

# Theoretische Informatik I

## Einheit 2.3

### Reguläre Ausdrücke



1. Anwendungen
2. Syntax und Semantik
3. Vereinfachungsregeln
4. Beziehung zu endlichen Automaten

- **Automaten beschreiben Abarbeitung von Sprachen**
  - **Operationale Semantik**: Symbole führen zu Zustandsänderungen
  - Bestimmte Wörter bzw. Symbolketten werden durch Zustände akzeptiert
  - Für Automaten ist **Sprache**  $\hat{=}$  Menge der akzeptierten Wörter
- **Wie beschreibt man Eigenschaften von Wörtern?**
  - **Deklarative Semantik**: äußere Form von Zeichenreihen einer Sprache  
z.B. *Wörter haben eine führende Null, dann beliebig viele Einsen*
  - Anwendungen brauchen präzise Beschreibungssprache für Wörter
    - Grundeinheiten von Programmiersprachen, Suchmuster für Browser, ...
- **Reguläre Ausdrücke als formale Syntax**
  - Kurze, prägnante Beschreibung des Aufbaus der Wörter einer Sprache  
z.B.  $01^*$ : “Zuerst eine Null, dann beliebig viele Einsen”

- **Suche nach Mustern in Texten**

- Suche ob/wo/wie oft eine bestimmte Zeichenkette im Text erscheint
- Textmuster kann Platzhalter enthalten

- **Beschreibe Textmuster durch reguläre Ausdrücke**

- Zahl: Ziffernfolge dann evtl. Punkt und nichtleere Ziffernfolge
- Formaler Ausdruck:

$(0+1+\dots+9)^*(\epsilon+(\cdot(0+1+\dots+9)(0+1+\dots+9)^*))$

- **Vielfältige Anwendungen**

- Google Suche nach einfachen Texten
- Erweiterte Google Suche nach Textmustern
- Unix Kommando **grep**: suche nach Textmustern in Dateien
- Programmiersprachen wie **PERL** und **AWK**
- Textsuche und Textersetzung in **Emacs**
- Lexikalische Analyse in Compilern

## Wichtigster Grundbestandteil von Compilern

- **Reguläre Ausdrücke beschreiben Token**
  - Logische Grundeinheiten von Programmiersprachen
  - z.B. Schlüsselwörter, Bezeichner, Dezimalzahlen, ...
- **“Lexer” transformieren reguläre Ausdrücke in Analyseprogramme**
  - Analyse kann die Token der Programmiersprache identifizieren
  - Zugrundeliegende Technik:  
Umwandlung regulärer Ausdrücke in DEAs

# REGULÄRE AUSDRÜCKE PRÄZISIERT (SYNTAX)

- **Syntax: Terme über  $\Sigma \cup \{\emptyset, \epsilon, +, \circ, *, (, )\}$**

Reguläre Ausdrücke sind induktiv wie folgt definiert

- $E = a$  ist ein regulärer Ausdruck für jedes  $a \in \Sigma$
- $E = \emptyset$  und  $F = \epsilon$  sind reguläre Ausdrücke
- Sind  $E$  und  $F$  reguläre Ausdrücke, dann sind auch  $E \circ F$ ,  $E^*$ ,  $E + F$  und  $(E)$  sind reguläre Ausdrücke

Mehr Ausdrücke möglich, aber nicht erforderlich

- **Konventionen zur Vereinfachung**

- $E \circ F$  wird üblicherweise als  $EF$  abgekürzt
- Definitivische Abkürzungen:  $E^+ \equiv EE^*$ ,  $[a_1 \dots a_n] \equiv a_1 + \dots + a_n$
- **Prioritätsregelungen** ermöglichen, überflüssige Klammern wegzulassen
  - $*$  (“Sternoperator”) bindet stärker als  $\circ$ , und dies stärker als  $+$
  - Verkettung  $\circ$  und Alternative  $+$  sind assoziativ

# REGULÄRE AUSDRÜCKE PRÄZISIERT (SEMANTIK)

- Reguläre Ausdrücke beschreiben **Sprachen** über  $\Sigma$
- Die Sprache  $L(E)$  ist **induktiv definiert**
  - Für für alle  $a \in \Sigma$  ist  $L(a) = \{a\}$  (einelementige Sprache, die nur  $a$  enthält)
  - $L(\emptyset)$  ist die leere Sprache (üblicherweise geschrieben als  $\emptyset$  oder  $\{\}$ )
  - $L(\epsilon) = \{\epsilon\}$  (einelementige Sprache, die nur das leere Wort enthält)
  - $L(E \circ F) = L(E) \circ L(F) = \{vw \mid v \in L(E) \wedge w \in L(F)\}$ 
    - steht für die **Verkettung (der Wörter) zweier Sprachen**
  - $L(E^*) = (L(E))^* = \{w_1 w_2 \dots w_n \mid n \in \mathbb{N} \wedge w_i \in L(E)\}$ 
    - \* steht für **Verkettung beliebig vieler Wörter einer Sprache (Kleene'sche Hülle)**
  - $L(E + F) = L(E) \cup L(F) = \{w \in \Sigma^* \mid w \in L(E) \vee w \in L(F)\}$ 
    - + steht für die **Vereinigung zweier Sprachen**
  - $L((E)) = L(E)$

# SPRACHEN VS. AUSDRÜCKE

- **Sprachen sind Mengen von Wörtern**

- Abstraktes semantisches Konzept: Ungeordnete Kollektion von Wörtern
- Beschreibung von Mengen (auf Folie, Tafel,...) benötigt textuelle **Notation**
- Notation benutzt **Kurzschreibweisen** wie  $\cup$ ,  $\circ$ ,  $*$  für Mengenoperationen  
... aber ist selbst nur ein Hilfsmittel zur Kommunikation

- **Reguläre Ausdrücke sind Terme**

- Eine syntaktische Beschreibungsform, die ein Computer versteht
- Reguläre Ausdrücke werden zur Beschreibung von Sprachen benutzt und sind **ähnlich** zur Standardnotation von Mengen

- **Reguläre Ausdrücke sind selbst keine Sprachen**

- Unterscheide **Ausdruck**  $E$  von **Sprache** des Ausdrucks  $L(E)$
- Man verzichtet auf den Unterschied wenn der Kontext eindeutig ist

# BEISPIELE REGULÄRER AUSDRÜCKE

- $a^*ba^*$ 
  - steht für die Menge aller Wörter über  $\Sigma=\{a,b\}$ , die genau ein  $b$  enthalten
  - $L(a^*ba^*) = \{w \in \Sigma^* \mid w \text{ enthält genau ein } b\} = \{w \in \Sigma^* \mid \#_b(w)=1\}$
- $\Sigma^*b\Sigma^*$ 
  - steht für  $\{w \in \Sigma^* \mid w \text{ enthält mindestens ein } b\} = \{w \in \Sigma^* \mid \#_b(w) \geq 1\}$
- $a^*(b+\epsilon)a^*$ 
  - steht für  $\{w \in \Sigma^* \mid w \text{ enthält maximal ein } b\} = \{w \in \Sigma^* \mid \#_b(w) \leq 1\}$
- $a\emptyset$ 
  - steht für die **leere Sprache**, denn die Verkettung einer Sprache mit der leeren Sprache ist immer leer
- $\emptyset^*$ 
  - steht für die Menge  $\{\epsilon\}$ , denn die beliebige Verkettung von Wörtern einer Menge enthält immer das leere Wort



**Beschreibe Menge aller Wörter, in denen 0 und 1 abwechseln**

## 1. Regulärer Ausdruck für die Sprache $\{01\}$

- 0 repräsentiert  $\{0\}$ , 1 repräsentiert  $\{1\}$
- Also ist  $L(01) = L(0) \circ L(1) = \{0\} \circ \{1\} = \{01\}$

## 2. Erzeuge $\{01, 0101, 010101, \dots\}$ durch Sternbildung

- $L((01)^*) = L(01)^* = \{01\}^* = \{\epsilon, 01, 0101, 010101, \dots\}$

## 3. Manche Wörter nicht erfaßt

- Start mit Eins statt Null:  $(10)^*$
- Start und Ende mit Null:  $(01)^*0$
- Start und Ende mit Eins:  $(10)^*1$

Vollständiger Ausdruck:  $(01)^* + (10)^* + (01)^*0 + (10)^*1$

## 4. Es geht auch kürzer

- Optional 1 am Anfang oder 0 am Ende:  $(\epsilon+1)(01)^*(\epsilon+0)$

# BESTIMMUNG DER SEMANTIK VON $(\epsilon+1)(01)^*(\epsilon+0)$

$$\begin{aligned}
 & L((\epsilon+1)(01)^*(\epsilon+0)) \\
 = & L((\epsilon+1)) \circ L((01)^*) \circ L((\epsilon+0)) \\
 = & L(\epsilon) \cup L(1) \circ L((01))^* \circ L(\epsilon) \cup L(0) \\
 = & \{\epsilon\} \cup \{1\} \circ (L(0) \circ L(1))^* \circ \{\epsilon\} \cup \{0\} \\
 = & \{\epsilon, 1\} \circ \{01\}^* \circ \{\epsilon, 0\} \\
 = & \{\epsilon, 1\} \circ \{w \mid \exists n \in \mathbb{N}. w = \underbrace{01\dots 01}_{n\text{-mal}}\} \circ \{\epsilon, 0\} \\
 = & \{w \mid \exists n \in \mathbb{N}. w = \underbrace{01\dots 01}_{n\text{-mal}} \vee w = 1 \underbrace{01\dots 01}_{n\text{-mal}} \\
 & \vee w = \underbrace{01\dots 01}_{n\text{-mal}} 0 \vee w = 1 \underbrace{01\dots 01}_{n\text{-mal}} 0\}
 \end{aligned}$$

= Die Menge aller Wörter, in denen 0 und 1 abwechseln  
(Mühsamer Beweis durch Induktion)

# “RECHENREGELN” FÜR REGULÄRE AUSDRÜCKE

Wie zeigt man  $(01)^* + (10)^* + (01)^*0 + (10)^*1 \cong (\epsilon+1)(01)^*(\epsilon+0)$  ?

## • Definiere Äquivalenz von Ausdrücken

–  $E \cong F$ , falls  $L(E) = L(F)$

## Beweise algebraische Gesetze regulärer Ausdrücke

– Liefert Hilfsmittel zur Vereinfachung regulärer Ausdrücke

## • Gesetze für Einheiten und Annihilatoren

–  $\emptyset + E \cong E \cong E + \emptyset$ :  $L(\emptyset + E) = L(\emptyset) \cup L(E) = \emptyset \cup L(E) = L(E)$

–  $\epsilon \circ E \cong E \cong E \circ \epsilon$ :  $L(\epsilon \circ E) = L(\epsilon) \circ L(E) = \{\epsilon\} \circ L(E) = L(E)$

–  $\emptyset \circ E \cong \emptyset \cong E \circ \emptyset$ :  $L(\emptyset \circ E) = L(\emptyset) \circ L(E) = \emptyset \circ L(E) = \emptyset = L(\emptyset)$

## Kommutativitätsgesetz für +

–  $E + F \cong F + E$ :  $L(E + F) = L(E) \cup L(F) = L(F) \cup L(E) = L(F + E)$

– Kommutativität von  $\circ$  gilt nicht:  $= L(01) = \{01\} \neq \{10\} = L(10)$

# “RECHENREGELN” FÜR REGULÄRE AUSDRÜCKE II

## ● Gesetze für Assoziativität von $\circ$ und $+$

–  $(E \circ F) \circ G \cong E \circ (F \circ G)$ :

$$L((E \circ F) \circ G) = L(E \circ F) \circ L(G) = L(E) \circ L(F) \circ L(G) = L(E) \circ L(F \circ G) = L(E \circ (F \circ G))$$

–  $(E + F) + G \cong E + (F + G)$ :

$$L((E + F) + G) = L(E + F) \cup L(G) = L(E) \cup L(F) \cup L(G) = \dots = L(E + (F + G))$$

## ● Distributivgesetze

–  $(E + F) \circ G \cong E \circ G + F \circ G$ :

$$L((E + F) \circ G) = (L(E) \cup L(F)) \circ L(G)$$

$$= \{w \in \Sigma^* \mid \exists u \in L(E) \cup L(F). \exists v \in L(G). w = uv\}$$

$$= \{w \in \Sigma^* \mid \exists u \in L(E). \exists v \in L(G). w = uv\} \vee \{w \in \Sigma^* \mid \exists u \in L(F). \exists v \in L(G). w = uv\}$$

$$= \{w \in \Sigma^* \mid \exists u \in L(E). \exists v \in L(G). w = uv\} \cup \{w \in \Sigma^* \mid \exists u \in L(F). \exists v \in L(G). w = uv\}$$

$$= L(E) \circ L(G) \cup L(F) \circ L(G) = L(E \circ G + F \circ G)$$

–  $G \circ (E + F) \cong G \circ E + G \circ F$

## ● Idempotenz von $+$ : $E + E \cong E$

## ● Hüllengesetze:

$$\emptyset^* \cong \epsilon, \quad \epsilon^* \cong \epsilon, \quad (E^*)^* \cong E^*$$

$$E^+ \cong E \circ E^* \cong E^* \circ E, \quad E^* \cong \epsilon + E^+$$

# BEWEISMETHODIK FÜR WEITERE ÄQUIVALENZEN

- **Beispiel: Nachweis von  $(E+F)^* \cong (E^*F^*)^*$**

- Sei  $w \in L((E+F)^*)$

- Dann  $w = w_1..w_k$  mit  $w_i \in L(E)$  oder  $w_i \in L(F)$  für alle  $i$

- Dann  $w = w_1..w_k$  mit  $w_i \in L(E^*F^*)$  für alle  $i$  (semantisches Argument)

- Also  $w \in L((E^*F^*)^*)$  Umkehrung analog

- **Beweis verwendet keine Information über  $E$  und  $F$**

- Man könnte genauso gut  $(a+b)^* \cong (a^*b^*)^*$  testen

- $(E+F)^* \cong (E^*F^*)^*$  gilt, weil  $(a+b)^* \cong (a^*b^*)^*$  gilt

- **Allgemeines Beweisverfahren**

- $E$  regulärer Ausdruck mit Metavariablen  $E_1, \dots, E_m$  für Sprachen  $L_1, \dots, L_m$

- Ersetze im Beweis für  $E \cong F$  alle Metavariablen durch Symbole  $a \in \Sigma$

- Teste Äquivalenz der konkreten Ausdrücke mit automatischem Prüfverfahren

↪ Einheit 2.5

Korrektheitsbeweis: Induktion über Struktur regulärer Ausdrücke

## Sprachen regulärer Ausdrücke sind endlich erkennbar

Für jeden regulären Ausdruck  $E$  gibt es einen  $\epsilon$ -NEA  $A$  mit

- $A$  hat genau einen akzeptierenden Zustand  $q_f$
- Der Startzustand von  $A$  ist in keinem  $\delta_A(q, a)$  enthalten
- Für alle  $a \in \Sigma$  ist  $\delta_A(q_f, a) = \emptyset$
- $L(E) = L(A)$

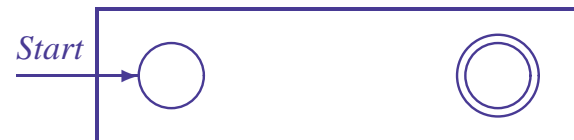
Beweis durch strukturelle Induktion über Aufbau regulärer Ausdrücke

### • Induktionsanfänge

– Für  $E = \epsilon$  wähle  $A =$



– Für  $E = \emptyset$  wähle  $A =$



– Für  $E = a$  wähle  $A =$



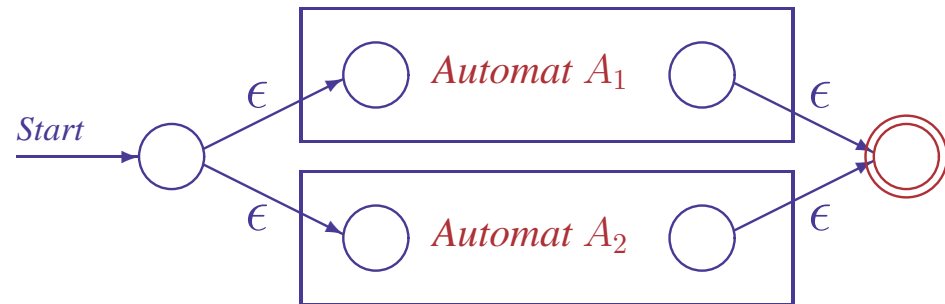
– **Korrektheit offensichtlich**, da jeweils maximal ein Zustandsübergang

# UMWANDLUNG REGULÄRER AUSDRÜCKE IN AUTOMATEN

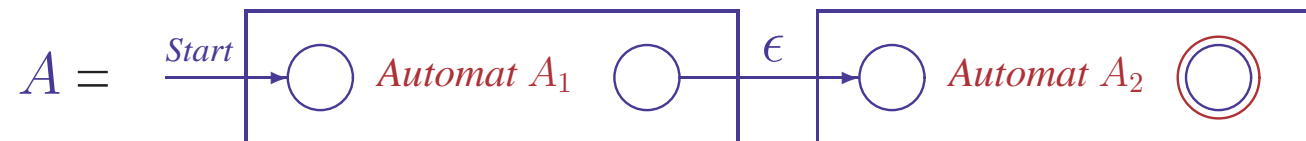
- **Induktionsannahme:** seien  $A_1$  und  $A_2$   $\epsilon$ -NEAs für  $E_1$  und  $E_2$

- **Induktionsschritt**

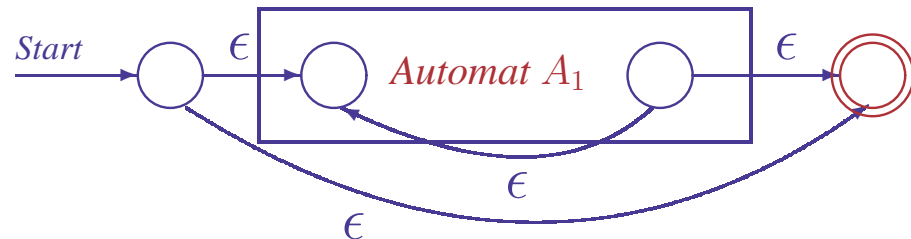
– Für  $E = E_1 + E_2$  wähle  $A =$



– Für  $E = E_1 \circ E_2$  wähle



– Für  $E = E_1^*$  wähle  $A =$



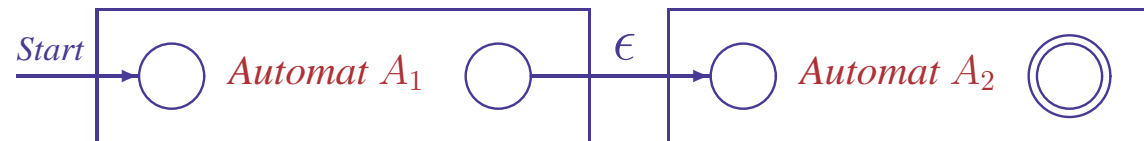
– Für  $E = (E_1)$  wähle  $A = A_1$

# KORREKTHEIT DER UMWANDLUNGEN

- **Klammern ändern nichts**

– Es ist  $L((E_1)) = L(E_1) = L(A_1) = L(A)$

- **Verkettung ist Verschaltung von Automaten**



Es gilt  $w \in L(E_1 \circ E_2)$

$\Rightarrow w \in L(E_1) \circ L(E_2) = L(A_1) \circ L(A_2)$

$\Rightarrow \exists u \in L(A_1). \exists v \in L(A_2). w = uv$

$\Rightarrow \exists u, v \in \Sigma^*. w = uv \wedge q_{f,1} \in \hat{\delta}_1(q_{0,1}, u) \wedge q_{f,2} \in \hat{\delta}_2(q_{0,2}, v)$

$\Rightarrow \exists u, v \in \Sigma^*. w = uv \wedge q_{0,2} \in \hat{\delta}(q_{0,1}, u) \wedge q_{f,2} \in \hat{\delta}(q_{0,2}, v)$

( $q_{0,2} \in \epsilon$ -Hülle( $q_{f,1}$ ))

$\Rightarrow q_{f,2} \in \hat{\delta}(q_{0,1}, w)$

(Definition  $\hat{\delta}$ )

$\Rightarrow w \in L(A)$

Argument ist umkehrbar, also  $w \in L(A) \Rightarrow w \in L(E_1 \circ E_2)$

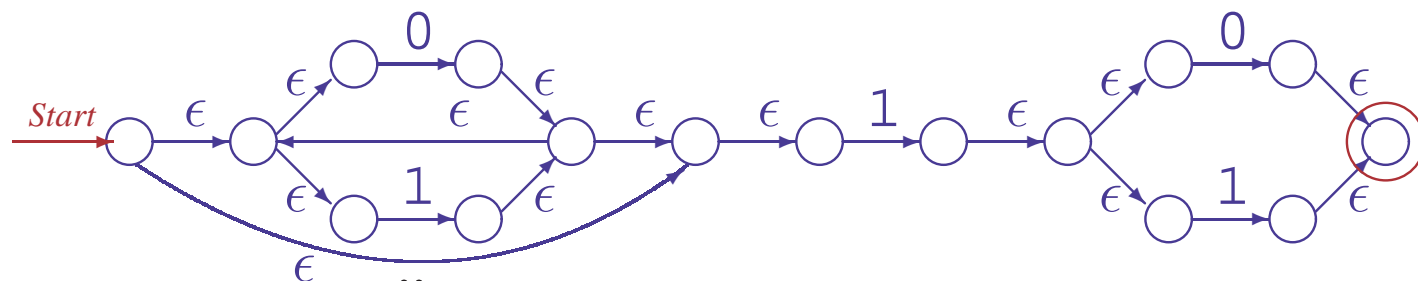
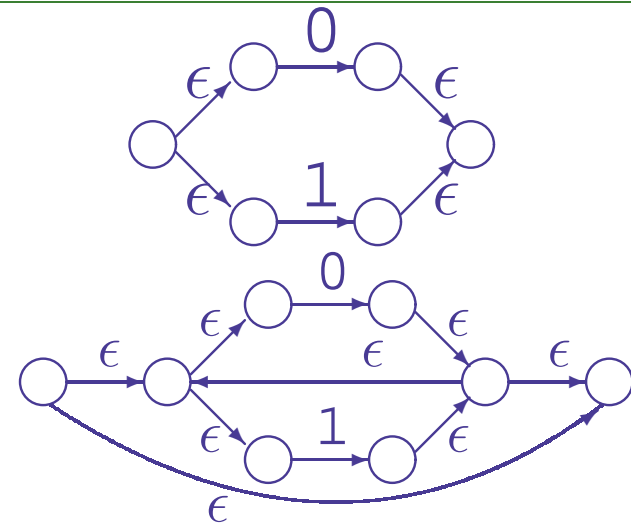
- **Sternbildung und Vereinigung ähnlich**



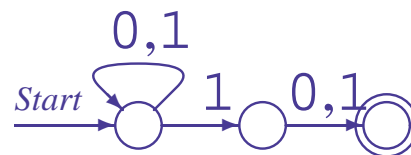
# UMWANDLUNG REGULÄRER AUSDRÜCKE AM BEISPIEL

**Konstruiere endlichen Automaten für  $(0+1)^*1(0+1)$**

- **Teilautomat für  $(0+1)$**
- **Teilautomat für  $(0+1)^*$**
- **Automat für  $(0+1)^*1(0+1)$**



- **Elimination von  $\epsilon$ -Übergängen**



# UMWANDLUNG VON ( $\epsilon$ -)NEAS IN REGULÄRE AUSDRÜCKE

## ● Ursprünglich: Pfadanalyse im Übergangsdiagramm

- Spezialisierung eines allgemeinen Verfahrens für Pfadanalyse in Graphen
- Definiere reguläre Ausdrücke für Pfade durch Automaten
- Berechnung Ausdrücke iterativ und kombiniere alle relevanten Ausdrücke
- **Kompliziertes und aufwendiges Verfahren**

Mehr dazu im Anhang

## ● Effizienterer Zugang: Elimination von Zuständen

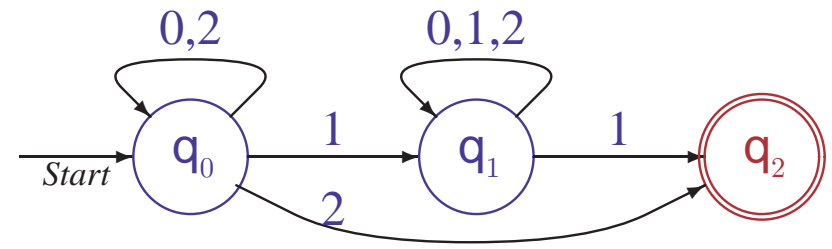
- Beschreibe Übergänge  $q_i \xrightarrow{a \in \Sigma} q_j$  durch reguläre Ausdrücke
- Beginne mit regulären Ausdrücken für direkte Übergänge
- Entferne einzelne Zustände und beschreibe die entstehenden Ausdrücke
- Liefert Ausdrücke für Übergänge zwischen Start- und Endzuständen

## ● Technisches Hilfsmittel: verallgemeinerte NEAs (**VNEAs**)

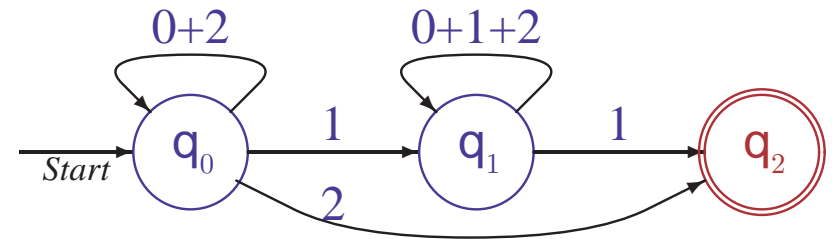
- NEA, dessen Überföhrungsfunktion  $\delta$  auf regulären Ausdrücken arbeitet
- **A akzeptiert**  $w$ , wenn es einen Pfad  $w = v_1..v_m$  von  $q_0$  zu einem  $q \in F$  gibt und alle  $v_i$  in der Sprache des entsprechenden regulären Ausdrucks liegen
- **Konsistente Formalisierung mühsam und ohne Erkenntnisgewinn**

# ZUSTANDSELIMINATION IN VNEAs

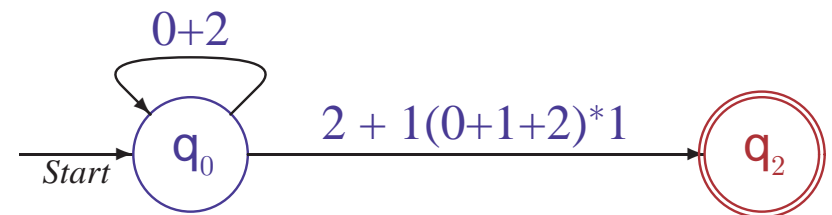
- **Ursprünglicher NEA**



- **Zugehöriger VNEA**

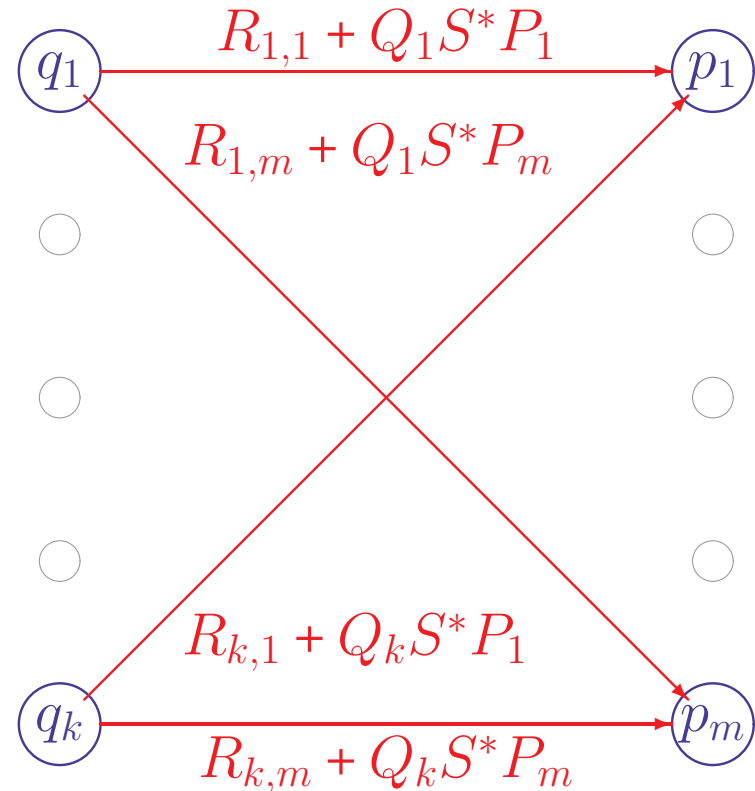
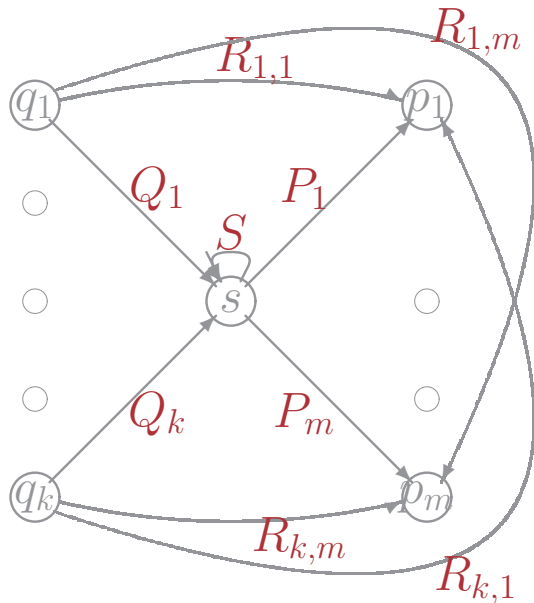


- **Nach Elimination von  $q_1$**



- Ausdruck für Übergang von  $q_0$  nach  $q_2$  ergibt sich aus Übergang  $q_0$  nach  $q_1$ , Schleife bei  $q_1$ , Übergang  $q_1$  nach  $q_2$  und existierendem Ausdruck für direkten Übergang von  $q_0$  nach  $q_2$

# ALLGEMEINE ZUSTANDSELIMINATION IN VNEAs



**Eliminiere Zustand  $s$**   
**mit Vorgängern  $q_1, \dots, q_k$**   
**und Nachfolgern  $p_1, \dots, p_m$**

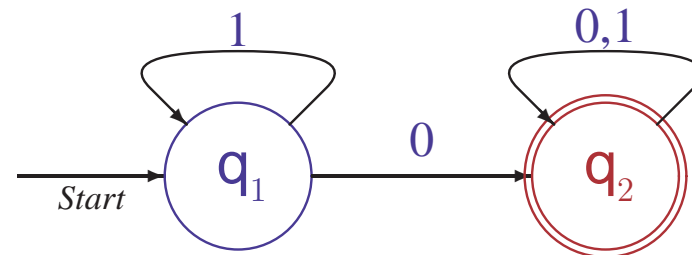
jedes  $p_i$  kann auch einer der Zustände  $q_1, \dots, q_k$  sein

- Eliminiere Pfad von  $q_1$  nach  $p_1$  über  $s$ :  $R_{1,1} + Q_1 S^* P_1$
- Eliminiere Pfad von  $q_1$  nach  $p_m$  über  $s$ :  $R_{1,m} + Q_1 S^* P_m$
- Eliminiere Pfad von  $q_k$  nach  $p_1$  über  $s$ :  $R_{k,1} + Q_k S^* P_1$
- Eliminiere Pfad von  $q_k$  nach  $p_m$  über  $s$ :  $R_{k,m} + Q_k S^* P_m$

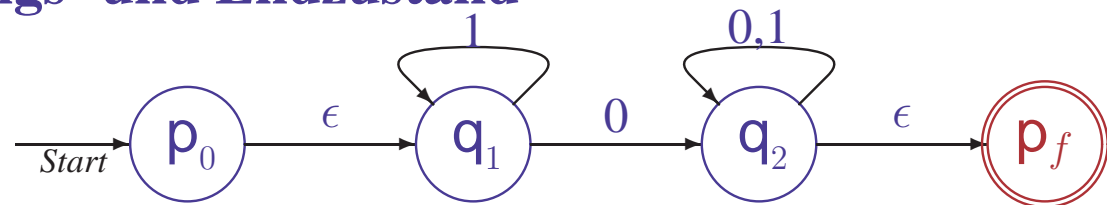
# UMWANDLUNG DURCH ZUSTANDSELIMINATION

- 1. Ergänze neuen Anfangszustand  $p_0$  und Endzustand  $p_f$** 
  - $\epsilon$ -Übergang von neuem zu altem Anfangszustand
  - $\epsilon$ -Übergang von allen alten Endzuständen zum neuen Endzustand
  - ↳ Anfangs- und Endzustand haben keine ein- bzw. ausgehenden Kanten
- 2. Transformiere endlichen Automaten in VNEA**
  - Ersetze Beschriftungen mit Symbolen  $a \in \Sigma$  durch reguläre Ausdrücke
- 3. Eliminiere alle Zustände außer  $p_0$  und  $p_f$** 
  - Iterative Anwendung des Eliminationsverfahrens
  - Vereinfache entstehende Ausdrücke nach jedem Schritt
- 4. Extrahiere regulären Ausdruck aus VNEA**
  - Zielausdruck ist Beschriftung der Kante von  $p_0$  nach  $p_f$

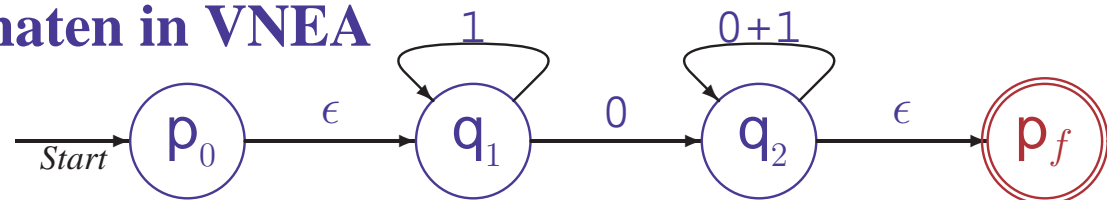
# UMWANDLUNG NEA $\rightarrow$ RA: EINFACHES BEISPIEL



## 1. Ergänze neuen Anfangs- und Endzustand

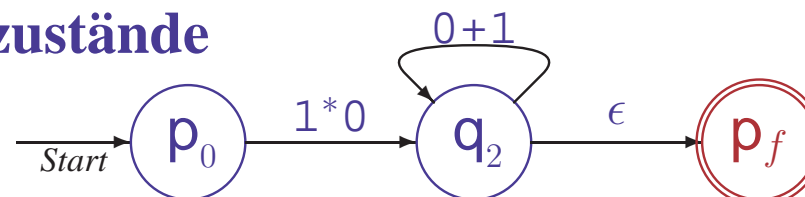


## 2. Transformiere Automaten in VNEA

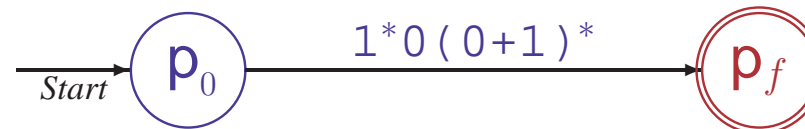


## 3. Eliminiere Zwischenzustände

– Eliminiere  $q_1$



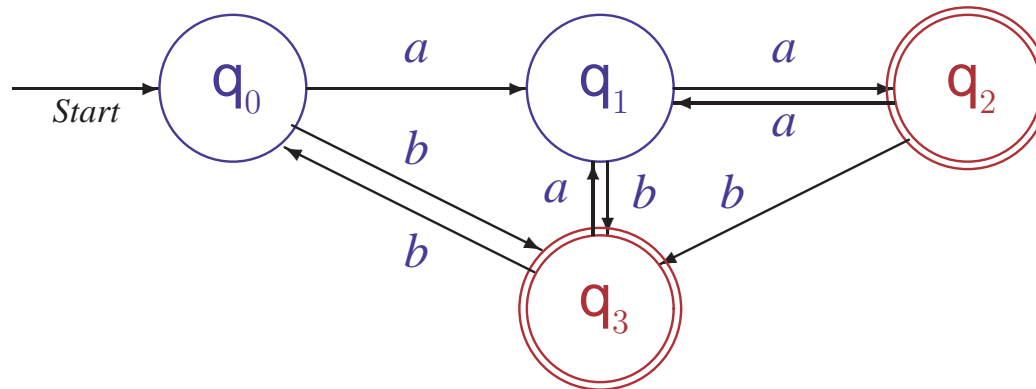
– Eliminiere  $q_2$



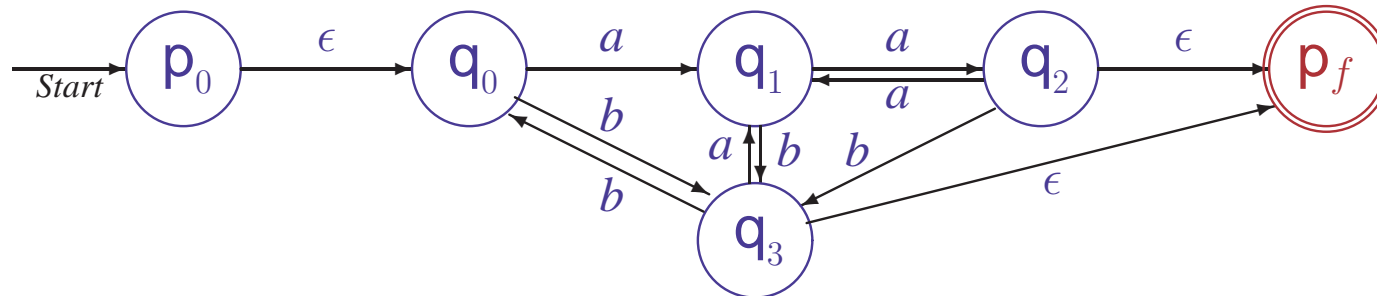
## 4. Extrahierter Ausdruck ist $1^*0(0+1)^*$

Umwandlung mit Pfadanalyseverfahren würde 12 aufwendige Schritte erfordern

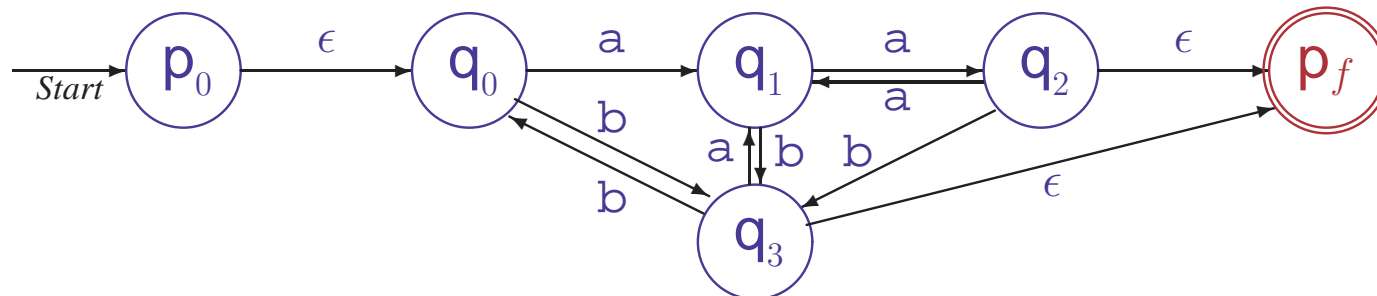
# UMWANDLUNG NEA $\rightarrow$ RA AM BEISPIEL



## 1. Ergänze neuen Anfangs- und Endzustand

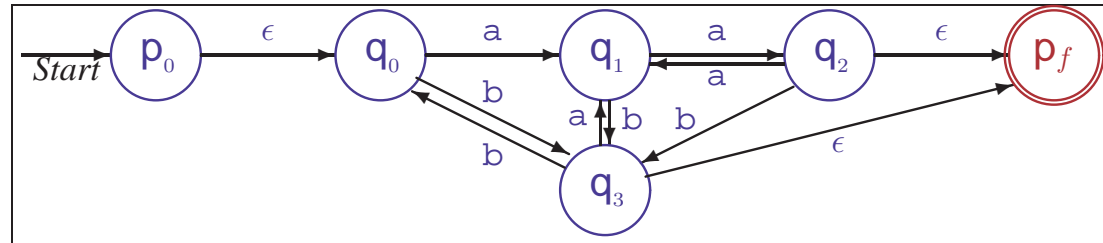


## 2. Transformiere Automaten in VNEA

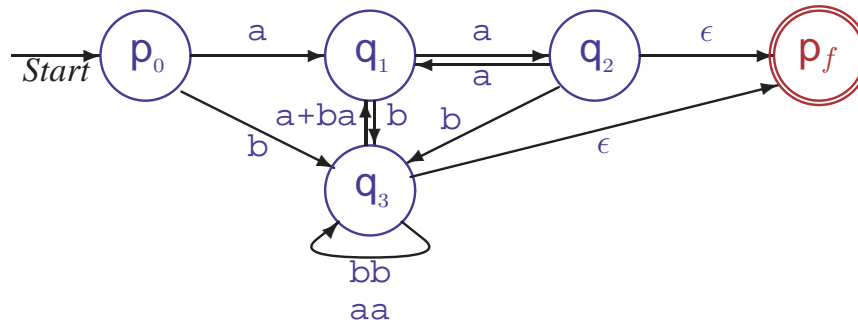


# UMWANDLUNG NEA → RA AM BEISPIEL (II)

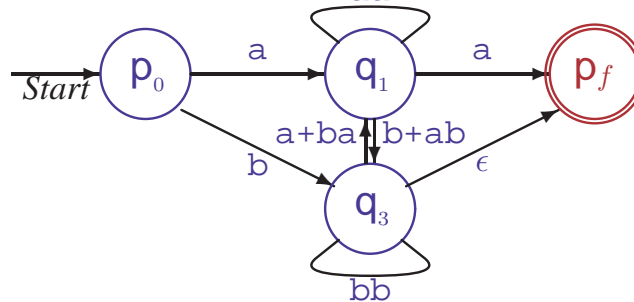
1. Ergänze neuen Anfangs- und Endzustand
2. Transformiere Automaten in VNEA
3. **Eliminiere Zwischenzustände**



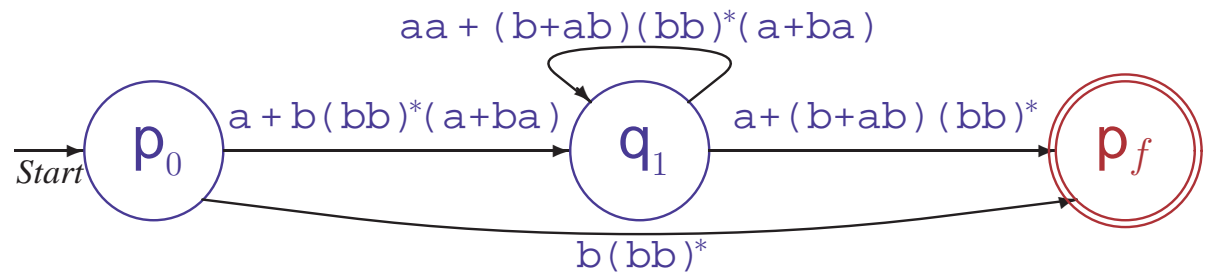
– Eliminiere  $q_0$



– Eliminiere  $q_2$



– Eliminiere  $q_3$



4. **Extrahierter Ausdruck ergibt sich nach Elimination von  $q_1$**

–  $b(bb)^* + (a + b(bb)^*(a+ba)) (aa + (b+ab)(bb)^*(a+ba))^* (a + (b+ab)(bb)^*)$



- **Algebraische Notation für Sprachen**
  - $\epsilon$ ,  $\emptyset$ , Symbole des Alphabets, Vereinigung, Verkettung, Sternoperator
  - Äquivalent zu endlichen Automaten
  - Gut zum **Nachweis algebraischer Gesetze** von Sprachen
  - Anwendung in Programmiersprachen und Suchmaschinen
- **Transformation in endliche Automaten**
  - Iterative Konstruktion von  $\epsilon$ -NEAs
  - Nachträgliche Optimierung durch Elimination von  $\epsilon$ -Übergängen
- **Transformation von Automaten in Ausdrücke**
  - Konstruktion durch **Elimination von Zuständen** in VNEAs
  - Historisch: Konstruktion von Ausdrücken für Abarbeitungspfade
  - Nachträgliche Optimierungen durch Anwendung algebraischer Gesetze

# ANHANG

## Originalmethode: allgemeines Graphanalyseverfahren

- Gegeben DEA  $A = (\{q_1, \dots, q_n\}, \Sigma, \delta, q_1, \{q_{f_1}, \dots, q_{f_m}\})$
- Definiere Ausdrücke für Pfade durch  $A$ 
  - $R_{ij}^k$ : Regulärer Ausdruck für Menge der Wörter  $w$  mit  $\hat{\delta}(q_i, w) = q_j$ ,  
so dass für alle  $v \neq w$  gilt:  $\hat{\delta}(q_i, v) = q_m \Rightarrow m \leq k$   
(Abarbeitung von  $w$  berührt keinen Zustand größer als  $k$ )
- Setze die  $R_{ij}^k$  zu Ausdruck für  $L(A)$  zusammen
  - Per Definition ist  $R_{ij}^n$  ein Ausdruck für Wörter  $w$  mit  $\hat{\delta}(q_i, w) = q_j$
  - Setze  $R = R_{1f_1}^n + \dots + R_{1f_m}^n$
  - Dann gilt  $L(R) = \bigcup_{j=1}^m \{w \in \Sigma^* \mid \hat{\delta}(q_1, w) = q_{f_j}\}$   
 $= \{w \in \Sigma^* \mid \exists q \in \{q_{f_1}, \dots, q_{f_m}\}. \hat{\delta}(q_1, w) = q\} = L(A)$

# ITERATIVE BESTIMMUNG DER AUSDRÜCKE $R_{ij}^k$

- **Basisfall  $R_{ij}^0$ :** Pfad darf zwischendurch keine Zustände berühren

- Pfadlänge 0 (nur für  $i=j$ ):  $\epsilon \in L(R_{ii}^0)$

- Pfadlänge 1:  $\{a \in \Sigma \mid \delta(q_i, a) = q_j\} \subseteq L(R_{ij}^0)$

- Ergebnis:  $R_{ii}^0 = \epsilon + a_1 + \dots + a_k$ , wobei  $\{a_1, \dots, a_k\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$

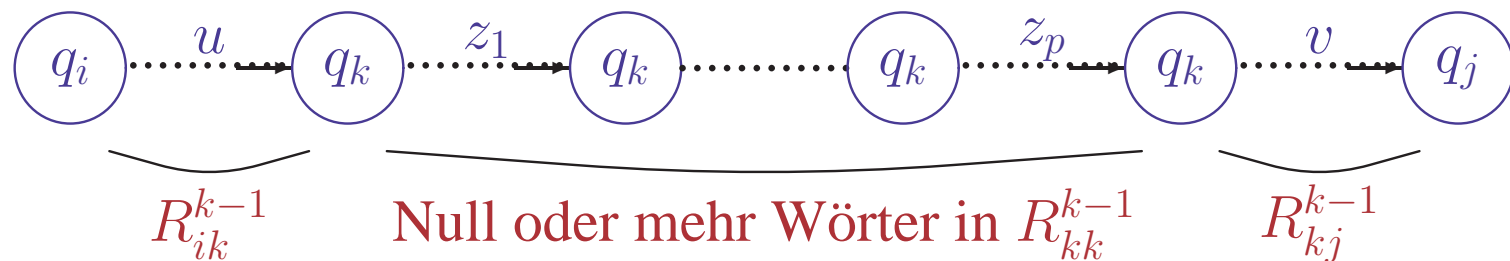
$$R_{ij}^0 = \emptyset + a_1 + \dots + a_k \quad (i \neq j)$$

- **Schrittfall  $R_{ij}^k$  ( $0 < k \leq n$ ):** zwei Alternativen

- Wörter  $w \in L(R_{ij}^k)$ , deren Pfad  $q_k$  nicht enthält, gehören zu  $L(R_{ij}^{k-1})$

- Wörter  $w \in L(R_{ij}^k)$ , deren Pfad  $q_k$  enthält:

Zerlege  $w$  in  $uz_1 \dots z_p v$  mit  $\hat{\delta}(q_i, u) = q_k \wedge \forall l \leq p. \hat{\delta}(q_k, z_l) = q_k \wedge \hat{\delta}(q_k, v) = q_j$



- Ergebnis:  $R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1} \circ (R_{kk}^{k-1})^* \circ R_{kj}^{k-1}$

# UMWANDLUNG VON AUTOMATEN AM BEISPIEL

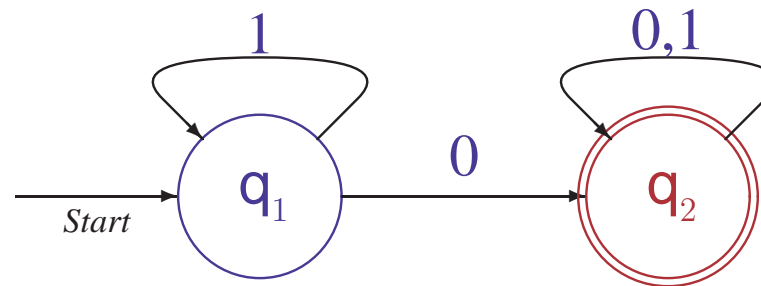
## • Basisfall

$$R_{11}^0 = \epsilon + 1$$

$$R_{12}^0 = 0$$

$$R_{21}^0 = \emptyset$$

$$R_{22}^0 = \epsilon + 0 + 1$$



## • Stufe 1

$$R_{11}^1 = R_{11}^0 + R_{11}^0 (R_{11}^0)^* R_{11}^0 = \epsilon + 1 + (\epsilon + 1) (\epsilon + 1)^* (\epsilon + 1) \mapsto 1^*$$

$$R_{12}^1 = R_{12}^0 + R_{11}^0 (R_{11}^0)^* R_{12}^0 = 0 + (\epsilon + 1) (\epsilon + 1)^* 0 \mapsto 1^* 0$$

$$R_{21}^1 = R_{21}^0 + R_{21}^0 (R_{11}^0)^* R_{11}^0 = \emptyset + \emptyset (\epsilon + 1)^* (\epsilon + 1) \mapsto \emptyset$$

$$R_{22}^1 = R_{22}^0 + R_{21}^0 (R_{11}^0)^* R_{12}^0 = \epsilon + 0 + 1 + \emptyset (\epsilon + 1)^* 0 \mapsto \epsilon + 0 + 1$$

## • Stufe 2

**Gebraucht wird nur  $R_{12}^2$**

$$R_{11}^2 = R_{11}^1 + R_{12}^1 (R_{22}^1)^* R_{21}^1 = 1^* + 1^* 0 (\epsilon + 0 + 1)^* \emptyset \mapsto 1^*$$

$$R_{12}^2 = R_{12}^1 + R_{12}^1 (R_{22}^1)^* R_{12}^1 = 1^* 0 + 1^* 0 (\epsilon + 0 + 1)^* (\epsilon + 0 + 1) \mapsto 1^* 0 (0 + 1)^*$$

$$R_{21}^2 = R_{21}^1 + R_{22}^1 (R_{22}^1)^* R_{21}^1 = \emptyset + (\epsilon + 0 + 1) (\epsilon + 0 + 1)^* \emptyset \mapsto \emptyset$$

$$R_{22}^2 = R_{22}^1 + R_{22}^1 (R_{22}^1)^* R_{22}^1 = (\epsilon + 0 + 1) + (\epsilon + 0 + 1) (\epsilon + 0 + 1)^* (\epsilon + 0 + 1) \mapsto (0 + 1)^*$$

**Regulärer Ausdruck des Automaten ist:  $1^* 0 (0 + 1)^*$**

# DAS PFADANALYSEVERFAHREN IST ZU KOMPLIZIERT

- **Konstruktion aller  $R_{ij}^k$  ist aufwendig**
  - Es müssen mehr als  $n^3$  Ausdrücke  $R_{ij}^k$  erzeugt werden
  - Ausdrücke  $R_{ij}^k$  können viermal so groß wie die  $R_{ij}^{k-1}$  werden
  - Ohne Vereinfachung der  $R_{ij}^k$  sind bis zu  $n^3 * 4^n$  Symbole zu erzeugen
- **Optimierungen des Verfahrens sind möglich**
  - Vermeide Vielfachkopien der  $R_{ij}^{k-1}$
  - Vereinfache Ausdrücke  $R_{ij}^k$  direkt nach Erzeugung
  - Liefert keine grundsätzliche Verbesserung

**Zustandselimination ist erheblich effizienter**