Lexikalische Analyse

Inhalt

- Theoretische Grundlagen
- Reguläre Sprachen und Ausdrücke
- Nichtdeterministische endliche Automaten (NEA)
- Reguläre Ausdrücke und NEAs
- Deterministische endliche Automaten (DEA)
- Spezifikation der lexikalischen Analyse
  - Zeichenklassen
  - Folgen regulärer Definitionen
  - Allbut-Konstrukt
- Generierung eines Scanners
- Implementierung eines Scanners

Lernziele

- Die theoretischen Grundbegriffe der lexikalischen Analyse (reguläre Ausdrücke, (nicht-)deterministische endliche Automaten) und ihre Zusammenhänge erklären können
- Aus regulären Ausdrücken minimale, deterministische endliche Automaten konstruieren können
- Typische Beispiele von (syntaktischen) Erweiterungen für reguläre Ausdrücke benennen und erklären können, wie man diese in den Scanner-Generierungsprozess integriert
- Verschiedene (effiziente) Implementierungsmöglichkeiten für Scanner beschreiben können

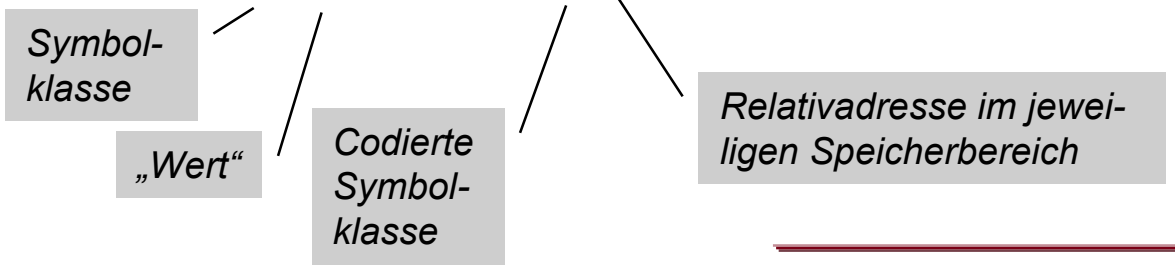
Theoretische Grundlagen (1/2)

Aufgabe der lexikalischen Analyse

- Zerlegung einer Zeichenfolge in eine Folge von lexikalischen Einheiten (*Symbole* ☛), die endlich vielen Klassen (*Symbolklassen* ☛) zugeordnet werden

Begriffe

- **Symbole**
  - Worte über einem Alphabet
  - z.B. xyz12, 123, **begin**, :=
- **Symbolklassen**
  - Mengen gleichartiger Symbole
  - z.B. Identifier (id), Integer-Konstanten (intconst), Zeichenketten (string)
- Darstellung von Symbolen
  - Geeignete Repräsentation der Symbole für die Weiterverarbeitung
  - z.B. (id, xyz12) oder (1,17) für xyz12



Theoretische Grundlagen (2/2)

Worte und Sprachen

- Sei Σ beliebiges (endliches) Alphabet
- **Worte** über Σ (= endliche Folgen von Zeichen aus Σ)
  - $x_1x_2... x_n$ Wort der Länge n ($n \geq 1$, $x_i \in \Sigma$)
  - ε leeres Wort (mit Länge 0)
- Mengen von Worten über Σ
  - $\Sigma^n =_{def} \{x_1x_2... x_n \mid x_i \in \Sigma\}$ Menge aller Worte der Länge n
  - $\Sigma^0 =_{def} \{\varepsilon\}$ Menge, die nur das leere Wort enthält
  - $\Sigma^* =_{def} \bigcup_{n\geq 0} \Sigma^n$ Menge aller Worte über Σ
  - $\Sigma^+ =_{def} \bigcup_{n\geq 1} \Sigma^n$ Menge aller nicht-leeren Worte über Σ
- **Konkatenation** von Worten x und y
  - $xy =_{def} x_1 x_2... x_n y_1 y_2... y_m$
    wenn $x = x_1 x_2... x_n$ und $y = y_1 y_2... y_m$
- Sei w = xyz
  - x **Präfix** von w (**echtes Präfix** $\Leftrightarrow x \neq \varepsilon \wedge x \neq w$)
  - z **Suffix** von w (**echtes Suffix** $\Leftrightarrow z \neq \varepsilon \wedge z \neq w$)
  - y **Teilwort** von w (**echtes Teilwort** $\Leftrightarrow y \neq \varepsilon \wedge y \neq w$)

Σ^* bildet mit Konkatenation ein Monoid
– ε ist neutrales Element
– Konkatenation ist assoziativ

Reguläre Sprachen und Ausdrücke (1/3)

Formale Sprache (über Σ)

- Teilmenge von Σ^*
- Operationen auf formalen Sprachen L, L_1, L_2
  - $L_1 \cup L_2$ — **Vereinigung** von Sprachen
  - $\mathbf{C}(L) =_{def} \Sigma^* \setminus L$ — **Komplement** einer Sprache
  - $L_1 L_2 =_{def} \{xy \mid x \in L_1, y \in L_2\}$ — **Konkatenation** von Sprachen
  - $L^n =_{def} \{x_1 x_2 \ldots x_n \mid x_i \in L, 1 \le i \le n\}$
  - $L^* =_{def} \bigcup_{n \ge 0} L^n$ — **Abschluss** einer Sprache (mit $L^0 =_{def} \{\varepsilon\}$)

Reguläre Sprachen und Ausdrücke (2/3)

Reguläre Sprache, regulärer Ausdruck (RA)

- **Reguläre Sprachen** über Σ
  - $\varnothing$, $\{\varepsilon\}$ und $\{a\}$ (für alle $a \in \Sigma$) sind reguläre Sprachen
  - Sind R, R_1 und R_2 reguläre Sprachen, dann auch $R_1 \cup R_2$, $R_1 R_2$, und R^*
  - Nichts sonst ist eine reguläre Sprache über Σ
- **Reguläre Ausdrücke** über Σ

  Äquivalente Beschreibungsformalismen: (links-/rechts-)reguläre bzw. Typ-3-Grammatiken
  - $\underline{\varnothing}$, ε und a (für $a \in \Sigma$) sind reguläre Ausdrücke
    und beschreiben die regulären Sprachen $\varnothing$, $\{\varepsilon\}$ und $\{a\}$
  - Sind r, r_1 und r_2 reguläre Ausdrücke, die die regulären Sprachen R, R_1 und R_2 beschreiben, dann auch $\underline{(}r_1 \underline{|} r_2\underline{)}$, $\underline{(}r_1\ r_2\underline{)}$ und $\underline{(}r\underline{*)}$,
    die die regulären Sprachen $R_1 \cup R_2$, $R_1 R_2$, und R^* beschreiben
  - Nichts sonst ist ein regulärer Ausdruck über Σ

Reguläre Sprachen und Ausdrücke (3/3)

Bemerkungen zur Notation

- Zeichen der Beschreibungssprache (d.h. $\underline{\varnothing}$, $\underline{|}$, $\underline{(}$, $\underline{)}$, $\underline{*}$) sind unterstrichen
  (entfällt, wenn keine Mehrdeutigkeiten möglich sind)
- Präzedenzen zur Klammereinsparung:
  Stern vor Konkatenation vor Alternativzeichen

d.h., z.B.
a*b|c = (((a*)b)|c)

- Alle Operatoren sind links-assoziativ (d.h. „Abarbeitung von links“)

Beispiele

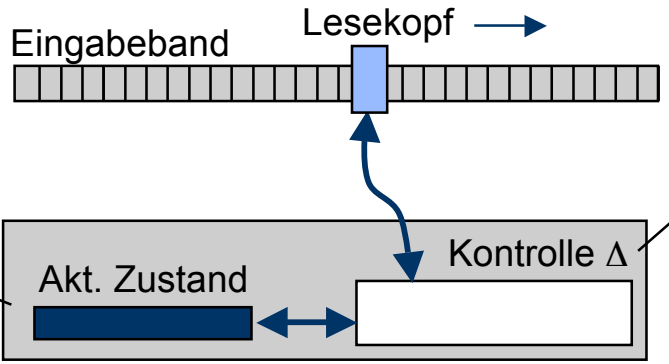
Regulärer Ausdruck	beschriebene reguläre Sprache	Elemente der regulären Sprache
a | b	{a, b}	a, b
ab*a	{a}{b}*{a}	aa, aba, abba, abbba, ...
(ab)*	{ab}*	ε, ab, abab, ...
abba	{abba}	abba
a*b|c	{a}*{b} ∪ {c}	c, b, ab, aab, aaab, ...

Endlicher Automat

- (mathematische) Maschine zum Erkennen (*Akzeptor*) regulärer Sprachen
- Charakterisiert durch
  - Lesekopf zum zeichenweisen Lesen des Eingabebandes (von links nach rechts)
  - 1 Variable („Akt. Zustand“), die nur endlich viele Zustände annehmen kann
  - Übergangsrelation Δ zur Kontrolle
    (legt neuen Zustand + Position Lesekopf fest)

Kein Rücksetzen

Beschränkte „Merkfähigkeit“



Merkt jeweils aktuellen Zustand

Im wesentlichen eine Zuordnung
Zustand X Zeichen → Zustand,
meist dargestellt durch Übergangstabelle

Nichtdeterministischer endlicher Automat (NEA)

- Tupel M = $(\Sigma, Q, \Delta, q_0, F)$, wobei
  - Σ (endliches) Eingabealphabet
  - Q endliche Menge von Zuständen
  - $q_0 \in Q$ Anfangszustand
  - $F \subseteq Q$ Menge der Endzustände
  - $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ Übergangsrelation

Auch $(q, \varepsilon, p) \in \Delta$ *möglich:*
⇒ *„spontaner Übergang“*

NEA (1/2)

Übliche Arbeitsweise eines NEA

- 1. Start im Anfangszustand;
  Lesekopf am Anfang des Eingabebandes
- 2. Folge von Schritten, solange Übergänge möglich sind;
  In jedem Schritt:
  - Einnehmen eines neuen Zustands („Folgezustand“)
  - Falls dabei Eingabezeichen gelesen, dann Lesekopf auf nächstes Zeichen, sonst nichts
- 3. Automat stoppt, wenn
  - die Eingabe erschöpft ist **oder**
  - kein weiterer Zustandsübergang möglich ist
- 4. Das Eingabewort ist *akzeptiert*, wenn
  - die Eingabe erschöpft ist **und**
  - der aktuelle Zustand ein Endzustand ist

NEA (2/2)

Arbeitsweise eines NEA als Scanner

- 1. Start im Anfangszustand
  Lesekopf auf dem ersten noch nicht „konsumierten“ Zeichen
- 2. Folge von Schritten, solange Übergänge möglich sind; In jedem Schritt:
  - Einnehmen eines neuen Zustands („Folgezustand“)
  - Falls dabei Eingabezeichen gelesen, dann Lesekopf auf nächstes Zeichen, sonst nichts
- 3. Scanner meldet
  - Das Finden eines Symbols
    - Wenn der aktuelle Zustand ein Endzustand ist und unter dem nächsten Eingabezeichen kein Übergang möglich ist
    - Wenn der aktuelle Zustand kein Endzustand ist, unter dem nächsten Eingabezeichen kein Übergang möglich ist, aber durch „Zurücksetzen“ ein Endzustand erreicht wird
  - Fehler, sonst

Strategie:
Längstes Präfix erkennen

Wesentlicher Unterschied

- Allgemeiner NEA: erkennt ganze Wörter
- NEA als Scanner: erkennt Teile von Zeichenreihen als Wörter

„Verhalten“ eines NEA (1/2)

Verhalten eines NEA

- Sei $M = (\Sigma, Q, \Delta, q_0, F)$ ein NEA, $p, q \in Q$ und $v, w \in \Sigma^*$ mit v Suffix von w
- Konfigurationen von M
  - **Konfiguration**: Paar (q, v) — *Zustand, „Resteingabe“*
  - **Anfangskonfiguration**: (q_0, w)
  - **Endkonfiguration**: (q_f, ε) mit $q_f \in F$
- Schritt-Relationen (Übergang von einer Konfiguration zu einer Folgekonfiguration)
  - **Ein-Schritt-Relation**: $\vdash_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$, wobei
    $(q, av) \vdash_M (p, v) \Leftrightarrow (q, a, p) \in \Delta \wedge a \in \Sigma \cup \{\varepsilon\}$
  - **Mehr-Schritt-Relation** $\vdash_M^*$: reflexiv-transitive Hülle von $\vdash_M$
- Von M **akzeptierte Sprache**: $L(M) =_{def} \{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (q_f, \varepsilon) \text{ mit } q_f \in F\}$

Intuitiv

- Ein endlicher Automat akzeptiert solche Worte, die ihn von der jeweiligen Anfangskonfiguration in eine Endkonfiguration führen
- Ein Scanner akzeptiert alle Worte als Symbole, die ihn von seinem Anfangszustand in einen Endzustand bringen — *Endkonfiguration nicht erforderlich*

„Verhalten“ eines NEA (2/2)

Beispiel (NEA zur Erkennung von Integer- und Real-Konstanten)

- Geg.: M = ({0,1,2,...,9,.,e}, {0,1,2,...,7}, Δ, 0, {1,7}) mit

Δ =

	0,1,...,9	.	e	ε
0	{1,2}	∅	∅	∅
1	{1}	∅	∅	∅
2	{2}	{3}	∅	∅
3	{4}	∅	∅	∅
4	{4}	∅	{5}	{7}
5	{6}	∅	∅	∅
6	{7}	∅	∅	∅
7	∅	∅	∅	∅

Zustand

nächstes Eingabezeichen

entspricht Fehlerzustand (kein Übergang)

- Arbeitsweise von M

q	w
0	0.53e02
1	.53e02
2	.53e02
3	53e02
4	3e02
4	e02
5	02
6	2
7	

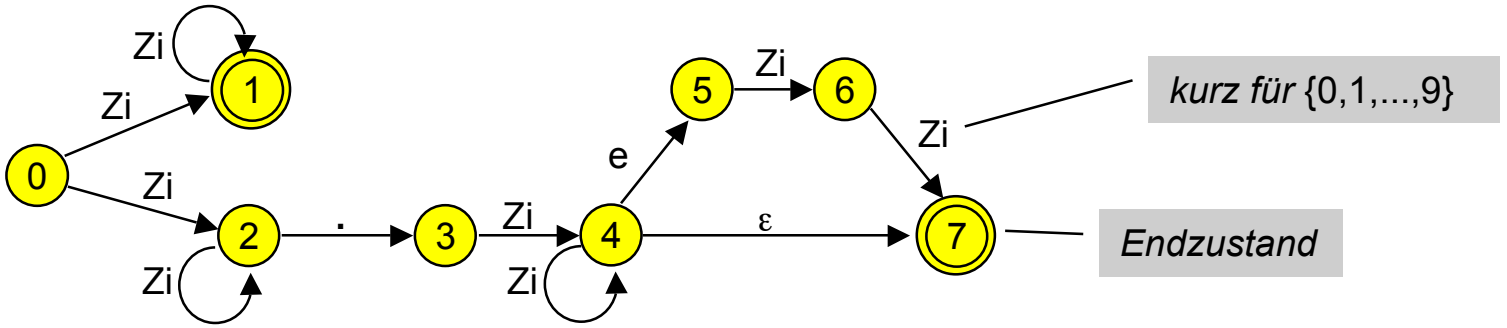
Übergangsdiagramm

Übergangsdiagramm ÜD

- Endlicher, gerichteter, kantenmarkierter Graph ÜD = (V, E, T, v_0, V_f) mit
  - V: endliche Menge von Knoten
  - E: endliche Menge von Kanten, mit Elementen aus T ∪ {ε} markiert
  - $v_0 \in V$: Startknoten
  - $V_f \subseteq V$: Menge der Endknoten
- **Akzeptierte Sprache** $L(ÜD) =_{def} \{w \in T^* \mid$ es gibt w-Weg von v_0 zu $v_f \in V_f$ in ÜD$\}$
- Zu NEA äquivalenter, alternativer Formalismus
  - Zustand — Knoten
  - Anfangszustand — Anfangsknoten
  - Endzustand — Endknoten
  - Übergang $(q, a, p) \in \Delta$ — Kante von q nach p mit Markierung a

*Weg im Graphen, so dass die Konkatenation der Kantenmarkierungen = w ∈ T**

Beispiel (ÜD zur Erkennung von Integer- und Real-Konstanten)

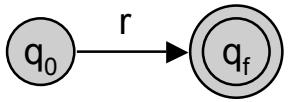


Reguläre Ausdrücke und NEAs (1/3)

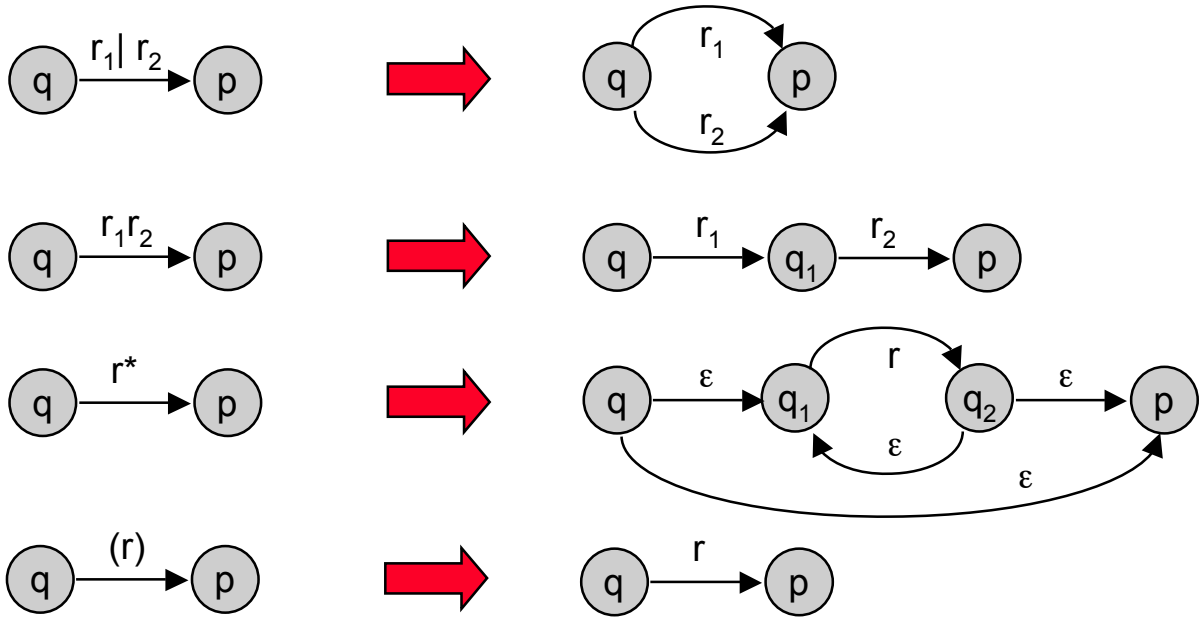
Satz

- Zu jedem RA r gibt es NEA, der die von r beschriebene reguläre Menge akzeptiert

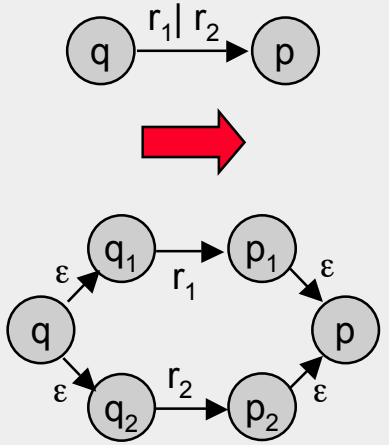
Beweis (konstruktiv)

- Start mit 

- Anwendung der folgenden Regeln (bis alle Kanten mit Einzelzeichen oder ε markiert sind)



Auch möglich
(vgl. Aho, Sethi, Ullman)

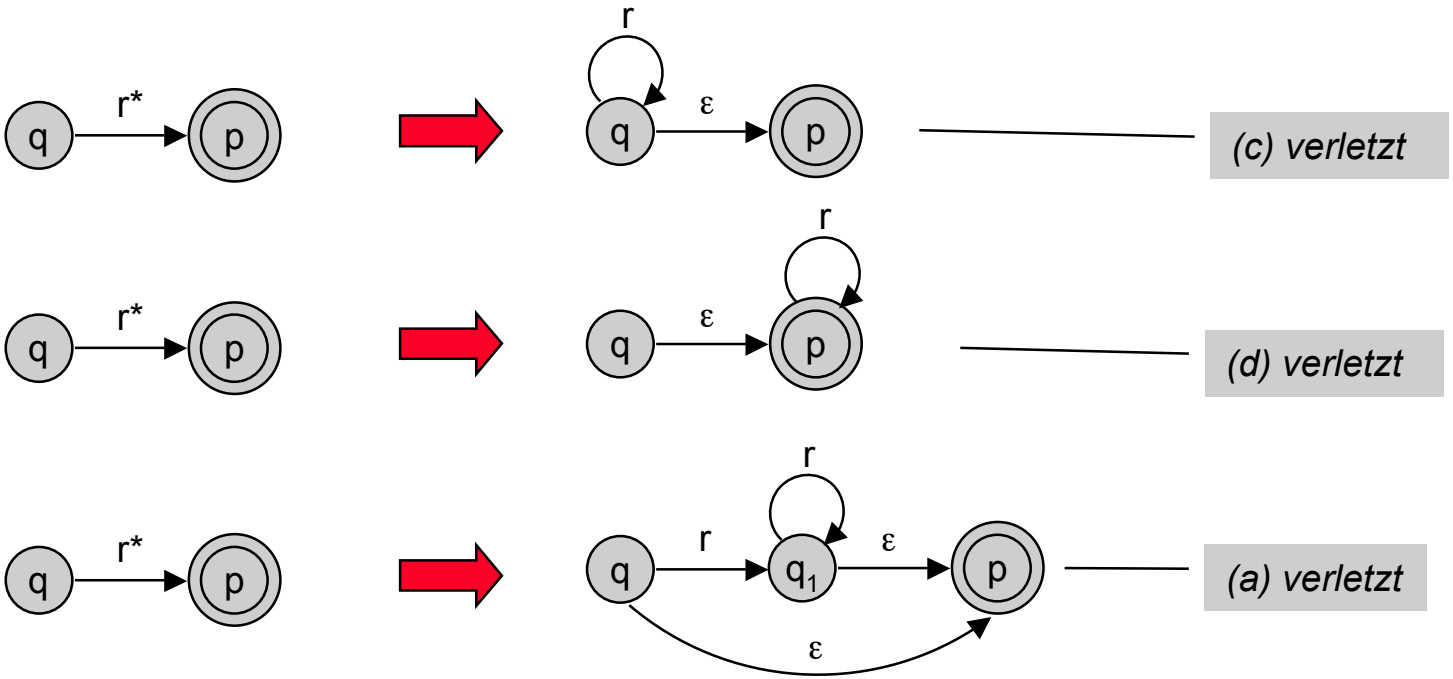


Reguläre Ausdrücke und NEAs (2/3)

Invarianten des Übersetzungsverfahrens

- (a) Höchstens doppelt so viele Zustände wie Zeichen in r
- (b) Genau ein Anfangs- und Endzustand
- (c) Anfangszustand hat keine eingehenden Kanten
- (d) Endzustand hat keine ausgehenden Kanten

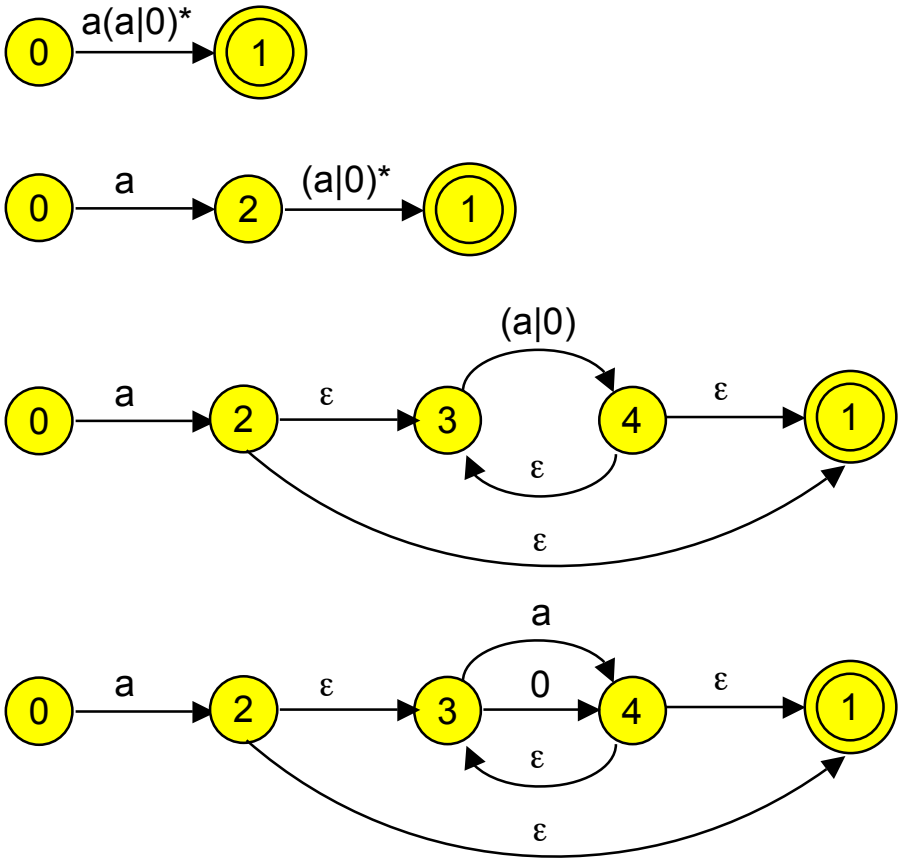
Varianten für den *-Operator



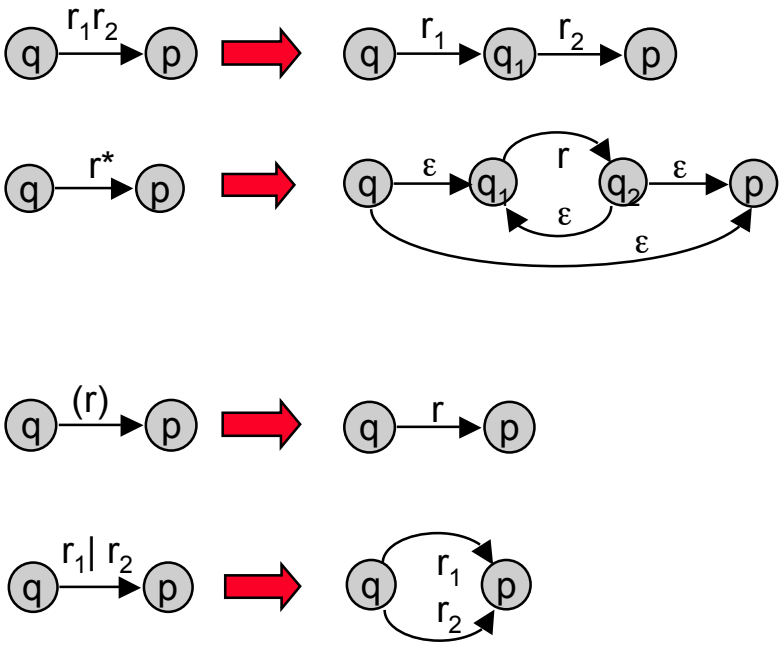
Reguläre Ausdrücke und NEAs (3/3)

Beispiel

- Geg.: regulärer Ausdruck a(a|0)*
- Konstruktion des Automaten



Angewandte Regel



Deterministischer endlicher Automat (DEA)

Deterministischer endlicher Automat (DEA)

- Sei $M = (\Sigma, Q, \Delta, q_0, F)$ ein NEA.
  M heißt **DEA** $\Leftrightarrow$ Δ ist (partielle) Funktion $\delta: Q \times \Sigma \rightarrow Q$ (d.h. rechts-eindeutige Relation)

Unterschied von DEA zu NEA

- Keine Übergänge unter ε
- Für jedes Paar (q, a) mit $q \in Q$ und $a \in \Sigma$ höchstens ein Nachfolgezustand

Übergang NEA $\rightarrow$ DEA

- Sei $M = (\Sigma, Q, \Delta, q_0, F)$ ein NEA
- **ε-Folgezustände** ε-FZ(q), $q \in Q$, $S \subseteq Q$
  - $\varepsilon\text{-FZ}(q) =_{def} \{p \mid (q,\varepsilon) \vdash_M^* (p,\varepsilon)\}$
  - $\varepsilon\text{-FZ}(S) =_{def} \cup_{q \in S} \varepsilon\text{-FZ}(q)$
- Zu M gehöriger DEA $M' =_{def} (\Sigma, Q', \delta, q_0', F')$ mit
  - $Q' \subseteq_{def} \mathbb{P}(Q)$
  - $q_0' =_{def} \varepsilon\text{-FZ}(q_0)$
  - $F' =_{def} \{S \in Q' \mid S \cap F \neq \emptyset\}$
  - $\delta(S,a) =_{def} \varepsilon\text{-FZ}(\{p \mid (q,a,p) \in \Delta \text{ für } q \in S\})$ für $a \in \Sigma$

Potenzmenge von Q

Grundidee: „Teilmengenkonstruktion“ (Myhill/Büchi)

Übergang NEA → DEA (1/2)

Satz

- Zu jedem NEA gibt es einen DEA, der dieselbe Sprache akzeptiert

Beweis (konstruktiv)

Algorithmus NEA → DEA
Eingabe: NEA M = $(\Sigma, Q, \Delta, q_0, F)$
Ausgabe: DEA M' = $(\Sigma, Q', \delta, q_0', F')$ gemäß Definition

```
q0' := ε-FZ(q0); Q' := {q0'}; marked(q0') := false; δ := ∅;
while exists S ∈ Q' and marked(S) = false do
    marked(S) := true;
    foreach a ∈ Σ do
        T := ε-FZ({p ∈ Q | (q,a,p) ∈ Δ and q ∈ S});
        if T ∉ Q'
            then  Q' := Q' ∪ {T};      (* neuer Zustand *)
                  marked(T) := false fi
        δ := δ ∪ {(S,a) ↦ T}           (* neuer Übergang *)
    od
od
```

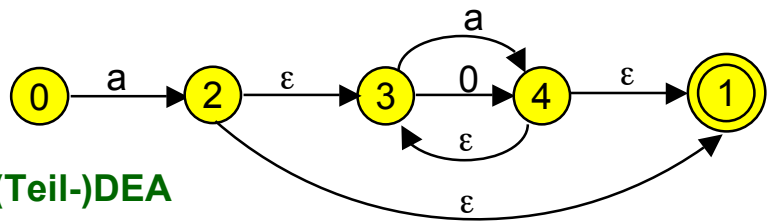
Initialisierung

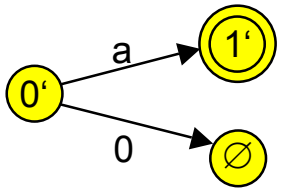
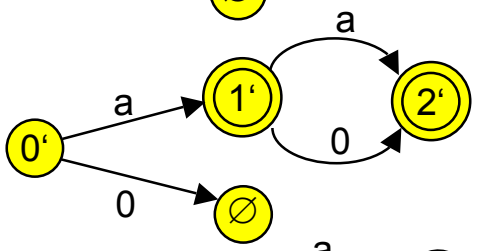
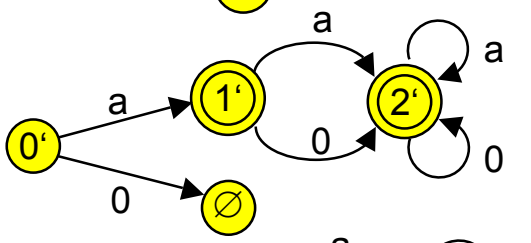
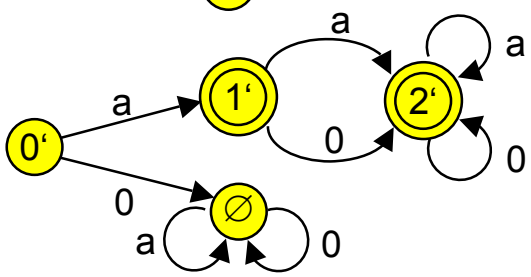
Abbruch:
Alle Zustände markiert

Übergang NEA → DEA (2/2)

Beispiel

Geg.: Σ = {0,a}; 0‘ = {0}; Q‘ = {0‘}



Ausgewählter Zustand	neues Q‘	neuer (Teil-)DEA
0‘	{0‘, 1‘, ∅} mit 1‘ = {1,2,3}	
erledigter Zustand		
1‘	{0‘, 1‘, 2‘, ∅} mit 2‘ = {1,3,4}	
2‘	{0‘, 1‘, 2‘, ∅}	
∅	{0‘, 1‘, 2‘, ∅}	
	Fehlerzustand	

Algorithmus NEA → DEA

```
q0‘ := ε-FZ(q0); Q‘ := {q0‘};
marked(q0‘) := false; δ := ∅;
while exists S ∈ Q‘ and marked(S) = false do
    marked(S) := true;
    foreach a ∈ Σ do
        T := ε-FZ({p ∈ Q | (q,a,p) ∈ Δ and q ∈ S});
        if T ∉ Q‘
            then  Q‘ := Q‘ ∪ {T};
                  marked(T) := false fi
        δ := δ ∪ {(S,a) ↦ T}
    od
od
```

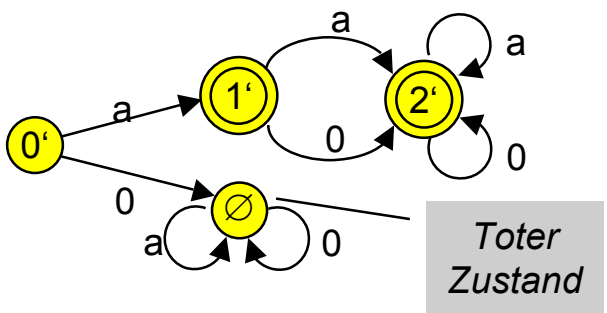
Minimaler deterministischer endlicher Automat

Satz

- Zu jedem DEA gibt es einen DEA mit minimaler Zustandsmenge (MinDEA), der dieselbe Sprache akzeptiert

Konstruktionsidee

- Zustandsmenge in Partition von Klassen von Zuständen mit „unterschiedlichem Akzeptanzverhalten“ (d.h. für mindestens 1 Wort wird aus einem der Zustände einer Klasse ein Endzustand erreicht und aus einem anderen nicht) aufteilen, die solange schrittweise verfeinert wird, bis keine Aufteilung mehr erforderlich ist
- Tote und unerreichbare Zustände eliminieren



Ursprünglicher Automat

Nicht-Endzustände

Start-Partition: {0‘, ∅}, {1‘, 2‘}

Endzustände

{0‘, ∅} muss aufgespaltet werden, da

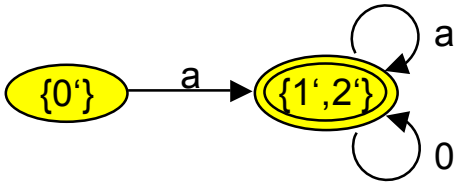
$0' \xrightarrow{a} \{1', 2'\}$

$\varnothing \xrightarrow{a} \{0', \varnothing\}$

{1‘, 2‘} nicht, da

$\{1', 2'\} \xrightarrow{a,0} \{1', 2'\}$

End-Partition: {{0‘}, {∅}, {1‘, 2‘}}



Minimaler Automat

Spezifikation der lexikalischen Analyse (1/3)

Reguläre Ausdrücke mit „syntactic sugar“

- *Zeichenklassen* ☛
- *Sequenzen regulärer Definitionen* ☛
- *Nichtrekursive Klammerung* ☛

Zeichenklassen

- Mengen von Zeichen, die in Symbolen ausgetauscht werden können, ohne die (zugehörige) Symbolklasse zu verändern

Beispiel: *eine ganze Zahl bleibt eine ganze Zahl, auch wenn man eine Ziffer ändert*

- Beispiel

Zeichenklassen

bu = a..z A..Z

zi = 0..9

Damit

zi zi* (ε | .zi zi* (ε | e zi zi))

Offensichtlich: bessere Lesbarkeit

Anstelle von

(0|1|2|3|4|5|6|7|8|9) (0|1|2|3|4|5|6|7|8|9)*
(ε | .(0|1|2|3|4|5|6|7|8|9) (0|1|2|3|4|5|6|7|8|9)*
(ε | e (0|1|2|3|4|5|6|7|8|9) (0|1|2|3|4|5|6|7|8|9)))

Spezifikation der lexikalischen Analyse (2/3)

Sequenzen regulärer Definitionen (über Σ)

- Folge von Definitionen
  - $A_1 = R_1 \quad A_2 = R_2 \quad ... \quad A_n = R_n$
  - wobei
    - $A_1, ... A_n$ paarweise verschiedene Bezeichner
    - R_i reguläre Ausdrücke über $\Sigma \cup \{A_1, ... A_{i-1}\}$ $(1 \le i \le n)$

wesentlich für azyklische Definition;
Alternative Forderung:
alle A_i lassen sich aus den R_j durch Einsetzen eliminieren

- Zu R_i korrespondierender regulärer Ausdruck
  - $R_i' =_{def} [R_1/A_1] [[R'_2/A_2] [... [R'_{i-1}/A_{i-1}] R_i ...]$ wobei
  - $[R_j/A_j]$ R: Ersetzung aller Vorkommen von A_j durch R_j in R

Beispiel

A_1

- intconst = zi zi*

realconst = intconst . intconst (e zi zi | ε)

A_2

- realconst' = [zi zi* / intconst] realconst =
  [zi zi* / intconst] intconst . intconst (e zi zi | ε) =
  zi zi* . zi zi* (e zi zi | ε)

Spezifikation der lexikalischen Analyse (3/3)

Nichtrekursive Klammerung

- Abkürzende Schreibweise für lexikalische Einheiten, die zwischen den regulären Ausdrücken R_1 und R_2 beliebige R_2 nicht enthaltende Zeichenfolgen haben dürfen

  R_1 allbut(R_2)

- Dadurch definierte reguläre Sprache (zwischen R_1 und R_2 alles aus Σ^* was R_2 nicht enthält)

  R_1 allbut(R_2) $=_{def}$ R_1 **C**($\Sigma^* R_2 \Sigma^*$) R_2 = R_1 ($\Sigma^* \setminus \Sigma^* R_2 \Sigma^*$) R_2

Beispiel

comment = (* allbut(*))

Generierung eines Scanners

Kernstück

- Erzeugung eines minimalen deterministischen endlichen Automaten aus einem regulären Ausdruck
  - In 3 (evtl. verschränkten) Schritten wie oben
  - Direkt aus dem der regulären Struktur entsprechenden „Syntaxbaum“ (Details siehe z.B. „Drachenbuch“)

Nun zu ergänzen

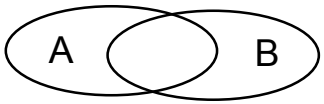
- Behandlung der syntaktischen Erweiterungen
  - Zeichenklassen
  - Folgen regulärer Definitionen
  - Nichtrekursive Klammerung

Zeichenklassen

Implementierung

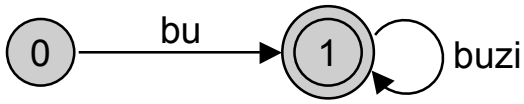
Eindeutige Zuordnung der Zechen zu ihren Klassen

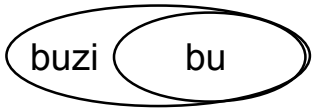
- Alphabet als Feld mit (Maschinendarstellungen der) Zeichen als Indices und Zeichenklassencodes als Komponentenwerte
- Problem: Nicht disjunkte Klassen A und B
- Lösung: Aufspaltung in A \ B, B \ A und A ∩ B

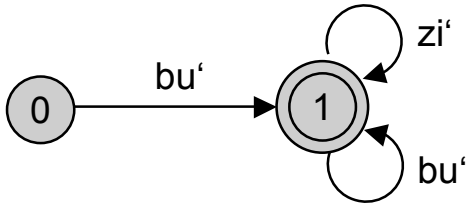


Beispiel

- MinDEA für Symbolklasse id = bu buzi*
- Zeichenklassen
  - bu = a..z
  - buzi = a..z 0..9
- Aufspaltung in
  - zi‘ = buzi \ bu = 0..9
  - bu‘ = bu ∩ buzi = bu
- Verwendung der neuen Symbolklassen







Folgen regulärer Definitionen (1/3)

Vorgehensweise

- Geg.: $A_1 = R_1 \quad A_2 = R_2 \quad \ldots \quad A_n = R_n$
- Scannergenerierung
  - 0. Umwandlung der R_i in jeweilige R'_i durch sukzessives Einsetzen
  - 1. Erzeugung von NEAs für die RAs R'_i (mit jeweils disjunkten Zustandsmengen)
  - 2. Zusammenfügen dieser NEAs über neuen Anfangszustand und ε-Übergängen zu einem NEA
  - 3. Algorithmus NEA → DEA
  - 4. Minimierung, falls nötig

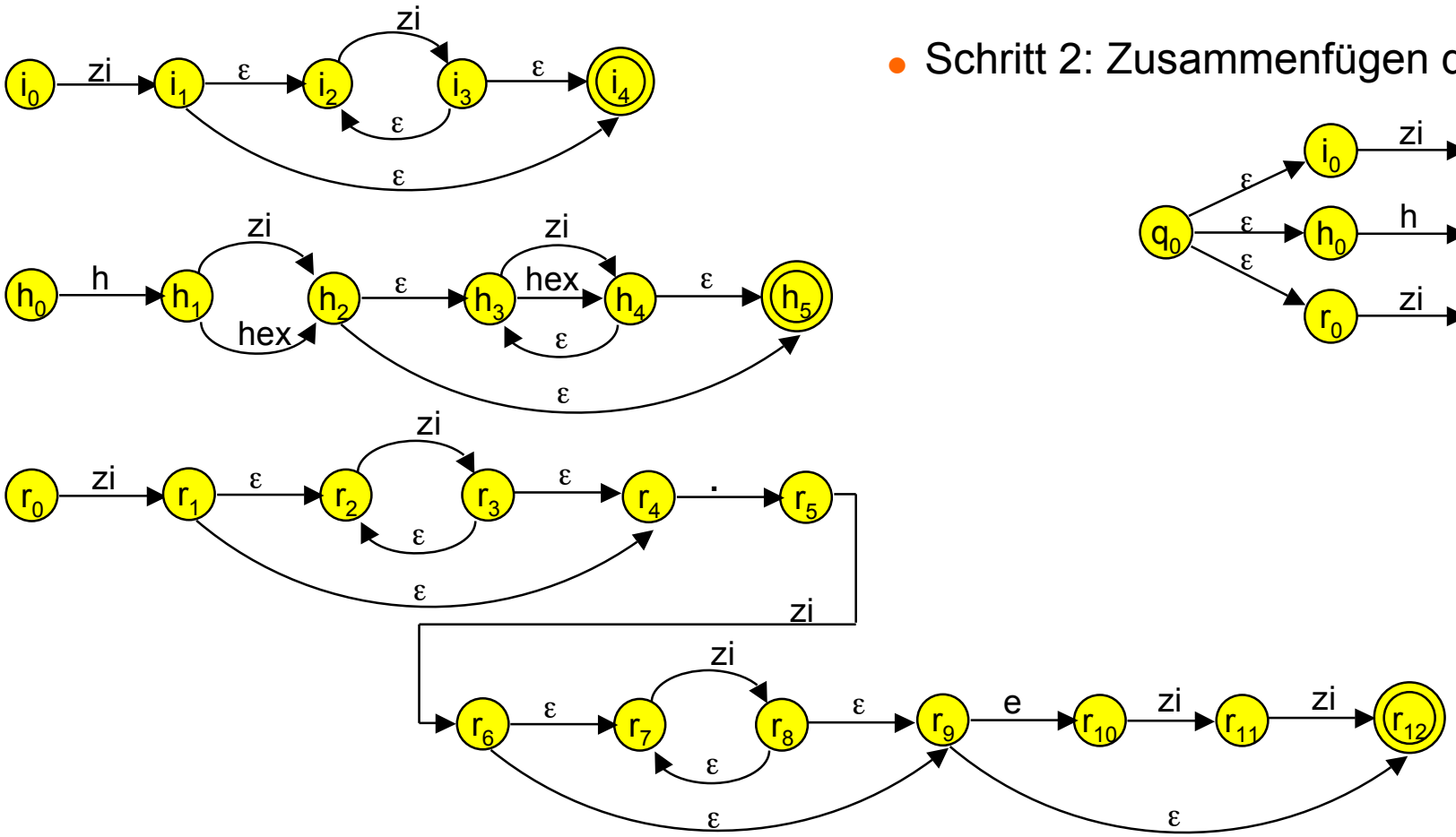
Beispiel

- Geg.:
  - zi = 0..9
  - hex = A..F
  - intconst = zi zi*
  - hexconst = h (zi | hex) (zi | hex)*
  - realconst = intconst **.** intconst (e zi zi | ε)
- Schritt 0: Ersetzung der Symbolklassennamen (rechts von =) durch ihre Definition
  - intconst = zi zi*
  - hexconst = h (zi | hex) (zi | hex)*
  - realconst = zi zi* **.** zi zi* (e zi zi | ε)

Folgen regulärer Definitionen (2/3)

Beispiel (Fortsetzung)

- Schritt 1: Erzeugung von NEAs (mit Algorithmus RA → NEA)

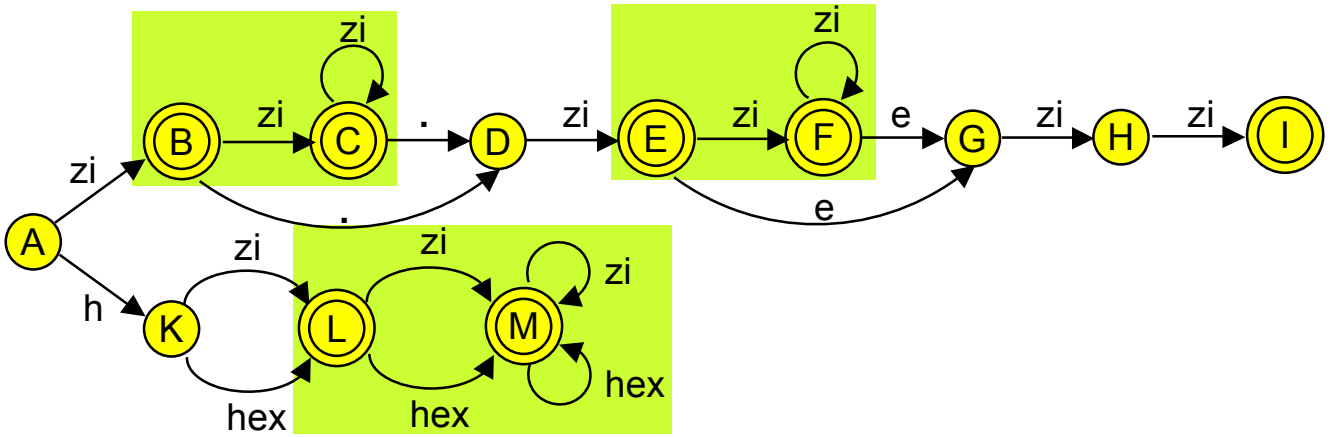


- Schritt 2: Zusammenfügen der drei NEAs

Folgen regulärer Definitionen (3/3)

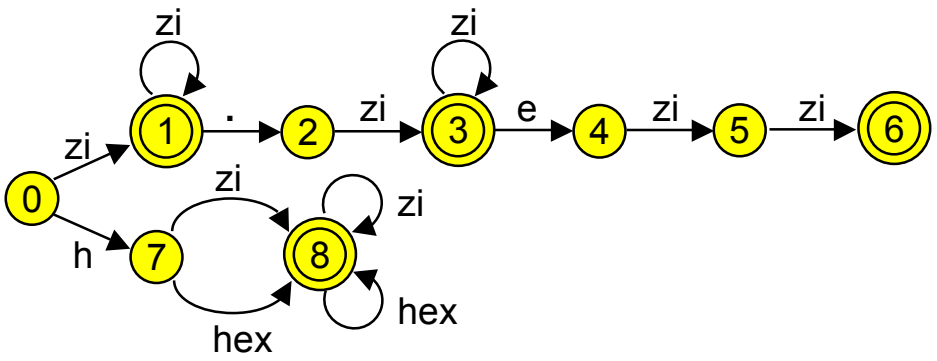
Beispiel (Fortsetzung)

- Schritt 3: Deterministisch machen dieses NEA (mit Algorithmus NEA → DEA)



Zustände
A = {q_0, i_0, h_0, r_0}
B = {i_1, i_2, i_4, r_1, r_2, r_4}
C = {i_2, i_3, i_4, r_2, r_3, r_4}
D = {r_5}
E = {r_6, r_7, r_9, r_{12}}
F = {r_7, r_8, r_9, r_{12}}
G = {r_{10}}
H = {r_{11}}
I = {r_{12}}
K = {h_1}
L = {h_2, h_3, h_5}
M = {h_3, h_4, h_5}

- Schritt 4: Minimierung



Zustände
0 = {A}
1 = {B, C}
2 = {D}
3 = {E, F}
4 = {G}
5 = {H}
6 = {I}
7 = {K}
8 = {L, M}

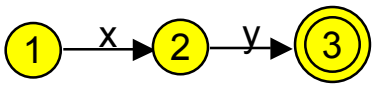
allbut-Konstrukt (1/2)

Vorgehensweise

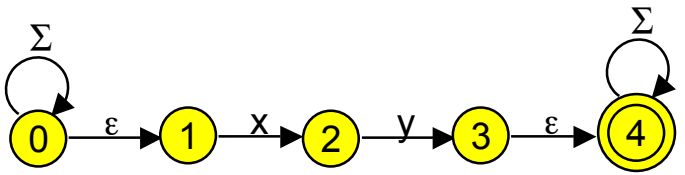
- Entsprechend der Definition: R_1 allbut(R_2) =$_{def}$ R_1 **C**($\Sigma^* R_2 \Sigma^*$) R_2
- Konstruktion des Mustererkennungsautomaten
  1. Konstruktion eines NEA für R_2
  2. Konstruktion eines NEA für $\Sigma^* R_2 \Sigma^*$
  3. Umwandlung in einen DEA
  4. Konstruktion des DEA der **C**($\Sigma^* R_2 \Sigma^*$) akzeptiert
     (Vertauschung von Endzuständen und Nicht-Endzuständen)
  5. Minimierung
  6. Konstruktion eines NEA für R_1 **C**($\Sigma^* R_2 \Sigma^*$) R_2
  7. Umwandlung in einen DEA

Beispiel

- Geg.: z allbut(xy) mit x,y,z $\in$ Σ
- 1. NEA für xy



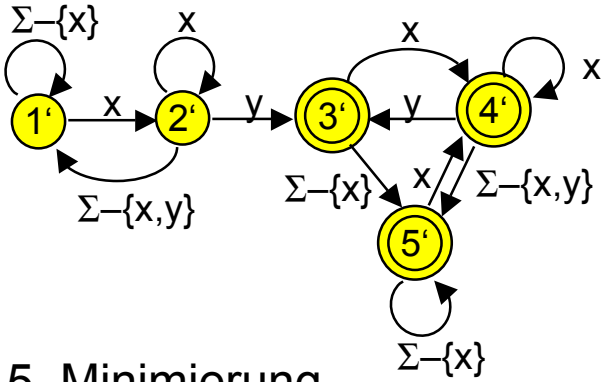
- 2. NEA für Σ^*xy Σ^*



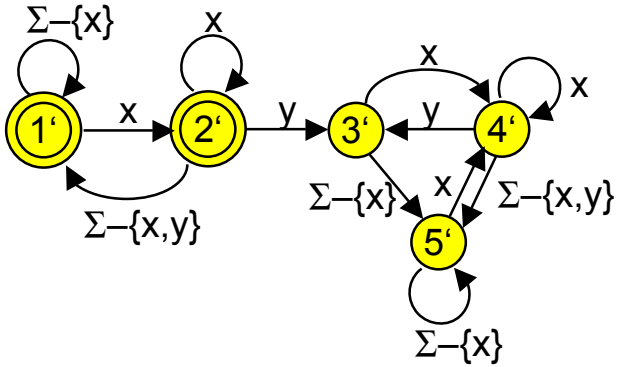
allbut-Konstrukt (2/2)

Beispiel (Fortsetzung)

- 3. DEA für $\Sigma^* xy \Sigma^*$



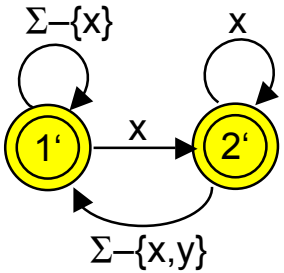
- 4. DEA für $\mathbf{C}(\Sigma^* xy \Sigma^*)$



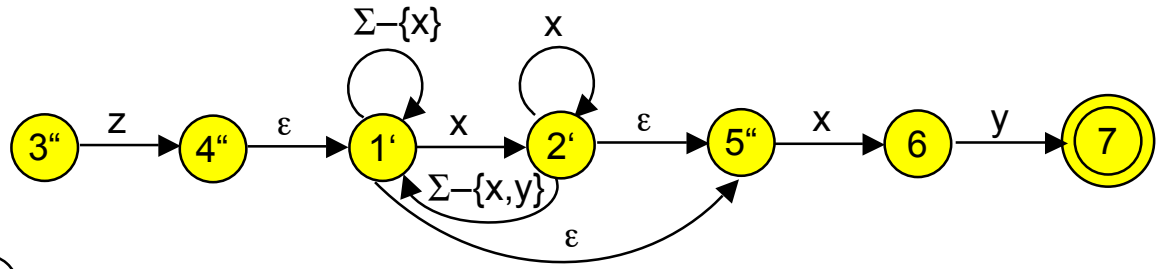
Neue Zustände

1‘ = {0,1}
2‘ = {0,1,2}
3‘ = {0,1,3,4}
4‘ = {0,1,2,4}
5‘ = {0,1,4}

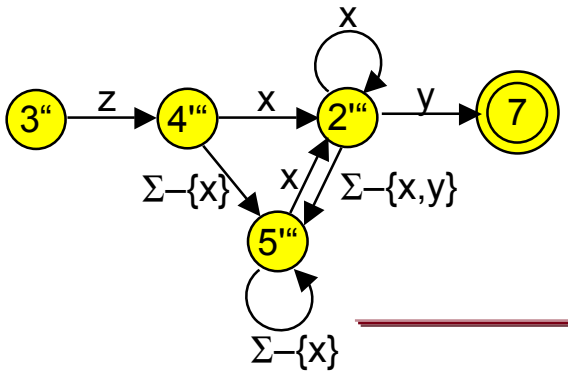
- 5. Minimierung



- 6. NEA für z $\mathbf{C}(\Sigma^* xy \Sigma^*)$ xy



- 7. DEA für z $\mathbf{C}(\Sigma^* xy \Sigma^*)$ xy



Neue Zustände

2‘‘‘ = {2‘,5“,6}
4‘‘‘ = {4“,1‘,5“}
5‘‘‘ = {1‘,5“}

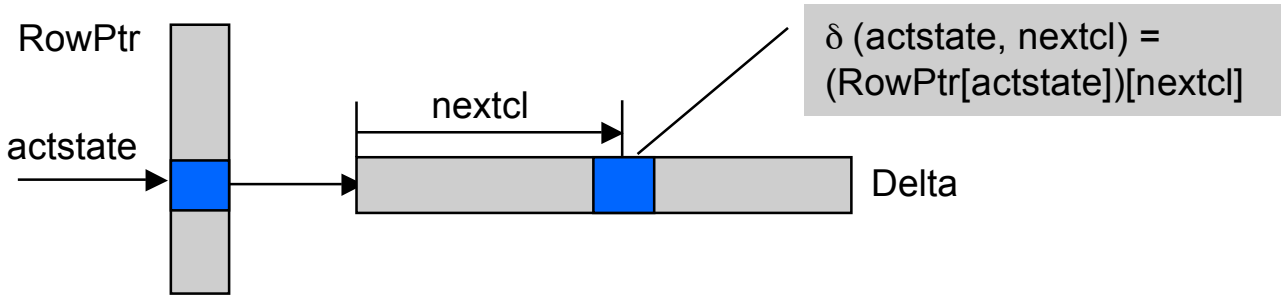
Implementierung eines Scanners (1/3)

Prinzipielle Idee

- Darstellung der Übergangsfunktion durch 2-dimensionale Matrix
  - Indiziert mit Zustand z und Zeichenklasse cl
  - Matrixelement M[z, cl] $=_{def} \delta(z, cl)$

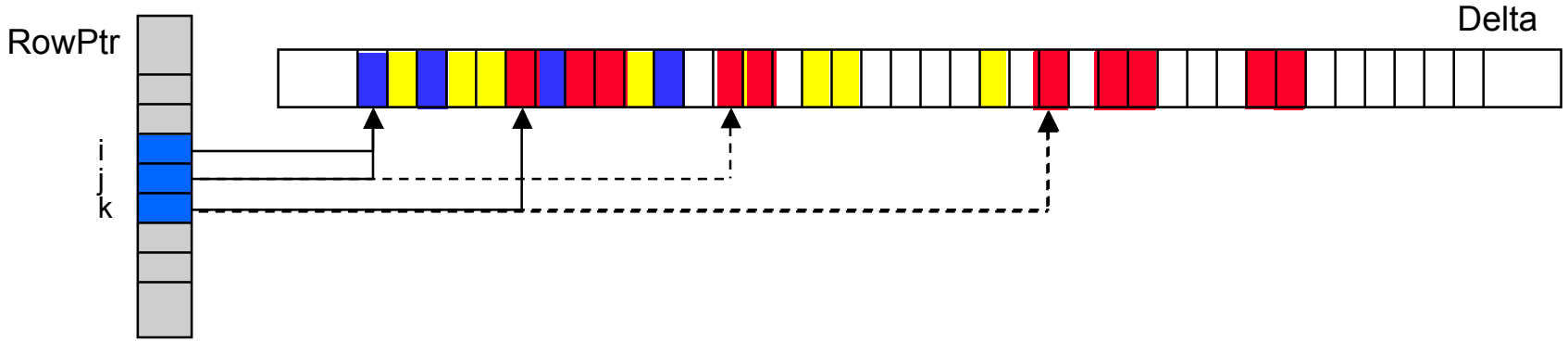
Tatsächliche Implementierung

- „1-dimensionales Feld von 1-dimensionalen Feldern“:
  - 1-dimensionales Feld RowPtr (indiziert mit Zuständen)
  - Elemente von RowPtr = Verweise auf Feld Delta (indiziert mit Zeichenklassen)
- Komprimierung von Delta durch Übereinanderlegen von Zeilen



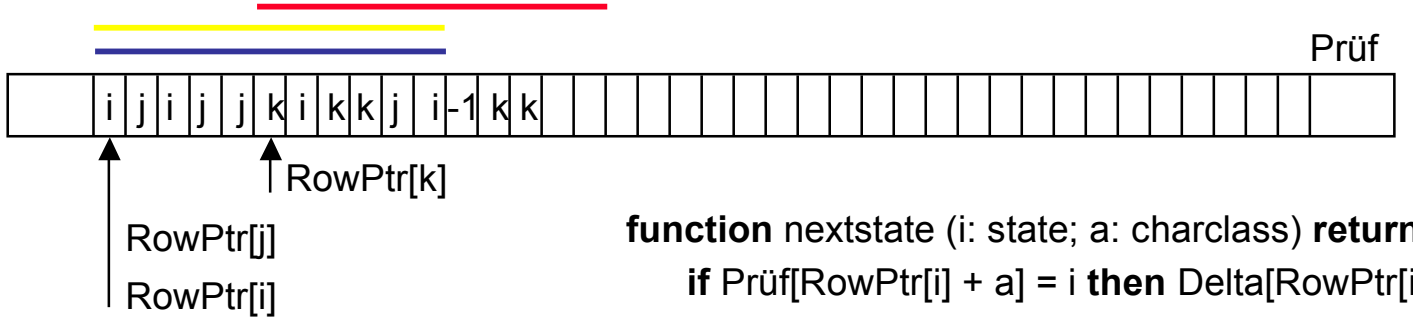
Implementierung eines Scanners (2/3)

Komprimierung



Problem

- Undefinierte Übergänge nicht mehr erkennbar
- Deshalb zusätzlich
  - Feld Prüf zur Erkennung definierter Übergänge: Prüf[RowPtr[i] + a] = i ⇔ delta(i, a) definiert
  - Implementierung der Übergangsfunktion durch Funktion nextstate



function nextstate (i: state; a: charclass) **returns** state
if Prüf[RowPtr[i] + a] = i **then** Delta[RowPtr[i] + a]
else –1 (* steht für undefiniert *) **fi**

Implementierung eines Scanners (3/3)

Weitere Möglichkeiten

- Alternative zur Matrix-Darstellung (bei manchen Anwendungen von Automaten effizienter): „*lazy transition evaluation*“
  - Berechnung der Automatenübergänge zur Laufzeit (wenn benötigt)
  - Ablage der berechneten Übergänge in einem Cache
- Eingabepuffer für blockweises Lesen (anstelle von zeichenweisem Lesen)