Zusammenfassung
Teile V - VI

LISP-Programm

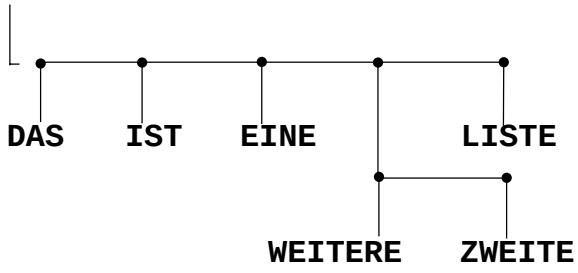
- **LISP – Programm:** **Folge von Ausdrücken, die nacheinander ausgewertet werden**
- **Ausdrücke**
  - Atome oder Liste
- **Form**
  - auswertbarer Ausdruck
- **Atome**
  - Zahlenliterale
  - Symbole (Literale, Konstante, Variablen)
  - Zeichenketten(literale) „a b c d“ „~ A“
- **Liste**

  (listelem listelem ... listelem)

  NIL

Liste für Daten und Funktionen

- **(DAS IST EINE (WEITERE ZWEITE) LISTE)**



- `(DEFUN VERTAUSCHE (PAAR)`

  `(LIST(CADR PAAR)(CAR PAAR))))`

Funktionen

- **Definition**
  - `(DEFUN FName [Parlist] [Rumpf] )`
- **QUOTE ist die Identitätsfunktion**
  - QUOTE wertet seine Argumente nicht aus!
  - `(QUOTE A)` kann kürzer dargestellt werden als `'A`
- **Die Wertzuweisung an ein symbolisches Atom geschieht durch die Systemfunktion** SETQ
  - `(SETQ L '( A B ))`
  - Ergebnis ist ein Seiteneffekt
- **Systemfunktion EVAL**
  - (eval (- 4 3))
  - Die Auswertung eines Ausdrucks kann damit explizit angestoßen werden.

Grundlegende Funktionen

- **CONS**
  - Aufbau einer Liste
- **CAR, CDR**
  - erstes Element einer Liste, Restliste
- **APPEND**
  - Die Liste, die durch Aneinanderfügen der Elemente der Listen gewonnen wird.
- **LIST**
  - Liste der Werte der Argumente in der gegebenen Reihenfolge
- **Prädikate**
  - ATOM, CONSP, NULL, LISTP. NUMBERP, EQ EQUAL

Funktionen

- **Klassifikation**
  - **Systemfunktionen**
    - spezielle Funktionen
    - normale Funktionen
  - **benutzerdefinierte Funktionen**
    - benannte Funktionen (DEFUN)
    - anonyme Funktionen (LAMBDA-Ausdruck)
- `(LAMBDA [Parlist] [Rumpf])` **analog zu DEFUN**
  - Definiert namenlose Funktion
  - Lambda-Ausdruck hat keinen Wert
  - `(Lambda-Ausdruck a1 a2 . . .an)`

Bindung

Bindung und Umgebung

- **Bindung**
  - Zuordnung von Symbolen zu Werten
- **Umgebung**
  - Gesamtheit der zu einem Zeitpunkt zugänglichen Bindungen
  - Aktuelle Umgebung
- **Definitionsumgebung einer Funktion**
- **Aufrufumgebung einer Funktion**
- **Gebundene Variable**
  - Variable, die in der Parameterliste einer Funktion auftaucht.
  - Lokale Variable
- **Freie Variable**
  - Variable, die im Rumpf einer Funktion steht, aber keinen Parameter bezeichnet.

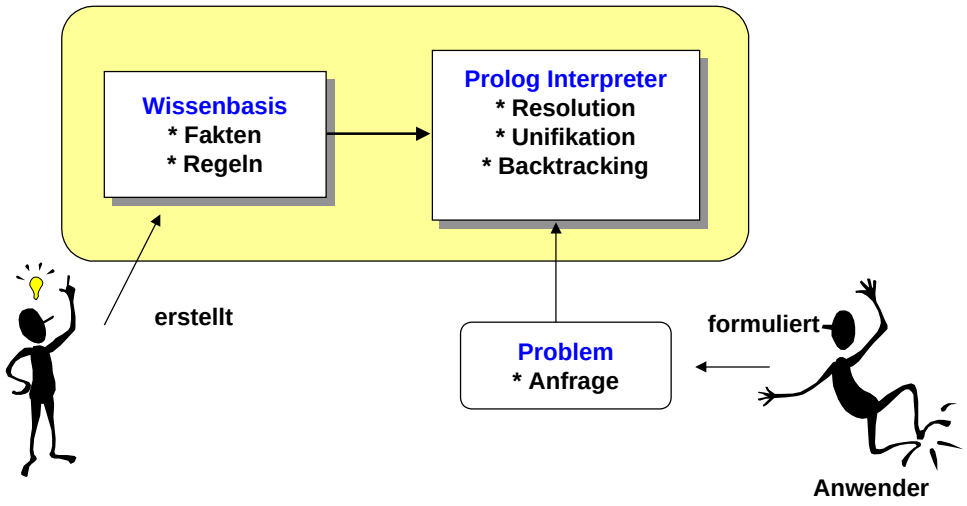
Funktionsobjekte

- **Die Funktion symbol-function**
  - erwartet einen Funktionsnamen als Argument und liefert ein Funktionsobjekt
- **Die Funktion function**
  - erwartet einen Lambda-Ausdruck und liefert ebenfalls ein Funktionsobjekt
  - Das Ergebnis ist eine sog. Closure
- **Closures sind (anonyme) Funktionen,**
  - die Variablen ihrer Umgebung referenzieren.
- **Ist das Argument von symbol-function eine selbstdefinierte Funktion,**
  - dann liefert diese auch eine Closure.
- **Als Kurzschreibweise**
  - für symbol-function und function kann #' benutzt werden.

Listendurchwanderung

- **Bei der Durchwanderung (mapping)**
  - geht man die Liste durch und wendet eine Funktion auf die Elemente der Liste an.
- **Mapping-Funktionen**
  - verlangen ein funktionales Argument; sind also Funktionen höherer Ordnung
  - Die Funktion wird entweder auf das erste Element der aktuellen Liste (car) oder auf die gesamte aktuelle Liste angewendet.
- **Das Gesamtergebnis der Durchwanderung der Liste**
  - kann gesammelt und als Liste zurückgegeben werden.
  - Sind nur Seiteneffekte von Interesse, kann die Liste unverändert zurückgegeben werden.
- **MAPC, MAPCAR**

Prolog Programmiermodell



Definitionen nach Sterling/Shapiro: Programm, Regel, Prädikat

- Ein **Logikprogramm** ist eine endliche Regelmenge.
- Statt Regel sagt man auch **Horn-Klausel** oder bei Eindeutigkeit auch einfach nur **Klausel.**
- Eine **Regel** hat die Form.

  $A <- B_1, B_2 , ... B_n.$ mit $n >= 0$

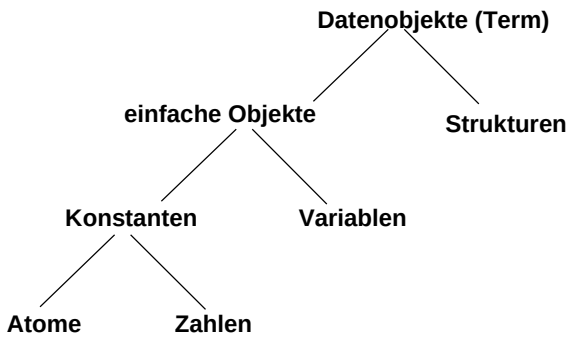
  A ist der **Regelkopf** und die B_i's sind der **Regelrumpf.**
  Eine Regel mit $n=0$ wird **Fakt** genannt. A und B_i's werden auch Ziele genannt.
- Ein **Ziel** hat die Form eines Prädikats (Prädikatenlogik 1. Stufe)

  $pred (t_1, t_2, ..., t_m)$ mit $m >=0$

  wobei $pred$ Prädikatnamen, m Stelligkeit und die tj Argumente.
  Prädikate beschreiben Objekte und Beziehungen zwischen diesen Objekten.
- Argumente für Prädikate sind **Terme**; induktiv definiert:
  - eine Konstante ist ein Term,
  - eine Variable ist ein Term,
  - sind $t_1, t_2, ..., t_k$ Terme und ist f ein Funktionssymbol, so ist auch $f (t_1, t_2, ..., t_k)$ ein Term.

Objekte in Prolog

- **Objekte**
  - repräsentieren Dinge der Vorstellungswelt oder sind Abstraktionen von realen Gegenständen.
- **Prolog kennt folgende Datenobjekte**
  - Prolog erkennt den Typ eines Objekts an seiner syntaktischen Struktur



Unifikation

- **Problem**
  - Wird eine Frage gestellt, so muss das Prolog-System entscheiden, ob der Term der Frage zu einem Term der Wissensbasis gleich ist oder nicht.
  - Nach Einführung von Termen mit Funktionssymbolen ist ein Mechanismus zum Auffinden „passender“ Regelköpfe zu definieren.
- **Prolog benutzt die *Unifikation*, um zu prüfen, ob zwei Terme miteinander übereinstimmen**
  - Zwei Terme sind ***unifizierbar***, falls es eine Substitution gibt, die die Terme gleich macht
  - Diese Substitution wird ***Unifikator*** genannt.
- **Beispiele**
  - t1 = `datum(D,M,1983)`, t2 = `datum(D1,may,Y1)`

    Die Substitution S = {D = D1,M = may, Y1= 1983} unifiziert t1 und t2

Resolutionsprinzip

- **Prolog versucht eine Anfrage auf der Menge der vorhandenen Fakten und Klauseln zu beweisen.**
  - Ein Faktum ist eine beweisbare Aussage
  - Es gilt
    - wenn jedes der Literale L1 , L2, L3, …, Ln beweisbar ist
    - und es eine Regel L :- L1 , L2, L3, …, Ln gibt
    - dann ist auch das Literal L beweisbar.
  - Beispiel:
    - ```
      istVaterVon(V,K) :- verheiratet(V,F) ,
                          istMutterVon(F,K).
      ```
  - Modus Ponens (Abtrennregel)
- **Resolutionsprinzip**
  - ist die Umkehrung des Modus Ponens
  - liefert ein Verfahren, um zu zeigen, ob eine Aussage beweisbar ist oder nicht.
      ```

Beweisstrategie

- **Resolution ist keine eindeutige Vorschrift**
  - Welches Literal einer Anfrage soll als nächstes bewiesen werden?
  - Durch welche Klausel soll ein Literal einer Anfrage abgeleitet werden?
- **Strategie**
  - A) Die Literale einer Anfrage werden von links nach rechts abgeleitet.
  - B) Die Klauseln werden von "oben" nach "unten" nach einer passenden durchsucht.
  - C) Falls das Prolog-System beim Beweisen in eine Sackgasse läuft, wird der letzte Ableitungsschritt rückgängig gemacht und die nächste passende Klausel zu einem neuen Ableitungsschritt verwendet (backtracking).

    Bem backtracking müssen Bindungen von Variablen, die beim Beweisschritt stattgefunden haben, rückgängig gemacht werden, damit neue Variablenbindung im alternativen Beweisschritt versucht werden kann.

Listen in Prolog

- **Eine Liste ist eine Folge von Objekten.**
  - Die Anzahl der Objekte einer Liste ist nicht festgelegt.
- **Eine Liste in Prolog ist entweder**
  - die leere Liste (dargestellt durch das Atom [])
  - die Struktur mit dem Funktor **.** und zwei Komponenten
    - die erste Komponente wird head genannt
    - die zweite Komponente ist wiederum eine Liste (tail)
- **Beispiele für Listen:**
  - []
  - . (10, [])
  - . (x, . (y, []))
  - . (a, . (b, . (c, [])))

Operatoren & Arithmetik

- **Prolog erlaubt die Verwendung von Operatoren.**
- **Operatoren werden in Infix-Notation geschrieben.**
  - z.B. 1+3 statt +(1,3)
- **Grundlegende arithmetische Operatoren sind**

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
mod	Modulo

- **Vergleichsoperatoren**

X > Y	X ist größer als Y
X >= Y	X ist größer als oder gleich Y
X < Y	X ist kleiner als Y
X =< Y	X ist kleiner als oder gleich Y
X =:= Y	X ist gleich Y
X =\= Y	X ist ungleich Y