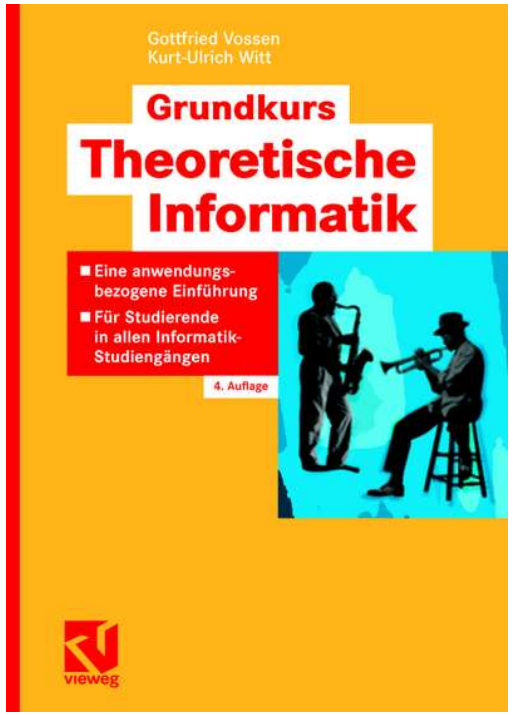
Automaten und Formale Sprachen

Endliche Automaten und Reguläre sprachen

Ralf Möller
Hamburg Univ. of Technology

Literatur

- **Gottfried Vossen, Kurt-Ulrich Witt:** Grundkurs Theoretische Informatik, Vieweg Verlag



Danksagung

- Kurs basiert auf Präsentationsmaterial von
  - G. Vossen (Uni Münster),
    K.-U. Witt (Hochschule Bonn-Rhein-Sieg)
  - Johannes Köbler (HU Berlin)
  - Thomas Ottmann (Uni Freiburg)
  - Lenore Blum (CMU)

Wiederholung: Alphabete

- Automaten verarbeiten Zeichenfolgen, die aus atomaren Symbolen bestehen.
- Menge der zugelassenen Zeichen: Endliches Alphabet Σ.

Beispiele:

- $\Sigma = \{\underline{50}, \underline{100}, \underline{200}\}$ $\quad |\Sigma| = 3$
- $\Sigma = \{a_1, a_2, a_3, \ldots, a_n\}$ $\quad |\Sigma| = n$
- $\Sigma = \{a, b, \ldots, z\}$ $\quad |\Sigma| = 26$
- $\Sigma = \emptyset$ $\quad |\Sigma| = 0$

Widerholung: Deterministische endliche Automaten

Ein deterministischer endlicher Automat (DFA) ist gegeben durch

- eine endliche menge S von Zuständen
- eine endliche Menge Σ von Eingabezeichen
- einen Anfangszustand $s_0 \in S$
- eine Endzustandsmenge $F \subseteq S$
- eine Übergangsfunktion $\delta : S \times \Sigma \rightarrow S$

Kurz: $A = (\Sigma, S, \delta, s_0, F)$

δ kann auch durch einen Zustandsübergangs Graphen oder als Menge von Tripeln (s, a, t) mit δ (s, a) = t gegeben sein

δ ist manchmal nicht total (überall definiert)

Wiederholung: Erweiterte Übergangsfunktion

Die Zustandsübergangsfunktion δ kann von Zeichen auf Wörter erweitert werden:

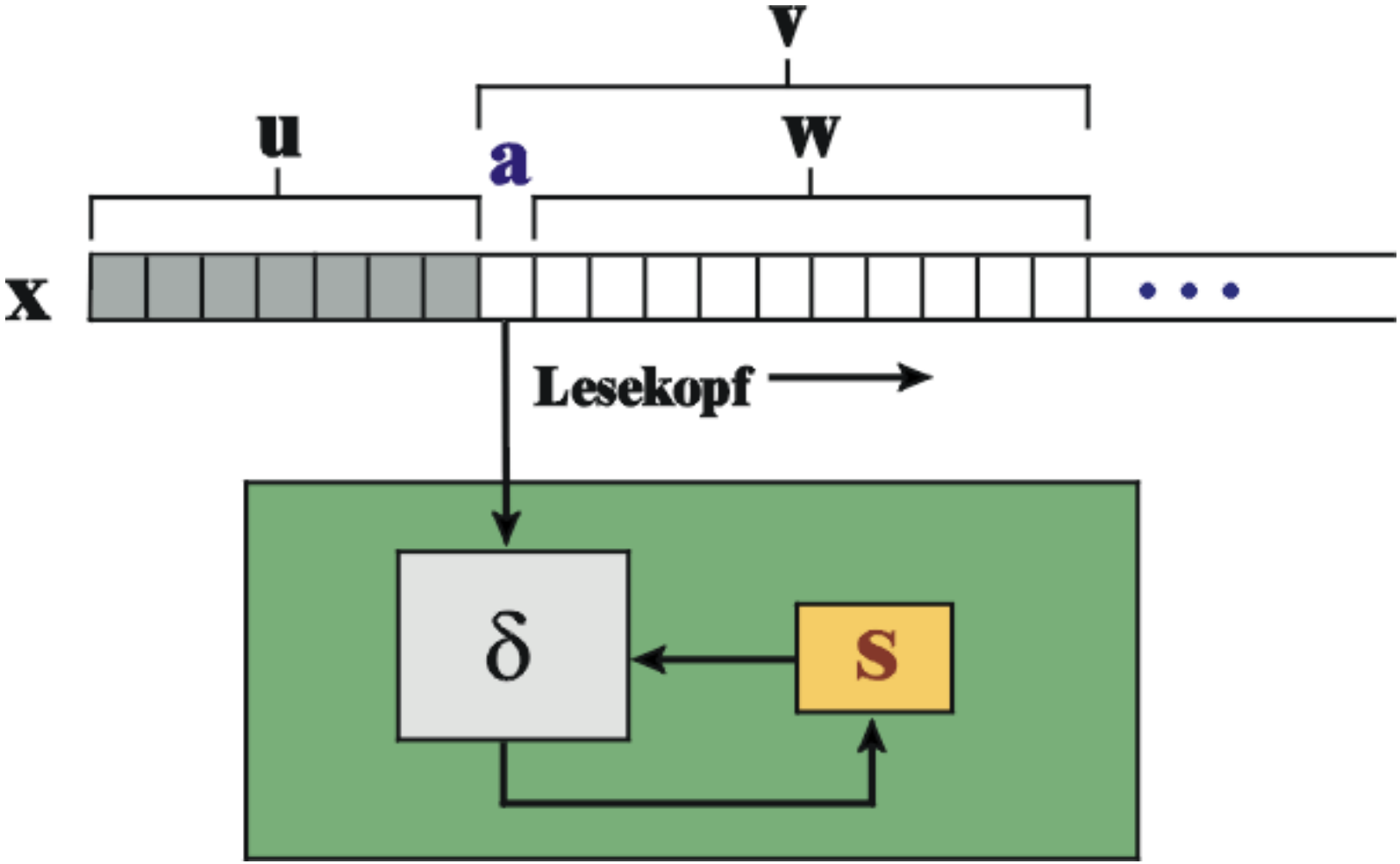
$$\delta^* : S \times \Sigma^* \rightarrow S \text{ definiert durch}$$

- $\delta^*(s, \varepsilon) = s$ für alle $s \in S$
- $\delta^*(s, aw) = \delta^*(\delta(s, a), w)$ für alle $a \in \Sigma, w \in \Sigma^*$

Für einen endlichen Automaten $A = (\Sigma, S, \delta, s_0, F)$ wird die von A akzeptierte Sprache (die Menge aller von A akzeptierten Eingabefolgen) $L(A) \subseteq \Sigma^*$ definiert durch:

$$L(A) = \{w; \delta^*(s_0, w) \in F\}$$

Wiederholung: Konfiguration eines endlichen Automaten



Widerholung: Konfigurationsübergänge

Ein Konfigurationsübergang $(s, v) \vdash (t, w)$ kann stattfinden, wenn $v = aw$ und $\delta(s, a) = t$ ist.

Die Abarbeitung eines Wortes $x = x_1x_2 \ldots x_r$ durch einen DFA kann als Folge von Konfigurationsübergängen beschrieben werden:

$$(s_0, x_1x_2 \ldots x_r) \vdash (s_1, x_2 \ldots x_r) \vdash \ldots \vdash (s_r, \varepsilon)$$

Mit $\vdash^*$ wird die transitiv-reflexive Hülle von $\vdash$ beschrieben.

Wiederholung: Reguläre Sprachen

- Für einen DFA $A = (\Sigma, S, \delta, s_0, F)$ ist

  $$L(A) = \{w \in \Sigma^* \; ; (s_0, w) \vdash^* (s, \varepsilon), s \in F\}$$

  die von A akzeptierte Sprache.
- Eine Sprache $L \subseteq \Sigma^*$ heißt regulär, wenn es einen DFA A gibt mit $L = L(A)$.
- Zwei DFA A und A‘ heißen äquivalent, falls sie die gleiche Sprache akzeptieren, wenn also gilt: $L(A) = L(A')$.

Theorie endlicher Automaten

Gibt Antworten auf folgende Fragen:

1. Wie entwirft man endliche Automaten für bestimmte Aufgaben (Synthese-Aufgabe)?
2. Wie analysiert man endliche Automaten? D.h. kann man die von endlichen Automaten akzeptierbaren Sprachen auch anders (automatenfrei) beschreiben?
3. Wie vereinfacht (reduziert, minimiert) man endliche Automaten? D.h. wie eliminiert man evtl. überflüssige Zustände?

Die Synthese endlicher Automaten ist ein kreativer Prozess!

Zur Analyse verwendet man: Reguläre Ausdrücke, Grammatiken, algebraische Hilfsmittel.

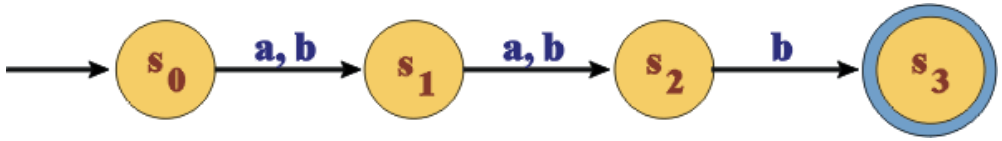
Die Reduzierung erfolgt durch Bildung von Äquivalenzklassen.

Beispiel einer Syntheseaufgabe

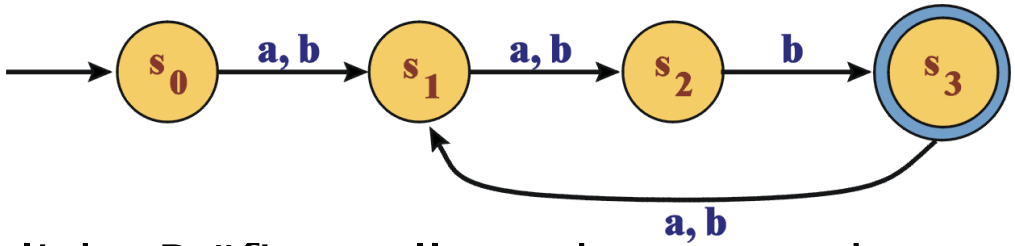
Finde einen DFA für

$L_{3b} = \{w \in \{a, b\}^* ; w = w_1 \ldots w_n, w_i \in \{a, b\}, w_{3i} \neq a, 1 \leq 3i \leq n, n \geq 0\}$

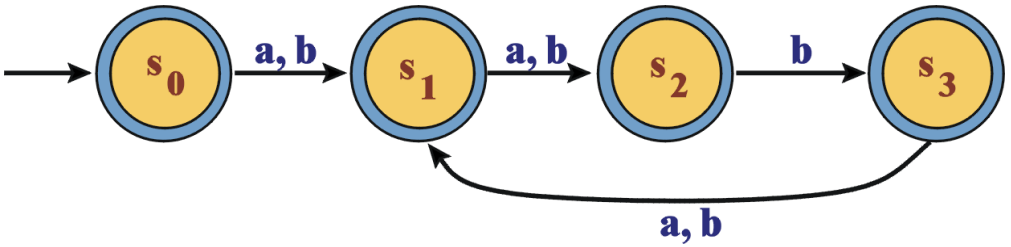
- Dreiergruppe von Zeichen, deren letztes ein b ist, muss erkannt werden:



- Beliebig viele Dreiergruppen derselben Art sind als Präfixe erlaubt:



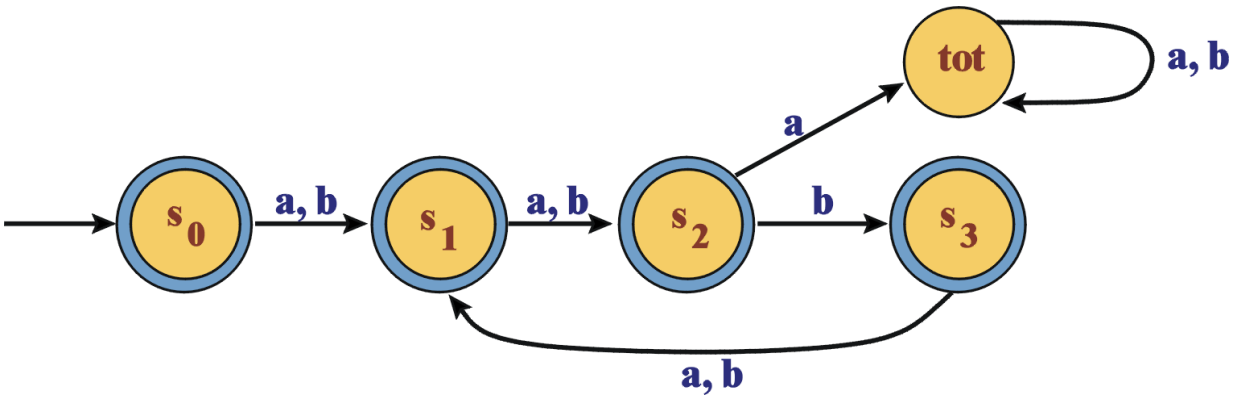
- Auch sämtliche Präfixe sollen erkannt werden:



Vollständige Automaten

- Bisher musste die Funktion δ eines DFA nicht total sein.
- Ein DFA $A = (\Sigma, S, \delta, s_0, F)$ heißt vollständig, wenn $dom(\delta) = S \times \Sigma$
- Jeder DFA $A = (\Sigma, S, \delta, s_0, F)$ kann durch Hinzunahme eines Zustands *tot* vervollständigt werden:
- Wenn δ(s, a) nicht definiert ist, ergänze δ(s, a) = *tot*

Beispiel



Anwendung von DFA zur Suche in Texten

Verschiedene Szenarien:

Dynamische Texte

- Texteditoren
- Symbolmanipulatoren
- Statische Texte

Literaturdatenbanken

- Bibliothekssysteme
- Gen-Datenbanken
- WWW-Verzeichnisse

Problemdefinition: Mustersuche in Texten

Gegeben:

Text $t_1 t_2 t_3 \ldots t_n \quad \in \Sigma^n$

Muster $p_1 p_2 \ldots p_m \quad \in \Sigma^m$

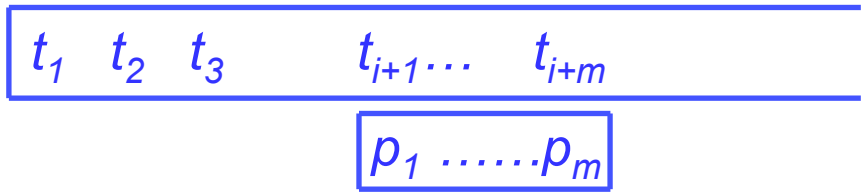
Gesucht: Ein oder alle Vorkommen des Musters im Text, d.h. Verschiebungen i mit $0 \leq i \leq n-m$ und

$p_1 = t_{i+1}$

$p_2 = t_{i+2}$

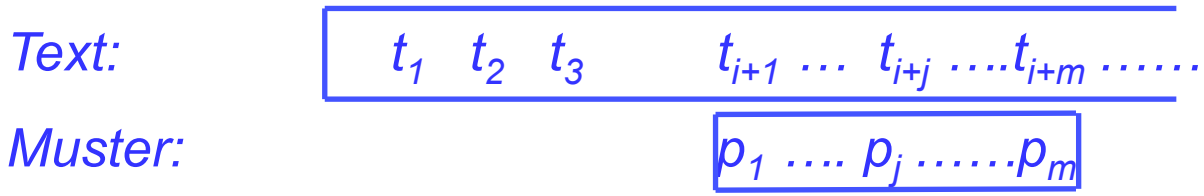
……..

$p_m = t_{i+m}$



Naives Verfahren

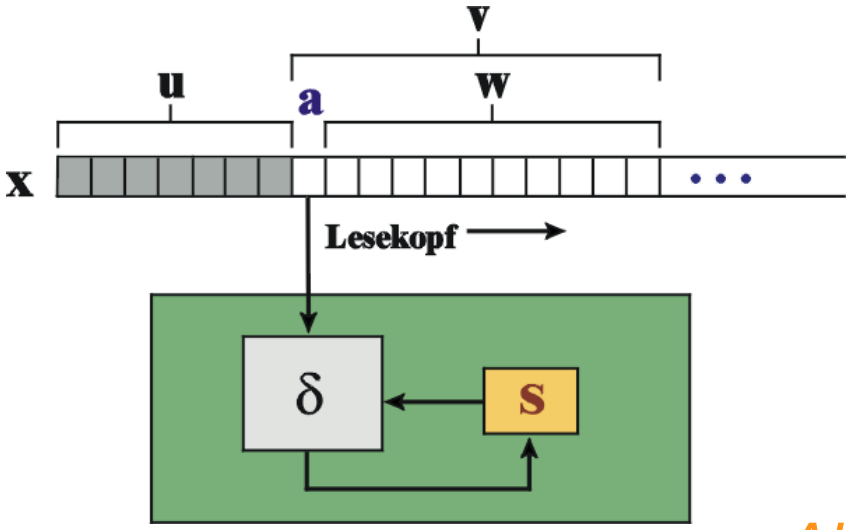
Für jede mögliche Verschiebung i mit $0 \leq i \leq n-m$
prüfe maximal m Zeichenpaare. Bei Mismatch beginne mit neuer Verschiebung!



Aufwandsabschätzung:
Was ist die Laufzeit im schlimmsten Fall?

Automatenbasiertes Verfahren

Konstruiere zum Muster $P = p_1 p_2 \ldots p_m$ einen DFA A_P,
der den Text $T = t_1 t_2 t_3 \ldots t_n$ einmal v.l.n.r. liest und in einem Endzustand ist, immer dann, wenn er das Ende eines Vorkommens von P in T erkannt hat.



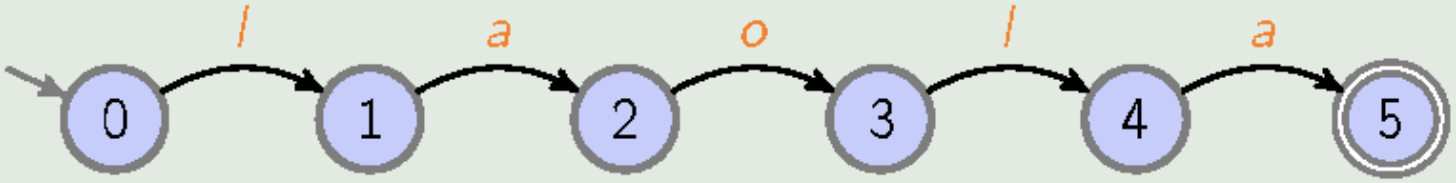
Beobachtung: Das Lesen und Verarbeiten eines Zeichens Verursacht nur konstanten Aufwand!

Alfred V. Aho und Margaret J. Corasick 1975

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

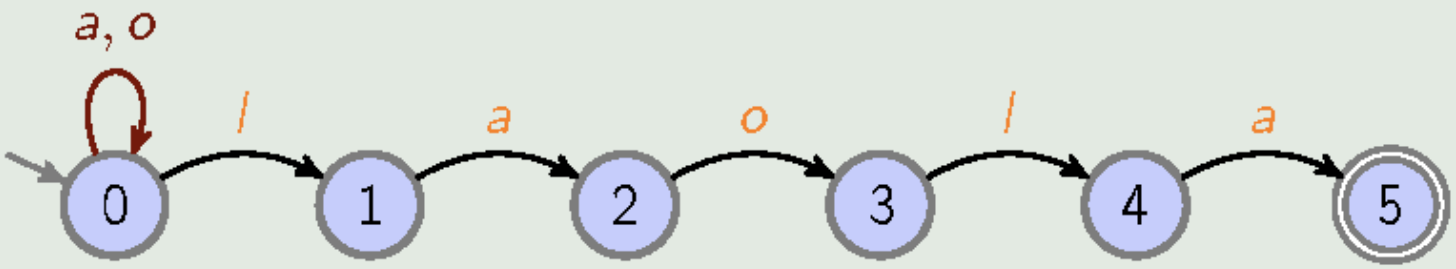


δ	0	1	2	3	4	5
a		2			5	
l	1			4		
o			3			

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

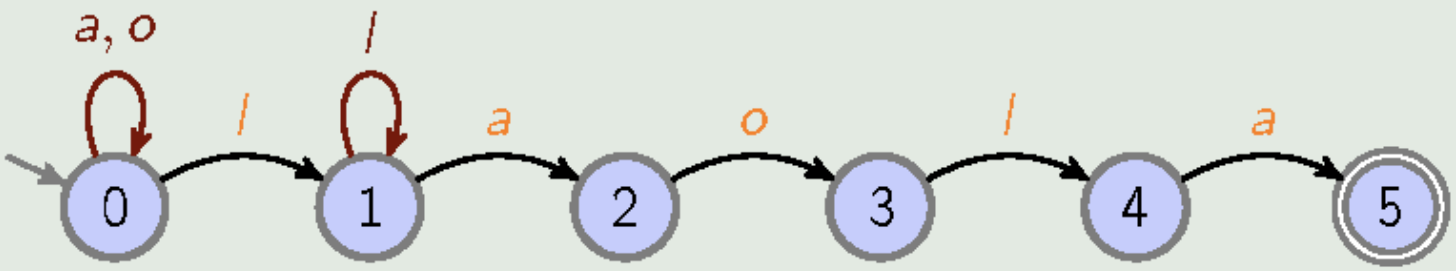


δ	0	1	2	3	4	5
a	0	2			5	
l	1			4		
o	0		3			

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

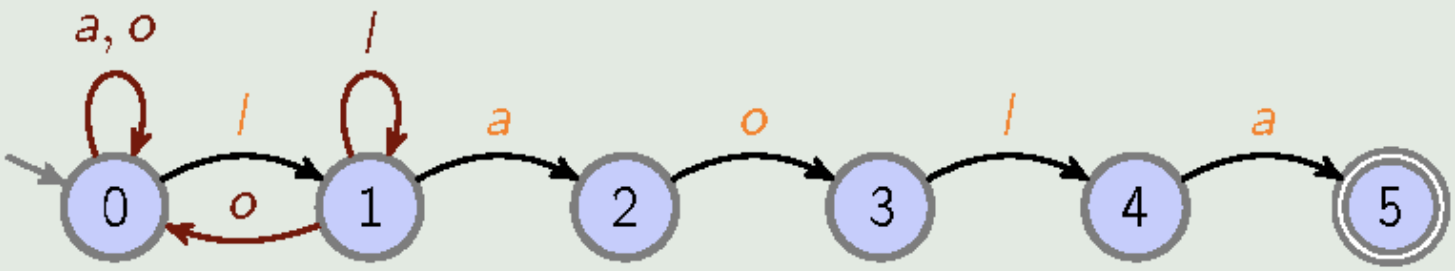


δ	0	1	2	3	4	5
a	0	2			5	
l	1	1		4		
o	0		3			

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

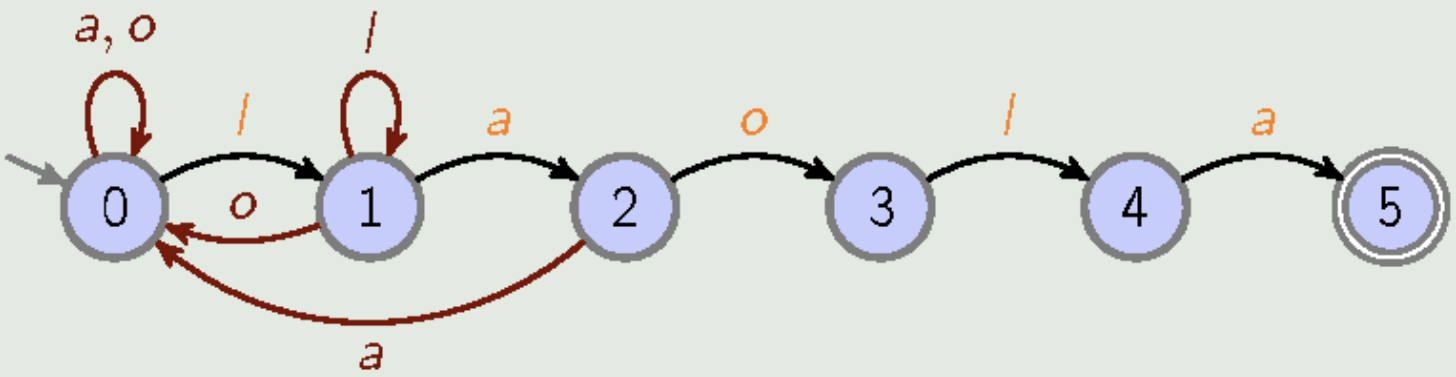


δ	0	1	2	3	4	5
a	0	2			5	
l	1	1		4		
o	0	0	3			

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

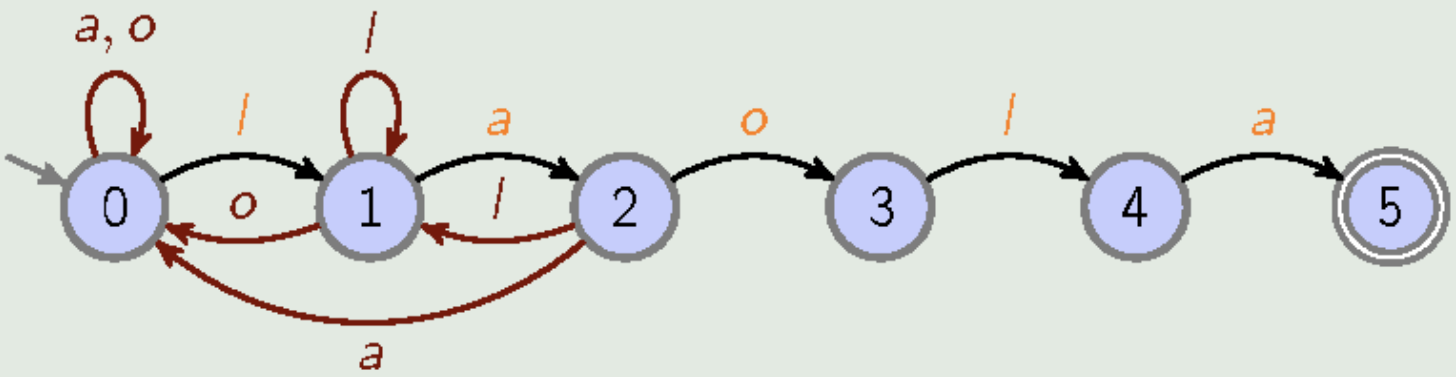


δ	0	1	2	3	4	5
a	0	2	0		5	
l	1	1		4		
o	0	0	3			

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

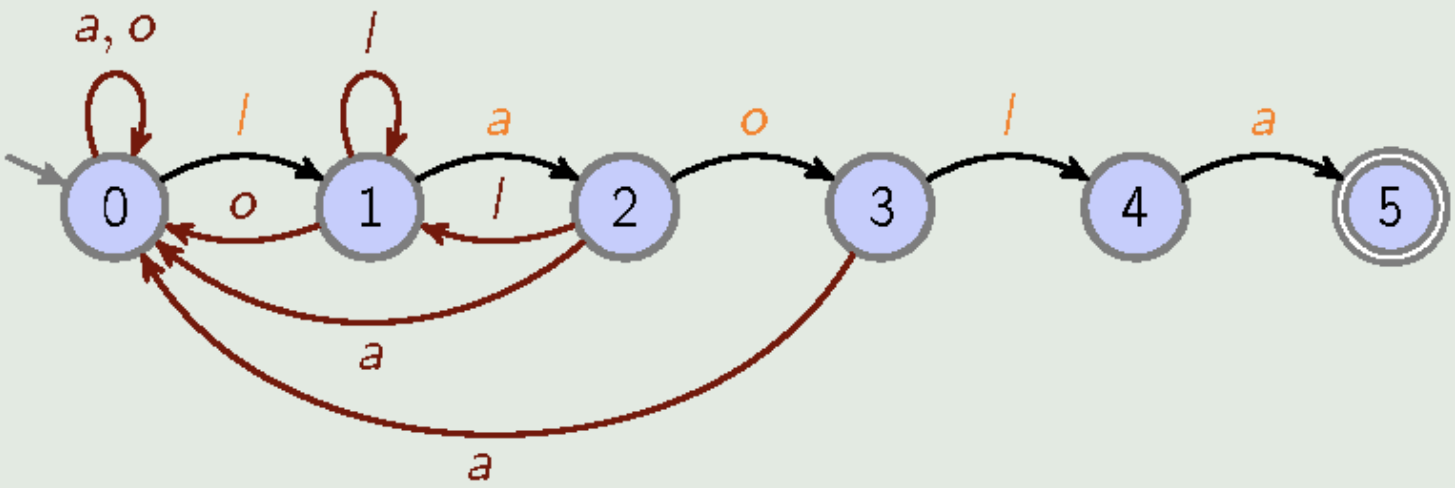


δ	0	1	2	3	4	5
a	0	2	0		5	
l	1	1	1	4		
o	0	0	3			

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

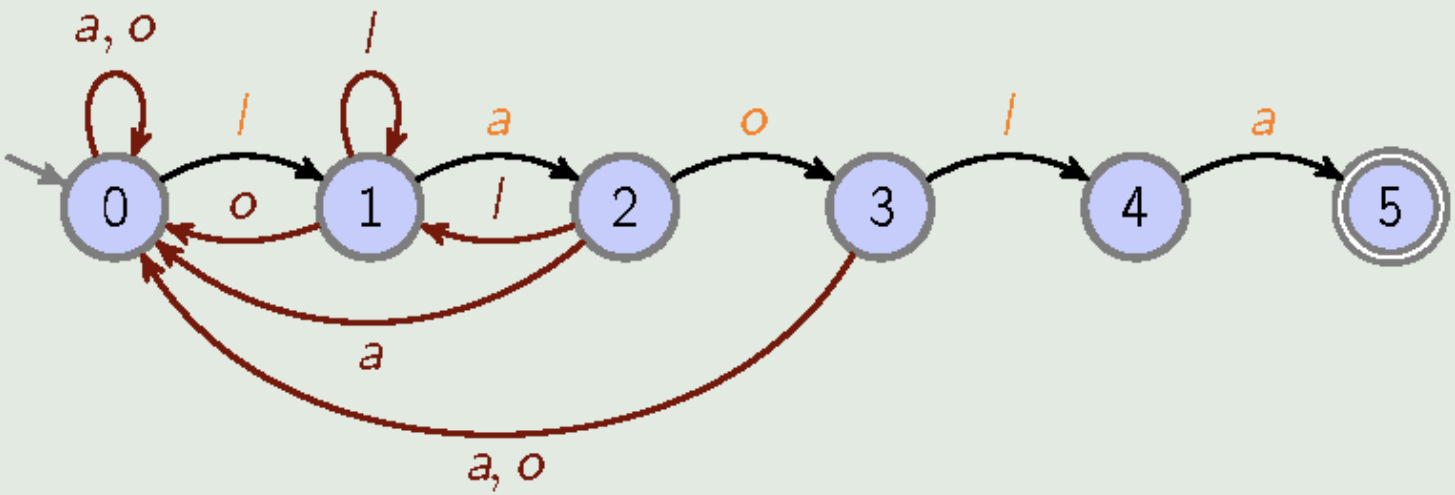


δ	0	1	2	3	4	5
a	0	2	0	0	5	
l	1	1	1	4		
o	0	0	3			

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

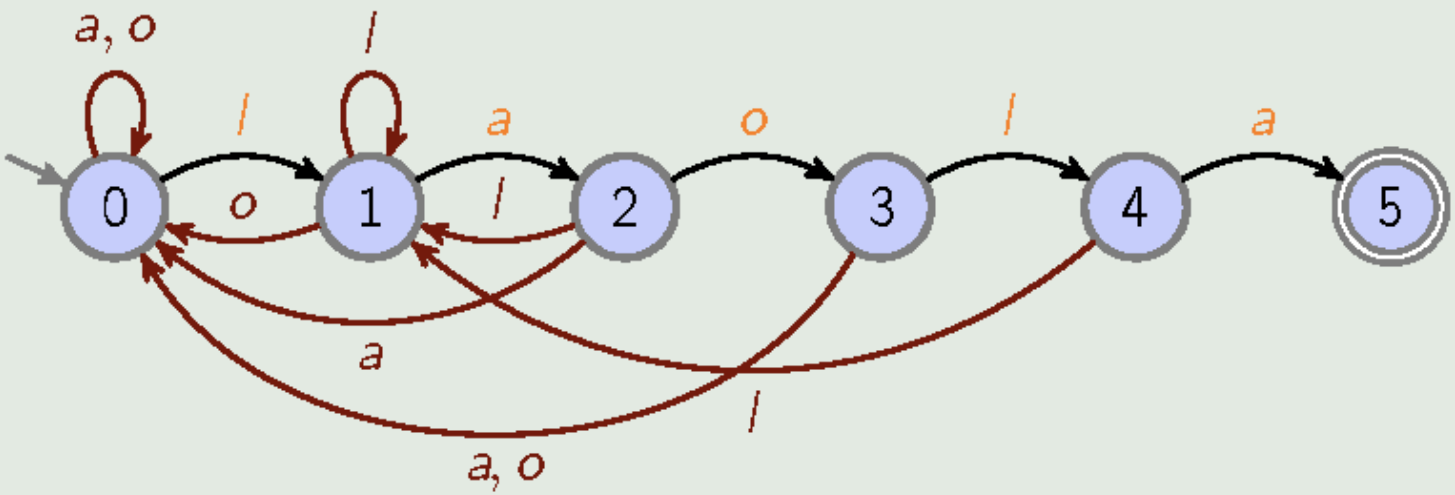


δ	0	1	2	3	4	5
a	0	2	0	0	5	
l	1	1	1	4		
o	0	0	3	0		

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

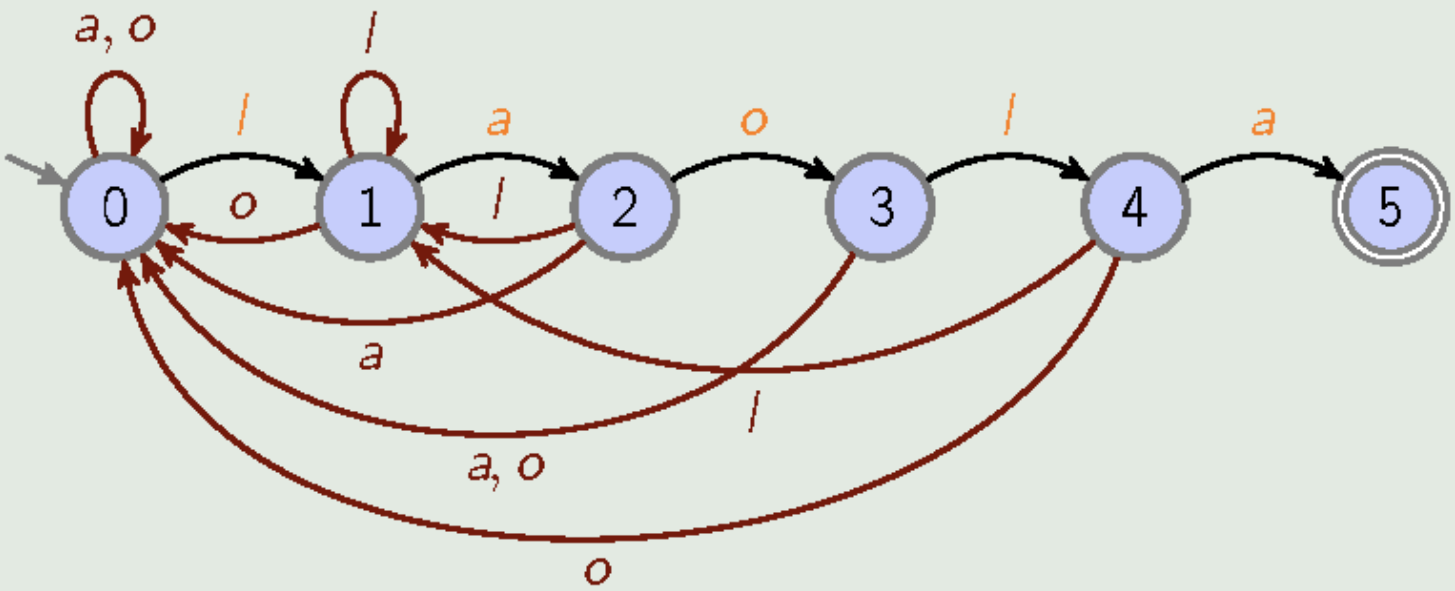


δ	0	1	2	3	4	5
a	0	2	0	0	5	
l	1	1	1	4	1	
o	0	0	3	0		

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

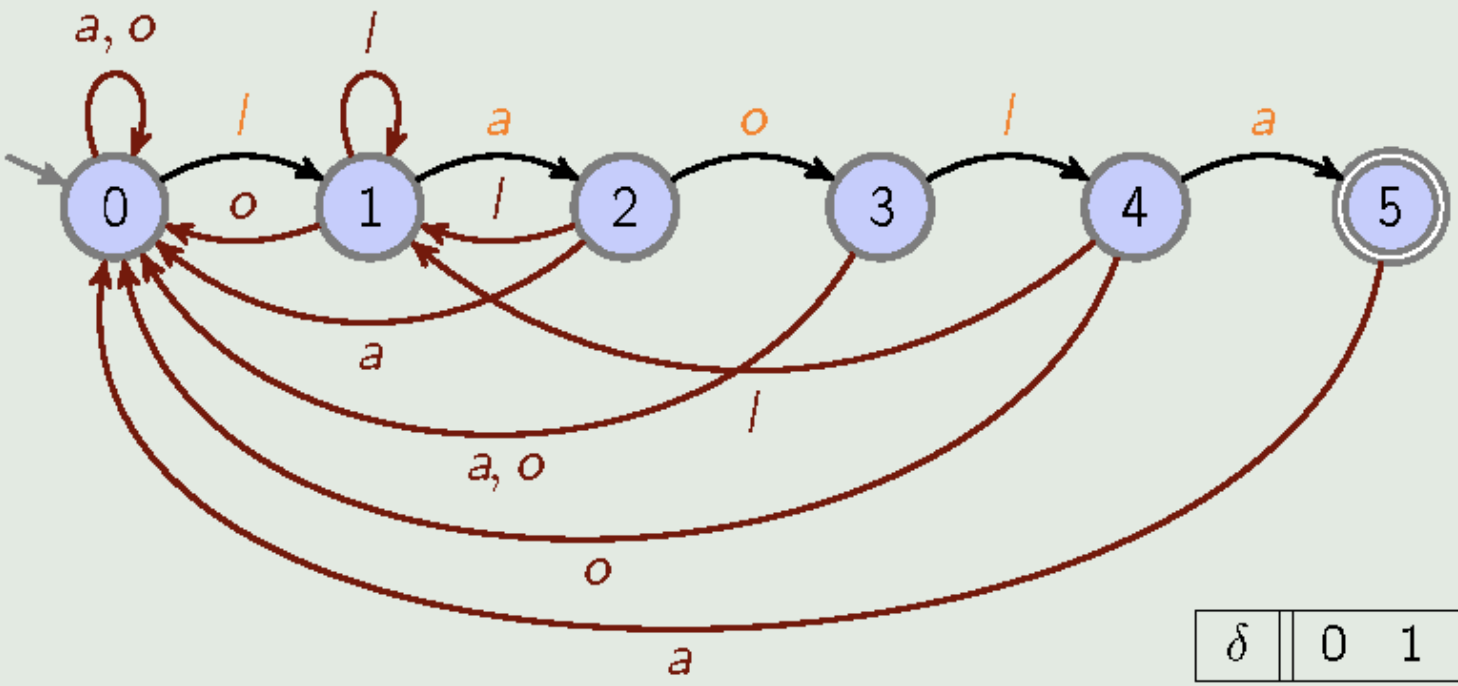


δ	0	1	2	3	4	5
a	0	2	0	0	5	
l	1	1	1	4	1	
o	0	0	3	0	0	

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

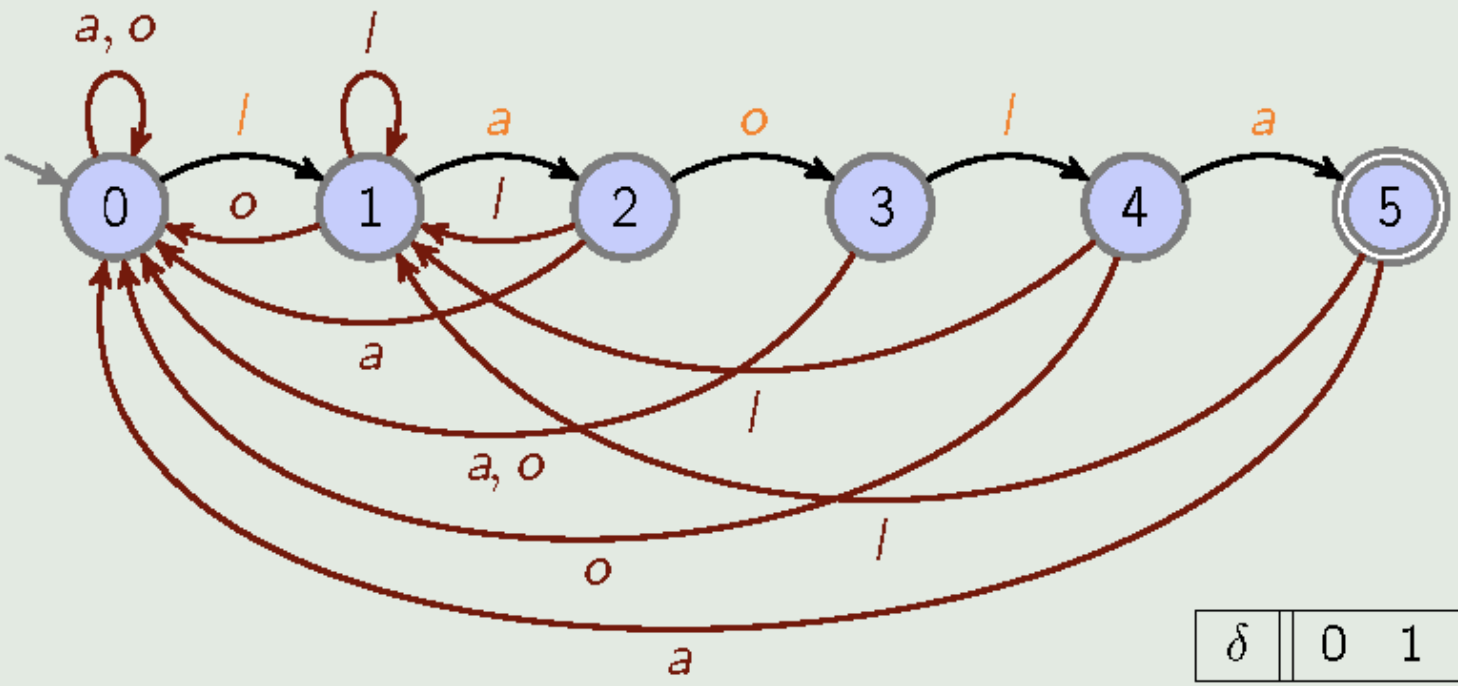


δ	0	1	2	3	4	5
a	0	2	0	0	5	0
l	1	1	1	4	1	
o	0	0	3	0	0	

String-Matching mit endlichen Automaten

Beispiel

Für das Muster $y = laola$ ergibt sich folgender DFA M_y:

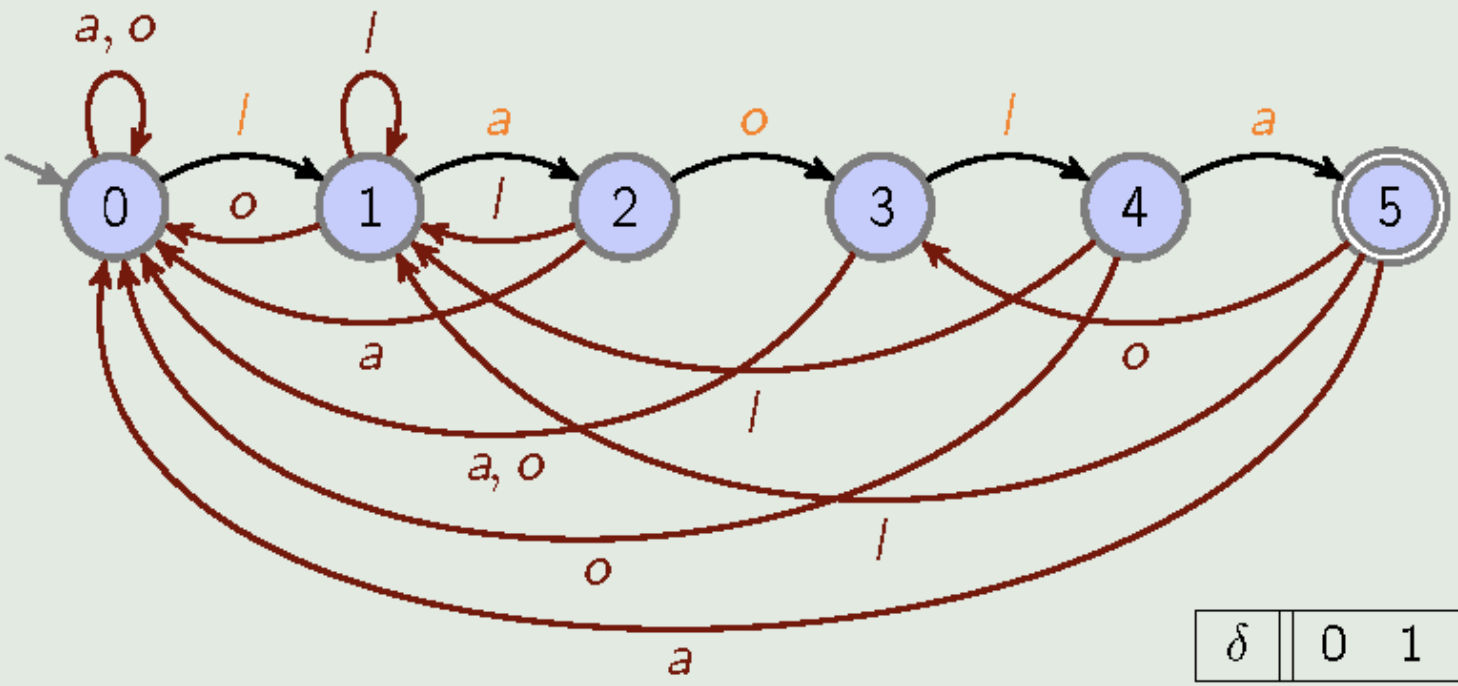


δ	0	1	2	3	4	5
a	0	2	0	0	5	0
l	1	1	1	4	1	1
o	0	0	3	0	0	

String-Matching mit endlichen Automaten

Beispiel

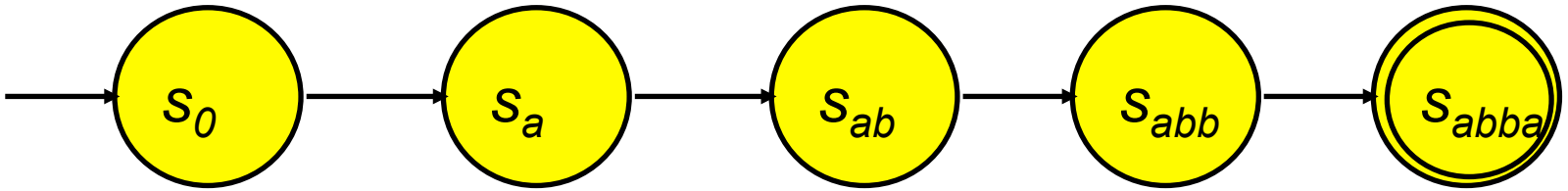
Für das Muster $y = laola$ ergibt sich folgender DFA M_y:



δ	0	1	2	3	4	5
a	0	2	0	0	5	0
l	1	1	1	4	1	1
o	0	0	3	0	0	3

Übung

Sei P = abba, ein DFA, der jedes Vorkommen von P in einem beliebigen Text aus $\Sigma^* = \{a, b\}^*$ entdeckt, kann so konstruiert werden:



Weitere Anwendungen

- fgrep
- Bioinformatik: String Matching
- Intrusion detection: Find certain patterns in sequences of TCP packaged
- Firewall: Apache modsecurity HTTP traffic analysis

Operationen für Formale Sprachen

Sei Σ ein fest gewähltes, endliches Alphabet.
Für Sprachen $A, B \subseteq \Sigma^*$ definiert man:

Vereinigung: $A \cup B = \{w;\ w \in A \text{ oder } w \in B\}$

Durchschnitt: $A \cap B = \{w;\ w \in A \text{ und } w \in B\}$

Verkettung: $A \bullet B = \{xy;\ x \in A \text{ und } y \in B\}$

Iteration $A^* = \{x_1 \ldots x_k;\ k \geq 0 \text{ und } x_i \in A \text{ für alle } i \text{ mit } 1 \leq i \leq k\}$

Reguläre Sprachen

Die Klasse der von endlichen Automaten akzeptierbaren Sprachen heißt auch Klasse der regulären Sprachen, m.a.W:

Eine Sprache $L \subseteq \Sigma^*$ heißt regulär, wenn es einen endlichen Automaten (einen DFA) A gibt mit L = L (A).

Satz: Die Klasse der regulären Sprachen ist abgeschlossen gegenüber Vereinigung, Durchschnitt und Komplement.

UNION **THEOREM**

Given two languages, L_1 and L_2, define the union of L_1 and L_2 as

$$L_1 \cup L_2 = \{ w \mid w \in L_1 \text{ or } w \in L_2 \}$$

***Theorem:* The union of two regular languages is also a regular language**

Theorem: *The union of two regular languages is also a regular language*

Proof: Let

$M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ *be finite automaton for* L_1

and

$M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ *be finite automaton for* L_2

We want to construct a finite automaton

$M = (Q, \Sigma, \delta, q_0, F)$ *that recognizes* $L = L_1 \cup L_2$

Idea: Run both M_1 and M_2 at the same time!

Q = pairs of states, one from M_1 and one from M_2

$= \{ (q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2 \}$

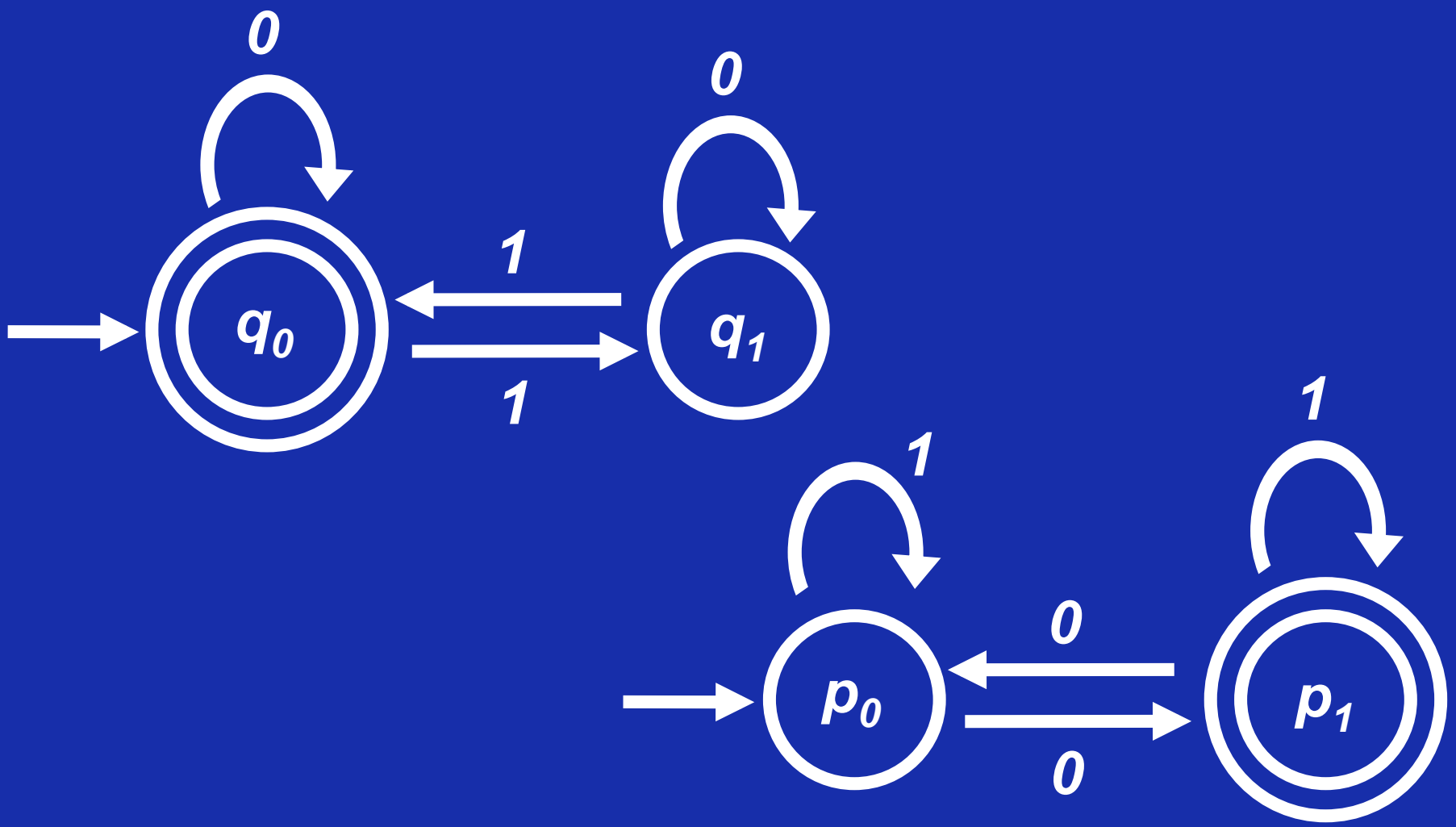
$= Q_1 \times Q_2$

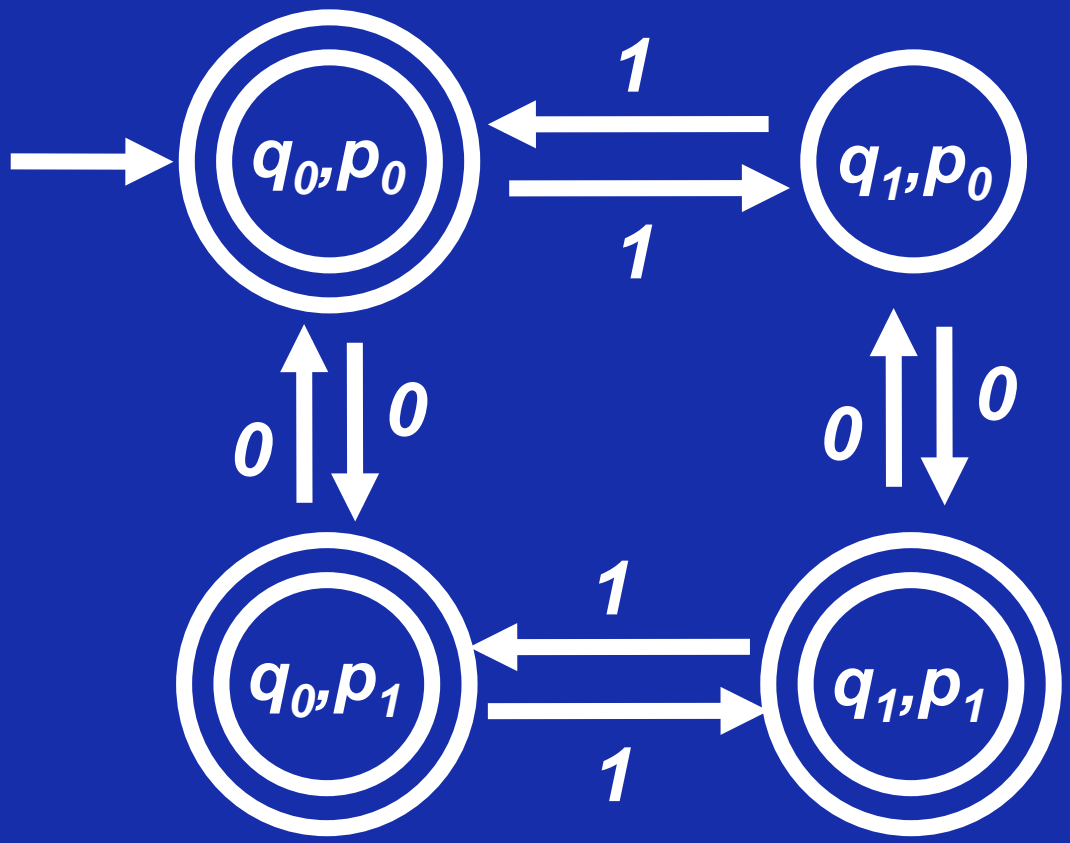
$q_0 = (q_0^1, q_0^2)$

$F = \{ (q_1, q_2) \mid q_1 \in F_1 \text{ or } q_2 \in F_2 \}$

$\delta((q_1,q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$

Theorem: *The union of two regular languages is also a regular language*




q_0,p_0
q_1,p_0
q_0,p_1
q_1,p_1
1
1
0
0
0
0
1
1

Intersection **THEOREM**

Given two languages, L_1 and L_2, define the intersection of L_1 and L_2 as

$$L_1 \cap L_2 = \{ w \mid w \in L_1 \text{ and } w \in L_2 \}$$

***Theorem:* The intersection of two regular languages is also a regular language**