Lösung.

Aufgabe 10

10.1) Implementieren Sie ein Prädikat loesche_alle, das als Eingabe eine flache Liste akzeptiert und das aus dieser Liste alle Vorkommen eines bestimmten einfachen Objekts (Atom/Zahl) löscht – im Unterschied zu dem in der Vorlesung gezeigten Prädikat, das nur jeweils ein Vorkommen löscht.

```
loesche_alle(X,[ ],[ ]).
loesche_alle(X,[X|R],L ) :- loesche_alle(X,R,L).
loesche_alle(X,[Y|R],[Y|L]) :- loesche_alle(X,R,L).
```

10.2) Schreiben Sie ein Prädikat loesche_letztes, das die Akkumulatortechnik benutzt um das letzte Element einer Liste zu löschen.

```
[_],Acc,Acc.
loesche_letztes([X|R],[X|Acc],L) :-
```

10.3) Erweitern Sie das Programm so, dass es nun die Liste wieder in der ursprünglichen Ordnung ausgibt: also [1,2,3,4] → [1,2,3]
10.4) Erweitern Sie das Prädikat aus 10.1 so, dass es auch tiefer strukturierte Listen (Listen, die weitere Listen enthalten) akzeptiert und eine flache Liste ausgibt, aus der alle Vorkommen des angegebenen Objekts gelöscht sind. Falls Sie möchten, können Sie für diese Aufgabe das built-in Prädikat atomic(X) verwenden. Eine Möglichkeit wäre, die Liste flachzumachen und dann erst alle Elemente zu löschen, oder aber schon "unterwegs" das löschen durchzuführen.

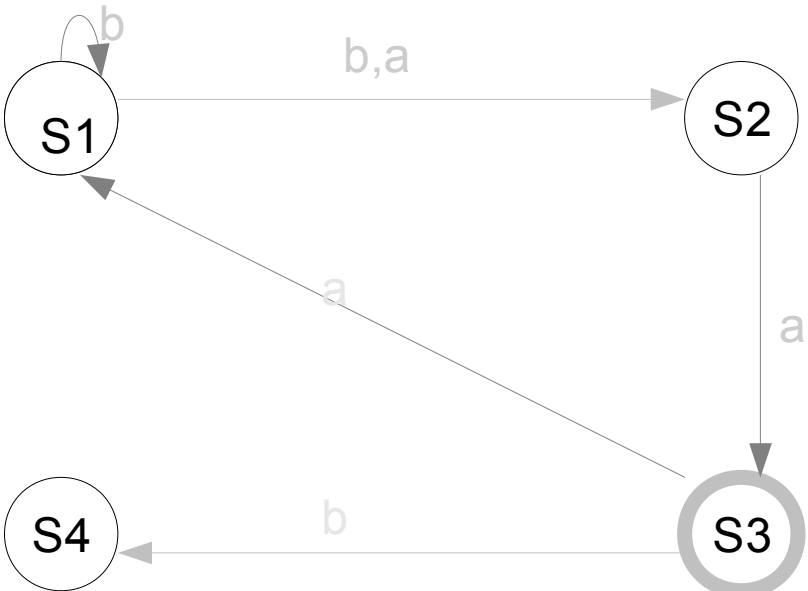
```
%10.1)
loesche_all(_,[],[]).
loesche_all(X,[X|R],L):- loesche_all(X,R,L).
loesche_all(X,[Y|R],[Y|L]):- X\=Y,loesche_all(X,R,L).
```

```
%10.2) & 10.3)
reverse(L,R) :- reverse_acc(L,[],R).

reverse_acc([X|L],Acc,R):- reverse_acc(L,[X|Acc],R).
reverse_acc([],Acc,Acc).

loesche_letzte(L1,L):-loesche_letzte(L1,[],L2),
                      reverse(L2,L).
loesche_letzte([_],Acc,Acc).
loesche_letzte([X|R],Acc,L):-loesche_letzte(R,[X|Acc],L).



%10.4)
%Lösung 1
verflach([],[]).
verflach([X|R],[X|L]):-atomic(X),verflach(R,L).
verflach([X|R],L):-verflach(X,L1),verflach(R,L2),append(L1,L2,L).
loesche_all_verflach(X,Eingabe,Ausgabe):-verflach(Eingabe,Zwisch),
                                         loesche_all(X,Zwisch,Ausgabe).
%----------------------------
%Lösung 2
loesche_all(_,[],[]):-!.
loesche_all(X,[X|R],L):-
        loesche_all(X,R,L).

loesche_all(X,[Y|R],[Y|L]):- dif(X,Y),atomic(Y),!,loesche_all(X,R,L).
loesche_all(X,[Y|R],L):-append(Y,R,L1),loesche_all(X,L1,L).
%----------------------------
%Lösung 3
loesche_all(_,[],[]):-!.
loesche_all(X,[X|R],L):-
        loesche_all(X,R,L).
loesche_all(X,[Y|R],[Y|L]):- dif(X,Y),atomic(Y),!,loesche_all(X,R,L).
loesche_all(X,L1,L):-append_liste(L1,L2),loesche_all(X,L2,L).
```

Aufgabe 11

11.1) Implementieren Sie einen endlichen Automaten in Prolog, der dem unten angegebenen Graphen entspricht.

11.2) Geben Sie alle Strings der Länge 4 an, die der Automat als Eingabe akzeptiert.



```
%11)
final(s3).
trans(s1,a,s2).
trans(s1,b,s2).
trans(s1,b,s1).
trans(s2,a,s3).
trans(s3,b,s4).
trans(s3,a,s1).
%silent(s2,s4).
%silent(s3,s1).
accept(State,[]):-final(State).
accept(State,[X|Reststring]) :- trans(State,X,State1),
accept(State1,Reststring).

%?- L=[_,_,_,_],accept(State,L).
%L = [b, b, a, a],
%State = s1 ;
%L = [b, b, b, a],
%State = s1 ;
%L = [a, a, a, a],
%State = s2 ;
%L = [a, a, b, a],
%State = s2 ;
%L = [a, b, a, a],
%State = s3 ;
%L = [a, b, b, a],
%State = s3 ;
%false.
```

Aufgabe 12

12.1) Schreiben Sie ein Prologprädikat, vorkommen/3 das zählt wie oft ein Element in einer Liste vorkommt.

12.2) Implementieren Sie das Prädikat mit Akkumulator

```
%12.1)
vorkommen_zahl(_,[],0).
vorkommen_zahl(X,[X|R],M):-
vorkommen_zahl(X,R,N), M is N+1.
vorkommen_zahl(X,[Y|R],M):-dif(X,Y),vorkommen_zahl(X,R,M).
```

%12.2)

```
vorkommen_zahl_acc(X,L,M):-vorkommen_zahl_acc(X,L,0,M).
vorkommen_zahl_acc(_,[],Acc,Acc).
vorkommen_zahl_acc(X,[X|R],Acc,M):-Acc1 is Acc+1,vorkommen_zahl_acc(X,R,Acc1,M).
vorkommen_zahl_acc(X,[Y|R],Acc,M):-dif(X,Y),vorkommen_zahl_acc(X,R,Acc,M).
```