

Grundbegriffe der Theoretischen Informatik

Sommersemester 2017 - Beate Bollig

Die Folien basieren auf den Materialien von Thomas Schwentick.

Teil A: Reguläre Sprachen

2: Endliche Automaten

Testprogramme für reguläre Sprachen

- Wir haben bereits einen (erweiterten) regulären Ausdruck für e-Mail-Adressen konstruiert:
$$([a-zA-Z][a-zA-Z0-9\-_]*\.)^*[a-zA-Z][a-zA-Z0-9\-_]*@[a-zA-Z][a-zA-Z0-9\-_]*\.[a-zA-Z]{2,4}$$
- Jetzt beschäftigen wir uns mit Testalgorithmen für reguläre Sprachen
 - Also: Algorithmen, die für eine gegebene Sprache L testen, ob ein Eingabewort w in L ist (**Wortproblem**)
- Im ersten Teil werden wir aus einem in Pseudocode geschriebenen Testalgorithmus ein Berechnungsmodell für reguläre Sprachen abstrahieren: **endliche Automaten**
- Im zweiten Teil werden wir eine flexiblere Variante endlicher Automaten betrachten, die die automatische Umwandlung von REs in endliche Automaten erleichtern wird

Ein „Programm“ zur Erkennung von e-Mail-Adressen

Alg. 2.1: Erkennung von e-Mail-Adressen

```
InLabel := false; Error := false;
Local := true; Letters := true
chars := 0; labs := 0;
while NOT EOF() do
    z := NextSymbol
    case z IN
        [a - zA - Z]:
            InLabel := true; chars := chars + 1
        [0-9\-\_]:
            if InLabel then
                chars := chars + 1; Letters := false
            else
                Error := true
        [.]:
            if InLabel then
                chars := 0; labs := labs + 1
                InLabel := false; Letters := false
            else
                Error := true
```


Alg. 2.1 (Fortsetzung)

```
while ... do {Fortsetzung while-Schleife}
    case ...
        [@]:
            if Local then
                chars := 0; labs := 0;
                InLabel := false;
                Local := false; Letters := true
            else
                Error := true
    if NOT Error AND Letters AND NOT Local
    AND labs ≥ 1 AND 2 ≤ chars ≤ 4 then
        Print(„OK“)
    else
        Print(„Nicht OK“)
```

- Das ist „einfach so runter programmiert“
 - Ist das Programm korrekt? ☐
- Gibt es einen Weg, um einen regulären Ausdruck **automatisch** und **zuverlässig** in ein Programm zu übersetzen?


Vom Programm zum Automaten

- Was ist besonders an Algorithmus 2.1?
 - Die Eingabe wird **Zeichen für Zeichen** gelesen und verarbeitet
 - Es gibt nur **begrenzt viele** (relevante) Kombinationen von **Werten** der Programm-Variablen:
 - * InLabel $\in \{\text{true}, \text{false}\}$
 - * Error $\in \{\text{true}, \text{false}\}$
 - * Local $\in \{\text{true}, \text{false}\}$
 - * Letters $\in \{\text{true}, \text{false}\}$
 - * chars $\in \{0, 1, 2, 3, 4, „\geq 5“\}$
 - * labs $\in \{0, „\geq 1“\}$


- Wir nennen jede mögliche **Kombination von Variablenwerten** einen **Zustand** 

- Beim Lesen eines Zeichens geht das Programm also von einem Zustand in einen anderen Zustand über

- Ein solches System aus (endlich vielen) Zuständen und Zustandsübergängen heißt **endliches Transitionssystem** oder **endlicher Automat**

 Die feinen Unterschiede zwischen diesen Begriffen werden wir später betrachten

- Wieviele Zustände hat der Automat für Algorithmus 2.1?

– Bei naiver Vorgehensweise: 

- Wir werden sehen:
das geht erheblich besser

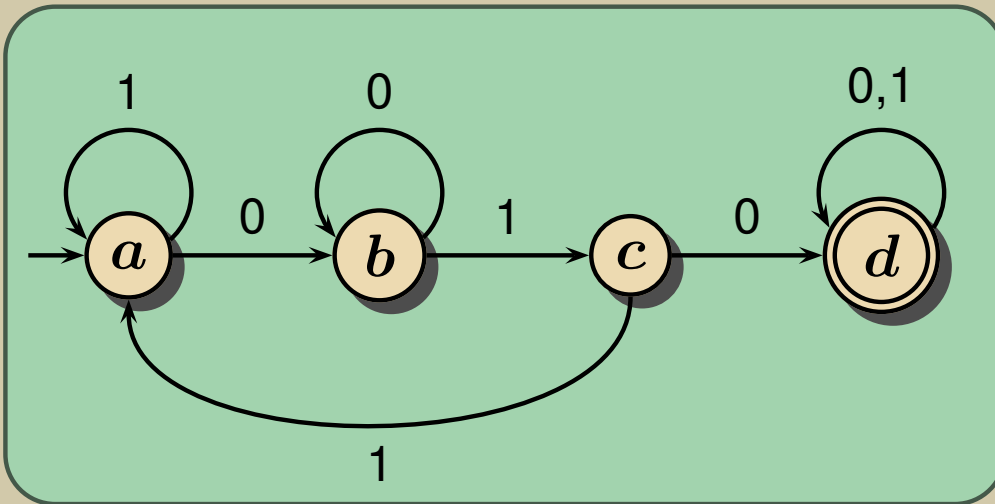
- Jetzt betrachten wir aber zunächst mal ein kleineres Beispiel eines Automaten

Inhalt

- ▷ **2.1 Endliche Automaten: Definition und Konstruktion**
- 2.2 Nichtdeterministische endliche Automaten
- 2.3 Von regulären Ausdrücken zu nichtdeterministischen Automaten

Endliche Automaten: Beispiel


Beispiel



- Eingabe: 001101010
- Eingabe: 101100

- Der String 001101010 wird von dem Automaten akzeptiert
- Der String 101100 wird von dem Automaten nicht akzeptiert

- Dieser Automat akzeptiert einen String genau dann, wenn er den Teilstring 010 enthält, also wenn er in der Sprache $L((0 + 1)^*010(0 + 1)^*)$ ist
- Wir sagen, dass der Automat die Sprache $L((0 + 1)^*010(0 + 1)^*)$ **entscheidet**

-  Die graphische Darstellung von Automaten ist zwar anschaulich, aber wir benötigen präzise Definitionen, die unzweideutig festlegen,
- was ein endlicher Automat ist, und
 - wie ein endlicher Automat „funktioniert“

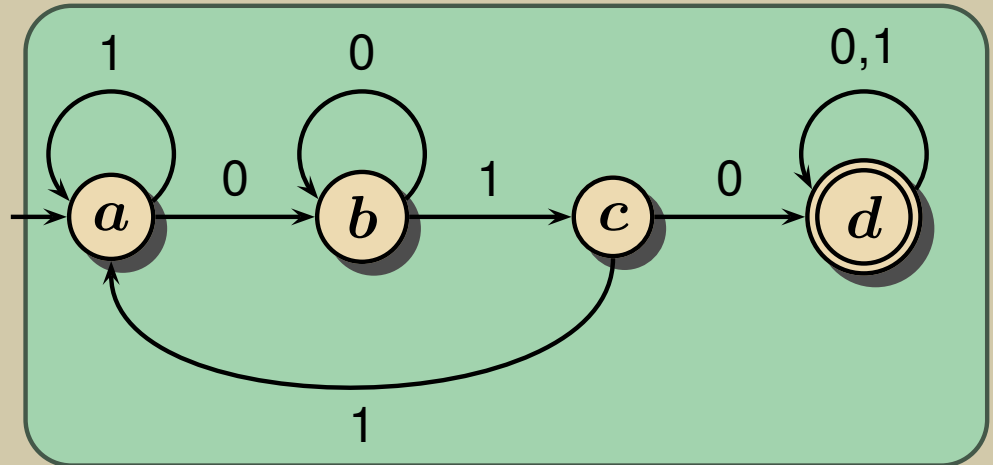
Endliche Automaten: Definition

Definition: Syntax endlicher Automaten 2.1

- Ein **endlicher Automat** \mathcal{A} besteht aus
 - einer endlichen Menge Q von **Zuständen**,
 - einem **Eingabealphabet** Σ ,
 - einer **Transitionsfunktion**
 $\delta : Q \times \Sigma \rightarrow Q$,
 - einem **Startzustand** $s \in Q$ und
 - einer Menge $F \subseteq Q$ **akzeptierenden Zustände**
- Wir schreiben $\mathcal{A} = (Q, \Sigma, \delta, s, F)$

- In der graphischen Darstellung von endlichen Automaten
 - wird der Startzustand durch eine „aus dem nichts kommende“ Kante markiert,
 - werden die akzeptierenden Zustände durch einen doppelten Rand markiert

Beispiel: der 010-Automat formal



- $\mathcal{A}_{010} = (Q_{010}, \{0, 1\}, \delta_{010}, s_{010}, F_{010})$
- $Q_{010} = \{a, b, c, d\}$
- $s_{010} = a$
- $F_{010} = \{d\}$
- $\delta_{010}(a, 0) = b, \delta_{010}(a, 1) = a, \dots$



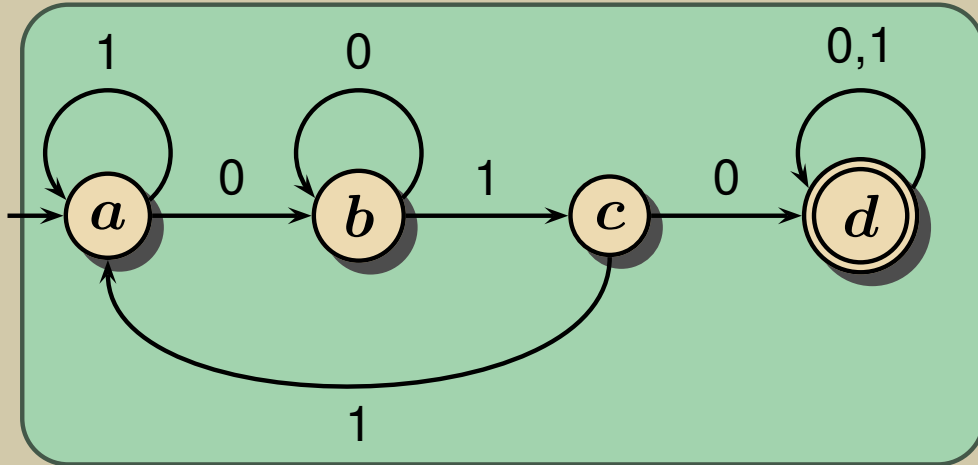
Abkürzung:

DFA für **deterministic finite automaton**

- Plural: **DFAs** für **deterministic finite automata**

Endliche Automaten: Semantik (1/2)

Beispiel



- **Informelle Semantik** endlicher Automaten:
 - Der Automat liest die Eingabe Zeichen für Zeichen, beginnend im Startzustand
 - Er geht dabei jeweils gemäß der Transitionsfunktion δ in einen Zustand über
 - Er **akzeptiert** die Eingabe, falls er am Ende in einem Zustand aus F ist
- Um Aussagen über Automaten **beweisen** zu können, benötigen wir wieder eine **formale Semantik**

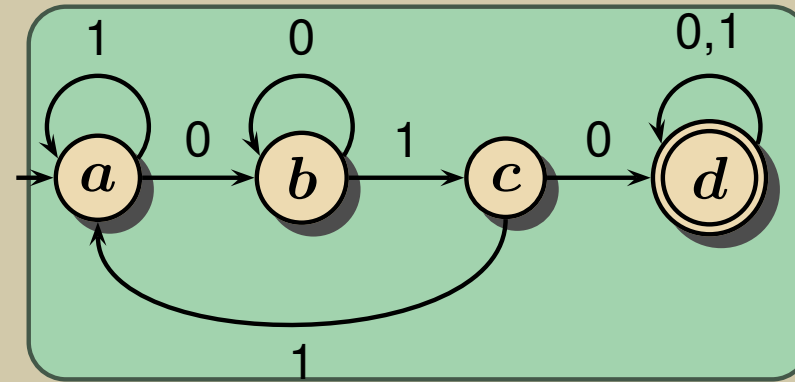
- Dazu definieren wir die **erweiterte Transitionsfunktion** δ^* :
 - $\delta^*(q, w)$ soll der Zustand sein, den der Automat annimmt, wenn er vom Zustand q aus w liest

Definition: Semantik endlicher Automaten

- Die erweiterte Transitionsfunktion $\delta^* : Q \times \Sigma^* \rightarrow Q$ eines Automaten $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ ist wie folgt induktiv definiert:
 - $\delta^*(q, \epsilon) \stackrel{\text{def}}{=} q$,
 - $\delta^*(q, u\sigma) \stackrel{\text{def}}{=} \delta(\delta^*(q, u), \sigma)$
für $u \in \Sigma^*, \sigma \in \Sigma$
- \mathcal{A} akzeptiert w $\stackrel{\text{def}}{\iff} \delta^*(s, w) \in F$
- Von \mathcal{A} entschiedene Sprache:
$$\underline{L(\mathcal{A})} \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}$$

Endliche Automaten: Semantik (2/2)

Beispiel



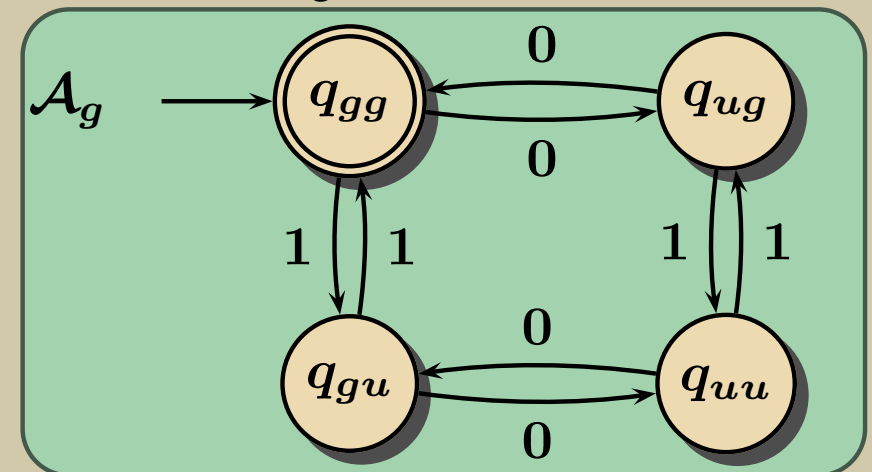
$$\begin{aligned} & \delta^*(a, 0110101) \\ &= \delta(\delta^*(a, 011010), 1) \\ &= \delta(\delta(\delta^*(a, 01101), 0), 1) \\ &= \delta(\delta(\delta(\delta^*(a, 0110), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(\delta^*(a, 011), 0), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(\delta(\delta^*(a, 01), 1), 0), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(\delta(\delta(\delta^*(a, 0), 1), 1), 0), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(\delta(\delta(\delta(\delta^*(a, \epsilon), 0), 1), 1), 0), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(\delta(\delta(a, 0), 1), 1), 0), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(b, 1), 1), 0), 1), 0), 1) \\ &= \delta(\delta(\delta(\delta(c, 1), 0), 1), 0), 1) \\ &= \delta(\delta(\delta(a, 0), 1), 0), 1) \\ &= \delta(\delta(b, 1), 0), 1) \\ &= \delta(c, 0), 1) \\ &= \delta(d, 1) \\ &= d \end{aligned}$$

Konstruktion endlicher Automaten: Beispiel

- Wie lässt sich zu einer gegebenen Sprache ein Automat konstruieren?
- Es sind folgende Fragen zu klären:
 - Welche Informationen muss der Automat sich merken (Zustände)?
 - * Welche Bedeutung haben dann die einzelnen Zustände?
 - Wie ändern sich diese Informationen durch das Lesen eines Zeichens (Transitionen)?
 - Wie ist die Initialisierung? (Startzustand)
 - Wie sollen die gemerkten Informationen das Akzeptieren der Eingabe beeinflussen?

Beispiel

- Konstruktion eines DFA für die Menge L_g aller Zeichenketten über $\{0, 1\}$ mit gerade vielen 0 und 1
- Informationen, die der Automat sich merkt:
 - Ist die Anzahl der bisher gelesenen 0-Zeichen gerade oder ungerade
 - Ist die Anzahl der bisher gelesenen 1-Zeichen gerade oder ungerade
- Das ergibt vier Zustände: q_{gg} , q_{gu} , q_{ug} , q_{uu} mit offensichtlicher Bedeutung:



- Wie ändern sich die Zustände? Klar!
- Welches ist der Startzustand? q_{gg}
- Welche Zustände sollen Akzeptieren bewirken? q_{gg}

Zwischenfrage

PINGO-Frage: `pingo.upb.de`

Wie viele Zustände benötigt ein DFA, der genau die Strings über $\{a, b\}$ akzeptiert, die mit a beginnen und gerade viele b 's haben?

- (A) 3
- (B) 4
- (C) 5
- (D) 6

Inhalt

2.1 Endliche Automaten: Definition und Konstruktion

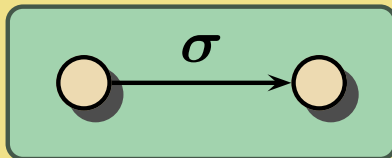
▷ **2.2 Nichtdeterministische endliche Automaten**

2.3 Von regulären Ausdrücken zu nichtdeterministischen Automaten

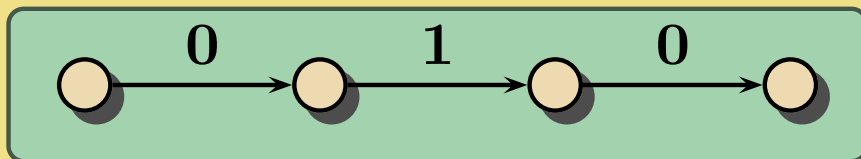
Von REs zu DFAs: Grundideen

- **Unser Ziel:** Wir suchen eine Methode zur Umwandlung von REs in Automaten
- Die Grundideen dafür sind sehr einfach und werden hier zunächst anhand von Automatenfragmenten vorgestellt
- Im Prinzip sollen REs induktiv in DFAs umgewandelt werden

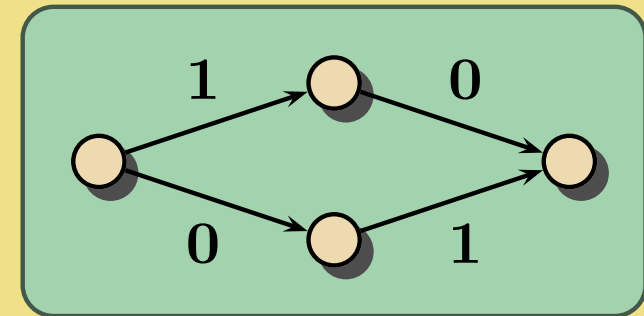
- Ein Zeichen σ eines regulären Ausdrucks wird in eine einzelne Transition übersetzt:



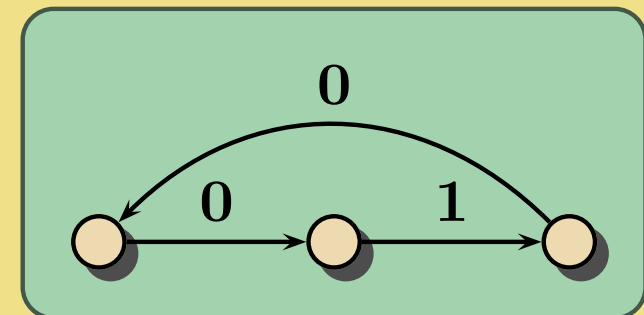
- Die **Konkatenation** von Zeichen entspricht der Hintereinanderausführung von Transitionen, z.B. für 010:



- Die **Auswahl** in REs entspricht einer Verzweigung im Automaten, z.B. ergibt sich für $10 + 01$:



- Die **Iteration** entspricht einer Schleife im Automaten: z.B. ergibt sich für $(010)^*$:



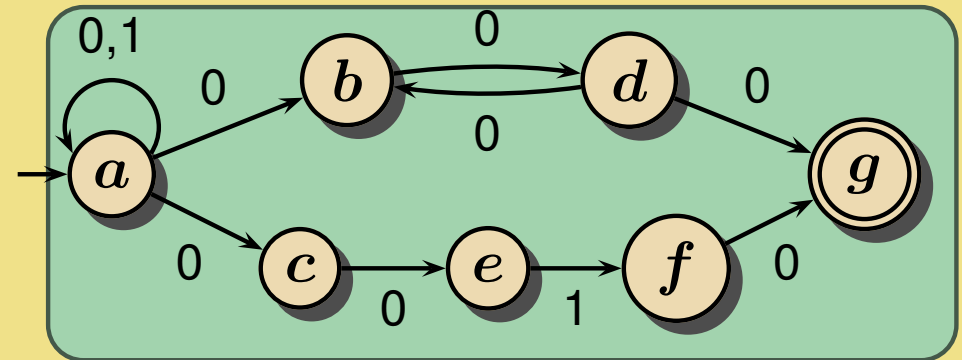
Von REs zu DFAs: Schwierigkeiten und Lösungsansatz

Schwierigkeiten

- Die Umsetzung der Auswahl $10 + 01$ ist einfach, da die beiden Teilausdrücke mit verschiedenen Zeichen anfangen
- Aber wie soll eine **Auswahl** wie $((00)^+ + 001)$ umgesetzt werden?
 - Soll eine gelesene 0 als erstes Zeichen von $(00)^+$ oder als erstes Zeichen von 001 angesehen werden? 2.2
- Und wie soll die **Konkatenation einer Iteration** mit einem einzelnen Zeichen wie $(0 + 1)^*0$ umgesetzt werden?
 - Soll eine 0 als Zeichen von $(0 + 1)^*$ oder als abschließende 0 betrachtet werden?
- Beide Schwierigkeiten kommen in dem Ausdruck $(0 + 1)^*((00)^+ + 001)0$ kombiniert vor

Lösungsansatz

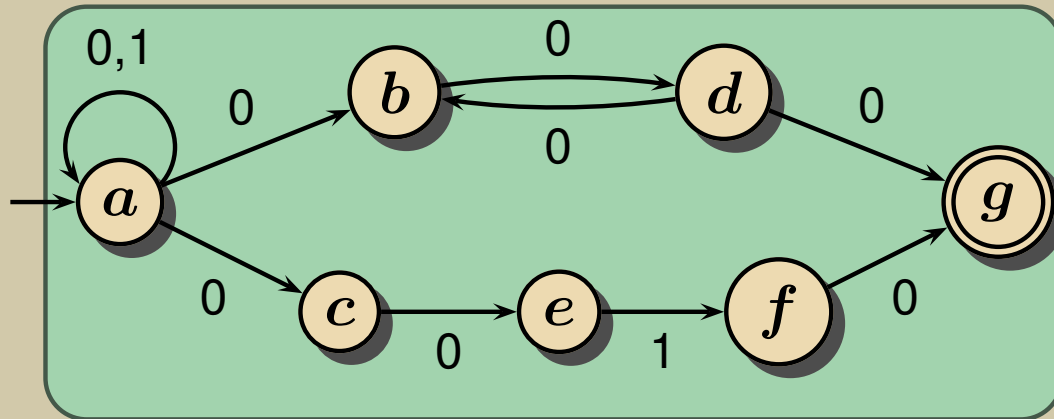
- Wir erweitern unser Automatenmodell und erlauben, dass ein Automat in einem Zustand mehrere Transitionen für das selbe gelesene Zeichen hat
- Der Ausdruck $(0 + 1)^*((00)^+ + 001)0$ lässt sich dann übersetzen in:



- Ein solcher Automat kann für dieselbe Eingabe verschiedene Berechnungen haben
- Wie soll dann definiert sein, dass er eine Eingabe akzeptiert?

Nichtdeterministische Endliche Automaten

Beispiel



0010010

- Wir sagen, „**der Automat akzeptiert ein Wort w** “, falls eine Berechnung **existiert**, in der w vollständig gelesen und dann ein akzeptierender Zustand erreicht wird
→ **nichtdeterministisches Akzeptieren**

Definition

- Ein nichtdeterministischer endlicher Automat $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ besteht aus
 - einer Zustandsmenge Q ,
 - einem Eingabealphabet Σ ,
 - einem Anfangszustand $s \in Q$,
 - einer Menge F akzeptierender Zustände,
 - sowie einer **Transitionsrelation**
$$\delta \subseteq Q \times \Sigma \times Q$$

Beispiel (Forts.)

- Im Beispiel enthält δ unter anderem:
 - $(d, 0, b)$
 - $(d, 0, g)$
 - $(e, 1, f)$
 - $(a, 1, a)$aber kein Tupel der Art (g, σ, p) für ein $\sigma \in \{0, 1\}$ und $p \in Q$

Nichtdeterministische Automaten: Notation


- **NFA:** Abkürzung für den Begriff „nichtdeterministischer Automat“
(**non-deterministic finite automaton**)
- Statt $(p, \sigma, q) \in \delta$ verwenden wir häufig die intuitivere Notation $p \xrightarrow{\sigma} q$
 - Um zu betonen, dass es sich um eine Transition im Automaten \mathcal{A} handelt, schreiben wir manchmal auch $p \xrightarrow{\sigma, \mathcal{A}} q$
- Zur Vorbereitung für die Definition der Semantik von NFAs formalisieren wir zunächst den informellen Begriff „Berechnung“ durch den formalen Begriff **Lauf**

NFAs: Semantik (1/2)

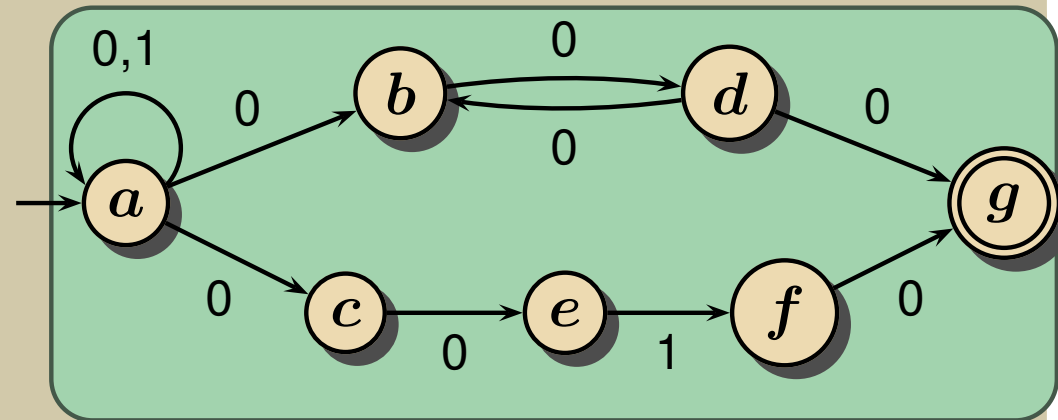
Definition: Lauf eines NFA

- Ein **Lauf** ρ eines NFAs $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ ist eine Folge der Art $q_0, \sigma_1, q_1, \dots, \sigma_n, q_n$, wobei
 - für alle $i \in \{0, \dots, n\}$: $q_i \in Q$,
 - für alle $i \in \{1, \dots, n\}$: $\sigma_i \in \Sigma$, und
 - für alle $i \in \{1, \dots, n\}$: $q_{i-1} \xrightarrow{\sigma_i} q_i$
- Wir sagen: ρ ist ein Lauf von q_0 nach q_n , der den String $w = \sigma_1 \cdot \dots \cdot \sigma_n$ liest

- Statt $\rho = q_0, \sigma_1, q_1, \dots, \sigma_n, q_n$ schreiben wir meist $\rho = q_0 \xrightarrow{\sigma_1} q_1 \cdot \dots \cdot q_{n-1} \xrightarrow{\sigma_n} q_n$
- Abkürzende Schreibweise:
 - $\underline{p \xrightarrow{w} q} \stackrel{\text{def}}{\iff}$ es gibt einen Lauf von p nach q , der den String w liest

 Spezialfall $n = 0$: q ist ein Lauf für das leere Wort

Beispiel



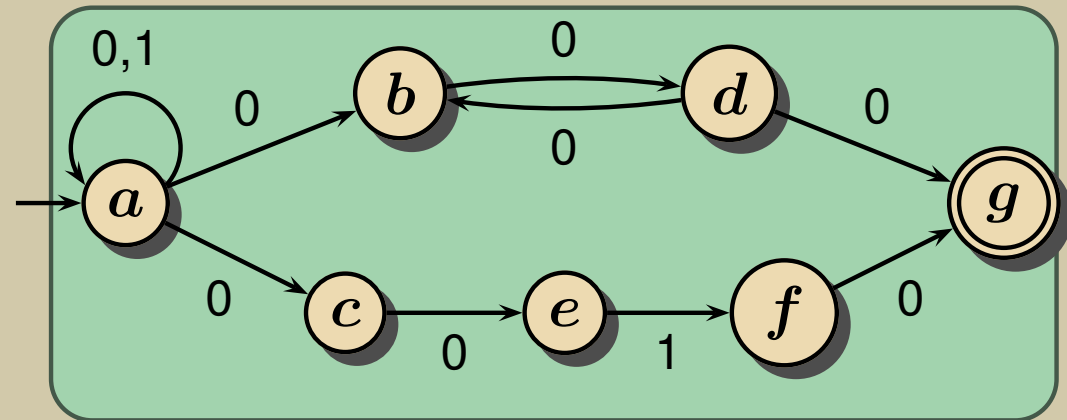
- $a, 0, b, 0, d, 0, b, 0, d, 0, g$ ist ein Lauf von a nach g , der das Wort **000000** liest
- $a, 1, a, 1, b, 0, d$ ist kein Lauf
- $b, 0, d, 0, g$ ist ein Lauf von b nach g , der das Wort **00** liest
- In der anderen Notation:
 - $a \xrightarrow{0} b \xrightarrow{0} d \xrightarrow{0} b \xrightarrow{0} d \xrightarrow{0} g$
 - $a \xrightarrow{00000} g$
 - $b \xrightarrow{0} d \xrightarrow{0} g$

NFAs: Semantik (2/2)

Definition: Semantik von NFAs

- Sei $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ ein NFA
- Ein Lauf von s zu einem Zustand aus F heißt akzeptierend
- $L(\mathcal{A})$ ist die Menge aller Strings, für die es einen akzeptierenden Lauf von \mathcal{A} gibt:
$$L(\mathcal{A}) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid s \xrightarrow{w} q, q \in F\}$$

Beispiel



- Sei \mathcal{A} der obige NFA
- $a \xrightarrow{0} a \xrightarrow{0} b \xrightarrow{0} d \xrightarrow{0} b \xrightarrow{0} d$ ist ein nicht akzeptierender Lauf für 00000
- $a \xrightarrow{0} b \xrightarrow{0} d \xrightarrow{0} b \xrightarrow{0} d \xrightarrow{0} g$ ist ein akzeptierender Lauf für 00000
- Da es einen akzeptierenden Lauf für 00000 gibt, ist $00000 \in L(\mathcal{A})$
- Für 01001 gibt es keinen akzeptierenden Lauf, deshalb ist $01001 \notin L(\mathcal{A})$

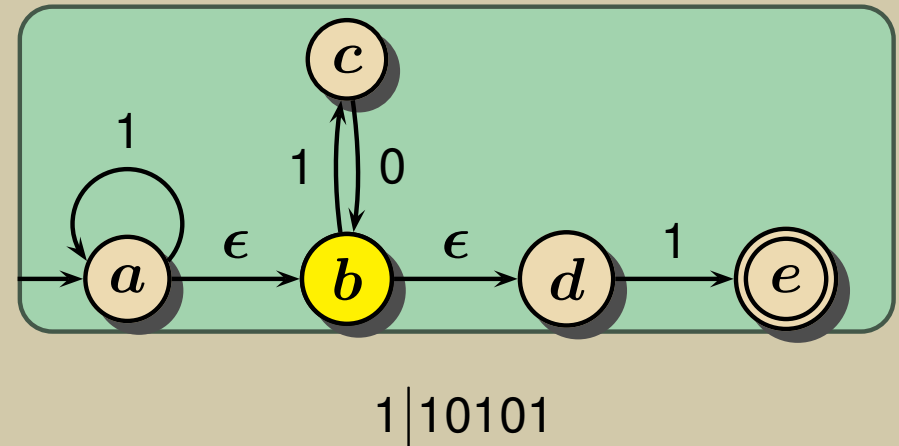
Bemerkungen zu nichtdeterministischen Automaten

- Nichtdeterminismus ist zunächst gewöhnungsbedürftig und hat für viele etwas „Beunruhigendes“
- Zur Beruhigung:
 - Nichtdeterministische Automaten sind für uns zunächst nur Mittel zum Zweck:
 - * Sie stellen einen Zwischenschritt zwischen regulären Ausdrücken und (deterministischen) endlichen Automaten dar
 - * Die am Ende resultierenden Testprogramme sind also deterministisch
 - Nichtdeterminismus ist zur Modellierung von Systemen allerdings häufig sehr hilfreich

NFA mit ϵ -Transitionen

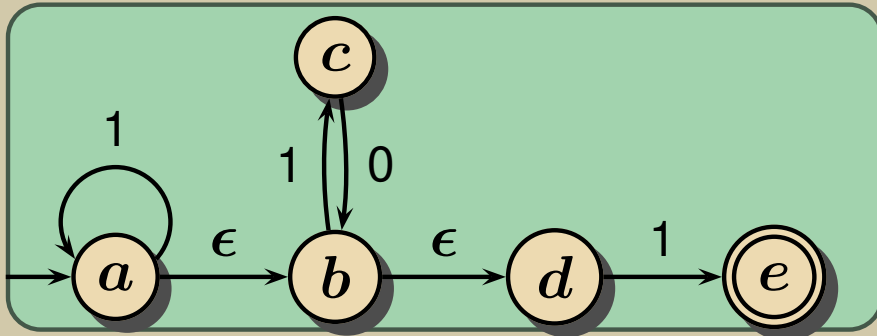
- Um den Übergang von regulären Ausdrücken (REs) zu nichtdeterministischen endlichen Automaten (NFAs) zu erleichtern, betrachten wir die Erweiterung von NFAs um **ϵ -Transitionen**
- Damit machen wir NFAs „noch nichtdeterministischer“:
 - Sie bekommen die Möglichkeit den Zustand zu wechseln, ohne ein Zeichen zu lesen
- Warum ist das praktisch?
 - Damit lassen sich Konkatenationen und Wiederholungen übersichtlicher im NFA repräsentieren
- Als Beispiel betrachten wir einen Automaten für den RE $1^*(10)^*1$

Beispiel



NFA mit ϵ -Transitionen: formal

Beispiel



Definition

- Ein nichtdeterministischer endlicher Automat $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ mit ϵ -Transitionen (ϵ -NFA) besteht aus

- einer Zustandsmenge Q ,
- einem Eingabealphabet Σ ,
- einem Anfangszustand $s \in Q$,
- einer Menge F akzeptierender Zustände,
- sowie einer **Transitionsrelation**

$$\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$$

- Im Beispiel gilt $a \xrightarrow{\epsilon} b$

- Wie lässt sich die Semantik von ϵ -NFAs definieren?

- Im Prinzip wie zuvor: aber Läufe dürfen jetzt auch Schritte der Art $p \xrightarrow{\epsilon} q$ enthalten, falls $(p, \epsilon, q) \in \delta$

- Wir überladen unsere Notation etwas und schreiben $p \xrightarrow{\epsilon} q$ auch, wenn es einen Lauf von p nach q gibt, der nur ϵ -Transitionen verwendet

- Bei einem ϵ -NFA bedeutet die Schreibweise $p \xrightarrow{w} q$, dass es einen Lauf von p nach q gibt, der w liest und zwischendurch möglicherweise auch noch ϵ -Transitionen verwendet

Inhalt

2.1 Endliche Automaten: Definition und Konstruktion

2.2 Nichtdeterministische endliche Automaten

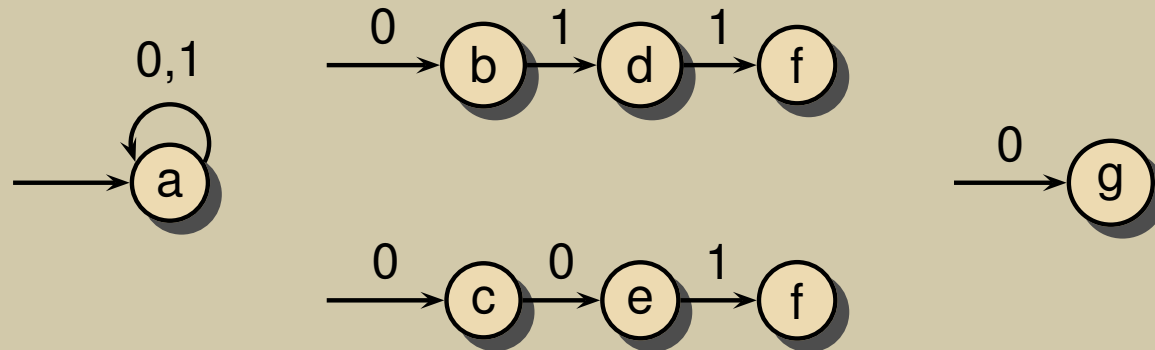
▷ **2.3 Von regulären Ausdrücken zu nichtdeterministischen Automaten**

Umwandlung RE -> NFA: Beispiel

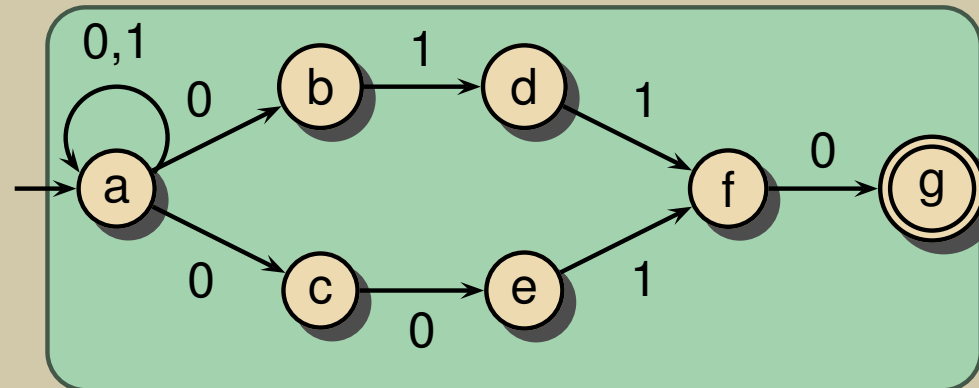
- Die Umwandlung von REs in NFAs lässt sich relativ einfach **induktiv** umsetzen (im Beispiel sogar ohne ϵ -Transitionen)

Beispiel

- Für $(0 + 1)^*(011 + 001)0$ ergeben sich Teilautomaten:



- ... aus denen sich der Gesamtautomat zusammensetzen lässt:



Vom RE zum ϵ -NFA (1/2)

Proposition 2.2

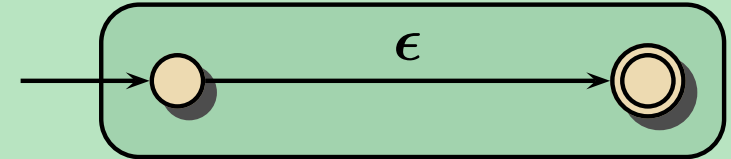
- Zu jedem RE α gibt es einen ϵ -NFA \mathcal{A} , so dass $L(\alpha) = L(\mathcal{A})$ gilt

Beweisskizze

- Wir zeigen etwas mehr: \mathcal{A} kann so konstruiert werden, dass
 - in den Startzustand keine Transitionen hineinführen, und
 - aus dem eindeutigen akzeptierenden Zustand keine Transitionen herausführen
- Also: $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ mit
 - wenn $p \xrightarrow{\sigma} q$ oder $p \xrightarrow{\epsilon} q$ dann $p \notin F$ und $q \neq s$
 - $|F| = 1$
- Wir konstruieren \mathcal{A} durch Induktion nach der Struktur von α
- Den Beweis, dass die Konstruktion korrekt ist, ersparen wir uns: er wird mit struktureller Induktion geführt

Beweisskizze (Forts.)

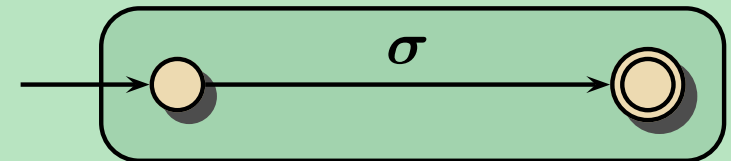
- $\alpha = \epsilon$:



- $\alpha = \emptyset$:



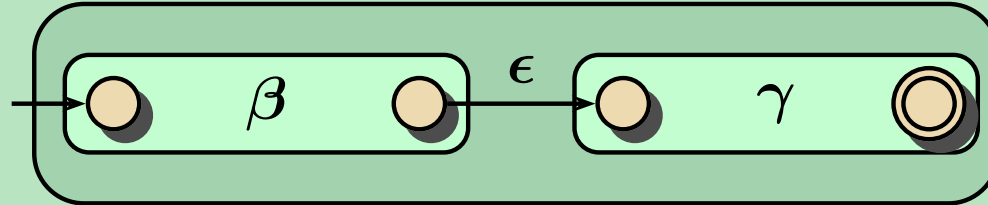
- $\alpha = \sigma$:



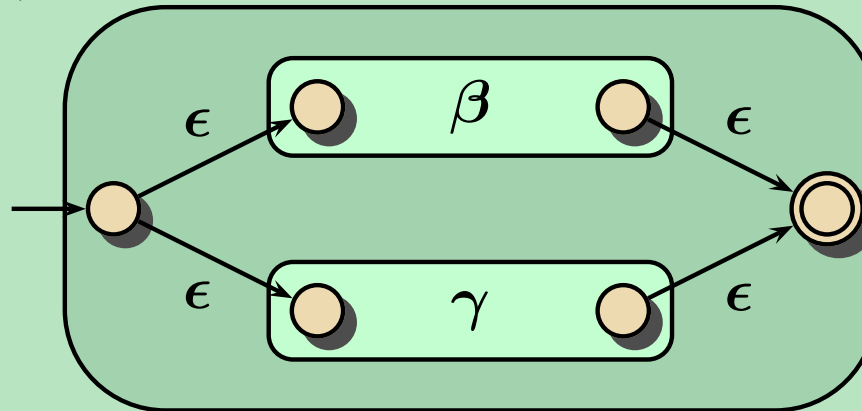
Vom RE zum ϵ -NFA (2/2)

Beweisskizze (Forts.)

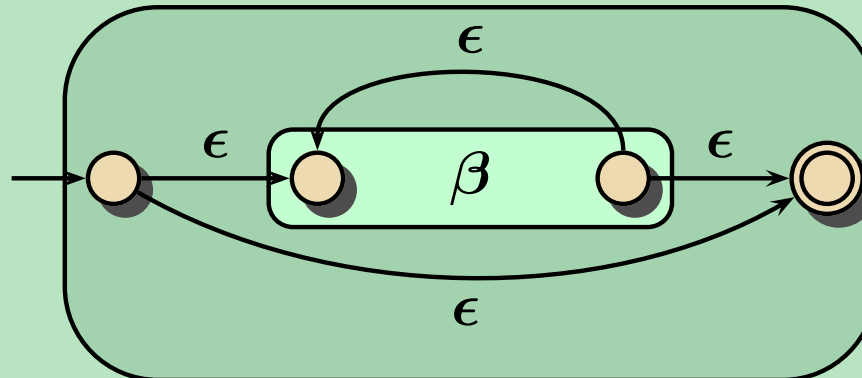
- $\alpha = \beta\gamma$:



- $\alpha = \beta + \gamma$:



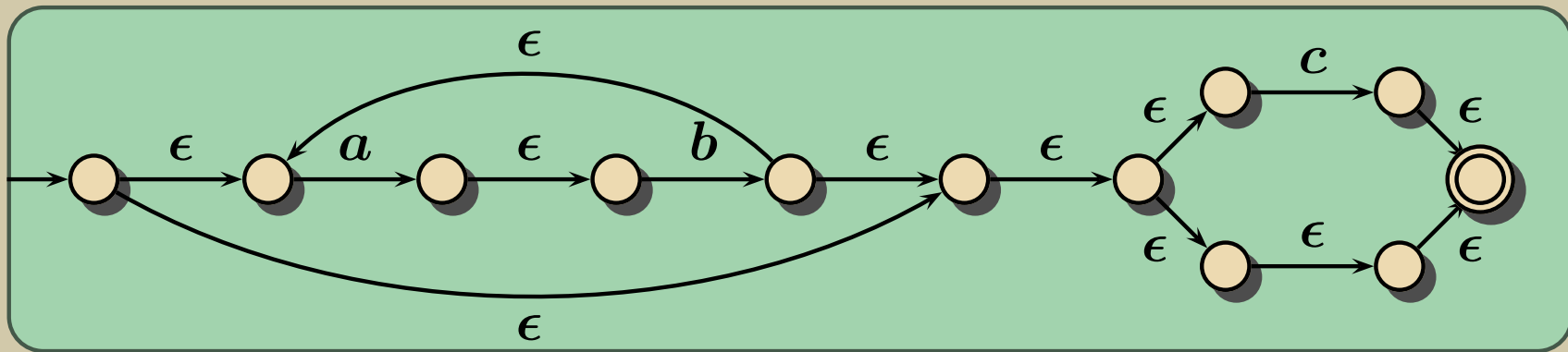
- $\alpha = \beta^*$:



Vom RE zum ϵ -NFA: Beispiel

Beispiel

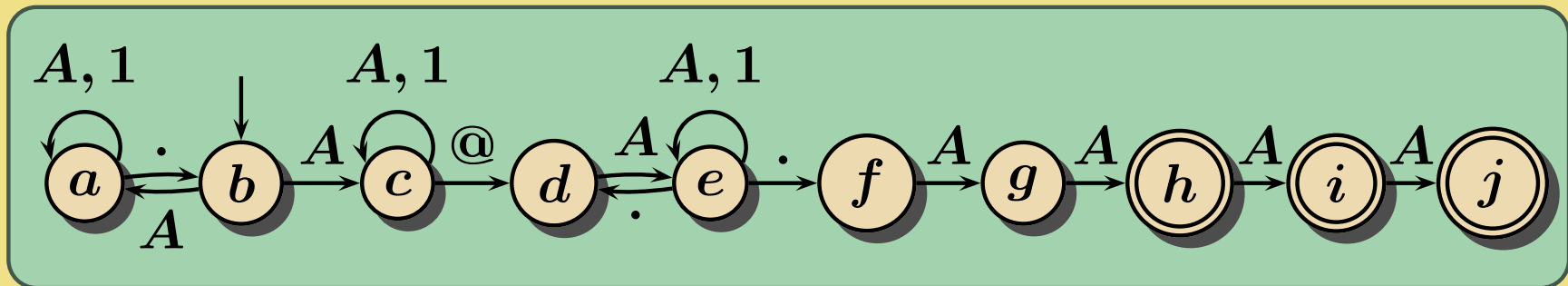
- Bei der Umwandlung des regulären Ausdrucks $(ab)^*(c+\epsilon)$ nach der beschriebenen Methode erhalten wir:



E-Mail-Adressen: Vom RE zum NFA

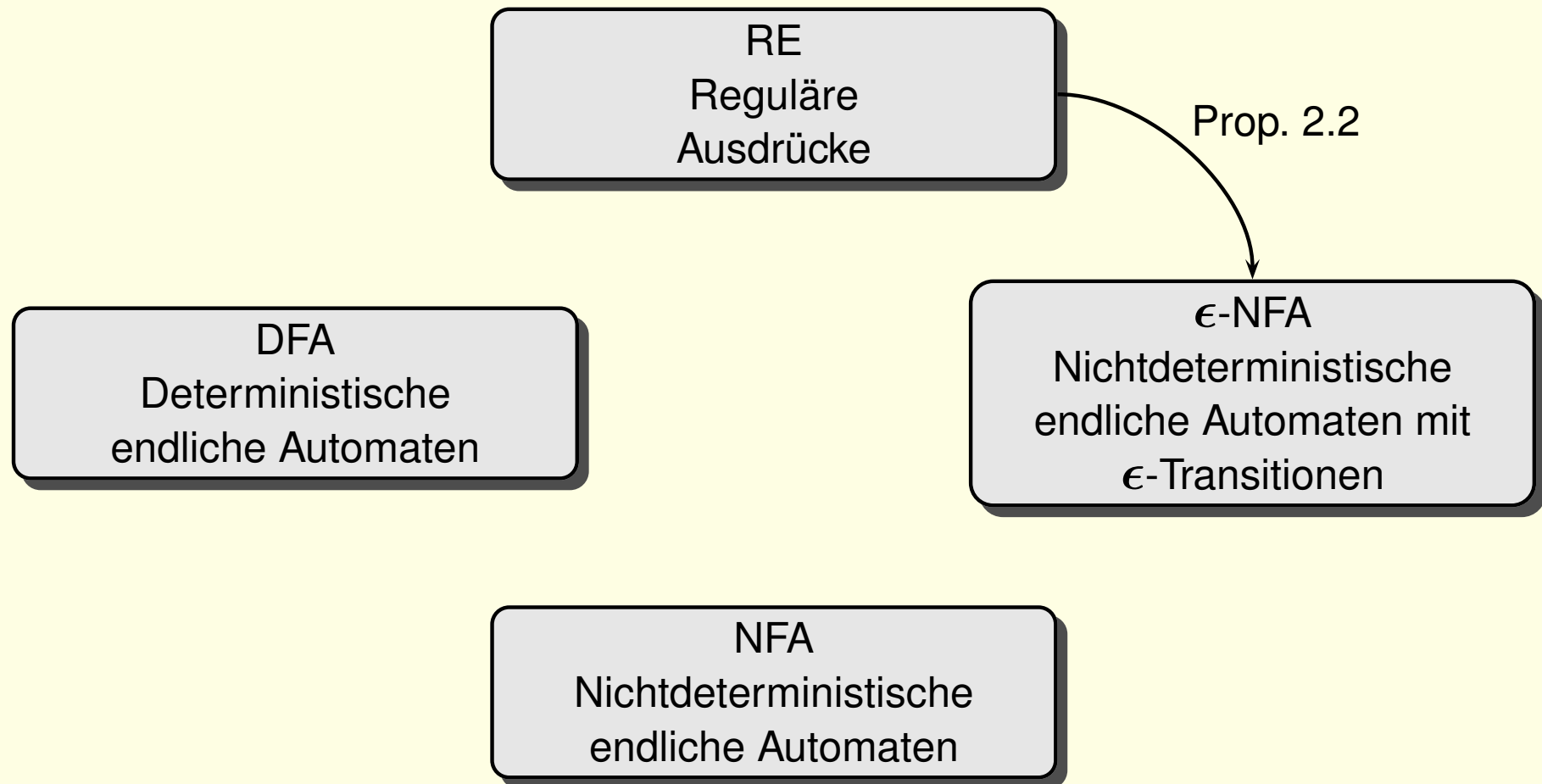
- Jetzt können wir also einen RE automatisch in einen äquivalenten ϵ -NFA umwandeln
- Wir betrachten das Beispiel des e-Mail-Ausdrucks:
$$([a-zA-Z][a-zA-Z0-9\-_]*\cdot)^*[a-zA-Z][a-zA-Z0-9\-_]*@([a-zA-Z][a-zA-Z0-9\-_]*\cdot)^+[a-zA-Z]\{2,4\}$$
- Statt des automatisch erzeugten ϵ -NFA betrachten wir aus Platzgründen allerdings einen etwas „optimierten“ NFA
- Wir verwenden im NFA einige Abkürzungen:
 - A steht für a, \dots, z, A, \dots, Z
 - 1 steht für $0, \dots, 9$ sowie $\backslash-$ und $\backslash_$

- NFA:



- Jetzt fehlt nur noch der Schritt zum DFA...

Die bisher betrachteten Modelle



Erläuterungen

Bemerkung 2.1

- Die akzeptierenden Zustände von DFAs werden häufig auch „Endzustände“ genannt
- Diese Begriffsbildung verwenden wir in dieser Veranstaltung nicht
- Denn:
 - DFAs halten an, wenn sie das Eingabewort gelesen haben, unabhängig davon, ob sie einen akzeptierenden Zustand erreicht haben
 - Den Begriff „Endzustand“ werden wir später noch für Zustände (für andere Berechnungsmodelle) verwenden, die zum sofortigen Anhalten führen
- Wir verwenden trotzdem den üblichen Buchstaben **F** für die Menge der akzeptierenden Zustände, auch wenn er sich von **final** herleitet

Bemerkung 2.2

- Wir verwenden hier $((00)^+)$ als Abkürzung für $00(00)^*$

δ : Funktion oder Relation?

- Wir bezeichnen mit δ in DFAs eine Funktion, in NFAs eine Relation
 - In DFAs könnten wir δ auch als Relation schreiben
 - * aber Funktionen sind dort intuitiver, da es zu jedem $p \in Q$ und $\sigma \in \Sigma$ nur genau einen Zustand q mit $(p, \sigma, q) \in \delta$ gibt
 - Umgekehrt ließe sich δ in NFAs auch als die Funktion definieren, die p und σ auf $\{q \mid (p, \sigma, q) \in \delta\}$ abbildet
 - * Für NFAs hat aber die Relationsschreibweise Vorteile

Mengen von Startzuständen

- In der Literatur werden NFAs manchmal auch mit einer Menge **I** von Startzuständen definiert

Zusammenfassung

Themen dieses Kapitels

- Definition von endlichen Automaten, deterministisch und nichtdeterministisch
- Semantik von endlichen Automaten
- Konstruktion endlicher Automaten

Kapitelfazit

- Endliche Automaten sind die Abstraktion von Programmen einer sehr einfachen Struktur
- Um reguläre Ausdrücke in DFAs umzuwandeln, sind nichtdeterministische endliche Automaten ein hilfreicher Zwischenschritt
- REs lassen sich leicht in ϵ -NFAs umwandeln