**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Логическое программирование»**

Тема: Решение логических задач

Вариант 1

|  |  |  |
| --- | --- | --- |
| Студентка гр. 8382 |  | Кузина А.М. |
| Студентка гр. 8382 |  | Кулачкова М.К. |
| Студент гр. 8382 |  | Мирончик П.Д. |
| Преподаватель |  | Родионов С.В. |

Санкт-Петербург

2022

**Цель работы**

Изучение возможности применения Пролога для решения логических задач, освоение подходов к формулированию и решению логических задач на языке Пролог.

**Задание**

Решить головоломку на языке Пролог.

**Вариант 1 – Миссионеры и каннибалы**

К берегу реки подошли 3 миссионера и 3 каннибала. Как им всем безопасно переправиться на другой берег, используя двухместную лодку, если каннибалы могут съесть миссионеров, оказавшихся в меньшинстве? Необходимо вывести порядок перемещений.

**Порядок выполнения работы**

Реализуется динамический предикат variables, использующийся для хранения статических переменных, а также предикаты set и get для их чтения и записи.

:- dynamic(variables/2).

set(Var, Val) :- (\+variables(Var, \_), !; retract(variables(Var, \_))), assertz(variables(Var, Val)).

get(Var, Val) :- variables(Var, Val).

При запуске программы, осуществляющей поиск решения задачи, с помощью этих предикатов задаются значения статических переменных type\_limit и boat\_limit, определяющих число персонажей каждого типа (миссионеров и каннибалов) и вместимость лодки. Это делает программу более универсальной: можно изменить эти параметры в одном месте, и остальной код найдет решение для новых условий.

Реализуется динамический предикат stored для хранения состояния решения задачи, задаваемого числом миссионеров и каннибалов на левом берегу и в лодке и берегом, у которого находится лодка. Также реализуется предикат store, который записывает состояние, если его еще нет в базе данных, и возвращает false в противном случае.

:- dynamic(stored/5).

store(LeftMiss, LeftCann, BoatMiss, BoatCann, Side) :-

\+stored(LeftMiss, LeftCann, BoatMiss, BoatCann, Side),

assertz(stored(LeftMiss, LeftCann, BoatMiss, BoatCann, Side)).

Сохранение состояния таким способом гарантирует, что решение не зациклится.

Реализуются предикаты, осуществляющие проверку условий. Предикат

is\_all\_alive(Miss, Cann) :- Miss >= Cann; Miss =:= 0.

проверяет, что текущее число персонажей каждого типа в заданной локации безопасно, то есть что в заданной локации миссионеров либо нет, либо их число не меньше числа каннибалов.

Предикат

is\_valid(LeftMiss, LeftCann, BoatMiss, BoatCann) :-

get(type\_limit, TypeLimit),

get(boat\_limit, BoatLimit),

RightMiss = TypeLimit - LeftMiss - BoatMiss,

RightCann = TypeLimit - LeftCann - BoatCann,

LeftMiss + BoatMiss =< TypeLimit,

LeftCann + BoatCann =< TypeLimit,

BoatMiss + BoatCann =< BoatLimit,

LeftMiss >= 0, LeftCann >= 0, BoatMiss >= 0, BoatCann >= 0, RightMiss >= 0, RightCann >= 0,

is\_all\_alive(LeftMiss, LeftCann),

is\_all\_alive(BoatMiss, BoatCann),

is\_all\_alive(RightMiss, RightCann).

проверяет корректность текущего состояния: соответствует ли общее число миссионеров и каннибалов установленному переменной type\_limit ограничению, не превышена ли вместимость лодки, не задается ли число персонажей в одной из локаций отрицательным числом, а также осуществляет вызов предиката is\_all\_alive для проверки, безопасно ли соотношение миссионеров и каннибалов на каждом из берегов и в лодке.

Запуск программы осуществляется предикатом go. Начальным является состояние, в котором все миссионеры и каннибалы находятся на левом берегу, лодка пуста и также находится на левом берегу. В процессе поиска решения перемещения лодки и персонажей сохраняются в список.

go :-

set(type\_limit, 3),

set(boat\_limit, 2),

get(type\_limit, TypeLimit),

on\_left(TypeLimit, TypeLimit, 0, 0, []).

Нахождение лодки у левого берега моделируется предикатом

on\_left(LeftMiss, LeftCann, BoatMiss, BoatCann, R) :-

is\_valid(LeftMiss, LeftCann, BoatMiss, BoatCann),

store(LeftMiss, LeftCann, BoatMiss, BoatCann, left),

((BoatMiss + BoatCann > 0, on\_right(LeftMiss, LeftCann, BoatMiss, BoatCann, ['-> drive to the right'| R]));

(LM is LeftMiss + 1, BM is BoatMiss - 1, on\_left(LM, LeftCann, BM, BoatCann, ['missioner left the boat'| R]));

(LC is LeftCann + 1, BC is BoatCann - 1, on\_left(LeftMiss, LC, BoatMiss, BC, ['cannibal left the boat'| R]));

(LM is LeftMiss - 1, BM is BoatMiss + 1, on\_left(LM, LeftCann, BM, BoatCann, ['missioner entered the boat'| R]));

(LC is LeftCann - 1, BC is BoatCann + 1, on\_left(LeftMiss, LC, BoatMiss, BC, ['cannibal entered the boat' | R]))).

Сначала осуществляется проверка корректности текущего состояния, затем предпринимается попытка сохранить состояние – на данном этапе осуществляется возврат из предиката, если решение уже находилось в текущем состоянии. Затем, если лодка не пуста, осуществляется переправа на правый берег и вызывается предикат on\_right. Иначе перебираются все возможные перемещения одного персонажа между лодкой и берегом:

* Миссионер вылезает из лодки на берег;
* Каннибал вылезает из лодки на берег;
* Миссионер садится в лодку;
* Каннибал садится в лодку.

После каждого из перемещений осуществляется рекурсивный вызов предиката on\_left в новом состоянии.

Предикат on\_right реализован аналогично:

on\_right(LeftMiss, LeftCann, BoatMiss, BoatCann, R) :-

is\_valid(LeftMiss, LeftCann, BoatMiss, BoatCann),

store(LeftMiss, LeftCann, BoatMiss, BoatCann, right),

((BoatMiss + BoatCann > 0, on\_left(LeftMiss, LeftCann, BoatMiss, BoatCann, ['-> drive to the right'| R]));

(BM is BoatMiss - 1, on\_right(LeftMiss, LeftCann, BM, BoatCann, ['missioner left the boat'| R]));

(BC is BoatCann - 1, on\_right(LeftMiss, LeftCann, BoatMiss, BC, ['cannibal left the boat'| R]));

(BM is BoatMiss + 1, on\_right(LeftMiss, LeftCann, BM, BoatCann, ['missioner entered the boat'| R]));

(BC is BoatCann + 1, on\_right(LeftMiss, LeftCann, BoatMiss, BC, ['cannibal entered the boat' | R]))).

Конечным является состояние, когда лодка у правого берега, и при этом в лодке и на левом берегу никого нет. Когда решение достигает такого состояния, поиск решения прекращается и список, содержащий историю перемещений, выводится в обратном порядке.

on\_right(0, 0, 0, 0, R) :- writereversed(R).

Для вывода списка в обратном порядке реализован вспомогательный предикат:

writereversed([H]) :- write(H), nl.

writereversed([H | T]) :- writereversed(T), write(H), nl.

**Пример работы программы**

Для нахождения решения требуется вызвать предикат go. Пример выполнения программы приведен на рис. 1.

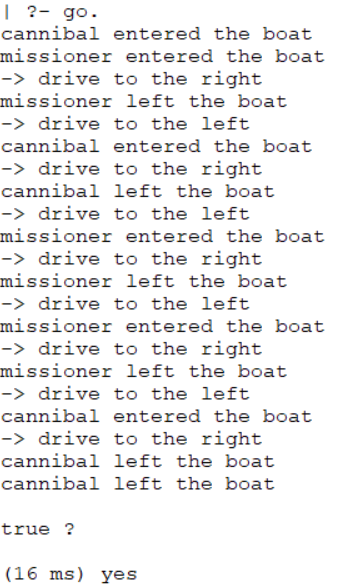


Рисунок – Пример выполнения программы

**Выводы**

Была реализована программа на языке Пролог, решающая логическую задачу про миссионеров и каннибалов.

Зоны ответственности членов бригады:

* Кузина А.М. – тестирование программы;
* Кулачкова М.К. – составление отчета;
* Мирончик П.Д. – написание программы.

Каждый участник бригады проконтролировал действия других участников и разобрался в проделанной ими работе

В ходе выполнения задания возникла следующая трудность: изначально было реализовано решение, схожее с примером про волка, козу и капусту из лекции, однако такое решение не было универсальным, а при установке лимита шагов, превышающего минимально необходимое значение, в решении появлялся цикл – оно все еще находилось и фактически было правильным, но в нем возникала последовательность перемещений, отменяющих друг друга. Эту проблему можно было бы решить созданием цикла по лимиту шагов, но такой способ был бы неэффективным, поэтому решение было полностью переделано, и в новое решение добавлено сохранение состояния.

Не трудность, а отличительная черта: формат вывода реализованной программы не такой, как в лекционных примерах. Выводятся перемещения лодки между берегами и перемещения персонажей между лодкой и берегом, но при этом не выводится, кто находится в лодке на каждом этапе.

Приложение А

исходный код программы

/\*

Бригада 1 группы 8382 - Кузина, Кулачкова, Мирончик

Задание: Миссионеры и каннибалы

К берегу реки подошли 3 миссионера и 3 каннибала. Как им всем безопасно переправиться на другой берег,

используя двухместную лодку, если каннибалы могут съесть миссионеров, оказавшихся в меньшинстве?

(вывести порядок перемещений).

\*/

:- dynamic(variables/2).

set(Var, Val) :- (\+variables(Var, \_), !; retract(variables(Var, \_))), assertz(variables(Var, Val)).

get(Var, Val) :- variables(Var, Val).

:- dynamic(stored/5).

store(LeftMiss, LeftCann, BoatMiss, BoatCann, Side) :-

\+stored(LeftMiss, LeftCann, BoatMiss, BoatCann, Side),

assertz(stored(LeftMiss, LeftCann, BoatMiss, BoatCann, Side)).

writereversed([H]) :- write(H), nl.

writereversed([H | T]) :- writereversed(T), write(H), nl.

go :-

set(type\_limit, 3),

set(boat\_limit, 2),

get(type\_limit, TypeLimit),

on\_left(TypeLimit, TypeLimit, 0, 0, []).

is\_all\_alive(Miss, Cann) :- Miss >= Cann; Miss =:= 0.

is\_valid(LeftMiss, LeftCann, BoatMiss, BoatCann) :-

get(type\_limit, TypeLimit),

get(boat\_limit, BoatLimit),

RightMiss = TypeLimit - LeftMiss - BoatMiss,

RightCann = TypeLimit - LeftCann - BoatCann,

LeftMiss + BoatMiss =< TypeLimit,

LeftCann + BoatCann =< TypeLimit,

BoatMiss + BoatCann =< BoatLimit,

LeftMiss >= 0, LeftCann >= 0, BoatMiss >= 0, BoatCann >= 0, RightMiss >= 0, RightCann >= 0,

is\_all\_alive(LeftMiss, LeftCann),

is\_all\_alive(BoatMiss, BoatCann),

is\_all\_alive(RightMiss, RightCann).

on\_left(LeftMiss, LeftCann, BoatMiss, BoatCann, R) :-

is\_valid(LeftMiss, LeftCann, BoatMiss, BoatCann),

store(LeftMiss, LeftCann, BoatMiss, BoatCann, left),

((BoatMiss + BoatCann > 0, on\_right(LeftMiss, LeftCann, BoatMiss, BoatCann, ['-> drive to the right'| R]));

(LM is LeftMiss + 1, BM is BoatMiss - 1, on\_left(LM, LeftCann, BM, BoatCann, ['missioner left the boat'| R]));

(LC is LeftCann + 1, BC is BoatCann - 1, on\_left(LeftMiss, LC, BoatMiss, BC, ['cannibal left the boat'| R]));

(LM is LeftMiss - 1, BM is BoatMiss + 1, on\_left(LM, LeftCann, BM, BoatCann, ['missioner entered the boat'| R]));

(LC is LeftCann - 1, BC is BoatCann + 1, on\_left(LeftMiss, LC, BoatMiss, BC, ['cannibal entered the boat' | R]))).

on\_right(0, 0, 0, 0, R) :- writereversed(R).

on\_right(LeftMiss, LeftCann, BoatMiss, BoatCann, R) :-

is\_valid(LeftMiss, LeftCann, BoatMiss, BoatCann),

store(LeftMiss, LeftCann, BoatMiss, BoatCann, right),

((BoatMiss + BoatCann > 0, on\_left(LeftMiss, LeftCann, BoatMiss, BoatCann, ['-> drive to the left'| R]));

(BM is BoatMiss - 1, on\_right(LeftMiss, LeftCann, BM, BoatCann, ['missioner left the boat'| R]));

(BC is BoatCann - 1, on\_right(LeftMiss, LeftCann, BoatMiss, BC, ['cannibal left the boat'| R]));

(BM is BoatMiss + 1, on\_right(LeftMiss, LeftCann, BM, BoatCann, ['missioner entered the boat'| R]));

(BC is BoatCann + 1, on\_right(LeftMiss, LeftCann, BoatMiss, BC, ['cannibal entered the boat' | R]))).