Algorithmus $DEA \rightsquigarrow DEA_{min}$

Eingabe: reduzierter DEA $M = (Q, \Sigma, \delta, q_0, F)$, ~~$\delta$ total~~

Ausgabe: äquivalenter minimaler DEA $M' = (Q', \Sigma, \delta', q_0', F')$

Methode:

(1) $t := 2$; $Q_1 := F$; $Q_2 := Q \setminus F$;

(2) **while** $\exists\, i \leq t,\ a \in \Sigma$ mit $\delta(Q_i, a) \not\subseteq Q_j \quad \forall\, j \leq t$

do

1. Wähle solch ein $i \leq t$, $a \in \Sigma$ und $j \leq t$ mit $\delta(Q_i, a) \cap Q_j \neq \emptyset$
2. $Q_{t+1} := \{ q \in Q_i \mid \delta(q, a) \in Q_j \}$;
   $Q_i := Q_i \setminus Q_{t+1}$;
   $t := t+1$

od;

(3) $Q' := \{ Q_1, Q_2, \ldots, Q_t \}$;
$q_0' := [q_0]$;
$F' := \{ [q] \in Q' \mid q \in F \}$
$\delta'([q], a) := [\delta(q, a)] \quad \forall\, q \in Q, a \in \Sigma$.

Dabei bezeichnet $[q]$ für $q \in Q$ diejenige Klasse Q_j in Q', die q enthält.

Korrektheitsbeweis Übung

Zeitanalyse: $O(k \cdot n^2)$, wobei $|\Sigma| = k$ und $|Q| = n$

in Vorlesung vorführen.

Frage:
Können wir obigen Algorithmus derart implementieren, dass diese Implementation wesentlich weniger Zeit benötigt

- Schritt (2) ist einziger Teil des Alg., der nicht leicht derart implementiert werden kann, so dass die Implementierung nur lineare Zeit benötigt

Beobachtung:

Falls wir für alle $q \in Q$ garantieren können, dass sich der Index der Menge, die q enthält, maximal $\log n$-mal ändert, dann könnte eine $O(k \cdot n \cdot \log n)$-Implementierung möglich sein.

⇝

Modifikation des Blockes der while-Schleife:

1. Wähle solch ein $i \leq t$, $a \in \Sigma$ und $j_1, j_2 \leq t$ mit
   - $j_1 \neq j_2$
   - $\delta(Q_i, a) \cap Q_{j_1} \neq \emptyset$ und $\delta(Q_i, a) \cap Q_{j_2} \neq \emptyset$
2. if $|\{q \in Q_i \mid \delta(q,a) \in Q_{j_1}\}| \leq |\{q \in Q_i \mid \delta(q,a) \in Q_{j_2}\}|$

   then
   $$Q_{t+1} := \{q \in Q_i \mid \delta(q,a) \in Q_{j_1}\}$$

   else
   $$Q_{t+1} := \{q \in Q_i \mid \delta(q,a) \in Q_{j_2}\}$$

   fi;

   $Q_i := Q_i \setminus Q_{t+1}$;
   $t := t+1$.

Eigenschaften:

- Wahl von $i \Rightarrow j_1$ und j_2 existieren.
- $|Q_{t+1}| \leq \frac{1}{2}|Q_i|$ bzgl. $|Q_i|$ vor der Neudefinition von Q_i.

$\Rightarrow$

Falls sich für einen Zustand q der Index derjenigen Menge, die q enthält, ändert, dann halbiert sich mindestens die Größe der korrespondierenden Menge

$\Rightarrow$

$\forall q \in Q$ ändert sich sein Index maximal $\log_2 n$-mal

Ziel:

Entwicklung einer Implementierung, so dass die Gesamtarbeit Transitionen zugerechnet werden kann, die einen Zustand enthalten, für den sich der Index der korrespondierenden Klasse ändert.

Hierzu benötigen wir Datenstrukturen, die folgende Operationen unterstützt:

1. Die Wahl von $i \leq t$, $a \in \Sigma$ mit $\delta(Q_i, a) \not\subseteq Q_j$ $\forall j \leq t$,
2. die Wahl von $j_1, j_2 \leq t$, $j_1 \neq j_2$ mit $\delta(Q_i, a) \cap Q_{j_1} \neq \emptyset$ und $\delta(Q_i, a) \cap Q_{j_2} \neq \emptyset$,
3. die Entscheidung ob

$$|\{q \in Q_i \mid \delta(q, a) \in Q_{j_1}\}| \leq |\{q \in Q_i \mid \delta(q, a) \in Q_{j_2}\}|$$

und

4. die Konstruktion von Q_{t+1} und $Q_i \setminus Q_{t+1}$.

Durchführung ~~~> Buch.

~~A13a)~~

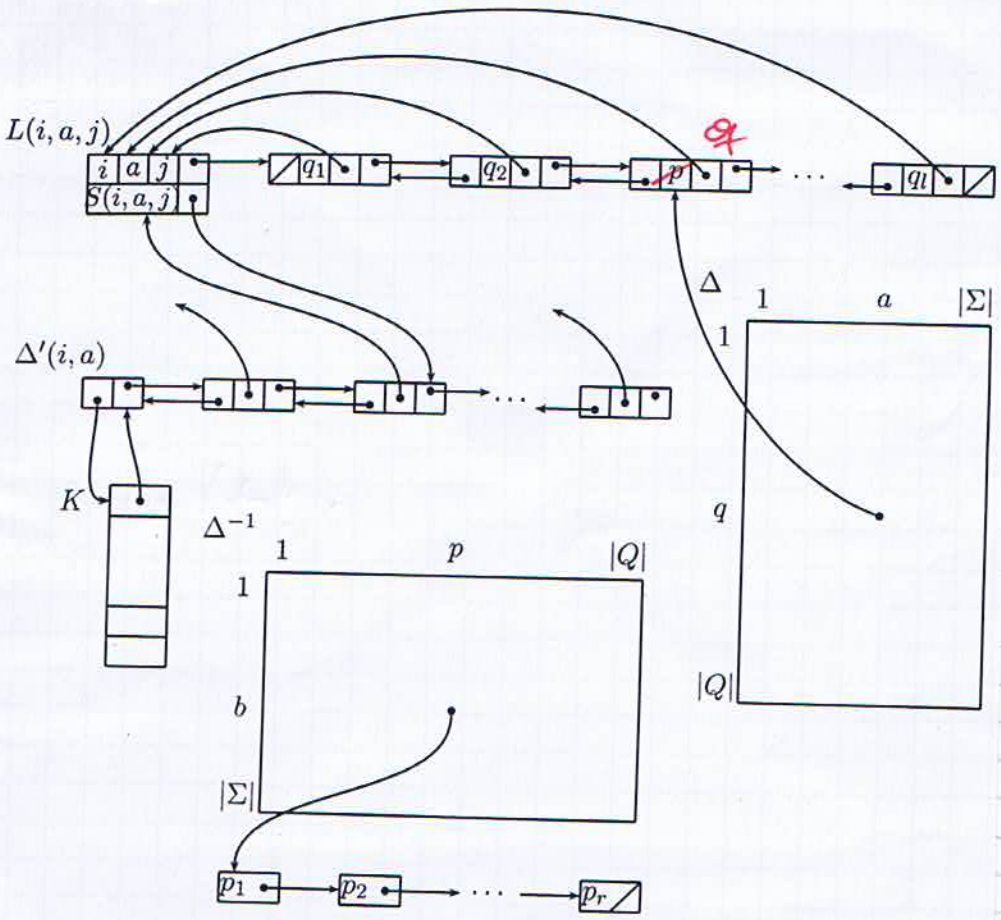


Abbildung 1.4: Die Datenstruktur D_t.

D_t, $t \geq 2$ Datenstruktur, die wir direkt nach der Konstruktion von Q_t erhalten haben.

Kernstück von D_t ist folgende Vergröberung δ' von δ:

$$\delta' \subseteq \{1,2,\dots,t\} \times \Sigma \times \{1,2,\dots,t\},$$

wobei

$$(i,a,j) \in \delta' \iff \exists\, q \in Q_i,\ p \in Q_j : \delta(q,a) = p.$$

- $\forall (i,a,j) \in \delta'$ enthält D_t eine doppelt verkettete Liste $L(i,a,j)$, wobei:
  - $L(i,a,j)$ enthält genau diejenige Zustände $q \in Q_i$ mit $\delta(q,a) \in Q_j$.
  - Jedes Listenelement enthält einen zusätzlichen Zeiger auf den Listenkopf
  - Listenkopf enthält Variable $S(i,a,j)$, in der die Größe der Liste $L(i,a,j)$ gespeichert ist.

- Zur Unterstützung der Wahl von $i \leq t$, $a \in \Sigma$ mit $\delta(Q_i, a) \notin Q_j$ $\forall j \leq t$ enthält D_t
  - $\forall$ Paare (i,a), $i \in \{1,2,\dots,t\}$, $a \in \Sigma$ Liste $\Delta'(i,a)$, wobei

    $\Delta'(i,a)$ enthält Zeiger auf Listenkopf von $L(i,a,j)$

    $\Leftrightarrow$ $(i,a,j) \in \delta'$.

    Listenkopf von $L(i,a,j)$ enthält Pointer auf diejenige Komponente in $\Delta'(i,a)$, die den Zeiger auf $L(i,a,j)$ enthält
  - Menge K verwaltet Zeiger auf diejenigen Listen $\Delta'(i,a)$, deren Länge ≥ 2 ist.

↪

Block der while-Schleife:

1. Unter Verwendung der Menge K bestimme Liste $\Delta'(i,a)$ der Länge ≥ 2 und wähle zwei beliebige Tripel (i,a,j_1) und (i,a,j_2) in $\Delta'(i,a)$ aus.
2. if $S(i,a,j_1) \leq S(i,a,j_2)$
   then
   $L(t+1,a,j_1) := L(i,a,j_1)$;
   $S(t+1,a,j_1) := S(i,a,j_1)$;
   Streiche Zeiger auf $L(i,a,j_1)$ aus $\Delta'(i,a)$ und füge Zeiger auf $L(t+1,a,j_1)$ in eine neue Liste $\Delta'(t+1,a)$ ein;
   if die Länge von $\Delta'(i,a) < 2$
   then
   Entferne Zeiger auf $\Delta'(i,a)$ aus K
   fi
   fi

01.02.

Sei

$$j_{min} = \begin{cases} j_1 & \text{falls } S(i,a,j_1) \leq S(i,a,j_2) \\ j_2 & \text{sonst} \end{cases}$$

Beobachtung:

Da sich für $q \in L(t+1, a, j_{min})$ der Index seiner Menge von i nach $t+1$ geändert hat, müssen wir

- für jeden $b \in \Sigma \setminus \{a\}$ die zu q korrespondierende Komponente aus seiner Liste $L(i, b, \ell)$ entfernen und in die Liste $L(t+1, b, \ell)$ einfügen und
- für alle $b \in \Sigma$ für alle p in einer Liste $L(k, b, i)$ mit $\delta(p, b) = q$ die zu p korrespondierende Komponente aus der Liste $L(k, b, i)$ entfernen und in die Liste $L(k, b, t+1)$ einfügen.

Zur effizienten Durchführung der obigen Operationen enthält die Datenstruktur D_t zusätzlich

- ein $(|Q| \times |\Sigma|)$-Feld Δ und
- ein $(|\Sigma| \times |Q|)$-Feld Δ^{-1},

wobei

- $\forall q \in Q \; \forall a \in \Sigma$ die Komponente $\Delta(q, a)$ enthält einen Zeiger auf die bzgl. der Listen $L(\cdot, a, \cdot)$ endeutig bestimmte Komponente, die q enthält und

- $\forall b \in \Sigma \; \forall p \in Q$ gilt:

  Komponente $\Delta^{-1}(b,p)$ enthält einen Zeiger auf eine Liste, die exakt die jenigen Zustände $q \in Q$ mit $\delta(q,b) = p$ enthält.

  Wir identifizieren diese Menge von Zuständen mit $\Delta^{-1}(b,p)$.

$D_t \rightsquigarrow D_{t+1}$:

Modifiziere für alle $q \in L(t+1, a, \text{junior})$ die Datenstruktur auf folgende Art und Weise:

(a) for alle $b \in \Sigma \setminus \{a\}$
do

- greife unter Verwendung von $\Delta(q,b)$ auf die eindeutig bestimmte Komponente bzgl. der Listen $L(i,b,\cdot)$, die q enthält, zu.
  (* Sei $L(i,b,\ell)$ die zugehörige Liste. *)
- $L(i,b,\ell) := L(i,b,\ell) \setminus \{q\}$
- $L(t+1,b,k) := L(t+1,b,k) \cup \{q\}$

Zur effizienten Bestimmung der Liste $L(t+1,b,k)$

verwalten wir ein zusätzliches $(|\Sigma| \times |Q|)$-Feld Γ. Die Komponente $\Gamma(b,k)$ enthält einen Zeiger auf die zuletzt generierte Liste $L(j,b,k)$.

- Falls $j \neq t+1$ müssen wir eine neue Liste $L(t+1,b,k)$ generieren und $\Gamma(b,k)$ modifizieren.

- $S(t+1,b,k) := S(t+1,b,k) + 1$;
- $S(i,b,k) := S(i,b,k) - 1$
- if $L(i,b,k)$ leer
  then
    streiche Zeiger auf $L(i,b,k)$ aus $\Delta'(i,b)$;
    if $|\Delta'(i,b)| < 2$
      then
        streiche Zeiger auf $\Delta'(i,b)$ aus k, falls vorhanden
      fi
  fi
  falls $L(t+1,b,k)$ neu generiert,
- Zeige Zeiger auf $L(t+1,b,k)$ in $\Delta'(t+1,b)$ und eventuell einen Zeiger auf $\Delta'(t+1,b)$ in k ein.

(b) for alle $b \in \Sigma$, alle $p \in \Delta^{-1}(b,q)$
  do
  - greife unter Verwendung von $\Delta(p,b)$ auf die korrespondierende Komponente

bzgl. der Listen $L(\cdot, b, i)$ zu.
(* Sei $L(z, b, i)$ diejenige Liste, die den Zustand p enthält *)

- $L(z, b, i) := L(z, b, i) \setminus \{p\}$;
- $L(z, b, t+1) := L(z, b, t+1) \cup \{p\}$;

Zur effizienten Bestimmung der Liste $L(z, b, t+1)$ verwalten wir analog zu oben ein zusätzliches $(|Q| \times |\Sigma|)$-Feld Γ:

- $S(z, b, t+1) := S(z, b, t+1) + 1$;
- $S(z, b, i) := S(z, b, i) - 1$
- **if** $S(z, b, i) = 0$
  then
    streiche Zeiger auf $L(z, b, i)$ aus $\Delta'(z, b)$;
    if $|\Delta'(z, b)| < 2$
    then
      streiche Zeiger auf $\Delta'(z, b)$ aus K, falls vorhanden
    fi
  fi
- Falls $L(z, b, t+1)$ neu besetzt wird, dann füge Zeiger auf $L(k, b, t+1)$ in $\Delta'(z, b)$ und eventuell einen Zeiger auf $\Delta'(z, b)$ in K ein.

Einige Bemerkungen zur Korrektheit!

Analyse:

Block while-Schleife:

1. $O(1)$
2. $O(1)$

$D_t \rightsquigarrow D_{t+1}$:

Streichen einer Komponente aus Liste mit Einfügen in andere Liste $O(1)$

Beobachtung:

- Jedes Mal, wenn eine Komponente q bewegt wird, wird eine Zeile $\delta(q,a) = p$ bzw. $\delta(p,a) = q$, wobei sich der Index der Klasse, die q enthält, ändert, betrachtet.
- Da jedes Mal, wenn eine Zeile von δ betrachtet wird, mindestens für einen der beiden Zustände in der Zeile sich der Index ändert, wird jede Zeile maximal $2 \cdot \log n$-mal betrachtet. Da es maximal kn Zeilen in δ gibt, ist die Gesamtzeit durch $O(kn \log n)$ begrenzt.

$\rightsquigarrow$

Satz 4.5

Der Algorithmus $DEA \rightsquigarrow DEA_{min}$ kann derart implementiert werden, dass seine Laufzeit

4.1.3 Zur Realisierung der lexikalischen Analyse

Aufgabe des Scanners in einem Übersetzer:

Eingabe:

Das Quellprogramm als Zeichenfolge über einem Basisalphabet Σ_0, das

- Buchstaben (A, B, ..., Z, a, b, ..., z evtl. einschließlich spezielle Zeichen wie $, % usw.),
- Ziffern (0, 1, ..., 9) und
- Sonderzeichen ('+', '-', '*', '/', <, ...)

enthalten kann.

Ausgabe:

Das Programm als Folge von Grundsymbolen zuzüglich

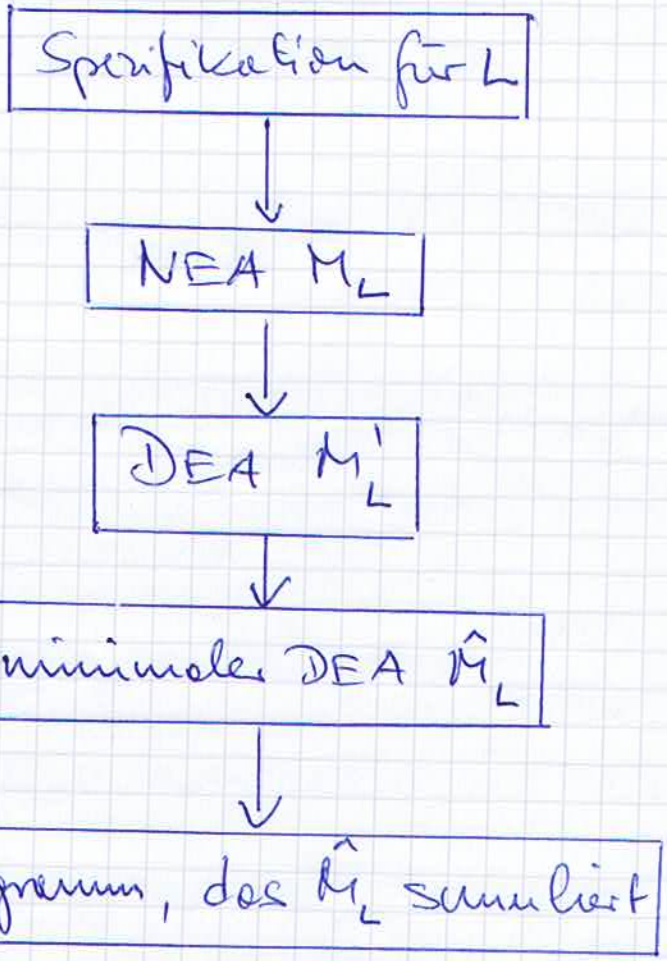
- Zusatzinformation (wie z.B. Grundsymbol ist ein Bezeichner)

abzüglich

- redundante Zeichenfolgen (z.B. Kommentare, Leerzeichen usw.)

- L lexikalische Einheiten der Quellsprache QS.

  Spezifikation durch eine reguläre Definition.



Generierung des Scanners.

Vereinigung

1. Konstruiere für jeden regulären Ausdruck einen NEA. Die Zustandsmengen der NEA's sind paarweise disjunkt.
2. Konstruiere aus diesen NEA's einen NEA M_L, der die Vereinigung der betreffenden

reguläre Mengen akzeptiert. (Separater Endzustand für jede reguläre Menge)

3. Konstruiere aus M_L einen DEA M'_L.

4. Konstruiere aus M'_L einen minimalen DEA $\hat{M}_L$. Betrachte hierbei die Endzustände von M'_L, die zu verschiedenen Endzuständen von M_L korrespondieren, als paarweise unterscheidbar.

4.2 Die Syntaxanalyse

stellt fest, ob gegebenes Programm syntaktisch korrekt ist.

zzgl. (falls korrekt)

Syntaxstruktur durch Ableitungsbaum

falls nicht korrekt

$\rightsquigarrow$ Fehlermeldung

Fortsetzung der Syntaxanalyse an der "frühest möglichen" Stelle mit dem Ziel

möglichst viele Fehler zu lokalisieren und zu melden.

- Definition der Syntax durch zfg.