Министерство образования Республики Беларусь

Учреждение образования
“Белорусский государственный университет
информатики и радиоэлектроники”

Факультет информационных технологий и управления

Кафедра интеллектуальных информационных технологий

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине «Логические основы интеллектуальных систем»
на тему
«Логическое программирование поиска решения задачи»

Вариант 4

Выполнил студент гр. 121703 Терлеев А.С
Проверил Ивашенко В. П.

Минск 2023

Цель:
Приобрести навыки логического программирования поиска решения задач. Реализовать программу на языке Prolog, решающую поставленную задачу.
Постановка задачи:
Необходимо реализовать программу решающую поставленную задачу. В результате выполнения программы должен выводиться результат работы, описывающий решение задачи.
Условие задачи:
Два берега реки. На одном из них полицейский с заключенным, мама с дочерьми и отец с сыновьями. Необходимо с помощью плота, вмещающего не более двух человек, переправить всех персонажей на другой берег реки. Управлять плотом могут только полицейский и родители. Заключённого нельзя оставлять ни с одним из членов семьи. Папе не разрешается оставаться с дочерьми без присутствия матери. Маме не разрешается находиться с сыновьями без присутствия отца.

Реализация:
Для решения задачи использовался SWI-Prolog.
Для представления состояние нашей задачи будет использоваться функциональный терм state(list, pos), в котором первый параметр определяет список людей на левом берегу второй параметр - местоположение плота. Pos может принимать значения left или right, что означает нахождение плота у левого или правого берега соответственно. Таким образом, state([son, police, crimenal], left) означает, что плот сейчас находится у левого берега, на левом берегу находится один сын, полицейский и преступник, тогда на правом берегу находятся отец, мать, один сын и две дочери. Тогда терминальным состоянием будет state([], any).

В программе используются:

1) **Встроенные предикаты:**
    a) forall(Cond, Action).
       Для всех возможных Cond, Action может быть доказано.
    b) member (X, List).
       Истинно, если X является членом списка List.
    c) subtract(Source, Subtrahend, Result).

Вычитает из списка Source содержимое списка Subtrahend и записывает результат в список Result.

d) sort(Source_list, Destination_list).
   Истинно, если результат сортировки в обычном порядке элементов списка Source_list может быть унифицирован со списком Destination_list. При этом все повторяющиеся элементы списка Source_list удаляются в полученном после сортировки результате.
e) format(Format_str, Arguments).
   Печатает содержимое Format_str, подставляя на указанные места соответствующие элементы списка Arguments.
f) Action1; Action2
   Дизъюнкция. Возвращает истину, если истинно Action1 или Action2. Вернёт истину, если истинно и Action1, и Action2.
g) Action1, Action2
   Конъюнкция. Возвращает истину, если истинно и Action1, и Action2.
h) Condition -> Action.
   Импликация. Action выполняется, если истинно Condition.
i) !
   Отсечение. Применяется, если хватает одного решения, если программист знает о единственности решения, если корректно лишь первое решение и нужно удалить другие решения.
j) =
   Достигается вместе с унификацией и может повлечь подстановку.
k) \+
   Описывает логическое отрицание.
l) :-
   Описывает логическое "если", и служит для разделения двух частей правила: заголовка и тела.

2) Факты:

a) son(son1). – факт, утверждающий, что son1 является сыном
b) son(son2). – факт, утверждающий, что son2 является сыном
c) daughter(daughter1). – факт, утверждающий, что daughter1 является дочерью

d) daughter(daughter2). – факт, утверждающий, что daughter2 является дочерью
e) adult(father). – факт, утверждающий, что father является взрослым
f) adult(mother). – факт, утверждающий, что mother является взрослым
g) adult(police). – факт, утверждающий, что police является взрослым
h) all([son1, son2, father, daughter1, daughter2, mother, criminal, police]). – список всех людей
i) allsafe([_]). – факт, который возвращает истину, если на берегу находится один человек.
j) allsafe([]). – факт, который возвращает истину, если на берегу нет людей.

3) Правила:

a) notsafe_(criminal, X) :- X \= police. – истинно, если с criminal находится не police
b) notsafe_(mother, Y) :- son(Y). – истинно, если с mother находится сын
c) notsafe_(father, Y) :- daughter(Y). – истинно, если с father находится дочь
d) notsafe(X, Y) :- notsafe_(X, Y); notsafe_(Y, X). – истинно, если notsafe_(X, Y) или notsafe_(Y, X)
e) safe(X, Y) :- \+ notsafe(X, Y). - истинно, если не notsafe(X, Y)
f) safebridge([X, Y]) :- (adult(X); adult(Y)), safe(X, Y), !. – истинно, если (X – взрослый или Y – взрослый) и safe(X, Y)
g) safebridge([X]) :- adult(X). – истинно, если X – Взрослый
h) allsafe(L) :-

```
forall(
    member(H, L),
    (
    adult(H);
    son(H), (member(mother, L) -> member(father, L); true);
    daughter(H), (member(father, L) -> member(mother, L); true);
```

```
        H = criminal, member(police, L)
        )
    ), !.
```

Истинно, если для всех элементов L выпоняется: H — взрослый, или H — сын и если в L есть mother, то в L должен быть father, или H — дочь и если в L есть father, то в L должен быть mother, или H - criminal, то в L должен быть police.

i) `allPairs([H | T], [H, P2]) :- member(P2, T).`
Первым и вторым аргументами принимает списки. Во второй список запишется результат, состоящий из двух элементов: H - голова первого списка и P2 - элемент хвоста T первого списка.

j) `allPairs([_ | T], P) :- allPairs(T, P).`
Первым аргументом и вторым аргументами принимает списки. Во второй список запишет результат allPairs для хвоста первого списка. В совокупности с предыдущим правилом allPairs позволяет перебрать все возможные пары элементов в каком-либо списке (используются для определения пар, которые поместятся на плоту).

k)
```
step(state(Left1, left), state(Left2, right)) :-
    step_(state(Left1, left), state(Left2, right)).
```
Определяет следующий шаг с помощью step_ в ситуации, когда плот находится на левом берегу.

l)
```
step(state(Left1, right), state(Left2, left)) :-
    all(All),
    subtract(All, Left1, Right1),
    step_(state(Right1, left), state(Right2, right)),
    subtract(All, Right2, Left2).
```
Определяет следующий шаг с помощью step_ в ситуации, когда плот находится на правом берегу. После чего определяет существ и предметы, которые остались на левом берегу.

m)
```
step_(state(Left1, left), state(Left2, right)) :-
    ( allPairs(Left1, OnBridge)
    ; member(A, Left1),
      OnBridge = [A]
    ),
    safebridge(OnBridge),
```

```
    subtract(Left1, OnBridge, Left2),
    allsafe(Left2),
    all(All),
    subtract(All, Left2, Right),
    allsafe(Right).
```

Истинно, если на береге можно найти такую пару(или одного человека) которая (который) могла (мог) бы переправиться на противоположный берег, при выполнении правил safebridge, для пары (человека), переправляются (отправляющегося) на плоту, при выполнении правила allsafe() для людей на левом и правом берегах после переправы.

n) notequal(state(L1, P1), state(L2, P2)) :-

```
\+ (
        P1 = P2,
        sort(L1, L),
        sort(L2, L)
        ).
```

Истинно, если state(L1, P1) не эквивалентен state(L2, P2) (если списки состоят из разных элементов или плот находится на разных берегах)

o) path(Inp, PrevSteps, [step(Inp, S1) | Steps]) :-

```
    step(Inp, S1),
    duplCheck(S1, PrevSteps),
    branching(S1, [step(Inp, S1) | PrevSteps], Steps).
```

Истинно, если существует путь от состояния Inp, до терминального состояния state([], _), является целью задачи.

Листинг кода программы:

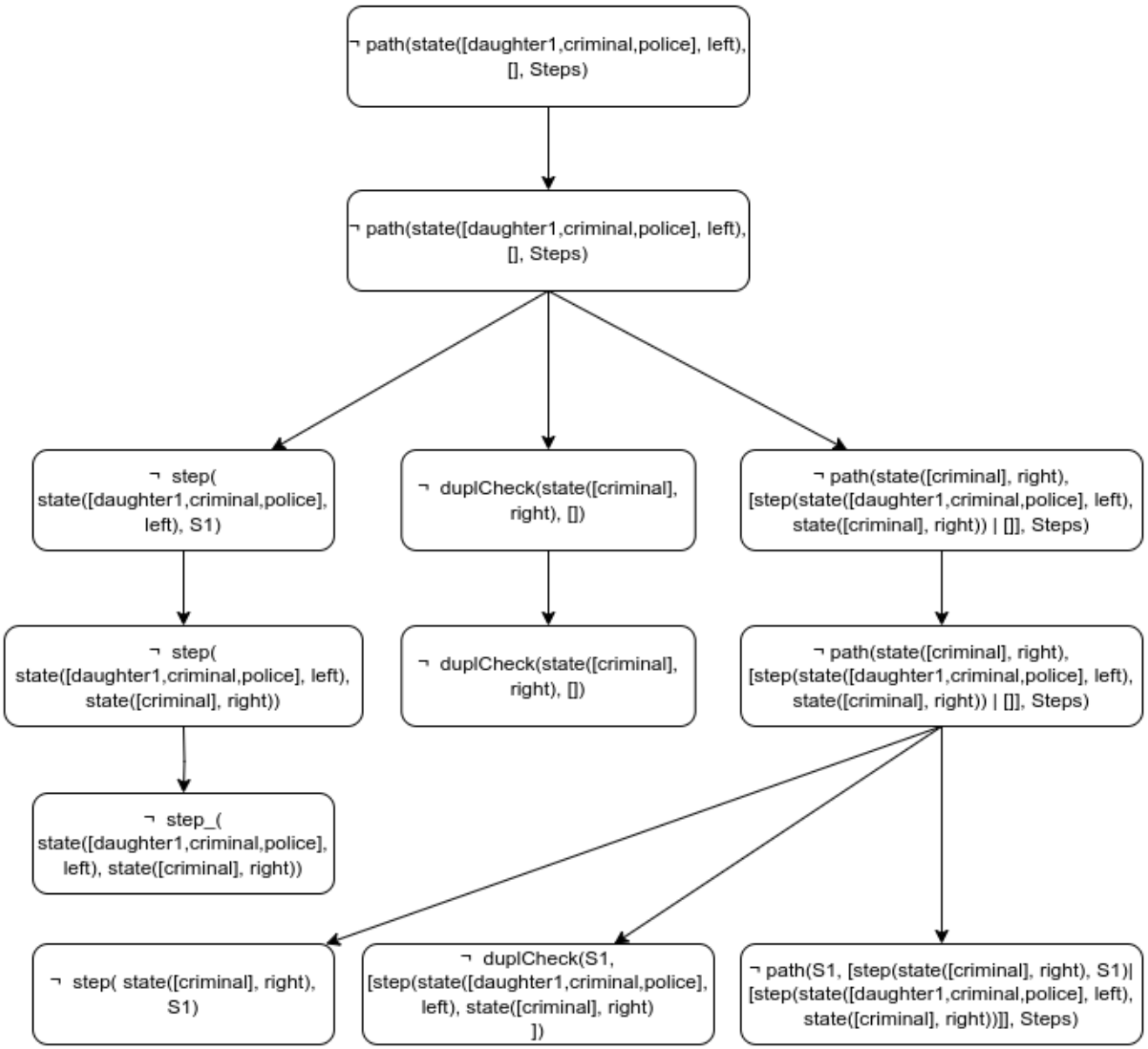
```
% Лабораторная работа по ЛОИС №2
% Вариант: 4
% Автор: Терлеев А.С.
% Источники:
% 1. Логические основы интеллектуальных систем. Практикум : учеб.- метод. пособие / В. В. Голенков [и др.]. – Минск : БГУИР, 2011. – 70 с. : ил. ISBN 978-985-488-487-5.
% 2. SWI Prolog [Электронный ресурс]. -- Режим доступа https://www.swi-prolog.org/
% 3. Искусственный интеллект: современный подход, 4-е изд., том 1 Решение проблем: знания и рассуждения / Рассел, С. и Питер, Н. – Москва : Диалектика, 2021 – 702 с.

son(son1).
son(son2).

daughter(daughter1).
daughter(daughter2).

adult(father).
adult(mother).
adult(police).

notsafe_(criminal, X) :- X \= police.
notsafe_(mother, Y) :- son(Y).
notsafe_(father, Y) :- daughter(Y).

notsafe(X, Y) :- notsafe_(X, Y); notsafe_(Y, X).

safe(X, Y) :- \+ notsafe(X, Y).

safebridge([X, Y]) :- (adult(X); adult(Y)), safe(X, Y), !.
safebridge([X]) :- adult(X).

all([
    son1, son2, father,
    daughter1, daughter2, mother,
    criminal, police
   ]).

allsafe(L) :-
```

```
   forall(
      member(H, L),
      (
         adult(H);
         son(H), (member(mother, L) -> member(father, L); true);
         daughter(H), (member(father, L) -> member(mother, L); true);
         H = criminal, member(police, L)
      )
   ), !.

allsafe([_]).
allsafe([]).

allPairs([H | T], [H, P2]) :-
 member(P2, T).

allPairs([_ | T], P) :-
 allPairs(T, P).

step_(state(Left1, left), state(Left2, right)) :-
   ( allPairs(Left1, OnBridge)
   ; member(A, Left1),
      OnBridge = [A]
   ),

   safebridge(OnBridge),

   subtract(Left1, OnBridge, Left2),
   allsafe(Left2),

   all(All),
   subtract(All, Left2, Right),
   allsafe(Right).

step(state(Left1, left), state(Left2, right)) :-
   step_(state(Left1, left), state(Left2, right)).

step(state(Left1, right), state(Left2, left)) :-
   all(All),
   subtract(All, Left1, Right1),
   step_(state(Right1, left), state(Right2, right)),
   subtract(All, Right2, Left2).

notequal(state(L1, P1), state(L2, P2)) :-
```

```
    \+ (
      P1 = P2,
       sort(L1, L),
       sort(L2, L)
      ).

duplCheck(_, []).

duplCheck(CurrState, [step(State1, _)|T]):-
    notequal(CurrState, State1),
    duplCheck(CurrState, T).

path(state([], _), _, []).

path(Inp, PrevSteps, [step(Inp, S1) | Steps]) :-
    step(Inp, S1),
    duplCheck(S1, PrevSteps),
    path(S1, [step(Inp, S1) | PrevSteps], Steps).

valid_state(state(Left, Bridge)) :-
    (Bridge == left; Bridge == right),
    all(All),
    subtract(All, Left, Right),
    append(Left, Right, AllCurr),
    sort(All, Tmp),
    sort(AllCurr, Tmp).

printStep(step(state(L1, Pos1), state(L2, _))) :-
    ( Pos1 = left
    -> subtract(L1, L2, Moved),
       all(All),
       subtract(All, L1, R1),
       format('left: ~w | right: ~w~n', [L1, R1]),
       subtract(All, L1, R2),
       format('left: ~w | raft: ~w -> | right: ~w~n~n', [L2, Moved, R2])
    ; Pos1 = right
    -> subtract(L2, L1, Moved),
       all(All),
       subtract(All, L1, R1),
       format('left: ~w | right: ~w~n', [L1, R1]),
       subtract(All, L2, R2),
       format('left: ~w | raft: ~w <- | right: ~w~n~n', [L1, Moved, R2])
    ).
```

```
solve(StartLeft, StartBridge):-
    valid_state(state(StartLeft, StartBridge)),

    sort(StartLeft, SortedStertLeft),
    all(All),
    allsafe(SortedStertLeft),
    subtract(All, SortedStertLeft, StartRight),
    allsafe(StartRight),

    path(state(SortedStertLeft, StartBridge), [], Steps),
    (format('~nSolution:~n'), forall(member(Step, Steps), printStep(Step))).
```

Дерево логического вывода:



Вывод:

В результате выполнения лабораторной работы, была разработана программа, решающая поставленную задачу, а также построено дерево логического вывода, соответствующее данной программе.

Список использованных источников:

1. Логические основы интеллектуальных систем. Практикум : учеб.-метод. пособие / В. В. Голенков [и др.]. – Минск : БГУИР, 2011. – 70 с. : ил. ISBN 978-985-488-487-5.
2. SWI Prolog [Электронный ресурс]. – Режим доступа https://www.swi-prolog.org/
3. Искусственный интеллект: современный подход, 4-е изд., том 1 Решение проблем: знания и рассуждения / Рассел, С. и Питер, Н. – Москва : Диалектика, 2021 – 702 с.