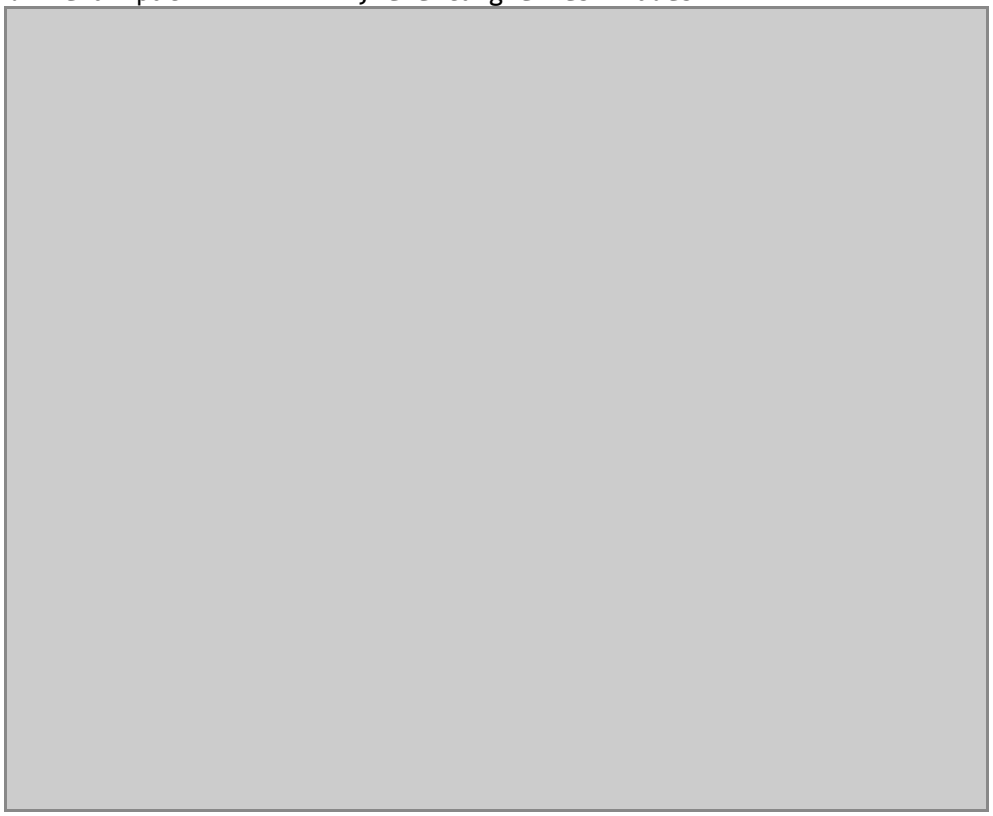
```
% Die Schnittstelle umfasst
%   start_description   ;Beschreibung des Startzustands
%   start_node          ;Test, ob es sich um einen Startknoten handelt
%   goal_node           ;Test, ob es sich um einen Zielknoten handelt
%   state_member        ;Test, ob eine Zustandsbeschreibung in einer Liste
%                        von Zustandsbeschreibungen enthalten ist
%   expand              ;Berechnung der Kind-Zustandsbeschreibungen
%   eval-path           ;Bewertung eines Pfades
```



```

start_description([
  block(block1),
  block(block2),
  block(block3),
  block(block4),  %mit Block4
  on(table,block2),
  on(table,block3),
  on(block2,block1),
  on(table,block4), %mit Block4
  clear(block1),
```

```
% Die Schnittstelle umfasst
%   start_description   ;Beschreibung des Startzustands
%   start_node          ;Test, ob es sich um einen Startknoten handelt
%   goal_node           ;Test, ob es sich um einen Zielknoten handelt
%   state_member        ;Test, ob eine Zustandsbeschreibung in einer Liste
%                        von Zustandsbeschreibungen enthalten ist
%   expand              ;Berechnung der Kind-Zustandsbeschreibungen
%   eval-path           ;Bewertung eines Pfades
/*
%<3 Blöcke>
start_description([
  block(block1),
  block(block2),
  block(block3),
  on(table,block2),
  on(table,block3),
  on(block2,block1),
  clear(block1),
  clear(block3),
  handempty
]).
goal_description([
  block(block1),
  block(block2),
  block(block3),
  on(table,block3),
  on(table,block1),
  on(block1,block2),
  clear(block3),
  clear(block2),
  handempty
]).
%</3 Blöcke>
 */

%<4 Blöcke>
start_description([
  block(block1),
  block(block2),
  block(block3),
  block(block4),
  on(table,block2),
  on(table,block3),
  on(block2,block1),
  on(table,block4),
  clear(block1),
```

```
  clear(block3),
  clear(block4), %mit Block4
  handempty
 ]).

goal_description([
  block(block1),
  block(block2),
  block(block3),
  block(block4), %mit Block4
  on(block4,block2), %mit Block4
  on(table,block3),
  on(table,block1),
  on(block1,block4), %mit Block4
%  on(block1,block2), %ohne Block4
  clear(block3),
  clear(block2),
  handempty
 ]).



start_node((start,_,_)).

goal_node((_,State,_)):-
  "Zielbedingungen einlesen"
  "Zustand gegen Zielbedingungen testen".


% Aufgrund der Komplexität der Zustandsbeschreibungen kann
state_member nicht auf
% das Standardprädikat member zurückgeführt werden.
%
state_member(_,[]):- !,fail.

state_member(State,[FirstState|_]):-
  "Test, ob State bereits durch FirstState beschrieben war. Tipp:
Eine

  Lösungsmöglichkeit besteht in der Verwendung einer
Mengenoperation, z.B. subtract"  ,!.

%Es ist sichergestellt, dass die beiden ersten Klauseln nicht
zutreffen.
state_member(State,[_|RestStates]):-
```

```
  clear(block3),
  clear(block4),
  handempty
]).

goal_description([
  block(block1),
  block(block2),
  block(block3),
  block(block4),
  on(block4,block2),
  on(table,block3),
  on(table,block1),
  on(block1,block4),

  clear(block3),
  clear(block2),
  handempty
]).
%</4 Blöcke>



start_node((start,_,_)).

goal_node((_,State,_)):-
  %used TODO:"Zielbedingungen einlesen"
  goal_description(Goal),
  %used TODO:"Zustand gegen Zielbedingungen testen"
  state_member(State,[Goal]).


% Aufgrund der Komplexität der Zustandsbeschreibungen kann state_member
nicht auf
% das Standardprädikat member zurückgeführt werden.
%
state_member(_,[]):- !,fail.

state_member(State,[FirstState|_]):-
  %used TODO:"Test, ob State bereits durch FirstState beschrieben war. Tipp:
Eine Lösungsmöglichkeit besteht in der Verwendung einer Mengenoperation,
z.B. subtract"  ,!.
  mysubset(State,FirstState).

%Es ist sichergestellt, dass die beiden ersten Klauseln nicht zutreffen.

state_member(State,[_|RestStates]):-
```

```
  "rekursiver Aufruf".


eval_path([(_,State,Value)|RestPath]):-
  eval_state(State,"Rest des Literals bzw. der Klausel"
  "Value berechnen".


```

```
 %used TODO:"rekursiver Aufruf".
 state_member(State,RestStates).

%<TEIL 2>
    %eval_path([(X,State,Value)|RestPath]):-
      eval_path(Path):-
        %used TODO:eval_state(State,"Rest des Literals bzw. der Klausel"
        length(Path,G),
        eval_state(Path,G).

    %<Suchverfahren>     //Ein Algorithmus auswählen

      %used TODO: A-Algorithmus
      /*  eval_state([(X,State,Value)|_],G) :-
          heuristik(State,H),
          F is G + H, %A-Algorithmus: f(n) = g(n) + h(n)
          %nl,write(['Moeglichkeit: ', X, ' Kosten: ', F]),  %ToDebug
          Value = F.
      */

      %used TODO: gierige Bestensuche
        eval_state([(_X,State,Value)|_],_) :-
          heuristik(State,H),
          %nl,write(['Moeglichkeit: ', X, ' Kosten: ', H]),  %ToDebug
          Value = H.


      %used TODO: optimistisches Bergsteigen
      %in Suche_Modul_Allgemein.pl gelöst!

      %used TODO: Bergsteigen mit Backtracking
      %in Suche_Modul_Allgemein.pl gelöst!

    %</Suchverfahren>

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %<Heuristiken>
      % Anzahl der Elemente des Ziels, die noch an der falschen Position sind.
     /* heuristik(State,Value) :-
        goal_description(Ziel),
        differenzmenge(Ziel,State,Differenz),
        length(Differenz,Value).
        */

      % Anzahl der Elemente, die schon an der richtigen Position sind
      heuristik(State,Value) :-
        goal_description(Ziel),

        schnittmenge(State,Ziel,Schnitt),
        length(Schnitt,AnzSchnitt),
        length(Ziel,AnzZiel),
```

```
139         Value is (AnzZiel - AnzSchnitt).
140
141
142     %</Heuristiken>
143
144
145     % Mengenoperationen
146     %-------------------
147     % differenzmenge(Menge_A, Menge_B, Differenzmenge)   "A ohne B"
148     differenzmenge([], _, []).
149     differenzmenge([Element|Tail], Menge, Rest) :-
150       member(Element, Menge), !,
151       differenzmenge(Tail, Menge, Rest).
152     differenzmenge([Head|Tail], Menge, [Head|Rest]) :-
153       differenzmenge(Tail, Menge, Rest).
154
155     % schnittmenge(Menge_A, Menge_B, Menge_C)
156     schnittmenge([], _, []).
157     schnittmenge([Element|Tail], Liste, Schnitt) :-
158       member(Element, Liste), !,
159       Schnitt = [Element|Rest],
160       schnittmenge(Tail, Liste, Rest).
161     schnittmenge([_|Tail], Liste, Rest) :-
162        schnittmenge(Tail, Liste, Rest).
163 %</TEIL 2>
164
165 action(pick_up(X),
166        [handempty, clear(X), on(table,X)],
167        [handempty, clear(X), on(table,X)],
168        [holding(X)]).
169
170 action(pick_up(X),
171        [handempty, clear(X), on(Y,X), block(Y)],
172        [handempty, clear(X), on(Y,X)],
173        [holding(X), clear(Y)]).
174
175 action(put_on_table(X),
176        [holding(X)],
177        [holding(X)],
178        [handempty, clear(X), on(table,X)]).
179
180 action(put_on(Y,X),
181        [holding(X), clear(Y)],
182        [holding(X), clear(Y)],
183        [handempty, clear(X), on(Y,X)]).
184
185
186 % Hilfskonstrukt, weil das PROLOG "subset" nicht die Unifikation von
    Listenelementen
187 % durchführt, wenn Variablen enthalten sind. "member" unifiziert hingegen.
```

```
hingegen.
%
mysubset([],_).
mysubset([H|T],List):-
  member(H,List),
  mysubset(T,List).


expand_help(State,Name,NewState):-
  "Action suchen"
  "Conditions testen"
  "Del-List umsetzen"
  "Add-List umsetzen".


expand((_,State,_),Result):-

findall((Name,NewState,_),expand_help(State,Name,NewState),Result).
```

```
%
mysubset([],_).
mysubset([H|T],List):-
  member(H,List),
  mysubset(T,List).


expand_help(State,Name,NewState):-
  %used TODO:"Action suchen"
  action(Name, ConditionList, DeleteList, AddList),

  %used TODO:"Conditions testen" Folie: "Unter welchen Vorbedingungen ist die Aktion anwendbar?"
  mysubset(ConditionList,State),

  %used TODO:"Del-List umsetzen"  Folie: "Welche Formeln gelten nach Ausführung der Aktion nicht mehr?"
  subtract(State, DeleteList, TempNewState),
      /*writeln(['STATE: ',State]), %%DEBUG
      writeln(['DEL: ', DeleteList]),
      writeln(['AFTER: ',TempNewState]),nl,*/

  %used TODO:"Add-List umsetzen". Folie: "Welche Formeln kommen nach Ausführung der Aktion hinzu?"
  append(TempNewState,AddList,NewState).
      /*writeln(['ADD: ', AddList]),  %%DEBUG
      writeln(['AFTER: ',NewState]),nl.*/


expand((_,State,_),Result):-
  findall((Name,NewState,_),expand_help(State,Name,NewState),Result).
```