Wiederholung letzter Woche

- **Endliche Automaten**
  - Unterschied zwischen Erkennern und Transducern
  - Unterschied zwischen deterministischen und nicht-deterministischen Automaten
- **Kontrollstruktur CASE**
- **Datenstruktur Array**
- **Naive Implementierung**
  - Übergänge als Funktionen
  - Eingabeband als Liste von Symbolen
- **Übergangstabellen**

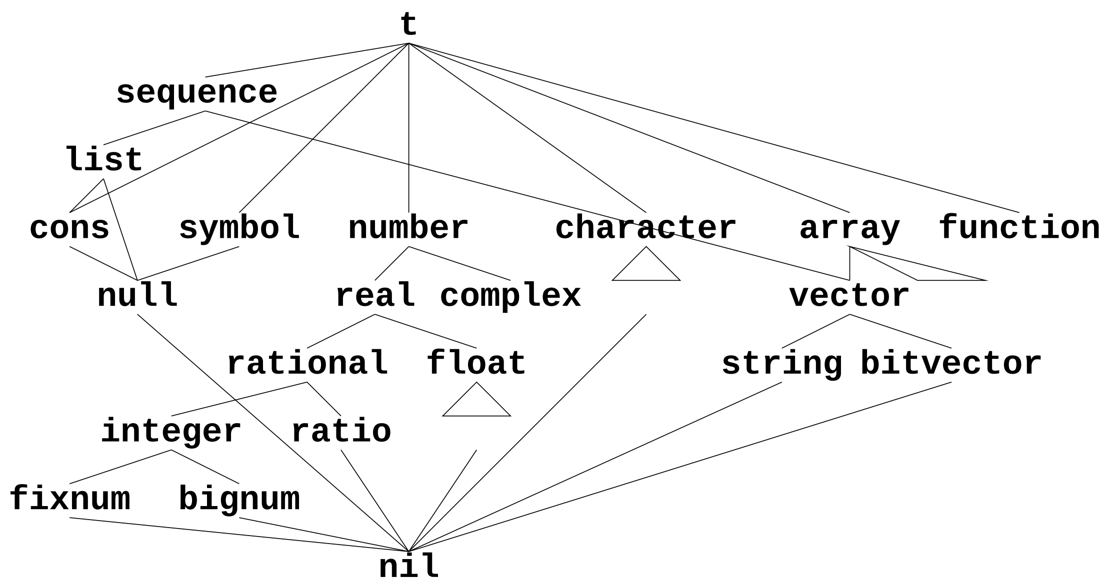
Automaten

Die Darstellung der Eingabe

Die Darstellung des Eingabebandes als Liste von Symbolen weist einige Probleme auf:

- Eingaben liegen typischerweise als Folge von Zeichen vor,
  z.B. beim Lesen mit `read-line()`
- Der Lisp-Reader ist bei Symbolen nicht case-sensitiv:
  `(eq 'a 'A)` $\rightarrow$ `T`
- Symbole belegen mehr Speicherplatz als notwendig
- Listen belegen mehr Speicherplatz als notwendig

Typenhierarchie



Datenstrukturen

Characters, *Strings* **und was wir damit machen können**

- *Characters* stellen einzelne Zeichen dar:
  `#\A  #\&  #\Space` …
- Konvertierung *characters* zu *integers* (vgl. `token2code()`, 1. Übung)
  `(char-code #\A)` → `65`
  `(char-code #\a)` → `97`
- Konvertierung *integers* zu *characters*
  `(code-char 97)` → `#\a`

Datenstrukturen

Characters, *Strings* und was wir damit machen können

- Prädikate über *characters*, siehe [Steele, 13.2]
  `(characterp #\f)` → `T`
  `(alpha-char-p #\f)` → `T`
  `(upper-case-p #\f)` → `NIL`
  `(digit-char-p #\4)` → `T`
  `(alphanumericp #\4)` → `T`
  `(char= #\A #\a)` → `NIL`
- *Strings* sind spezialisierte Vektoren von *characters*
- Extraktion einer Teilkette eines *strings*
  `(subseq "evil" 1 3)` → `"vi"`

Datenstrukturen

Characters, *Strings* und was wir damit machen können

`(char` ⟨*string*⟩ ⟨*index*⟩`)` beschreibt ein einzelnes Feldelement eines *strings*, also einen *character*. Als sogenannte generalisierte Variable erlaubt dieser Ausdruck sowohl den Zugriff auf die *characters* eines *strings*, als auch deren (destruktive) Veränderung.

`(defparameter mystring "Haus")`

- **Zugriff**
  `(char mystring 0)` → `#\H`
- **Veränderung**
  `(setf (char mystring 0) #\M)`
  `mystring` → `"Maus"`

Automaten

Verarbeitung von *character*-Sequenzen

```
(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)))
```

- Automatenbeschreibung als Übergangstabelle; Spalte 0 zeigt Endzustände
- Kodierung der Eingabe als Sequenz von *characters*

Automaten

Verarbeitung von *character*-Sequenzen

```
(defun traverse (table input &optional (state 0))
  (if (zerop (length input))
      (aref table state 0)
    (let ((code (token2code (elt input 0))))
      (when code
        (let ((next (aref table state code)))
          (when next
            (traverse table (subseq input 1) next)))))))
```

```
(traverse *haha* "haha")  →  T
(traverse *haha* '(#\h #\a #\h #\a)  →  T
(traverse *haha* "Haha")  →  NIL
```

Diese Variante ist ineffizient durch das Testen über `length()`, den linearen Zugriff durch `elt()` und durch das Kopieren mit `subseq()`.

Automaten

Verarbeitung von *strings*

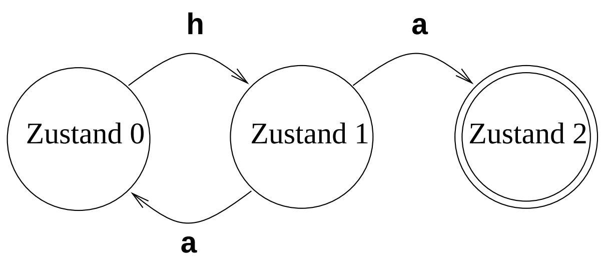
Eine optimierte und auf *strings* spezialisierte Fassung verwaltet die Länge der Eingabe und die aktuelle Position in optionalen Parametern.

```
(defun traverse (table input &optional (state 0)
                                       (position 0)
                                       (n (length input)))
  (if (= position n)
    (aref table state 0)
    (let ((code (token2code (schar input position))))
      (when code
        (let ((next (aref table state code)))
          (when next
            (traverse table input next (+ position 1) n)))))))
```


Automaten

Nichtdeterministische Endliche Automaten

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

Nichtdeterministische Endliche Automaten

- Nichtdeterministische Automaten können eine Sprache kompakter repräsentieren. Dafür ist die Verwaltung mehrerer Übergänge aus einem Zustand heraus aufwendiger, sprich langsamer.
- Bei eindeutiger Eingabe kann jeder nichtdeterministische Erkenner in einen äquivalenten deterministischen Erkenner transformiert werden. Dies ist bei Transducern nicht immer der Fall.
- Die Natur der Eingabe kann aber eine nichtdeterministische Verarbeitung erfordern, z.B. ein Automat, der über den möglicherweise mehrdeutigen part-of-speech einer Eingabe eine Phrase konstruieren will.

Automaten

Nichtdeterministische Endliche Automaten

```
(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 = (+ position 1)
        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.