# Theoretische Informatik I

## Einheit 2.3

## Reguläre Ausdrücke

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

# Eine algebraische Beschreibung für Sprachen

- **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"

# Anwendung: Lexikalische Analyse

## 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 als Suchmuster in Unix

grep *searches files for lines containing a match to a given pattern*

- A regular expression is a pattern that describes a set of strings. Regular expressions are constructed by using various operators to combine smaller expressions.
- Fundamental building blocks are expressions that match a single character.
- A bracket expression is a list of characters enclosed by `[` and `]`. It matches any single character in that list. For example, `[0123456789]` matches any single digit.
- Within a bracket expression, a range expression consists of two characters separated by a hyphen. It matches any single character that sorts between the two characters.
  For example, in the default C locale, `[a-d]` is equivalent to `[abcd]`.
- Certain named classes of characters are predefined within bracket expressions.
  They are `[:alnum:]`, `[:alpha:]`, `[:cntrl:]`, `[:digit:]`, ...
- The period `.` matches any single character.
- The caret `^` and the dollar sign `$` are metacharacters that match the empty string ...
- A regular expression may be followed by one of several repetition operators:
  `?`: The preceding item is optional and matched at most once.
  `*`: The preceding item will be matched zero or more times.
  `+` The preceding item will be matched one or more times.
- Two regular expressions may be concatenated; the resulting regular expression matches any string concatenating two substrings that match the subexpressions.
- Two regular expressions may be joined by the infix operator `|`
  The resulting regular expression matches any string matching either subexpression.

# Reguläre Ausdrücke beschreiben Sprachen

- **Ausdrücke für einfache Basissprachen**
  - Leere Sprache – Sprache ohne Elemente
  - Einelementige Sprachen: Sprache, die nur das leere Wort enthält
    Sprache, die nur ein Symbol $a \in \Sigma$ enthält

- **Ausdrücke für Komposition von Sprachen**
  - Vereinigung zweier Sprachen

    $$\boldsymbol{L \cup M} = \{w \in \Sigma^* \mid w \in L \ \vee \ w \in M\}$$
  - Verkettung der Wörter zweier Sprachen

    $$\boldsymbol{L \circ M} = \{w \in \Sigma^* \mid \exists u \in L.\ \exists v \in M.\ w = uv\}$$
  - Verkettung einer festen Anzahl von Wörtern einer Sprache

    $$\boldsymbol{L^n} = \{w \in \Sigma^* \mid \exists u_1, .., u_n \in L.\ w = u_1..u_n\}\ (n \in \mathbb{N}) \quad (L^0 = \{\epsilon\})$$
  - (Kleene'sche) Hülle: Verkettung beliebig vieler Wörter einer Sprache

    $$\boldsymbol{L^*} = \bigcup_{n \geq 0} L^n = \{w \in \Sigma^* \mid \exists n \in \mathbb{N}.\ \exists u_1, .., u_n \in L.\ w = u_1..u_n\}$$

**Mehr Ausdrücke möglich, aber nicht erforderlich**

# Reguläre Ausdrücke präzisiert

- **Syntax:** **Terme über $\Sigma \cup \{\emptyset, \epsilon, +, \circ, *, (, )\}$**
  - $\emptyset$, $\epsilon$, und $\mathtt{a}$ (für alle $a \in \Sigma$) sind reguläre Ausdrücke
  - Sind $E$ und $F$ reguläre Ausdrücke, dann auch $E \circ F$, $E^*$, $E + F$ und $(E)$
- **Semantik: Sprachen über $\Sigma$**

  Die Sprache $L(E)$ des regulären Ausdrucks $E$, ist induktiv definiert
  - $L(\emptyset) = \emptyset, \quad L(\epsilon) = \{\epsilon\}, \quad L(\mathtt{a}) = \{a\}$ (für alle $a \in \Sigma$)
  - $L(E \circ F) = L(E) \circ L(F) \qquad = \{v\,w \mid v \in L(E) \wedge w \in L(F)\}$,

    $L(E^*) \quad = (L(E))^* \qquad = \{w_1 w_2 .. w_n \mid n \in \mathbb{N} \wedge w_i \in L(E)\}$,

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

    $L((E)) \;= L(E)$
- **Konventionen**
  - $E \circ F$ wird üblicherweise als $EF$ abgekürzt
  - Definitorische Abkürzungen: $E^+ \equiv EE^*$, $[a_1 \ldots a_n] \equiv a_1 + \ldots + a_n$
  - Prioritätsregelungen ermöglichen es, überflüssige Klammern wegzulassen
    - $*$ ("Sternoperator") bindet stärker als $\circ$, und dies stärker als $+$
    - Verkettung $\circ$ und Vereinigung $+$ sind assoziativ

# Entwicklung regulärer Ausdrücke

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

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

– $\mathtt{0}$ repräsentiert $\{0\}$, $\mathtt{1}$ repräsentiert $\{1\}$

– Also ist $L(\mathtt{01}) = L(\mathtt{0}) \circ L(\mathtt{1}) = \{0\} \circ \{1\} = \{01\}$

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

– $L((\mathtt{01})^*) = L(\mathtt{01})^* = \{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\}$

## 3. Manche Wörter nicht erfaßt

– Start mit Eins statt Null: $(\mathtt{10})^*$

– Start und Ende mit Null: $(\mathtt{01})^*\mathtt{0}$

– Start und Ende mit Eins: $(\mathtt{10})^*\mathtt{1}$

Vollständiger Ausdruck: $(\mathtt{01})^* + (\mathtt{10})^* + (\mathtt{01})^*\mathtt{0} + (\mathtt{10})^*\mathtt{1}$

## 4. Es geht auch kürzer

– Optional $1$ am Anfang oder $0$ am Ende: $(\epsilon+\mathtt{1})(\mathtt{01})^*(\epsilon+\mathtt{0})$

# Bestimmung der Semantik von $(\epsilon+1)(01)^*(\epsilon+0)$

$$
\begin{aligned}
& L((\epsilon+1)(01)^*(\epsilon+0)) \\
= {} & L((\epsilon+1)) \circ L((01)^*) \circ L((\epsilon+0)) \\
= {} & L(\epsilon) \cup L(1) \circ L((01))^* \circ L(\epsilon) \cup L(0) \\
= {} & \{\epsilon\} \cup \{1\} \circ (L(0) \circ L(1))^* \circ \{\epsilon\} \cup \{0\} \\
= {} & \{\epsilon, 1\} \circ \{01\}^* \circ \{\epsilon, 0\} \\
= {} & \{\epsilon, 1\} \circ \{w \mid \exists n \in \mathbb{N}.\ w = \underbrace{01\ldots01}_{n-mal}\} \circ \{\epsilon, 0\} \\
= {} & \{w \mid \exists n \in \mathbb{N}.\ w = \underbrace{01\ldots01}_{n-mal} \vee w = 1\underbrace{01\ldots01}_{n-mal} \\
& \qquad \vee w = \underbrace{01\ldots01}_{n-mal}0 \vee w = 1\underbrace{01\ldots01}_{n-mal}0\ \}
\end{aligned}
$$

= Die Menge aller Wörter, in denen 0 und 1 abwechseln

(Mühsamer Beweis durch Induktion)

# 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

# "Rechenregeln" für reguläre Ausdrücke

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

- **Definiere Äquivalenz von Ausdrücken**
  - $\boldsymbol{E} \cong \boldsymbol{F}$, falls $L(E) = L(F)$
- **Beweise algebraische Gesetze regulärer Ausdrücke**
  - Liefert Hilfsmittel zur Vereinfachung regulärer Ausdrücke
- **Einheiten und Annihilatoren**
  - $\emptyset+\boldsymbol{E} \cong \boldsymbol{E} \cong \boldsymbol{E}+\emptyset$: $L(\emptyset+E) = L(\emptyset)\cup L(E) = \emptyset\cup L(E) = L(E)$
  - $\epsilon\circ\boldsymbol{E} \cong \boldsymbol{E} \cong \boldsymbol{E}\circ\epsilon$: $L(\epsilon\circ E) = L(\epsilon)\circ L(E) = \{\epsilon\}\circ L(E) = L(E)$
  - $\emptyset\circ\boldsymbol{E} \cong \emptyset \cong \boldsymbol{E}\circ\emptyset$: $L(\emptyset\circ E) = L(\emptyset)\circ L(E) = \emptyset\circ L(E) = \emptyset = L(\emptyset)$
- **Kommutativität von +**
  - $\boldsymbol{E}+\boldsymbol{F} \cong \boldsymbol{F}+\boldsymbol{E}$: $L(E+F) = L(E)\cup L(F) = L(F)\cup L(E) = L(F+E)$
  - Kommutativität von $\circ$ gilt nicht: $= L(01) = \{01\} \neq \{10\} = L(10)$

# "Rechenregeln" für reguläre Ausdrücke II

- **Assoziativität von $\circ$ und $+$**

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

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

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

  – $L((E+F)+G) = L(E+F) \cup L(G) = L(E) \cup L(F) \cup L(G) = \ldots = L(E+(F+G))$

- **Distributivgesetze**

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

  $$\begin{aligned} & L((E+F) \circ G) = (L(E) \cup L(F)) \circ L(G) \\ & = \{w \in \Sigma^* \mid \exists u \in L(E) \cup L(F). \exists v \in L(G). w = uv\} \\ & = \{w \in \Sigma^* \mid \exists u \in L(E). \exists v \in L(G). w = uv \vee \exists u \in L(F). \exists v \in L(G). w = uv\} \\ & = \{w \in \Sigma^* \mid \exists u \in L(E). \exists v \in L(G). w = uv\} \cup \{w \in \Sigma^* \mid \exists u \in L(F). \exists v \in L(G). w = uv\} \\ & = L(E) \circ L(G) \cup L(F) \circ L(G) = L(E \circ G + F \circ G) \end{aligned}$$

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

- **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 $w \in L((E^*F^*)^*)$
- **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 $E_1,..,E_m$ für Sprachen $L_1,..,L_m$
  - Ersetze im Beweis für $E \cong F$ alle Metavariablen durch Symbole $a \in \Sigma$
  - Teste Äquivalenz der konkreten Ausdrücke mit automatischem Prüfverfahren $\mapsto$ Einheit 2.5

Korrektheitsbeweis: Induktion über Struktur regulärer Ausdrücke

# Umwandlung regulärer Ausdrücke in Automaten

## 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 $A =$

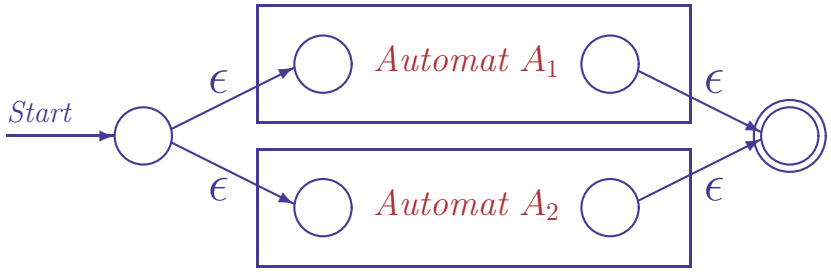
– Für $E = \emptyset$ wähle $A =$

– Für $E =$ a wähle $A =$

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

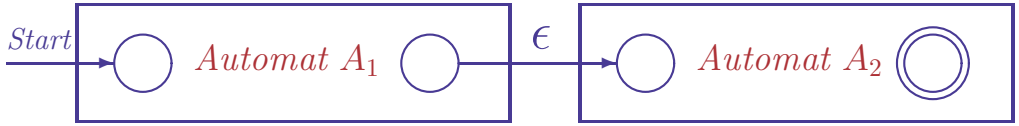
# Umwandlung regulärer Ausdrücke in Automaten

- **Induktionsannahme: seien $A_1$ und $A_2$ $\epsilon$-NEAs für $E_1$ und $E_2$**
- **Induktionsschritt**
  - Für $E = E_1 + E_2$ wähle $A =$

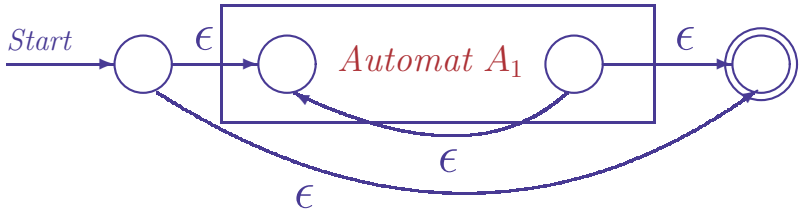



  - Für $E = E_1 \circ E_2$ wähle

$A =$



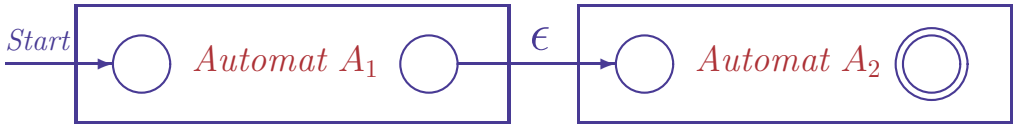

  - Für $E = E_1^*$ wähle $A =$




  - 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 $w \in L(E_1 \circ E_2)$

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

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

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

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

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

$\Rightarrow\ w \in L(A)$

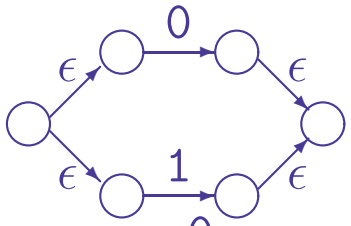
Argument ist umkehrbar, also $w \in L(A) \Rightarrow w \in L(E_1 \circ E_2)$

- **Sternbildung und Vereinigung ähnlich**

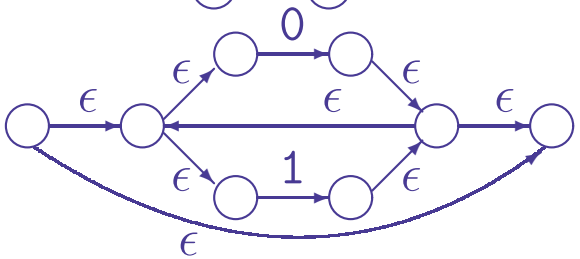
# Umwandlung regulärer Ausdrücke am Beispiel

## Konstruiere endlichen Automaten für `(0+1)*1(0+1)`

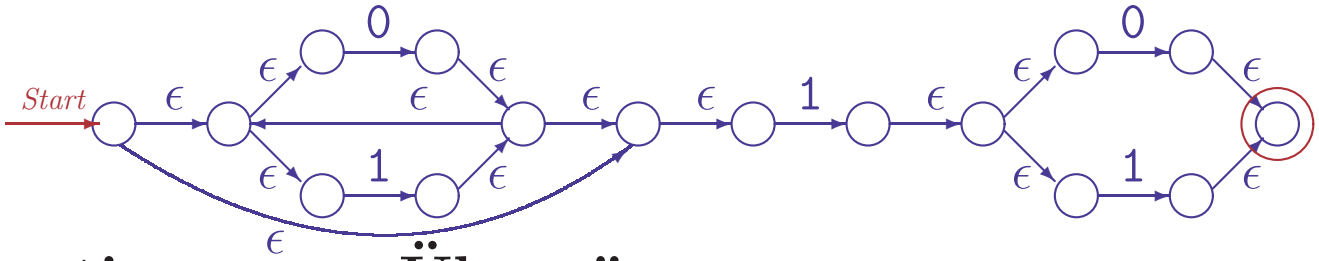
- **Teilautomat für** `(0+1)`



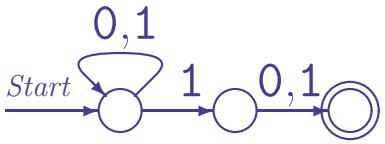

- **Teilautomat für** `(0+1)*`




- **Automat für** `(0+1)*1(0+1)`




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



## Originalmethode: allgemeines Graphanalyseverfahren

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

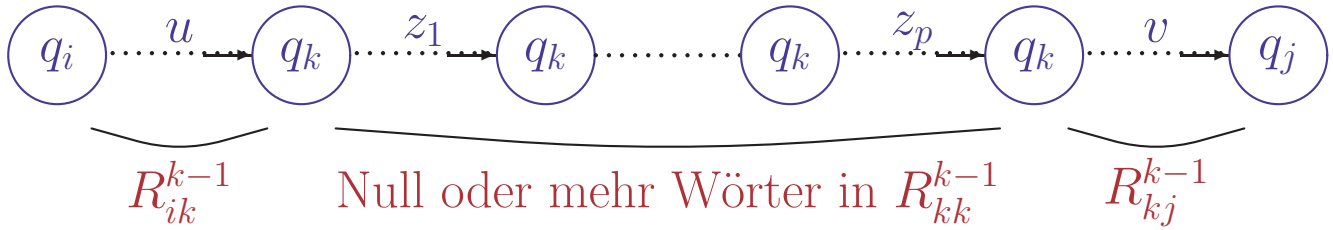
# Iterative Bestimmung der Ausdrücke $R_{ij}^k$

- **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: $R_{ii}^0 = \epsilon + \mathsf{a}_1 + \ldots + \mathsf{a}_k$, wobei $\{a_1, .., a_k\} = \{a \in \Sigma \mid \delta(q_i, a){=}q_j\}$

    $R_{ij}^0 = \emptyset + \mathsf{a}_1 + \ldots + \mathsf{a}_k$ $(i{\neq}j)$

- **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 $uz_1..z_pv$ mit $\hat{\delta}(q_i, u){=}q_k \wedge \forall l{\leq}p.\hat{\delta}(q_k, z_l){=}q_k \wedge \hat{\delta}(q_k, v){=}q_j$

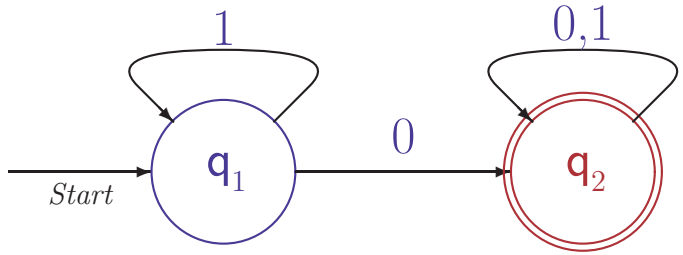



  - Ergebnis: $R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1} \circ (R_{kk}^{k-1})^* \circ R_{kj}^{k-1}$

# Umwandlung von Automaten am Beispiel

- **Basisfall**

$R^0_{11} = \epsilon + 1$
$R^0_{12} = 0$
$R^0_{21} = \emptyset$
$R^0_{22} = \epsilon + 0 + 1$




- **Stufe 1**

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

- **Stufe 2** **Gebraucht wird nur $R^2_{12}$**

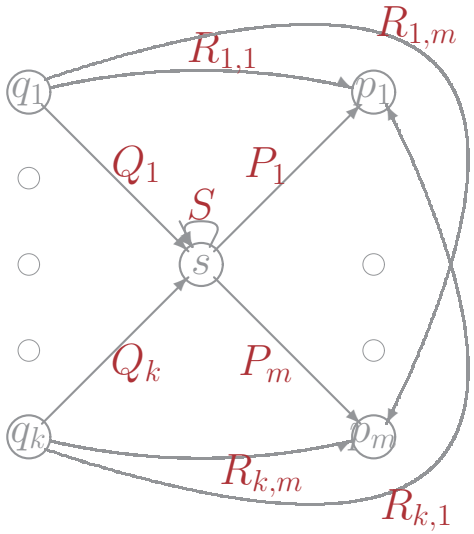
$R^2_{11} = R^1_{11} + R^1_{12}(R^1_{22})^* R^1_{21} = 1^* + 1^*0(\epsilon+0+1)^*\emptyset \mapsto 1^*$
$R^2_{12} = R^1_{12} + R^1_{12}(R^1_{22})^* R^1_{22} = 1^*0 + 1^*0(\epsilon+0+1)^*(\epsilon+0+1) \mapsto 1^*0(0+1)^*$
$R^2_{21} = R^1_{21} + R^1_{22}(R^1_{22})^* R^1_{21} = \emptyset + (\epsilon+0+1)(\epsilon+0+1)^*\emptyset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

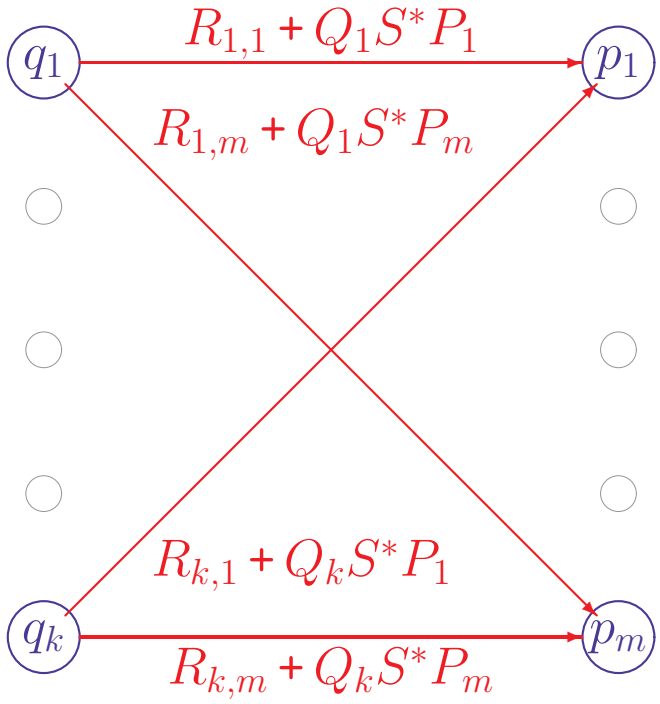
**Regulärer Ausdruck des Automaten: $1^*0(0+1)^*$**

# Eine effizientere Umwandlungsmethode

- **Direkte Umwandlung ist kompliziert und 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
  - Optimierung des Verfahrens: vermeide Vielfachkopien der $R_{ij}^{k-1}$
- **Effizienterer Zugang: Elimination von Zuständen**
  - Statt Pfade zu verlängern, lege Zustände des Automaten zusammen
  - Ersetze Übergänge $q_i \xrightarrow{a \in \Sigma} q_j$ durch Übergänge mit regulären Ausdrücken
  - Schrittweise Umwandlung erzeugt regulären Ausdruck des Automaten
- **Technisches Hilfsmittel: RA-Automaten**
  - Überführungsfunktion $\delta$ arbeitet auf regulären Ausdrücken
  - $A$ 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 RA-Automaten




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

– 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 $s$: $R_{k,m} + Q_k S^* P_m$

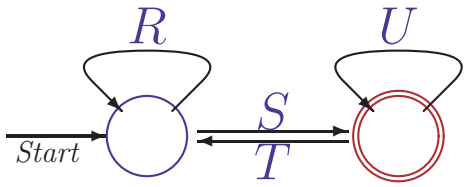
# Umwandlung durch Zustandselimination

1. **Transformiere Automaten in RA-Automaten**
   – Ersetze Beschriftungen mit Symbolen $a \in \Sigma$ durch reguläre Ausdrücke

2. **Für $q \in F$ eliminiere alle Zustände außer $q_0$ und $q$**
   – Iterative Anwendung des Eliminationsverfahrens

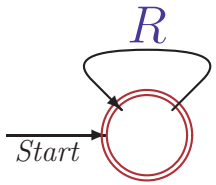
3. **Bilde regulären Ausdruck aus finalem Automaten**

   – $q_0 \neq q$:




$(R^* + SU^*T)^*SU^*$

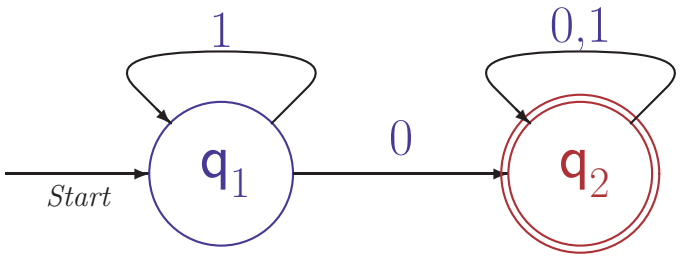
   – $q_0 = q$:



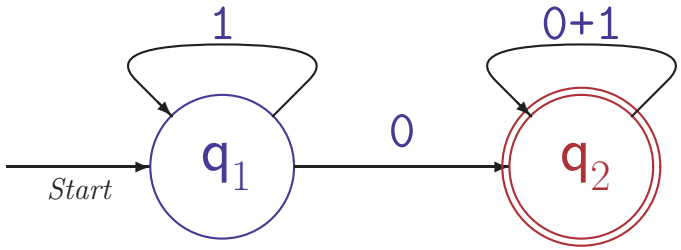

$R^*$

4. **Vereinige Ausdrücke aller Endzustände**
   – Bilde Summe aller entstandenen regulären Ausdrücke

# Umwandlung durch Zustandselimination am Beispiel



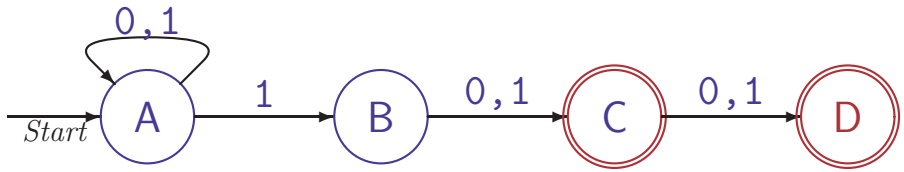

- **Transformiere in RA-Automaten**



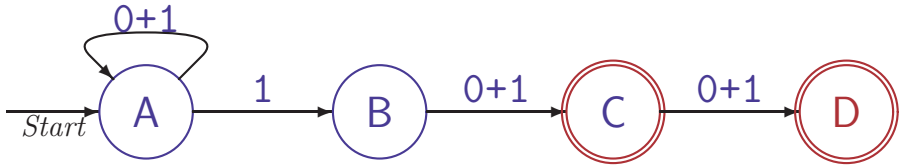

- **Keine Zustände zu eliminieren**

- **Bilde regulären Ausdruck aus finalem Automaten**
  - Extrahierter Ausdruck: `(1 + 0(0+1)*∅)*0(0+1)*`
  - Nach Vereinfachung: `1*0(0+1)*`

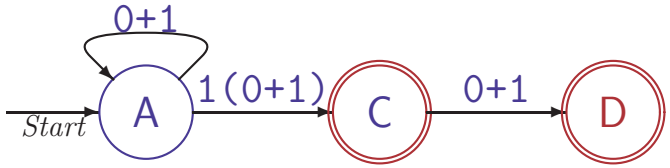
# Umwandlung durch Zustandselimination II



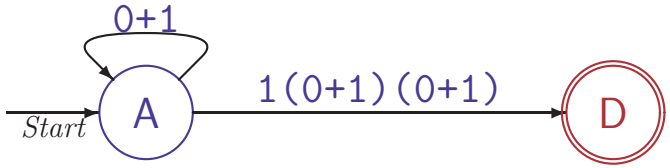

- **Transformiere in RA-Automaten**




- **Elimination von Zustand *B***

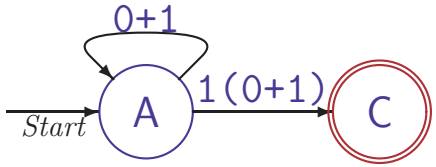



- **Elimination von Zustand *C* für Endzustand *D***




(0+1)*1(0+1)(0+1)

- **Elimination von Zustand *D* für Endzustand *C***




(0+1)*1(0+1)

- **Gesamter Ausdruck:** (0+1)*1(0+1) + (0+1)*1(0+1)(0+1)

# Reguläre Ausdrücke – Zusammenfassung

- **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 von Ausdrücken für Verarbeitungspfade im Automaten
  - Konstruktion durch Elimination von Zuständen in RA Automaten
  - Nachträgliche Optimierungen durch Anwendung algebraischer Gesetze