Probeklausur Einführung in die symbolische Programmierung 2010 Vorlesung

Aufgabe 1: zur Sortierung 10 Punkte

1.1) Schreiben Sie eine Prozedur um zwei sortierte Listen in eine dritte Liste zu sortieren.
(benutzen Sie keine speziellen Built-In Sortierungsprädikate)
?- merge([2,5,6,6,8],[1,3,5,9],L).
L=[1,2,3,5,5,6,6,8,9]

```
%1.1)
merge([X|L1],[Y|L2],[X|L3]):-
        X<Y,!,merge(L1,[Y|L2],L3).
merge([X|L1],[Y|L2],[X,Y|L3]):-
        X=Y,!,merge(L1,L2,L3).
merge([X|L1],[Y|L2],[Y|L3]):-
        X>Y,!,merge([X|L1],L2,L3).
merge([],L2,L2):-!.
merge(L1,[],L1).
```

1.2) Eliminieren Sie nun doppelte Einträge aus der Liste.

```
%1.2)

remove_dups([], []).
remove_dups([First | Rest], NewRest) :-
          member(First, Rest),!, remove_dups(Rest, NewRest).
remove_dups([First | Rest], [First | NewRest]) :-
not(member(First, Rest)), remove_dups(Rest, NewRest). %wegen cut not member unnötig

%kuerzer moeglich da sortierte Liste
remove_dups2([],[]).
remove_dups2([_],[_]).
remove_dups2([First,First|Rest],NoDuplicates) :- !,
              remove_dups2([First|Rest],NoDuplicates).
remove_dups2([First|Rest],[First|NoDuplicates]) :-
              remove_dups2(Rest,NoDuplicates).
```

direkt in merge haben wir nur eine Lösung für Listen ohne Duplikate in den Einzellisten
für Listen mit [1,2,2,3] und [1,2,2,3] bräuchte man ein Lookahead oder einen Stack.

```
merge2([X|L1],[Y|L2],[X|L3]):-
        X<Y,!,merge2(L1,[Y|L2],L3).
merge2([X|L1],[Y|L2],[X|L3]):-
        X=Y,!,merge2([X|L1],L2,L3).
merge2([X|L1],[Y|L2],[Y|L3]):-
        X>Y,!,merge2([X|L1],L2,L3).
merge2([],L2,L2):-!.
merge2(L1,[],L1).
```

1.3) Schreiben Sie ein Programm, das auch unsortierte Listen sortieren kann
Standardsortieralgorithmus siehe Vorlesung, bubblesort, insertionsort,quicksort.
1.4) Die in der Vorlesung vorgestellte Variante von Quicksort funktioniert schlecht (Zeitkomplexität

für bereits sortierte Listen.) Begründen Sie warum.
/* Bei QuickSort wird die Liste mithilfe des Pivot-Elements an einem willkürlichen Punkt in zwei Teile geteilt: die kleineren Elemente, die gleichen oder größeren Elemente. Diese zwei Teillisten werden sortiert und danach zur Ergebnisliste verbunden. Im bester Fall wird das Trennelement so gewählt, dass die beiden entstehenden Teillisten etwa gleich groß sind. In diesem Fall gilt für die Laufzeit für O(n*log(n)). Im schlechtesten Fall wird das Trennelement so gewählt, dass es das größte oder das kleinste Element der Liste ist, dann ist die Laufzeit quadratisch. Für sortierte Listen Trennt das Pivot immer in eine leere und eine Liste mit allen restlichen Elementen da wir für die Prologimplementation den Kopf der Liste verwenden. Für aufsteigende Sortierung sind alle Elemente grösser, also in Big für absteigende alle kleiner also in Small*/

Aufgabe 2: zum Programmablauf 10 Punkte

Betrachten Sie folgendes Beispielprogramm:

```
p(4).
p(7) :- !.
p(9).
```

Schreiben Sie alle Antworten von Prolog (Backtracking mit ';'ausglösen, nachdem eine Antwort gegeben wurde). Achten Sie auf die richtige Reihenfolge in der Prolog die Antworten gibt.

2.1) ?- p(X).

2.2) ?- p(X), p(Y).

2.3) ?- p(X),!, p(Y).

```
/*
2.1)
?- p(X).
X = 4 ;
X = 7.

2.2)
?- p(X),p(Y).
X = 4,
Y = 4 ;
X = 4,
Y = 7 ;
X = 7,
Y = 4 ;
X = 7,
Y = 7.
2.3)
?- p(X),!,p(Y).
X = 4,
Y = 4 ;
X = 4,
Y = 7.
*/
```

Aufgabe 3: was ist ein Prologprogramm 15 Punkte

3.1) woraus besteht ein Prologprogramm

3.2) erklären Sie die grundlegende Abarbeitungsweise eines Prologprogramms

3.3) erklären Sie anhand von append die Wirkungsweise eines Akkumulators

3.4) erklären Sie anhand der Negation die Wirkungsweise des Cut Prädikats.

Aufgabe 3:
3.1)
Ein Prolog Programm ist eine geordnete Menge von Klauseln. Klauseln können entweder Fakten oder Regeln sein.
Fakten sind von Natur aus wahr,wie mann(tobias).
Regeln: Kopf:-Rumpf,wie vater(X,Y):-mann(X),kind(Y,X).(X,Y sind Variablen) Der Kopf einer Regel gilt, wenn alle Bedingungen des Regelrumpfes erfüllt sind.
3.2)
Um eine Anfrage zu beantworten sucht der Prolog Interpreter im gegebenen Programm von oben nach unten nach einer Klausel, deren Kopf sich mit der Anfrage syntaktisch zur Deckung bringen lässt.Wenn es sich um ein Faktum handelt, ist der Beweis gelungen. Bei einer Regel wird in der Anfrage der Regelkopf durch den zugehörigen Regelrumpf ersetzt, dessen Teille dann nacheinander von links nach rechts auf die gleiche Weise zu beantworten versucht.

3.3)

Bsp: Umkehrung einer Liste mit append
```
reverse_liste([],[]).
reverse_liste([X|Xs],Ys):-reverse_liste(Xs,Zs),append(Zs,[X],Ys).
```
Nachteil von mit append:
1.Die Rechenzeit steigt linear mit der Länge der ersten Liste,da diese komplett abgebaut wird, um Elementweise vorne an die zweite angehängt zu werden.
2.mehrmalige rekursive Aufrufe kosten die Zeit.

Akkumulator ist eine Hilfsvariable,um Ergebnisse zu sammeln und am Schluss auszugeben.Durch die Akkumulatortechnik wird die Anzahl der rekursiven Aufrufe minimiert.
```
reverse_liste(L1,L2):-reverse_acc(L1,[],L2).
reverse_acc([X|R],Acc,L2):-reverse_acc(R,[X|Acc],L2).
reverse_acc([],Acc,Acc).
```

3.4
Der cut ! Operator wird dazu benutzt, um das Backtracking zu steuern.
Ist der Cut einmal eingesetzt, kann links von ihm kein Backtracking mehr
stattfinden.
Bsp: not(X):-X,!,fail.
not(X).
Wenn es eine Belegung für X gibt, dann wird not(X) mit fail bewertet und
Backtracking ist nicht erwünscht. Wenn es keine Belegung für X gibt,
kommt die zweite Klausel für not(X) durch und die Anfrage wird mit 'yes' bewertet.

Aufgabe 4: Grammatik 15 Punkte

Gegeben Sei die folgende Grammatik

```
s --> np,vp.
np --> det, n.
vp --> v, np.
det --> [a].
det --> [the].
n    --> [cat].
n --> [mouse].
v --> [scares].
v --> [hates].
```

4.1) Zeigen Sie dass folgender Satz von der Grammatik akzeptiert wird: [the, mouse, scares, the, mouse]. Beschreiben Sie welche Prologregeln wann aufgerufen werden. Erklären Sie wie es zur Abarbeitung der Terminale kommt.
Aufruf ist: ?- s([the, mouse, scares, the, mouse],[]).

4.1
1) s --> np,vp.
2) np --> det,n.
3) det -->[the].
4) n --> [mouse].
np:[the mouse] wird akzeptiert.
5) vp --> v,np.
6) v -->[scares].
7) np --> det,n.
8) det -->[the].
9) n --> [mouse].
vp:[scares the mouse] wird akzeptiert.
Der ganze Satz: 'the mouse scares the mouse' wird akzeptiert.

4.2) Benutzen Sie nun nur die Nominalphrase der Grammtik einschliesslich ihrer Elemente mit einem Beispielterminal nämlich [mouse]. Geben Sie die Schreibweise ohne DCG Pfeile im internen Format von Prolog an (DCG Regeln in Standardprolog aufgelöst)

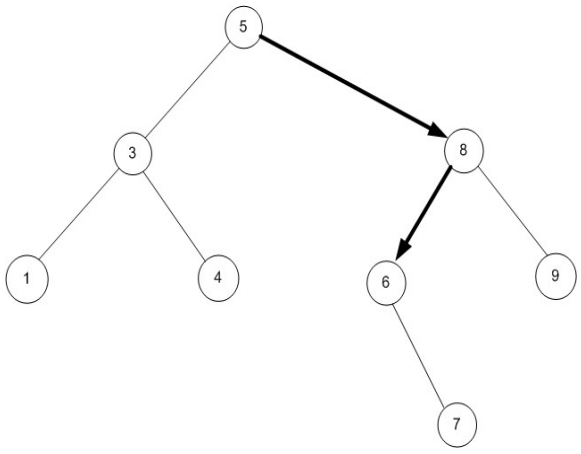
```
%4.2
?-np([the,mouse],[]).
np(A, C) :-
      det(A, B),
      n(B, C).
det([the|A], A).
n([mouse|A], A).
```

4.3) Implementieren Sie nun den Plural für die obige Grammatik (mouse – mice, cat -- cats, scares – scare, hates – hate) geben Sie die notwendigen ergänzenden Grammatikregeln an. Führen Sie in das DCG-Format eine Variable ein, die die Numerus Kongruenz sicherstellt, also sicherstellt wenn das Nomen Plural ist, dass auch das Verb Plural ist (Sätze wie [the, mice scares, the, cat] sollen also ungrammatisch sein).

```
s-->np(Num),vp(Num).
np(Num)-->det(Num),n(Num).
vp(Num)-->v(Num),np(_).
det(sg) --> [a].
det(sg) --> [the].
det(pl) --> [the].
n(sg) --> [cat].
n(pl) --> [cats].
n(sg) --> [mouse].
n(pl)--> [mice].
v(sg) --> [scares].
v(p)-->[scare].
v(sg) --> [hates].
```

Zusatzaufgabe 5: Datenstrukturen 10 Punkte

Gegeben ist ein binärer Baum D = t(Left,X,Right), der von links nach rechts sortiert ist. Dies bedeutet alle Knoten im linken Subbaum sind kleiner als X und alle im rechten Subbaum sind grösser als X. Beide Subbäume sind wieder sortiert.



Gegeben Sei folgendes Prologprogramm:

```
in(X,t(_,X,_))).
in(X,t(Left,Root,Right)) :-
Root > X,
in(X,Left).
in(X,t(Left,Root,Right)) :-
Y > Root,
in(X,Right).
```

5.1) erklären Sie kurz was dieses Programm macht.

%5.1
Wenn X mit der Wurzel von Baum D unifiziert, dann ist X gefunden.
Wenn X kleiner als die Wurzel von Baum D ist, dann wird X rekursiv im linken Teilbaum des Baums D gesucht, sonst wird X rekursiv im rechten Teilbaum des Baums D gesucht.

%5.2) geben Sie die aufgerufenen Regeln + Variablenbelegungen für folgenden Aufruf an:
?- in(6,D). Wobei D der oben gezeichnete Baum mit der Wurzel 5 sein soll.

%5.2
?-in(6,t(Left,5,Right)).
1)6>5
2)in(6,Right)
3)in(6,t(Left,8,Right))
4)8>6
5)in(6,Left)
6)in(6,t(Left,6,Right)).
7)true.