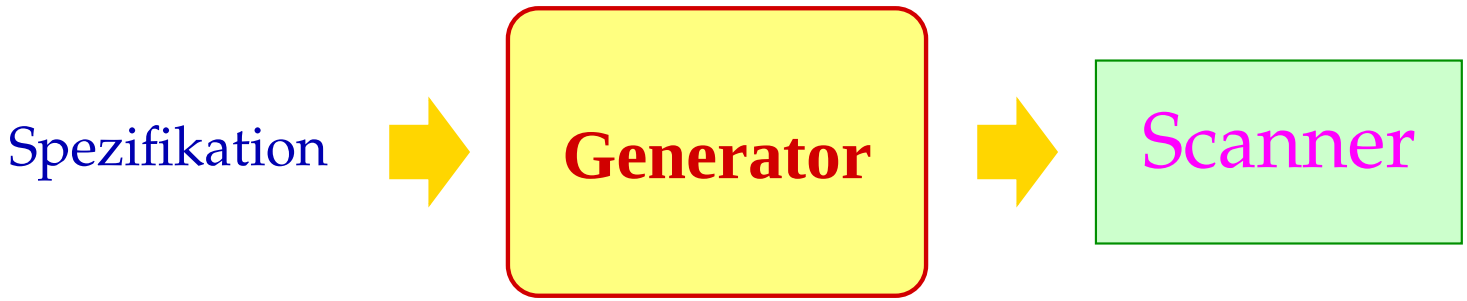
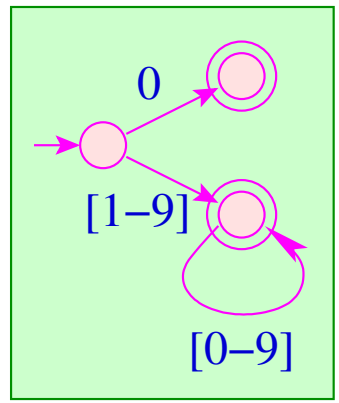
... in unserem Fall:



... in unserem Fall:

0 | [1-9][0-9]*



Spezifikation von Token-Klassen: Reguläre Ausdrücke;

Generierte Implementierung: Endliche Automaten + X :-)

1.1 Grundlagen: Reguläre Ausdrücke

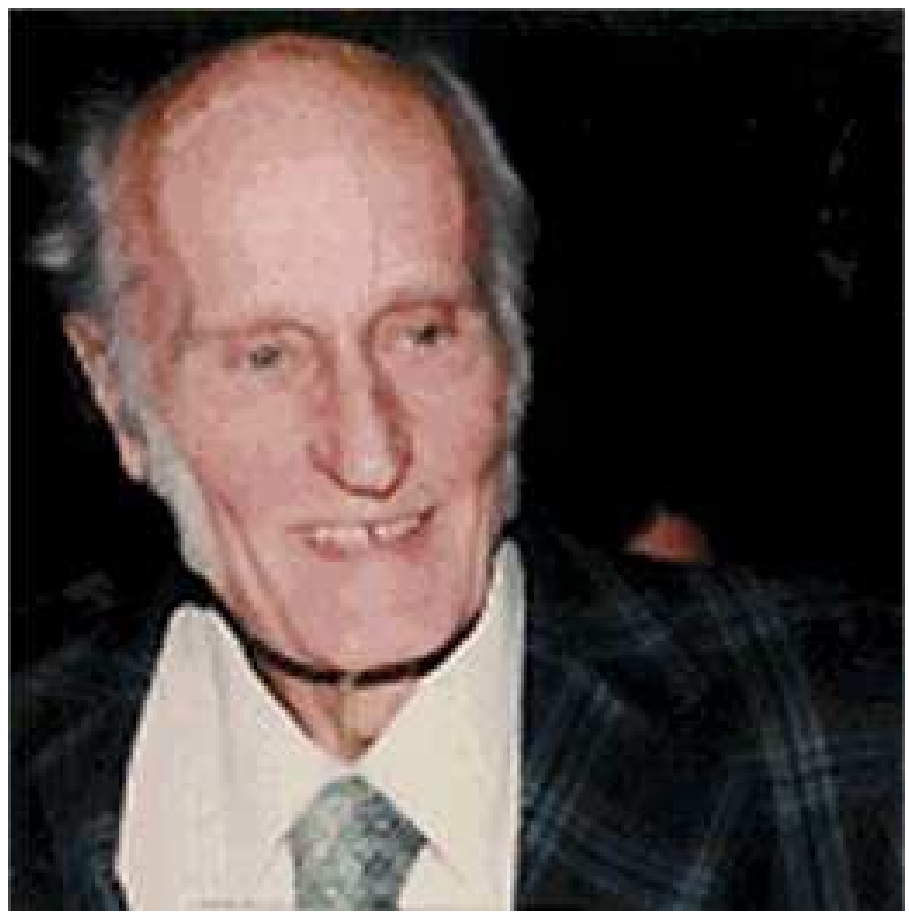
- Programmtext benutzt ein endliches Alphabet Σ von Eingabe-Zeichen, z.B. ASCII :-)
- Die Menge der Textabschnitte einer Token-Klasse ist i.a. regulär.
- Reguläre Sprachen kann man mithilfe regulärer Ausdrücke spezifizieren.

1.1 Grundlagen: Reguläre Ausdrücke

- Programmtext benutzt ein endliches Alphabet Σ von Eingabe-Zeichen, z.B. ASCII :-)
- Die Menge der Textabschnitte einer Token-Klasse ist i.a. regulär.
- Reguläre Sprachen kann man mithilfe regulärer Ausdrücke spezifizieren.

Die Menge $\mathcal{E}_\Sigma$ der (nicht-leeren) regulären Ausdrücke ist die kleinste Menge $\mathcal{E}$ mit:

- $\epsilon \in \mathcal{E}$ (ϵ neues Symbol nicht aus Σ);
- $a \in \mathcal{E}$ für alle $a \in \Sigma$;
- $(e_1 \mid e_2), (e_1 \cdot e_2), {e_1}^* \in \mathcal{E}$ sofern $e_1, e_2 \in \mathcal{E}$.



Stephen Kleene, Madison Wisconsin, 1909-1994

Beispiele:

$$((a \cdot b^*) \cdot a)$$

$$(a \mid b)$$

$$((a \cdot b) \cdot (a \cdot b))$$

Beispiele:

$$((a \cdot b^*) \cdot a)$$

$$(a \mid b)$$

$$((a \cdot b) \cdot (a \cdot b))$$

Achtung:

- Wir unterscheiden zwischen Zeichen $a, 0, |, \ldots$ und Meta-Zeichen $(, |,), \ldots$
- Um (hässliche) Klammern zu sparen, benutzen wir Operator-Präzedenzen:

$$^* > \cdot > |$$

und lassen "$\cdot$" weg :-)

Beispiele:

$$((a \cdot b^*) \cdot a)$$

$$(a \mid b)$$

$$((a \cdot b) \cdot (a \cdot b))$$

Achtung:

- Wir unterscheiden zwischen Zeichen $a, 0, |, \ldots$ und Meta-Zeichen $(, |,), \ldots$
- Um (hässliche) Klammern zu sparen, benutzen wir Operator-Präzedenzen:

$$^* > \cdot > |$$

  und lassen "$\cdot$" weg :-)
- Reale Spezifikations-Sprachen bieten zusätzliche Konstrukte wie:

$$e? \equiv (\epsilon \mid e)$$
$$e^+ \equiv (e \cdot e^*)$$

  und verzichten auf "ϵ" :-)

Spezifikationen benötigen eine Semantik :-)

Im Beispiel:

Spezifikation	Semantik
ab^*a	$\{ab^na \mid n \geq 0\}$
$a \mid b$	$\{a, b\}$
$abab$	$\{abab\}$

Für $e \in \mathcal{E}_\Sigma$ definieren wir die spezifizierte Sprache $[\![e]\!] \subseteq \Sigma^*$ induktiv durch:

$$
\begin{array}{lcl}
[\![\epsilon]\!] & = & \{\epsilon\} \\
[\![a]\!] & = & \{a\} \\
[\![e^*]\!] & = & ([\![e]\!])^* \\
[\![e_1 | e_2]\!] & = & [\![e_1]\!] \cup [\![e_2]\!] \\
[\![e_1 \cdot e_2]\!] & = & [\![e_1]\!] \cdot [\![e_2]\!]
\end{array}
$$

Beachte:

- Die Operatoren $(_)^*, \cup, \cdot$ sind die entsprechenden Operationen auf Wort-Mengen:

$$
\begin{aligned}
(L)^* &= \{w_1 \ldots w_k \mid k \geq 0, w_i \in L\} \\
L_1 \cdot L_2 &= \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}
\end{aligned}
$$

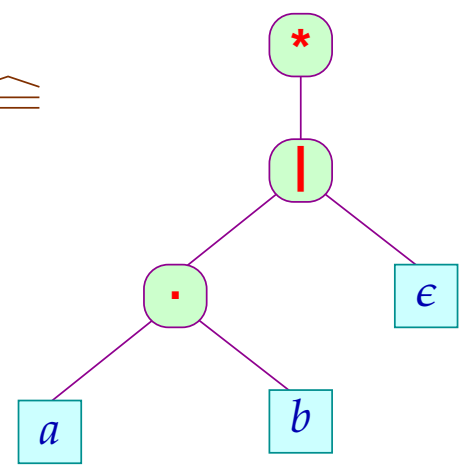
Beachte:

- Die Operatoren $(_)^*, \cup, \cdot$ sind die entsprechenden Operationen auf Wort-Mengen:

$$
\begin{aligned}
(L)^* &= \{w_1 \ldots w_k \mid k \geq 0, w_i \in L\} \\
L_1 \cdot L_2 &= \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}
\end{aligned}
$$

- Reguläre Ausdrücke stellen wir intern als markierte geordnete Bäume dar:

$(ab|\epsilon)^*$ $\widehat{=}$



Innere Knoten: Operator-Anwendungen;

Blätter: einzelne Zeichen oder ϵ.

Finger-Übung:

Zu jedem regulären Ausdruck e können wir einen Ausdruck e' (evt. mit "?") konstruieren so dass:

- $[\![e]\!] = [\![e']\!]$;
- Falls $[\![e]\!] = \{\epsilon\}$, dann ist $e' \equiv \epsilon$;
- Falls $[\![e]\!] \neq \{\epsilon\}$, dann enthält e' kein "ϵ".

Finger-Übung:

Zu jedem regulären Ausdruck e können wir einen Ausdruck e' (evt. mit "?") konstruieren so dass:

- $[\![e]\!] = [\![e']\!]$;
- Falls $[\![e]\!] = \{\epsilon\}$, dann ist $e' \equiv \epsilon$;
- Falls $[\![e]\!] \neq \{\epsilon\}$, dann enthält e' kein "ϵ".

Konstruktion:

Wir definieren eine Transformation $\mathcal{T}$ von regulären Ausdrücken durch:

$$
\begin{array}{lcl}
\mathcal{T}[\epsilon] & = & \epsilon \\
\mathcal{T}[a] & = & a \\
\mathcal{T}[e_1|e_2] & = & \text{case } (\mathcal{T}[e_1], \mathcal{T}[e_2]) \text{ of } \quad (\epsilon, \epsilon) \ : \ \epsilon \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad \mid \quad (e_1', \epsilon) : \ e_1'? \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad \mid \quad (\epsilon, e_2') : \ e_2'? \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad \mid \quad (e_1', e_2'): \ (e_1' \mid e_2') \\
\mathcal{T}[e_1 \cdot e_2] & = & \text{case } (\mathcal{T}[e_1], \mathcal{T}[e_2]) \text{ of } \quad (\epsilon, \epsilon) \ : \ \epsilon \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad \mid \quad (e_1', \epsilon) : \ e_1' \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad \mid \quad (\epsilon, e_2') : \ e_2' \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad \mid \quad (e_1', e_2'): \ (e_1' \cdot e_2') \\
\mathcal{T}[e^*] & = & \text{case } \mathcal{T}[e] \text{ of } \quad \epsilon : \ \epsilon \\
 & & \qquad\qquad\qquad\qquad \mid \quad e_1: \ e_1{}^* \\
\mathcal{T}[e?] & = & \text{case } \mathcal{T}[e] \text{ of } \quad \epsilon : \ \epsilon \\
 & & \qquad\qquad\qquad\qquad \mid \quad e_1: \ e_1?
\end{array}
$$

Unsere Anwendung:

Identifier in Java:

```
le = [a-zA-Z_\$]
di = [0-9]
Id = {le} ({le} | {di})*
```

.

Unsere Anwendung:

Identifier in Java:

```
le = [a-zA-Z_\$]
di = [0-9]
Id = {le} ({le} | {di})*
```

Bemerkungen:

- "`le`" und "`di`" sind Zeichenklassen.
- Definierte Namen werden in "`{`", "`}`" eingeschlossen.
- Zeichen werden von Meta-Zeichen durch "\" unterschieden.

Unsere Anwendung:

Identifier in Java:

```
le = [a-zA-Z_\$]
di = [0-9]
Id = {le} ({le}|{di})*
```

Gleitkommazahlen:

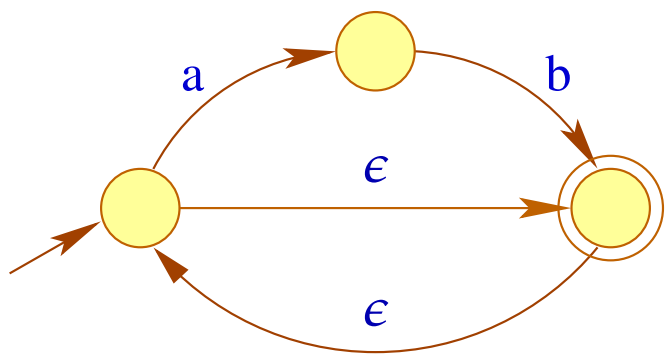
```
Float = {di}* (\.{di}|{di}\.) {di}*((e|E)(\+|\-)?{di}+)?
```

Bemerkungen:

- "`le`" und "`di`" sind Zeichenklassen.
- Definierte Namen werden in "`{`", "`}`" eingeschlossen.
- Zeichen werden von Meta-Zeichen durch "`\`" unterschieden.

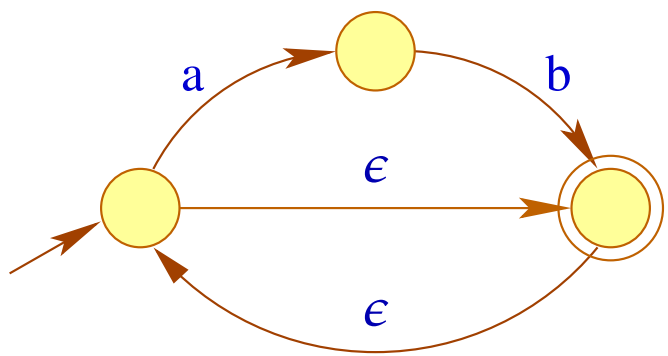
1.2 Grundlagen: Endliche Automaten

Beispiel:



1.2 Grundlagen: Endliche Automaten

Beispiel:



Knoten: Zustände;

Kanten: Übergänge;

Beschriftungen: konsumierter Input :-)

Michael O. Rabin, Stanford University

Dana S. Scott, Carnegy Mellon University, Pittsburgh

Formal ist ein nicht-deterministischer endlicher Automat mit ϵ-Übergängen (ϵ-NFA) ein Tupel $A = (Q, \Sigma, \delta, I, F)$ wobei:

Q	eine endliche Menge von Zuständen;
Σ	ein endliches Eingabe-Alphabet;
$I \subseteq Q$	die Menge der Anfangszustände;
$F \subseteq Q$	die Menge der Endzustände und
δ	die Menge der Übergänge (die Übergangs-Relation) ist.

Formal ist ein nicht-deterministischer endlicher Automat mit ϵ-Übergängen (ϵ-NFA) ein Tupel $A = (Q, \Sigma, \delta, I, F)$ wobei:

Q	eine endliche Menge von Zuständen;
Σ	ein endliches Eingabe-Alphabet;
$I \subseteq Q$	die Menge der Anfangszustände;
$F \subseteq Q$	die Menge der Endzustände und
δ	die Menge der Übergänge (die Übergangs-Relation) ist.

Für ϵ-NFAs ist:

$$\delta \quad \subseteq \quad Q \times (\Sigma \cup \{\epsilon\}) \times Q$$

Formal ist ein nicht-deterministischer endlicher Automat mit ϵ-Übergängen (ϵ-NFA) ein Tupel $A = (Q, \Sigma, \delta, I, F)$ wobei:

Q	eine endliche Menge von Zuständen;
Σ	ein endliches Eingabe-Alphabet;
$I \subseteq Q$	die Menge der Anfangszustände;
$F \subseteq Q$	die Menge der Endzustände und
δ	die Menge der Übergänge (die Übergangs-Relation) ist.

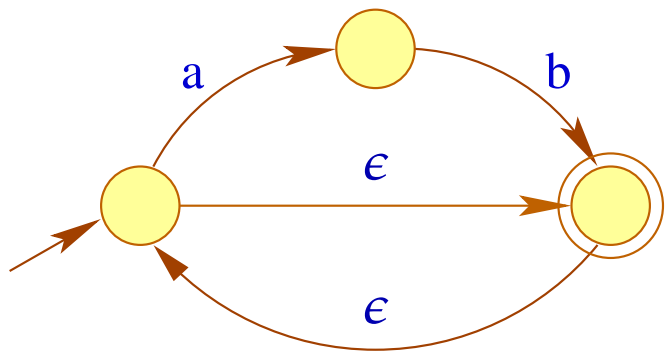
Für ϵ-NFAs ist:

$$\delta \quad \subseteq \quad Q \times (\Sigma \cup \{\epsilon\}) \times Q$$

- Gibt es keine ϵ-Übergänge (p, ϵ, q), ist A ein NFA.
- Ist $\delta : Q \times \Sigma \to Q$ eine Funktion und $\#I = 1$, heißt A deterministisch (DFA).

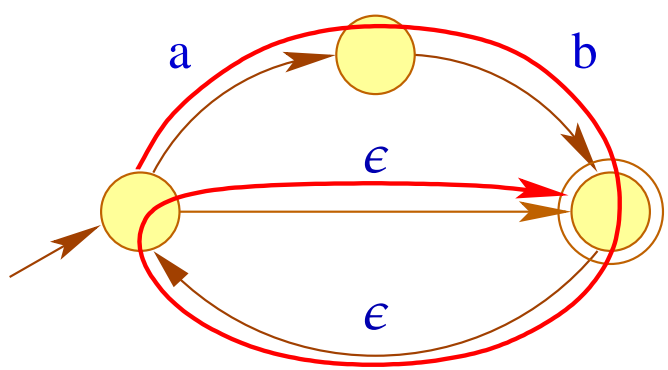
Akzeptierung

- 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:

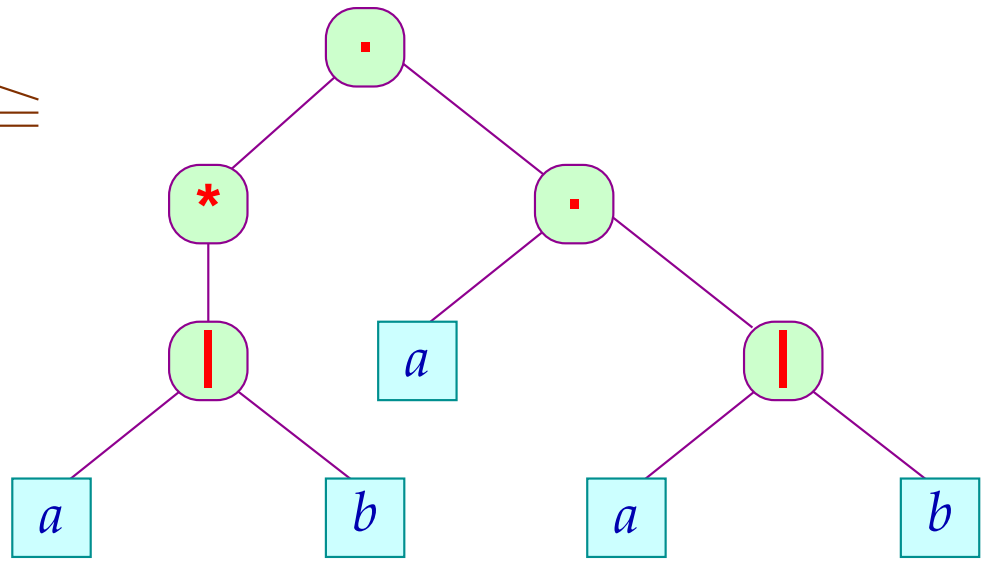
Für jeden regulären Ausdruck e kann (in linearer Zeit :-) ein ϵ-NFA konstruiert werden, der die Sprache $[\![e]\!]$ akzeptiert.

Idee:

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

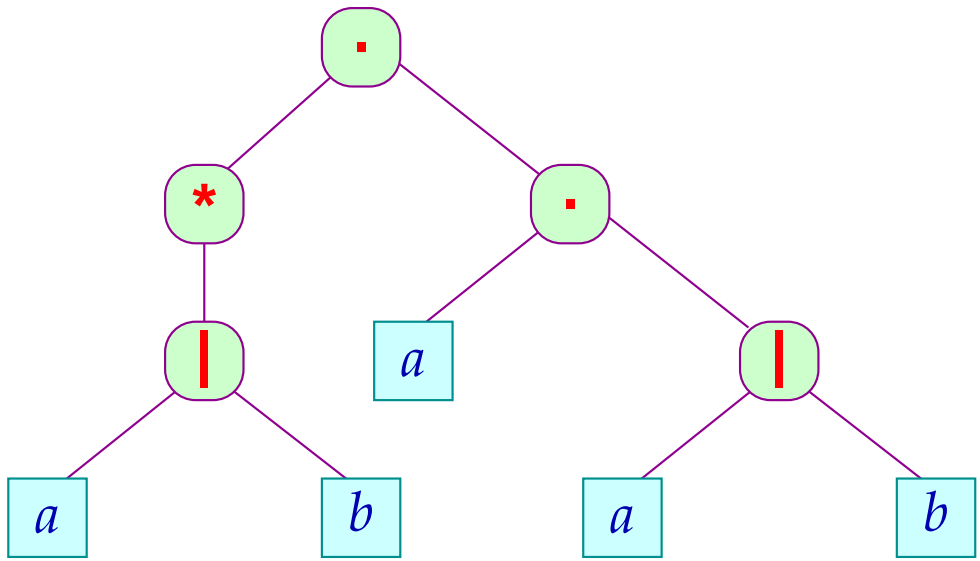
Beispiel:

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



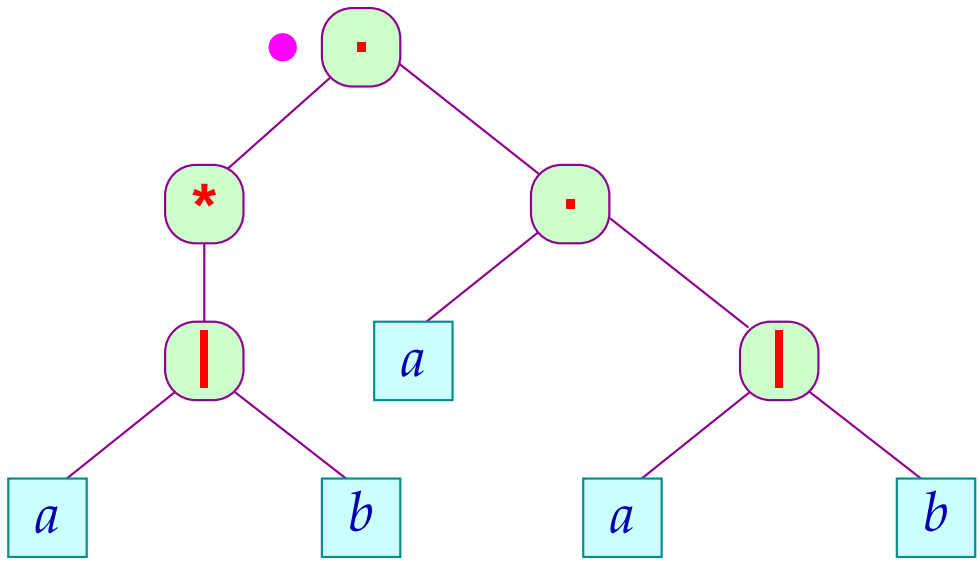
Beispiel:

$w \quad = \quad bbaa \quad :$



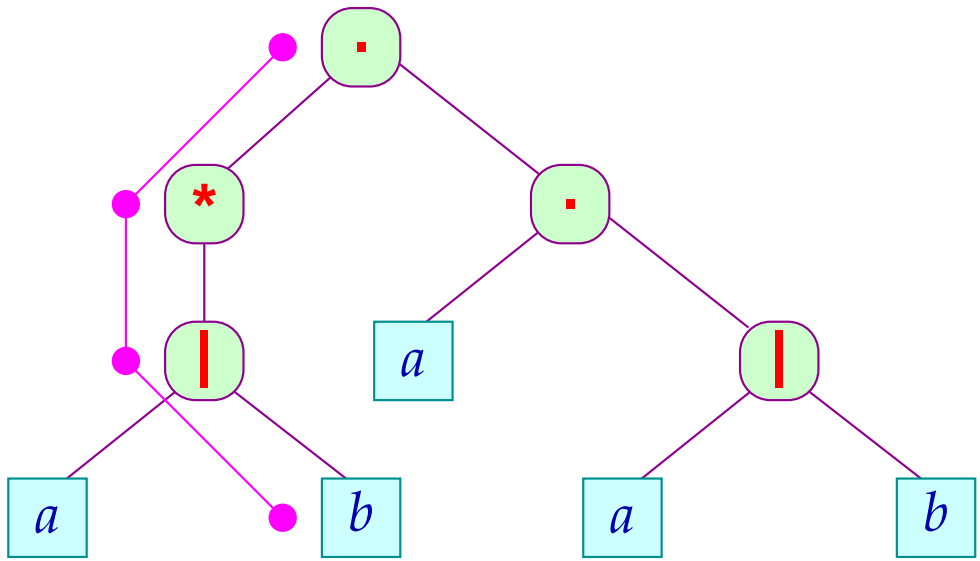
Beispiel:

$w \quad = \quad bbaa \quad :$



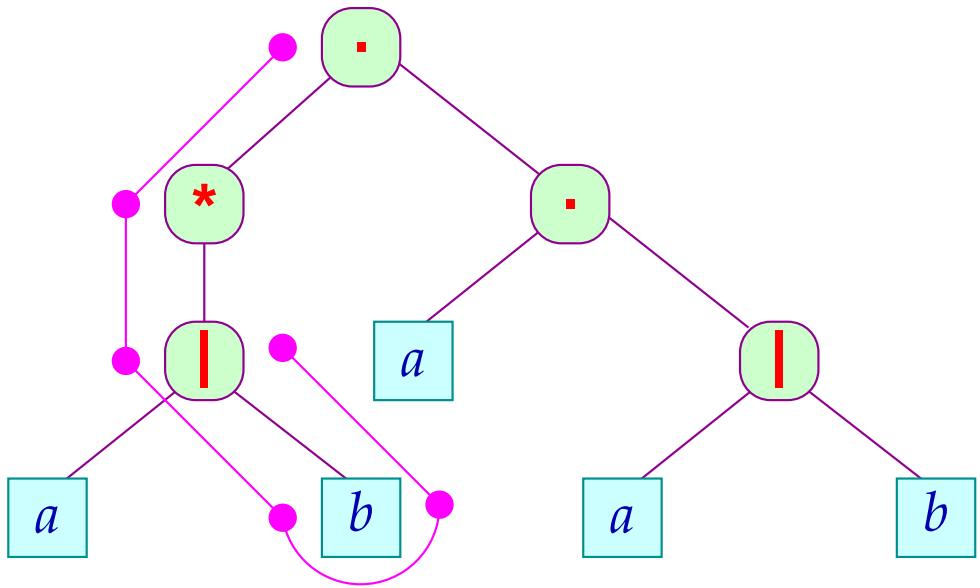
Beispiel:

$w \quad = \quad bbaa \quad :$



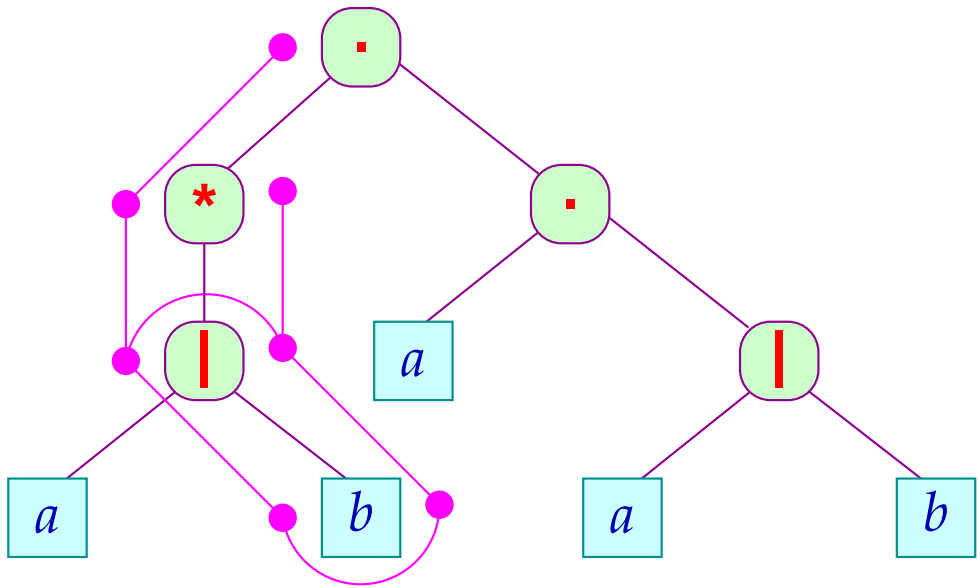
Beispiel:

$w \quad = \quad bbaa \quad :$



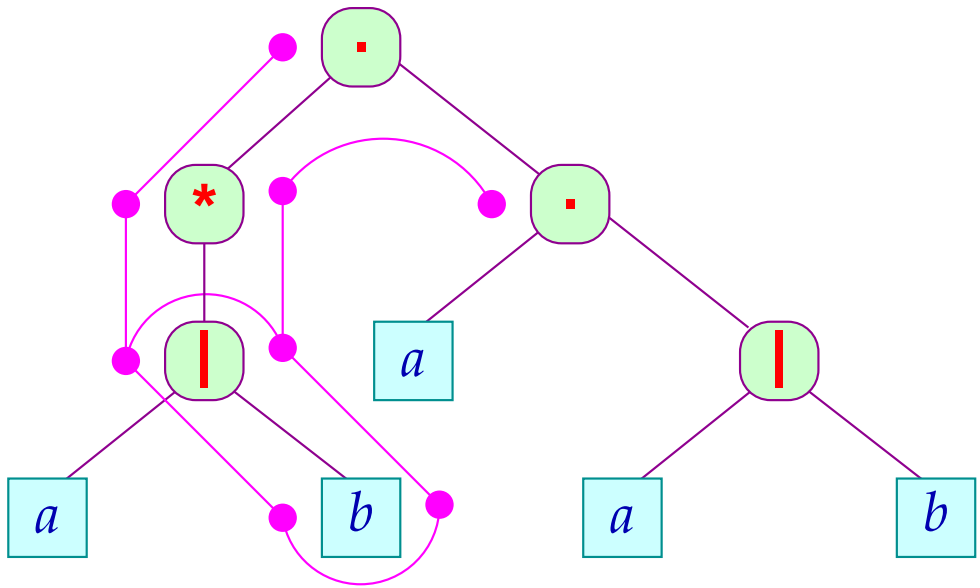
Beispiel:

$w \quad = \quad bbaa \quad :$



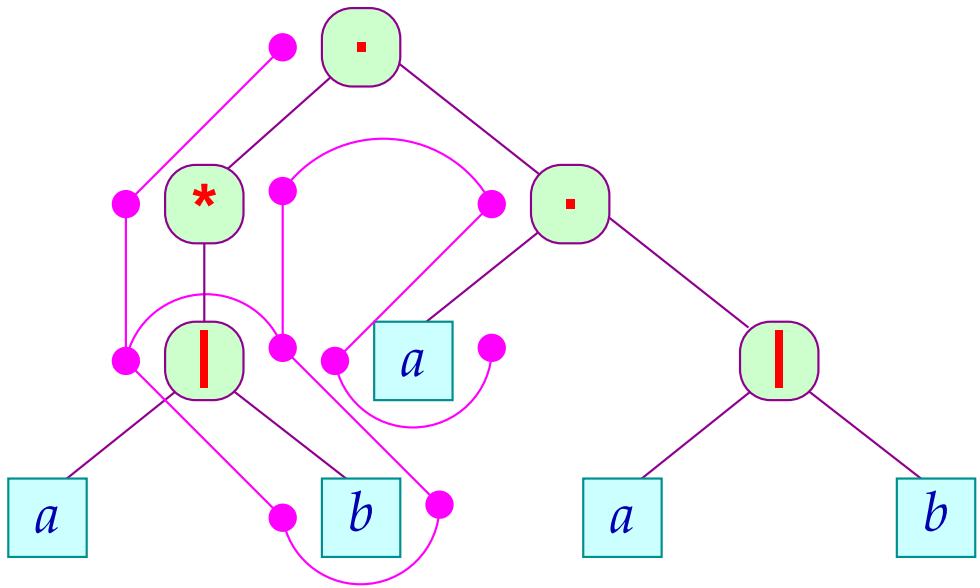
Beispiel:

$w \quad = \quad bbaa \quad :$



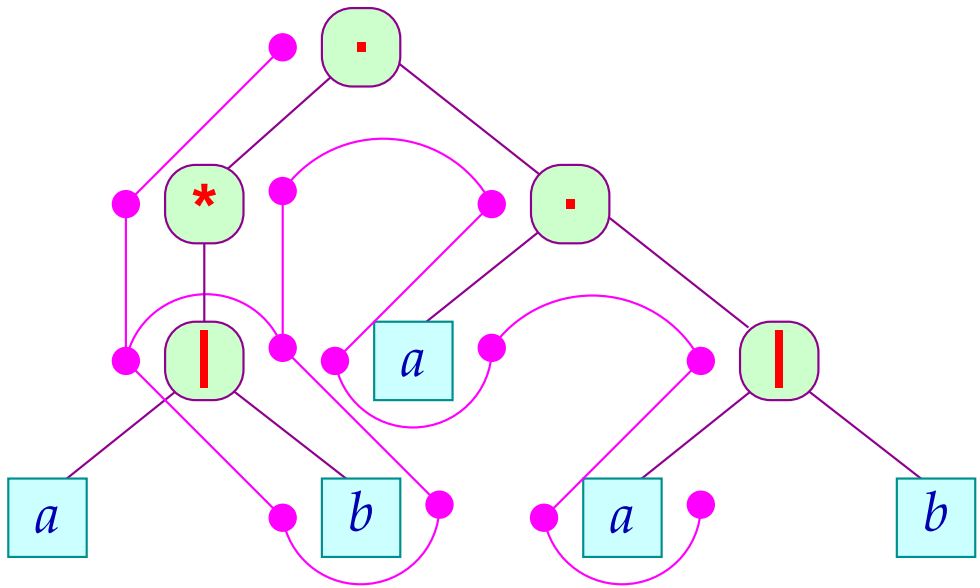
Beispiel:

$w \quad = \quad bbaa \quad :$



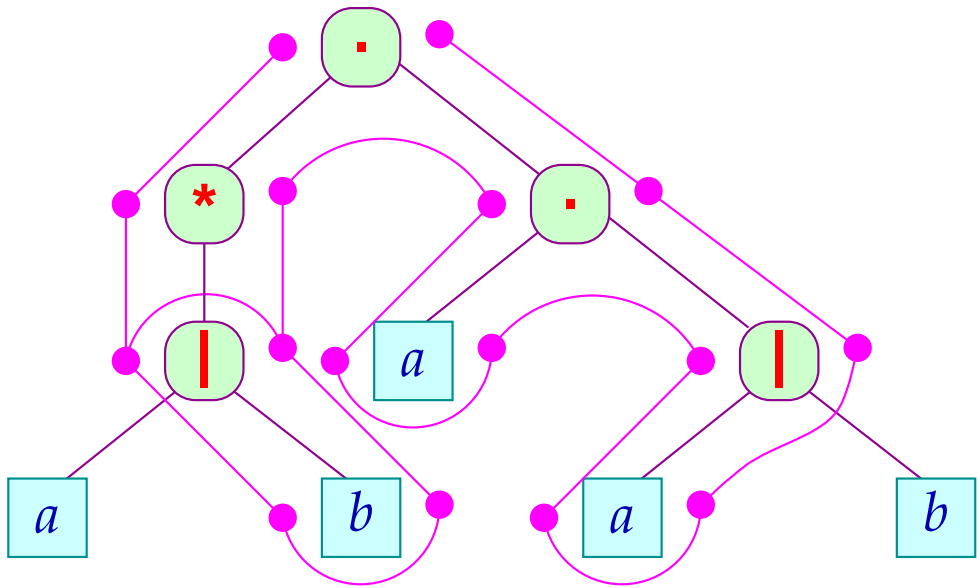
Beispiel:

$w \quad = \quad bbaa \quad :$



Beispiel:

$w \quad = \quad bbaa \quad :$

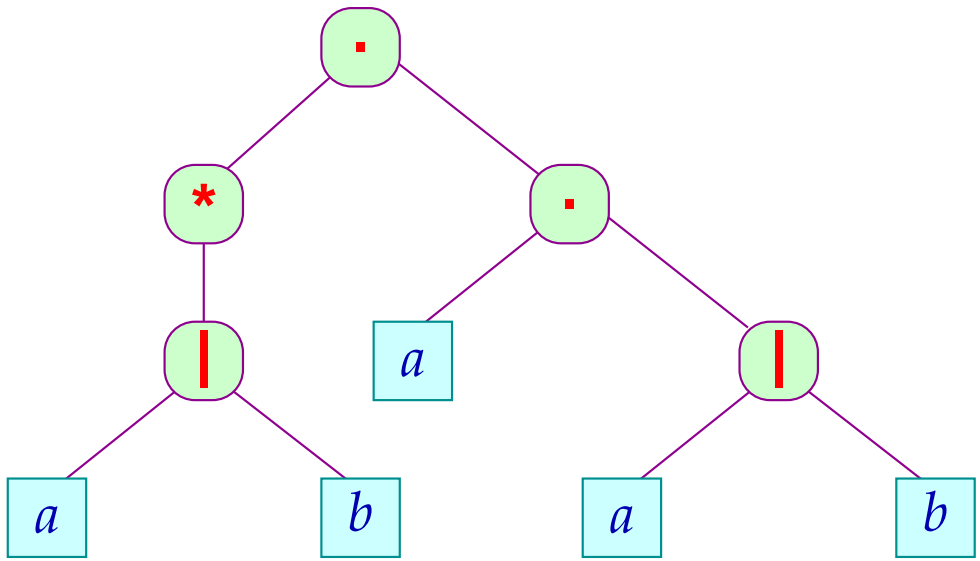


Beachte:

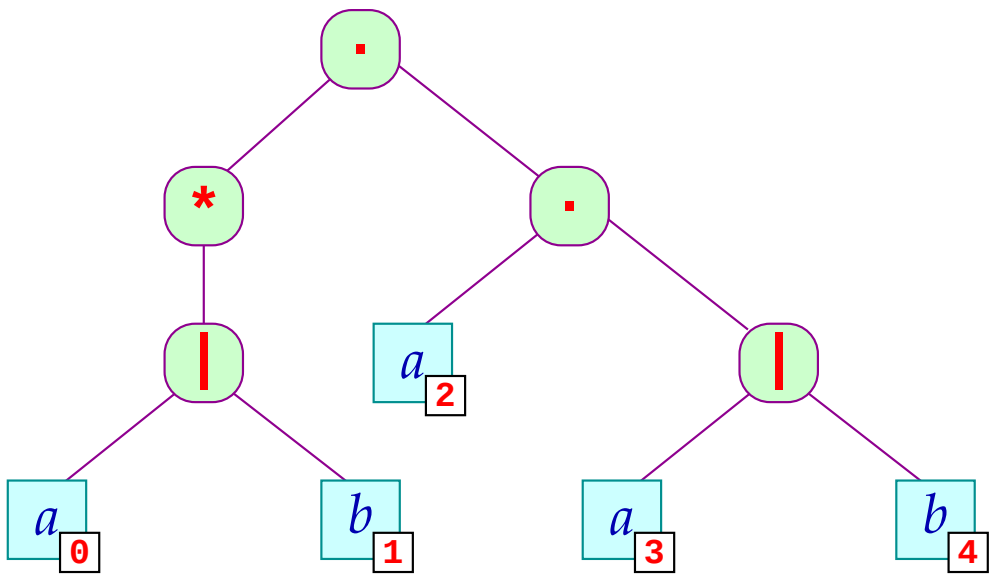
- Gelesen wird nur an den Blättern.
- Die Navigation im Baum erfolgt ohne Lesen, d.h. mit ϵ-Übergängen.
- Für eine formale Konstruktion müssen wir die Knoten im Baum bezeichnen.
- Dazu benutzen wir (hier) einfach den dargestellten Teilausdruck :-)
- Leider gibt es eventuell mehrere gleiche Teilausdrücke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

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

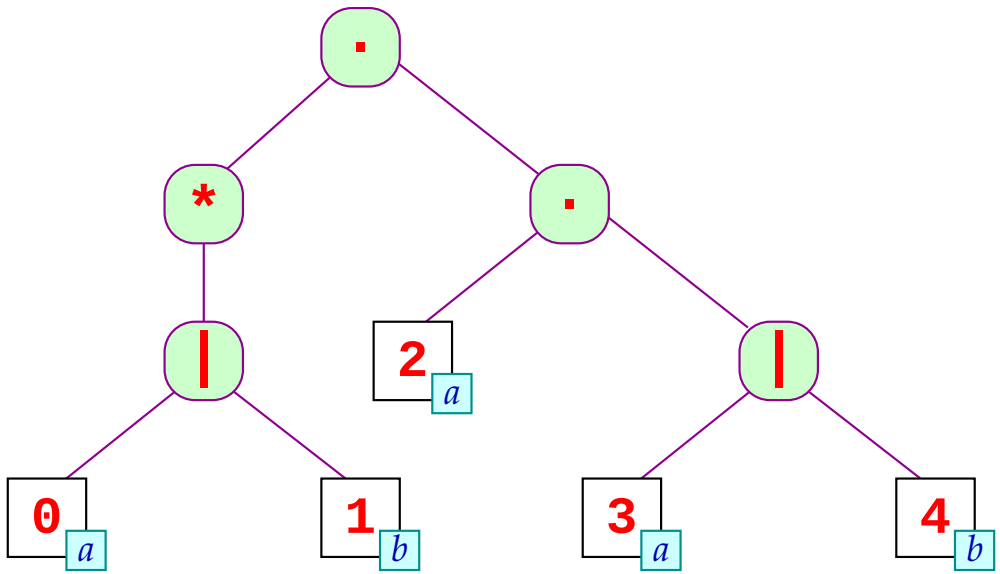
... im Beispiel:



... 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 :-(

$\Longrightarrow$

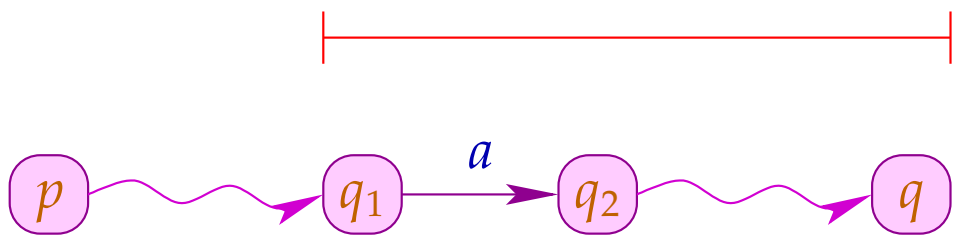
Strategie:

(1) Beseitigung der ϵ-Übergänge;

(2) Beseitigung des Nichtdeterminismus :-)

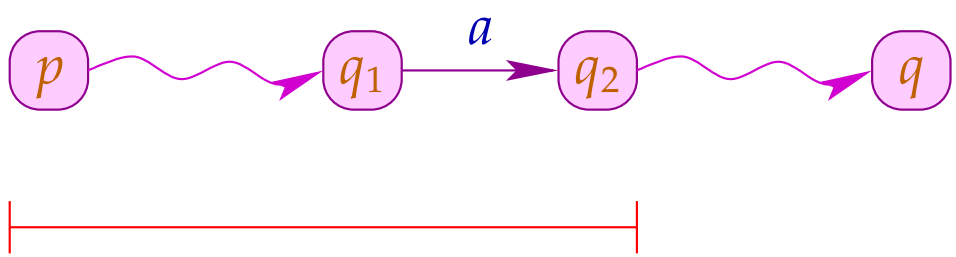
Beseitigung von ϵ-Übergängen:

Zwei einfache Ansätze:



Beseitigung von ϵ-Übergängen:

Zwei einfache Ansätze:



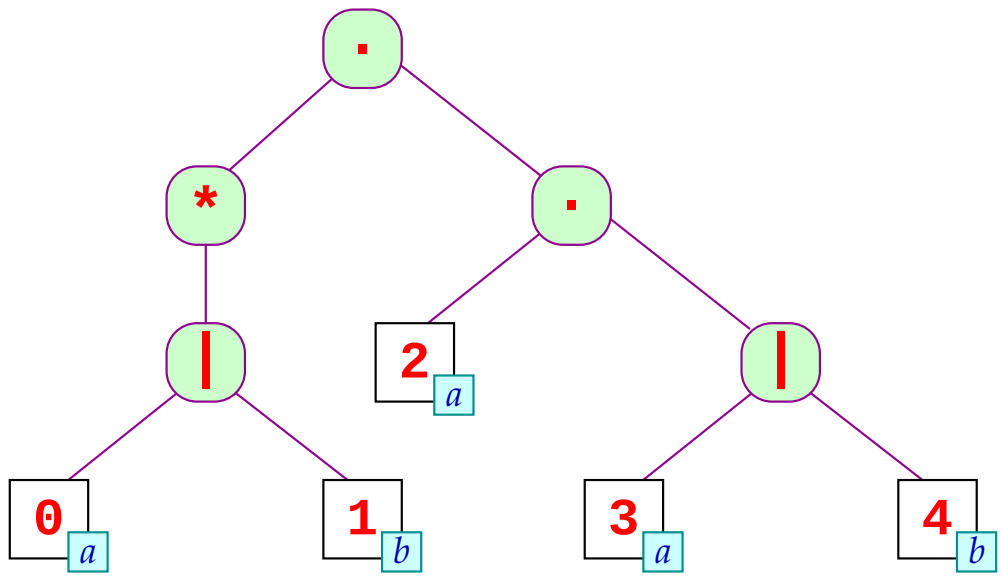
Wir benutzen hier den zweiten Ansatz.

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

1. Schritt:

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

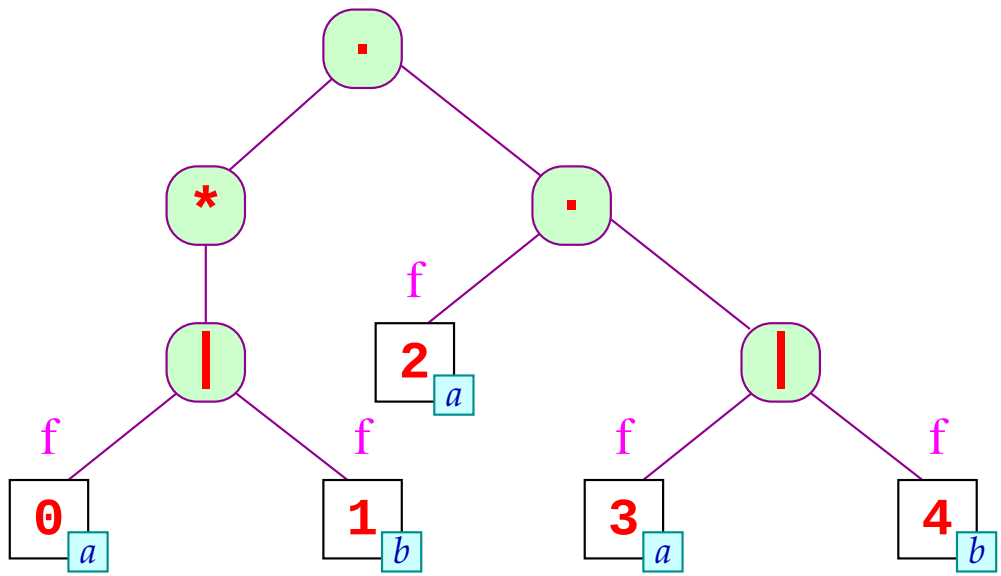
... im Beispiel:



1. Schritt:

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

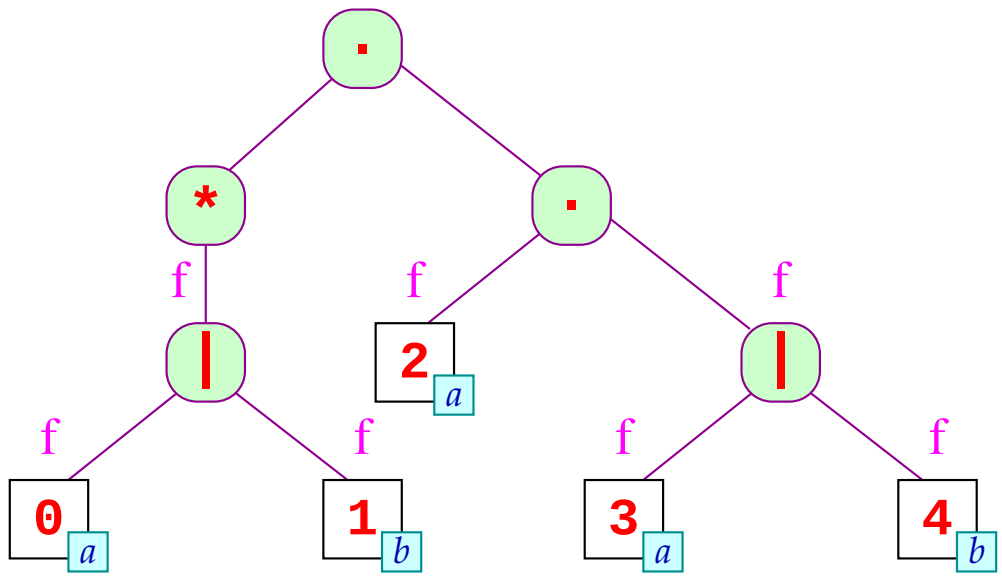
... im Beispiel:



1. Schritt:

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

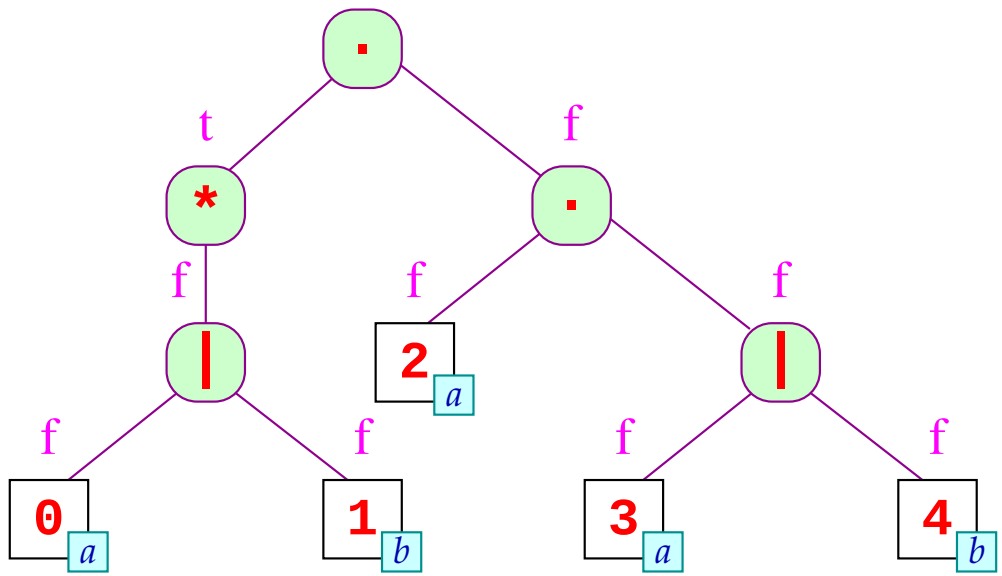
... im Beispiel:



1. Schritt:

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

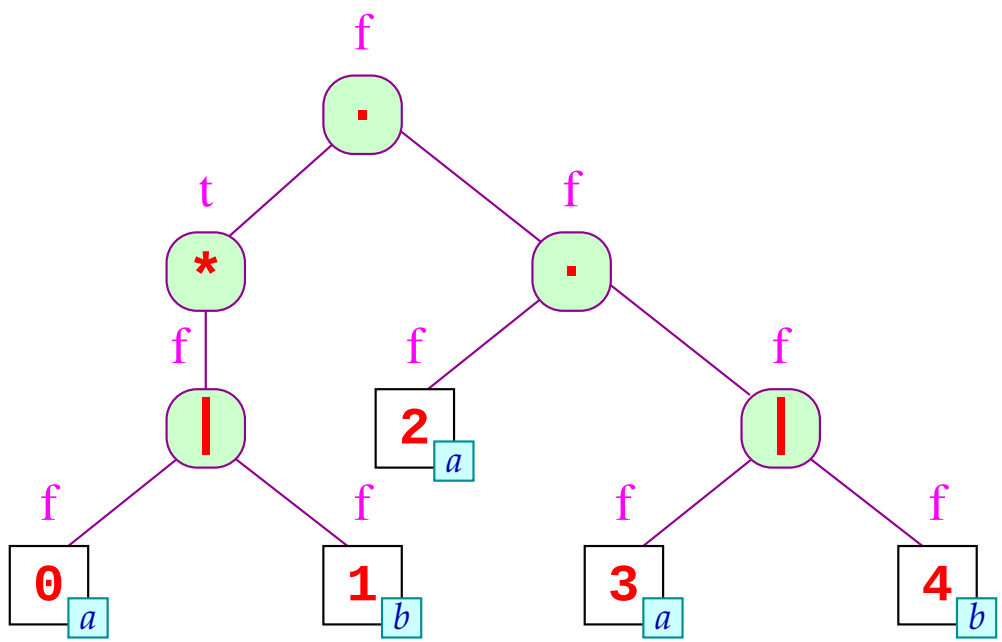
... im Beispiel:



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 $\mathsf{empty}[r] = (x \equiv \epsilon)$.

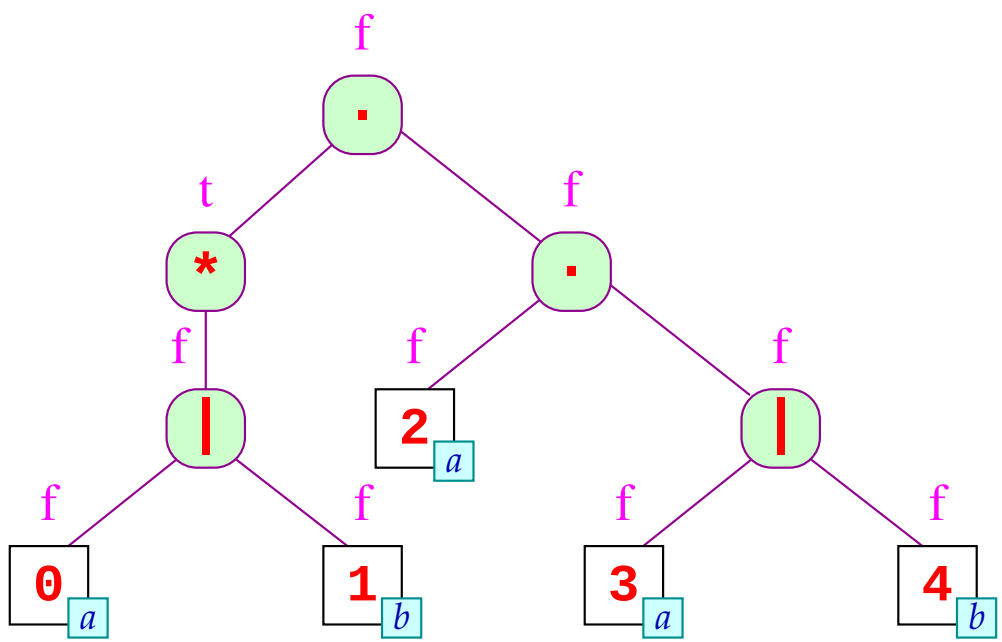
Andernfalls:

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

2. Schritt:

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

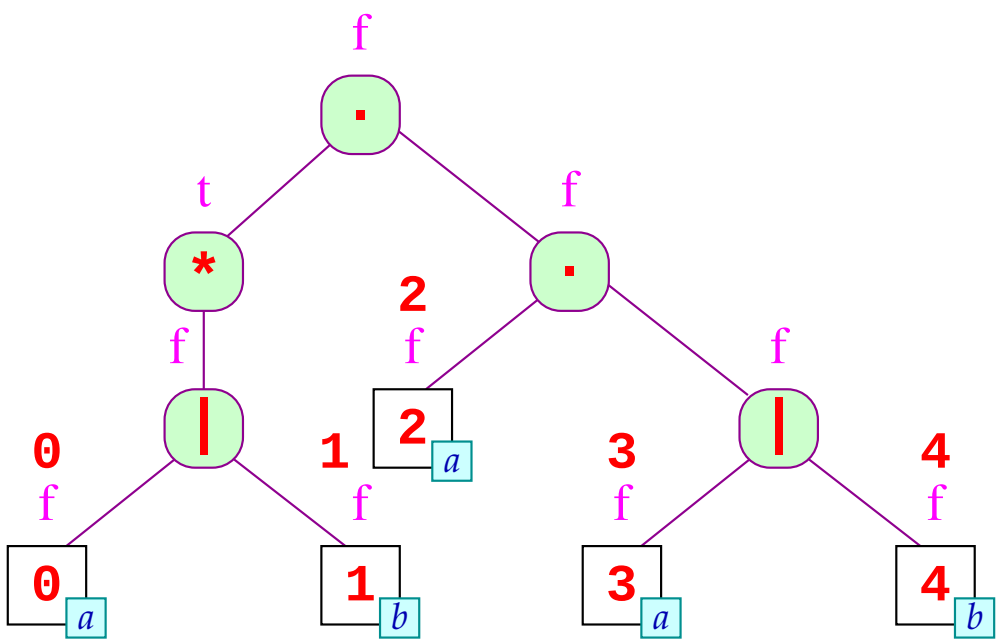
... im Beispiel:



2. Schritt:

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

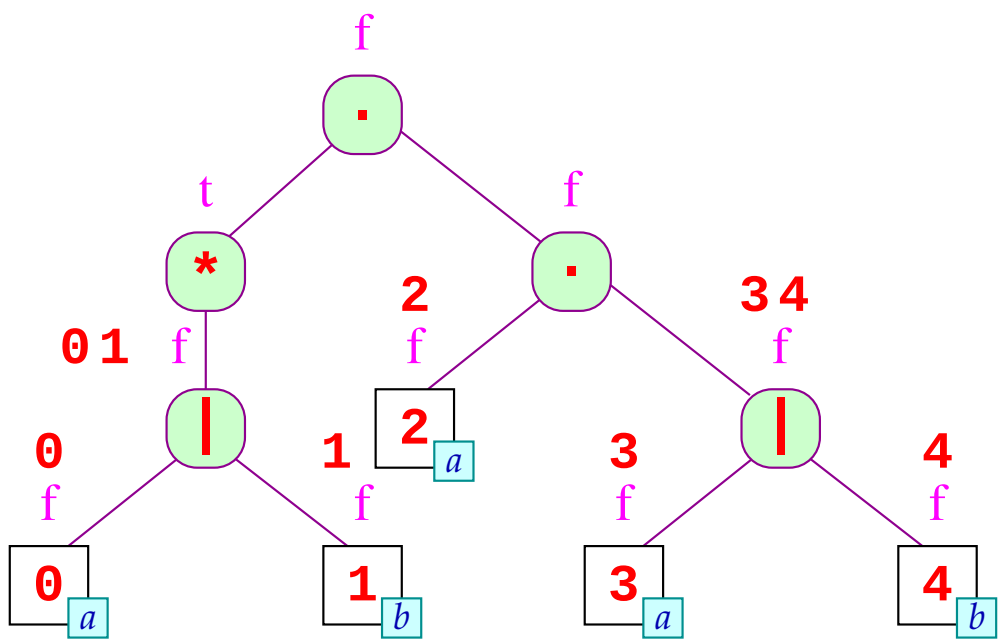
... im Beispiel:



2. Schritt:

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

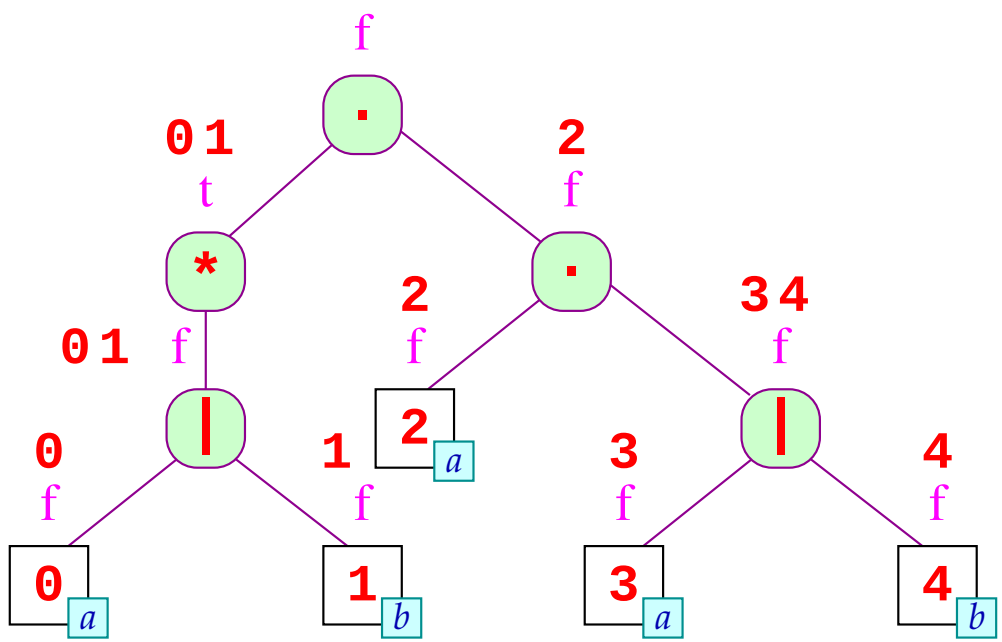
... im Beispiel:



2. Schritt:

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

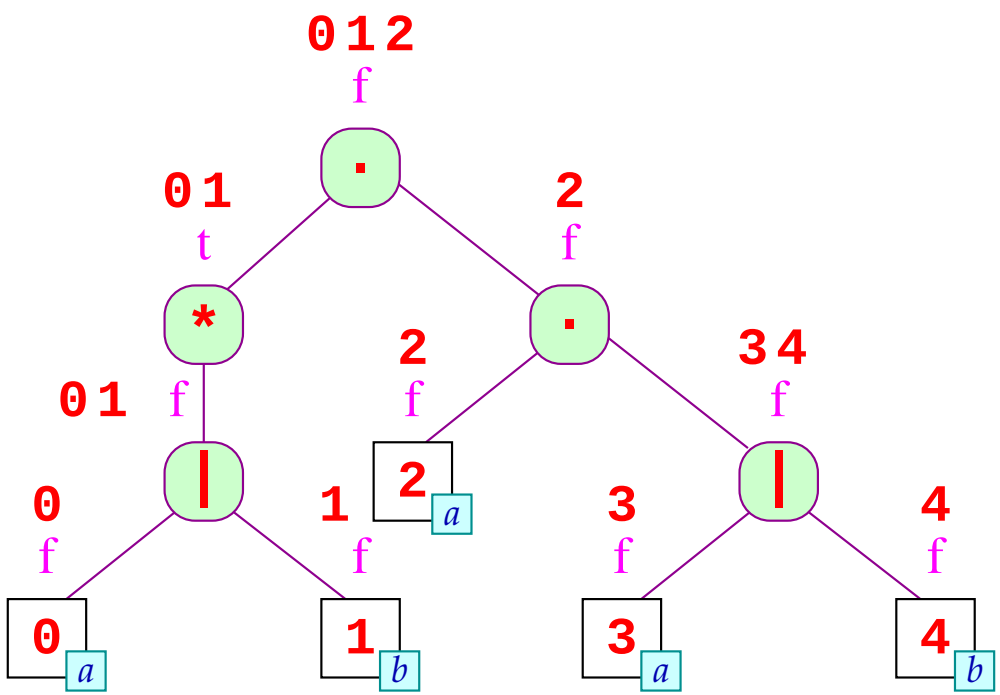
... im Beispiel:



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

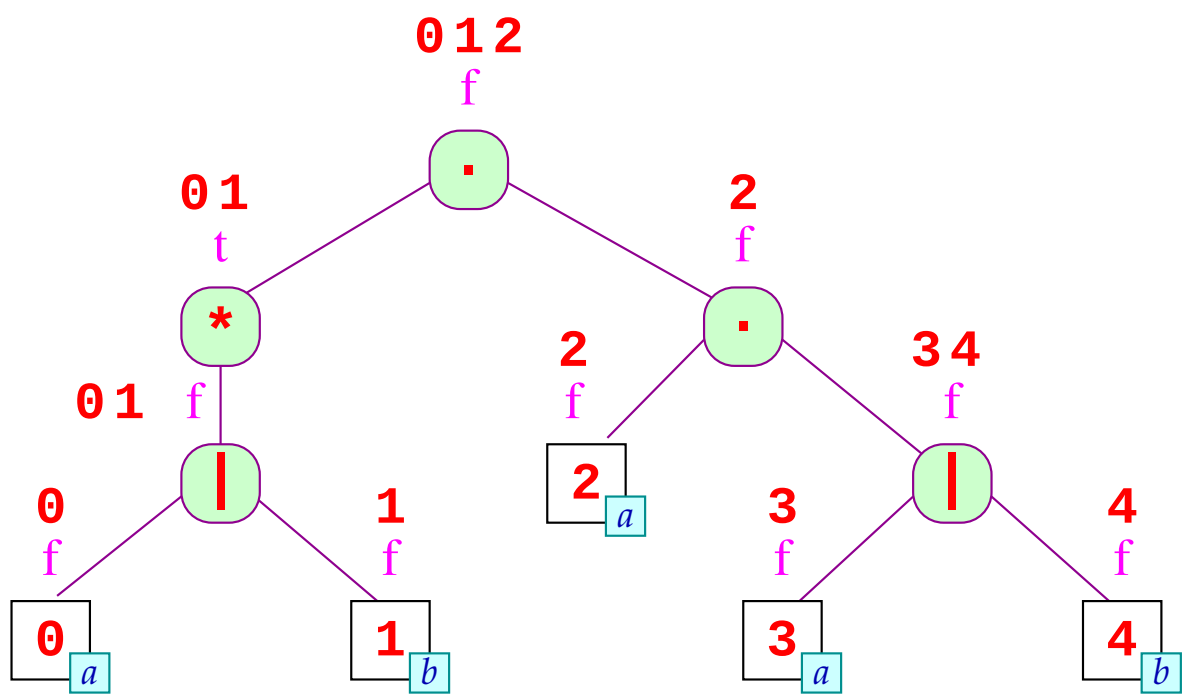
Andernfalls:

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

3. Schritt:

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

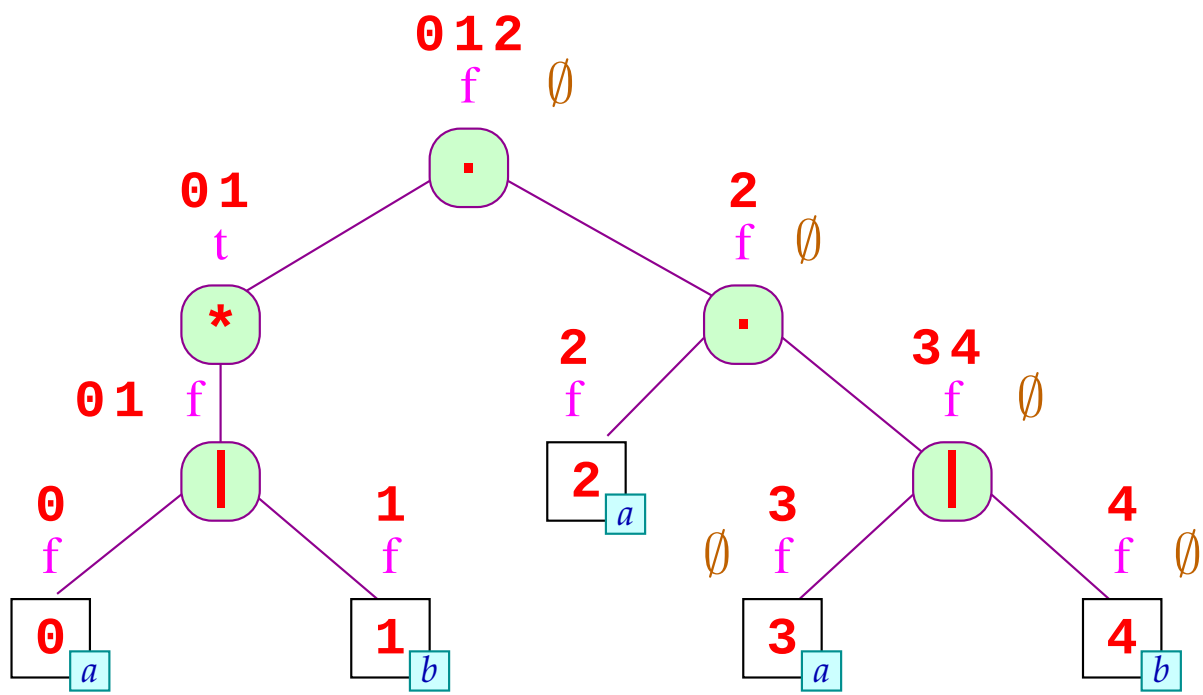
... im Beispiel:



3. Schritt:

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

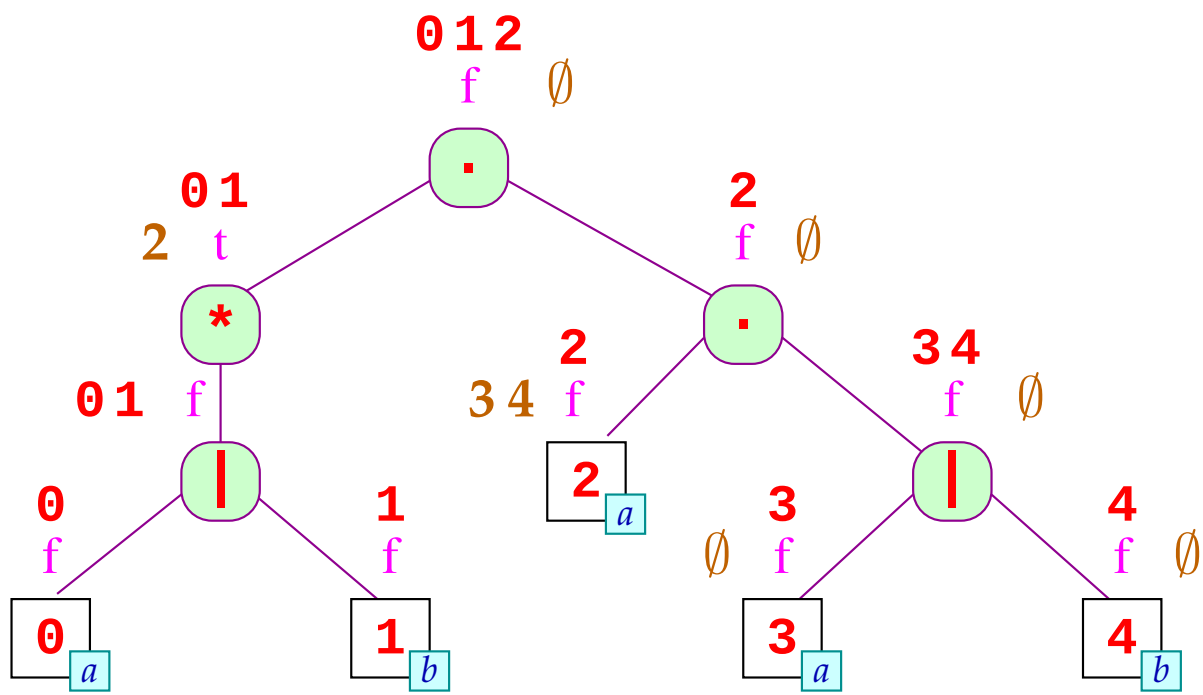
... im Beispiel:



3. Schritt:

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

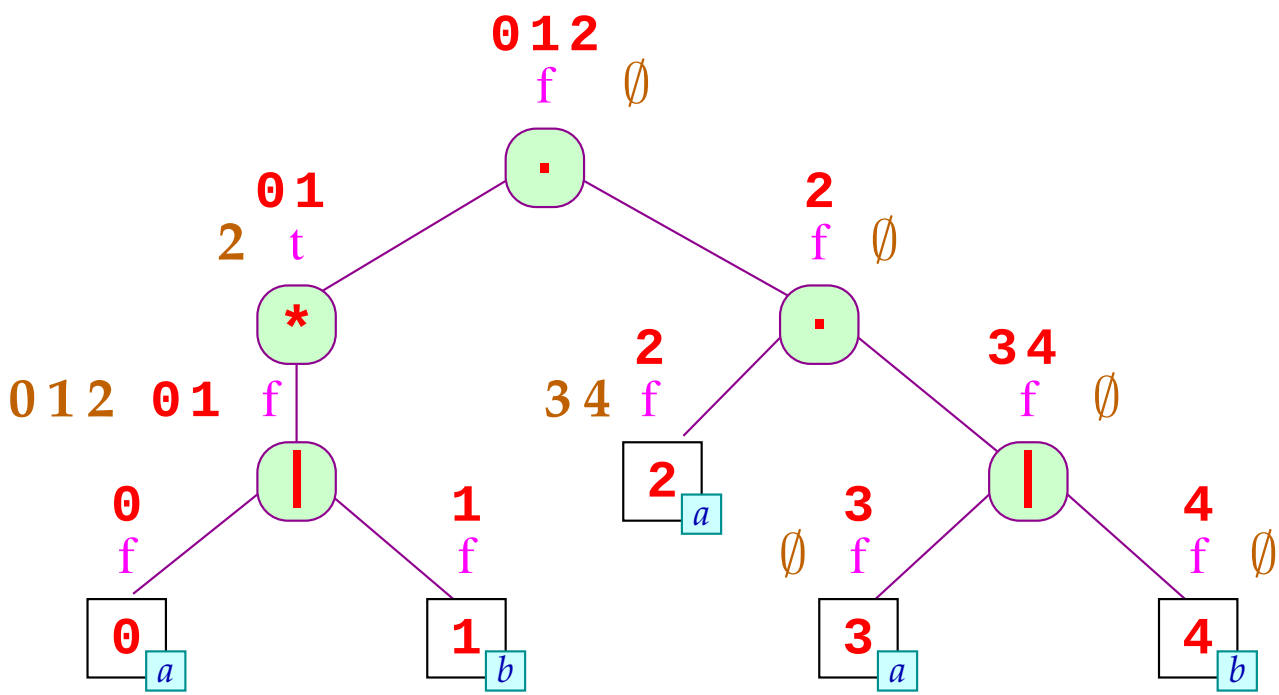
... im Beispiel:



3. Schritt:

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

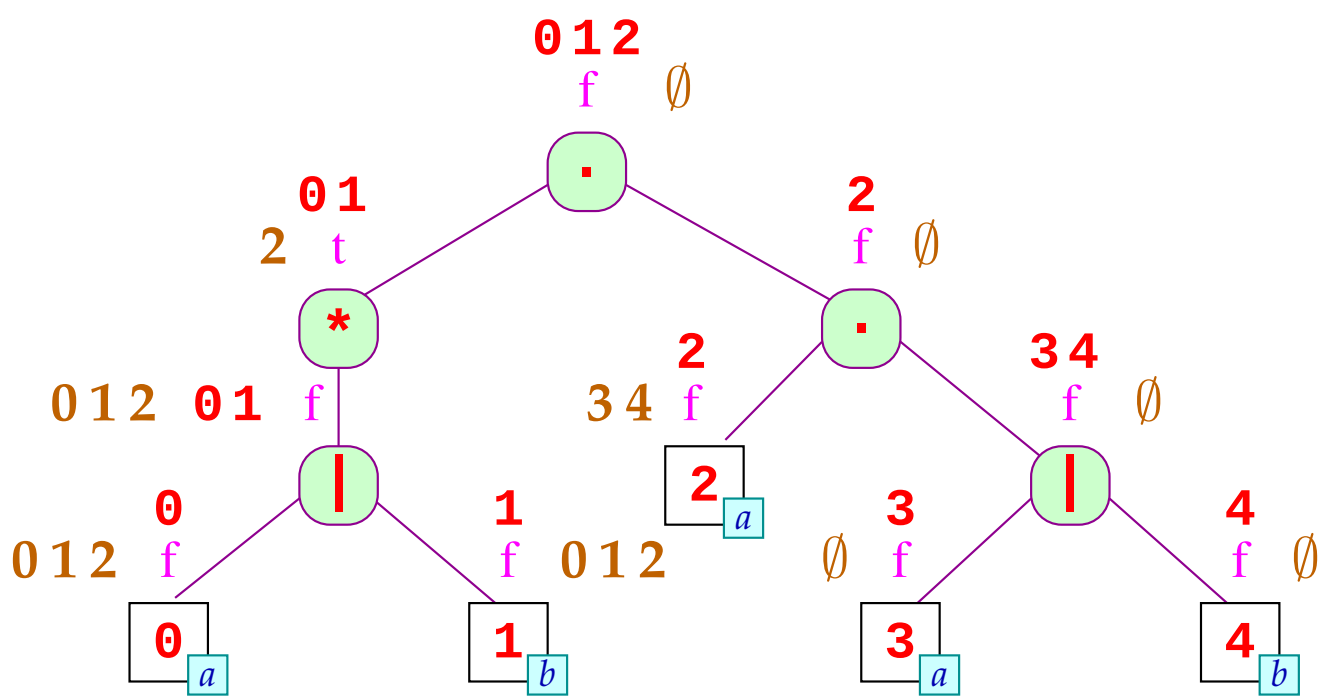
... im Beispiel:



3. Schritt:

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

... im Beispiel:



Implementierung: DFS pre-order Traversierung ;-)

Für die Wurzel haben wir:

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

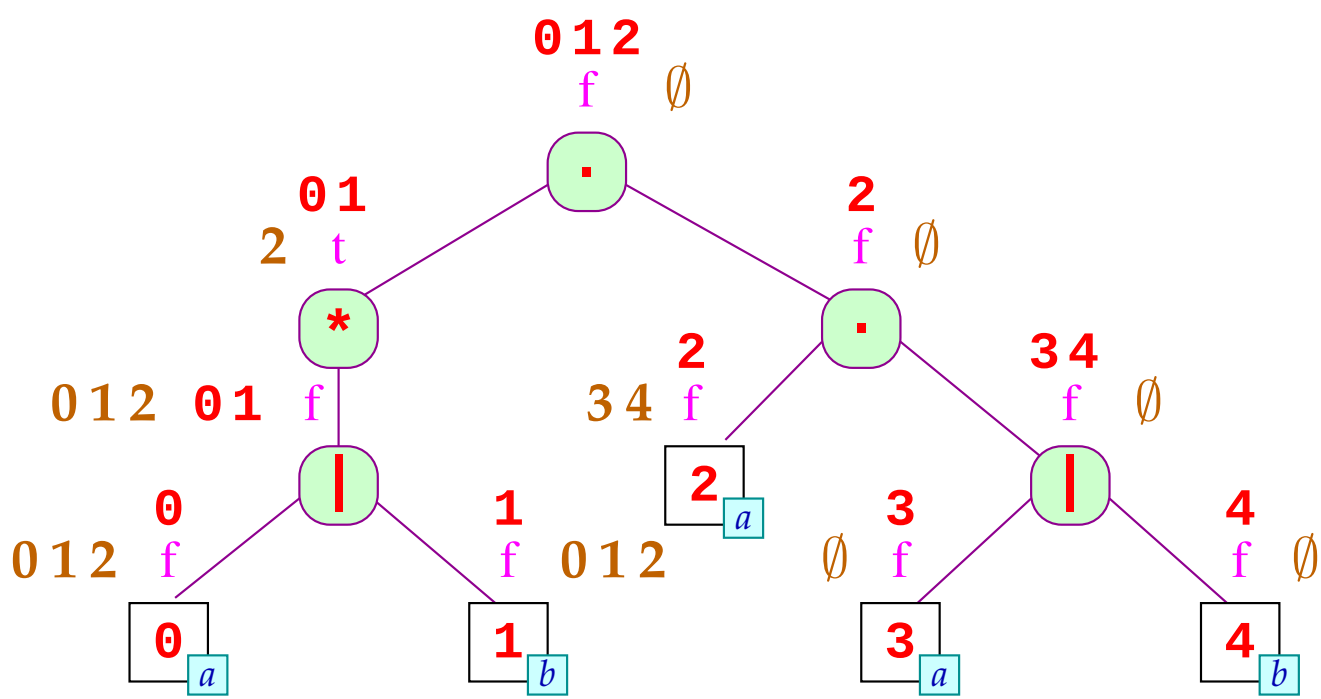
Ansonsten machen wir eine Fallunterscheidung über den Kontext:

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

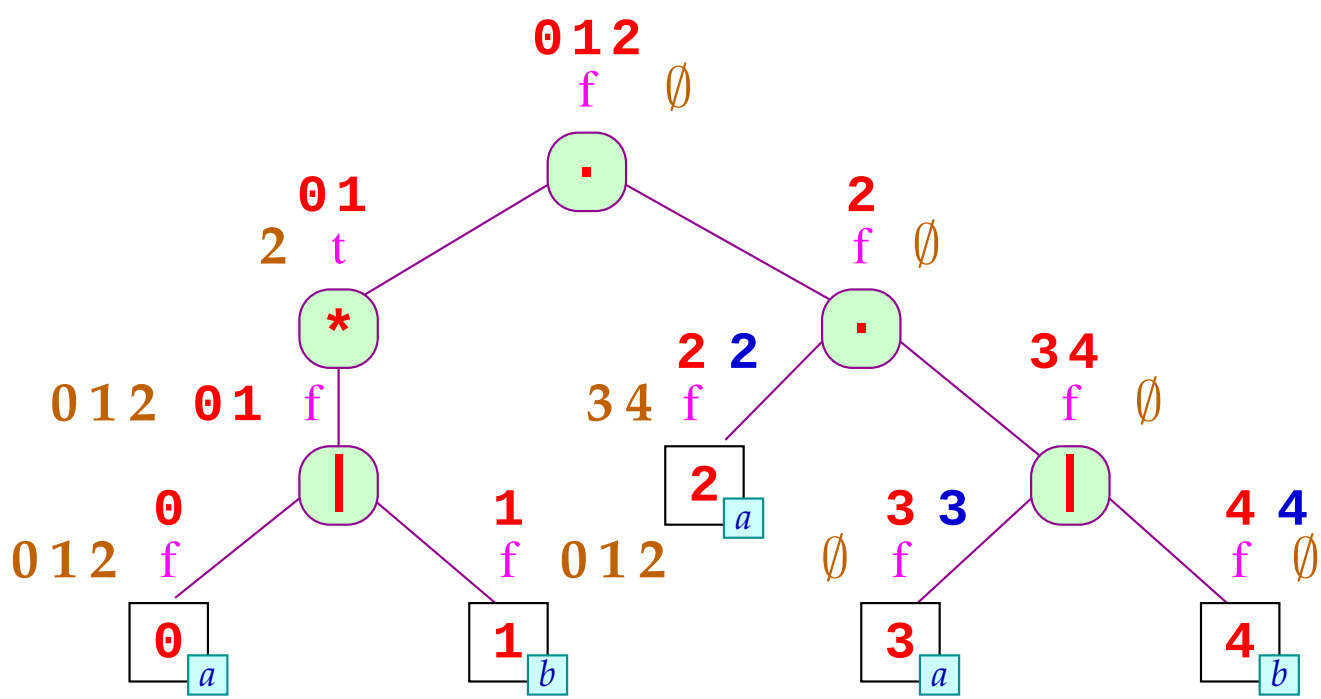
... im Beispiel:



4. Schritt:

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

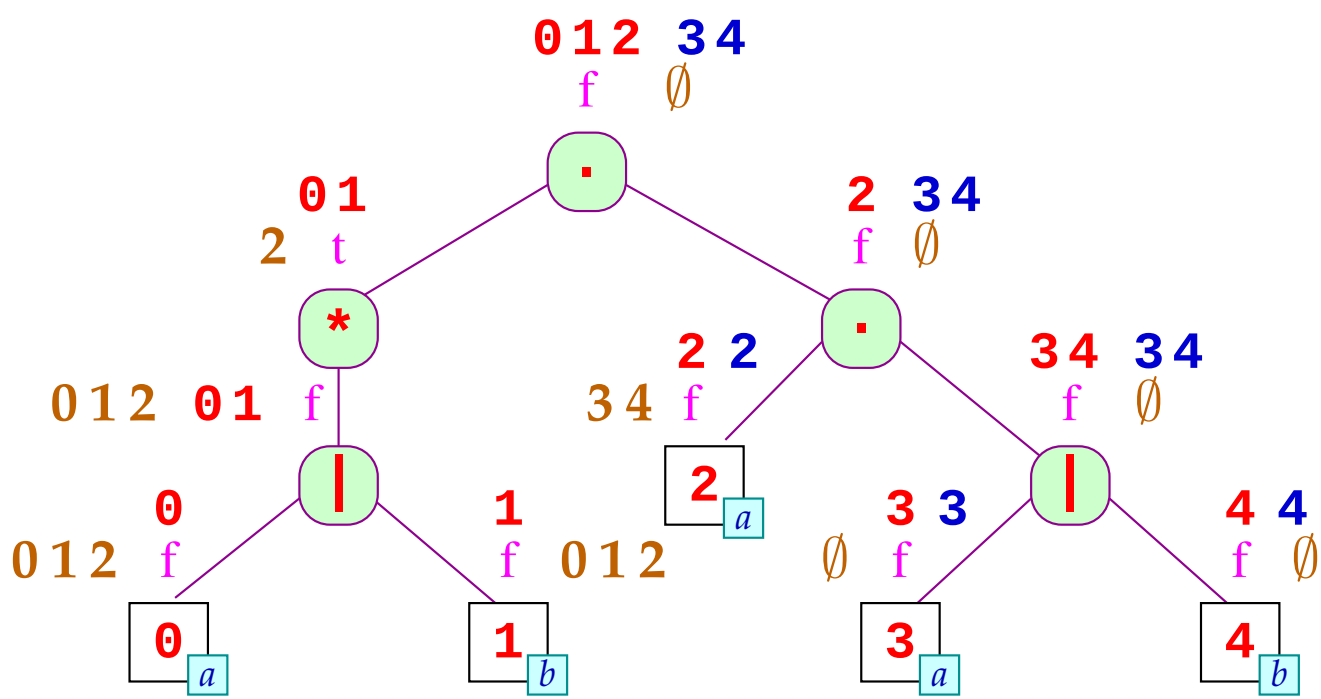
... im Beispiel:



4. Schritt:

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

... 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:

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

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.