Theoretische Informatik

Äquivalenzsatz und Anwendungen

Reguläre Sprachen

- reguläre Ausdrücke $\Rightarrow$ NFA
- DFA $\Rightarrow$ regulärer Ausdruck
- Äquivalenzsatz für reguläre Sprachen
- flex

© H. Peter Gumm, Philipps-Universität Marburg

Reguläre Ausdrücke

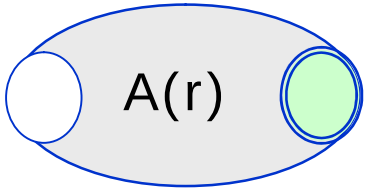
- Gegeben: Regulärer Ausdruck r
- Problem:
  - Suche nach Teilstrings, die auf r passen.
- Idee:
  - Konstruiere einen Automaten A_r mit $L(A_r) = L(r)$.
  - String gefunden, wenn A_r einen Endzustand erreicht
- Strategie:
  - Konstruiere zunächst einen Automaten mit ε-Transitionen
  - Wandele diesen in deterministischen Automaten um
  - Minimiere

© H. Peter Gumm, Philipps-Universität Marburg

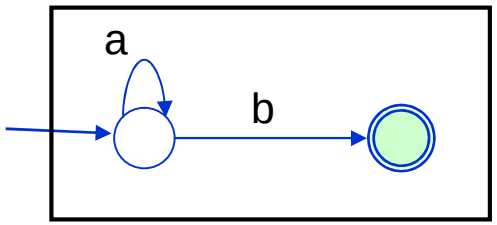
Reguläre Ausdrücke

Zu jedem regulären Ausdruck r gibt es einen endlichen Automaten A(r) mit L(r) = L(A(r)).

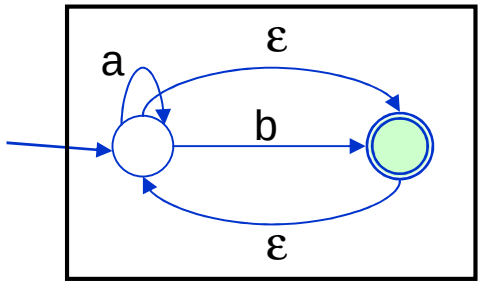
- Konstruktion induktiv über Aufbau von r



Ein bisschen Vorsicht ist notwendig:



Automat für
a*b

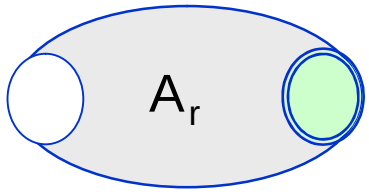


Nicht Automat für
(a*b)*

© H. Peter Gumm, Philipps-Universität Marburg

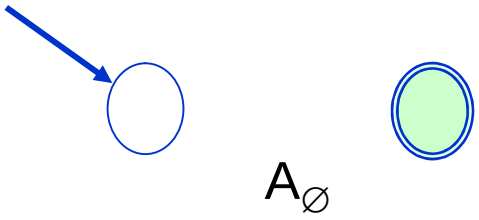
Induktive Konstruktion - Anfang

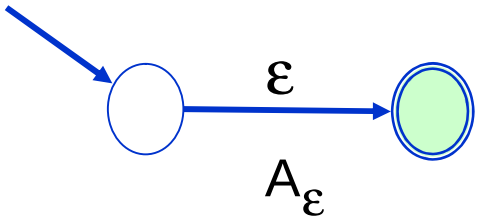
- Zu regulärem Ausdruck r konstruieren wir A_r mit
  - genau einem Endzustand q_f
  - keine Transition führt in den Anfangszustand
  - keine Transition verlässt den Endzustand

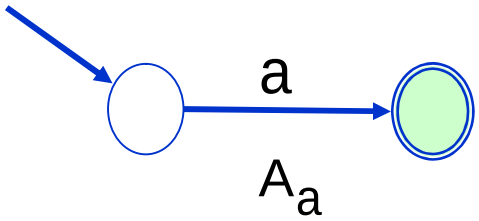


1. Konstante Reguläre Ausdrücke

$$\emptyset,\ \varepsilon,\ a \quad \text{für jedes } a \in \Sigma$$





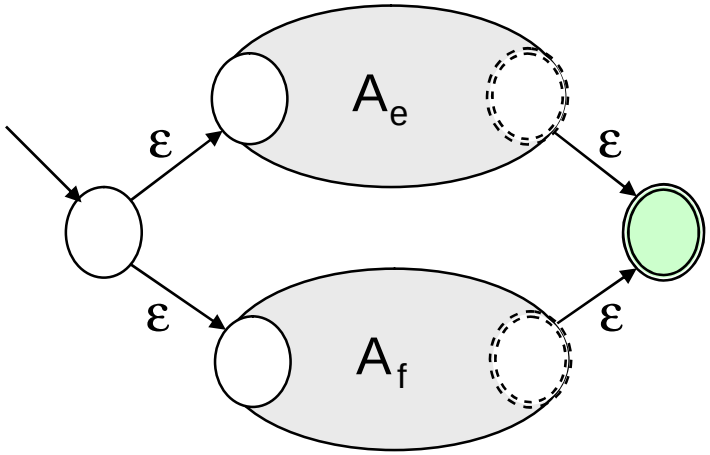


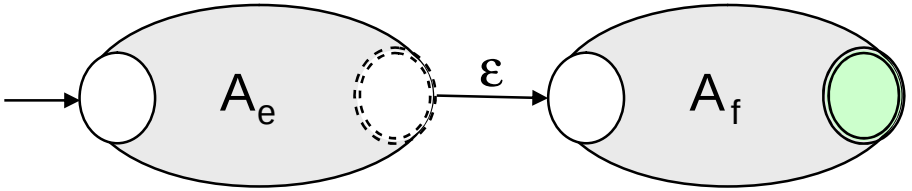
© H. Peter Gumm, Philipps-Universität Marburg

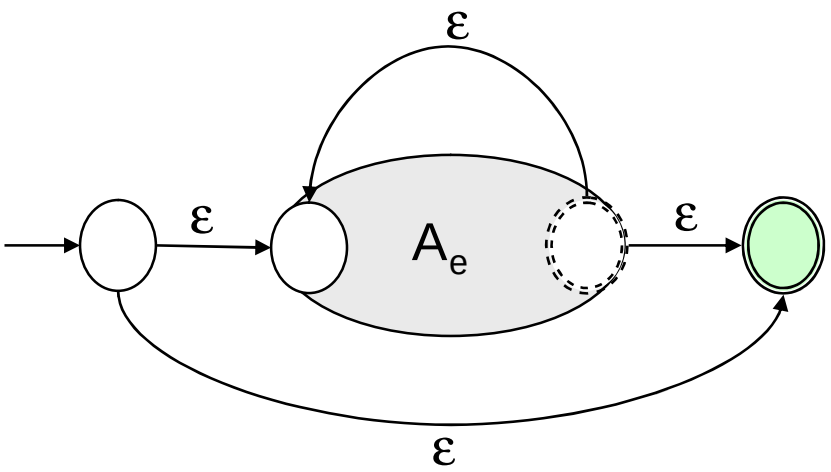
Induktive Konstruktion - Rekursion

2. Sind e und f reguläre Ausdrücke, dann auch

e+f, ef, e*



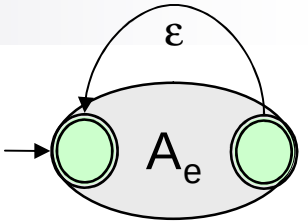




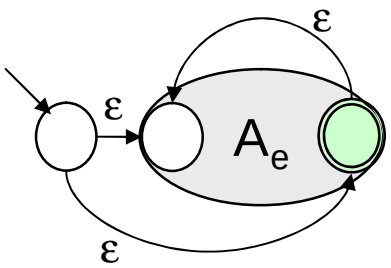
© H. Peter Gumm, Philipps-Universität Marburg

Aufgaben

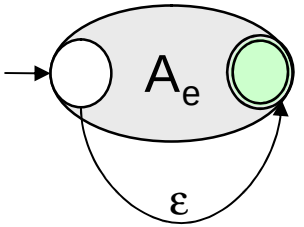
- Der Student Hans Acker findet die Konstruktion des Automaten für e* viel zu umständlich, weil zwei neue „sinnlose Zustände“ hinzukommen. Er schlägt vor
  - eine ε-Transition vom Endzustand in den Anfangszustand
  - den Anfangszustand auch zu einem Endzustand zu machen (Bild 1)



1. Zeigen Sie H. Acker, dass seine Optimierung einen Bug hat. Finden Sie einen möglichst einfachen regulären Ausdruck e, so dass seine Konstruktion einen Automaten liefert, der mehr Worte erkennt, als nur L(e*) .



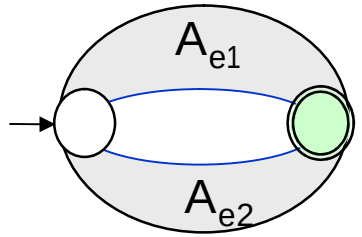
- Am nächsten Morgen hat H.Acker den Bug eingesehen und liefert sein verbessertes Programm. Es produziert den Automaten in Bild 2



2. Zeigen Sie ihm seinen Denkfehler.

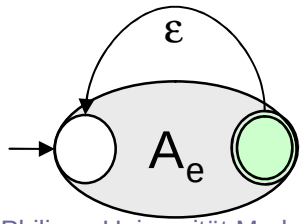
3. e? ist definiert als e+ε . Wie gewinnt man laut allgemeiner Konstruktion aus dem Automaten für e einen solchen für e?

4. H.Acker möchte optimieren und schlägt für e? den Automaten in Bild 3 vor.
   1. Was ist der Fehler ?
   2. Konstruieren Sie einen möglichst einfachen Automaten für e?



5. O.K, aber die Vereinigung, den Automaten für $e_1 + e_2$, könnte man doch sicher einfacher haben, indem man die Anfangs- und Endzustände identifiziert wie in Bild 4. Wo ist der Fehler ?

6. Gestern hätte H.Acker noch einen optimierten Automaten für e^+ gemäß Bild 5 konstruiert. Jetzt ist er total verunsichert. Geben Sie ihm seine Sicherheit zurück! Überzeugen Sie ihn, dass der Automat in Bild 5 tatsächlich e^+ implementiert.



© H. Peter Gumm, Philipps-Universität Marburg

Automat ⟩ reg. Ausdruck

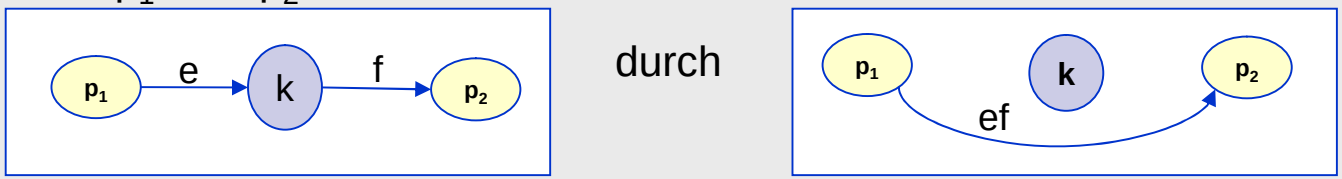
Zu jedem endlichen Automaten A gibt es einen regulären Ausdruck r_A mit $L(A)=L(r_A)$.

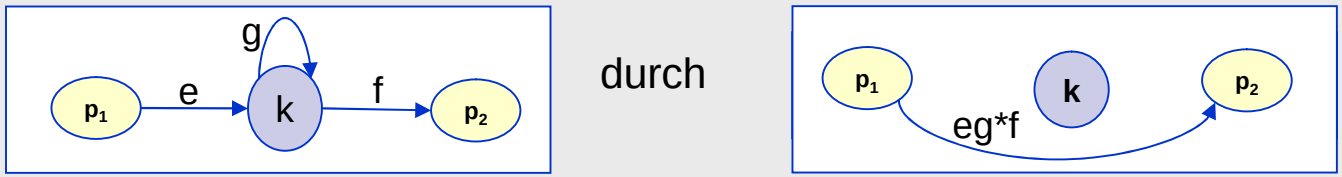
1. Ersetze Endzustände durch einen neuen Endzustand q_f

2. Elimination innerer Knoten $k \notin \{q_0, q_f\}$

Für je zwei Knoten $p_1 \neq k \neq p_2$

ersetze



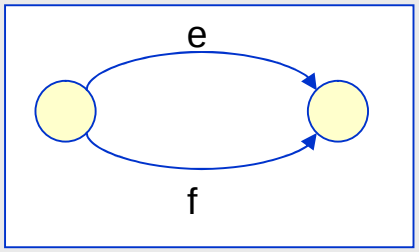


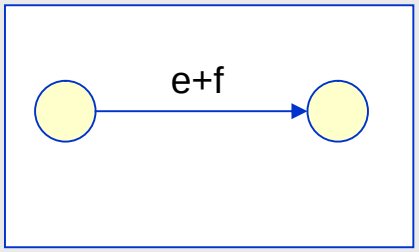
danach entferne k.

© H. Peter Gumm, Philipps-Universität Marburg

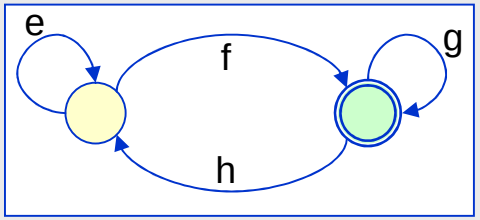
Automat ⇒ regulärer Ausdruck

3. Elimination paralleler Kanten

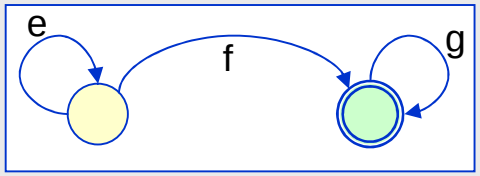
Ersetze 

durch 

4. Zum Schluss hat der Automat zwei Zustände. Ersetze ihn durch reg. Ausdruck

Ersetze 

durch `e*f(g + he*f)*`

bzw. 

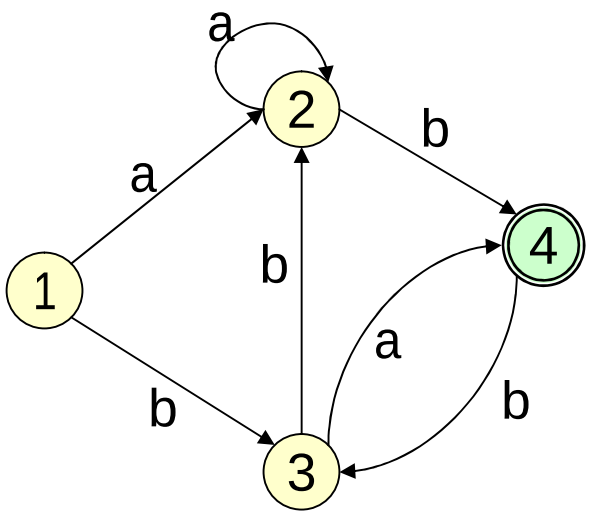
durch `e*fg*`

etc.

© H. Peter Gumm, Philipps-Universität Marburg

Beispiel

- Konstruiere regulären Ausdruck für den Automaten



Nur ein Endzustand – o.k.

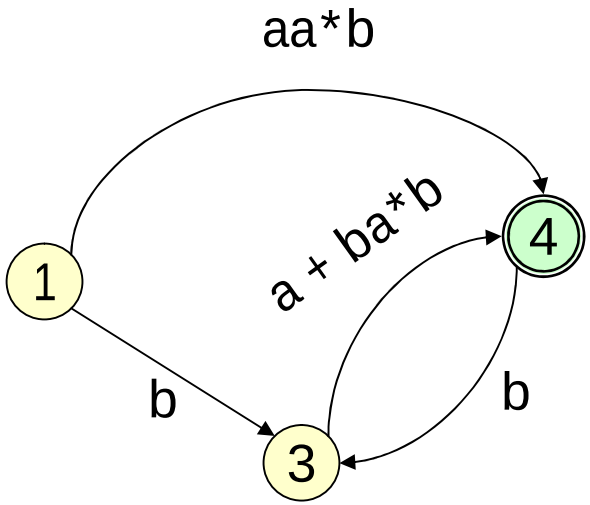
Eliminiere 2:

- Es gibt zwei Läufe über 2:
  - 1 aa*b 4 und 3 ba*b 4

© H. Peter Gumm, Philipps-Universität Marburg

Beispiel

- Konstruiere regulären Ausdruck für den Automaten



Nur ein Endzustand – o.k.

Eliminiere 2:

- Es gibt zwei Läufe über 2:
  - 1 aa*b 4 und 3 ba*b 4

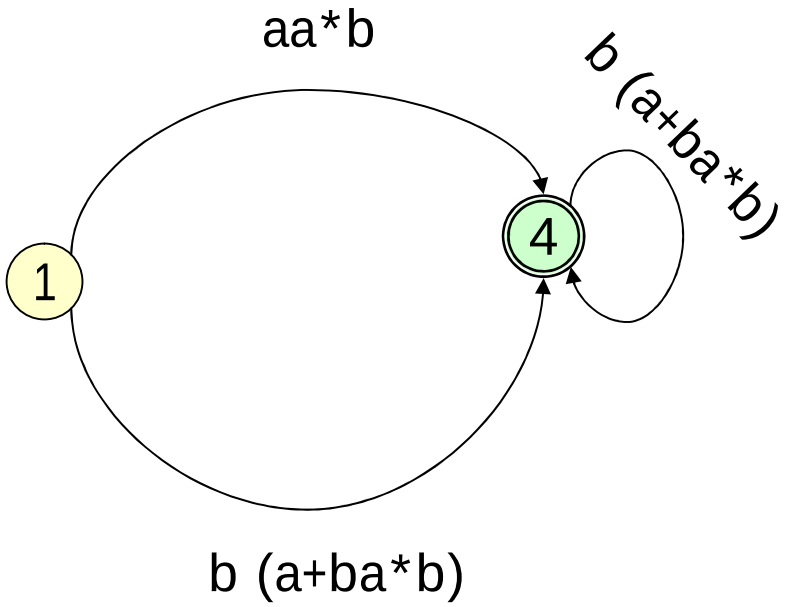
Eliminiere 3:

- Es gibt zwei Läufe über 3:
  - 1 b (a+ba*b) 4 und
  - 4 b (a+ba*b) 4

© H. Peter Gumm, Philipps-Universität Marburg

Beispiel

- Konstruiere regulären Ausdruck für den Automaten



Nur ein Endzustand – o.k.

Eliminiere 2:

- Es gibt zwei Läufe über 2:
  - 1 aa*b 4 und 3 ba*b 4

Eliminiere 3:

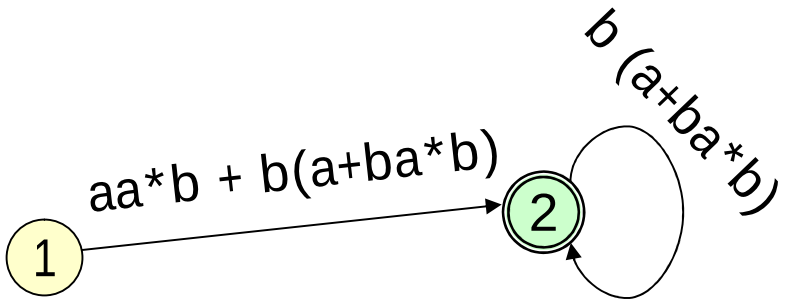
- Es gibt zwei Läufe über 3:
  - 1 b (a+ba*b) 4 und
  - 4 b (a+ba*b) 4

Verschmelze Kanten

© H. Peter Gumm, Philipps-Universität Marburg

Beispiel

- Konstruiere regulären Ausdruck für den Automaten



Ergebnis:

(aa*b + b(a+ba*b))(b(a+ba*b))*

Nur ein Endzustand – o.k.

Eliminiere 2:

- Es gibt zwei Läufe über 2:
  - 1 aa*b 4 und 3 ba*b 4

Eliminiere 3:

- Es gibt zwei Läufe über 3:
  - 1 b (a+ba*b) 4 und
  - 4 b (a+ba*b) 4

Verschmelze Kanten

© H. Peter Gumm, Philipps-Universität Marburg

Äquivalenzsatz für reguläre Sprachen

Äquivalenzsatz für reguläre Sprachen

- Reguläre Ausdrücke,
- deterministische endliche Automaten,
- nichtdeterministische endliche Automaten,
- Automaten mit ε-Transitionen

erkennen/spezifizieren die gleichen Sprachen.

Man sagt, sie sind *äquivalent*.

- Zu jedem regulären Ausdruck *e* gibt es einen ε-NFA A_e, so dass $L(e)=L(A_e)$.
- Zu jedem ε-NFA A gibt es einen DFA PA mit L(A) = L(PA)
- Zu jedem DFA A gibt es einen regulären Ausdruck e_A mit $L(A)=L(e_A)$

© H. Peter Gumm, Philipps-Universität Marburg

Folgerungen

- Sind L, L_1 und L_2 reguläre Sprachen, dann sind auch regulär:
  - $L_1 \cap L_2$ - Produktautomat
  - $\Sigma^* - L$ - Komplementautomat
  - $L_1 - L_2$ - $L_1 \cap (\Sigma^* - L_2)$
- Obwohl $\cap$ bzw. $\Sigma^* - L$ nicht durch reguläre Operatoren ausdrückbar sind
  - in jedem Einzelfall neue Konstruktion erforderlich

© H. Peter Gumm, Philipps-Universität Marburg

Mehr Operationen auf regulären Sprachen

- Ist L regulär dann auch L_a (Ableitung nach a)
  - Hinweis: L=L(r) für regulären Ausdruck r
  - Induktion über Aufbau von r (Differentiationsregeln)
    - $\emptyset_a = \varepsilon_a = \emptyset$
    - $\{a\}_a = \varepsilon$, $\{b\}_a = \emptyset$ falls $a \neq b$
    - $(L+M)_a = L_a + M_a$
    - ... etc.

© H. Peter Gumm, Philipps-Universität Marburg

Entscheidbare Probleme

- Gegeben ein reguläre Sprachen L, L_1, L_2
  - als reguläre Ausdrücke
  - als DFA
  - als NFA
- Aus dem Äquivalenzsatz folgt, dass alle möglichen Fragestellungen entscheidbar sind, z.B.:
  - Ist $L = \emptyset$?
    - Leerheitsproblem
  - Ist $L_1 = L_2$?
    - Äquivalenzproblem
  - Gegeben $w \in \Sigma^*$, ist $w \in L$?
    - Wortproblem

© H. Peter Gumm, Philipps-Universität Marburg

In der Praxis

1. Spezifiziere mit regulärem Ausdruck
   - Einfach zu bestimmen
   - Textuell repräsentierbar
2. Erzeuge äquivalenten ε-NFA
   - OK zur Spezifikation
   - Keine adäquate textuelle Darstellung
   - Bindeglied zwischen RA und DFA
3. Wandle um in DFA
   - Schlecht für Spezifizierung
   - Einfach zu implementieren
4. Minimiere
   - Entferne nicht erreichbare Zustände
   - Verschmelze äquivalente Zustände
5. Implementiere DFA tabellengesteuert
   - Sehr effizient

Dafür gibt es Werkzeuge

z.B.: flex

© H. Peter Gumm, Philipps-Universität Marburg

Professionelles Werkzeug: flex

- flex ist ein Scanner-Generator
  - Aus regulärer Definition r erzeugt flex ein C-Programm, das einen Scanner für r implementiert
- Original UNIX-tool: lex
  - flex ist Open Source –Alternative
  - gleiche Input-Sprache
  - gleiches Verhalten
- Varianten für fast alle andere Sprachen erhältlich
  - Java, Pascal, ML, Haskell, Prolog, …

© H. Peter Gumm, Philipps-Universität Marburg

flex-Inputdatei

- drei Abschnitte, getrennt durch %%
  - Definitionen von token und Buchstabenklassen
  - %%
  - reguläre Ausdrücke mit Aktionen
  - %%
  - optional: main()

Beispieldatei für flex

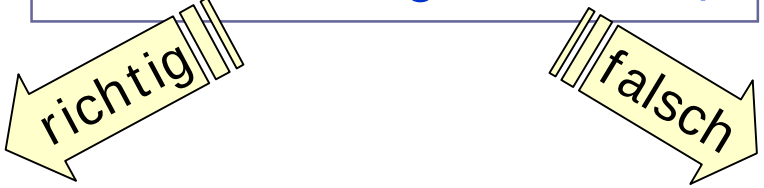
```
letter  [a-zA-Z]
digit   [0-9]
%%
[ \t]+                          ;
"+"                             { return(PLUS)  }
"-"                             { return(MINUS) }
"*"                             { return(TIMES) }
"/"                             { return(QUOT)  }
{digit}+                        { sscanf(yytext,"%d",&yylval);
                                  return(NUM)}
{letter}({letter}|{digit})*     { return(ID)}
%%
```

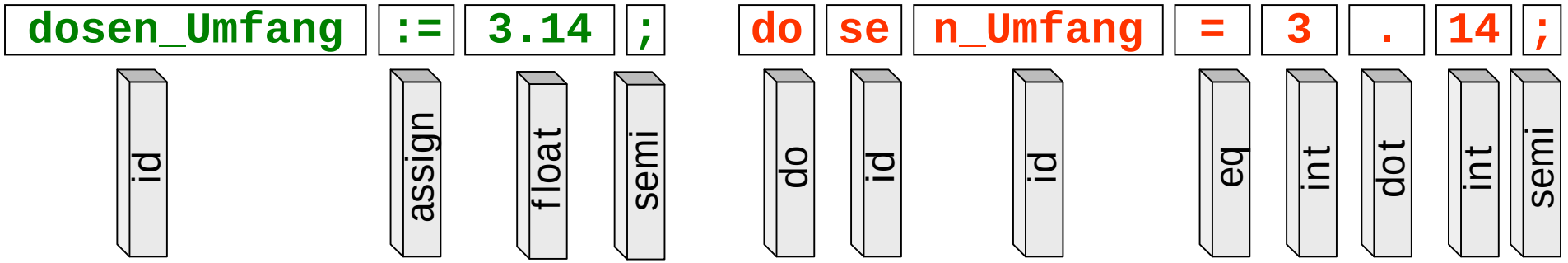
© H. Peter Gumm, Philipps-Universität Marburg

flex-Strategie

- Zu jedem regulären Ausdruck gehört eine Aktion
  - ausgeführt, sobald der Ausdruck erkannt wurde
  - falls Scanner front-end eines Compilers:
    - Tokenübergabe: return(Token);
    - beliebige zusätzliche C-Anweisungen möglich
- Aus Input wird längstmöglicher Teilstring abgeschnitten, der auf eines der Muster passt.

Bsp.: `dosen_Umfang := 3.14;`





© H. Peter Gumm, Philipps-Universität Marburg

flex auch sonst nützlich

- flex ist für viele Zwecke nützliches Werkzeug
  - Beispiel: Schreibe Programm um
    - alle Großbuchstaben in Kleinbuchstaben zu verwandeln
    - Leerzeichen am Zeilenende zu entfernen
    - mehrfache Leerzeichen durch eins zu ersetzen
- Input-Datei „myFile.l“ mit Inhalt

```
# Dies ist die Datei „myFile.lxi“
%%
[A-Z]  putchar(yytext[0]+‘a‘-‘A‘);
[ ]+$  ;
[ ]+   putchar(‘ ‘);
%%
```

Console:

```
> flex myFile.lxi
> cc lex.yy.c –ll
> mv a.out myProg
```

Interne Variable **yytext[]** enthält stets den zuletzt erkannten String

-ll linkt mit der lex-library

© H. Peter Gumm, Philipps-Universität Marburg

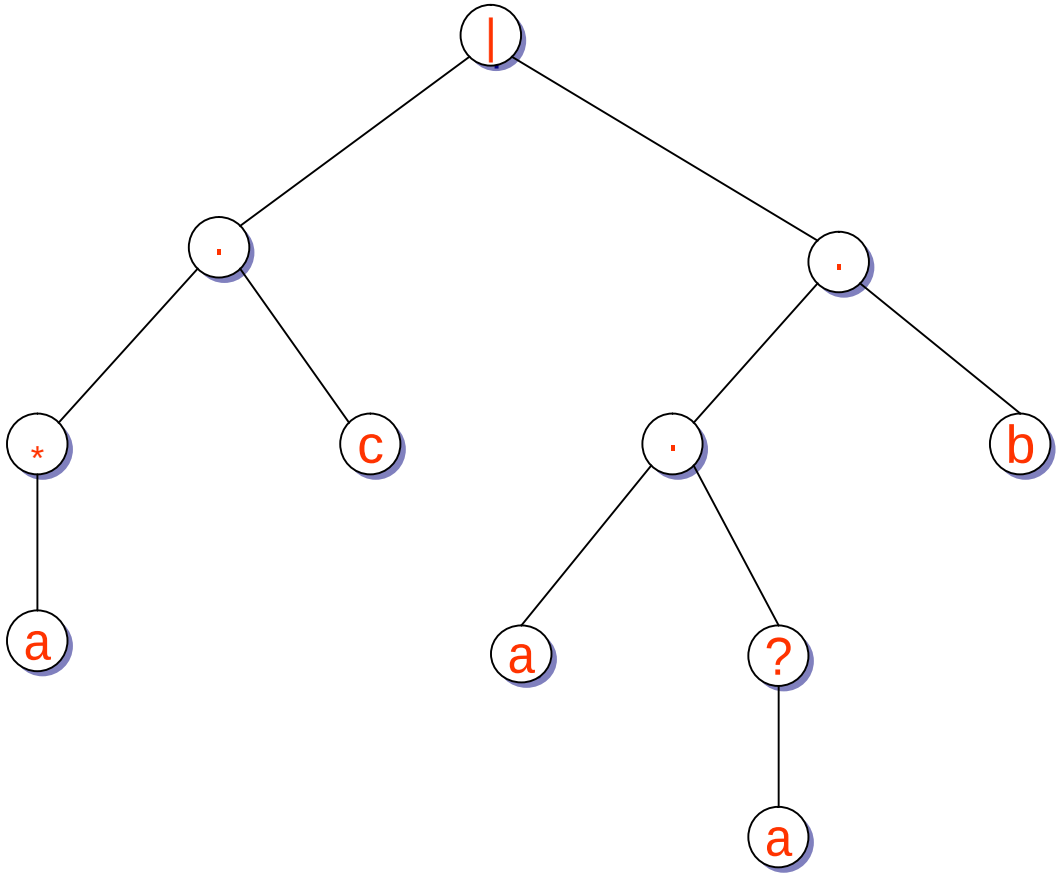
Berry-Sethi-Algorithmus

- Erzeugt NDA für regulären Ausdruck r
  - NDA hat viele nützliche Eigenschaften
    - Geringe Anzahl von Zuständen
      - beschränkt durch Länge von r
    - $q \in \delta(p,e_1) \cap \delta(p,e_2) \Rightarrow e1=e2$
- Syntaxgesteuert
  - Baumtraversierung
    - 2 mal bottom up
    - 1 mal top down

© H. Peter Gumm, Philipps-Universität Marburg

Berry-Sethi - Beispiel: (a*c | aa?b)

1. Starte mit Syntaxbaum



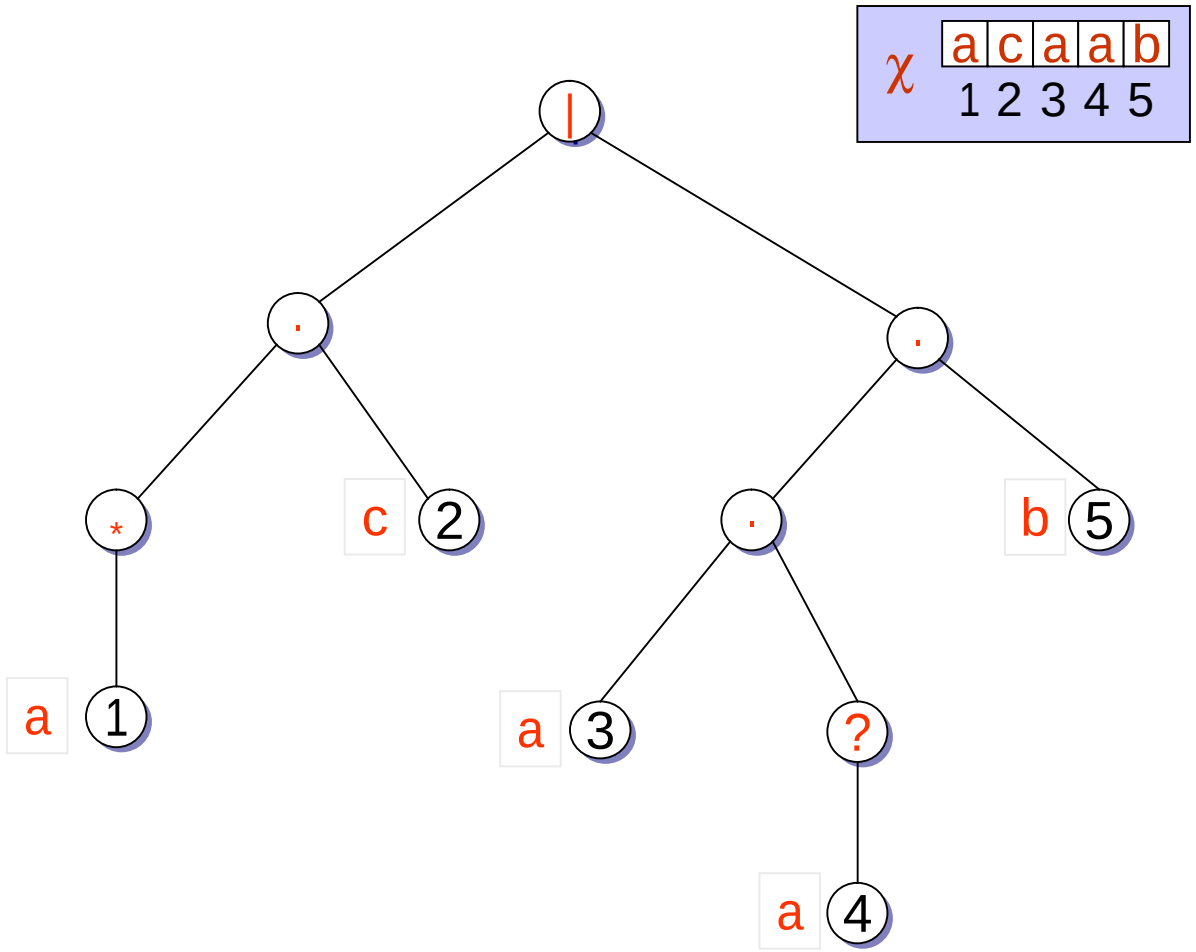
© H. Peter Gumm, Philipps-Universität Marburg

Berry-Sethi - Beispiel: (a*c | aa?b)

1. Starte mit Syntaxbaum

2. nummeriere Blattknoten

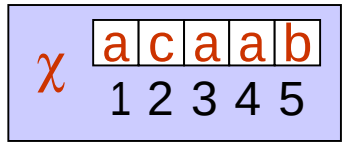
   $\chi(n)$ = Zeichen am Knoten n

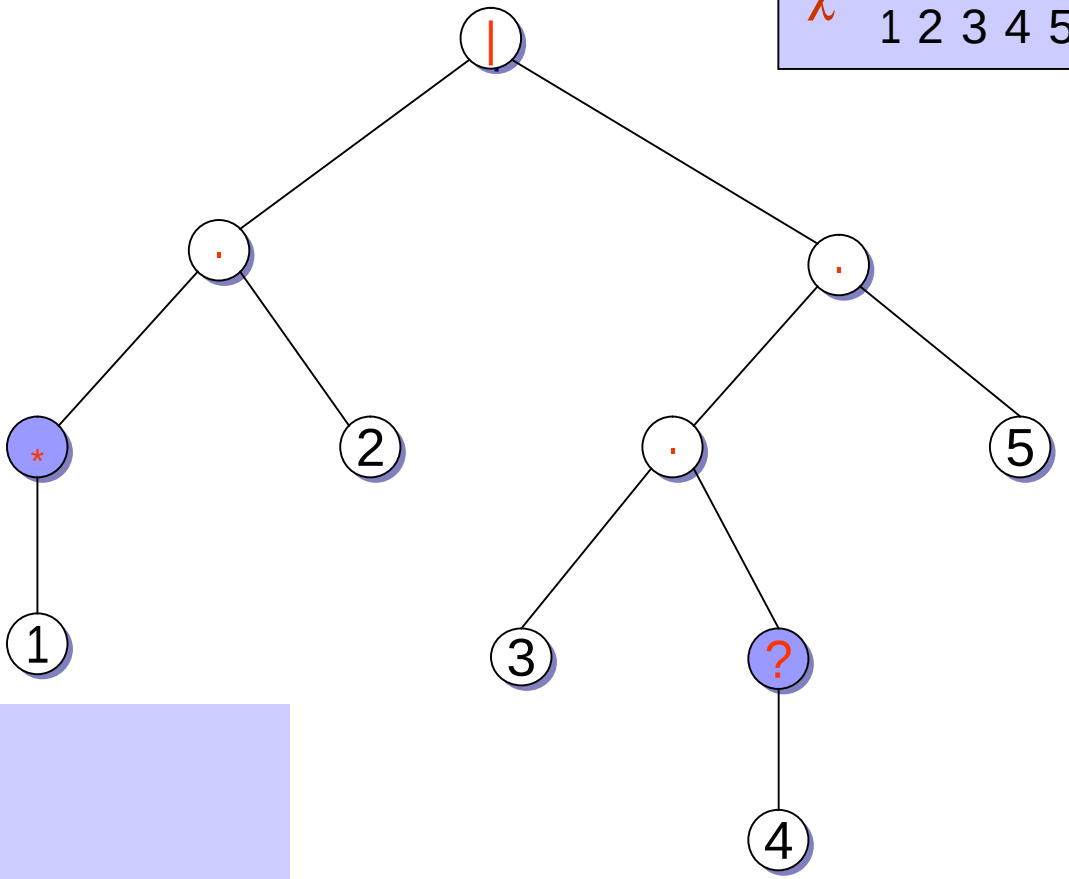


© H. Peter Gumm, Philipps-Universität Marburg

Berry-Sethi - Beispiel: (a*c | aa?b)

1. Starte mit Syntaxbaum
2. nummeriere Blattknoten
   $\chi(n)$ = Zeichen am Knoten n
3. Markiere Optionale Unterbäume
   Bottom-up Baumdurchlauf





Optional(e) = false

Optional(r_1?) = true

Optional(r_1*) = true

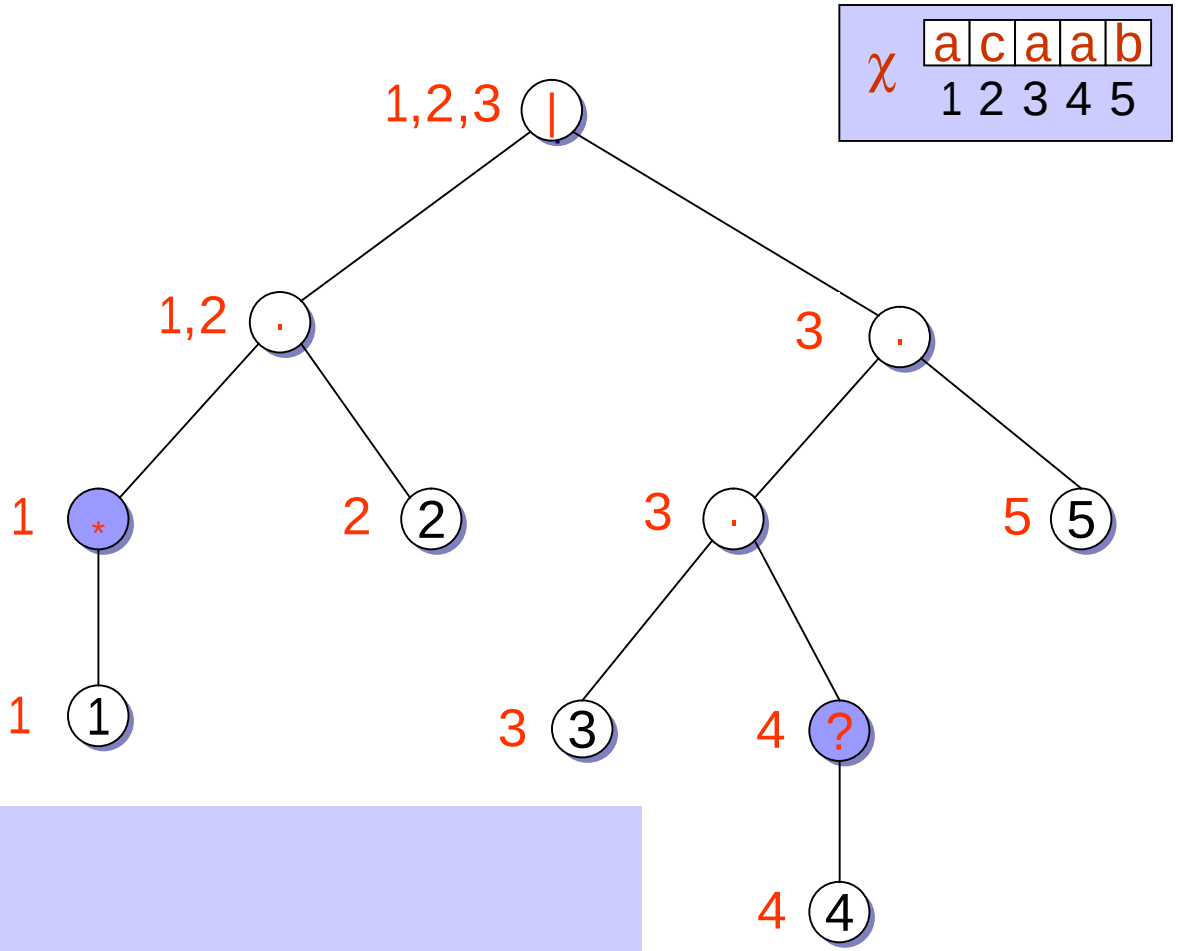
Optional($r_1 \mid r_2$) = Optional(r_1) $\vee$ Optional(r_2)

Optional($r_1 \cdot r_2$) = Optional(r_1) $\wedge$ Optional(r_2)

© H. Peter Gumm, Philipps-Universität Marburg

Berry-Sethi - Beispiel: (a*c | aa?b)

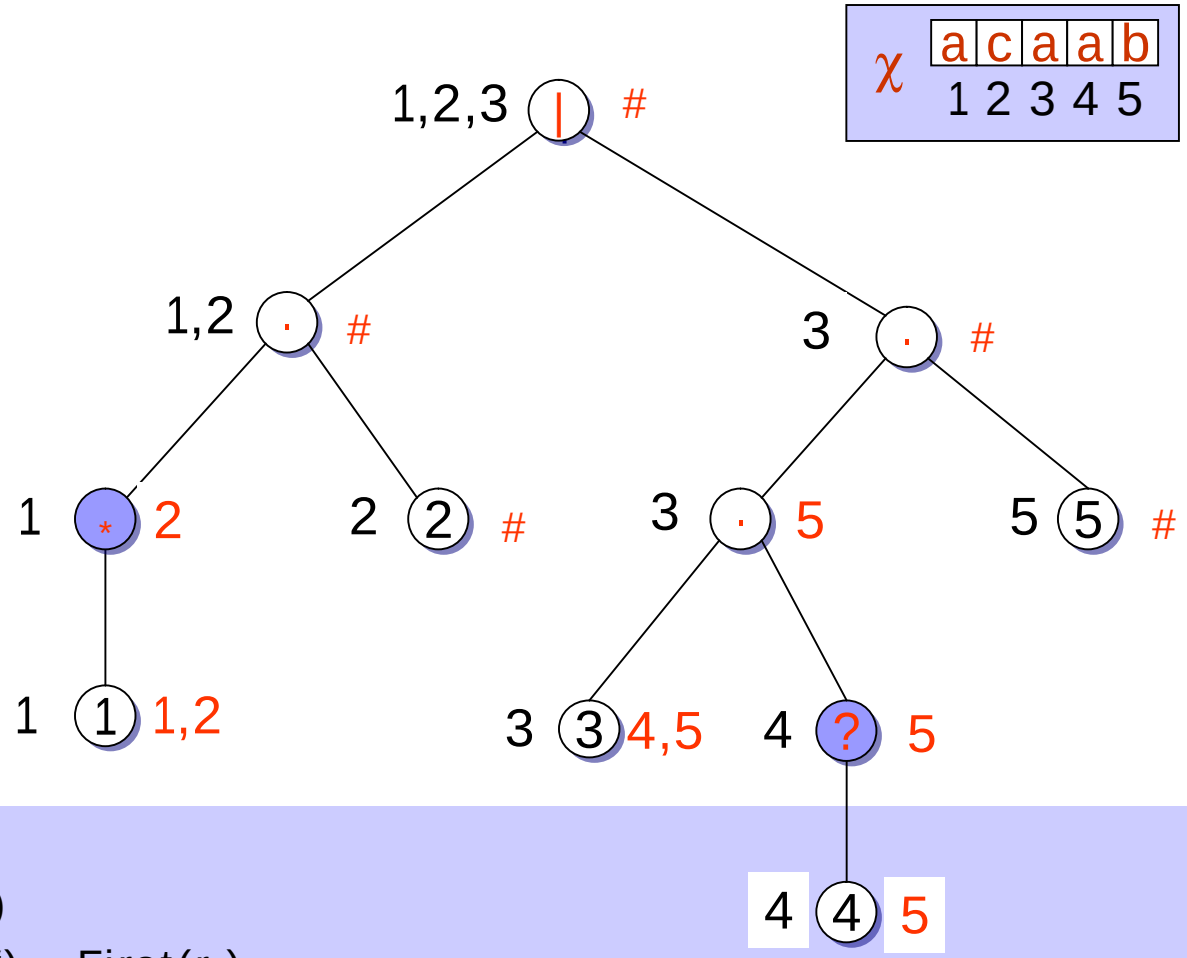
1. Starte mit Syntaxbaum
2. nummeriere Blattknoten
3. Markiere Optionale Unterbäume
4. Berechne First(k) für jeden Knoten k



First(n) = { n }
First(r_1?) = First(r_1)
First(r_1*) = First(r_1)
First(r_1 | r_2) = First(r_1) ∪ First(r_2)
First(r_1 · r_2) = First(r_1) ∪ (if Optional(r_1) then First(r_2) else { })

Berry-Sethi - Beispiel: (a*c | aa?b)

1. Starte mit Syntaxbaum
2. nummeriere Blattknoten
3. Markiere Optionale Unterbäume
4. Berechne First(k) für jeden Knoten k
5. Berechne, was auf einen Teilstring r‘ folgen kann.
   - top down
   - # = endOfString-Marke



Follow(r) = { # }

r‘ = r_1? : Follow(r_1) = Follow(r‘)

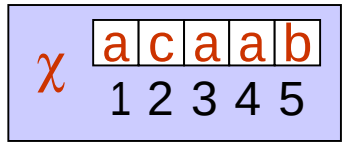
r‘ = r_1* : Follow(r_1) = Follow(r‘) ∪ First(r_1)

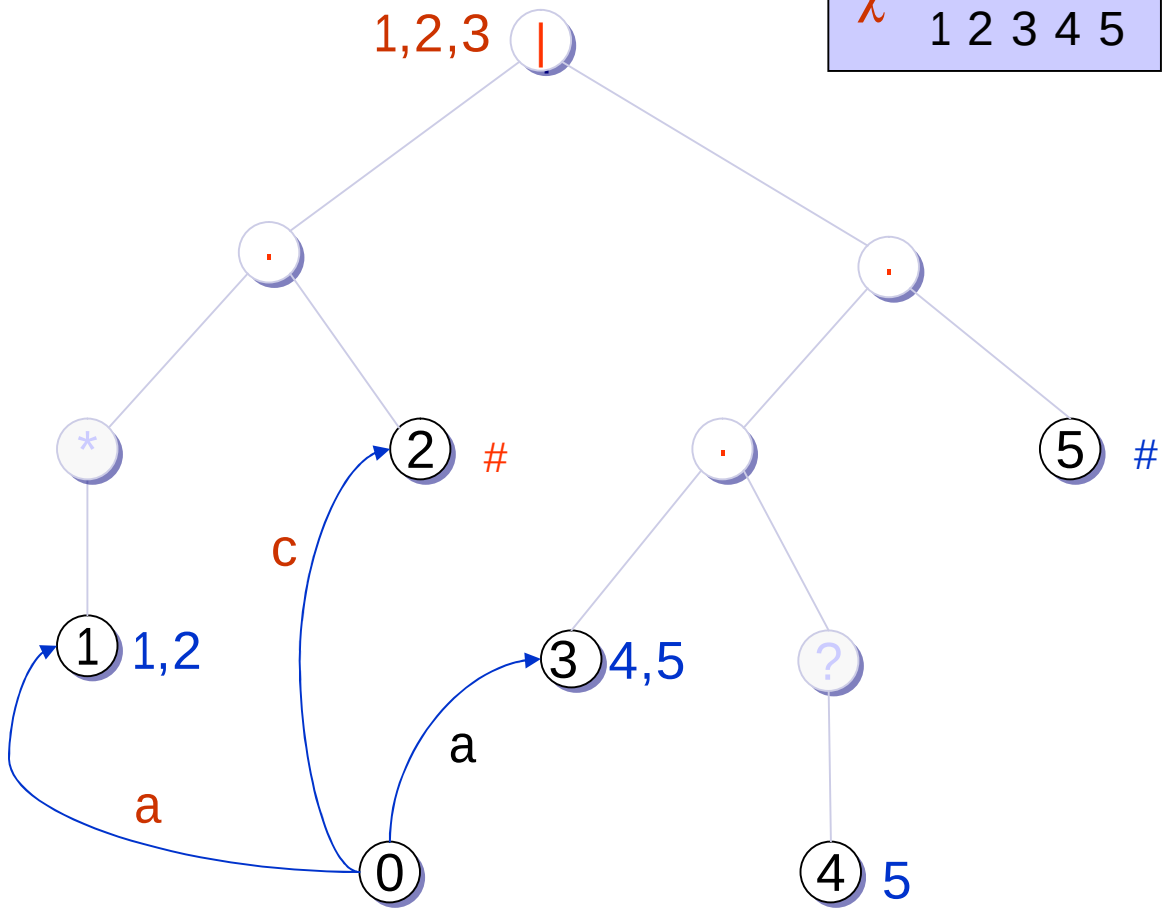
r’ = r_1|r_2 : Follow(r_1) = Follow(r_2) = Follow(r’)

r’ = $r_1 \cdot r_2$: Follow(r_2) = Follow(r’) und Follow(r_1) = First(r_2) ∪ (if Optional(r_2) then Follow(r’))

Berry-Sethi - Beispiel: (a*c | aa?b)

1. Starte mit Syntaxbaum
2. nummeriere Blattknoten
3. Markiere Optionale Unterbäume
4. Berechne First(k) für jeden Knoten k
5. Berechne, was auf einen Teilstring r‘ folgen kann.
   - top down
   - # = endOfString-Marke
6. Addiere Startknoten 0



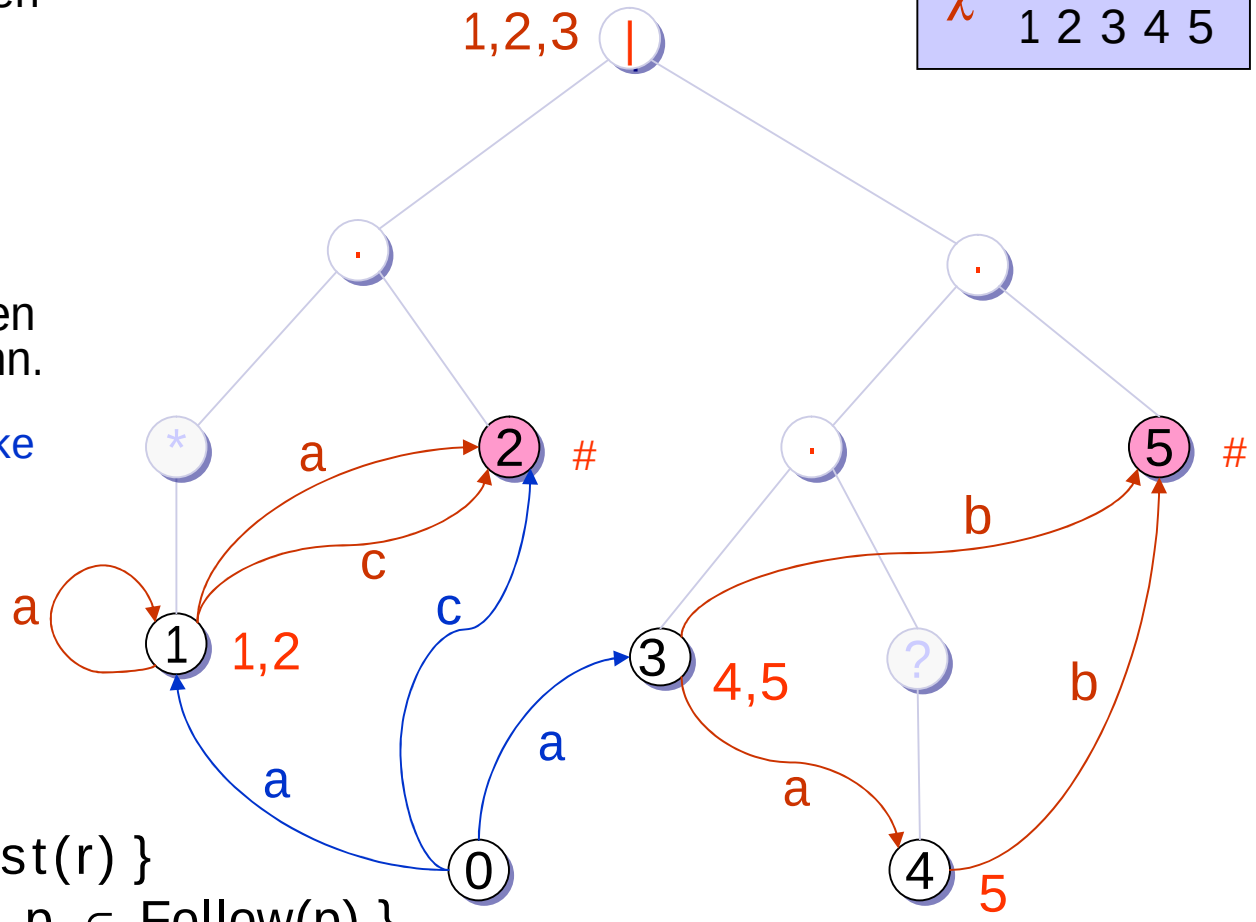


$$\delta = \{ (0,\chi(p),p) \mid p \in First(r) \} \cup \{(p,\chi(p_1),p_1) \mid p \in P,\ p_1 \in Follow(p) \}$$

© H. Peter Gumm, Philipps-Universität Marburg

Berry-Sethi - Beispiel: (a*c | aa?b)

1. Starte mit Syntaxbaum
2. nummeriere Blattknoten
3. Markiere Optionale Unterbäume
4. Berechne First(k) für jeden Knoten k
5. Berechne, was auf einen Teilstring r‘ folgen kann.
   - top down
   - # = endOfString-Marke
6. Addiere Startknoten 0



$\delta = \{ (0,\chi(p),p) \mid p \in First(r) \}$
$\cup \{ (p,\chi(p_1),p_1) \mid p \in P, p_1 \in Follow(p) \}$

$F = \{ p \mid \# \in Follow(p) \}$

© H. Peter Gumm, Philipps-Universität Marburg