

# Formale Systeme

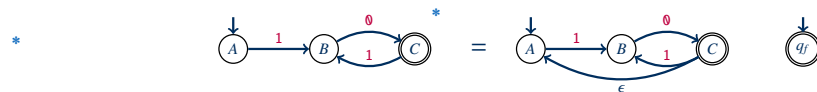
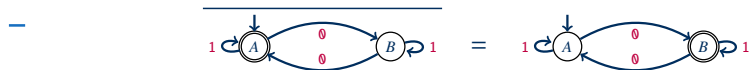
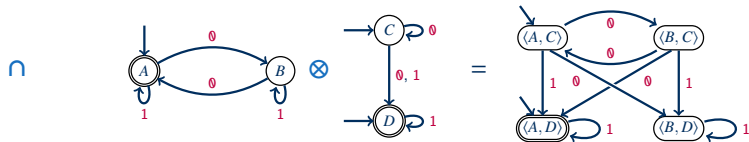
## 6. Vorlesung: Reguläre Ausdrücke

Markus Krötzsch

Professur für Wissensbasierte Systeme

TU Dresden, 30. Oktober 2025

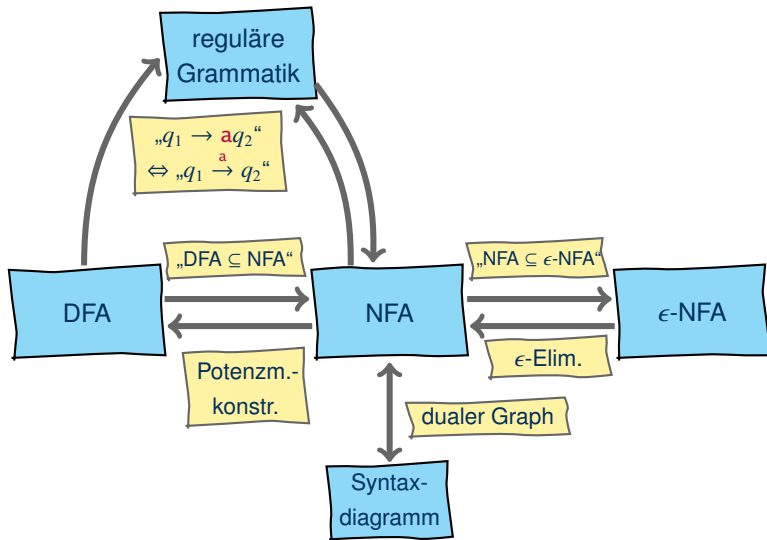
# Wiederholung: Operationen auf Automaten



# NFAs mit Wortübergängen



# Darstellungen von Typ-3-Sprachen



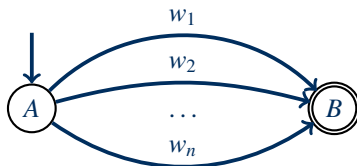
# Reguläre Ausdrücke

# Endliche Sprachen

Eine einfache Beobachtung:

**Satz:** Jede endliche Sprache ist regulär.

**Beweis:** Man kann eine beliebige endliche Sprache  $\{w_1, \dots, w_n\}$  durch einen NFA mit Wortübergängen erkennen:



Wie in der letzten Vorlesung gezeigt, kann dieser in einen NFA umgeformt werden. Jeder NFA akzeptiert eine reguläre Sprache. □

# Sprachen konstruieren?

Wir haben gesehen:

- Jede endliche Sprache ist regulär.
- Durch Anwendung von  $\cap$ ,  $\cup$ ,  $\bar{\phantom{x}}$ ,  $\circ$  und  $*$  entstehen aus regulären Sprachen immer wieder reguläre Sprachen.

Eine natürliche Frage ist also:

Welche regulären Sprachen kann man durch Anwendung von  $\cap$ ,  $\cup$ ,  $\bar{\phantom{x}}$ ,  $\circ$  und  $*$  aus endlichen Sprachen konstruieren?

Alle!

# Die kleinste $(\cup, \circ, *)$ -abgeschlossene Klasse

Überraschender Weise sind  $\cap$  und  $\bar{\phantom{x}}$  nicht einmal nötig!

**Satz:** Alle regulären Sprachen können durch Anwendung von  $\cup$ ,  $\circ$  und  $*$  aus endlichen Sprachen konstruiert werden.

Mit  $\circ$  und  $\cup$  kann man jede endliche Sprache leicht aus den Sprachen  $\emptyset$ ,  $\{\epsilon\}$  und  $\{a\}$  ( $a \in \Sigma$ ) konstruieren.

Mit den bekannten Abschlusseigenschaften erhält man also:

**Satz:** Die Klasse der regulären Sprachen ist die kleinste Klasse von Sprachen mit den folgenden Eigenschaften:

- Sie enthält die Sprachen  $\emptyset$ ,  $\{\epsilon\}$  und  $\{a\}$  für alle  $a \in \Sigma$
- Sie ist abgeschlossen unter den Operatoren  $\cup$ ,  $\circ$  und  $*$

## Beweisplan:

- Definiere diese Klasse syntaktisch: **reguläre Ausdrücke**
- Zeige, dass diese genau die regulären Sprachen darstellen



# Reguläre Ausdrücke

Das motiviert die Einführung einer eigenen Syntax:

Die Menge der **regulärer Ausdrücke** über einem Alphabet  $\Sigma$  ist induktiv<sup>a</sup> wie folgt definiert:

- $\emptyset$  ist ein regulärer Ausdruck
- $\epsilon$  ist ein regulärer Ausdruck
- $a$  ist ein regulärer Ausdruck für jedes  $a \in \Sigma$
- Wenn  $\alpha$  und  $\beta$  reguläre Ausdrücke sind,  
dann sind auch  $(\alpha\beta)$ ,  $(\alpha \mid \beta)$  und  $(\alpha)^*$  reguläre Ausdrücke

---

<sup>a</sup>D.h. die Menge der regulären Ausdrücke ist die kleinste Menge, welche die Bedingungen erfüllt.

Anmerkung: Manchmal werden  $(\alpha + \beta)$  statt  $(\alpha \mid \beta)$  und  $(\alpha \circ \beta)$  oder  $(\alpha \cdot \beta)$  statt  $(\alpha\beta)$  verwendet

Beispiele regulärer Ausdrücke sind  $(a(b)^*)$  oder auch  $(((\epsilon \mid \epsilon))^*)^*$ .

# Reguläre Ausdrücke als formale Sprache

Man kann die Menge der regulären Ausdrücke über dem Alphabet  $\Sigma = \{\sigma_1, \dots, \sigma_n\}$  auch als kontextfreie Grammatik über dem Alphabet  $\Sigma \cup \{\emptyset, \epsilon, (, ), |, *\}$  beschreiben:

$$S \rightarrow \emptyset \mid \epsilon \mid A \mid (SS) \mid (S|S) \mid (S)^*$$

$$A \rightarrow \sigma_1 \mid \dots \mid \sigma_n$$

Solche Notationen werden in der Praxis oft weiter vereinfacht:

- Endliche Mengen als Nichtterminale:

$$S \rightarrow \emptyset \mid \epsilon \mid \Sigma \mid (SS) \mid (S|S) \mid (S)^*$$

- Mehrere Nichtterminale als Hinweis auf unterschiedliche Ausdrücke:

$$\alpha, \beta \rightarrow \emptyset \mid \epsilon \mid \Sigma \mid (\alpha\beta) \mid (\alpha|\beta) \mid (\alpha)^*$$

# Semantik regulärer Ausdrücke

Reguläre Ausdrücke beschreiben die erwarteten Sprachen:

Die **Sprache eines regulären Ausdrucks**  $\alpha$  wird mit  $\mathbf{L}(\alpha)$  bezeichnet und rekursiv definiert:

- $\mathbf{L}(\emptyset) = \emptyset$
- $\mathbf{L}(\epsilon) = \{\epsilon\}$
- $\mathbf{L}(a) = \{a\}$  für jedes  $a \in \Sigma$
- $\mathbf{L}((\alpha\beta)) = \mathbf{L}(\alpha) \circ \mathbf{L}(\beta)$
- $\mathbf{L}((\alpha \mid \beta)) = \mathbf{L}(\alpha) \cup \mathbf{L}(\beta)$
- $\mathbf{L}((\alpha)^*) = \mathbf{L}(\alpha)^*$

**Beispiel:**  $\mathbf{L}(((ab))^*) = (\mathbf{L}((ab)))^* = (\mathbf{L}(a) \circ \mathbf{L}(b))^* = (\{a\} \circ \{b\})^* = \{ab\}^*$

# Äquivalenz regulärer Ausdrücke

Zwei reguläre Ausdrücke  $\alpha$  und  $\beta$  sind **äquivalent**, in Symbolen  $\alpha \equiv \beta$ , wenn  $L(\alpha) = L(\beta)$ .

Typische Rechenregeln der Sprachoperationen gelten analog:

$$\alpha \mid \beta \equiv \beta \mid \alpha$$

$$\alpha(\beta \mid \gamma) \equiv \alpha\beta \mid \alpha\gamma$$

$$(\beta \mid \gamma)\alpha \equiv \beta\alpha \mid \gamma\alpha$$

$$\epsilon^* \equiv \epsilon$$

$$\epsilon\alpha \equiv \alpha\epsilon \equiv \alpha$$

$$\emptyset\alpha \equiv \alpha\emptyset \equiv \emptyset$$

$$\emptyset \mid \alpha \equiv \alpha \mid \emptyset \equiv \alpha$$

$$\emptyset^* \equiv \epsilon$$

# Vereinfachte Klammerung

Reguläre Ausdrücke können durch Klammerungsregeln vereinfacht werden: \* bindet stärker als Konkatenation bindet stärker als |

**Beispiel:**  $ab^* | bc$  ist kurz für  $((a(b)^*) | (bc))$

Konkatenation und Alternative sind assoziativ:

$$L((\alpha(\beta\gamma))) = L(((\alpha\beta)\gamma)) \quad L((\alpha | (\beta | \gamma))) = L(((\alpha | \beta) | \gamma))$$

Daher ist es unproblematisch, dass die Klammerregeln die Reihenfolge nicht spezifizieren.

**Beispiel:**  $101010$  könnte für genau 42 verschiedene reguläre Ausdrücke stehen, unter anderem für  $(((((10)1)0)1)0)$ ,  $((10)((10)(10)))$  und  $(1((0((10)1))0))$ .

# Kurzschreibweisen für reguläre Ausdrücke

Man kann eine Reihe von Kurzschreibweisen definieren:

- $\alpha^+$  ist kurz für  $\alpha(\alpha)^*$
- $\alpha?$  ist kurz für  $(\alpha \mid \epsilon)$
- $\alpha\{n, m\}$  mit  $0 \leq n \leq m$  ist kurz für  $\underbrace{(\alpha \dots \alpha)}_{n\text{-mal}} \mid \dots \mid \underbrace{(\alpha \dots \alpha)}_{m\text{-mal}}$

Implementierungen regulärer Ausdrücke bieten auch Kurzformen für Alternativen einzelner Symbole („Character Classes“):

- $.$ : beliebiges Symbol, d.h.  $\sigma_1 \mid \dots \mid \sigma_n$  falls  $\Sigma = \{\sigma_1, \dots, \sigma_n\}$
- $[\theta_1 \dots \theta_\ell]$ : beliebiges Symbol aus einer Liste, d.h.  $\theta_1 \mid \dots \mid \theta_\ell$
- $[\wedge \theta_1 \dots \theta_\ell]$ : beliebiges Symbol **nicht** aus einer Liste, d.h.  $\sigma_1 \mid \dots \mid \sigma_n$  mit  $\{\sigma_1, \dots, \sigma_n\} = \Sigma \setminus \{\theta_1, \dots, \theta_\ell\}$
- Weitere Kurzformen für praktisch wichtige Fälle, z.B. `\s` oder `[:blank:]` für Leerzeichen, `\d` oder `[:digit:]` für Ziffern

**Theoretiker:innen meiden Kurzformen** (mehr Formen = mehr Fälle in Beweisen und Definitionen)

# Regexps in der Praxis

Reguläre Ausdrücke sind von großer praktischer Bedeutung

- Mustererkennung in Texten (Pattern Matching)
- Lexer/Tokenizer
- Suche nach Muster in Datenbanken
- ...

Unterschiede zur reinen Lehre:

- **Pattern Matching:** (1) spezifiziere eine Sprache (Suchwörter) und (2) finde deren Vorkommen (Matches) in einem längeren Wort (Text)  $\leadsto$  Möglichkeiten zur Steuerung des zweiten Teils nötig (z.B. **greedy** vs. **lazy** matching)
- **Referenzen:** praktische Implementierungen erlauben es meist, Teile eines Matches im Muster wieder zu verwenden  $\leadsto$  keine reguläre Sprache mehr
- **Escaping:** Unterscheidung von Steuerzeichen (Metazeichen) und Alphabetssymbolen ist praktisch aufwändig

reg·ex `/^[a-z0-9]/`

noun, plural regrets

Have you tried using an XML parser instead?



# Von Regulären Ausdrücken zu Automaten

# Zielstellung

**Behauptung:** Eine Sprache  $L$  ist genau dann regulär, wenn es einen regulären Ausdruck  $\alpha$  gibt mit  $L(\alpha) = L$ .

**Beweis (Plan):**

- Teilbehauptung 1: Für jeden regulären Ausdruck  $\alpha$  gibt es einen NFA  $M$ , so dass  $L(\alpha) = L(M)$

Zwei mögliche Beweismethoden:

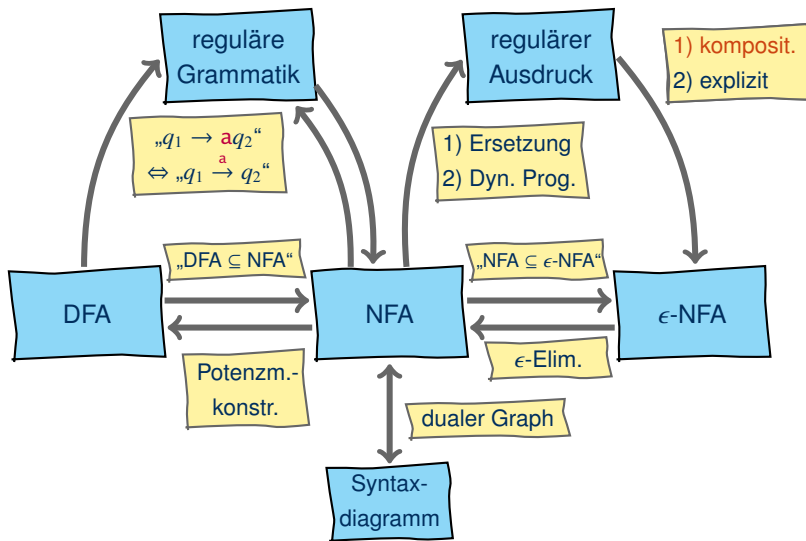
- (1) Kompositionelle Methode
- (2) Explizite Konstruktion

- Teilbehauptung 2: Für jeden NFA  $M$  gibt es einen regulären Ausdruck  $\alpha$ , so dass  $L(\alpha) = L(M)$

Zwei mögliche Beweismethoden:

- (1) Ersetzungsmethode
- (2) Dynamische Programmierung

# Darstellungen von Typ-3-Sprachen



# Rekursive Komposition von $\epsilon$ -NFA

Die Struktur regulärer Ausdrücke kann mit Operationen auf Automaten direkt abgebildet werden.

Für einen Ausdruck  $\alpha$  definieren wir rekursiv den  $\epsilon$ -NFA  $\mathcal{M}(\alpha)$ :

## Grundfälle:

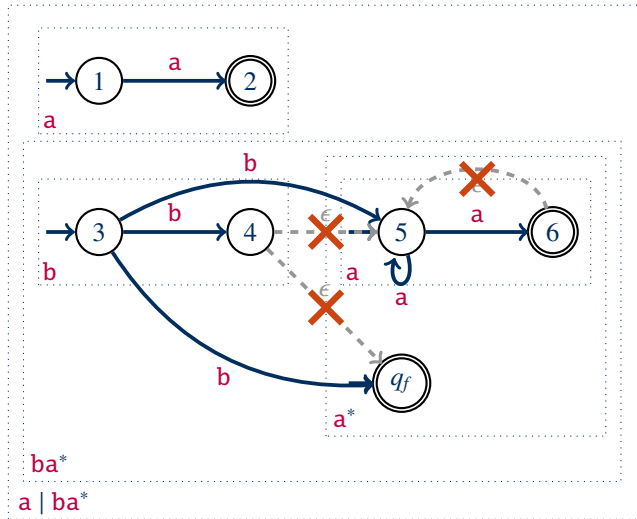
- Wenn  $\alpha = \emptyset$  dann  $\mathcal{M}(\alpha) = \rightarrow A$
- Wenn  $\alpha = \epsilon$  dann  $\mathcal{M}(\alpha) = \rightarrow A$
- Wenn  $\alpha = a$  dann  $\mathcal{M}(\alpha) = \rightarrow A \xrightarrow{a} B$

**Rekursive Fälle:** Wir bezeichnen mit  $\text{elim}_\epsilon(\mathcal{M})$  den NFA, der aus einem  $\epsilon$ -NFA  $\mathcal{M}$  durch Eliminierung der  $\epsilon$ -Übergänge entsteht.

- Wenn  $\alpha = (\beta\gamma)$  dann  $\mathcal{M}(\alpha) = \text{elim}_\epsilon(\mathcal{M}(\beta) \odot \mathcal{M}(\gamma))$
- Wenn  $\alpha = (\beta \mid \gamma)$  dann  $\mathcal{M}(\alpha) = \mathcal{M}(\beta) \oplus \mathcal{M}(\gamma)$
- Wenn  $\alpha = (\beta)^*$  dann  $\mathcal{M}(\alpha) = \text{elim}_\epsilon(\mathcal{M}(\beta)^*)$

# Beispiel

Regulärer Ausdruck:  $a \mid ba^*$



# Korrektheit der Kompositionsmethode

**Satz:** Für jeden regulären Ausdruck  $\alpha$  gilt  $\mathbf{L}(\alpha) = \mathbf{L}(\mathcal{M}(\alpha))$ .

**Beweis:** Die Gleichheit folgt aus der Definition von  $\mathbf{L}(\alpha)$  und der Korrektheit der Operationen auf Automaten und der  $\epsilon$ -Eliminierung.

Formal ist der Beweis eine **strukturelle Induktion**: wir konstruieren unsere Argumentation entlang der Struktur regulärer Ausdrücke.

- **Induktionsanfang:** Für die Grundfälle  $\alpha = \emptyset$ ,  $\alpha = \epsilon$  und  $\alpha = a$  ist die Behauptung leicht zu sehen
- **Induktionshypothese (IH):** Die Behauptung wurde bereits für  $\beta$  und  $\gamma$  gezeigt, d.h.  $\mathbf{L}(\beta) = \mathbf{L}(\mathcal{M}(\beta))$  und  $\mathbf{L}(\gamma) = \mathbf{L}(\mathcal{M}(\gamma))$
- **Induktionsschritt:** Im Fall  $\alpha = (\beta\gamma)$  gilt:

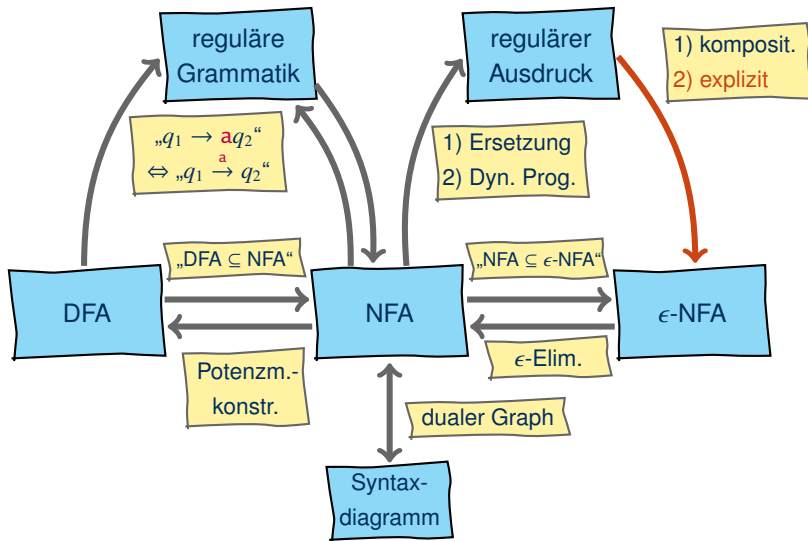
$$\begin{aligned}\mathbf{L}(\alpha) &\stackrel{\text{Def.}}{=} \mathbf{L}(\beta) \circ \mathbf{L}(\gamma) \stackrel{\text{IH}}{=} \mathbf{L}(\mathcal{M}(\beta)) \circ \mathbf{L}(\mathcal{M}(\gamma)) \\ &\stackrel{(1)}{=} \mathbf{L}(\mathcal{M}(\beta) \odot \mathcal{M}(\gamma)) \stackrel{(2)}{=} \mathbf{L}(\text{elim}_{\epsilon}(\mathcal{M}(\beta) \odot \mathcal{M}(\gamma))) \stackrel{\text{Def.}}{=} \mathbf{L}(\mathcal{M}(\alpha)).\end{aligned}$$

(1) Korrektheit der Operation  $\odot$ ; (2) Korrektheit der  $\epsilon$ -Eliminierung

Die Fälle  $\alpha = (\beta \mid \gamma)$  und  $\alpha = (\beta)^*$  sind analog.

□

# Darstellungen von Typ-3-Sprachen



# Explizite Konstruktion des NFA

## Idee:

- Beginne mit einem „NFA mit RegExp-Übergängen“, bei dem Kanten mit regulären Ausdrücken beschriftet sind
- Eliminiere diese Übergänge schrittweise, ähnlich wie beim Eliminieren von Wortübergängen

Es ist einfacher, das für reguläre Ausdrücke zu tun, die kein  $\emptyset$  enthalten



# Eliminierung von $\emptyset$

Der folgende einfache Algorithmus erzeugt reguläre Ausdrücke ohne innere Vorkommen von  $\emptyset$ :

**Eingabe:** regulärer Ausdruck  $\alpha$

**Ausgabe:** äquivalenter regulärer Ausdruck  $\beta$  ohne  $\emptyset$

(falls  $\mathbf{L}(\alpha) \neq \emptyset$ ) oder  $\emptyset$  (falls  $\mathbf{L}(\alpha) = \emptyset$ )

Wende die folgenden Ersetzungsregeln erschöpfend auf Teilausdrücke von  $\alpha$  an:

- $(\gamma \mid \emptyset) \mapsto \gamma$  und  $(\emptyset \mid \gamma) \mapsto \gamma$
- $(\gamma\emptyset) \mapsto \emptyset$  und  $(\emptyset\gamma) \mapsto \emptyset$
- $(\emptyset)^* \mapsto \epsilon$

Gib das Ergebnis dieser Ersetzungen aus.

Die Korrektheit des Algorithmus folgt aus der Korrektheit der angewendeten Ersetzungsregeln

# Explizite Konstruktion von NFAs

Für den Ausdruck  $\emptyset$  können wir einen NFA direkt angeben (wie vorn); andernfalls gehen wir wie folgt vor:

**Gegeben:** regulärer Ausdruck  $\alpha$  ohne  $\emptyset$

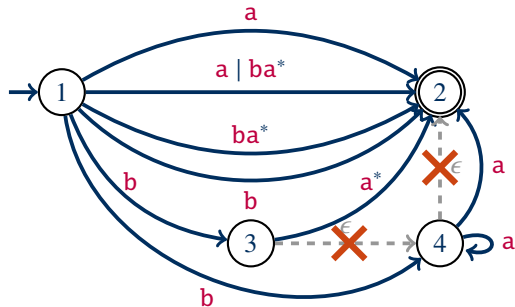
Initialisierung:  $\mathcal{M}_\alpha = \rightarrow A \xrightarrow{\alpha} B$

Solange es in  $\mathcal{M}_\alpha$  Übergänge  $q \xrightarrow{\beta} p$  gibt, die mit einem Ausdruck  $\beta \notin \{\epsilon\} \cup \Sigma$  beschriftet sind, wende eine der folgenden Regeln an:

- Ersetze  $q \xrightarrow{(\gamma_1\gamma_2)} p$  durch  $q \xrightarrow{\gamma_1} r \xrightarrow{\gamma_2} p$
- Ersetze  $q \xrightarrow{(\gamma_1 \mid \gamma_2)} p$  durch  $q \xrightarrow{\gamma_1} p$  und  $q \xrightarrow{\gamma_2} p$
- Ersetze  $q \xrightarrow{(\gamma)^*} p$  durch  $q \xrightarrow{\epsilon} r \xrightarrow{\gamma} r \xrightarrow{\epsilon} p$

# Beispiel

Regulärer Ausdruck:  $a \mid ba^*$

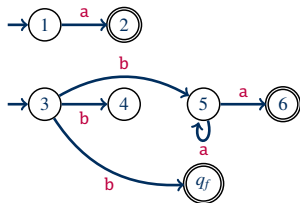


$\epsilon$ -Übergänge können wie gewohnt eliminiert werden, um einen NFA zu erhalten

# Vergleich NFA-Konstruktionen

Ausdruck:  $a \mid ba^*$

## Kompositioneller Ansatz



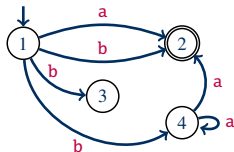
Linear viele Zustände\*

Meist etwas größer

Korrektheitsbeweis aus  
Abschlusseigenschaften

Fast ohne Löschen (außer  $\epsilon$ -Übergänge)

## Expliziter Ansatz



Linear viele Zustände\*

Meist etwas kleiner

Korrektheitsbeweis erfordert  
neue Argumentation

Übergänge werden oft gelöscht

\*) Bzgl. Länge des reg. Ausdrucks bzw. bzgl. Anzahl seiner Operationen

# Zusammenfassung und Ausblick

**Reguläre Ausdrücke** sind eine praktisch wichtige Methode zur Beschreibung (beliebiger) regulärer Sprachen

Die **kompositionelle Methode** wendet Automaten-Operationen an, um aus einem regulären Ausdruck einen NFA zu erzeugen

Die **explizite Methode** verwendet Übergänge mit regulären Ausdrücken, die schrittweise expandiert werden

## Offene Fragen:

- Wie kommt man zurück von NFA zu regulärem Ausdruck?
- Welche Sprachen sind nicht regulär?
- Wie kann man Automaten systematisch vereinfachen?