```
% Eimerproblem
move((E8,E5,E3), E5 ausschuetten, (E8,0,E3)) :- E5 > 0.

move((E8,E5,E3), E8 -> E3, (0,E5,R3)) :- E8 > 0, E8 + E3 < 3, R3 is E8 + E3.
move((E8,E5,E3), E8 -> E3, (R8,E5,3)) :- E8 > 0, 3 =< E8 + E3, R8 is E8-(3-E3).

solve(X,X,[],_).
solve(X,Y,[M|Ms],Fs) :- move(X,M,F),not(member(F,Fs)),solve(F,Y,Ms,[F|Fs]).


% Ziege-Kohl
gegenueber(links,rechts).
gegenueber(rechts,links).

harmlos(_,X,Y) :- gegenueber(X,Y).
harmlos(Ufer,Ufer,Ufer).

erlaubt((Mann,Ziege,Wolf,Kohl)) :- harmlos(Mann,Ziege,Wolf), harmlos(Mann,Ziege,Kohl).

fahrt((U,U,Wolf,Kohl), "Ziege", (UNeu,UNeu,Wolf,Kohl)) :- gegenueber(U,UNeu).
fahrt((U, Ziege,Wolf,Kohl), "leer", (UNeu,Ziege,Wolf,Kohl)) :- gegenueber(U,UNeu).

erreichbar(S,_,[],S).
erreichbar(S,Besucht,[Fahrt|Fahrten],Z) :-
fahrt(S,Fahrt,ZwischenS), erlaubt(ZwischenS), not(member(ZwischenS,Besucht)),
erreichbar(ZwischenS,[ZwischenS|Besucht],Fahrten,Z).

loesung(Fahrten) :- start(S), ziel(Z), erreichbar(S,[],Fahrten,Z).
start((links,links,links,links)).
ziel((rechts,rechts,rechts,rechts)).


% splits
splits(L, ([], L)).
splits([X|L], ([X|S], E)) :- splits(L, (S, E)).

splitsAlt(L, (Xs, Ys)) :- append(Xs, Ys, L).


% treemember wit binary search
treemember(X,tree(K,L,R)) :- X = K.
treemember(X,tree(K,L,R)) :- X < K,treemember(X,L).
treemember(X,tree(K,L,R)) :- K < X,treemember(X,R).
```

```
% coin changer
remove([(X,A)|L],X,[(X,ANew)|L]) :- A>0, ANew is A-1.
remove([X|L],Y,[X|L1]) :- remove(L,Y,L1).

change(0,_,[]).
change(X,CoinsAvailable,ChangeNew) :-
remove(CoinsAvailable,C,CANew), C=<X, XNew is X-C,
change(XNew,CANew,Change), put(Change,C,ChangeNew).


% Labyrinth
% Aus Vorlesung:
member(X, [X|_]).
member(X, [_|R]) :- member(X, R).

% Aufgabenteil a)
lengthof([], 0).
lengthof([_|R], NewLength) :- lengthof(R, Length), NewLength is Length + 1.

% Aufgabenteil b)
nachbar((X, Y), osten, (XNeu, Y)) :- XNeu is X + 1, frei((XNeu, Y)).
nachbar((X, Y), westen, (XNeu, Y)) :- XNeu is X - 1, frei((XNeu, Y)).
nachbar((X, Y), norden, (X, YNeu)) :- YNeu is Y + 1, frei((X, YNeu)).
nachbar((X, Y), sueden, (X, YNeu)) :- YNeu is Y - 1, frei((X, YNeu)).

% Vorgegeben :
wege(Schritte, MaxSchritte) :- start(Start), ziel(Ziel),
findeweg(Start, Ziel, MaxSchritte, [], Schritte).

% Aufgabenteil c)
findeweg(Ziel, Ziel, _, _, []).
findeweg(Start, Ziel, MaxSchritte, Besucht, [Schritt|Schritte]) :-
lengthof(Besucht, NumBesucht), NumBesucht < MaxSchritte,
nachbar(Start, Schritt, Nachbar), not(member(Nachbar, Besucht)),
findeweg(Nachbar, Ziel, MaxSchritte, [Nachbar|Besucht], Schritte).

% Alternative (ohne lengthof):
findeweg2(Ziel, Ziel, _, _, []).
findeweg2(Start, Ziel, MaxSchritte, Besucht, [Schritt|Schritte]) :-
MaxSchritte > 0,
nachbar(Start, Schritt, Nachbar),
not(member(Nachbar, Besucht)),
NeuMaxSchritte is MaxSchritte - 1,
findeweg2(Nachbar, Ziel, NeuMaxSchritte, [Nachbar|Besucht], Schritte).
```