Logikprogrammierung (in Lisp)

Thomas Rühr, Lorenz Mösenlechner

29. November 2011

Prolog

- Logische Programmiersprache (**Pro**grammation en **Log**ique)
- ursprünglich zur Verarbeitung natürlicher Sprache entwickelt
- gut geeignet für
  - logische Schlussfolgerungen (eingebaute Tiefensuche)
  - Verwaltung von Wissen
  - Unifikation von Termen (Pattern Matching)

Vergleich funktional - logisch

Funktion $\Leftrightarrow$ Logische Folgerung

Menge von Funktionen $\Leftrightarrow$ Fakten und Regeln

Funktionsberechnung $\Leftrightarrow$ Anfrage verifizieren

```
(defun member (e l)
  (unless (null l)
    (or (equal e (first l))
        (member e (rest l)))))
```

$\Leftrightarrow$

```
member(E,[E—R]).
member(E,[O—R]) :-
  member(E,R).
```

(member 2 '(1 2 3)) $\Leftrightarrow$ *member(2,[1, 2, 3])*

T $\Leftrightarrow$ yes

Bestandteile von Prolog

- Fakten

```
wrote(terry,shrdlu).
program(shrdlu).
```

- Regeln

```
programmer(X) :- wrote(X,Y), program(Y).
```

- Anfragen

```
?- wrote(terry,shrdlu).
?- wrote(terry,X).
?- wrote(X,shrdlu).
?- programmer(terry).
```

Prolog in Lisp

- Fakten:

```
(<- (wrote terry shrdlu))
(<- (program shrdlu))
```

- Regeln:

```
(<- (programmer ?x) (wrote ?x ?y) (program ?y))
```

- Anfragen:

```
(?- (programmer terry))
```

Mehrere Antworten (1)

- weitere Fakten

(<- (wrote joseph eliza))
(<- (program eliza))

- Anfrage

(?- (programmer ?x))
?X = TERRY
;
?X = JOSEPH
;
No.

-

Mehrere Antworten (2)

- Prolog gibt immer erste Anwort aus.
- Für weitere Antworten: ; (Semikolon)
- Keine weiteren Antworten erwünscht: . (Punkt)
- Alle Antworten auf einmal:

*(?- (***bagof** *?x (programmer ?x) ?result))*
?X = JOSEPH
?RESULT = (TERRY JOSEPH TERRY JOSEPH)
*(?- (***setof** *?x (programmer ?x) ?result))*
?X = JOSEPH
?RESULT = (TERRY JOSEPH)

Die Bedeutung von „nein“

- „Die Aussage stimmt nicht.“

```
wrote(terry,eliza).
```

- „Ich weiß es nicht.“

```
program(hacker).
```

- Gleiches Prinzip bei *not*

Listen

- leere Liste: *()*
- Konstruktor implizit durch Zugriffsfunktion:
  - *(?x . ?y)* ?x Kopf, ?y Rest der Liste
  - Prädikat für Konstruktion/Zugriff:

```
(<- (split-list (?x . ?y) ?x ?y))
(?- (split-list (1 2 3) ?f ?r))
?F = 1
?R = (2 3)
(?- (split-list ?r 3 (4 5)))
?R = (3 4 5)
```

Anonyme Variablen

- Manchmal ist man am Wert einer Variablen nicht interessiert.
- Beispiel: member

```
(<- (member ?x (?x . ?xs)))
(<- (member ?x (?y . ?ys)) (member ?x ?ys))
```

- Anonyme Variable: ?

```
(<- (member ?x (?x . ?_)))
(<- (member ?x (?_ . ?ys)) (member ?x ?ys))
```

Gleichheit

- == bedeutet Unifikation
- zum Rechnen („Zuweisung“): *is*
- Prolog rechnet nur, wenn man es dazu zwingt:

```
(<- (fak 0 1))
(<- (fak ?n ?m)
    (is ?y (- ?n 1)) (and (fak ?y ?x) (== ?m (* ?x ?n))))
(?- (fak 3 ?x))
?X = (* 3 (* 2 (* 1 1))).
```

```
(<- (fak ?n ?m)
    (is ?y (- ?n 1)) (and (fak ?y ?x) (is ?m (* ?x ?n))))
(?- (fak 3 ?x))
?X = 6
```

Vergleich Prolog - Lisp

- Gemeinsamkeiten
  - Eingebaute Listenverarbeitung
  - Automatische Speicherverwaltung
  - Dynamische Typisierung
  - Einheitliche Syntax für Programm und Daten
  - Interaktive Umgebung
  - Erweiterbarkeit
- Unterschiede
  - Lisp für allgemeine Zwecke
  - Prolog zum Aufbau einer Datenbasis und für logische Inferenzen