Universität Bonn
Institut für Informatik III
Prof. Dr. R. Manthey
Dr. A. Behrend
Dipl.-Inform. C. Dorau

2.6.2008

Übungen
Deskriptive Programmierung
SS 2008

Blatt 4

Aufgabe 1 (Prolog-Tutorial)**.**

1. Definieren Sie die Relationen `prefix/2` und `suffix/2`, welche jeweils einen Präfix bzw. Suffix einer Liste bestimmen.
2. Definieren Sie eine Relation `nth_member/3`, welche das n-te Element einer Liste ermittelt.
3. Definieren Sie eine Relation `sublist/2`, die eine Teilliste aus einer gegebenen Ausgangsliste bestimmt.
4. Definieren Sie eine Relation `halves/3`, die eine Liste in zwei gleich große Hälften (wenn möglich) aufteillt.
5. Definieren Sie die Prädikate `even_list/2` und `odd_list/2`, die jeweils die geraden bzw. ungeraden Elemente einer Liste (bzgl. ihrer Position in der Liste) als neue Liste ausgeben.
6. Definieren Sie eine Relation `reverse/2`, die eine gegebene Liste umkehrt.
7. Definieren Sie eine Relation `perm(List,Perm_of_List)`, die eine Permutation einer Eingabeliste bestimmt.
8. Definieren Sie Prädikate, die jeweils gegeben eine Liste die Menge der Kombinationen und der Variationen mit und ohne Wiederholung der Listenelmente bestimmen.
9. Definieren Sie eine Relation `merge/3`, die gegeben zwei sortierte Eingabelisten durch Mischen eine sortierte Ausgabeliste bestimmt.

Aufgabe 2 (Prolog-Tutorial)**.**

1. Machen Sie sich vertraut mit den Meta-Prädikaten *var/1, nonvar/1, functor/3, arg/3* und dem univ-Operator *../2*, indem Sie folgende Beispiele unter SWI-Prolog testen:

   UNIV-Operator

   ```
   ?- f(a,b,c)=..X.
   ?- X=..[a,b,c].
   ?- [a=b,a+b]=..[Functor,Arg1|Args],Arg1=..X.
   ```

   VAR/1 und NONVAR/1

   ```
   ?- var(X).
   ?- var(g(X)).
   ?- nonvar(X).
   ?- nonvar(var(X)).
   ```

   FUNCTOR/3 und ARG/3

   ```
   ?- functor(f(1,2),f,2).
   ?- functor(f(1,2),F,A).
   ?- functor(X,f,A).
   ?- functor(T,.,2).
   ?- functor(T,f,3).

   ?- arg(1,f(a,X),Res).
   ?- arg(2,foo(boo,moo),moo).
   ?- arg(2,[a,b,c],Res).
   ?- arg(1,[a,b,c],Res).
   ?- arg(2,term1(term2(a,b),c),c).
   ```

2. Probieren Sie folgende Anfragen mit dem `findall/3`-Prädikat aus und erklären Sie die Ergebnisse:

   ```
   ?- findall(X,(member(X,[a,b,c,d]), not member(X,[b,e,d,f])),Result).
   ?- findall(X,member(a,[b,d,e]),Result).
   ?- findall(_,(member(X,[a,b,c]),assert(found(X))),_),
      findall(X,(retract(found(X))),Result).
   ?- findall(X,member(Y,[a,b,c]),Result),Result=[a|T].
   ```

Aufgabe 3 (Haskell-Tutorial*)**.** Mergesort ist (wie Quicksort) ein Divide-And-Conquer-Algorithmus. Dabei wird im divide-Schritt die zu sortierende Inputliste sukzessive in Teillisten zerlegt, bis diese nur noch die Länge 0 oder 1 haben. Danach werden im conquer-Schritt die sortierten Teillisten wieder konkateniert, wobei jeweils 2 sortierte Inputlisten zu einer sortierten Outputliste vermischt werden.

1. Definieren Sie in Haskell eine Funktion `merge/2`, die aus 2 gegebenen sortierten Listen durch Mischen eine sortierte Outputliste bestimmt.

2. Definieren Sie nun eine Funktion `merge_sort/1`, die aus einer unsortierten Liste die sortierte Liste bestimmt.

3. Ändern Sie ihr Programm so, dass Duplikate in der unsortierten Inputliste ignoriert werden.