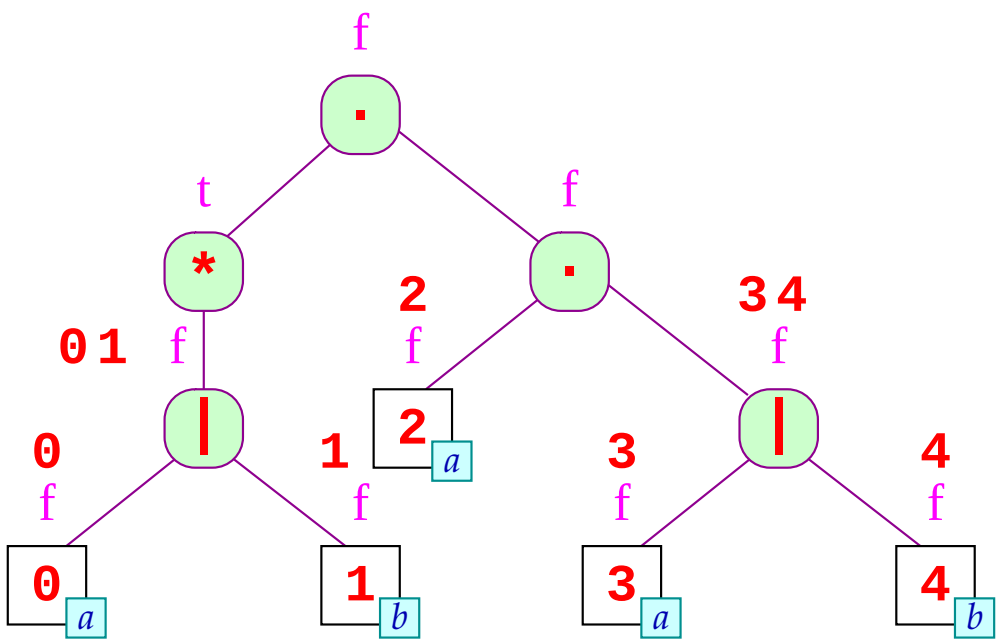
... im Beispiel:



2. Schritt:

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

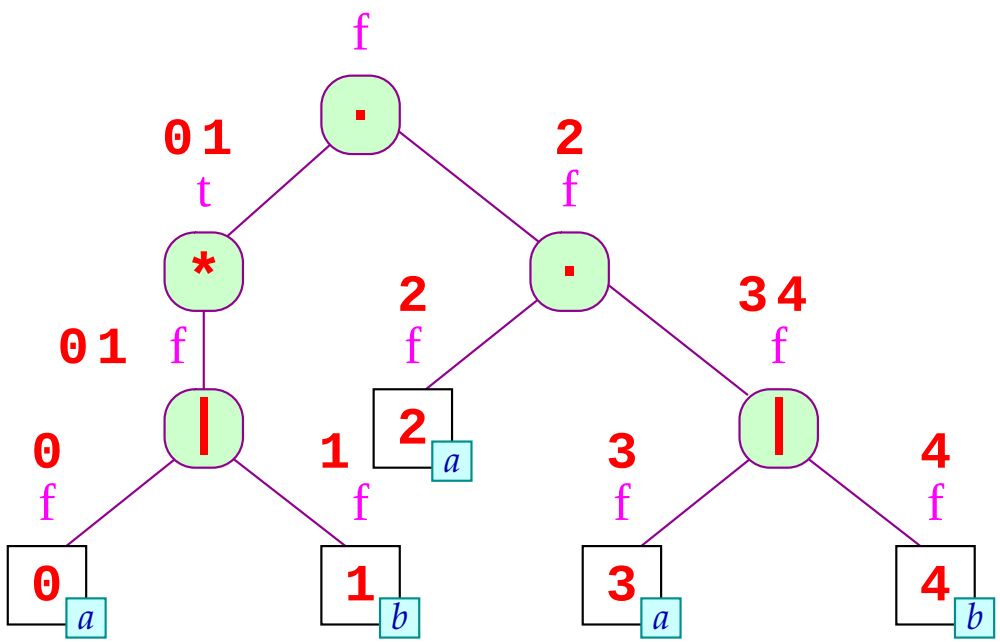
... im Beispiel:



2. Schritt:

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

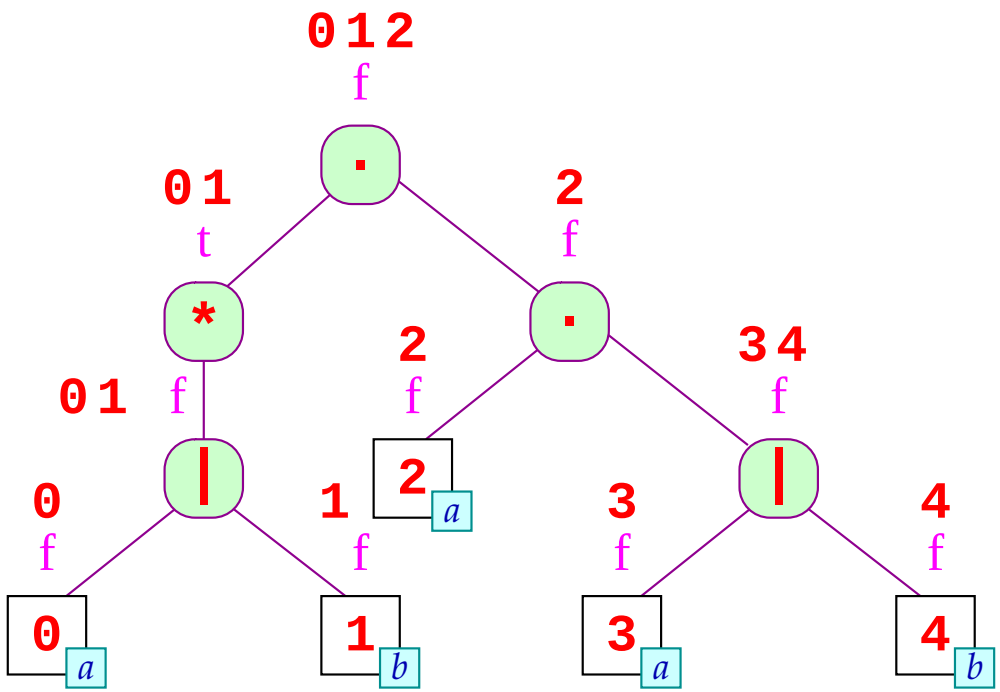
... im Beispiel:



2. Schritt:

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

... im Beispiel:



Implementierung: DFS post-order Traversierung

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

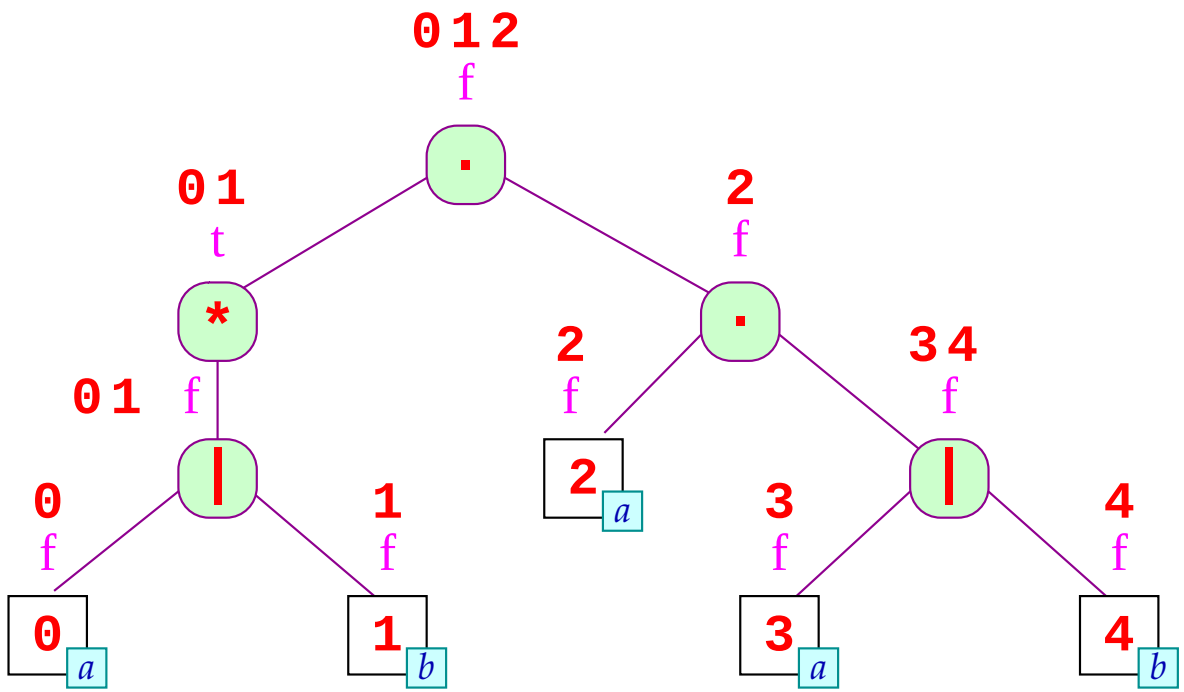
Andernfalls:

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

3. Schritt:

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

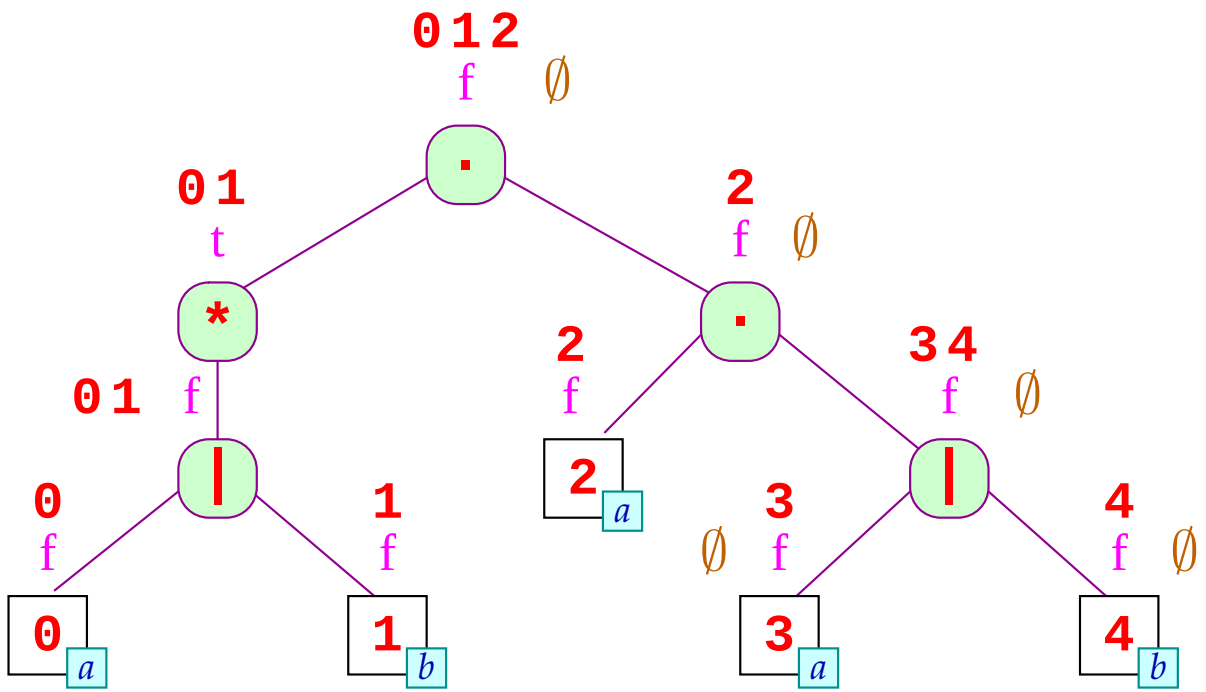
... im Beispiel:



3. Schritt:

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

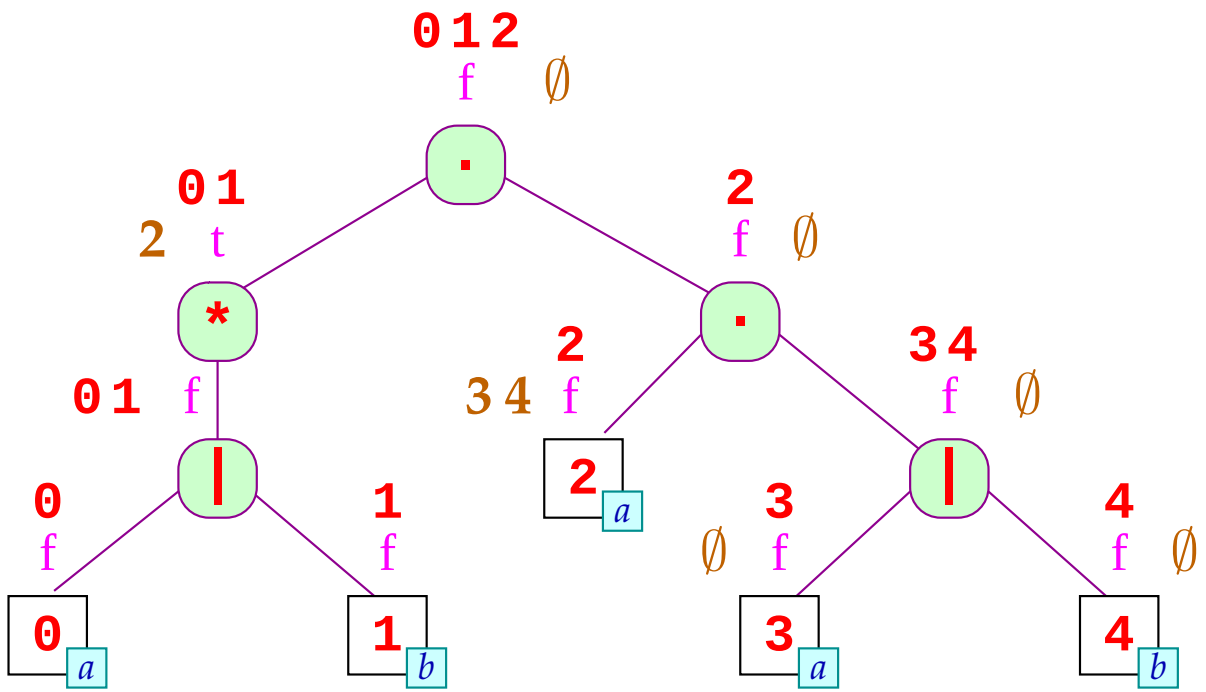
... im Beispiel:



3. Schritt:

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

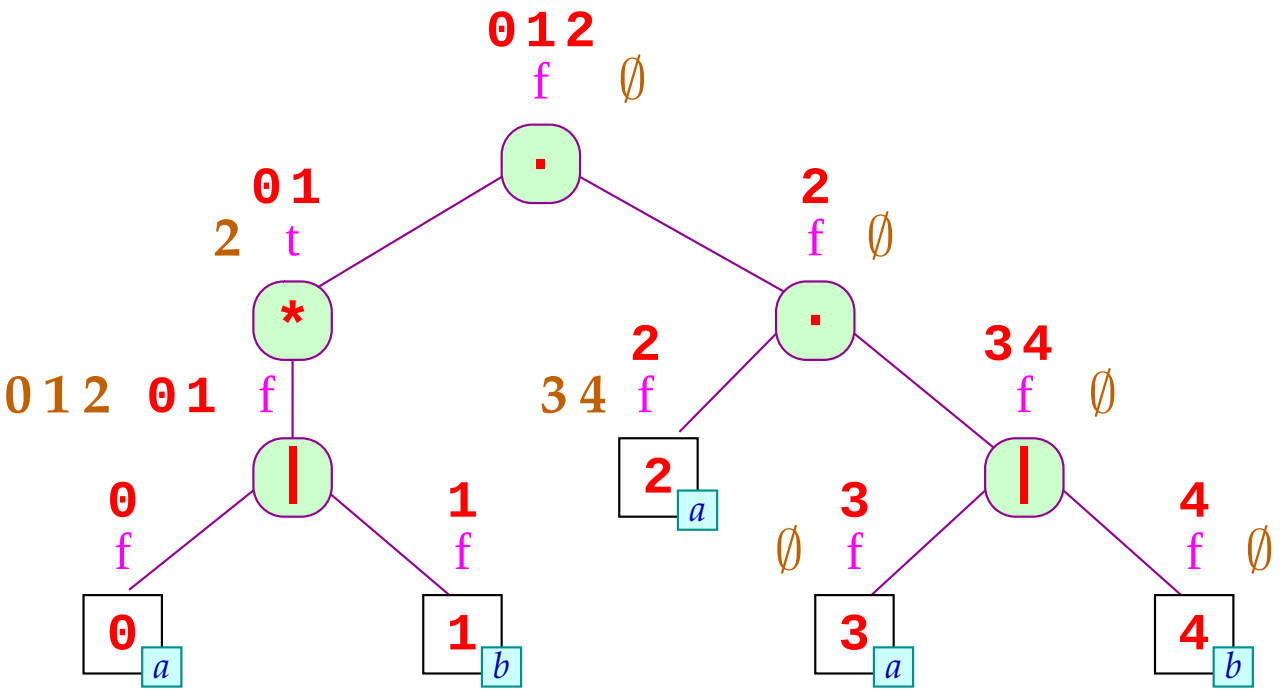
... im Beispiel:



3. Schritt:

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

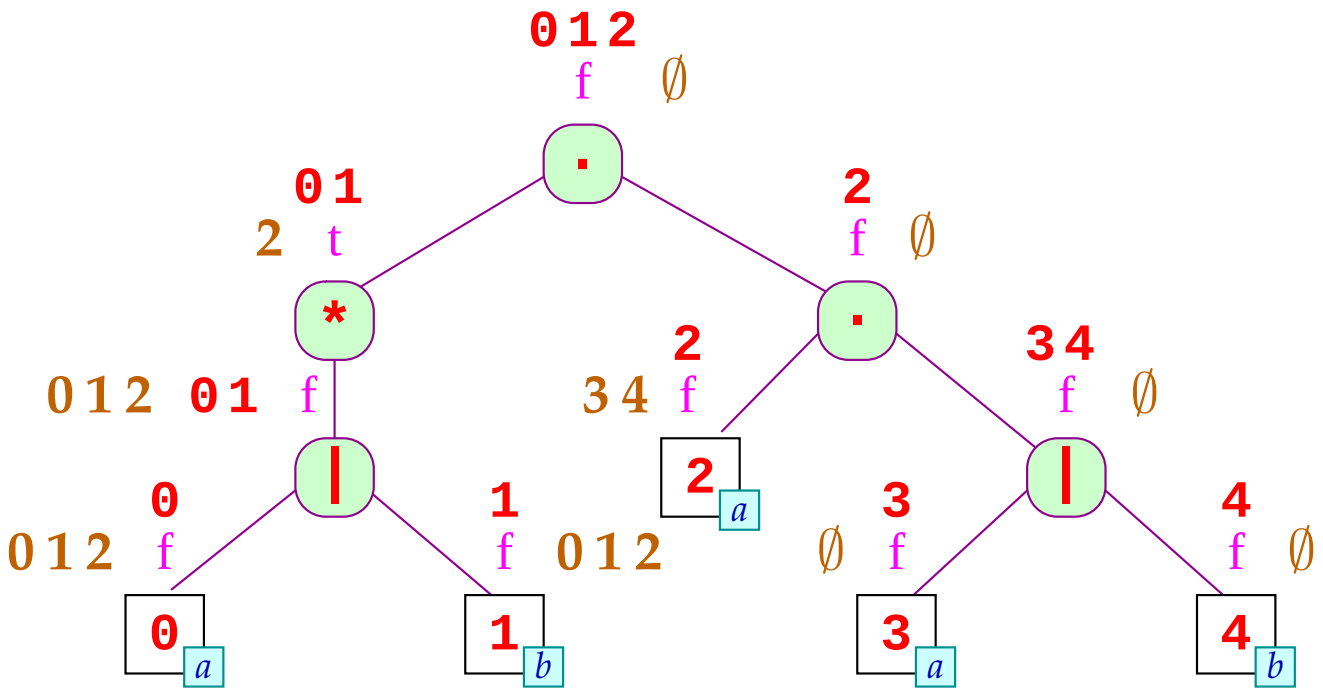
... im Beispiel:



3. Schritt:

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

... im Beispiel:



Implementierung: DFS pre-order Traversierung ;-)

Für die Wurzel haben wir:

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

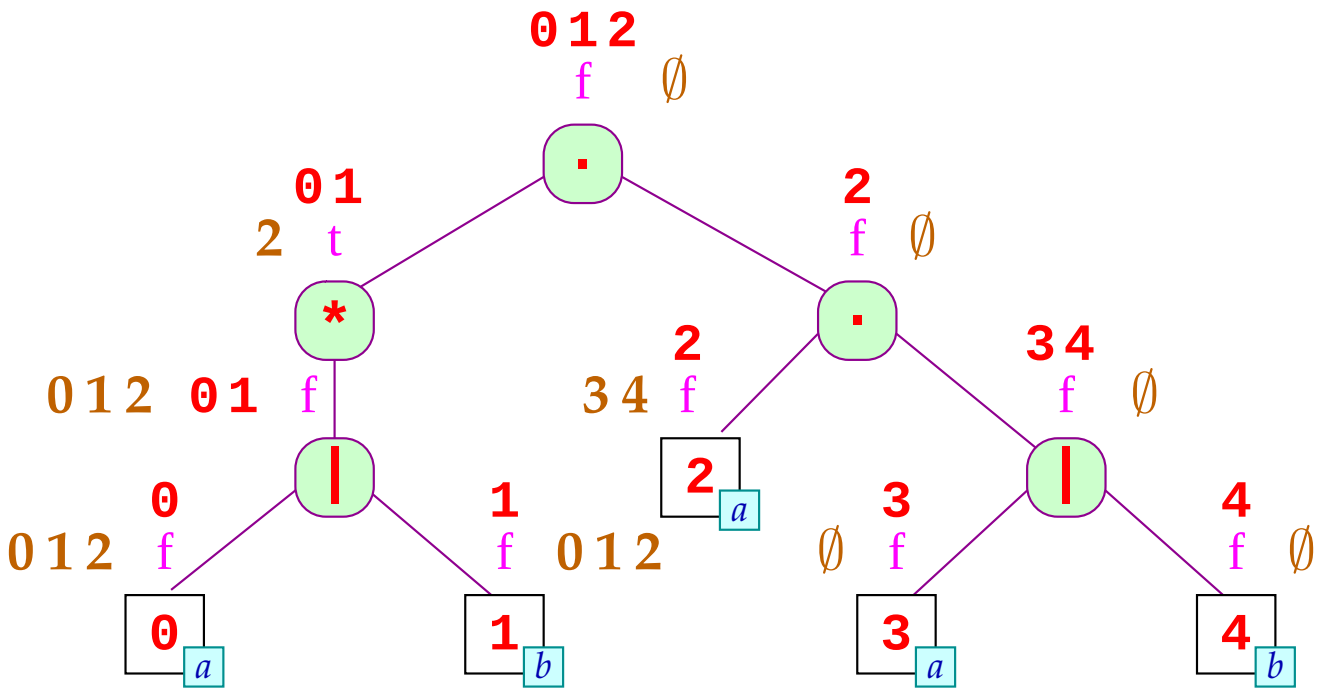
Ansonsten machen wir eine Fallunterscheidung über den Kontext:

r	Regeln
$r_1 \mid r_2$	$\mathsf{next}[r_1] = \mathsf{next}[r]$ $\mathsf{next}[r_2] = \mathsf{next}[r]$
$r_1 \cdot r_2$	$\mathsf{next}[r_1] = \begin{cases} \mathsf{first}[r_2] \cup \mathsf{next}[r] & \text{falls } \mathsf{empty}[r_2] = t \\ \mathsf{first}[r_2] & \text{falls } \mathsf{empty}[r_2] = f \end{cases}$ $\mathsf{next}[r_2] = \mathsf{next}[r]$
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 Blätter: $\mathsf{last}[r] = \{i \text{ in } r \mid (\boxed{i}\boxed{x}\bullet, \epsilon, r\bullet) \in \delta^*\}$

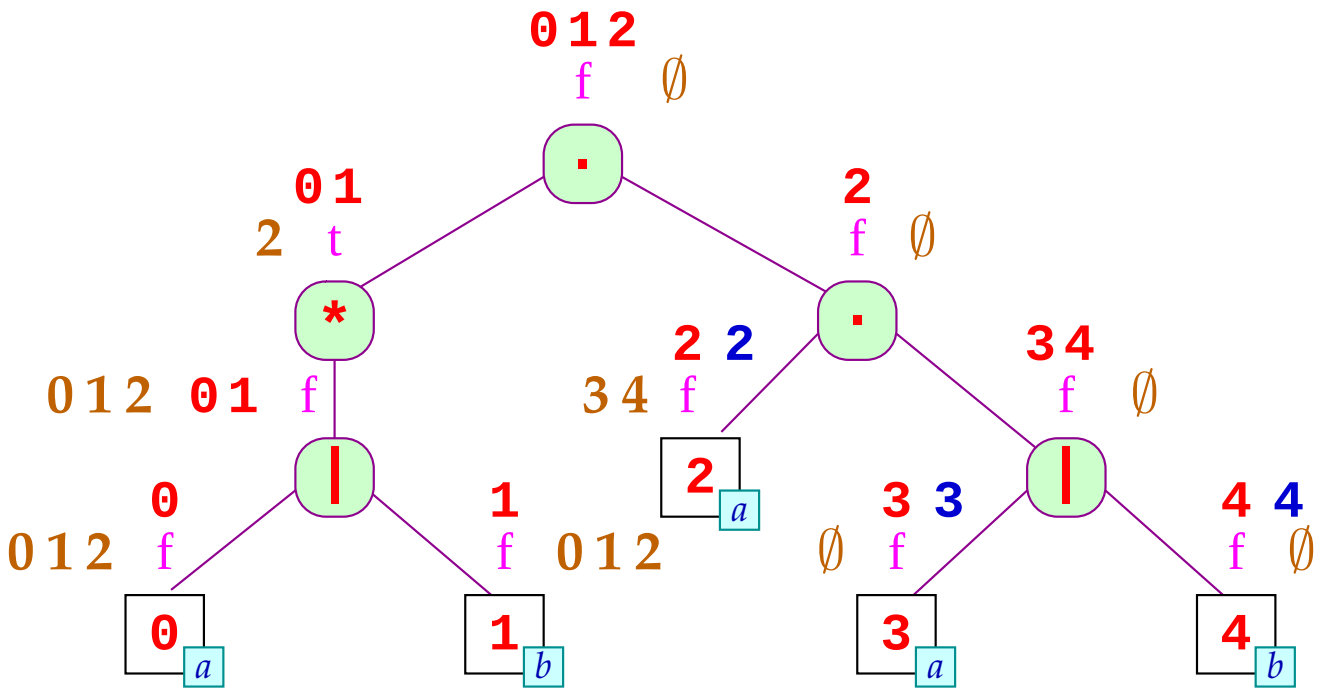
... im Beispiel:



4. Schritt:

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

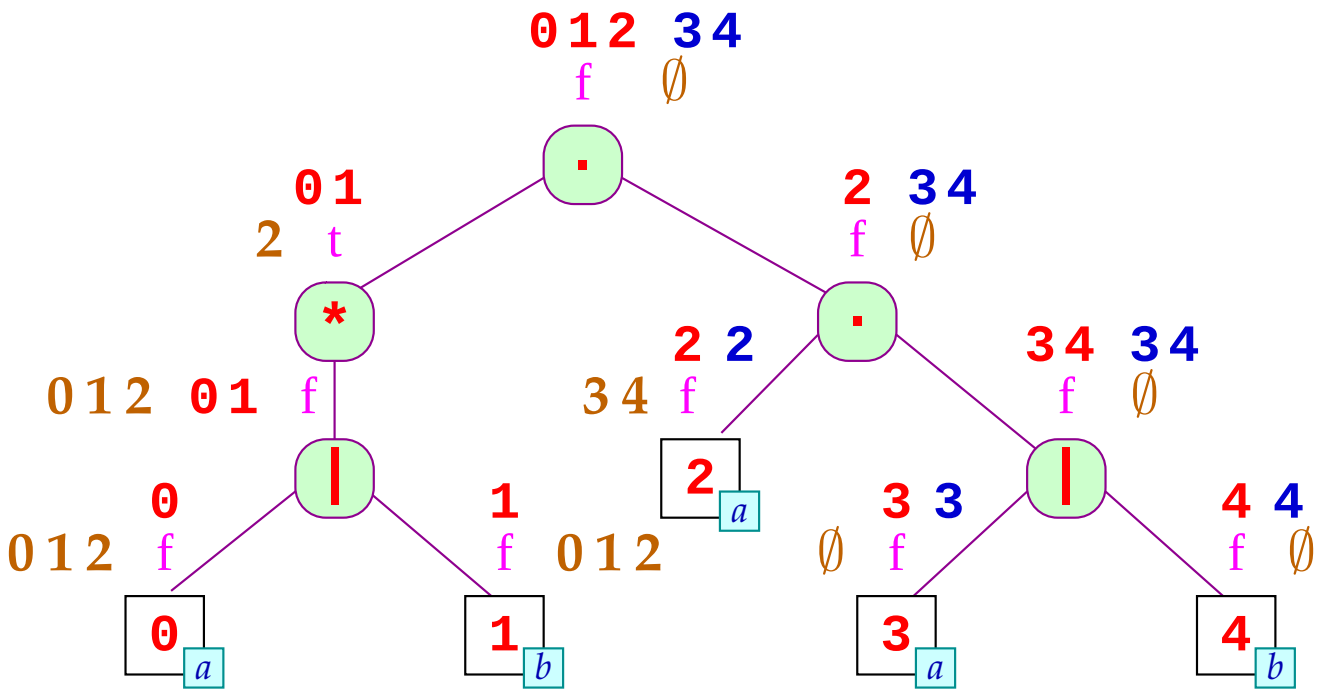
... im Beispiel:



4. Schritt:

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

... 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:

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

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: $\{\bullet e\} \cup \mathsf{last}[e]$.

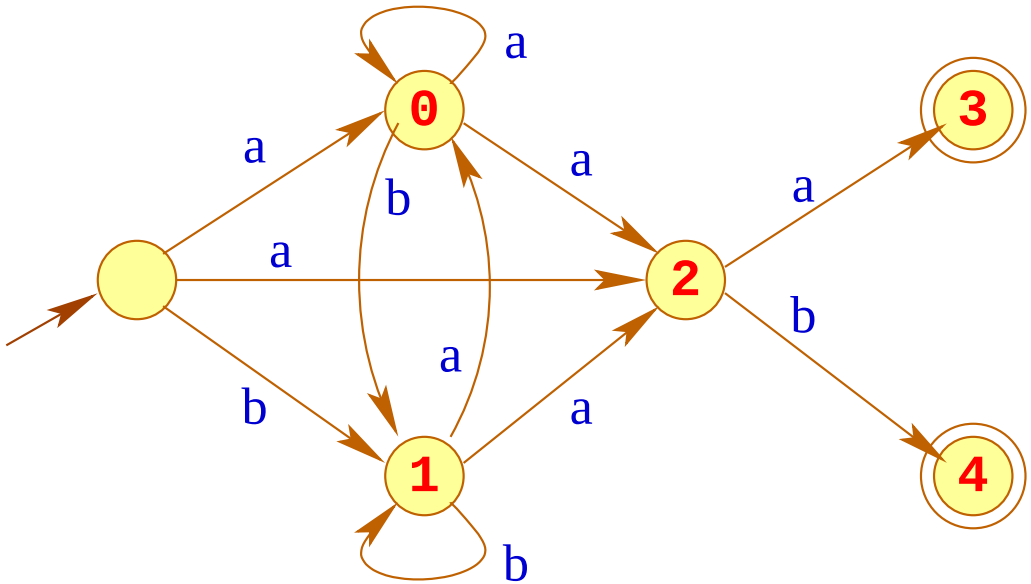
Übergänge:

$(\bullet e, a, i\bullet)$ 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.

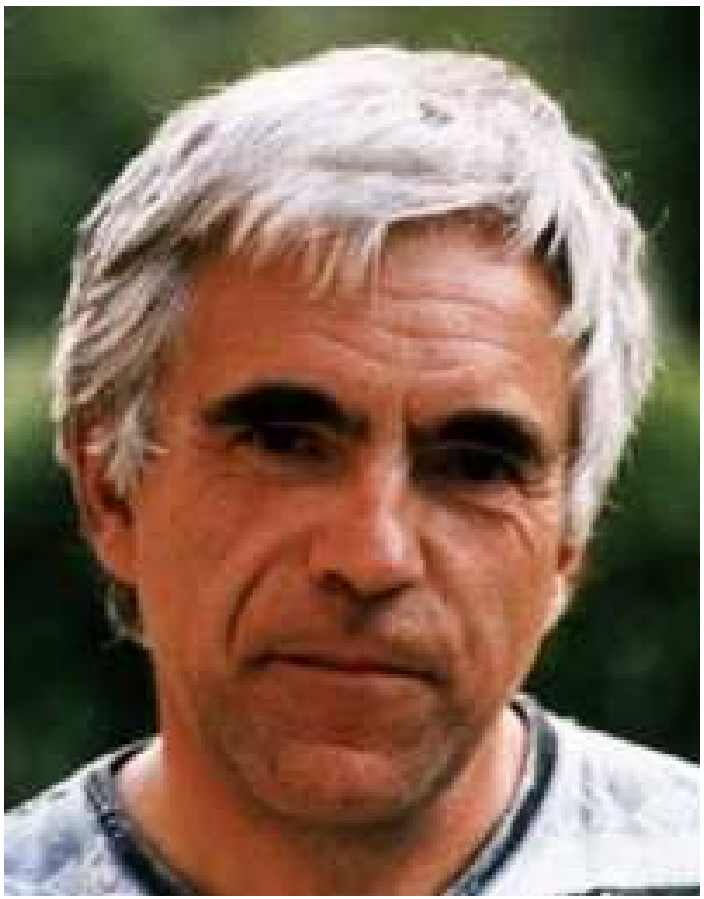
Den resultierenden Automaten bezeichnen wir mit A_e.

... im Beispiel:

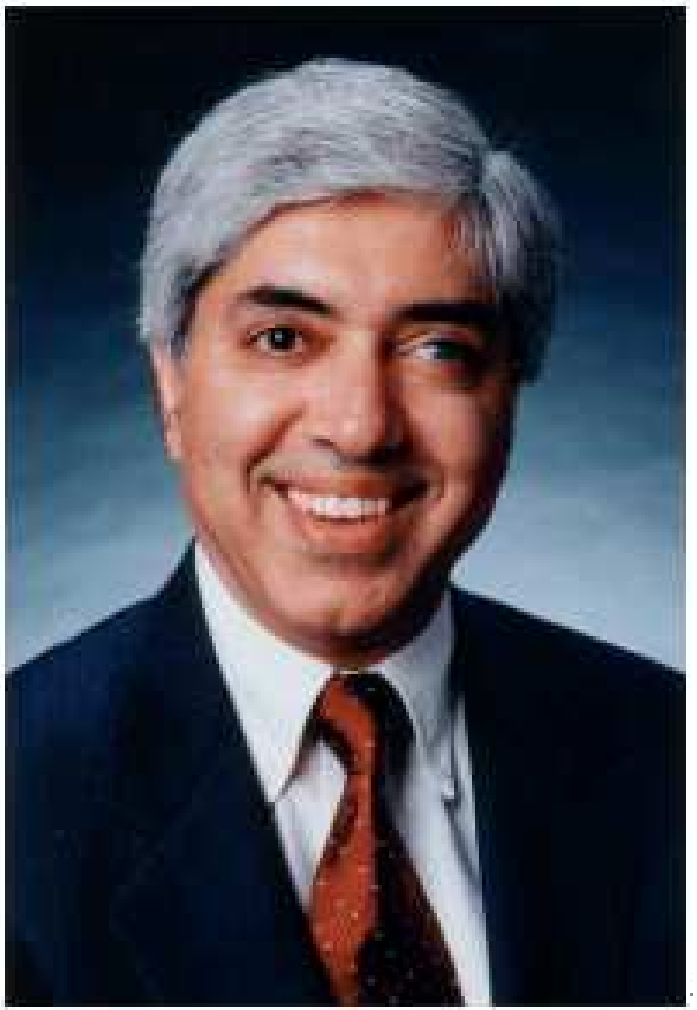


Bemerkung:

- 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 ...

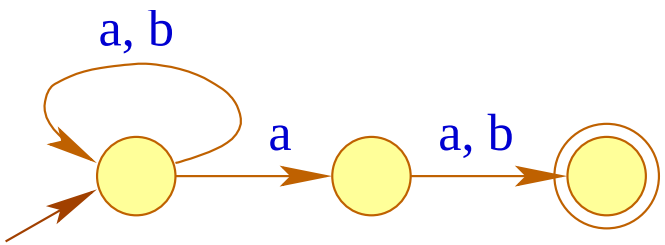


Gerard Berry, Esterel Technologies



Ravi Sethi, Research VR, Lucent Technologies

Der erwartete Automat:

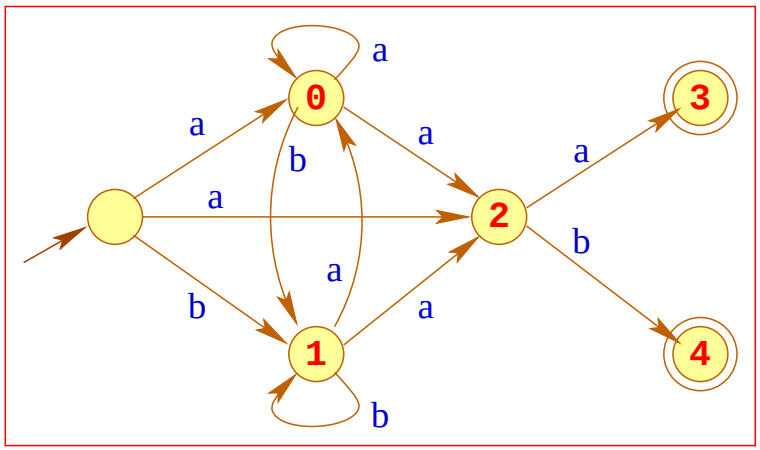


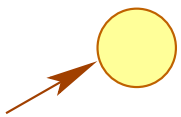
Bemerkung:

- 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

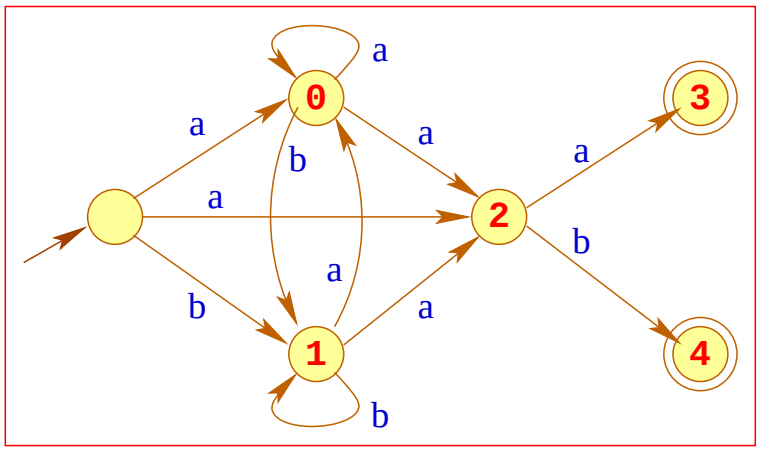
$\Longrightarrow$ Teilmengen-Konstruktion

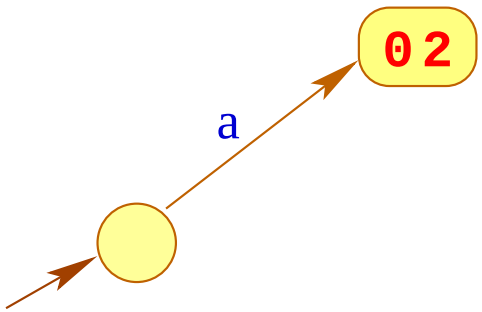
... im Beispiel:



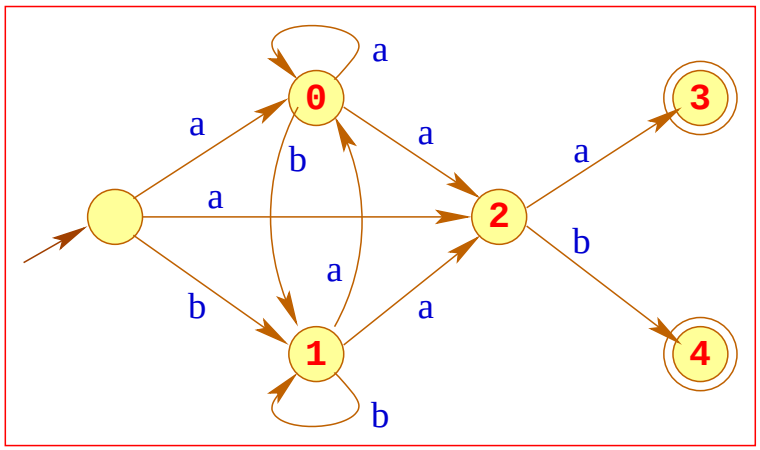


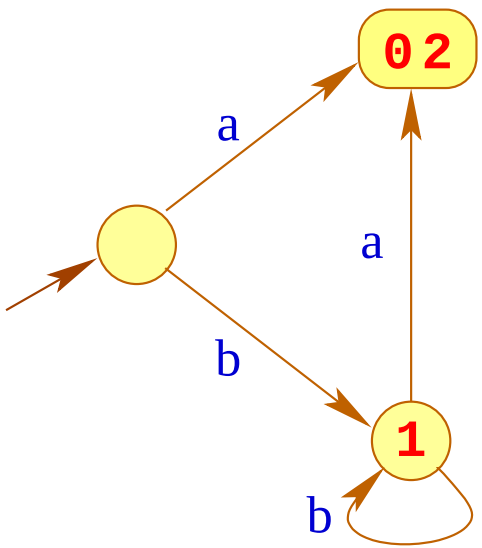
... im Beispiel:



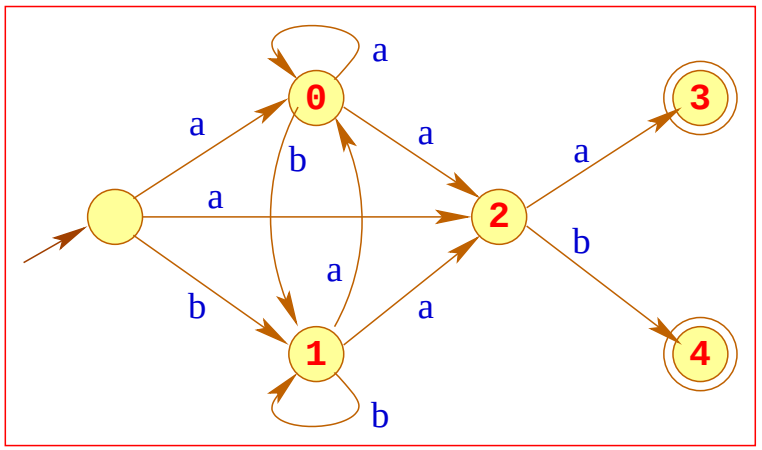


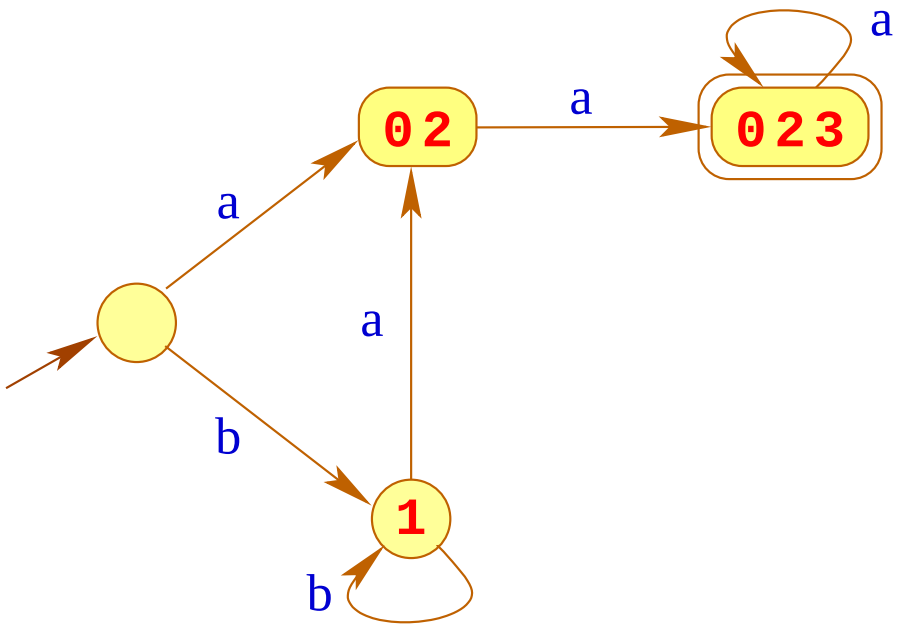
... im Beispiel:



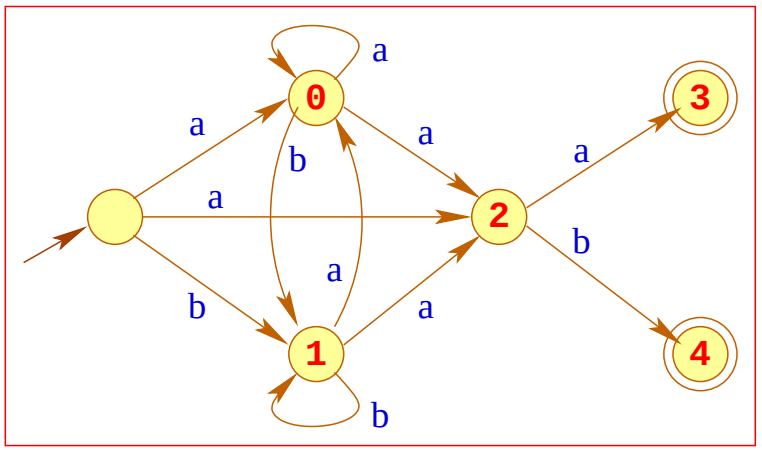


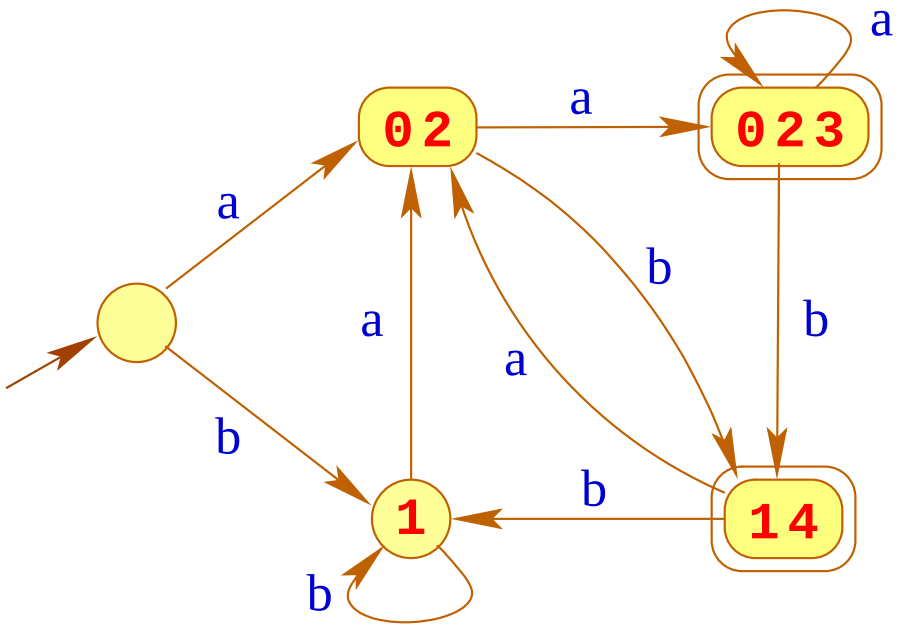
... im Beispiel:





... 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:

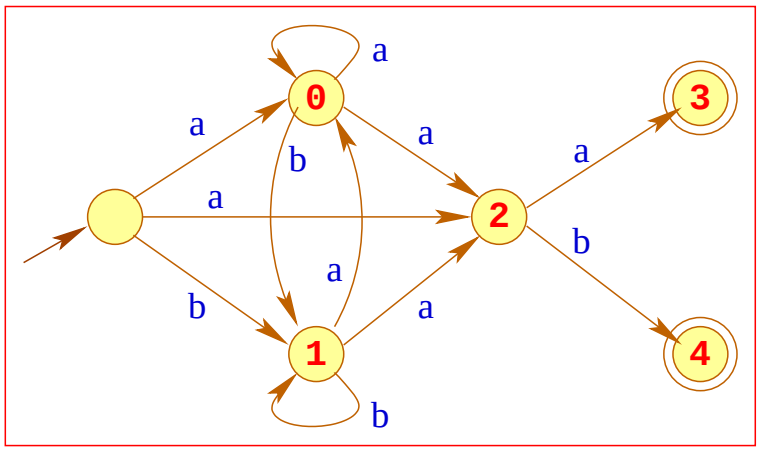
- Leider gibt es exponentiell viele Teilmengen von Q :-(
- Um nur nützliche Teilmengen zu betrachten, starten wir mit der Menge $Q_{\mathcal{P}} = \{I\}$ und fügen weitere Zustände nur nach Bedarf hinzu ...
- d.h., wenn wir sie von einem Zustand in $Q_{\mathcal{P}}$ aus erreichen können :-)
- Trotz dieser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

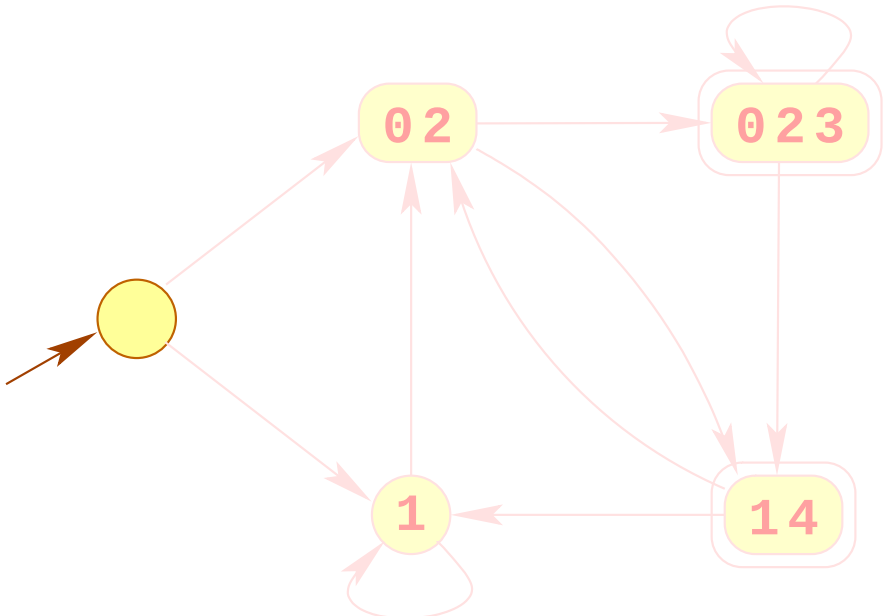
  ... was aber in der Praxis (so gut wie) nie auftritt :-))

- 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:

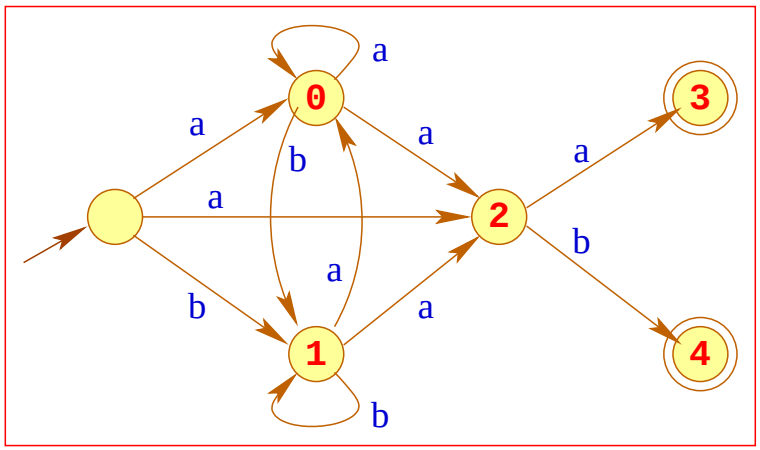
a	b	a	b

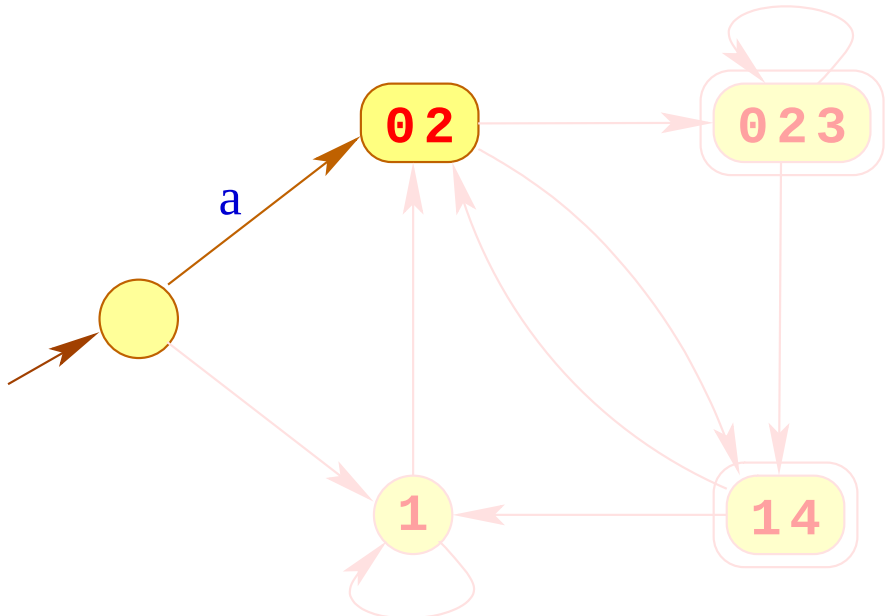




... im Beispiel:

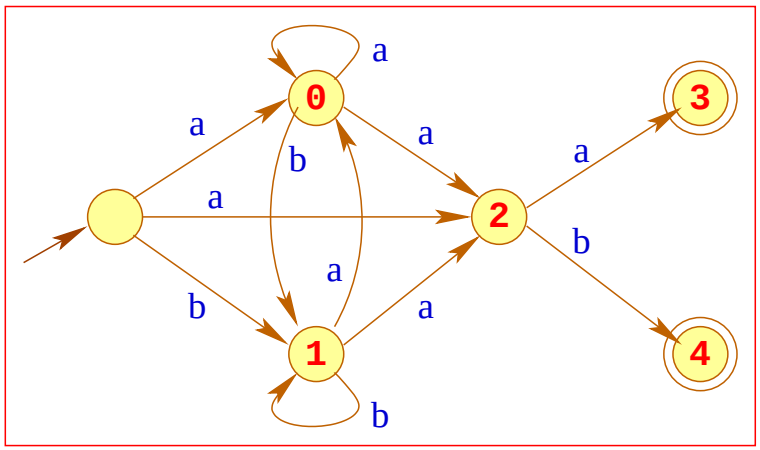
a	b	a	b

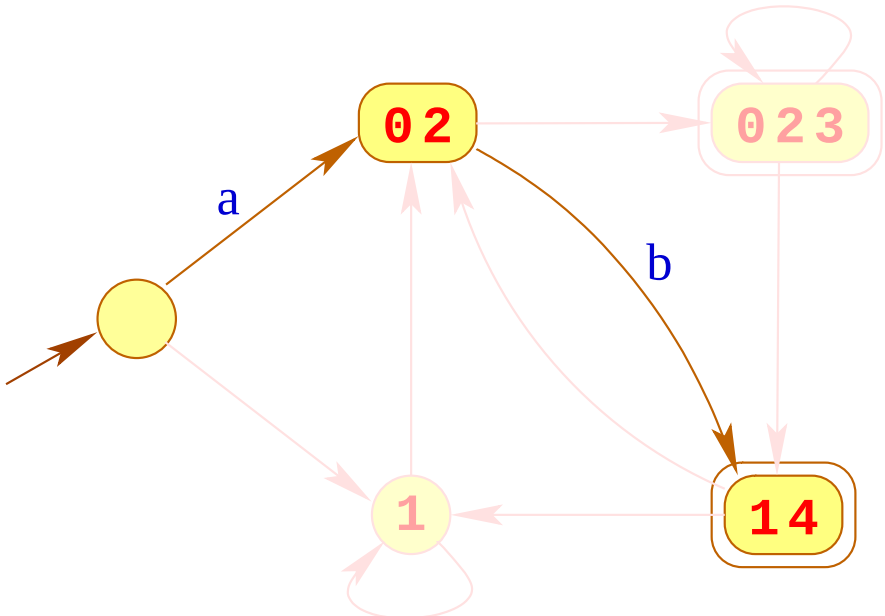




... im Beispiel:

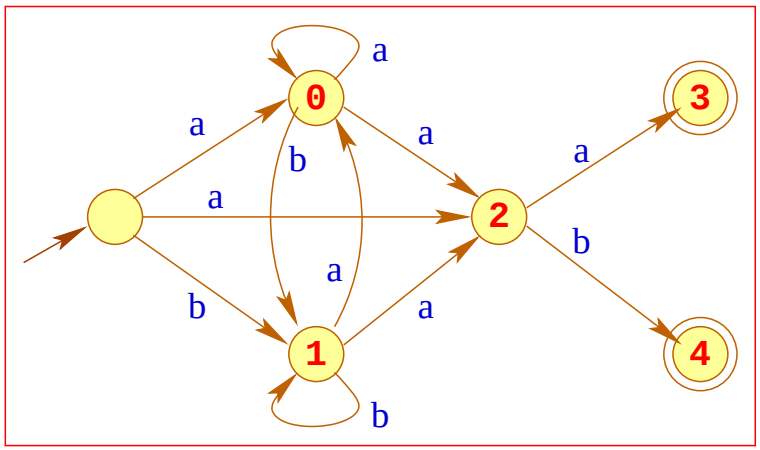
a	b	a	b

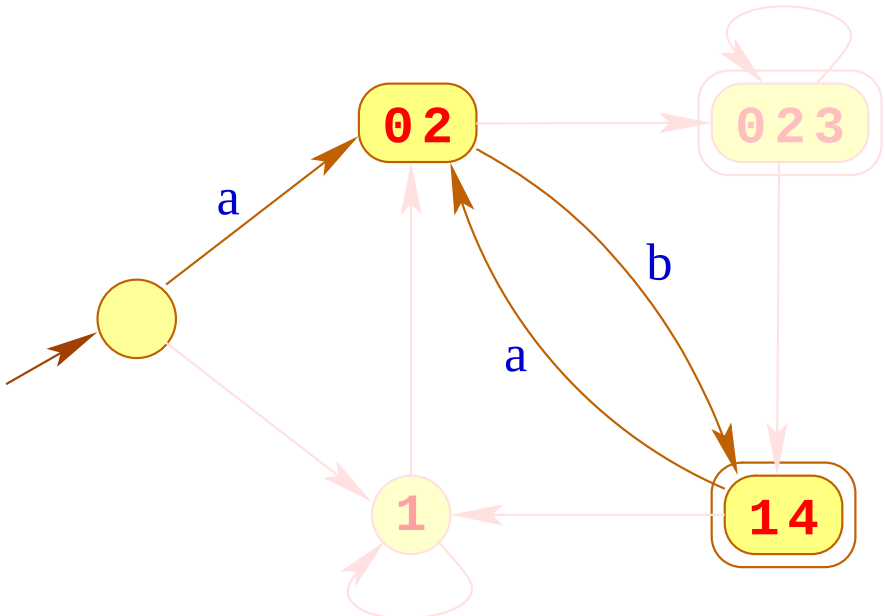




... im Beispiel:

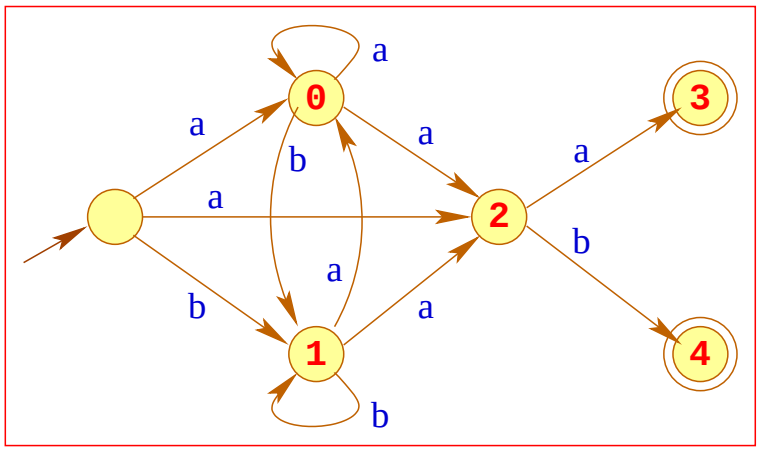
a	b	a	b

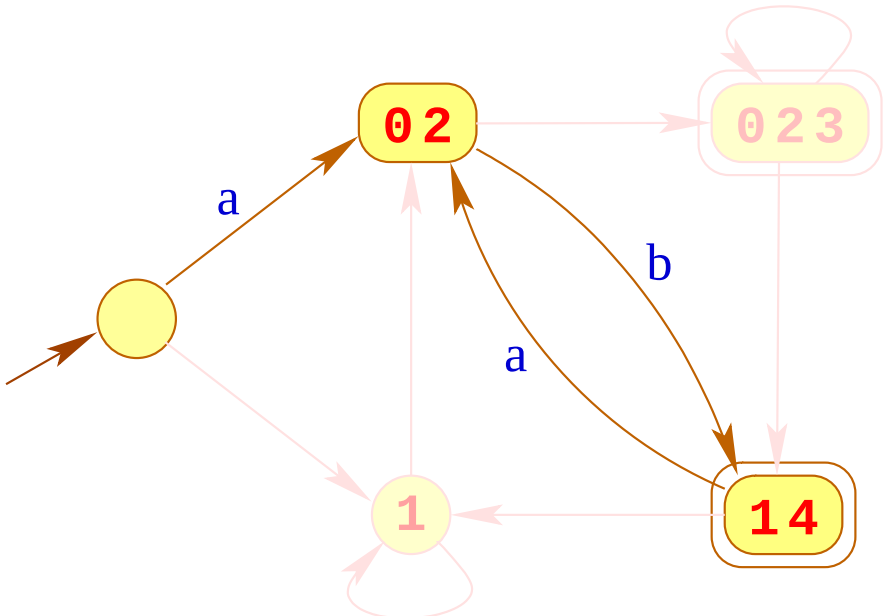




... im Beispiel:

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

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

1.3 Design eines Scanners

Eingabe (vereinfacht): eine Menge von Regeln:

$$
\begin{array}{ll}
e_1 & \{\ \texttt{action}_1\ \} \\
e_2 & \{\ \texttt{action}_2\ \} \\
\multicolumn{2}{c}{\ldots} \\
e_k & \{\ \texttt{action}_k\ \}
\end{array}
$$

1.3 Design eines Scanners

Eingabe (vereinfacht): eine Menge von Regeln:

$$
\begin{array}{ll}
e_1 & \{\ \texttt{action}_1\ \} \\
e_2 & \{\ \texttt{action}_2\ \} \\
\multicolumn{2}{c}{\ldots} \\
e_k & \{\ \texttt{action}_k\ \}
\end{array}
$$

Ausgabe: ein Programm, das

- ... von der Eingabe ein maximales Präfix w liest, das $e_1 \mid \ldots \mid e_k$ erfüllt;
- ... das minimale i ermittelt, so dass $w \in [\![e_i]\!]$;
- ... für w $\texttt{action}_i$ ausführt.

Implementierung:

Idee:

- Konstruiere den DFA $\mathcal{P}(A_e) = (Q, \Sigma, \delta, \{q_0\}, F)$ zu dem Ausdruck $e = (e_1 \mid \ldots \mid e_k)$;
- Definiere die Mengen:

$$
\begin{aligned}
F_1 &= \{q \in F \mid q \cap \mathsf{last}[e_1] \neq \emptyset\} \\
F_2 &= \{q \in (F \backslash F_1) \mid q \cap \mathsf{last}[e_2] \neq \emptyset\} \\
&\ldots \\
F_k &= \{q \in (F \backslash (F_1 \cup \ldots \cup F_{k-1})) \mid q \cap \mathsf{last}[e_k] \neq \emptyset\}
\end{aligned}
$$

- Für Eingabe w gilt: $\delta^*(q_0, w) \in F_i$ genau dann wenn der Scanner für w $\texttt{action}_i$ ausführen soll :-)