Theoretische Informatik I

Einheit 2.4

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
  - Bestimme Worte bzw. Symbolketten werden durch Zustände akzeptiert
  - Für Automaten ist Sprache $\hat{=}$ Menge der akzeptierten Worte

- **Wie beschreibt man Eigenschaften von Worten?**
  - **Deklarative Semantik**: äußere Form von Zeichenreihen einer Sprache

    z.B. *Worte haben eine führende Null, dann beliebig viele Einsen*
  - Anwendungen brauchen präzise Beschreibungssprache für Worte
    - Grundeinheiten von Programmiersprachen, Suchmuster für Browser, ...

- **Reguläre Ausdrücke als formale Syntax**
  - Kurze, prägnante Beschreibung des Aufbaus der Worte einer Sprache

    z.B. `01`*: "Zuerst eine Null, dann beliebig viele Einsen"

Reguläre Ausdrücke beschreiben Sprachen

- **Ausdrücke für einfache Grundmengen**
  - Leere Menge
  - Sprache, die nur das leere Wort enthält
  - Sprache, die nur das Symbol $a \in \Sigma$ enthält

- **Ausdrücke für Komposition von Mengen**
  - Vereinigung $L \cup M$ von Sprachen
  - Verkettung $L \circ M$ von Sprachen

    $\boldsymbol{L \circ M} = \{w \in \Sigma^* \mid \exists u \in L.\ \exists v \in M.\ w = uv\}$
  - (Kleene'sche) Hülle L^* von Sprachen

    $\boldsymbol{L^n} = \{w \in \Sigma^* \mid \exists u_1, .., u_n \in L.\ w = u_1..u_n\}\ (n \in \mathbb{N}) \quad (L^0 = \{\epsilon\})$

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

- **Reguläre Ausdrücke sind nicht selbst Mengen**
  - Nur eine syntaktische Beschreibungsform, die ein Computer versteht

ANWENDUNG: 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.

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: Terme über $\Sigma \cup \{\emptyset, \epsilon, +, \circ, ^*, (,)\}$**
  - $\emptyset$, ϵ, und $\mathtt{a}$ (für alle $a \in \Sigma$) sind reguläre Ausdrücke
  - Sind E und F reguläre Ausdrücke, dann auch $E+F$, $E \circ F$, E^* und (E)
- **Semantik: Sprachen über Σ**
  - $L(E)$: Sprache des regulären Ausdrucks E, induktiv definiert
  - $L(\emptyset) = \emptyset$, $L(\epsilon) = \{\epsilon\}$, $L(\mathtt{a}) = \{a\}$ (für alle $a \in \Sigma$)
  - $L(E+F) = L(E) \cup L(F)$, $L(E \circ F) = L(E) \circ L(F)$, $L(E^*) = (L(E))^*$, $L((E)) = L(E)$
- **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

Sprachen vs. Ausdrücke

- **Sprachen sind Mengen**
  - Abstraktes Semantisches Konzept: Ungeordnete Kollektion von Worten
  - 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**
  - Syntaktisches Konstrukt: Struktur, die ein Computer versteht
  - Reguläre Ausdrücke werden zur Beschreibung von Sprachen benutzt

    und sind ähnlich zur Standardnotation von Mengen
- **Reguläre Ausdrücke sind selbst keine Mengen**
  - Unterscheide Ausdruck E von Sprache des Ausdrucks $L(E)$
  - Man verzichtet auf den Unterschied wenn der Kontext eindeutig ist

Entwicklung regulärer Ausdrücke

Beschreibe Menge aller Worte, in denen 0 und 1 abwechseln

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

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

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

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

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

3. Manche Worte nicht erfaßt

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

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

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

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

4. Es geht auch kürzer

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

Algebraische Gesetze für regulärer Ausdrücke

Hilfsmittel zur Vereinfachung regulärer Ausdrücke

- **Äquivalenz von Ausdrücken**
  - $\boldsymbol{E \cong F}$, falls $L(E) = L(F)$ "E äquivalent zu F"
  - Was beweist $\texttt{(01)}^* + \texttt{(10)}^* + \texttt{(01)}^*\texttt{0} + \texttt{(10)}^*\texttt{1} \cong \texttt{(}\epsilon\texttt{+1)(01)}^*\texttt{(}\epsilon\texttt{+0)}$?
  - Algebraische Gesetze erlauben Umschreiben in äquivalente Ausdrücke

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

  $\boldsymbol{(E{\circ}F){\circ}G \cong E{\circ}(F{\circ}G)}$:
  - $L((E{\circ}F){\circ}G) = L(E{\circ}F){\circ}L(G) = L(E){\circ}L(F){\circ}L(G) = L(E){\circ}L(F{\circ}G) = L(E{\circ}(F{\circ}G))$

  $\boldsymbol{(E{+}F){+}G \cong E{+}(F{+}G)}$:
  - $L((E{+}F){+}G) = L(E{+}F){\cup}L(G) = L(E){\cup}L(F){\cup}L(G) = \ldots = L(E{+}(F{+}G))$

- **Kommutativität von +**
  - $\boldsymbol{E{+}F \cong F{+}E}$: $L(E{+}F) = = L(E){\cup}L(F) = L(F){\cup}L(E) = L(F{+}E)$
  - Kommutativität von $\circ$ gilt nicht: $= L(\texttt{01}) = \{01\} \neq \{10\} = L(\texttt{10})$

Algebraische Gesetze II

- **Einheiten und Annihilatoren**
  - $\emptyset{+}E \cong E \cong E{+}\emptyset$, $\epsilon{\circ}E \cong E \cong E{\circ}\epsilon$, $\emptyset{\circ}E \cong \emptyset \cong E{\circ}\emptyset$
- **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**
  - $(E^*)^* \cong E^*$, $\emptyset^* \cong \epsilon$, $\epsilon^* \cong \epsilon$
  - $E^+ \cong E{\circ}E^* \cong E^*{\circ}E$, $E^* \cong \epsilon + E^+$

Beweismethodik für algebraische Gesetze

- **Beispiel: Nachweis von $\texttt{(}E\texttt{+}F\texttt{)}^* \cong \texttt{(}E^*F^*\texttt{)}^*$**
  - Sei $w \in L(\texttt{(}E\texttt{+}F\texttt{)}^*)$
  - 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
  - Also $w \in L(\texttt{(}E^*F^*\texttt{)}^*)$
- **Beweis benötigt keine Information über E und F**
  - Man könnte genausogut konkrete Symbole verwenden

    $\texttt{(}E\texttt{+}F\texttt{)}^* \cong \texttt{(}E^*F^*\texttt{)}^*$ gilt weil $\texttt{(a+b)}^* \cong \texttt{(a}^*\texttt{b}^*\texttt{)}^*$ gilt
- **Allgemeines Beweisprinzip**
  - E regulärer Ausdruck mit Metavariablen $F_1,..,F_m$ (für Sprachen $L_1,..,L_m$)

    C entsprechender Ausdruck mit Symbolen $\texttt{a}_1..\texttt{a}_m$ statt der F_i

    $w \in L(E)$ ist zerlegbar in $w = w_1..w_k$ mit $w_i \in L_{j_i}$ g.d.w. $v{=}\texttt{a}_{j_1}..\texttt{a}_{j_k} \in L(C)$

    Beweis durch strukturelle Induktion über Aufbau regulärer Ausdrücke
  - Prüfverfahren für $E \cong F$ ersetzt alle Metavariablen durch Symbole $a \in \Sigma$ und testet dann Gleichheit der konkreten Ausdrücke $\mapsto$ später

Umwandlung regulärer Ausdrücke in Automaten

Sprachen regulärer Ausdrücke sind endlich erkennbar

Für jeden regulären Ausdruck E gibt es einen ϵ-NEA A mit

– A hat genau einen akzeptierenden Zustand q_f

– Der Startzustand von A ist in keinem $\delta_A(q,a)$ enthalten

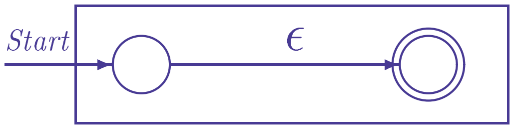
– Für alle $a \in \Sigma$ ist $\delta_A(q_f,a) = \emptyset$

– $L(E) = L(A)$

Beweis durch strukturelle Induktion über Aufbau der regulären Ausdrücke

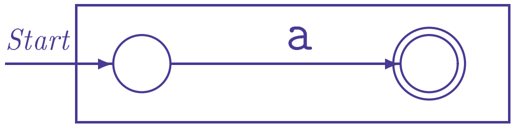
- **Induktionsanfänge**

– Für $E = \epsilon$ wähle $A =$



– Für $E = \emptyset$ wähle $A =$

– Für $E = \mathtt{a}$ wähle $A =$



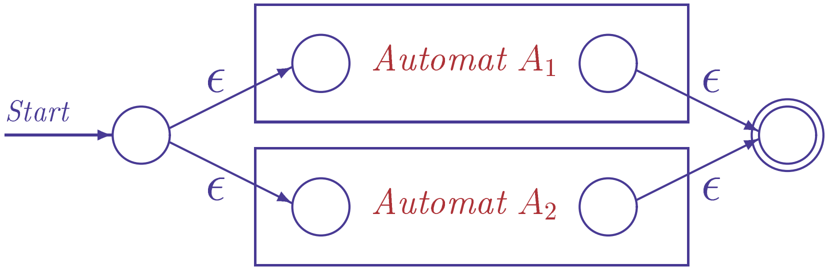
– **Korrektheit offensichtlich**, da jeweils maximal ein Zustandsübergang

Umwandlung regulärer Ausdrücke in Automaten

- **Induktionsannahme: seien A_1 und A_2 ϵ-NEAs für E_1 und E_2**

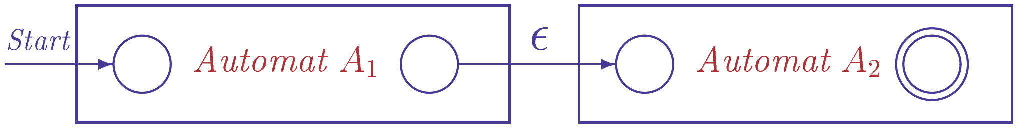
- **Induktionsschritt**

  – Für $E = E_1 + E_2$ wähle $A =$

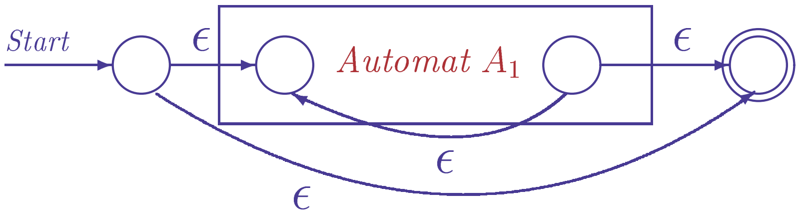


  – Für $E = E_1 \circ E_2$ wähle

  $A =$



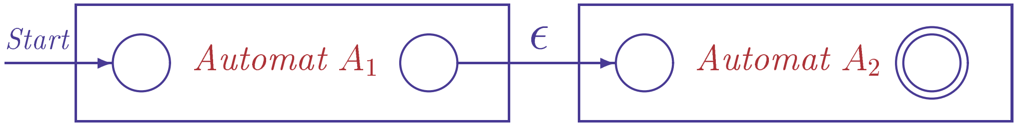
  – Für $E = E_1^*$ wähle $A =$



  – 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 $w \in L(E_1 \circ E_2)$

$$\Rightarrow\ w \in L(E_1) \circ L(E_2) = L(A_1) \circ L(A_2)$$
$$\Rightarrow\ \exists u \in L(A_1).\exists v \in L(A_2).w = uv$$
$$\Rightarrow\ \exists u,v \in \Sigma^*.w = uv \wedge q_{f,1} \in \hat{\delta}_1(q_{0,1},u) \wedge q_{f,2} \in \hat{\delta}_2(q_{0,2},v)$$
$$\Rightarrow\ \exists u,v \in \Sigma^*.w = uv \wedge q_{0,2} \in \hat{\delta}(q_{0,1},u) \wedge q_{f,2} \in \hat{\delta}(q_{0,2},v) \qquad (q_{0,2} \in \epsilon\text{-Hülle}(q_{f,1}))$$
$$\Rightarrow\ q_{f,2} \in \hat{\delta}(q_{0,1},w) \qquad (\text{Definition } \hat{\delta})$$
$$\Rightarrow\ w \in L(A)$$

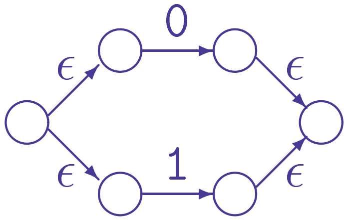
  - Argument ist umkehrbar, also $w \in L(A)\ \Rightarrow\ w \in L(E_1 \circ E_2)$

Sternbildung und Vereinigung in Übungen?

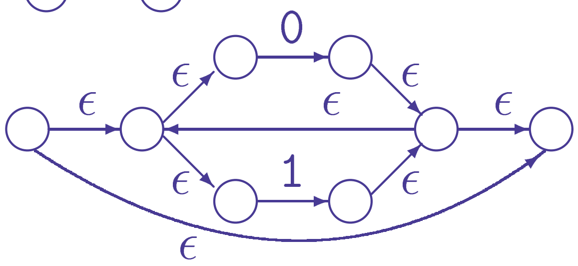
Umwandlung regulärer Ausdrücke am Beispiel

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

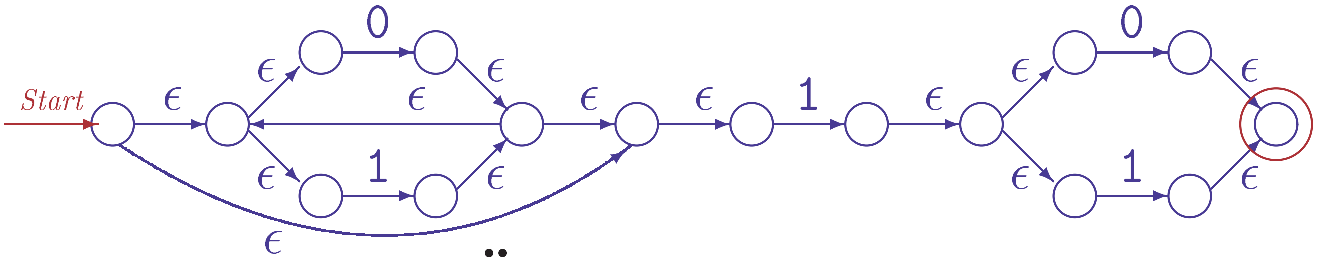
- **Teilautomat für (0+1)**



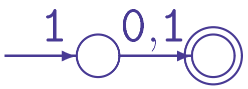
- **Teilautomat für (0+1)***



- **Automat für (0+1)*1(0+1)**



- **Elimination von ϵ-Übergängen**



Umwandlung von Automaten in Reguläre Ausdrücke

- **Gegeben** $A = (\{q_1, ..., q_n\}, \Sigma, \delta, q_1, \{q_{f_1}, .., q_{f_m}\})$
- **Definiere Ausdrücke für Pfade durch** A
  - $R_{ij}^{(k)}$: Regulärer Ausdruck für Menge der Worte w mit $\hat{\delta}(q_i, w) = q_j$
    so daß für alle $\epsilon{\neq}v{\sqsubseteq}w$ $(v{\neq}w)$ gilt: $\hat{\delta}(q_i, v){=}q_n \Rightarrow n{\leq}k$
    (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 Worte w mit $\hat{\delta}(q_i, w) = q_j$
  - Setze $R = R_{1f_1}^{(n)} + ... + R_{1f_m}^{(n)}$
  - Dann gilt $L(R)$

$$= \bigcup_{j=1}^{m} \{w \in \Sigma^* \mid \hat{\delta}(q_1, w) = q_{f_j}\}$$

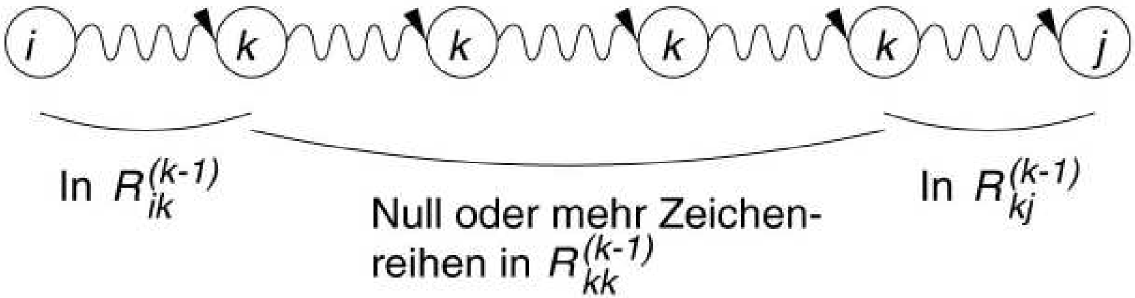
$$= \{w \in \Sigma^* \mid \exists q \in \{q_{f_1}, .., q_{f_m}\}.\ \hat{\delta}(q_1, w) = q\}$$

$$= L(A)$$

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: $R_{ii}^0 = \epsilon + a_1 + .. + a_k$, $R_{ij}^0 = \emptyset + a_1 + .. + a_k$ $(i{\neq}j)$
    wobei $\{a_1, .., a_k\} = \{a \in \Sigma \mid \delta(q_i, a){=}q_j\}$

- **Schrittfall R_{ij}^k** $(0{<}k{\leq}n)$
  - Worte $w \in L(R_{ij}^k)$, deren Pfad q_k nicht enthält: $L(R_{ij}^{k-1}) \subseteq L(R_{ij}^k)$
  - Worte $w \in L(R_{ij}^k)$, 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

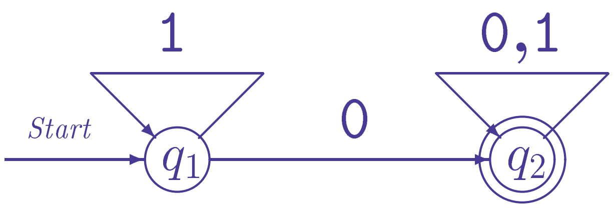
- **Basisfall**

$R^0_{11} = \epsilon + 1$

$R^0_{12} = 0$

$R^0_{21} = \emptyset$

$R^0_{22} = \epsilon + 0 + 1$



- **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**

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

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

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

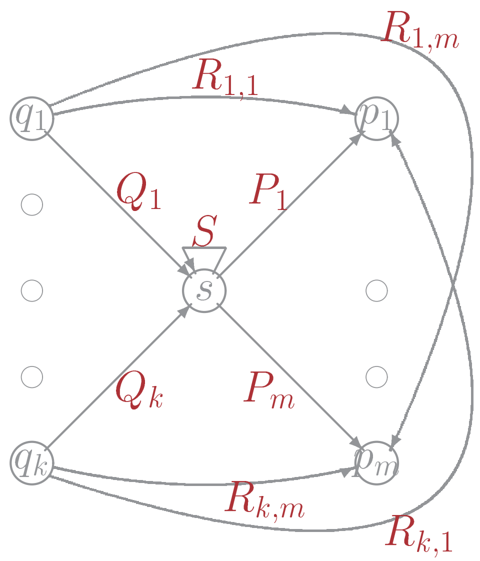
$$R^2_{22} = R^1_{22} + R^1_{22}(R^1_{22})^* R^1_{22} = (\epsilon+0+1) + (\epsilon+0+1)(\epsilon+0+1)^*(\epsilon+0+1) \mapsto (0+1)^*$$

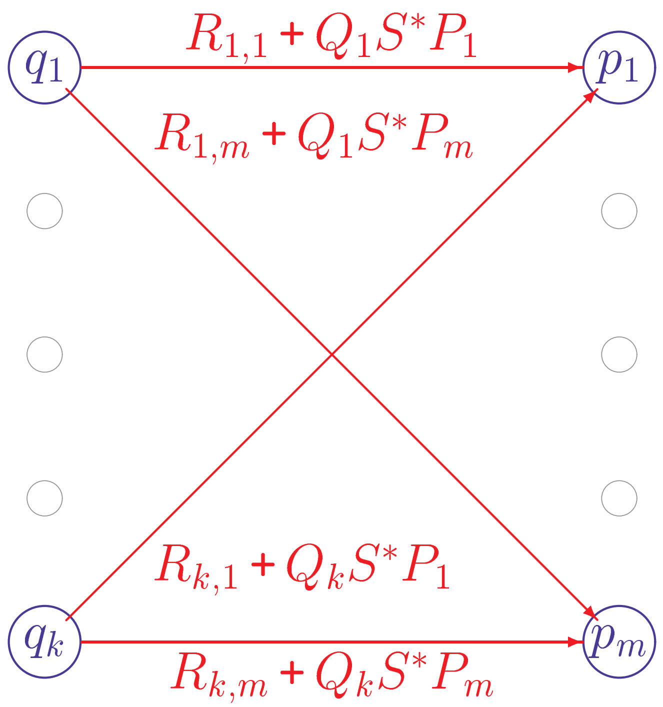
Regulärer Ausdruck des Automaten: $1^* 0(0+1)^*$

Eine effizientere Umwandlungsmethode

- **Direkte Umwandlung ist sehr aufwendig**
  - Es müssen 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 δ 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 $p_1, .., p_n$

– Eliminiere Pfad von q_1 nach p_1 über s: $R_{1,1} + Q_1 S^* P_1$

$\vdots$

– 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: $R_{k,1} + Q_k S^* P_1$

$\vdots$

– 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

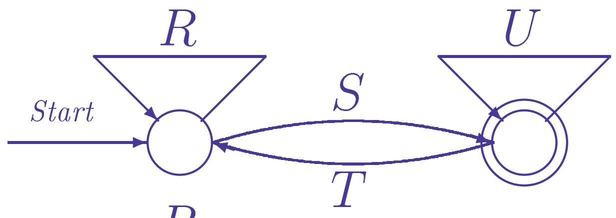
– Ersetze Beschriftungen mit Symbolen $a \in \Sigma$ durch reguläre Ausdrücke

2. Für $q \in F$ eliminiere alle Zustände außer q_0 und q

– Iterative Anwendung des Eliminationsverfahrens

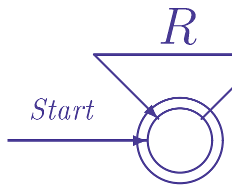
3. Bilde regulären Ausdruck aus finalem Automaten

– $q_0 \neq q$:



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

– $q_0 = q$:

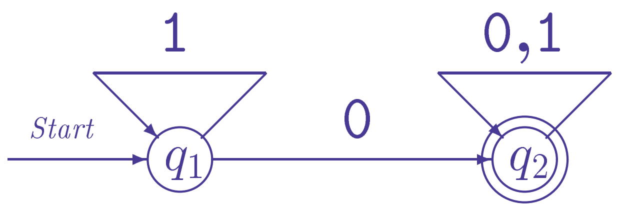


R^*

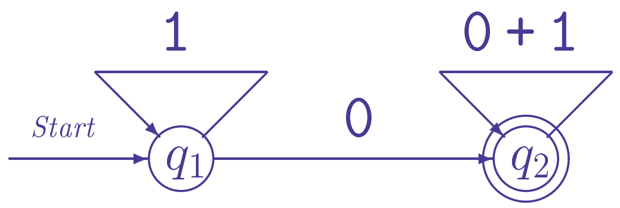
4. Vereinige Ausdrücke aller Endzustände

– Bilde Summe aller entstandenen regulären Ausdrücke

Umwandlung durch Zustandselimination am Beispiel



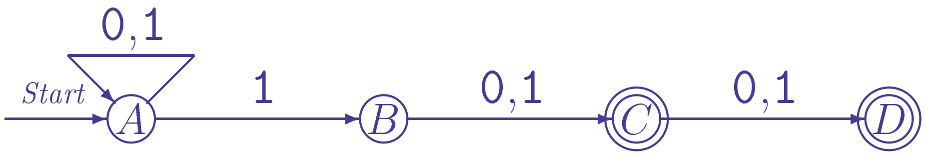
- **Transformiere in RA-Automaten**



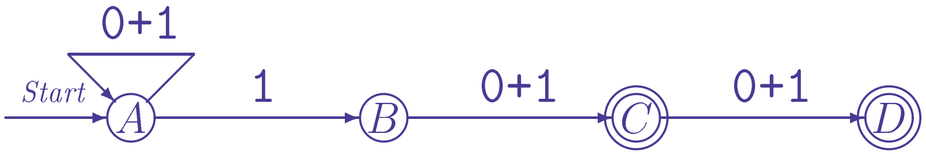
- **Keine Zustände zu eliminieren**

- **Bilde regulären Ausdruck aus finalem Automaten**
  - Extrahierter Ausdruck: $(1 + 0(0+1)^*\emptyset)^*0(0+1)^*$
  - Nach Vereinfachung: $\boxed{1^*0(0+1)^*}$

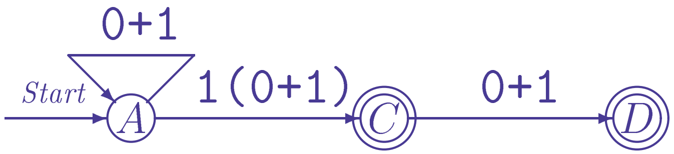
Umwandlung durch Zustandselimination am Beispiel



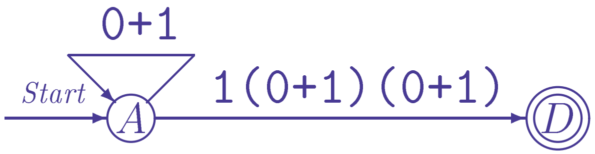
- **Transformiere in RA-Automaten**



- **Elimination von Zustand *B***

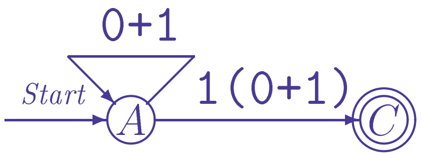


- **Elimination von Zustand *C* für Endzustand *D***



$(0+1)^*1(0+1)(0+1)$

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



$(0+1)^*1(0+1)$

- **Gesamter Ausdruck:** $\boxed{(0+1)^*1(0+1) + (0+1)^*1(0+1)(0+1)}$

Reguläre Ausdrücke – Zusammenfassung

- **Algebraische Notation für Sprachen**
  - ϵ, $\emptyset$, 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 ϵ-NEAs
  - Nachträgliche Optimierung durch Elimination von ϵ-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