Einführung in die Computerlinguistik: Automaten

Prof. Dr. Katja Markert
WS 2019/20
Institut für Computerlinguistik
(Folien größtenteils von Dr. Yannick Versley, Prof. Dr. Anette Frank, Prof. Dr. Dan Jurafsky; alle Fehler sind meine)

Heute

1. Recap

2. Endliche Automaten: deterministische und nicht-deterministische

3. Äquivalenz endlicher Automaten und regulärer Ausdrücke. Thompson-Algorithmus konvertiert einen regulären Ausdruck in einen nicht-deterministschen Automaten. Kleene-Algorithmus konvertiert Automaten zu regulärem Ausdruck.

4. Äquivalenz deterministischer und nicht-deterministischer Automaten. Powerset Construction konvertiert nicht-deterministische Automaten in deterministische.

Reguläre Sprachen

- **Formale Definition regulärer Sprachen**:

1. Die leere Menge ($\emptyset$) ist eine reguläre Sprache
2. Für alle $a \in \sum \cup \varepsilon$ ist $\{a\}$ eine reguläre Sprache
   ($\sum$: Alphabet)
2. Sind L, L_1 und L_2 reguläre Sprachen, so sind auch
   a. $L_1 \circ L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$, dh. die ***Konkatenation*** von L_1 und L_2
   b. $L_1 \cup L_2 = \{ x \mid x \in L_1 \vee x \in L_2 \}$, die ***Vereinigung oder Disjunktion*** von L_1 und L_2
   c. $L^* = \varepsilon \cup L \cup L \circ L \cup L \circ L \circ L \cup \ldots$, der ***Kleene-Abschluss*** von L_1

reguläre Sprachen

Abschlusseigenschaften regulärer Sprachen

Man kann zeigen:

- Intersektion: sind zwei Sprachen L_1 und L_2 reguläre Sprachen, so ist auch die **Intersektion** von L_1 und L_2 eine reguläre Sprache;

  Differenz: $L_1 – L_2$

  Komplement: $\sum^* – L_1$

  Inversion: L_1^R

Formale Sprachen

Wofür sind Formale Sprachen wichtig?

- Formale *endliche* Spezifikation für die *Erkennung/Generierung einer infiniten Menge sprachlicher Sequenzen*
- Effiziente Algorithmen für automatische Verarbeitung regulaere Sprachen und weiterer Klassen (höherstufiger) formaler Sprachen
- Spezielle Eigenschaften regulärer (und anderer) formaler Sprachen und entsprechender formaler Grammatiken („Chomsky Hierarchie“)
  Welche Klassen formaler Sprachen eignen sich für die Modellierung und Algorithmisierung bestimmter linguistischer Phänomene?
- Reguläre Sprachen (finite Automaten): Modellierung natürlicher Sprache in den Teilbereichen:
  Phonologie, Morphologie, Syntax

Hierarchie formaler Sprachen

Chomsky Hierarchie formaler Sprachen

Reguläre Sprachen

- (Typ-3)
- Kontextfreie Sprachen
- (Typ-2)
- Kontextsensitive Sprachen (Typ-1)
- Typ-0 Sprachen

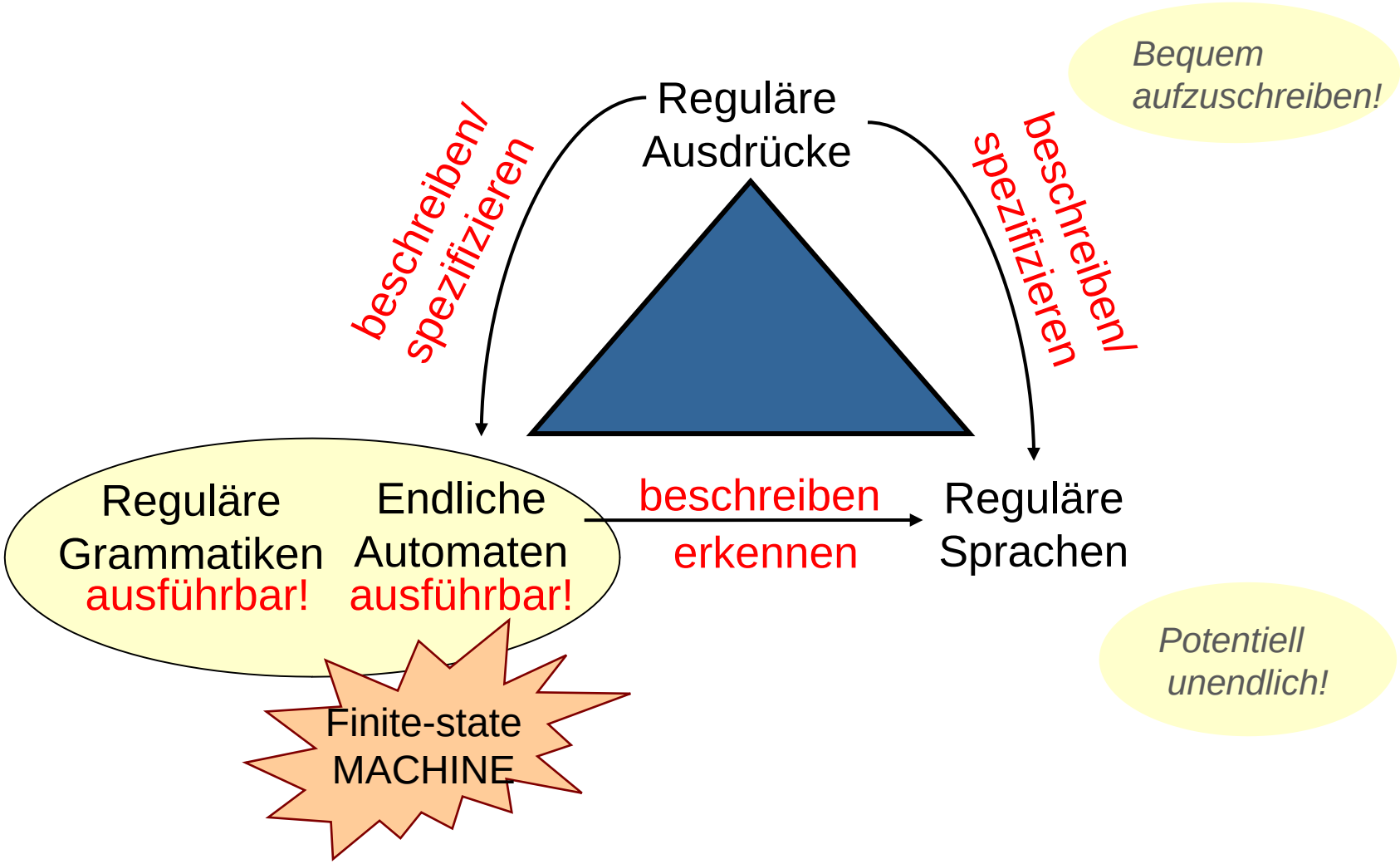
Hierarchie formaler Grammatiken und Automaten

Reguläre Phrasenstruktur (PS)-grammatiken

- Finite-state/endliche Automaten
- Kontextfreie PS Grammatiken
- Push-down (Keller-)Automaten
- Baumadjunktionsgrammatiken
- Linear abgeschl. Automaten
- Allgemeine PS Grammatiken
- Turing Maschine

Steigende Komplexität
Abnehmende Effizienz

Endliche Automaten und regulaere Sprachen



Endliche Automaten

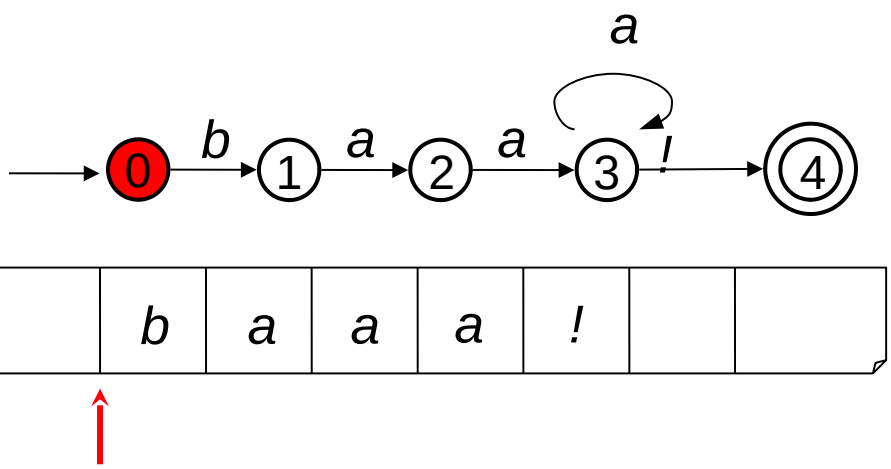
Wie führen wir Suche mit regulären Ausdrücken durch?

- Konstruktion eines **endlichen Automaten A**, der dem regulären Suchausdruck m entspricht
- Abgleich von Eingabezeichenketten s mit Automat A: Erkennungsproblem:
- *Ist s ein Element der durch A definierten Sprache?*

- *Endlicher Automat = Finite State Automaton = FSA = finite state machine*

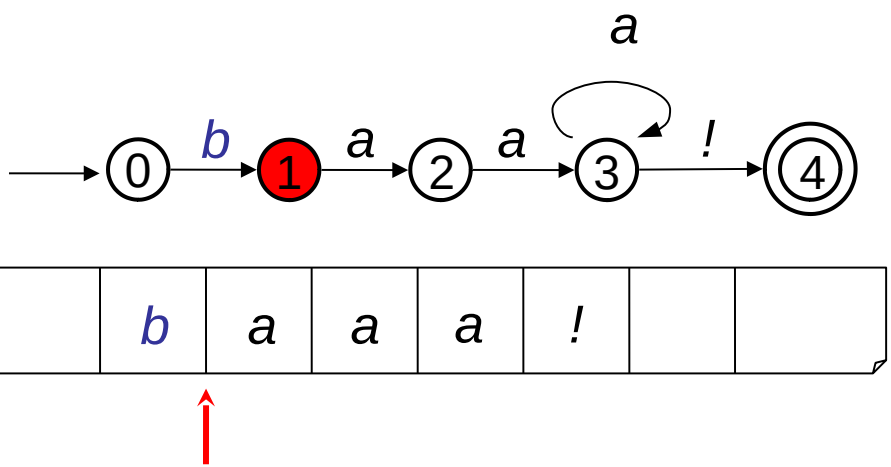
Endliche Automaten

- Datenstruktur: *Gerichteter Graph*
  - Menge von Knoten (*Zustände*)
  - Menge von etikettierten gerichteten Kanten (*Übergänge*) (von einem Knoten zu einem anderen oder demselben Knoten)
  - Start- und Endzustand (bzw. Endzustände)
- Verarbeitung von Zeichenketten mit endlichen Automaten: *Erkennung*
  - Automat:
  - Erkennung:



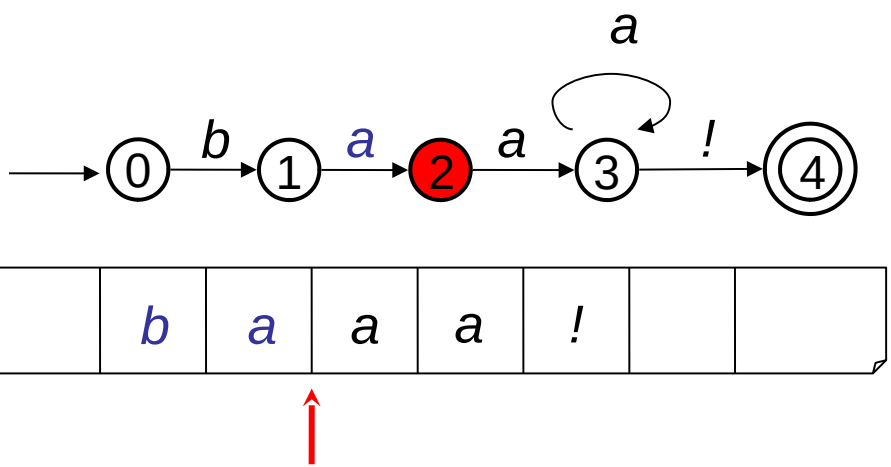
Endliche Automaten

- Datenstruktur: *Gerichteter Graph*
  - Menge von Knoten (*Zustände*)
  - Menge von etikettierten gerichteten Kanten (*Übergänge*) (von einem Knoten zu einem anderen oder demselben Knoten)
  - Start- und Endzustand (bzw. Endzustände)
- Verarbeitung von Zeichenketten mit endlichen Automaten: *Erkennung*
  - Automat:
  - Erkennung:



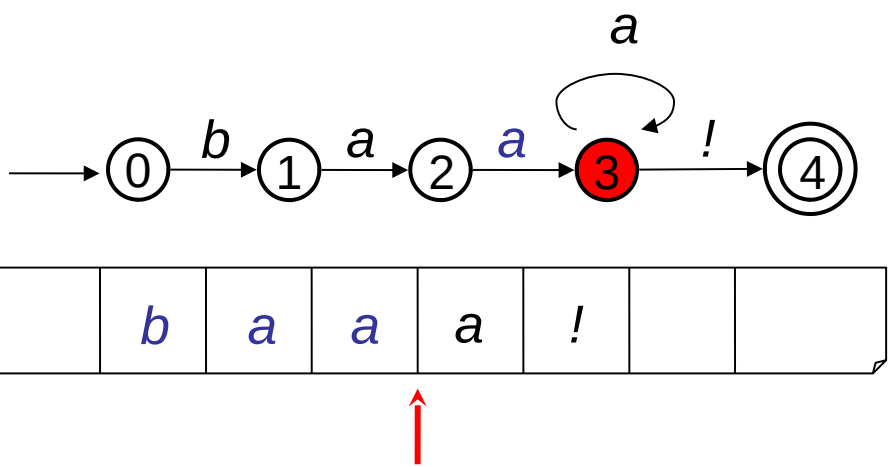
Endliche Automaten

- Datenstruktur: *Gerichteter Graph*
  - Menge von Knoten (*Zustände*)
  - Menge von etikettierten gerichteten Kanten (*Übergänge*) (von einem Knoten zu einem anderen oder demselben Knoten)
  - Start- und Endzustand (bzw. Endzustände)
- Verarbeitung von Zeichenketten mit endlichen Automaten: *Erkennung*
  - Automat:
  - Erkennung:



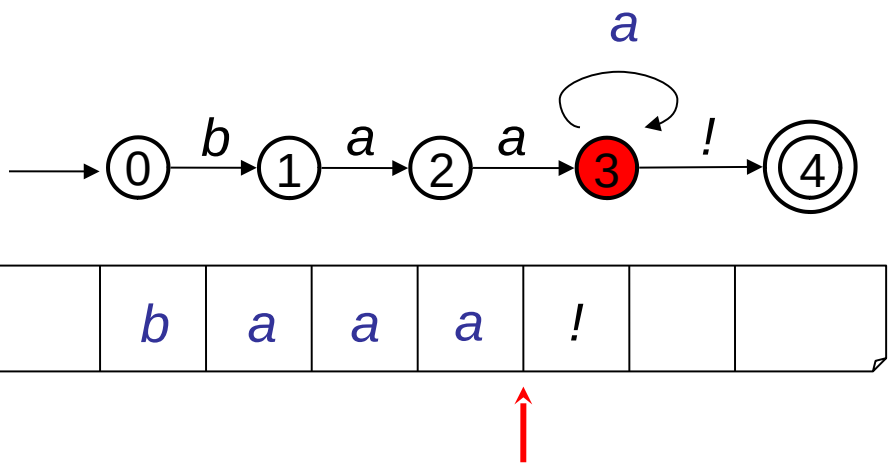
Endliche Automaten

- Datenstruktur: *Gerichteter Graph*
  - Menge von Knoten (*Zustände*)
  - Menge von etikettierten gerichteten Kanten (*Übergänge*) (von einem Knoten zu einem anderen oder demselben Knoten)
  - Start- und Endzustand (bzw. Endzustände)
- Verarbeitung von Zeichenketten mit endlichen Automaten: *Erkennung*
  - Automat:
  - Erkennung:



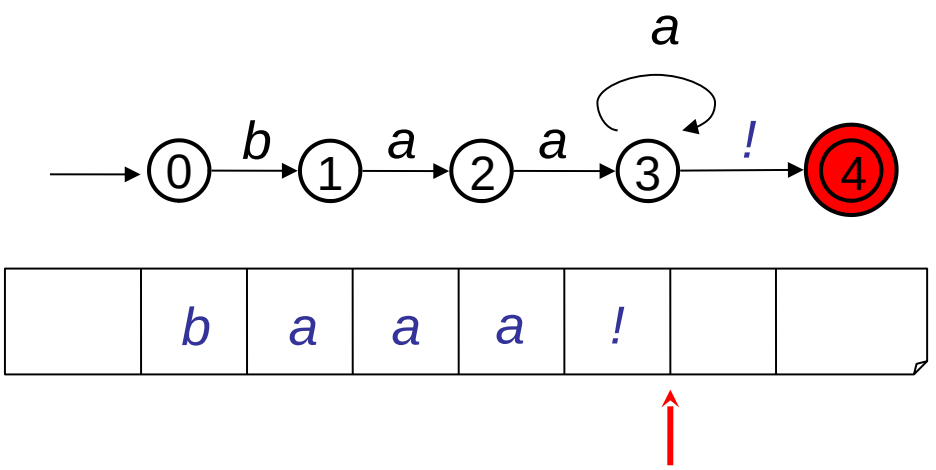
Endliche Automaten

- Datenstruktur: *Gerichteter Graph*
  - Menge von Knoten (*Zustände*)
  - Menge von etikettierten gerichteten Kanten (*Übergänge*) (von einem Knoten zu einem anderen oder demselben Knoten)
  - Start- und Endzustand (bzw. Endzustände)
- Verarbeitung von Zeichenketten mit endlichen Automaten: *Erkennung*
  - Automat:
  - Erkennung:



Endliche Automaten

- Datenstruktur: *Gerichteter Graph*
  - Menge von Knoten (*Zustände*)
  - Menge von etikettierten gerichteten Kanten (*Übergänge*) (von einem Knoten zu einem anderen oder demselben Knoten)
  - Start- und Endzustand (bzw. Endzustände)
- Verarbeitung von Zeichenketten mit endlichen Automaten: *Erkennung*
  - Automat:
  - Eingabe:



Erkannt oder nicht erkannt?

- ***Erkennungsproblem:*** ein Wort w wird von einem Automaten akzeptiert, falls sich der Automat nach dem Lesen von w in einem Endzustand befindet
- ***Erkannte Sprache:*** alle Worte, die der Automat ausgehend vom Startzustand lesen kann, so dass nach dem Lesen ein Endzustand erreicht wird

Gründe für das Nichterkennen eines Wortes:

- Fall 1: Die Eingabe enthält nicht genügend Eingabezeichen um zum Endzustand zu gelangen (z.B. *baa*)
- Fall 2: In einem gegebenen Zustand gibt es keine ausgehende Kante, deren Symbol mit dem nächsten Eingabezeichen übereinstimmt (z.B. *baab!* oder *baa!a*)

Datenstruktur für endliche Automaten

Übergangstabelle

Übergangsrelation δ

- δ(<Zustand>, <Symbol>) = Nachzustand
- Bsp: δ(3, *a*) = 3,
- δ(3, *b*) = undefiniert

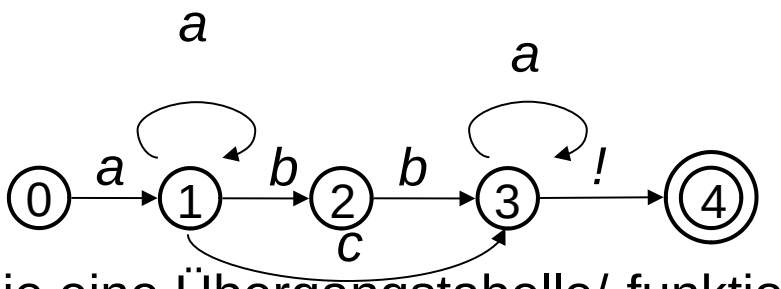
Ein Automat ist *deterministisch*, wenn es in jedem Zustand für jedes Symbol einen eindeutigen Nachzustand gibt (keine Wahl) (δ **ist eine *Funktion*** !)

	Eingabe		
Zustand	*b*	*a*	*!*
0	1	-	-
1	-	2	-
2	-	3	-
3	-	3	4
4	-	-	-

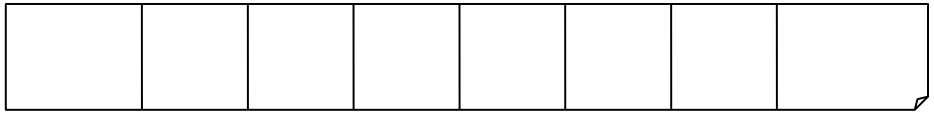
Quick Test

Geben Sie Beispiele für

- Sequenzen, die von diesem Automaten erkannt werden
- Sequenzen, die von diesem Automaten nicht erkannt werden



- Erstellen Sie eine Übergangstabelle/-funktion für diesen Automaten.



Formale Definition eines deterministischen (!) endlichen Automaten

Ein *Deterministischer Endlicher Automat (DEA)* ist ein Fünf-Tupel
$A = < Q, \sum, q_0, F, \delta >$, wobei

- Q : endliche, nichtleere Menge von Zuständen (Knoten) $q_0, \ldots q_n$
- $\sum$: endliches Alphabet von Eingabesymbolen
- $q_0 \in Q$: Anfangszustand
- $F \subseteq Q$: Menge von Endzuständen
- δ : Übergangsfunktion, $\delta : Q \times \sum \rightarrow Q$

Ein Automat ist *deterministisch,* wenn es in jedem Zustand für jedes Eingabesymbol einen eindeutigen (oder keinen) Nachzustand gibt.

Erkennung in *linearer Zeit O(n)* für Eingaben der Länge *n*

- Pro Eingabezeichen ein Test: gibt es einen Übergang oder nicht?
- Anzahl der Tests pro Knoten: maximal die Größe des Alphabets (konstant)

Effizienz ist besonders wichtig für die Verarbeitung großer Dokumente!

Algorithmus für die Erkennung mit DEA

```
function D-Recognize(input, fsa) returns accept or reject
    Index ← 0
    current-state ← initial-state
    loop
        if index = length(input) then
            if current-state is a final-state then
              return accept
            else
              return reject
        elsif transitions[current-state, input[index]] ist leer then
            return reject
        else
            current_state ← transitions[current-state, input[index]]
             Index ← index +1
        end
```

a) Alle Eingabesymbole sind konsumiert

Endzustand erreicht

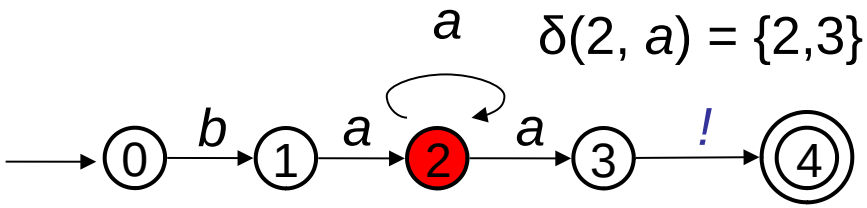
Endzustand nicht erreicht

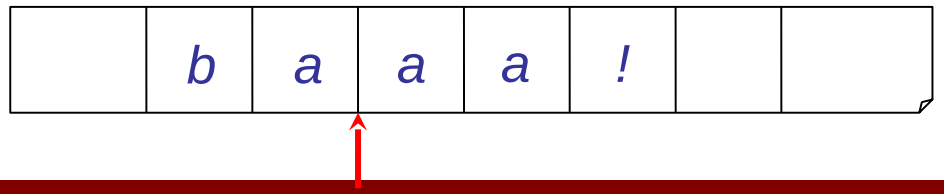
Kein Übergang für aktuellen Input

b) Vollziehe Übergang für aktuelles Eingabesymbol

Nichtdeterministische Endliche Automaten (NEA)

- Es gibt auch *nichtdeterministische Endliche Automaten (NEA).* Hier kann es für einen Zustand und ein Eingabesymbol *unterschiedliche Nachzustände* geben: Beim Durchlaufen des Automaten muss also eine *Wahl* getroffen werden.

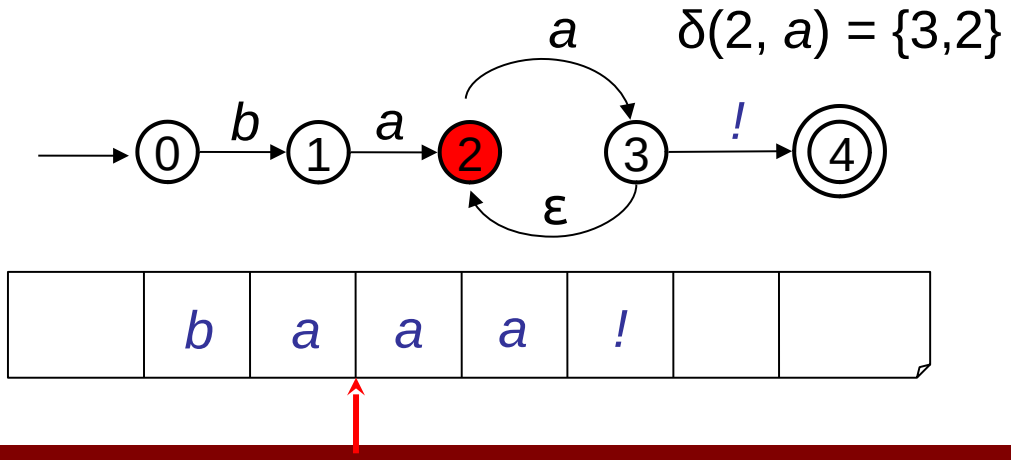




Nichtdeterminismus:
a) Alternative Übergänge für ein Eingabesymbol

Nichtdeterministische Endliche Automaten (NEA)

- Es gibt auch *nichtdeterministische Endliche Automaten (NEA).* Hier kann es für einen Zustand und ein Eingabesymbol *unterschiedliche Nachzustände* geben: Beim Durchlaufen des Automaten muss also eine *Wahl* getroffen werden.



Nichtdeterminismus:
b) durch ε-Übergänge

Nichtdeterministische Endliche Automaten (NEA)

Ein nicht-d*eterministischer endlicher Automat (ε-NEA)* ist ein Fünf-Tupel

$A = < Q, \sum, q_0, F, \delta >$, wobei

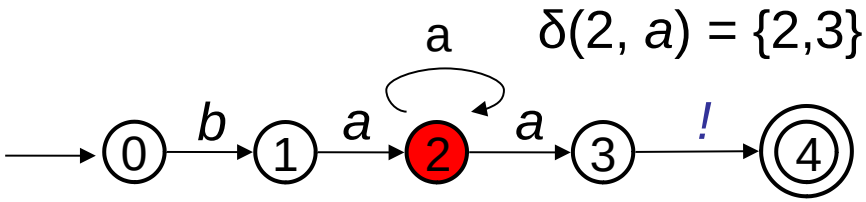
Q : endliche, nichtleere Menge von Zuständen (Knoten) $q_0, \ldots q_n$

$\sum$: endliches Alphabet von Eingabesymbolen

$q_0 \in Q$: Anfangszustand

$F \subseteq Q$: Menge von Endzuständen

$\delta \subseteq Q \times (\sum \cup \{\varepsilon\}) \times Q$ bzw δ: $Q \times (\sum \cup \{\varepsilon\}) \rightarrow P(Q)$ (Potenzmenge von Q)



Wie sieht die Übergangstabelle in unserem Beispiel aus?

Nichtdeterministische Endliche Automaten (NEA)

Konsequenzen für den Algorithmus:

- Um zu entscheiden, ob eine Zeichenfolge erkannt (d.h. in der Sprache enthalten) ist oder nicht, müssen potentiell *alle* Möglichkeiten untersucht werden.
- Verschiedene algorithmische Ansätze:
- **Backtracking:** choice points, zurücksetzen zu verbleibenden Alternativen.
- **Vorausschau (Look-ahead):** Bevor wir einen Übergang nehmen, versuchen wir durch *Vorausschau* zu entscheiden, welche Alternativen in der Folge möglich bzw. ausgeschlossen sind.
- **Parallele Verarbeitung:** *Paralleles Austesten aller Möglichkeiten*. Hierfür gibt es spezielle Algorithmen.

  Üblicherweise: Konstruktion des **Potenzautomaten**: jeder möglichen Menge von NEA-Zuständen entspricht ein DEA-Zustand

Backtrackingbeispiel mit Tiefensuche (last in, first out)

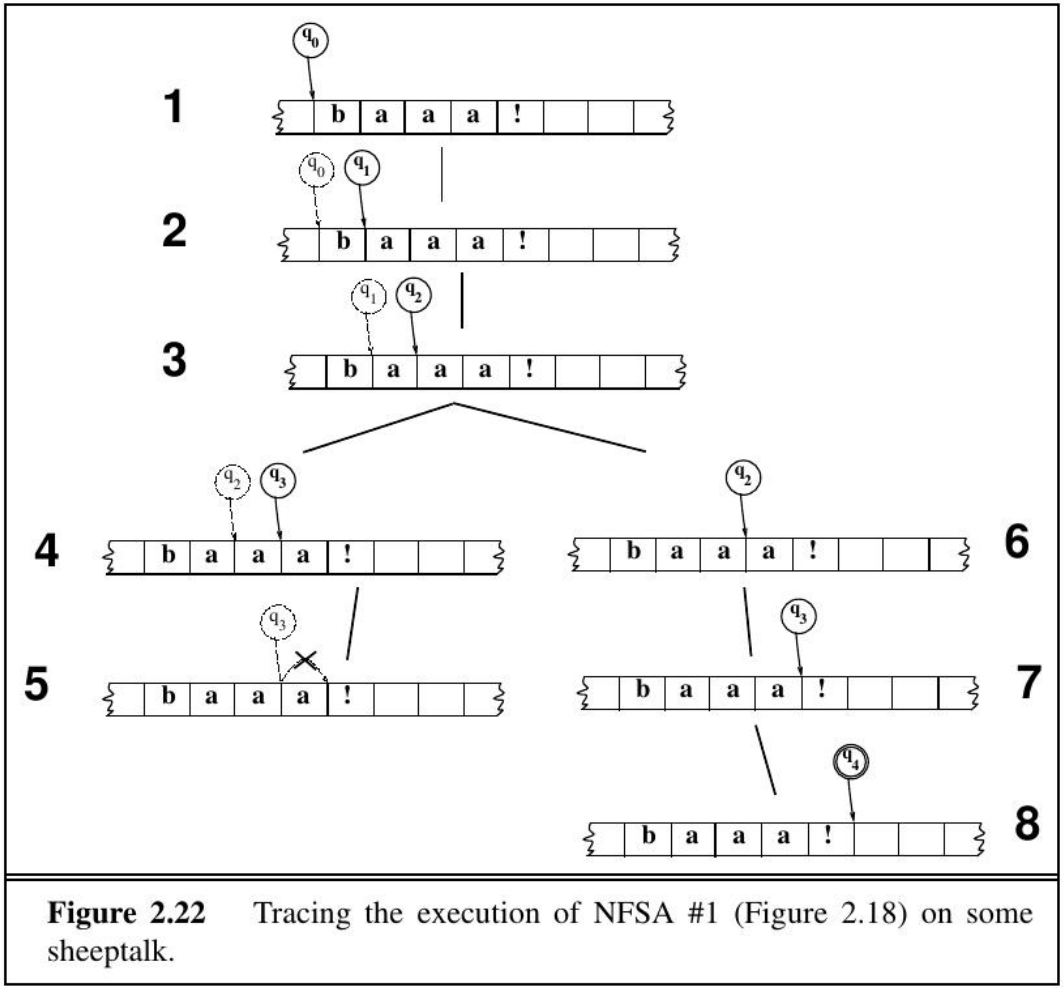


Figure 2.22 Tracing the execution of NFSA #1 (Figure 2.18) on some sheeptalk.

Bild aus Jurafsky und Martin(2nd edition)

Backtrackingbeispiel mit Breitensuche (first in, first out)

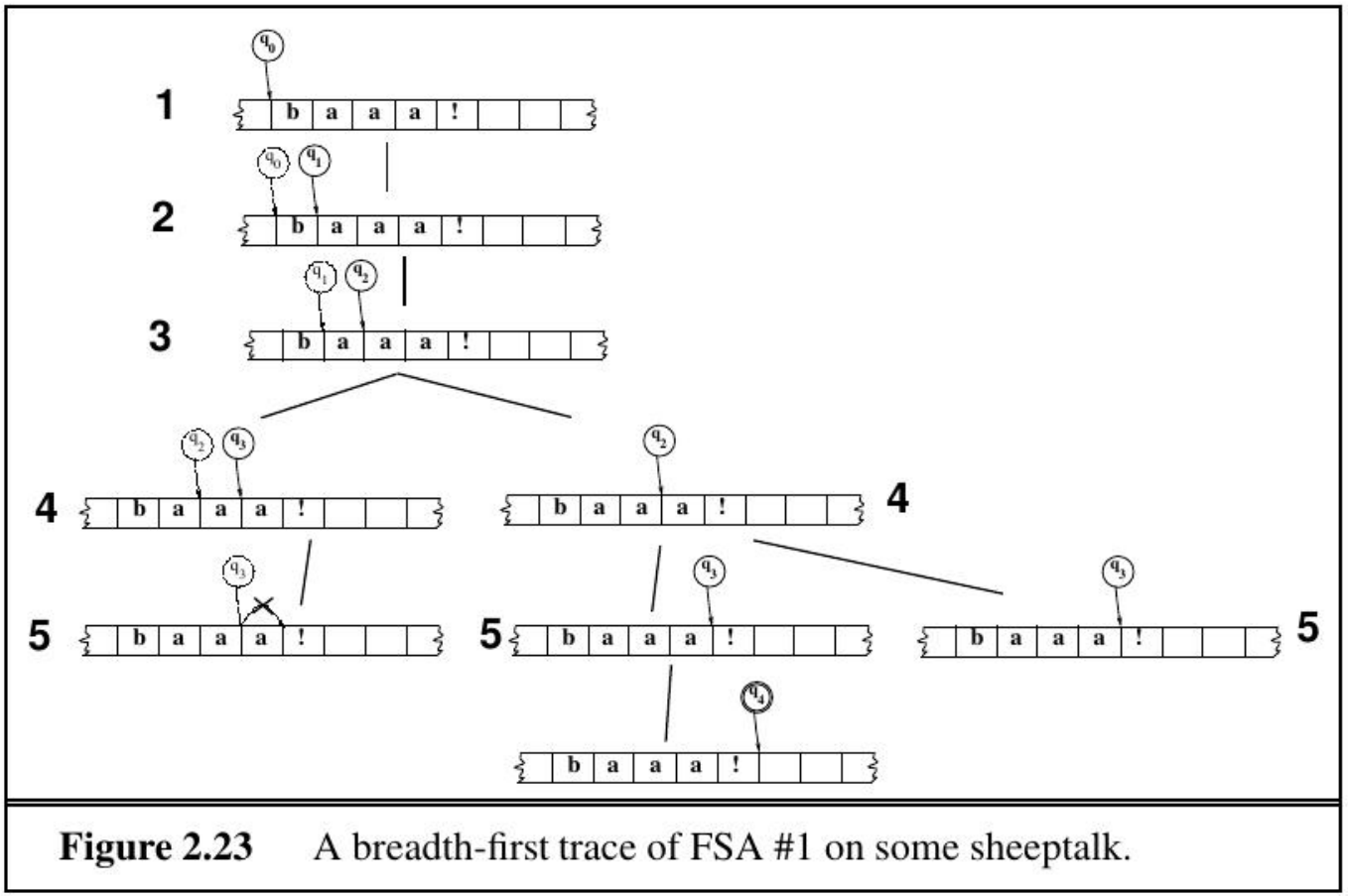


Figure 2.23 A breadth-first trace of FSA #1 on some sheeptalk.

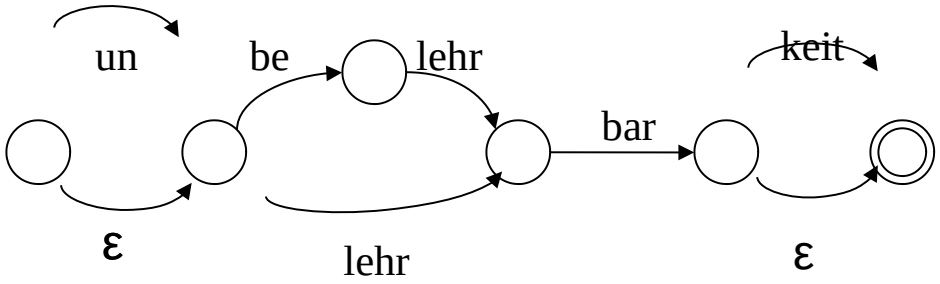
Bild aus Jurafsky und Martin

Wie kommt es zu Nichtdeterminismus?

- Sprache L_{lehr} = { *lehrbar, lehrbarkeit, belehrbar, belehrbarkeit, unbelehrbar, unbelehrbarkeit, unlehrbar, unlehrbarkeit* }

- Regulärer Ausdruck für L_{lehr} : (un | ε) (belehr | lehr) bar (keit | ε)

Konstruktion eines NEA aus dem regulären Ausdruck für L_{lehr}
Verwendung von ε-Übergängen: Übergang in einen Nachzustand durchLesen des leeren Zeichens.



Äquivalenz FSA – reguläre Ausdrücke

- Teil 1: Jeder reguläre Ausdruck kann in einen (evtl. nichtdeterministischen) FSA übersetzt werden.

- Sogenannter Thompson-Algorithmus als konstruktiver Beweis. Erst einfache Tafelbeispiele. Dann Algorithmus. Dann komplexere Übung (ab|c)*

Definition eines FSA durch Reguläre Ausdrücke (Thompson-Algorithmus)

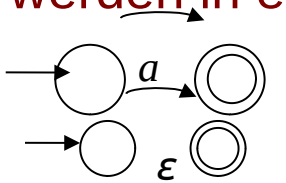
- Jeder reguläre Ausdruck denotiert eine reguläre Sprache

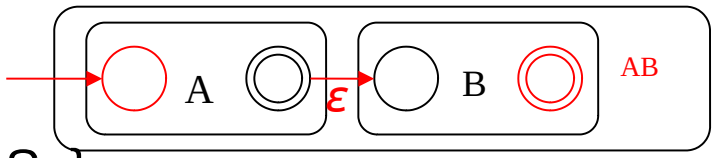
$L(\varepsilon) = \{\varepsilon\}$
$L(a) = \{a\}$ for all a

Abschlusseigenschaften

$L(R1R2) = L(R1)L(R2)$
$L(R1 \cup R2) = L(R1) \cup L(R2)$
$L(R1^*) = L(R1)^*$

- Jeder reguläre Ausdruck kann übersetzt werden in einen FSA.
  - Ein FSA für a (mit $L(a) = \{a\}$)
  - Ein FSA für ε (mit $L(\varepsilon) = \{\varepsilon\}$):
  - Konkatenation von zwei FSAs A und B:
    Verbinde alle Endzustände von A mit dem Anfangszustand von B



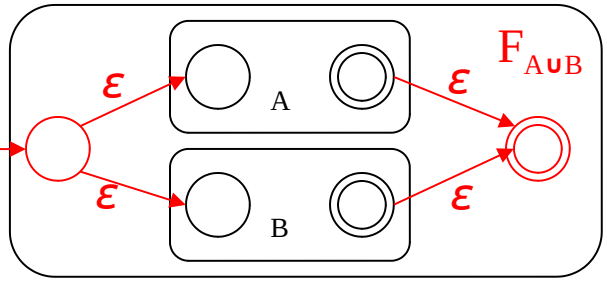


$S_{AB} = S_A$, $F_{AB} = F_B$

$\delta_{AB} = \delta_A \cup \delta_B \cup \{(q_i, \varepsilon, q_j) \mid q_i \in F_A, q_j = S_B\}$

Definition eines FSA durch Reguläre Ausdrücke

- Vereinigung zweier FSAs A und B:



$S_{A \cup B} = q_0$ (neuer Startzustand)

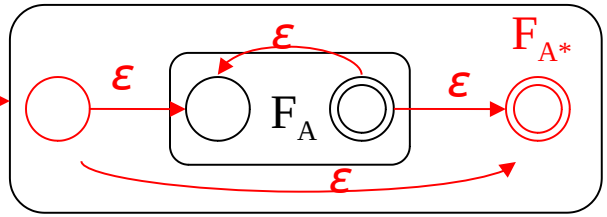
$F_{A \cup B} = \{ q_j \}$ (neuer Endzustand)

$\delta_{A \cup B} = \delta_A \cup \delta_B \cup$

$\{(q_0, \varepsilon, q_z) \mid q_0 = S_{A \cup B}, (q_z = S_A \text{ or } q_z = S_B)\}$

$\cup \{(q_z, \varepsilon, q_j) \mid (q_z \in F_A \text{ or } q_z \in F_B), q_j = F_{A \cup B}\}$

- Kleene Star über einem FSA A :



$S_{A^*} = q_0$ (neuer Startzustand)

$F_{A^*} = \{ q_j \}$ (neuer Endzustand)

$\delta_{A^*} = \delta_A \cup$

$\cup \{(q_i, \varepsilon, q_z) \mid q_i \in F_A, q_z = S_A)\}$

$\cup \{(q_0, \varepsilon, q_z) \mid q_0 = S_{A^*}, (q_z = S_A \text{ or } q_z = F_{A^*})\}$

$\cup \{(q_z, \varepsilon, q_j) \mid q_z \in F_A , q_j = F_{A^*}\}$

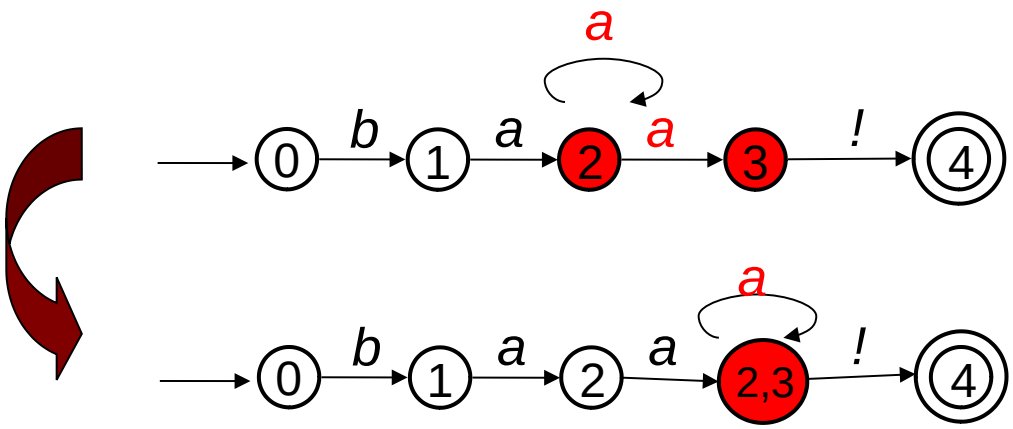
Äquivalenz reguläre Ausdrücke und FSAs

Wir haben hier gesehen, dass jeder reguläre Ausdruck in einen FSA verwandelt werden kann.

Umgekehrt gilt auch, dass jeder FSA in einen regulären Ausdruck übersetzt werden kann.
Für den formalen Beweis (Kleene-Algorithmus)
verweise ich auf Hopcroft et al (2013)
Introduction to automata theory, languages and computation

Äquivalenz von DEA und NEA

- ***Zu jedem NEA gibt es einen äquivalenten DEA, der dieselbe Sprache akzeptiert***
- Algorithmus basiert auf der Idee der *parallelen Verarbeitung* von Alternativen (Äquivalenzklassen; Powerset Construction)



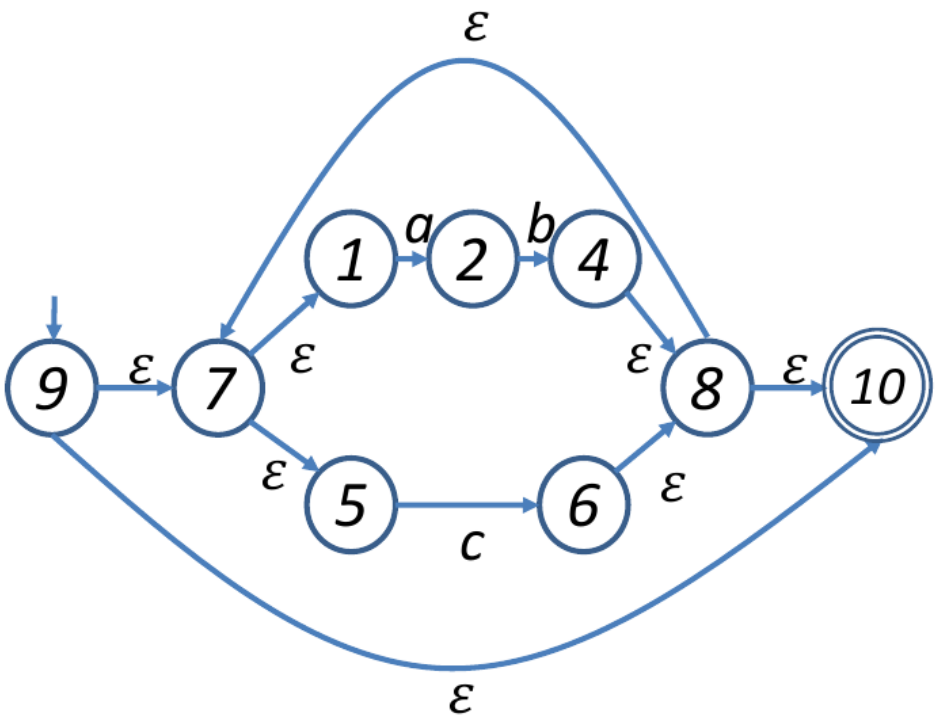
Determinierung durch Powerset Construction

Zwei Funktionen:

- ε-Hüllenbildung: Forme aus einem Zustand s die Menge aller Zustände, die durch ε Übergänge erreicht werden können. Enthält auch den Zustand s selbst.
- Move: nimmt Zustand s und Eingabesymbol und gibt die Menge der Zustände zurück, die von s mit einem einzigem Übergang des Symbols erreicht werden können. Formaler für jeden Zustand s_i:

  - Betrachte *alle* Folgezustände s_x, die mit Lesen eines Zeichens *a* erreicht werden können
  - Fasse alle s_{x1} .. s_{xn} zusammen zu einem neuen Zustand $s_{x1\text{-}xn}$ und verzeichne einen neuen Übergang $\delta(<s_i,a>) = s_{x1\text{-}xn}$

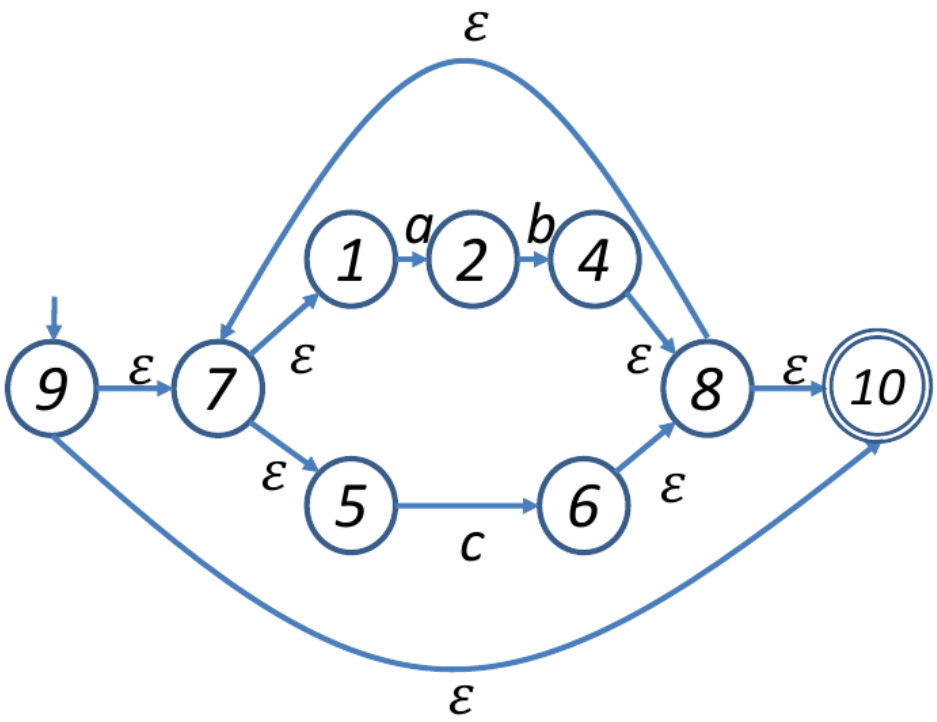
Powerset Construction: Beispiel



NFA	DFA	a	b	c

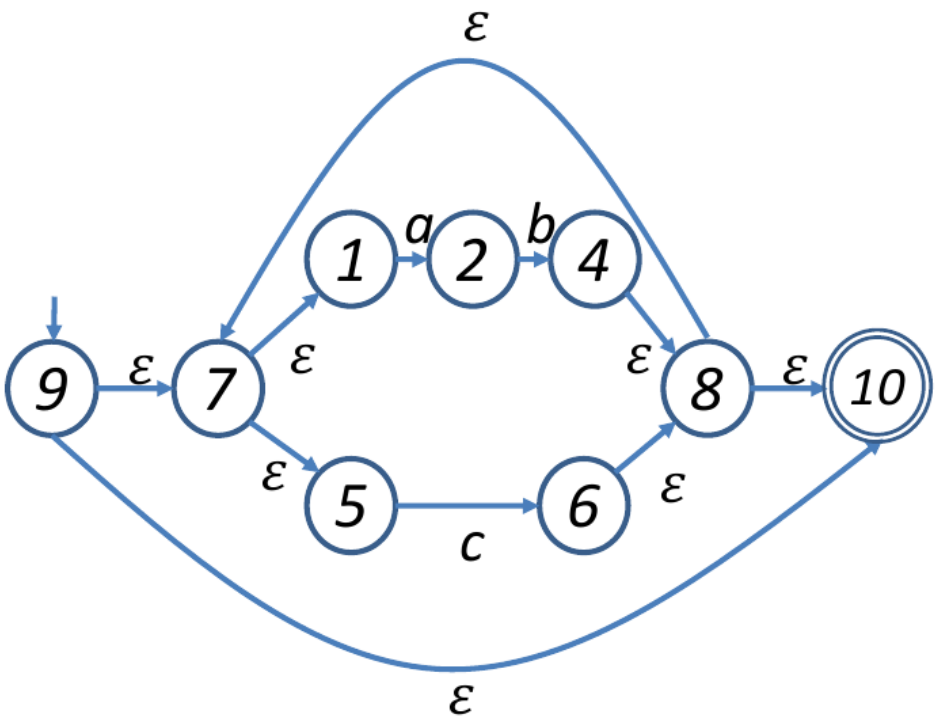
Powerset Construction: Beispiel

ε-closure des Anfangszustandes



NFA	DFA	a	b	c
{9,7,1,5,10}	A			

Powerset Construction: Beispiel

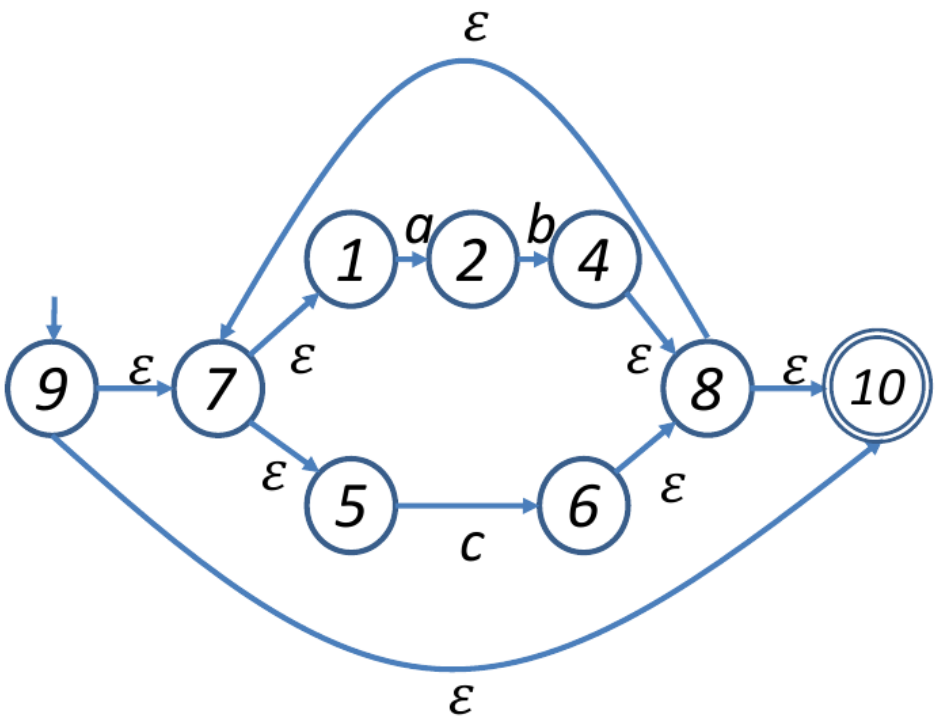


Markiere A

NFA	DFA	a	b	c
{9,7,1,5,10}	A √			

Powerset Construction: Beispiel

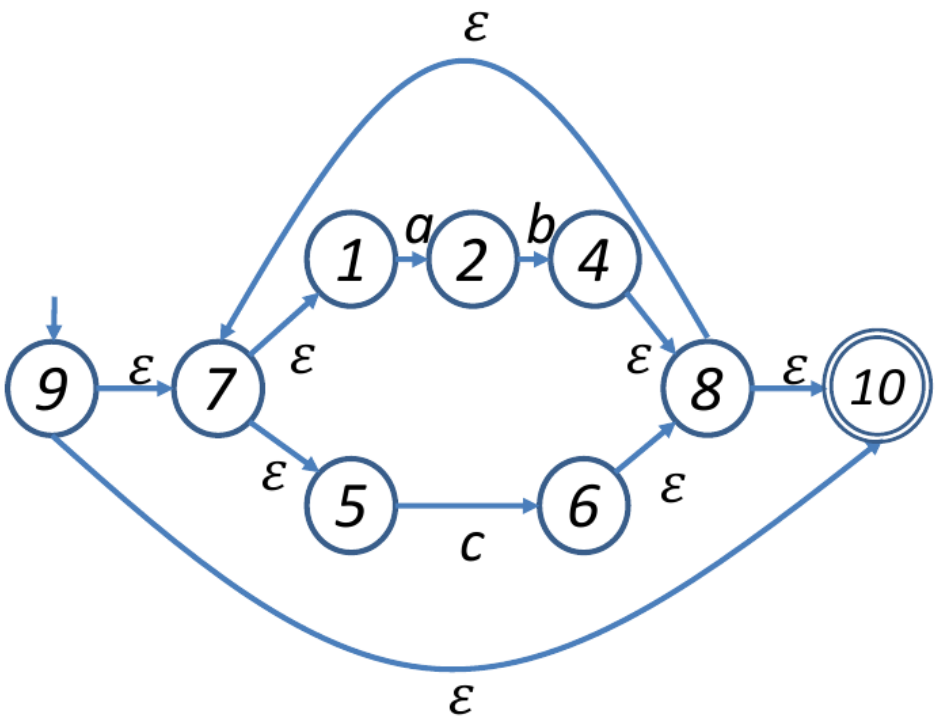
Berechne ε-closure (move (A,a))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B		
{2}	B			

Powerset Construction: Beispiel

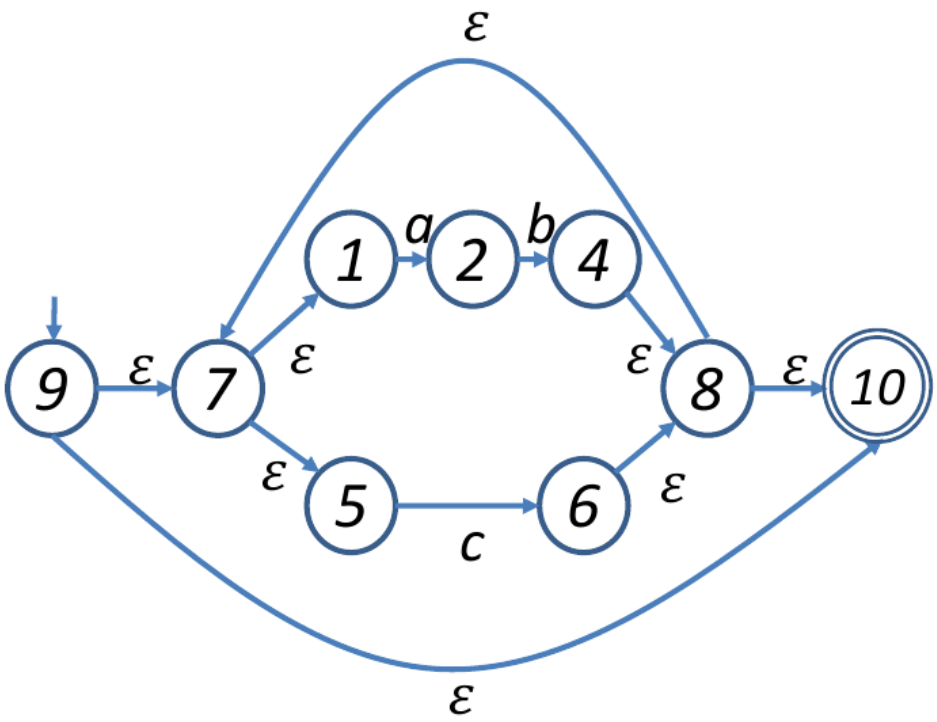
Berechne ε-closure (move (A,b))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	
{2}	B			

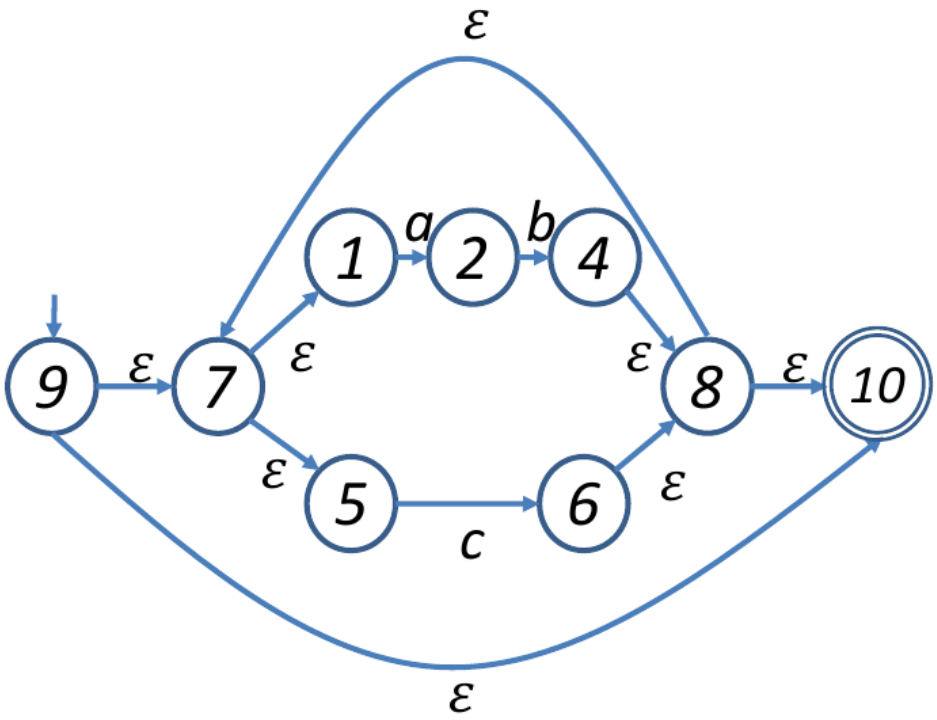
Powerset Construction: Beispiel

Berechne ε-closure (move (A,c))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B			
{6,8,7,10,1,5}	C			

Powerset Construction: Beispiel

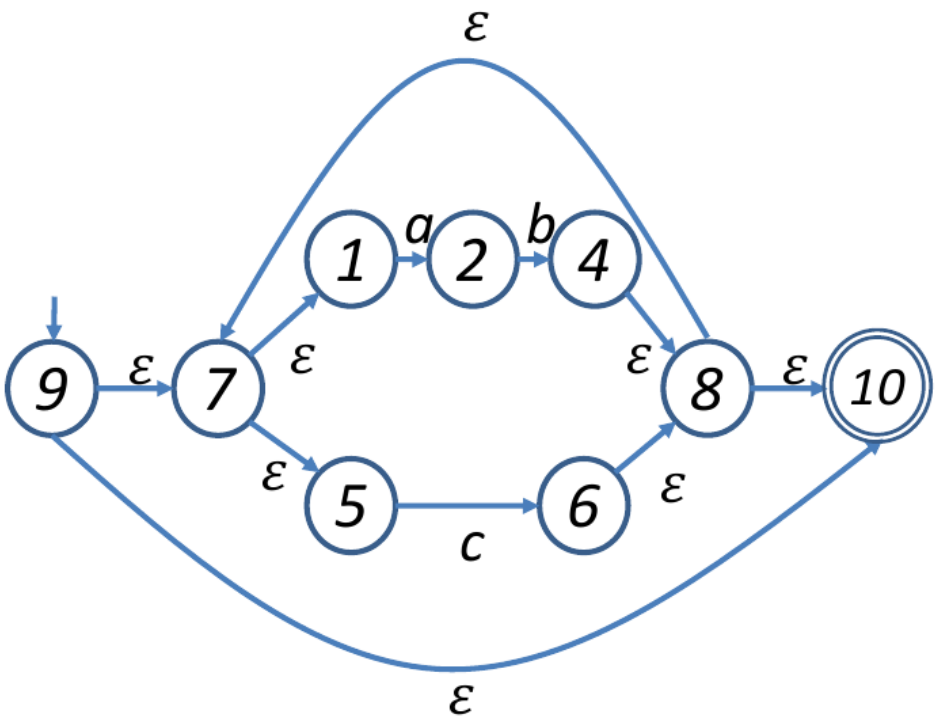


Markiere B

NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √			
{6,8,7,10,1,5}	C			

Powerset Construction: Beispiel

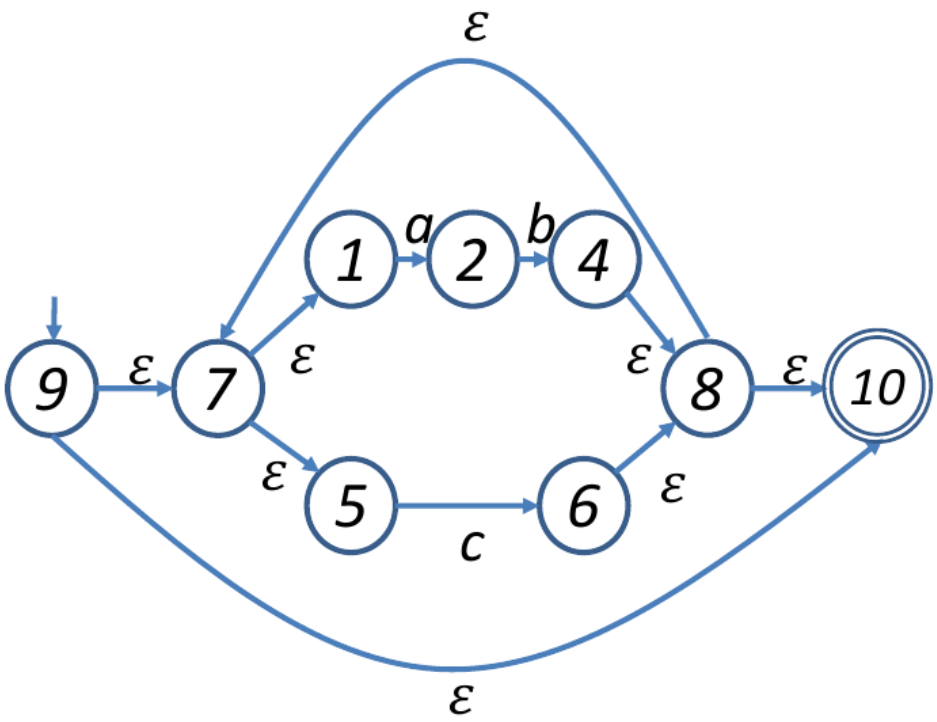
Berechne ε-closure (move (B,a))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-		
{6,8,7,10,1,5}	C			

Powerset Construction: Beispiel

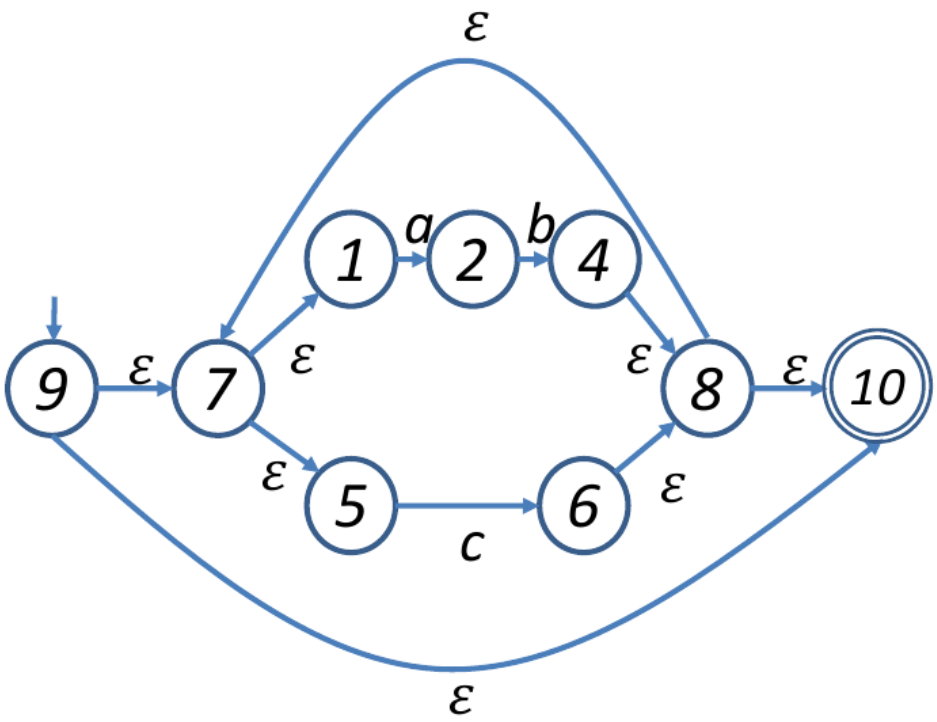
Berechne ε-closure (move (B,b))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	
{6,8,7,10,1,5}	C			
{4,7,8,1,10,5}	D			

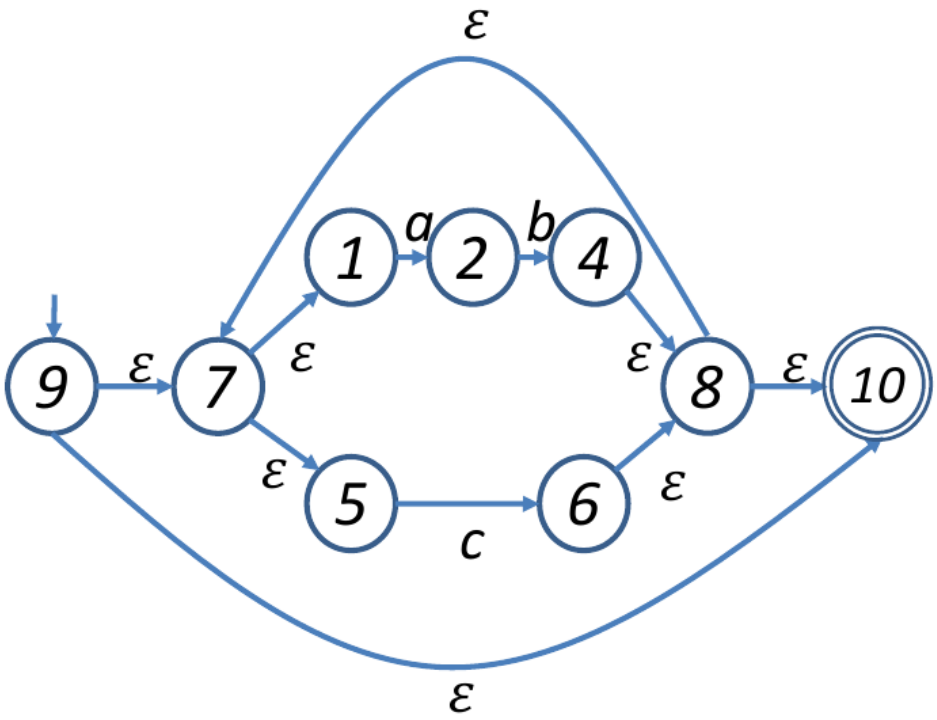
Powerset Construction: Beispiel

Berechne ε-closure (move (B,c))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C			
{4,7,8,1,10,5}	D			

Powerset Construction: Beispiel

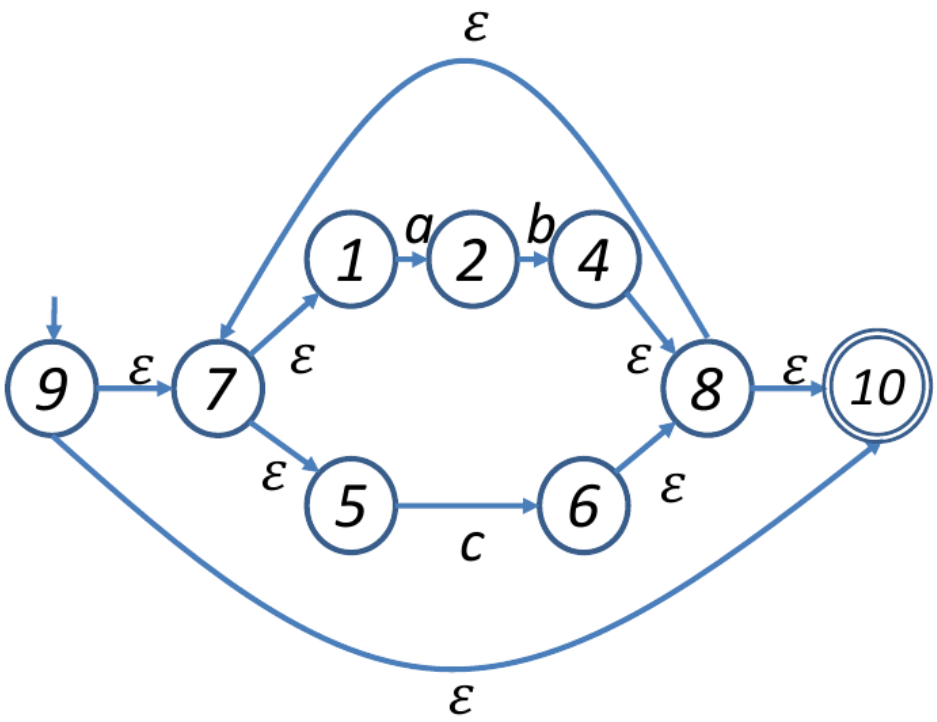


Markiere C

NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √			
{4,7,8,1,10,5}	D			

Powerset Construction: Beispiel

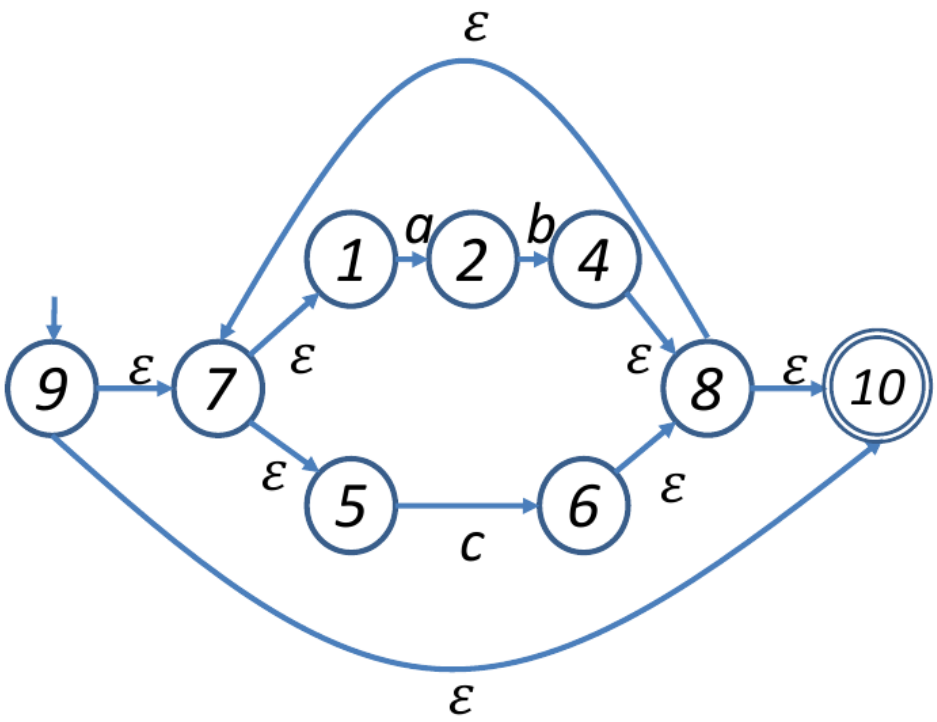
Berechne ε-closure (move (C,a))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B		
{4,7,8,1,10,5}	D			

Powerset Construction: Beispiel

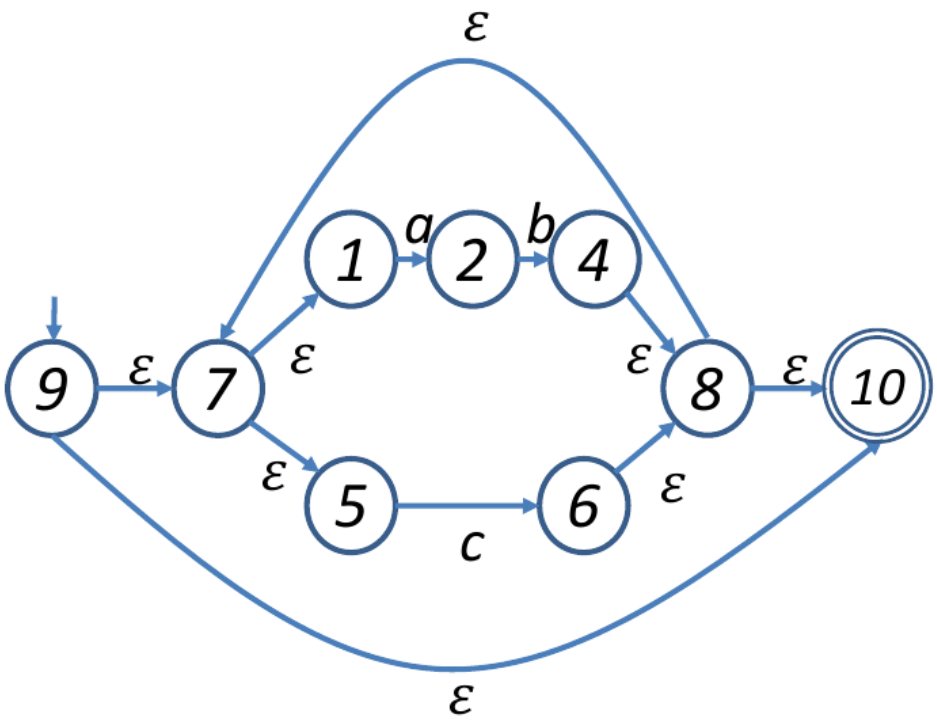
Berechne ε-closure (move (C,b))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B	-	
{4,7,8,1,10,5}	D			

Powerset Construction: Beispiel

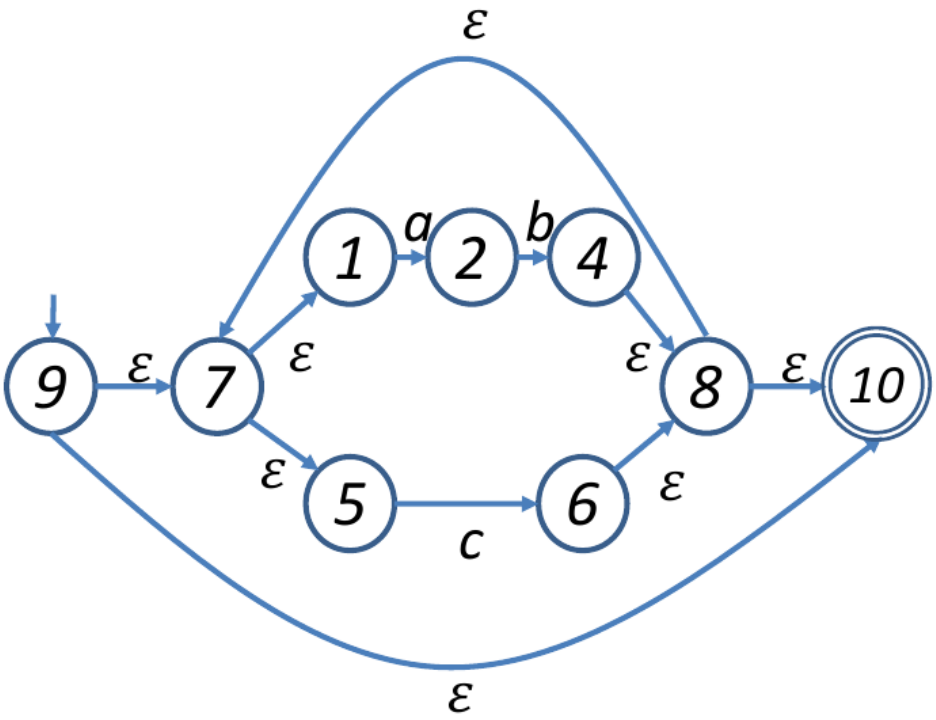
Berechne ε-closure (move (C,b))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B	-	
{4,7,8,1,10,5}	D			

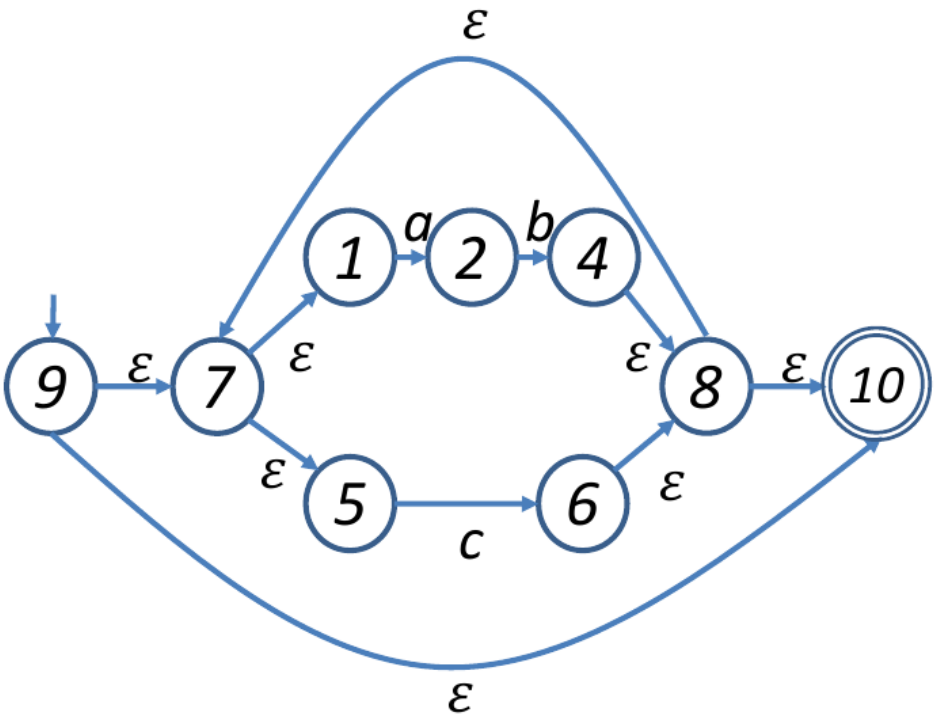
Powerset Construction: Beispiel

Berechne ε-closure (move (C,c))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B	-	C
{4,7,8,1,10,5}	D			

Powerset Construction: Beispiel

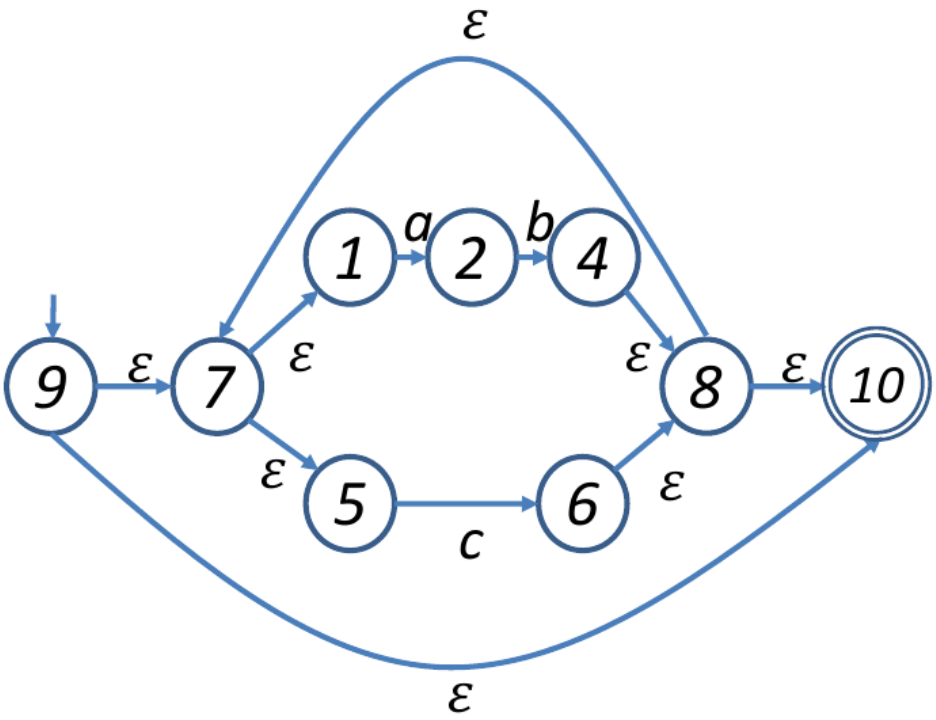


Markiere D

NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B	-	C
{4,7,8,1,10,5}	D √			

Powerset Construction: Beispiel

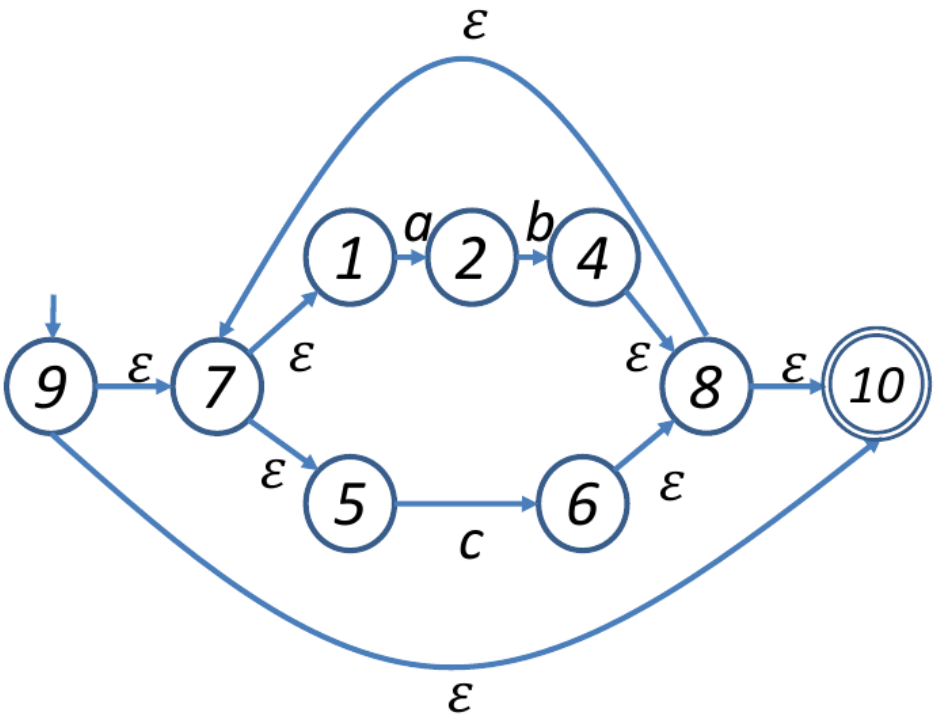
Berechne ε-closure (move (D,a))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B	-	C
{4,7,8,1,10,5}	D √	B		

Powerset Construction: Beispiel

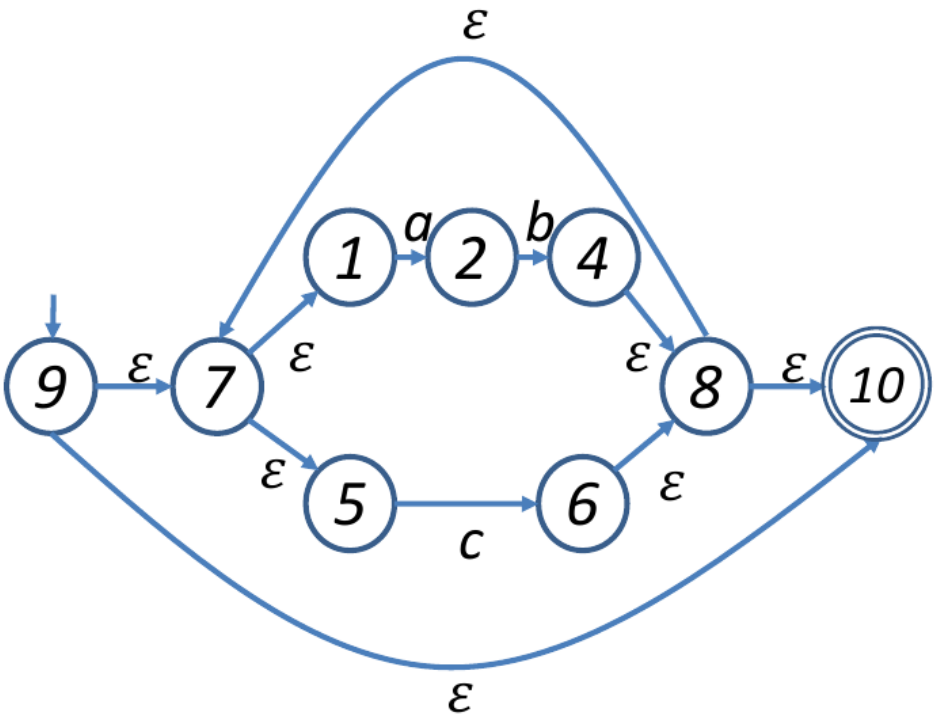
Berechne ε-closure (move (D,b))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B	-	C
{4,7,8,1,10,5}	D √	B	-	

Powerset Construction: Beispiel

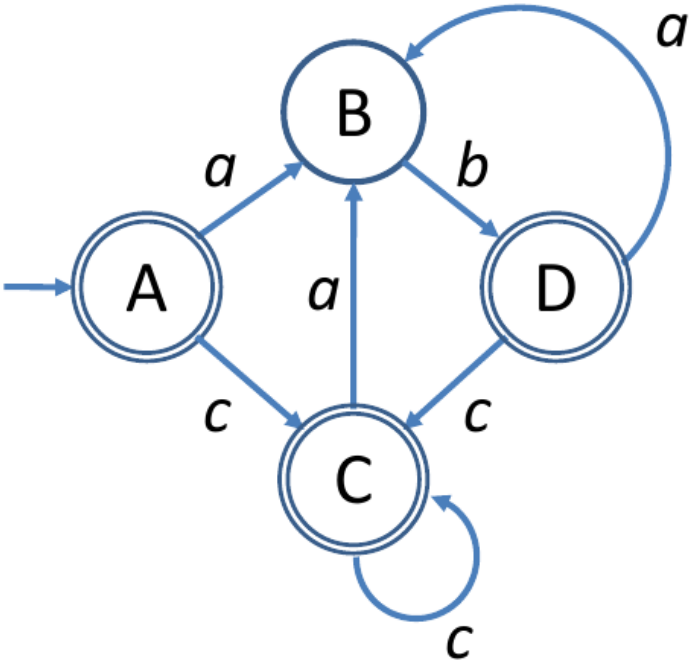
Berechne ε-closure (move (D,c))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B	-	C
{4,7,8,1,10,5}	D √	B	-	C

Powerset Construction: Beispiel

Berechne ε-closure (move (D,c))



NFA	DFA	a	b	c
{9,7,1,5,10}	A √	B	-	C
{2}	B √	-	D	-
{6,8,7,10,1,5}	C √	B	-	C
{4,7,8,1,10,5}	D √	B	-	C

Powerset Construction:Algorithmus

1. Kreiere Anfangszustand der DFA durch ε-closure des Anfangszustands der NFA (ε-closure(q0))

2. Für jeden neuen DFA-Zustand S:

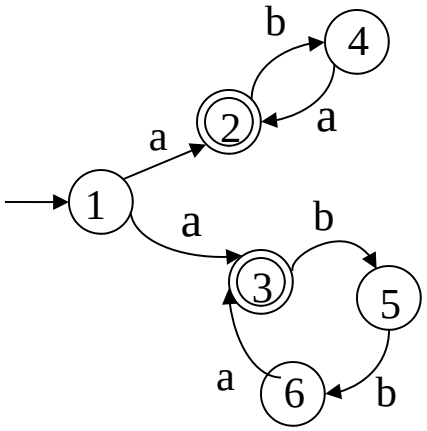
    Für jedes mögliche Inputsymbol I:
        Bilde ε-closure (move (S,i)). Dies gibt (evtl. neuen) Zustand in DFA

3. Wiederhole Schritt 2, bis wir keine neuen DFA-Zustände mehr erhalten

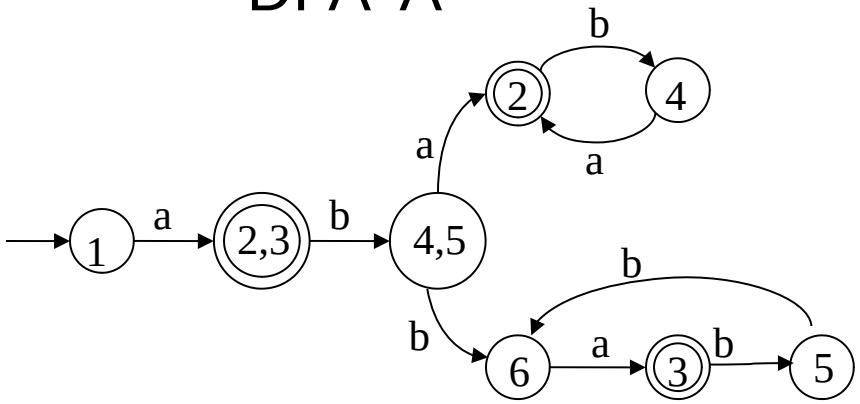
4. Die Endzustände des DFA sind die, die einen Endzustand des NFA enthalten.

Determinierung durch Powerset Construction: Beispiel 2

NFA A



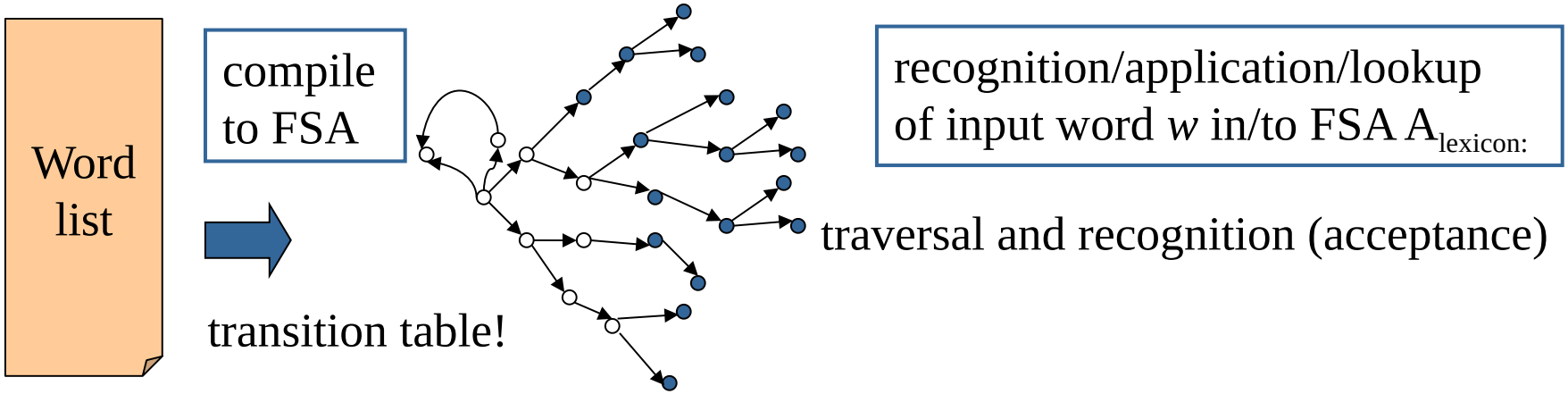
DFA A’



L(A) = L(A’) = a(ba)* ∪ a(bba)*

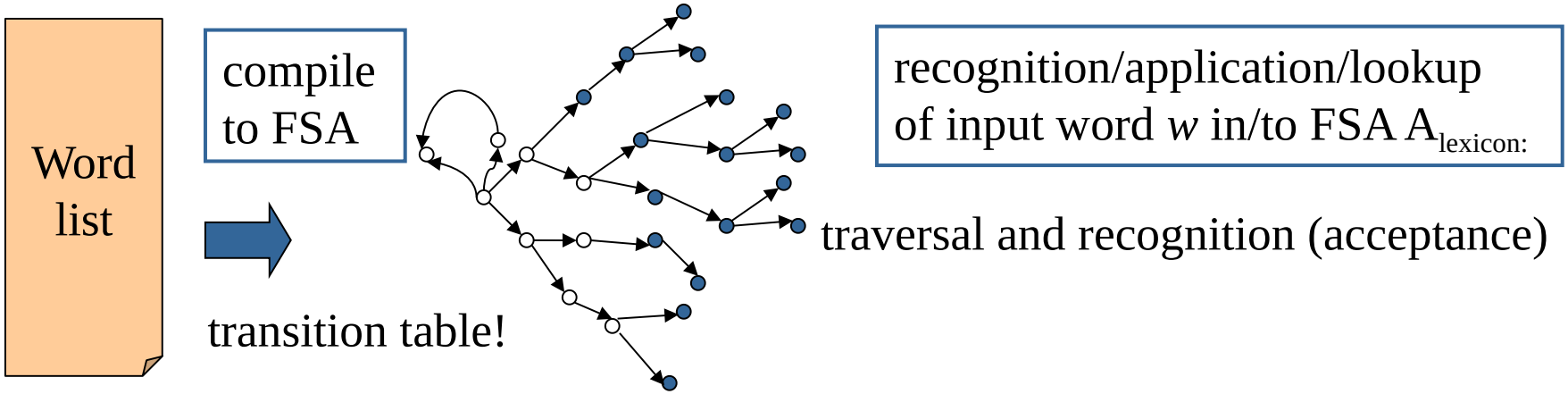
Anwendungen: Reguläre Ausdrücke und FSA

- Erkennung regulärer Sprachen, Manipulation von Zeichenketten
  - Lexikon-basierte Suche: suche im Text nach Wörtern eines Lexikons
  - Kompilation der Lexikoneinträge in einen FSA durch Vereinigung
  - Teste Eingabewörter im Text auf Akzeptanz im Lexikon-FSA



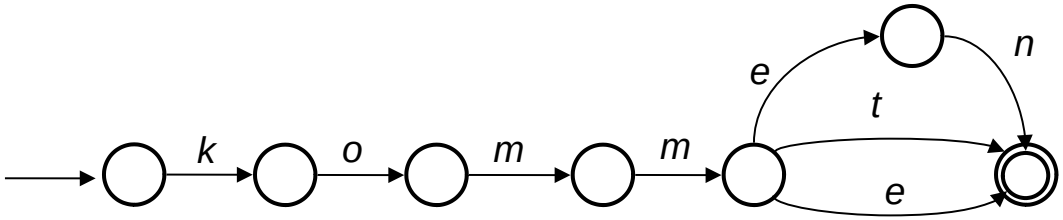
Anwendungen: Reguläre Ausdrücke und FSA

- Erkennung regulärer Sprachen, Manipulation von Zeichenketten
  - Lexikon-basierte Suche: suche im Text nach Wörtern eines Lexikons
  - Kompilation der Lexikoneinträge in einen FSA durch Vereinigung
  - Teste Eingabewörter im Text auf Akzeptanz im Lexikon-FSA

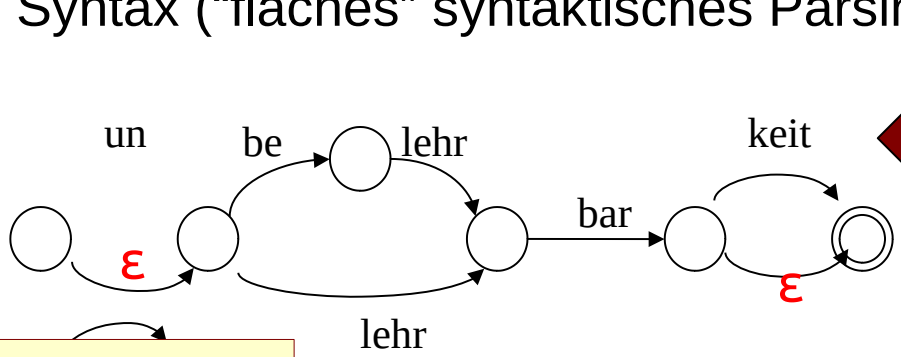


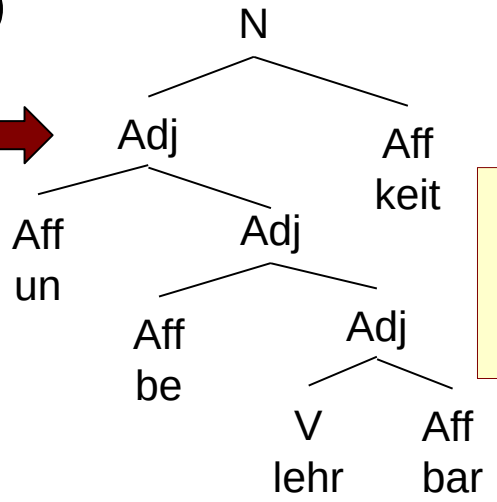
Anwendungen: Reguläre Ausdrücke und FSA

- Modellierung phonologischer und morphologischer Prozesse
  - */komm(e|t|en)/ : {komme, kommt, kommen}*



- Approximierung nicht-regulärer Sprachen in Morphologie und Syntax (“flaches” syntaktisches Parsing)





FSA/
Reg. Grammatik

Kontext-
freie
Grammatik

Off-the-shelf finite-state tools

- **SFST** (Stuttgarter FST-Library, Helmut Schmid)
  Open Source, UTF-8 kompatibel
  http://www.cis.uni-muenchen.de/~schmid/tools/SFST/
  /resources/platforms/sfst-1.3
  pysfst Python-Wrapper
- **Geertjan van Noord's finite-state tools**
  http://www.let.rug.nl/~vannoord/Fsa/
- **OpenFST**
  www.openfst.org; https://github.com/tmbdev/pyopenfst (Python-Wrapper)
- **Xerox finite-state tools**
  XFST Tools (provided with Beesley and Karttunen: Finite-State Morphology, CSLI Publications und: SCL Resources)
  http://www.stanford.edu/~laurik/fsmbook/home.html
  ella: /resources/platforms/xfst-8.1.4

Fazit zweiter Teil

Mit dem Wissen aus diesem Teil können Sie:

- den Unterschied zwischen deterministischen und nichtdeterministischen endlichen Automaten erkennen
- einen nichtdeterministischen Automaten in einen deterministischen umwandeln (Potenzautomat)
- aus einem regulären Ausdruck einen (nichtdeterministischen) endlichen Automaten machen.

Literatur:

- Aufgabenblatt 2
- Jurafsky und Martin Kapitel 2.2 (2nd edition)
- Hopcroft et al. (2013)
  Introduction to automata theory, languages and computation