Letzter/Dieser Kurs

Einführung in LISP: Letzter Kurs

Datenstrukturen

Kontrollstrukturen

Allgemeines zum Programmieren

Anwendung von LISP für Computerlinguistische Algorithmen: Dieser Kurs

Automaten: Morphologie

Parsing: Syntax

Unifikation: Syntax/Semantik

Automaten

Endliche Automaten:

- lesen eine **Eingabe** Stück für Stück
  - endliches Eingabealphabet
  - Eingabeband (mit Zeiger)
- gehen dabei von **Zustand** zu Zustand
  - endliche Zustandsmenge
  - Transitionsfunktion: Eingabe, Zustand → Ausgabe, Zustand
  - kann eindeutig sein (*deterministisch*) oder auch nicht (*nicht-deterministisch*)
- produzieren eine **Ausgabe**
  - Akzeptor: sagt am Ende JA oder NEIN
  - allgemein: erzeugen bei jedem Schritt eine Ausgabe

Automaten

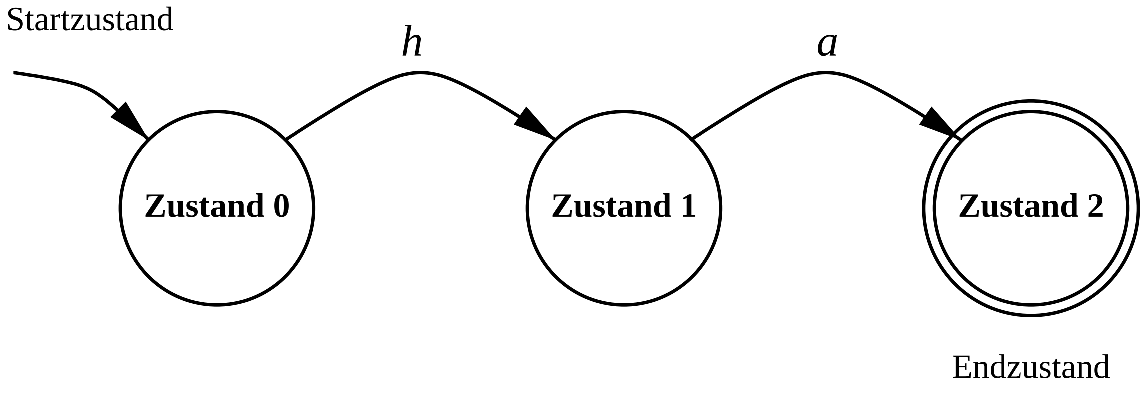
Deterministische Endliche Automaten

- Anwendung für
  - *Tokenerkennung* (Informatik)
  - *Morphologie* (Computerlinguistik)
  - approximatives Parsing
  - *"finite state methods for ..."*
- Äquivalent zu *regulären Ausdrücken*

Automaten

Deterministische Endliche Automaten

Automat zum Erkennen von `h a`



CASE

Wiederholung: **CASE**

```
(case bit (0 'null)
          (1 'eins))
```

- `case keyform {(({key}*) | key} {form}*)}*`
- `keyform` wird zuerst ausgewertet
- jeder weitere Ausdruck beginnt mit `key`s, die passen müssen
- ein `key` darf (eigentlich) nur in einem Ausdruck vorkommen
- `otherwise` oder `t` für den *else* Fall

CASE

Wiederholung: **CASE**

```
(case tense (present 'gegenwart)
            (future 'zukunft)
            ((past
              pastperfect
              pluperfect) 'vergangenheit)
            (otherwise 'keine-zeit))
```

Felder (Arrays)

make-array, aref, setf, ...

```
(setq tic-tac-toe
      (make-array (list 3 5) :initial-contents '((NULL O X O NULL)
                                                 (EINS - - X EINS)
                                                 (ZWEI O X X ZWEI))))
(aref tic-tac-toe 0 3)  --->  O
(make-array '(4711) :initial-element 42)  --->  ??
```

- `make-array dimensions &key :initial-element :initial-contents :element-type`
  - `dimensions` ist eine Liste
  - `:initial-element` wird für alle Elemente eingetragen
  - `:initial-contents` ist eine Liste von Listen von Listen...

Felder (Arrays)

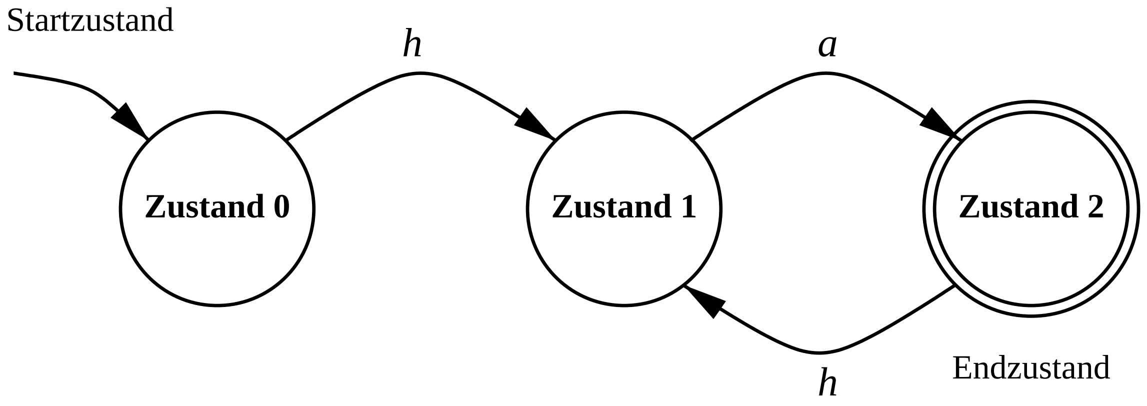
Arbeiten mit Feldern (Arrays):

- `(aref array &rest subscripts)`: Zugriff auf ein Element – *Zählen beginnt bei* `0` *!*
  Auch: `(apply #'aref tic-tac-toe '(0 3))`
- `(setf (aref tic-tac-toe 1 2) 'O)` ...und `O` gewinnt.
- `(array-rank array)`: Anzahl der Dimensionen
- `(array-dimension array axis-number)`: Länge der Dimension `axis-number`
  Zählen beginnt bei `0` *!*
- `(array-dimensions array)`: Liste der Längen der Dimensionen
- `(length (array-dimensions array)) == (array-rank array)`

Automaten

Deterministische Endliche Automaten

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



Automaten

Naive Implementierung:

- Für jeden Zustand wird eine eigene Funktion geschrieben.
- Input als Liste von Eingabesymbolen.
- Ausgabe: `T` oder `NIL`

Automaten

```
(defun haha (input)
  (state0 input))

(haha '(h a h a h a))
```

```
(defun state0 (input)
  (case (first input)
    (h (state1 (rest input)))
    (t nil)))

(defun state1 (input)
  (case (first input)
    (a (state2 (rest input)))
    (t nil)))

(defun state2 (input)
  (case (first input)
    (h (state1 (rest input)))
    ((nil) t)
    (t nil)))
```


Automaten

Implementierung mit Transitionstabelle (Übergangstabelle).

```
(defparameter *haha*
    (make-array (list 3 3)
       :initial-contents '((nil 1   nil)
                           (nil nil 2  )
                           (t   1   nil))))
```

- Jede Zeile repräsentiert einen Zustand
- Spalte 0 repräsentiert, ob *Endzustand*
- Weitere Spalten kodieren den Folgezustand, abhängig von der Eingabe

Automaten

```
(defparameter *haha*
    (make-array (list 3 3)
       :initial-contents '((nil 1   nil)
                           (nil nil 2  )
                           (t   1   nil))))
```

```
(defun token2code (token)
  (case token
    (h 1)
    (a 2)))
```

- Weitere Spalten kodieren den Folgezustand, abhängig von der Eingabe
- Spalte 1: Eingabe `h`
- Spalte 2: Eingabe `a`
- `token2code` bildet von Eingabesymbol auf Index ab.

Automaten

```
(defparameter *haha*
    (make-array (list 3 3)
       :initial-contents '((nil 1   nil)
                           (nil nil 2  )
                           (t   1   nil))))
```

```
(defun token2code (token)
  (case token
    (h 1)
    (a 2)))
```

```
(defun traverse (input &optional (state 0))
  (if input
    (let ((next (aref *haha* state (token2code (first input)))))
      (when next
        (traverse (rest input) next)))
    (aref *haha* state 0)))
```