Wiederholung letzter Woche

- **Nichtdeterministische Endliche Automaten**
  - Repräsentation der Transitionstabelle als `array`
  - *Listen* von Folgezuständen
  - Angepasste `traverse` Funktion
- **Kontrollstruktur CASE**
- **Datenstruktur Array**
- **Aufsammeln von Ergebnissen mit `append` in `loop`**

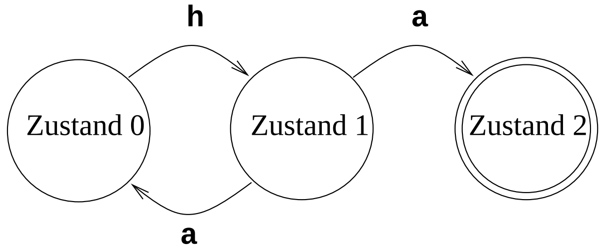
Neu diese Woche

- **Epsilon-Übergänge**
- **Verknüpfung von Automaten**
  - Disjunktion (`join`)
  - Konkatenation (`concatenate`)
  - Kleene-Stern (`kleene`)

Automaten

Wiederholung: NDA

Endliche Automaten sind nichtdeterministisch, wenn sie mindestens einen Zustand haben, aus dem es mit dem gleichen Eingabesymbol mehrere Übergänge gibt.



Automat zum Erkennen von `( h a )`$^+$

Automaten

Wiederholung: Transitionstabelle

```
(defparameter *haha*
    (make-array (list 3 3)
                :initial-contents
                '((nil (1) nil  )
                  (nil nil (0 2))
                  (2   nil nil  ))))
```

Die Felder in der Übergangstabelle enthalten nicht mehr einzelne Folgezustände, sondern Listen von Folgezuständen.

Automaten

Nichtdeterministische Endliche Automaten

```
(defun traverse (table input &optional
                              (state 0) (position 0) (n (length input)))
  (if (= position n)
    (when (aref table state 0) (list (aref table state 0)))
    (loop
        with forward = (1+ position)
        for next in (aref table state (token2code (schar input position)))
        append (traverse table input next forward n))))
```

Die Folgezustände werden per `loop` nacheinander verfolgt. Rückgabewert ist die Liste aller erreichten Endzustände, d.h. es handelt sich um eine exhaustive Suche.

Automaten

Weitere Beispiele (I)

```
;;;
;;; *absterna* ist ein Automat, der Strings der Form ab*a erkennt
;;;                                                      ^- (Kleene Star)
;;;
(defparameter *absterna* (make-array (list 3 (+ *code-maximum* 1))))
(setf (aref *absterna* 0 (token2code #\a)) '(1)
      (aref *absterna* 1 (token2code #\a)) '(2)
      (aref *absterna* 1 (token2code #\b)) '(1)
      (aref *absterna* 2 0               ) 'absterna)
```

Automaten

Neue Form der Transitionstabelle

```
USER(55): *absterna*
#2A((NIL      NIL (1) NIL)
    (NIL      NIL (2) (1))
    (ABSTERNA NIL NIL NIL))
```

Achtung: Die zweite Spalte ist reserviert für Epsilon-Übergänge. `token2code` liefert immer Werte ≥ 2, siehe auch Übung 2.

Automaten

Weitere Beispiele (II)

```
;;;
;;; *abbsterna* ist ein Automat, der Strings der Form a(bb)*a erkennt
;;;
(defparameter *abbsterna* (make-array (list 4 (+ *code-maximum* 1))))
(setf (aref *abbsterna* 0 (token2code #\a)) '(1)
      (aref *abbsterna* 1 (token2code #\a)) '(3)
      (aref *abbsterna* 1 (token2code #\b)) '(2)
      (aref *abbsterna* 2 (token2code #\b)) '(1)
      (aref *abbsterna* 3 0               ) 'abbsterna)
```

Automaten

Erweiterung um Epsilon-Übergänge

- Epsilon-Übergänge repräsentieren Übergänge, *ohne Lesen vom Eingabeband.* Dies kann auch als das Lesen des leeren Wortes ϵ interpretiert werden.
- Automaten mit Epsilon-Übergängen sind immer nicht-deterministisch.
- Epsilon-Übergänge werden in der bisher reservierten (siehe Übung 2) Spalte vermerkt. Also Spalte 1, oder die zweite Spalte.

Automaten

Anwendung von Epsilon-Übergängen

Zur Erinnerung: Die Automaten für `ha`$^+$ und `hi`$^+$:

```
(defparameter *haha*
   (make-array (list 3 (+ *code-maximum* 1))))
(setf (aref *haha* 0 (token2code #\h)) '(1)
      (aref *haha* 1 (token2code #\a)) '(2)
      (aref *haha* 2 (token2code #\h)) '(1)
      (aref *haha* 2 0) 'ha)
```

```
(defparameter *hihi*
   (make-array (list 3 (+ *code-maximum* 1))))
(setf (aref *hihi* 0 (token2code #\h)) '(1)
      (aref *hihi* 1 (token2code #\i)) '(2)
      (aref *hihi* 2 (token2code #\h)) '(1)
      (aref *hihi* 2 0) 'hi)
```

Automaten

Kombination der Automaten mit Epsilon-Übergängen

```
(defparameter *hahi*
   (make-array (list 3 (+ *code-maximum* 1))))
(setf (aref *hahi* 0 (token2code #\h)) '(1)
      (aref *hahi* 1 (token2code #\a)) '(2)
      (aref *hahi* 2 (token2code #\h)) '(1)
      ;;(aref *haha* 2 0) 'ha
      (aref *hahi* 2 1               ) '(3)  ;; epsilon!!
      (aref *hihi* 3 (token2code #\h)) '(4)
      (aref *hihi* 4 (token2code #\i)) '(5)
      (aref *hihi* 5 (token2code #\h)) '(4)
      (aref *hihi* 5 0               ) 'hahi)
```


Automaten

Kombination der Automaten mit Epsilon-Übergängen

Wie funktioniert die Kombination?

- Verschieben eines oder beider Automaten in der Tabelle (dabei muss man eventuell benötigte neue Zustände beachten)
- Hinzufügen der neuen Epsilon-Übergänge
- Setzen/Löschen der Endzustände

Automaten

Traversieren von Epsilon-Übergängen

```
(defun traverse-string (table input &optional
                                    (state 0) (position 0) (n (length input)))
  (append
   ;; Folge zunaechst allen Epsilon-Uebergaengen und sammele die Endzustaende
   (loop for next-eps-state in (aref table state 1)
         append (traverse-string table input next-eps-state position n))
   ;; Arbeite das naechste Eingabezeichen ab
   ;; oder
   ;; falls die Eingabe abgearbeitet ist
   (if (= n position)
       ;; ist es ein Endzustand ? Dann bestimme seinen "Wert".
       (when (aref table state 0) (list (aref table state 0)))
     ;; sonst arbeite das naechste Eingabezeichen ab.
     (loop for next-state in
           (aref table state (token2code (schar input position)))
           append (traverse-string table input next-state (1+ position) n)))))
```


Automaten

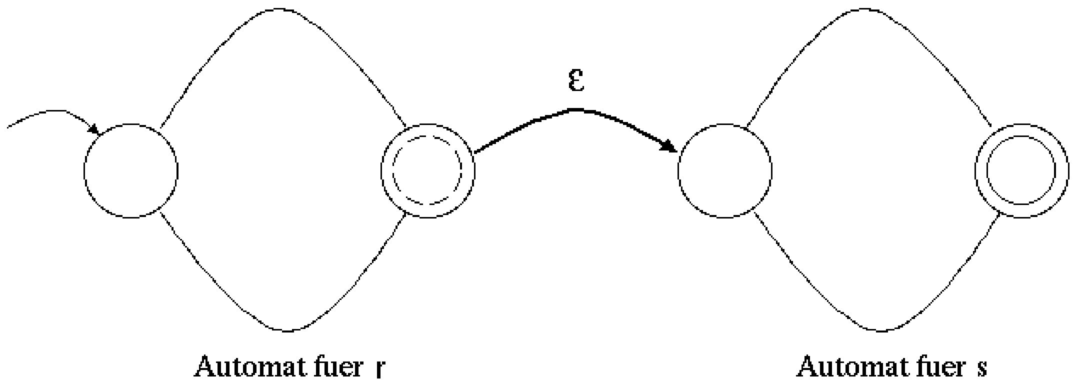
Reguläre Ausdrücke

Bei *regulären Ausdrücken* sind folgende drei Operationen definiert. Erkenner für Reguläre Ausdrücke können mit Automaten realisiert werden. Dafür brauchen wir also entsprechende Verknüpfungen von Automaten.

- Konkatenation
- Disjunktion
- Kleene-Stern

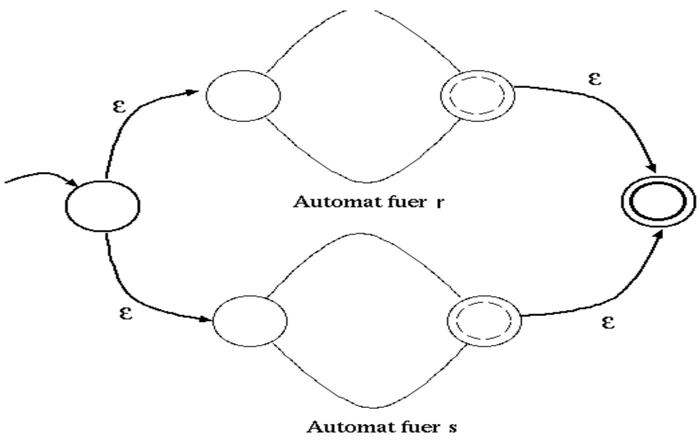
Automaten

Konkatenation von Automaten r und s



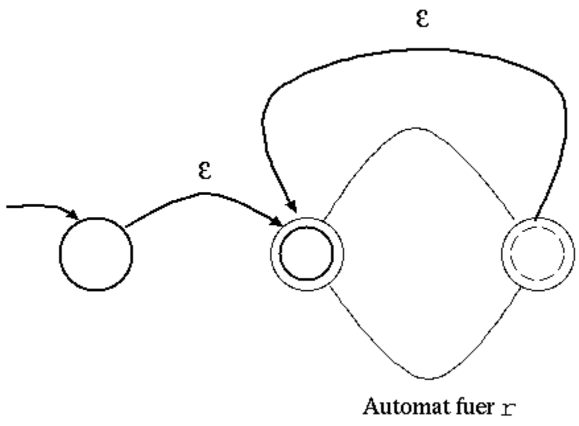
Automaten

Disjunktion von Automaten r und s



Automaten

Kleene-Stern eines Automaten r



Automaten

Vereinfachung von Automaten

Zu jedem Nicht-Deterministischen Endlichen Automaten kann man einen Deterministischen Endlichen Automaten konstruieren, der die gleiche Sprache akzeptiert.
D.h., die Epsilon-Übergänge können eliminiert werden.
Endliche Automaten können *minimiert* werden. Die minimale Größe ist eindeutig.
Im allgemeinen ist ein äquivalenter Deterministischer (auch minimierter) Endlicher Automat *exponentiell* größer als der ursprüngliche Nicht-Deterministische Endliche Automat.