```
/*========================= INIT EVOLUTION ====================evolution*/


initevolution:-

   initothers,                          /* Parameter und Befehlsliste,      */
   initcritic,                          /* Parameter Bew.Fkt.,              */
   initgenoperators,                    /* Param. fuer gen.Op. besetzen,    */
   initpool,                            /* Pool erzeugen,                   */
   print('evolution initialised'),      /* Meldung.                         */
   nl.


/* ----------------------- INIT CRITIC -------------------initevolution*/

/* Vorbesetzen der Gewichtungen in der Bewertungsfunktion                   */
/* (Parameter zur Erzeugung des Selektionsdrucks).                          */

initcritic:-

   debug(wtime,1) ,                 /* Ausfuehrungszeit                    */
   debug(wlength,1) ,               /* Plarlaenge                          */
   debug(wfields,10) ,              /* betretene Felder (problemspez.)     */
   debug(werrornr,1) ,              /* fehlerhafte Schritte                */
   debug(wcorrectnr,0) .            /* korrekte Schritte                   */


/*--------------------------- INIT POOL -----------------initevolution*/

initpool:-

   retractall(plan(_,_,_)),             /* alle vorh. Plaene loeschen,      */
   debug(poolsize,10),                  /* Individuenanzahl festsetzen,     */
   poolsize(P),                         /* Anzahl in P holen,               */
   randompool1(P),                      /* erzeugen der zufaelligen Pop.;   */
   print('random pool generated'),      /* Meldung,                         */
   nl,
   criticall.                           /* alle Plaene bewerten.            */


randompool1(0).                 /* erzeugt die zufaellige                   */
randompool1(N):-                /* Anfangspopulation:                       */
   planlength(L),               /* Parameter Planlaenge holen,              */
   randomplan(L,Rplan),         /* einen Plan erzeugen,                     */
   normalize(Rplan,Newplan),    /* Marken normalisieren,                    */
   asserta(plan(N,0,Newplan)), /* Plan als Arg. von plan speichern.         */
   N1 is N-1,
   randompool1(N1).


randomplan(0,[]):- !.           /* erzeugt zufaelligen Plan,                */
randomplan(N,[Head|Tail]):-     /* N zaehlt abwaerts.                       */
   randominstruction(Head),     /* zufaelliges Primitives mit Parametern    */
   N1 is N-1,                   /* erzeugen, bis Planlaenge erreicht.       */
   randomplan(N1,Tail).
```

```
randominstruction(Head):-
   numberofprimitives(Nr),                  /* Aus der Liste aller        */
   random(Nr,X) ,                           /* vorhandenen Primitiven     */
   primitivelist(List),                     /* zufaellig eines            */
   associate(List,X,Primitive) ,            /* auswaehlen,                */
   getparameters(Primitive,Parlist) ,       /* Parameter dazu holen,      */
   append([Primitive|Parlist],Newlist) ,    /* Befehl "primitive(Parlist)"*/
   Head =.. Newlist.                        /* bilden.                    */

getparameters(Primitive,Parlist):-          /* Holt Typliste der Parameter*/
   parametertypelist(List),                 /* des Primitiven, dann zu    */
   associate(List,Primitive,Typelist),      /* jedem Typ ein evtl. weiter */
   maplist(randomparameter,Typelist,Parlist). /* parametrisiertes Atom    */


randomparameter(condition,Par):-            /* Besetzung der Parameter der*/
   conditionlist(List),                     /* Primitiven, problem-       */
   numberofconditions(Nr),                  /* spezifisch!                */
   random(Nr,X),
   associate(List,X,Cond),
   getparameters(Cond,Parlist),             /* (rekursiver Aufruf von     */
   append([Cond|Parlist],Newlist),          /* getparameters.)            */
   Par =.. Newlist.

randomparameter(object,Par):-
   random(0,N),                             /* Vorsicht bei Aenderungen   */
   tabobject(Olist),                        /* der Laenge der Standard-   */
   associate(Olist,N,Par).                  /* liste fuer Objekte.        */

randomparameter(marke,marke(N)):-
   maxnumberofmarks(Max),
   random(Max,N).

randomparameter(direction,Par):-
   random(0,N),                             /* Vorsicht bei Aenderungen.  */
   tabdirection(Dlist),
   associate(Dlist,N,Par).

randomparameter(marknr,Marknr):-
   maxnumberofmarks(Max),
   random(Max,Marknr).


/*---------------------------- normalize ------------randompool1,gen.Op.*/

normalize(Plan,Newplan):-                   /* "ordnet" Marken:           */
                                            /* Mehrfachvorkommen werden   */
   eliminatemarks(Plan,New),                /* eliminiert, die verbleibenden */
   debug(markcount,1),                      /* programmstrukturerhaltend  */
   sublist(is_a_mark,New,Marklist),         /* durchnumeriert.            */
   sort(Marklist,Mlist),                    /* ( <=  built in predicate)  */
   renamemarks(Mlist,New,Newplan).


is_a_mark(marke(X),marke(X)).

eliminatemarks([],[]).                      /* laesst nur das jeweils 1. Vor- */
eliminatemarks([marke(X)|Rest],[marke(X)|New]):-        /* kommen  */
   deleteall(marke(X),Rest,New1),           /* einer Marke urveraendert, weitere*/
   eliminatemarks(New1,New).                /* Vorkommen werden geloescht */
eliminatemarks([ElseRest],[ElseNew]):-      /* (vermeidet Mehrdeutig-*/
```

```
   eliminatemarks(Rest,New).        /* keiten).                          */

renamemarks(¢[],Plan,Plan).                  /* die tatsaechlich vorhan-  */
renamemarks(¢marke(X)|Rest],Plan,Newplan):- /* denen (und nicht nur als */
   markcount(Y),                             /* Sprungziel angegebenen)   */
   renameall(marke(X),Plan,marke(Y),New),    /* Marken in der Liste (1.   */
   increment(markcount),                     /* Arg.) werden umbenannt, sie*/
   renamemarks(Rest,New,Newplan).            /* erhalten Zahlenwerte 1 bis */
                                             /* Listenlaenge.             */
renameall(_,¢[],_,¢[]).
renameall(marke(X),¢marke(X)|R],marke(Y),¢marke(Y)|T]):-
   renameall(marke(X),R,marke(Y),T).                 /* ersetzt alle     */
renameall(marke(X),¢jump(Condition,marke(X))|R],    /* Vorkommen von     */
          marke(Y),¢jump(Condition,marke(Y))|T]):- /* marke(X) im Plan  */
   renameall(marke(X),R,marke(Y),T).                 /* durch marke(Y),   */
renameall(marke(X),¢Else|R],marke(Y),¢Else|T]):-    /*auch in jump-     */
   renameall(marke(X),R,marke(Y),T).                 /* Befehlen.        */


/*-------------------------criticall-----------------------------initpool*/

criticall:-                          /* Bewertung aller Plaene: Nummern   */
                                     /* des besten und schlechtesten Plans */
   poolsize(P),                      /* (aus dem letzten Lauf) und alte   */
   retractall(worstplan(_)),         /* Bewertungssumme entfernen,        */
   retractall(bestplan(_)) ,
   retractall(measuresum(_)),        /* Bewertungsschleife aufrufen,      */
   criticloop(P),
   findworst(P,P,10000),             /* schlechtesten und                 */
   findbest(P,P,-1000),              /* besten Plan bestimmen und         */
   print('all plans criticised'),    /* Meldung ausgeben.                 */
   nl , nl .

criticloop(P):-                      /* Schleifenkonstruktion             */
   debug(criticcount,P) ,
   repeat ,
   retract(plan(N,_,Planlist)) ,
   criticloop1(N,Planlist) .

criticloop1(N,Planlist):-            /* Plan Nummer N bewerten,           */

   ( retract(measuresum(M));
     M = 0 ),                        /* executeandcritic steht in         */
   executeandcritic(Planlist,Planmeasure),                /* EVOLUTION.*/
   ( Planmeasure > 5,                /* Hier wird die wegen der Plan-     */
     Newsum is M + Planmeasure;      /* auswahl modifizierte Bewertungs-  */
     Newsum is M + 5 ),              /* summe aktualisiert.               */
   asserta(measuresum(Newsum)),
   asserta(plan(N,Planmeasure,Planlist)),  /* Plan mit Bewertung        */
   decrement(criticcount), !,              /* speichern.                */
   criticcount(C).


findworst(0,Minind,_):-              /* Sucht den schlechtesten Plan der  */
   asserta(worstplan(Minind)).       /* Population und speichert dessen   */
findworst(N,Minind,Min):-            /* Nummer unter worstplan.           */
   plan(N,Value,_),
   N1 is N-1,
   ( ( Value < Min,
       findworst(N1,N,Value) );
     findworst(N1,Minind,Min) ).
```

```
findbest(0,Maxind,_):-              /* analog zu findworst fuer den       */
   asserta(bestplan(Maxind)).       /* besten Plan.                       */
findbest(N,Maxind,Max):-
   plan(N,Value,_),
   N1 is N-1,
   ( ( Value > Max,
       findbest(N1,N,Value) );
     findbest(N1,Maxind,Max) ).



/*------------------------ INIT GENOPERATORS ---------------initevolution*/

initgenoperators:-                  /* Verteilung der Wahrscheinlichkeit auf die */
                                    /* genetischen Operatoren.                   */
   debug(probmut,5),                /* ACHTUNG: Die Zahlen entsprechen nicht der */
   debug(probparamut,10),           /* prozentualen Wahrscheinlichkeit: man muss */
   debug(probdel,15),               /* dazu die Differenz mit der vorherigen     */
   debug(probcro,55),               /* (Zeile darueber) Zahl bilden.             */
   debug(probins,60) ,
   debug(probinv,85),               /* Wird statt 100 eine groessere Zahl einge- */
   debug(probjumpins,100).          /* tragen (fuer "feinere" Auswahl), so muss  */
                                    /* in genoperators die Zufallszahl entsprech-*/
                                    /* end bestimmt werden!                      */



/*-------------------------- INIT OTHERS -----------------initevolution*/

initothers:-                                /* Vorbesetzen von verschiedenen */
   debug(adapttime,2),                      /* Parametern des G.A..          */
   debug(numberoftests,2),                  /* Arg. von adapttime >= 1!      */
   debug(maxnumberofmarks,5),               /* die Laenge der Primitiven- und*/
   debug(numberofconditions,4),             /* der Bedingungenliste werden   */
   debug(numberofprimitives,8),             /* hier gespeichert, da sie oft  */
   debug(planlength,10),                    /* geaendert werden.             */
   debug(history,off),
   retractall(conditionlist(_)),            /* Problemspezifische Teile:     */
   retractall(primitivelist(_)),            /* evtl. geaenderte Listen der   */
   retractall(tabdirection(_)),             /* Sprachelemente entfernen und  */
   retractall(tabobject(_)),                /* Standardlisten geschrieben.   */
   retractall(parametertypelist(_)),        /* Verteilung der Auswahlwahr-   */
   asserta(primitivelist(¢¢1 ! step!,       /* scheinlichkeiten fuer die     */
                         ¢2 ! step!,        /* Primitiven in                 */
                         ¢3 ! step!,        /* den Plaenen der Anfangspop.   */
                         ¢4 ! step!,        /* durch unterschiedlich         */
                         ¢5 ! jump!,        /* haefiges Auftreten in dieser  */
                         ¢6 ! marke!,       /* Liste.                        */
                         ¢7 ! jump!,
                         ¢8 ! marke!!)) ,
   asserta(conditionlist(¢¢1 ! true!,
                         ¢2 ! true!,
                         ¢3 ! true!,
                         ¢4 ! recognize!!)) ,
   asserta(tabdirection(¢¢1 ! left!,
                        ¢2 ! right!,
                        ¢3 ! forward!,
                        ¢4 ! backward!,
                        ¢5 ! north!,
                        ¢6 ! south!,
                        ¢7 ! east!,
                        ¢8 ! west!!)) ,
```

```
asserta(tabobject(¢¢1 | wall:,
                  ¢2 | wall:,
                  ¢3 | wall:,
                  ¢4 | wall:,
                  ¢5 | ¢:,
                  ¢6 | ¢:,
                  ¢7 | ¢:,
                  ¢8 | _ ::)),
asserta(parametertypelist(¢¢step      | ¢direction::,
                          ¢recognize  | ¢direction,object ::,
                          ¢jump       | ¢condition,marke::,
                          ¢true       | ¢::,
                          ¢marke      | ¢marknr:::)).
```

```
/*======================  E V O L U T I O N  ============================*/

/* Prozeduren zum Starten bzw. Fortsetzen des genetischen Algorithmus.  */


evolution:-                         /* "Normalstart" des g. A. :          */
                                    /* In initevolution ( >>> Datei ) wird*/
   initevolution ,                  /* (u.a.) die Anzahl T der Generationen*/
   adapttime(T) ,                   /* als Argument von adapttime vorbesetzt.*/
   evoloop(T) ,                     /* T Generationen werden durchlaufen und*/
   poolreport .                     /* der entstandene Pool ausgegeben.   */


run(S,Time):-                       /* g. A. lauft mit Anfangszahl S fuer */
                                    /* Pseudozufallszahlengenerator ( andere*/
   debug(seed,S) ,                  /* Zahlenfolge als bei evolution ).   */
   initevolution ,                  /* Anzahl der Generationen im zweiten */
   evoloop(Time) ,                  /* Argument "Time".                   */
   poolreport .                     /* Dann Ausgabe des Pools.            */


contevolution(N):-                  /* g. A. wird auf vorhandenem Pool fort-*/
                                    /* gesetzt, Argument  N = Anzahl weitere*/
   evoloop(N) ,                     /* Generationen,                      */
   poolreport.                      /* danach Ausgabe.                    */


/*-------------------------- EVOLOOP ------------------------evolution*/

/* Kern des g. A. ; in dieser Schleife wird je Durchlauf ( Generation ) */
/* ein neuer Plan erzeugt und bewertet. Danach muss er sich im Pool     */
/* durchsetzen oder er wird verworfen.                                  */


evoloop(N):-                        /* Die Schleifenkonstruktion mit      */
                                    /* mehrfachem Aufruf von evoloop1     */
   debug(evocount,N) ,              /* durch backtracking statt           */
   repeat ,                         /* Rekursion vermeidet                */
   evoloop1 .                       /* Stackueberlauf.                    */

evoloop1:-                          /* Ist das Argument von history       */
                                    /* ungleich off, so wird bei          */
   evocount(N) ,                    /* jedem Schleifendurchlauf eine      */
   ( history(off) ;                 /* Meldung ausgegeben.                */
     printheaderevoloop(N) ) ,      /* ( "sequentielles oder"! )          */
   whirlpool(Born_plan) ,               /* NEUEN PLAN ERSTELLEN,         */
   executeandcritic(Born_plan,Measure) ,  /* DIESEN PLAN BEWERTEN.       */
   survival_of_the_fittest(Born_plan,Measure,N) , /* LEBENSKAMPF.        */
   decrement(evocount) , ! ,        /* cut bewirkt bei N ungleich 1       */
   N = 1 .                          /* backtrack zu obigem repeat.        */


printheaderevoloop(N):-             /* gibt eine Meldung in evoloop1      */
                                    /* aus: Nummer des evoloop-           */
   print('evoloop Nr. ') ,          /* Durchlaufs ( absteigend von        */
   print(N) ,                       /* N bis 1 ) und die Bewertung        */
   print('   current best Value : ') ,  /* des im Moment besten Plans.     */
   bestplan(Nr) ,                   /* (in survival_of_the_fittest        */
   plan(Nr,Value,_) ,               /* bei neuem bestplan aufge-          */
   print(Value) , nl .              /* rufen).                            */
```

```
/*============================ WHIRL PCCL ========================evoloop*/

/* In whirlpool wird ein Plan aus dem Pool ausgewaehlt und auf diesen   */
/* dann ein genetischer Operator angewandt. Dabei ist die Auswahl-      */
/* wahrscheinlichkeit fuer einen Plan proportional zu seiner Bewertung. */


whirlpool(Newplanlist):-            /* pickplan benoetigt eine Zufallszahl */
                                    /* ( Range ) mit:                      */
   measuresum(Sum) ,                /* 0 < Range <= Summe der Bewertungen  */
   random(Sum,Range) ,              /* aller Plaene, ausserdem die Groesse */
   poolsize(P) ,                    /* des Pools.                          */
   pickplan(P,0,Range,Picked) ,     /* Planauswahl und Merken der Nummer   */
   debug(actualplan,Picked) ,       /* als Argument von actualplan.        */
   plan(Picked,_,Oldplanlist) ,     /* Planliste holen und g. Op. anwenden,*/
   genoperators(Oldplanlist,Newplanlist) ,   /* >>> Datei genoperators. */
   ( ( history(full) ,                       /* Ausgbe der Meldung      */
       tab(5) ,                              /* "whirled" nur bei       */
       print('whirled') ) ;                  /* history(full).          */
     true ) .


pickplan(1,_,_,1).                  /* pickplan addiert schrittweise       */
                                    /* die Bewertungen der Plaene.         */
pickplan(P,Sofar,Range,Next):-      /* Dann wird der Plan, bei dem         */
                                    /* Range ueberschritten wird,          */
   plan(P,Measure,_),               /* "genetisch" veraendert.             */
   ( Measure > 0,                   /* Plaene mit Bewertung <5 werden      */
     Newsofar is Sofar + Measure;   /* hier mit 5 gefuehrt, damit auch     */
     Newsofar is Sofar+ 5 ),        /* sie noch ausgewaehlt werden;        */
   ( ( Range =< Newsofar,           /* ausserdem gaebe es mit neg. Be-     */
       Next = P );                  /* wertungen durcheinander.            */
     ( P1 is P-1 ,
       pickplan(P1,Newsofar,Range,Next) ) ) .


/*------------------------- EXECUTE AND CRITIC ----------------evoloop*/

/* Der Plan ( die Planliste wird im ersten Argument uebergeben ) wird   */
/* ein paar Mal ausgefuehrt, aus den dabei erhaltener Ergbnissen eine   */
/* Bewertung des Plans gebildet ( Measure ).                            */

executeandcritic(Plan,Measure):-

   numberoftests(Nroftests),        /* Anzahl Testlaeufe holen,            */
   debug(numoftests,Nroftests),     /* unter numoftests fuer exandcrit     */
   debug(allmeasure,0),             /* ablegen. Bewertung auf 0 setzen     */
   repeat,                          /* und die Schleife fuer die           */
   exandcrit(Plan),                 /* Testlaeufe durchfuehren.            */
   allmeasure(Measure).             /* Bewertung in Measure uebergeben.    */


exandcrit(Plan):-

   numoftests(Nroftests),
   guidedexec(Plan),                /* Testlauf durchfuehren               */
   critic(Plan,M),                  /* und Ergebnisse dieses Laufs be-     */
   retract(allmeasure(Old)),        /* werten. Die endgueltige Bewertung*/
   New is Old + M,                  /* des Plans ergibt sich aus der       */
   asserta(allmeasure(New)),        /* Summe dieser Einzelwertungen        */
   decrement(numoftests), ! ,       /* ( im Argument von allmeasure ).     */
   Nroftests = 1.
```

```
/*------------------------- GUIDED EXECUTION ----------------exandcrit*/

guidedexec(Born_plan):-                 /* Testlauf ( kontrollierte Aus-    */
                                        /* fuehrung eines Plans ) starten:  */
   initexec ,                           /* Vorbereitungen,                  */
   guidedrun(Born_plan) ,               /* Start,                           */
   ( ( history(full) ,                  /* Meldung bei history(full).       */
       tab(5) ,
       print('plan executed') ,
       nl ) ;
     true ) .


initexec:-                          /* Vorbereiten der Planausfuehrung:     */
                                    /* initrobby, initenvironment, fields¢; */
   initrobby ,                      /* sind problemspezifisch und muessen   */
   initenvironment,                 /* bei anderen Problemen natuerlich     */
   debug(fields,¢;) ,               /* durch entsprechende Prozeduren er-   */
   debug(maxansprung,100) ,         /* setzt werden.                        */
   debug(time,0) ,
   debug(errornumber,0) ,
   debug(correctnumber,0).


guidedrun(Planlist):-

   debug(maxinstrnumber,250),       /* Anzahl der Schritte festlegen    */
   write_plan_in_database(Planlist),/* Planliste in gdb schreiben,      */
   initializejmpnrs(Planlist),      /* Zaehltabelle fuer Anspruenge     */
   debug(instructionpointer, 1),    /* und Befehlszaehler vorbereiten.  */
   length(Planlist,Length),         /* ( built-in-predicate )           */
   repeat,                          /* Schleife: Ausfuehrung der        */
   ausfuehrung(Length).             /* Primitiven.                      */


write_plan_in_database(Planlist):-  /* Der Plan wird in folgender       */
                                    /* Art in die globale Datenbasis    */
   retractall(instruction(_,_)),    /* des Prologinterpreters ge-       */
   debug(instrcount,1),             /* schrieben:                       */
   wrplindb(Planlist),              /* instruction(1,Primitives,PPP)    */
   instrcount(Count),               /* instruction(2,Primitives,PPP)    */
   Catchall is Count+1,             /* ...                              */
   assertz(instruction(Catchall,_))./* instruction(N,Primitives,PPP)    */
                                    /* instruction(N+1,_)               */
                                    /* Dabei steht PPP fuer die         */
wrplindb(¢!).                       /* Parameter des Primitiven.        */
wrplindb(¢[Instr|Tail¦):-           /* Nachdem alte instructions        */
   instrcount(Count),               /* entfernt wurden, schreibt        */
   assertz(instruction(Count,Instr)),/* wrplindb den Plan.              */
   increment(instrcount),
   wrplindb(Tail).

initializejmpnrs(Planlist):-            /* Zum mitzaehlen, wie oft      */
   retractall(jmpnr(_,_)),              /* die Spruenge ausgefuehrt     */
   sublist(has_a_mark,Planlist,Marknrlist),/* werden, wird eine Liste  */
   map(inijnr,Marknrlist).              /* von Sprungzielen in der      */
                                        /* folgenden Form geschrieben:  */
has_a_mark(jump(_,marke(X)),marke(X)).  /* jmpnr(marke(NR1),Zaehler1)   */
                                        /* jmpnr(marke(NR2),Zaehler2)   */
inijnr(marke(Marknr)):-                 /* ...                          */
   asserta(jmpnr(marke(Marknr),0)).     /* (Zaehler auf 0 vorbesetzt.)  */
```

```
ausfuehrung(Length):-                  /* Der aktuelle Plan (in Datenbasis)  */
   Length = 0;                         /* wird ausgefuehrt. Fuer die Inter-  */
   maxinstrnumber(N),                  /* pretation der Primitiven siehe     */
   instructionpointer(Ip),             /* Datei OPERATIONS. Zuerst wird die  */
   instruction(Ip,Instr),              /* aktuelle Operation geholt und      */
   call(Instr),                        /* ausgefuehrt.                       */
   increment(time),                    /* time kann auch in OPERATIONS ent-  */
   increment(instructionpointer),      /* sprechend dem Rechenaufwand eines  */
   ( ( retract(errordetected(_)),      /* Primitiven neu berechnet werden.   */
       increment(errornumber) );       /* ip und Zaehler fuer korr./fehlerh. */
     increment(correctnumber) ),       /* Schritt erhoehen, Zaehler fuer     */
   decrement(maxinstrnumber),          /* Schritte insgesamt herunterzaehlen.*/
   !,                                  /* Terminierung bei Zeit-, max. Anz.  */
   ( ( time(T),                        /* Schritte-Ueberschreitung, Prg.Ende.*/
       T=250 );                        /* Achtung: unsaubere Terminierung    */
     N=1;                              /* von ausfuerung(length) bei         */
     ( instructionpointer(I),          /* maxansprung(marke) ueberschritten! */
       I > Length ) ).                 /* (siehe OPERATIONS, "jump")         */


/*------------------------- CRITIC ----------------------------exandcrit*/

critic(Born_plan,Measure):-            /* Bewertung eines Planes berechnen:  */
                                       /* Parameter (in ausfuerung/OPERATIONS */
   wlength(Wlen) ,                     /* bestimmte Leistungsdaten) und dazu- */
   length(Born_plan,L) ,               /* gehoerige Gewichtungen holen.      */
   wtime(Wtim) ,
   time(T) ,
   wfields(Wfie) ,
   fields(Flist) ,
   length(Flist,F) ,
   werrornr(Werr) ,
   errornumber(E) ,
   wcorrectnr(Wcrn) ,
   correctnumber(C) ,

   Testpos is Wfie*F + Wcrn*C ,        /* Bewertungsfunktion                 */
   Testneg is Wlen*L + Wtim*T + Werr*E ,  /* daraus berechnen.               */
   Measure is Testpos - Testneg .
```

```
/*====================== SURVIVAL OF THE FITTEST ================evoloop*/

/* Hier wird entschieden, ob sich ein neuer Plan im Pool durchsetzt und */
/* einen anderen verdraengt, oder ob er verworfen wird. In der jetzigen */
/* Version ueberlebt ein neuer Plan, wenn er besser ist als der Plan,   */
/* aus dem er entstanden ist, wobei dieser dann verdraengt wird.        */

survival_of_the_fittest(Born_plan,Measure,N):-

   ( bestplan(Bestnr) ;
     Bestnr = 1 ) ,
   ( actualplan(Actnr) ;
     Actnr = 1 ) ,
   plan(Actnr,Oldv,_) ,
   plan(Bestnr,Bestvalue,_) ,
   ( Oldv > Measure ;                          /* Neuer Plan zu schlecht; */
     retract(plan(Actnr,_,_)) ,                /* neuer Plan besser oder  */
     retractall(actualplan(_)) ,               /* gleichgut wie der alte, */
     asserta(plan(Actnr,Measure,Born_plan)),   /* dieser wird entfernt,   */
     measuresum(Sum),                          /* und der neue gespeichert*/
     ( Measure =< 5;                           /* Modifizierte Bewertungs-*/
       ( Oldv =< 5,                            /* summe erneuern (Plaene  */
         Newsum is Sum - 5 + Measure;          /* mit Bewertung < 5 werden*/
         Newsum is Sum - Oldv + Measure ),     /* mit 5 beruecksichtigt). */
       debug(measuresum,Newsum) ),
     ( Bestvalue >= Measure ;
       retractall(bestplan(_)) ,               /* Neuer Plan sogar besser */
       asserta(bestplan(Actnr)) ,              /* als bisher bester: best-*/
       printheaderevoloop(N) ) ) ,             /* plan erneuern und       */
   ( ( history(full) ,                         /* Meldung ausgeben.       */
       tab(5) ,
       print('survival_of_the_fittest finished') ,
       nl ) ;
     true ) .


/*--------------------------- POOLREPORT -------------(cont-)evolution*/

/* Ausgabe des Pools: Plan-Nummer, Bewertung, Plan.                      */

poolreport:-
   poolsize(P) ,                    /* Poolgroesse holen                */
   printpool(P) .                   /* und Ausgabeschleife rufen.       */

printpool(0).
printpool(N):-
   plan(N,Value,List) ,
   print('plannumber ') ,
   print(N) ,
   print(',    effectiveness ') ,
   print(Value) , nl ,
   pprint(List) , nl , nl ,
   N1 is N-1 ,
   printpool(N1) .

pprint([]).                         /* druckt Planliste in lesbarer    */
pprint([Head|Tail]):-               /* Form aus.                       */
   ( ( Head = marke(_),
       print(Head) );
     ( tab(3),
       print(Head) ) ), nl,
   pprint(Tail).
```

```
/*========================== GENETIC OPERATORS ==========================*/


genoperators(Oldplanlist,Newplanlist):-  /* Auswahl eines gen. Op.      */
   probmut(Probmut),                     /* aufgrund ihrer Anwendungs-  */
   probparamut(Probparamut),             /* wahrscheinlichkeiten        */
   probdel(Probdel),
   probcro(Probcro),
   probins(Probins),
   probjumpins(Probjumpins),
   probinv(Probinv),
   random(10,Q),
   P is Q - 1,                                    /* P ist nun Zufallszahl*/
   ( ( P<Probmut,                                 /* zwischen 0 und 99    */
       mutation(Oldplanlist,Newplanlist) );       /* Vergleiskaskade,     */
     ( P<Probparamut,                             /* die feststellt       */
       paramutation(Oldplanlist,Newplanlist) );   /* wecher Operator von  */
     ( P<Probdel,                                 /* P ausgewaehlt wurde  */
       delete(Oldplanlist,Newplanlist) );
     ( P<Probcro,                                 /* Dann Aufruf des Op.  */
       crossover(Oldplanlist,Newplanlist) );
     ( P<Probins,
       insert(Oldplanlist,Newplanlist) );
     ( P<Probinv,
       invert(Oldplanlist,Newplanlist) );
     ( P<Probjumpins, jumpinsert(Oldplanlist,Newplanlist) );
     Newplanlist=Oldplanlist).


/*--------------------- PARAMETER MUTATION ----------------------------*/


paramutation(Oldplanlist,Newplanlist):-

   length(Oldplanlist,O),                 /* Laenge berechnen          */
   random(0,Index),                       /* index zufaellig: 0<index<O */
   findinstr(Index,Instr,Oldplanlist),    /* Instr. an Indexstelle suchen */
   Instr =.. [Head|_],
   getparameters(Head,Parlist),           /* mit neuen Parametern belegen */
   append([Head],Parlist,Newlist),
   Newinstr =.. Newlist,
   substindex(Index,Newinstr,Oldplanlist,Newplanlist1),
   normalize(Newplanlist1,Newplanlist). /* Liste mit den neuen Parmetern*/
                                        /* normalisiern               */

findinstr(1,Instr,[Instr|_]).          /* Instruktion an Indexstelle   */
findinstr(N,Instr,[_|Rest]):-          /* suchen und abliefern         */
   N1 is N - 1,findinstr(N1,Instr,Rest).


/*---------------------------- MUTATION -------------------------------*/

mutation(Oldplanlist,Newplanlist):-

   length(Oldplanlist,O),                 /* Laenge berechnen          */
   random(0,Index),                       /* zufaelligen Index festsetzen */
   randominstruction(Instr),              /* zuf. neue Instruktion holen  */
   substindex(Index,Instr,Oldplanlist,Newplanlist1), /* neue Instr.einf.*/
   normalize(Newplanlist1,Newplanlist). /* mit neuer Instr. norm.      */

/*---------------------------- CROSSOVER ------------------------------*/
```

```
crossover(Old,New):-

   poolsize(P),                          /* Auswahl eines Plans aus     */
   random(P,Nr),                         /* dem Pool                    */
   plan(Nr,_,List),
   retractall(actualplan(_)),
   asserta(actualplan(Nr)),
   cross(Old,List,New).

cross(List1,List2,Newlist):-             /* Neu wird zu einem Plan der  */
   m_split(List1,L1,L2),                 /* einen Teil von Old enthaelt */
   m_split(L2,Insertpart,_),
   distinguish(Insertpart,Insertion),    /* distinguish wichtig damit   */
   m_split(List2,K1,K2),                 /* keine Ueberschneidungen bei */
   m_split(K2,_,K4),                     /* der Markennumerierungent-   */
   append(K1,Insertion,New),             /* stehen (100 addieren)       */
   append(New,K4,Newlis),
   normalize(Newlis,Newlist).




/*-------------------------------. DELETE -------------------------------*/


delete(Old,New):-
   length(Old,L),                          /* zufaellig eine zu        */
   random(L,Index),                        /* loeschende Operation     */
   delete1(Index,Old,New).                 /* waehlen                  */

delete1(1,[_ | Oldrest],Oldrest).          /* loeschen der ausgew.   */
delete1(N,[A | Oldrest],[A | Newrest]):-   /* Operation aus dem Plan */
   N1 is N-1 ,
   delete1(N1,Oldrest,Newrest). /* hier kein Normalisieren erforderlich */


/*------------------------------- INSERT -------------------------------*/


insert(Old,New):-

   length(Old,L),                          /* zufaellig Einfuegeposition  */
   random(L,N),                            /* waehlen                     */
   insert1(N,Old,New1),
   normalize(New1,New).

insert1(1,List,[Newinstr | List]):-              /* einfuegen einer    */
   randominstruction(Newinstr).                  /* "randominstruction"*/
insert1(N,[First | Rest],[First | Newrest]):-    /* an der Selle die   */
   N1 is N-1,                                    /*in N uebergeben wird*/
   insert1(N1,Rest,Newrest).


/*---------------------------- INSERTJUMP ----------------------------*/


jumpinsert(Old,New):-

   length(Old,L),                          /* zwei zufaellige Positionen*/
```

```
   random(L,Place),                          /* eine fuer den Sprung mit   */
   K is L+1,                                 /* zufaelliger Bedingung      */
   random(K,Place2),                         /* die andere fuer die Marke  */
   randomparameter(condition,Par),
   ins(Old,Place,jump(Par,marke(111)),List),  /* Einfuegen des Sprungs  */
   ins(List,Place2,marke(111),Newlist),       /* Einfuegen der Marke    */
   normalize(Newlist,New).

ins(List,1,Thing,[Thing | List]).          /* Einfuegeprozedur       */
ins([First | Rest],N,Thing,[First | Newrest]):-
   N1 is N-1,
   ins(Rest,N1,Thing,Newrest).



/*---------------------------------- INVERT -----------------------------*/


invert(Old,New):-

   length(Old,L),                         /* Anfangs und Endpunkt des zu inv. */
   random(L,B1),                          /* Teilbereichs festlegen           */
   random(L,B2),
   ( ( B1 =< B2,
       Break1 = B1,
       Break2 = B2 );                     /* evtl vertauchen, wenn B1>B2      */
     ( Break1 = B2,
       Break2 = B1 ) ),
   invert1(Break1,Break2,Old,New).

invert1(1,High,Old,New):-                 /* wenn bei B1 angelangt, dann */
   reverse_head(High,Old,New).            /* Bereich umdrehen            */
invert1(Low,High,[A | Oldrest],[A | Newrest]):-
   Low1 is Low - 1,                       /* Ablaufen bis zu B1          */
   High1 is High - 1,
   invert1(Low1,High1,Oldrest,Newrest).




/*-------------------------------------------------------------------------*/
/*---------------------------- AUXILIARIES --------------------------------*/
/*-------------------------------------------------------------------------*/


substindex(1,A,[_ | Rest],[A | Rest]).    /* ersetzt N-tes Element */
substindex(N,A,[X | Rest],[X | Sub]):-    /* des Oldplan durch A   */
   N1 is N-1,
   substindex(N1,A,Rest,Sub).
substindex(_,_,[],[]).



reverse_head(1,List,List).
reverse_head(Index,Old,New):-                 /* Listenanfang bis Index  */
   separate_head(Index,Old,Oldhead,Oldtail), /* abspalten und           */
   reverse(Oldhead,Newhead),                 /* verkehrt herum          */
   append(Newhead,Oldtail,New).              /* wieder anhaengen        */
```

```
separate_head(1,$A | Rest!,$A!,Rest).
separate_head(N,$A | Rest!,$A | Resthead!,Resttail):-
   N1 is N - 1,
   separate_head(N1,Rest,Resthead,Resttail).


/*-------------------------------------------------------------------*/

distinguish(Plan,Newplan):-          /* Hilfsprozedur um Markenueber-  */
   maplist(add100,Plan,Newplan).     /* schneidungen bei Crossover zu  */
                                     /* vermeiden. Es wird 100 zu jeder*/
add100(marke(X),marke(Y)):-          /* Markennummer des Plans addiert */
   Y is X+100.
add100(jump(Condition,marke(X)),jump(Condition,marke(Y))):-
   Y is X+100.
add100(A,A).

m_split(List,L1,L2):-                          /* teilt List an       */
   length(List,Length) ,                       /* zufaellig festgelegter */
   ( ( Length = 0,                             /* Stelle              */
       L1 = $!,
       L2 = $! );
     ( random(Length,R),
       m_spl(R,List,L1,L2) ) ).

m_spl(_,$!,$!,!).
m_spl(0,List,$!,List).
m_spl(N,$A|Rest!,$A|R1!,R2):-      /* Durchlaufen der Liste bis      */
   N1 is N-1,                          /* N dann Abbruch der Rekursion */
   m_spl(N1,Rest,R1,R2).

/*----------------------- PARAMETER REPORT ---------------------------*/
/* Berechnung der Auswahlwahrscheinlichkeiten der Gen.Ops. und       */
/* Ausgabe aller wichtiger Parameter des genetischen Algorithmus     */

paramreport:-
   print('--------------- POOL PARAMS -----------------') , nl,
   poolsize(Ps) , planlength(Pl) ,
   print('poolsize : '),print(Ps),nl,
   print('planlength : '),print(Pl),
   nl,nl,
   print('------------ EVOLOOP PARAMS -----------------') , nl,
   adapttime(A),print('adapttime :  '),print(A), nl,
   maxinstrnumber(T),print('maxinstrnumber :  '),print(T),nl,
   numberoftests(Noft),print('numberoftests :  '),print(Noft),
   nl,nl,
   print('----------- Prob. of genetic Operators --------------'),
   nl,
   probmut(Mut),probparamut(Paramut),probdel(Del),probcro(Cro),
   probins(Ins),probinv(Inv),probjumpins(Jumpins),
   Jumpinss is Jumpins - Inv, Invv is Inv - Ins,
   Inss is Ins - Cro, Croo is Cro - Del,
   Dell is Del - Paramut, Paramutt is Paramut - Mut,
   print('p(mutation)% '),print(Mut),nl,
   print('p(paramutation)% '),print(Paramutt),nl,
   print('p(delete)% '),print(Dell),nl,
   print('p(crossover)% '),print(Croo),nl,
   print('p(insert)% '),print(Inss),nl,
   print('p(invert)% '),print(Invv),nl,
   print('p(jumpinsert)% '),print(Jumpinss),nl,
   nl,
   print('------------ CRITIC PARAMS -----------------') , nl,
```

```
    wlength(Wlen),wtime(Wtim),wfields(Wfie),
    werrornr(Werr),wcorrectnr(Wcrn),
    print('wtime : '),print(Wtim),nl,
    print('wlength : '),print(Wlen),nl,
    print('wfields : '),print(Wfie),nl,
    print('werrornr : '),print(Werr),nl,
    print('correcttnr : '),print(Wcrn),    nl .
```

```
/* problemspezifischer Teil, der die Interpretation oer                    */
/* Repraesentationssprache ausfuehrt                                       */

jump(Condition,marke(X)):-      /* Interpretation des Primitiven jump,     */
   call(Condition) , ! ,        /* nur dann, wenn Bedingung erfuellt.      */
   jmpnr(marke(X),Nr) ,
   maxansprung(Max) , ! ,
   ( ( Nr < Max ,
       instruction(Ip,marke(X)) ,     /* Hole Instructionpointer          */
       debug(instructionpointer,Ip), /* und umsetzen                      */
       Nr1 is Nr+1 ,
       retract(jmpnr(marke(X),_)) ,
       asserta(jmpnr(marke(X),Nr1)) );    /* 2. "oder"- Zweig fuer        */
     debug(time,249) ).                   /* Terminierung von Ausfuehrung*/

jump(_,marke(X)) .          /* falls Bedingung fuer Sprung nicht erfuellt*/
                            /* oder falls maxansprung ueberschritten,    */
                            /* Sprung nicht ausfuehren.                  */

marke(_) .                  /* Marken werden ohne Aktion ueberschritten  */

step(Dir):-
   getnext(Dir,Xnew,Ynew) ,                    /* neue Position holen   */
   robbysroom(Number) ,
   room(Number,¢Xmax,Ymax¦,List) ,             /* Raumgroesse holen     */
   ( ( not(inrectangle(Xnew,Ynew,Xmax,Ymax)) , /* Schritt nicht moeglich*/
       asserta(errordetected(step)) ) ;        /* Fehler melden         */
     ( associate(List,¢Xnew,Ynew¦,Result) ,
       ( ( not(Result = ¢¦) ,
           asserta(errordetected(step)) ) ;
         ( retract(robbyslocallocation(_,_)) ,       /* neue Position   */
           asserta(robbyslocallocation(Xnew,Ynew)) , /* speichern       */
           ( retract(fields(F)) ;                    /* Liste betretener*/
             F = ¢¦ ) ,                              /* Felder erneuern.*/
           ( ( member(¢Xnew,Ynew¦,F) ,                asserta(fields(F)) ) ;
             asserta(fields(¢¢Xnew,Ynew¦ ¦ F¦)) ) ,
           ( Dir = forward ;
             Dir = backward ;
             ( retract(robbysdirection(Curdir)) , /* Blickrichtung      */
               transform(Curdir,Dir,Newdir) ,     /* anpassen,wenn nicht*/
               asserta(robbysdirection(Newdir))   /* vor oder zurueck   */
             )
           )
         )
       )
     )
   ).


recognize(Reldir,Result):-                  /* liefert in Result wall, */
   getnext(Reldir,Xnew,Ynew) ,              /* falls das Feld in der   */
   robbysroom(Number) ,                     /* angeg. Richtung ausser- */
   room(Number,¢Xmax,Ymax¦,List) ,          /* halb der Raumgrenzen    */
   ( ( not(inrectangle(Xnew,Ynew,Xmax,Ymax)) ,          /* liegt.     */
       ! , Result = wall ) ;
     associate(List,¢Xnew,Ynew¦,Result) ) .

recognizeandjump(Reldir,Result,marke(X)):-  /* Wenn recognize das gefor-*/
   ( recognize(Reldir,Result) ,             /* derte Ergebnis liefert, */
     jump(marke(X)) ) ;                     /* dann Sprung ausfuehren. */
   true.
```

```
getnext(Reldir,Xnew,Ynew):-                 /* liefert das in der bzgl. dem*/
   robbyssituation(_,X , Y,Absdir) ,        /* Roboter angegebenen Richtung*/
   transform(Absdir,Reldir,Dir) ,           /* naechste Feld (Xnew,Ynew).  */
   Goal =.. [Dir,First,Second,_,_] ,
   call( Goal ) ,
   Xnew is X + First ,
   Ynew is Y + Second .


transform(Dir,forward,Dir).       /* Bei forward keine Richtungaenderung */

transform(Curdir,left,Dir):-              /* Drehung nach links            */
   Goal =.. [Curdir,_,_,Dir,_] ,
   call(Goal) .                           /* neu Blickrichtung berechnen   */

transform(Curdir,right,Dir):-             /* Drehung nach rechts           */
   Goal =.. [Curdir,_,_,_,Dir] ,
   call(Goal) .                           /* neu Blickrichtung berechnen   */

transform(Curdir,backward,Dir):-          /* backward als zwei mal rechts  */
   Goal1 =.. [Curdir,_,_,_,Middir] ,      /* realisiert                    */
   call(Goal1) ,
   Goal2 =.. [Middir,_,_,_,Dir] ,
   call(Goal2) .

transform(_,Absdir,Absdir).         /* Absolute Richtung wird immer zur    */
                                    /* Blickrichtung                       */

north(0,1,west,east).               /* Festlegungen der Folgerichtungen    */
south(0,-1,east,west).
east(1,0,north,south).
west(-1,0,south,north).


inrectangle(X,Y,Xmax,Ymax):-        /* Pruefung ob (X,Y) ausserhalb des    */
   Xmax >= X ,                      /* Bereichs (Xmax,Ymax)                */
   X >= 1 ,
   Ymax >= Y ,
   Y >= 1 .


/*---------------------------- ROBOT ---------------------------------*/

initrobby:-                                 /* Alte Roboterdaten werden */
                                            /* geloescht falls vorhanden*/
   ( retract(robbysroom(_)) ; true ) ,      /* Initialisieren auf       */
   asserta(robbysroom(room1)) ,                   /* Raum 1            */
   ( retract(robbysdirection(_)) ; true ) ,       /* Rel. Richtung nord */
   asserta(robbysdirection(north)) ,              /* Position (1,1)     */
   ( retract(robbyslocallocation(_,_)) ; true ) ,
   asserta(robbyslocallocation(1,1)) .


robbyssituation(Room,Posx,Posy,Absdir):-    /* Sammelt Daten ueber      */
                                            /* Raum Position und abs.   */
   robbyslocallocation(Posx,Posy) ,         /* Richtung                 */
   robbysroom(Room) ,
   robbysdirection(Absdir) .
```

```
robbyreport:-                                          /* Druckt aktuellen Raum   */
                                                       /* Position und relative   */
   robbyssituation(Room,Posx,Posy,Absdir) ,            /* Richtung des "Roboters" */
   print(' Room : ') , print(Room) ,
   print('  Position :') ,
   print(Posx) , print(',') ,
   print(Posy) , nl ,
   print(' Direction of movement : ') ,
   print(Absdir) , nl .


/*------------------------- INIT ENVIRONMENT --------------------------*/

/* Zwei Zufallszahlen X und Y bestimmen die Abmessungen des in Felder   */
/* unterteilten Raumes,  6 < X < 15 ,  4 < Y < 13 .                     */
/* ( Der Raumname und die leere Liste hinter der Koordinatenliste sind  */
/*   fuer Erweiterungen vorgesehen. )                                   */

initenvironment:-
   random(3,X1) , X is X1+6 ,
   random(3,Y1) , Y is Y1+4 ,
   (retract(room(room1,_,_)) ; true) ,  /* falls vorh. alten Raum entf. */
   asserta(room(room1,¢X,Y!,¢!)) .      /* und neue Groesse speichern.  */
```

```
/*------------------------  R A N D O M  ------------------------------*/

seed(127).                                  /* random erzeugt eine Pseudo- */
                                            /* zufallszahl N mit:          */
random(R,N):-                               /*   1  <=  N  <=  R .         */
                                            /* "Pseudo": gleiche           */
   retract(seed(S)),                        /* Anfangspopulation bei const.*/
   N is (S mod R) + 1,                      /* Planlaenge und Poolgroesse. */
   Newseed is (125*S + 1) mod 100007 ,
   asserta(seed(Newseed)) ,!.


/*---------------------  P R I M I T I V E S  ---------------------------*/

/* Hilfsprozeduren, die verstreut im ganzen g.A. benoetigt werden.        */


debug(Pred,Val):-                           /* "Variablenspeicher", um die */
   Inclause =.. [Pred,_],                   /* Parameteruebergaben nicht   */
   Clause =.. [retractall,Inclause],        /* zu sehr aufzublaehen.       */
   call(Clause),                            /* "Umstaendliche" Formulierung*/
   Newinclause =.. [Pred,Val],              /* wegen vermuteter Interpreter-*/
   New =.. [asserta,Newinclause],           /* Singularitaeten.            */
   call(New).


decrement(Pred):-                           /* Erniedrigt das Argument des */
   Clause1 =.. [Pred,Oldcount],             /* fact mit Namen Pred um 1.   */
   Clause2 =.. [retract,Clause1] ,          /* (wegen der Umstaende siehe  */
   call(Clause2) ,                          /* debug)                      */
   Newcount is Oldcount - 1 ,
   Clause3 =.. [Pred,Newcount] ,
   Clause4 =.. [asserta,Clause3] ,
   call(Clause4) .


increment(Pred):-                           /* Erhoeht das Argument des    */
   Clause1 =.. [Pred,Oldcount] ,            /* fact Pred um eins.          */
   Clause2 =.. [retract,Clause1] ,
   call(Clause2) ,
   Newcount is Oldcount + 1 ,
   Clause3 =.. [Pred,Newcount] ,
   Clause4 =.. [asserta,Clause3] ,
   call(Clause4) .


retractall(X):-                             /* Entfernt alle Vorkommen des */
   retract(X),                              /* Pradikates mit Namen X, egal, ob es */
   fail.                                    /* als fact oder als rule vorkommt.    */
retractall(X):-
   retract((X:- Y)),
   fail.
retractall(_).


associate([A | Result] | _],A,Result).      /* holt aus den Standard-  */
associate([_ | Z ],A,Result):-              /* listen das Element mit  */
   associate(Z,A,Result).                   /* dem Bezeichner A.       */
associate([],_,[]).
```

```
maplist(_,¢[],¢[]).                    /* Ruft die Prozedur P so oft auf,   */
                                       /* wie die beiden Listen lang sind.  */
maplist(P,¢X|L],¢Y|M]):-               /* Dabei werden die Elemente der 1.  */
                                       /* Liste als erste Argumente von P,  */
    Q =.. ¢P,X,Y],                     /* die Elemente der 2. Liste als     */
    call(Q),                           /* zweite Argumente verwendet.       */
    maplist(P,L,M).                    /* P muss true liefern!              */


map(_,¢[]).                            /* wie maplist, aber nur je ein      */
map(Action,¢X|L]):-                    /* Argument X (aus der Liste) fuer   */
    Q =.. ¢Action,X],                  /* action. Fuehrt also mit allen     */
    call(Q),                           /* El. der Liste Action(X) aus.      */
    map(Action,L).                     /* Action muss true liefern!         */


sublist(_,¢[],¢[]).                        /* wie maplist, jedoch wird bei  */
sublist(Pred,¢X|Rest],¢Y|Tail]):-          /* fail von Pred (ist erlaubt)   */
    Q =.. ¢Pred,X,Y],                      /* das erste Arg. von Pred durch */
    call(Q),                               /* das naechste El. der 1. Liste */
    sublist(Pred,Rest,Tail).               /* ersetzt (2. Arg. wird in      */
sublist(Pred,¢_|Rest],List):-              /* diesem Fall nicht geaendert). */
    sublist(Pred,Rest,List).               /* (Fuer buildmarklist)          */


append(¢[],Y,Y).                           /* Element hinten an Liste       */
append(¢A | X],Y,¢A | Z]):-              /* anhaengen.                    */
    append(X,Y,Z).

delete(X,¢[],¢[]).                         /* Erstes Vorkommen eines El.    */
delete(X,¢X | R],R).                       /* X aus einer Liste loeschen.   */
delete(X,¢Y | R],¢Y | Z]):-
    delete(X,R,Z).

deleteall(X,¢[],¢[]).                      /* Alle Vorkommen eines Elementes*/
deleteall(X,¢X | R],S):-                   /* X aus einer Liste loeschen.   */
    deleteall(X,R,S).
deleteall(X,¢Y | R],¢Y | Z]):-
    deleteall(X,R,Z).

member(X,¢X|_]).                           /* Stellt fest, ob X in der Liste*/
member(X,¢_|Y]):-                          /* enthalten ist.                */
    member(X,Y).

reverse(X,Y):-                         /* reverse schreibt die Elemente der */
    revzap(X,¢[],Y).                   /* Liste X in umgekehrter Reihenfolge*/
                                       /* in die Liste Y.                   */
revzap(¢[],Y,Y).                       /* (Hilfsliste noetig)               */
revzap(¢A | X],Y,Z):-
    revzap(X,¢A | Y],Z) .
```

```
cprolog -g 1000 -l 1000 -t 1000



Dec  3 10:41 1986  start Page 1


start:- consult(initevolution),
        consult(evolution),
        consult(genoperators),
        consult(primitives),
        consult(operations).




Dec 23 15:42 1986  evo1 Page 1

cprolog -g 1000 -l 1000 -t 1000 <evost/evostart1



Dec 23 15:41 1986  evost/evostart1 Page 1


consult(initevolution),
consult(evolution),
consult(genoperators),
consult(primitives),
consult(operations).
initevolution.

debug(wtime,1).
debug(wlength,10).
debug(wfields,20).
debug(werrornr,15).
debug(wcorrectnr,2).

debug(probmut,5).
debug(probparamut,10).
debug(probdel,35).
debug(probcro,55).
debug(probins,60).
debug(probinv,80).
debug(probjumpins,101).

debug(poolsize,10).
debug(planlength,10).

debug(maxinstrnumber,25).
debug(numberoftests,1).

paramreport.
contevolution(200).
contevolution(400).
contevolution(800).
halt.
```

```
Feb 11 14:05 1987  data/data10 Page 1


C-Prolog version 1.5
| ?- |    |    |    |    initevolution consulted 7560 bytes 8040.5313 sec
evolution consulted 6743 bytes 7546.2188 sec.
genoperators consulted 7228 bytes 6545.875 sec.
primitives consulted 5696 bytes 5170.875 sec.
operations consulted 2316 bytes 2139.062 sec.

yes
| ?- random pool generated
all plans criticised

evolution initialised

| ?- |    --------------- POOL PARAMS -----------------
poolsize : 10
  planlength : 10

------------ EVOLOOP PARAMS -----------------
adapttime :  2
maxinstrnumber :  250
numberoftests :  1

---------- Prob. of genetic Operators --------------
p(mutation)% 10
p(paramutation)% 10
p(delete)% 10
p(crossover)% 25
p(insert)% 10
p(invert)% 10
p(jumpinsert)% 25

------------ CRITIC PARAMS -----------------
wtime : 1
wlength : 1
wfields : 4
werrornr : 0
correcttnr : 2

yes
| ?- evoloop Nr. 200   current best Value : 69
evoloop Nr. 197   current best Value : 117
evoloop Nr. 196   current best Value : 141
evoloop Nr. 194   current best Value : 235
evoloop Nr. 192   current best Value : 236
evoloop Nr. 184   current best Value : 237
evoloop Nr. 176   current best Value : 238
evoloop Nr. 152   current best Value : 245
evoloop Nr. 145   current best Value : 246
evoloop Nr. 143   current best Value : 247
evoloop Nr. 114   current best Value : 248
evoloop Nr. 106   current best Value : 251
evoloop Nr. 96    current best Value : 252
evoloop Nr. 87    current best Value : 256
evoloop Nr. 16    current best Value : 257
evoloop Nr. 5    current best Value : 258
evoloop Nr. 1    current best Value : 259
plannumber 10,     effectiveness 259
    step(backward)
    step(right)
    step(left)
    step(forward)
marke(1)
```

```
    step(north)
    jump(recognize(forward,wall),marke(5))
    step(north)
    step(forward)
marke(5)
marke(3)
    step(east)
    step(west)
    jump(true,marke(3))
marke(4)
    step(west)
    step(right)


plannumber 9,     effectiveness 2
    step(north)
marke(1)
    step(west)
    step(left)
    jump(true,marke(1))
    jump(true,marke(1))
    step(forward)
    step(right)
    jump(true,marke(3))
    jump(true,marke(3))


plannumber 3,     effectiveness 16
    step(north)
    step(east)
marke(2)
marke(1)
marke(3)
    step(east)
marke(4)
    step(right)
    jump(true,marke(204))


plannumber 7,     effectiveness 2
    step(backward)
    step(left)
    step(backward)
    jump(recognize(forward,_3210),marke(2))
    step(west)
    step(forward)
    step(left)
    step(forward)
marke(1)
    step(left)


plannumber 6,     effectiveness 257
    step(east)
    step(north)
    step(forward)
marke(4)
    step(east)
    step(west)
    jump(true,marke(4))
marke(3)
    step(west)
```

```
plannumber 5,     effectiveness 2
    jump(true,marke(2))
    step(backward)
marke(3)
    step(left)
    step(west)
marke(1)
    jump(true,marke(5))
    step(right)
    step(north)
marke(2)


plannumber 4,     effectiveness 242
marke(1)
    jump(true,marke(3))
marke(3)
marke(2)
    step(forward)
    jump(recognize(right,wall),marke(1))
    step(east)
marke(6)
    step(south)
marke(9)
marke(8)
    step(north)
    jump(true,marke(6))
marke(7)
    step(north)
    step(forward)
    jump(recognize(left,wall),marke(8))
marke(5)
marke(4)
    step(east)


plannumber 3,     effectiveness 2
    step(south)
marke(2)
    jump(true,marke(3))
    step(backward)
marke(1)
    step(south)
marke(3)
    step(south)
    step(east)
    step(north)


plannumber 2,     effectiveness 3
    jump(true,marke(3))
    jump(true,marke(3))
    step(forward)
    step(north)
marke(2)
marke(3)
marke(1)
    step(forward)
    step(south)
marke(4)
```

```
   jump(true,marke(102))


plannumber 1,     effectiveness 243
marke(4)
marke(1)
   jump(true,marke(3))
marke(3)
   jump(true,marke(4))
   step(right)
   jump(true,marke(104))



yes
| ?- evoloop Nr. 169   current best Value : 260
evoloop Nr. 113   current best Value : 261
evoloop Nr. 73   current best Value : 264
evoloop Nr. 32   current best Value : 340
plannumber 10,     effectiveness 264
   step(backward)
   step(right)
   step(left)
   step(forward)
marke(1)
   step(north)
   step(right)
   jump(recognize(forward,wall),marke(5))
   step(north)
   step(forward)
marke(5)
   step(east)
   step(west)
   jump(true,marke(5))
marke(4)
   step(right)


plannumber 9,     effectiveness 258
   step(north)
   step(right)
marke(2)
   step(forward)
   step(east)
marke(3)
   step(east)
   step(south)
marke(9)
marke(8)
   step(east)
   step(west)
   jump(true,marke(9))
   jump(true,marke(7))
   step(north)
marke(4)
marke(5)
   step(west)
   jump(true,marke(3))
   jump(true,marke(3))


plannumber 8,     effectiveness 23
```

```
    step(north)
    step(forward)
marke(5)
    step(east)
    step(west)
    step(north)
    step(forward)
marke(4)
marke(1)
marke(2)
    step(east)
marke(3)
    step(right)
    jump(true,marke(204))


plannumber 7,     effectiveness 244
    step(backward)
marke(4)
marke(2)
    step(east)
    step(west)
    jump(true,marke(4))
marke(3)
    step(right)
    step(left)
    step(forward)
marke(1)
    step(left)


plannumber 6,     effectiveness 25
    step(east)
    step(north)
    step(forward)
marke(1)
    step(east)
    step(west)
    jump(true,marke(1))


plannumber 5,     effectiveness 2
    jump(true,marke(2))
    step(backward)
marke(3)
    step(left)
    step(west)
marke(1)
    jump(true,marke(5))
    step(right)
    step(north)
marke(2)


plannumber 4,     effectiveness 265
marke(1)
    jump(true,marke(3))
marke(3)
marke(2)
    step(forward)
    step(right)
    step(left)
```

```
   step(forward)
marke(6)
   step(north)
   step(right)
   jump(recognize(forward,wall),marke(8))
   step(north)
   step(forward)
marke(8)
   step(east)
   step(west)
   jump(true,marke(3))
marke(7)
marke(5)
marke(4)


plannumber 3,     effectiveness 254
   step(south)
   step(right)
marke(5)
   jump(true,marke(6))
   step(west)
   step(east)
marke(4)
marke(6)
   step(left)
   step(forward)
   step(north)
   step(right)
marke(3)
   step(south)
marke(1)
   step(backward)
   jump(true,marke(3))
   step(north)


plannumber 2,     effectiveness 340
   jump(true,marke(3))
   jump(true,marke(3))
marke(3)
   step(forward)
   step(north)
marke(1)
   step(right)
   jump(recognize(forward,wall),marke(4))
   step(north)
   step(forward)
marke(4)
   step(south)
   jump(true,marke(3))
marke(2)


plannumber 1,     effectiveness 245
marke(4)
marke(1)
   jump(true,marke(3))
marke(3)
   jump(true,marke(4))
```

```
C-Prolog version 1.5
| ?- |    |    |    |    initevolution consulted 7560 bytes 7796.9375 sec.
evolution consulted 6748 bytes 7545.8438 sec.
genoperators consulted 7228 bytes 6594.5313 sec.
primitives consulted 5696 bytes 5270.5313 sec.
operations consulted 2316 bytes 2275.5234 sec.

yes
| ?- random pool generated
all plans criticised

evolution initialised

| ?- |    ---------- POOL PARAMS ----------
poolsize : 10
  planlength : 10

---------- EVOLOOP PARAMS ----------
adapttime :  2
maxinstrnumber :  250
numberoftests :  1

---------- Prob. of genetic Operators ----------
p(mutation)% 10
p(paramutation)% 10
p(delete)% 0
p(crossover)% 40
p(insert)% 5
p(invert)% 10
p(jumpinsert)% 20

---------- CRITIC PARAMS ----------
wtime : 1
wlength : 2
wfields : 6
werrornr : 0
correcttnr : 1

yes
| ?- evoloop Nr. 195   current best Value : 3
evoloop Nr. 187   current best Value : 6
evoloop Nr. 134   current best Value : 13
evoloop Nr. 132   current best Value : 19
evoloop Nr. 130   current best Value : 22
evoloop Nr. 140   current best Value : 24
evoloop Nr. 67   current best Value : 25
evoloop Nr. 43   current best Value : 54
evoloop Nr. 20   current best Value : 86
plannumber 10,    effectiveness 18
marke(1)
   step(forward)
   step(right)
   step(forward)
   step(north)
marke(2)
   jump(recognize(east,wall),marke(2))
   step(forward)
   step(forward)


plannumber 9,    effectiveness 2
   step(north)
```

```
marke(1)
    step(west)
    step(left)
    jump(true,marke(1))
    jump(true,marke(1))
    step(forward)
    step(right)
    jump(true,marke(3))
    jump(true,marke(3))


plannumber 8,    effectiveness 22
    step(north)
    step(east)
    step(south)
marke(1)
    step(left)
    step(east)
    step(forward)


plannumber 7,    effectiveness 86
marke(2)
    step(backward)
    step(left)
    step(east)
    step(forward)
    step(left)
    jump(true,marke(2))
    step(forward)
marke(1)
    step(left)


plannumber 6,    effectiveness 19
marke(1)
    step(east)
    step(backward)
    step(south)
    step(north)
    step(forward)
    step(right)
    step(left)


plannumber 5,    effectiveness 2
    jump(true,marke(2))
    step(backward)
marke(3)
    step(left)
    step(west)
marke(1)
    jump(true,marke(5))
    step(right)
    step(north)
marke(2)


plannumber 4,    effectiveness 54
marke(1)
    jump(true,marke(4))
    step(north)
```

```
marke(4)
marke(3)
    step(forward)
    step(right)
    step(forward)
    jump(recognize(right,wall),marke(1))
    jump(true,marke(3))
    step(backward)
    step(south)
    step(north)
    step(north)


plannumber 3,     effectiveness 5
    step(south)
    step(forward)
    step(north)


plannumber 2,     effectiveness 2
    jump(true,marke(3))
    jump(true,marke(3))
    step(forward)
    step(north)
marke(2)
marke(3)
marke(1)
    step(forward)
    step(south)
    step(west)


plannumber 1,     effectiveness 2
marke(2)
    jump(true,marke(2))
    step(right)
marke(3)
    step(west)
    step(forward)
marke(1)
    jump(true,marke(2))
    step(west)



yes
| ?- evoloop Nr. 156   current best Value : 88
evoloop Nr. 105   current best Value : 102
evoloop Nr. 82   current best Value : 136
evoloop Nr. 77   current best Value : 256
plannumber 10,    effectiveness 22
marke(1)
    step(forward)
    step(right)
    step(left)
    step(forward)
    step(north)
marke(2)
    jump(recognize(east,wall),marke(2))
    step(forward)
    step(forward)
```

```
plannumber 9,     effectiveness 6
    step(north)
marke(1)
    step(forward)
    step(right)
    jump(true,marke(3))
    jump(true,marke(3))


plannumber 8,     effectiveness 30
    step(north)
    step(east)
    step(south)
marke(1)
    step(left)
    step(east)
    step(forward)
    step(east)
    step(forward)


plannumber 7,     effectiveness 133
marke(2)
    step(backward)
    jump(true,marke(3))
marke(3)
    step(left)
    step(east)
    step(forward)
    step(east)
    jump(recognize(north,wall),marke(4))
    jump(true,marke(2))
    step(forward)
marke(1)
marke(4)
    step(left)


plannumber 6,     effectiveness 246
    jump(true,marke(5))
marke(1)
    step(east)
marke(5)
marke(3)
marke(4)
marke(2)
    step(forward)
    jump(recognize(forward,¢),marke(4))
    step(right)
    step(forward)
marke(4)
    jump(true,marke(2))
    step(forward)
    step(east)


plannumber 5,     effectiveness 2
    jump(true,marke(2))
    step(backward)
marke(3)
    step(left)
```

```
    step(west)
marke(1)
    jump(true,marke(5))
    step(right)
    step(north)
marke(2)


plannumber 4,     effectiveness 256
marke(1)
    jump(true,marke(4))
    step(north)
marke(3)
marke(4)
marke(2)
    step(forward)
    jump(recognize(forward,?!),marke(4))
    step(right)
    step(forward)
    jump(recognize(right,wall),marke(1))
    jump(true,marke(2))
    step(backward)
    step(south)
    step(north)
    step(north)


plannumber 3,     effectiveness 55
    step(south)
    step(forward)
    step(north)
    step(backward)
    step(south)
marke(2)
    step(left)
    step(forward)
marke(1)
    step(north)
marke(4)
marke(5)
marke(3)
    step(forward)
    jump(recognize(forward,?!),marke(5))
    step(right)
    step(forward)


plannumber 2,     effectiveness 2
    jump(true,marke(3))
    jump(true,marke(3))
    step(forward)
    step(north)
marke(2)
marke(3)
marke(1)
    step(forward)
    step(south)
    step(west)


plannumber 1,     effectiveness 2
marke(2)
```

```
    jump(true,marke(2))
    step(right)
marke(3)
    step(west)
    step(forward)
marke(1)
    jump(true,marke(2))
    step(west)



yes
| ?- plannumber 10,     effectiveness 227
marke(1)
    step(forward)
marke(3)
marke(2)
    step(forward)
    jump(recognize(forward,4),marke(3))
marke(5)
marke(6)
marke(4)
    step(forward)
    step(east)
marke(10)
marke(8)
marke(9)
marke(7)
    step(forward)
    jump(recognize(forward,4),marke(9))
    step(left)
    step(right)
    step(forward)
    jump(true,marke(7))


plannumber 9,     effectiveness 16
    step(north)
    step(east)
    step(forward)
    step(east)
marke(2)
marke(1)
    step(north)


plannumber 8,     effectiveness 47
marke(3)
    step(north)
    step(east)
    step(south)
marke(1)
    step(left)
    jump(true,marke(3))
    step(east)
    step(forward)
    step(east)
    step(forward)
    jump(true,marke(2))
marke(2)
    step(left)
    step(east)
```

```
    step(forward)
     .

plannumber 7,     effectiveness 172
marke(2)
    step(backward)
    jump(true,marke(3))
marke(3)
    step(left)
    step(left)
    step(east)
    step(east)
    step(east)
    jump(recognize(north,wall),marke(4))
    jump(true,marke(2))
    jump(recognize(backward,wall),marke(5))
    step(forward)
marke(1)
marke(5)
marke(4)
    step(left)


plannumber 6,     effectiveness 246
    jump(true,marke(5))
marke(1)
    step(east)
marke(5)
marke(3)
marke(4)
marke(2)
    step(forward)
    jump(recognize(forward,¢1),marke(4))
    step(right)
    step(forward)
marke(4)
    jump(true,marke(2))
    step(forward)
    step(east)


plannumber 5,     effectiveness 2
    jump(true,marke(2))
    step(backward)
marke(3)
    step(left)
    step(west)
marke(1)
    jump(true,marke(5))
    step(right)
    step(north)
marke(2)


plannumber 4,     effectiveness 256
marke(1)
    jump(true,marke(4))
    step(north)
marke(3)
marke(4)
marke(2)
    step(forward)
```

```
    jump(recognize(forward,¢!),marke(4))
    step(right)
    step(forward)
    jump(recognize(right,wall),marke(1))
    jump(true,marke(2))
    step(backward)
    step(south)
    step(north)
    step(north)


plannumber 3,     effectiveness 171
    step(south)
    step(forward)
marke(6)
    step(north)
    step(backward)
    step(south)
marke(2)
    step(left)
    step(forward)
marke(1)
    step(north)
marke(4)
marke(5)
marke(3)
    step(forward)
    jump(true,marke(6))
    jump(recognize(forward,↑!),marke(5))
    step(right)


plannumber 2,     effectiveness 234
    jump(true,marke(3))
    jump(true,marke(3))
    step(forward)
    step(north)
marke(2)
marke(3)
marke(1)
    step(forward)
    step(east)
marke(7)
marke(5)
marke(6)
marke(4)
    step(forward)
    jump(recognize(forward,¢!),marke(6))
    step(right)
    step(forward)
    jump(true,marke(4))
    step(forward)
    step(east)


plannumber 1,     effectiveness 2
marke(2)
    jump(true,marke(2))
    step(right)
marke(3)
    step(west)
    step(forward)
```

```
marke(1)
    jump(true,marke(2))
    step(west)


yes
| ?-
% Prolog execution halted
```