Funktionales Programmieren
(Praktische Informatik 3)

Berthold Hoffmann
Studiengang Informatik
Universität Bremen

Winter 03/04

Vorlesung vom 02.02.2004
Logisches Programmieren II;
Schlußbemerkungen

Inhalt der Vorlesung

- Organisatorisches
- Logisches Programmieren
  - Die Windsors und `rev` live!
  - Permutationen und Sortieren
  - Schnitt (cut)
  - Quicksort
  - `assert` und `retract`
- Rückblick und Ausblick

Organisatorisches

Der studienbegleitende Leistungsnachweis

- Bitte Scheinvordruck ausfüllen.
  - Siehe Anleitung.
  - Erhältlich vor FB3-Verwaltung (MZH Ebene 7)
  - Nur wer ausgefüllten Scheinvordruck abgibt, erhält auch einen.
- Bei Sylvie Rauer (MZH 8080) oder Christian Maeder (MZH 8055) abgeben (oder zum Fachgespräch mitbringen)

Das Fachgespräch

- Dient zur Überprüfung der Individualität der Leistung.
  - Insbesondere: Teilnahme an der Beabeitung der Übungsblätter.
  - Es ist keine Prüfung.
- Dauer: ca. 10 Min; einzeln, auf Wunsch mit Beisitzer
- Inhalt: Übungsblätter
- Bearbeitete Übungsblätter mitbringen — es werden zwei Aufgaben besprochen, die erste könnt Ihr Euch aussuchen.
- Termine:
  - Mo. 09.02–Fr. 13.02 — in den Tutorien eintragen

Logisches Programmieren

Vordefinierte Datentypen in Prolog

- Daten
  - Ganze Zahlen `1`, `42`
  - Gleitpunktzahlen `3.14`
  - Zeichen `'*'` (syntaktischer Zucker für $ord(*)$.)
  - Zeichenketten `‘hello! ‘  ‘‘World!’’`
- Operationen
  - `+`, `-`, `*`, `/`, `//`, `mod`
  - Vergleich `<`, `>`, `=<`, `>=`
  - Zuweisung `X is Y+3*Z`
- Operanden müssen vollständig instanziiert sein

Listen in Prolog

- Operationen
  - Leere Liste `[]`
  - Listenaufbau `[H|T]`
  - Lange Listen `[X1,12,13,X2]`
- Listen sind heterogen, weil ungetypt
  - `[1,node(L,3,leaf),[1,2,3],[],[1],[1,2]]`

Alte Bekannte

- Die Windsors

```
mother(queenMum,elizabethII).
...
father(charles,harry).
parent(P,C)     :- father(P,C);
                   mother(P,C).
grandma(GM,GC) :- mother(GM,P), parent(P,GC).
```

- Listenumkehr

```
rev  []    a = a
rev (h:t) a = rev t (h:a)
```

Listenverkettung

- Verketten von Listen

```
append([],X,X).
append([X|Xs],Y,[X|Z]) :- append(Xs,Y,Z).
```

- Unkonventionelle Benutzung von `append`

```
unordered(L) :- append(_,[X,Y|_],L), X>Y.
multiple(L) :-
      append(_,[X|L2],L), append(_,[X|_],L2).
```

Permutationen

- Permutationen einer Liste

```
permutation([], []).
permutation([X|Xs],Y) :-
        permutation(Xs,Z), insert(X,Z,Y).
```

- Einfügen in eine Liste

```
insert(Y,XZ,XYZ) :-
      append(X,Z,XZ), append(X,[Y|Z],XYZ).
```

- `permutation` ist symmetrisch
  `permutation(X, Y)` = `permutation(X, Y)`
  Praktisch gilt das nicht.

Sortieren, ganz abstrakt

- Sortiert?

```
sorted([]).
sorted([_]).
sorted([X,Y|Xs]) :- X<Y, sorted([Y|Xs]).
```

- Vertauschen, bis es sortiert ist!

```
nsort(X,Y) :- permutation(X,Y), sorted(Y).
```

Quicksort

- Wenn's schnell gehen soll!

```
qsort([],[]).
qsort([X],[X]).
qsort([X|Xs],Ys) :-
    partition(X,Xs,L,H),
    qsort(L,SL), qsort(H,SH),
     append(SL,[X|SH],Ys).
```

- Partitionieren

```
partition(_,[],[],[]).
partition(X,[Y|Ys],L,[Y|H]) :-
     X <  Y, partition(X,Ys,L,H).
partition(X,[Y|Ys],[Y|L],H) :-
     X >= Y, partition(X,Ys,L,H).
```

Quicksort, generisch

- Sortieren bzgl. einer Ordnungsrelation `R`

```
qSort([],R,[]).
qSort([X],R,[X]).
qSort([X|Xs],R,Ys) :-
    partitioN(X,Xs,L,R,H),
    qSort(L,R,SL), qSort(H,R,SH),
    append(SL,[X|SH],Ys).
```

- Partitionieren bzgl. einer Ordnungsrelation `R`

```
partitioN(_,[],[],_,[]).
partitioN(X,[Y|Ys],L,R,[Y|H]) :-
     compare(R,X,Y), partitioN(X,Ys,L,R,H).
partitioN(X,[Y|Ys],[Y|L],R,H) :-
                        partitioN(X,Ys,L,R,H).
```

- `compare(`r`,`l`,`r`)` ruft eine Ordnungsrelation $r \in \{<, =, >\}$ mit den Termen l und r auf.

Der Schnitt

- Kappen des Suchbaums
- Beispiel
  - `P(...) :- Q(...), !, R(...)`
  - Wenn die Klausel angewendet wird und `Q` gelang, werden keine weiteren Möglichkeiten für `Q` und `P` untersucht.
  - Kappen von fehlschlagenden Versuchen
  - Kappen von irrelevanten Lösungen
  - Vorsicht! Keine relevanten Lösungen kappen!

- `qsort` mit Schnitt

```
qqsort([],[]).
qqsort([X],[X]).
qqsort([X|Xs],Ys) :-
    partition(X,Xs,L,H),
    qqsort(L,SL), qqsort(H,SH), !,
    append(SL,[X|SH],Ys).
```

- `qqsort([···],X)` liefert genau eine Lösung
- `qqsort(U,[···])` liefert no

Negation

- closed world assumption
  - Was nicht definiert ist, gilt nicht!
- In Prolog: Negation by failure
  - `not(`$P(\cdots)$`)` liefert yes, wenn $P(\cdots)$ no liefert
  - Funktioniert nur, wenn $P(\cdots)$ einen endlichen Suchbaum hat
  - Beispiel `not(member(X,Xs))` könnte für Aufgabe 14 nützlich sein.

- Beispiel `partition`.

```
partitiON(_,[],[],_,[]).
partitiON(X,[Y|Ys],L,R,[Y|H]) :-
     compare(R,X,Y), partitiON(X,Ys,L,R,H).
partitiON(X,[Y|Ys],[Y|L],R,H) :-
     not(compare(R,X,Y)), partitiON(X,Ys,L,R,H).
```

Der Baron lässt grüßen!

- Metaprogrammierung
- Klauseln sind auch nur Terme
  `':-'(p(···),','(q(···),r(···),))` ist ein Term
- `asserta(`K`)` und `assertz(`K`)` fügen die Klausel K vorne bzw. hinten ins Programm ein.
- `retract(`K`)` entfernt alle Klauseln K', die sich mit K unifizieren lassen.
- Damit lässt sich richtig zaubern!

Zusammenfassung und Ausblick

Einsatzfelder von Prolog

- Datenbanken
- Expertensysteme
- Grammatiken für natürliche Sprachen

Stärken und Schwächen von Prolog

- Stärken
  - einfach
- Schwächen
  - keine Typisierung
  - keine Funktionen (deterministische Berechnungen)
  - `assert` und `retract` sind maschinennah
  - ineffizient (?)

Logisches Programmieren nach Prolog

- + Typen + Funktionen
  - Mercury (Melbourne)
  - Gödel (Bristol)
- + . . . + Nebenläufigkeit + Objektorientiertheit
  - Oz und Mozart (MPI Saarbrücken)

Hilfe!

- Haskell: primäre Entwicklungssprache an der AG BKB
  - Entwicklungsumgebung für formale Methoden (Uniform Workbench)
  - Werkzeuge für die Spezifikationssprache CASL
- Wir suchen studentische Hilfskräfte
  - für diese Projekte
- Wir bieten:
  - Angenehmes Arbeitsumfeld
  - Interessante Tätigkeit
- Wir suchen Tutoren für PI3
  - im WS 04/05 — meldet Euch!