Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных технология и систем»

Лабораторная работа №8

**РЕШЕНИЕ ЛОГИЧЕСКИХ ЗАДАЧ МЕТОДОМ ПОИСКА НА ПРОСТРАНСТВЕ СОСТОЯНИЙ**

по дисциплине:

«Функциональное и логическое программирование»

**Отчёт**

Принял преподаватель:

\_\_\_\_\_\_\_\_\_\_ / Михайлов Д.В

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_\_\_\_ / Лехновский А. Д.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**Великий Новгород**

**2021**

1. **Цель и задачи данной лабораторной работы**

Целью лабораторной работы является овладение методологией решения логических задач с применением известных на сегодняшний день стратегий поиска в пространстве состояний.

1. **Задания на лабораторную работу**

**Задача №1:**

Изучить на приведенном примере задачи о волке, козе и капусте работу базовой программы для решения задач методом поиска в глубину.

**Задача №2:**

Решить задачу о миссионерах и каннибалах. Программа для решения задачи должна сохранять все состояния в памяти, а также позволять просматривать эти состояния.

Три миссионера и три каннибала находятся на левом берегу реки. Все хотят перебраться на другой берег. Здесь же небольшая лодка, вмещающая не более двух человек. Если на каком-то берегу каннибалов окажется больше, чем миссионеров, то они съедят миссионеров. Если окажется больше миссионеров, то они обратят каннибалов в свою веру. Найти последовательность поездок, гарантирующих безопасность миссионерам и свободу вероисповедания каннибалам.

1. **Решения поставленных задач**

Для выполнения поставленной задачи был использован Visual Prolog 5.2.

Под ***состоянием*** задачи будем понимать некоторая конфигурация исследуемой динамически меняющейся системы. При этом начальную и целевую конфигурации принято рассматривать в качестве, соответственно, начального и целевого состояний.

***Оператор*** преобразует одно состояние в другое. В общем случае мы будем предполагать, что операторы – это вычисления, преобразующие одни описания состояний в другие

**Задача №1:**

Данная программы запускается с помощью вызова предиката **test\_dfs ()**. Последовательность переходов из начального состояния в целевое строится с помощью правила **solve\_dfs()**, предикат **move()** служит для определения возможности перехода из одного состояния в другое, а поиск достижимого состояния происходит через правило **update()**. Начальное состояние задается через факт **initial\_state ()**, а целевое через **final\_state()**. С помощью правила **legal()** проверяется допустимость состояния, а в правилах **illegal()** описаны недопустимые состояния.

Состояния представляются тройками **wgc(B, L, R)**, где **B** – местонахождение лодки (и фермера) – левый или правый берег, **L** – список находящихся на левом берегу, **R** – список находящихся на правом берегу.

**Начальное состояние:**

wgc («Лодка на левом берегу.», [«Волк»,«Коза»,«Капуста»], [ ]).

**Конечное состояние:**

wgc («Лодка на правом берегу.», [ ], [«Волк»,«Коза»,«Капуста»]).

Код программы для решения задачи приведен в *Приложении 1*.

В результате мы получим некую последовательность переправ, где указан перевозимый в лодке груз или его отсутствие. Нечётные элементы последовательности указывают с чем плыл фермер в лодке на правый берег, а чётные – на левый. Сама последовательность представлена на рис.1

Пример работы программы:

**

*Рисунок №1. Результат работы программы для задачи №1 при вызове test\_dfs(Result).*

**Задача №2:**

В данной программе, под общим состоянием подразумевается следующая конфигурация:

**state**(M, K, P),

где M – количество миссионеров и K – количество каннибалов, в результирующей последовательности, эти значения показывают количество миссионеров и количество каннибалов, которые находятся в данным момент на лодке, P – указание на берег, к которому плывет или уже находится лодка. В графе состояний первые два аргумента вычисляются по отношению к левому берегу, а P - на каком берегу находится лодка.

Начальное состояние (**initial\_state**): state(3, 3, 'Левый берег')

Целевое состояние (**final\_state**): state(0, 0, 'Правый берег')

Правила **new\_state()** определяют возможные переходы из состояния к состоянию для переправы с левого берега на правый и для переправы с правого берега на левый.

Правило **possible\_move()** делает перебор всех возможных вариантов размещения каннибалов и миссионеров на лодке (0-1, 1-0, 1-1, 2-0, 0-2).

Правило **possible\_state()** совершает проверку того, что на берегу не осталось миссионеров больше, чем каннибалов и наоборот.

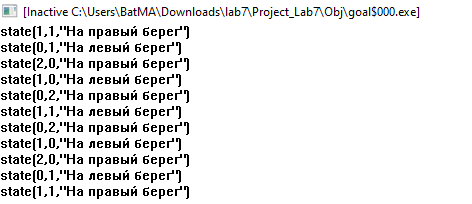
Правилами **move()** определены переходы состояний.

Код программы для решения задачи приведен в *Приложении 2*.

Последовательность переправ, гарантирующая безопасность миссионерам и свободу вероисповедания каннибалам представлена на рис. 2.

Данная программы запускается с помощью вызова **start(Begin, Moves),print\_lst\_col(Moves),nl**.

Пример работы программы:

**

*Рисунок №2. Результат работы программы для задачи №2 при вызове start(Begin, Moves),print\_lst\_col(Moves),nl.*

1. **Вывод**

В данной лабораторной работе, я изучил методологию решения логических задач с применением стратегий поиска в пространстве состояний.

**Приложение 1**

Код программы:

DOMAINS

slst = string\*

wgc = wgc(string,slst,slst)

history = wgc\*

PREDICATES

nondeterm member(wgc,history) % определение принадлежности списку

nondeterm member(string,slst)

initial\_state(wgc)% Начальное состояние

final\_state(wgc)% Конечное состояние

nondeterm move(wgc, string) % Определение возможности перехода

nondeterm update(wgc, string,wgc) % Поиск состояния, достижимого из заданного

update\_boat(string, string)% Изменение местонахождения лодки

nondeterm select(string, slst, slst)% Выбор груза для перевозки

nondeterm precedes(string, string) % Порядок сортировки

nondeterm insert(string, slst, slst) % Вставка элемента

nondeterm update\_banks(string, string, slst, slst, slst, slst) % Перевозка

legal(wgc)%Допустимость состояния

nondeterm illegal(slst)%Недопустимые состояния

nondeterm solve\_dfs(wgc, history, slst) % Построение последовательности состояний

test\_dfs(slst) % Поиск в графе поиском в глубину

CLAUSES

%Определение принадлежности списку

member(Head,[Head|\_]).

member(Element,[\_|Tail]):- member(Element,Tail).

%Начальное и конечное состояние

initial\_state(wgc("Лодка на левом берегу.",["Волк","Коза","Капуста"],[])).

final\_state(wgc("Лодка на правом берегу.",[],["Волк","Коза","Капуста"])).

%Определение возможности перехода из состояния в состояние

move(wgc("Лодка на левом берегу.",L,R),Cargo):- member(Cargo,L).

move(wgc("Лодка на правом берегу.",L,R),Cargo):- member(Cargo,R).

move(wgc(B,L,R),"Без груза").

%Поиск состояния, достижимого из заданного

update(wgc(B,L,R),Cargo,wgc(B1,L1,R1)):-

update\_boat(B,B1),

update\_banks(Cargo,B,L,R,L1,R1).

%Изменение местонахождения лодки

update\_boat("Лодка на левом берегу.","Лодка на правом берегу.").

update\_boat("Лодка на правом берегу.","Лодка на левом берегу.").

%Выбор груза для перевозки

select(X,[X|T],T).

select(X,[H|T],[H|D]):- select(X,T,D).

%Порядок сортировки

precedes("Волк",X).

precedes(X,"Капуста").

%Вставка элемента в список

insert(X,[Y|Ys],[X,Y|Ys]):- precedes(X,Y).

insert(X,[Y|Ys],[Y|Zs]):- precedes(Y,X),insert(X,Ys,Zs).

insert(X,[],[X]).

%Перевозка

%Изменение состава обитателей берегов

%Первый аргумент - перевозимый в лодке груз

%Второй аргумент - текущее местонахождение лодки (левый или правый берег)

%Третий и четвертый аргументы - текущее состав обитателей левого и правого берега

%Пятый и шестой аргументы - новый состав правого и левого берега

%Ситуация, когда фермер переправляется через реку без груза

update\_banks("Без груза",B,L,R,L,R).

%Ситуация перевозка обитателей одного берега на другой

update\_banks(Cargo,"Лодка на левом берегу.",L,R,L1,R1):-

select(Cargo,L,L1),insert(Cargo,R,R1).

update\_banks(Cargo,"Лодка на правом берегу.",L,R,L1,R1):-

select(Cargo,R,R1),insert(Cargo,L,L1).

%Допустимость состояния

legal(wgc("Лодка на левом берегу.",L,R)):-

not(illegal(R)).

legal(wgc("Лодка на правом берегу.",L,R)):-

not(illegal(L)).

illegal(List):-

member("Волк",List),

member("Коза",List).

illegal(List):-

member("Коза",List),

member("Капуста",List).

%Построение посоедовательности состояний

solve\_dfs(State,History,[]):-

final\_state(State).

solve\_dfs(State, History, [Move|Moves]) :-

move(State, Move),

update(State, Move, State1),

legal(State1),

not(member(State1, History)),

solve\_dfs(State1, [State1|History], Moves).

% Поиск в глубину в графе

test\_dfs(Moves):-

initial\_state(State),

solve\_dfs(State, [State], Moves),!.

GOAL

test\_dfs(Result).

**Приложение 2**

Код программы:

DOMAINS

state = state(integer,integer,string)

history = state\*

ilst = integer\*

PREDICATES

start(string,history)

nondeterm solve(history,history)

initial\_state(string,history)

final\_state(history)

nondeterm new\_state(state,state)

nondeterm possible\_move(integer,integer)

nondeterm possible\_state(integer,integer)

move(state,state,state)

nondeterm member(integer,ilst)

nondeterm member(state,history)

nondeterm print\_lst\_col(history)

CLAUSES

%Печать элементов списка столбцом

print\_lst\_col([]).

print\_lst\_col([Head|Tail]):-

write(Head),nl,print\_lst\_col(Tail).

%Запуск программы

start(Begin, Moves) :-

initial\_state(Begin, State),

solve(State, Moves),!.

solve([NewState|ListState], [LastMove|TailMove]) :-

new\_state(NewState, NextState),

not(member(NextState, ListState)),

move(NewState, NextState, LastMove),

solve([NextState, NewState|ListState], TailMove).

%Окончание работы программы

solve(State, []):-

final\_state(State).

%Начальное состояние

initial\_state("Начальное состояние : все находятся на левом берегу", [state(3, 3, "Левый берег")]).

%Целевое состояние

final\_state([state(0, 0, "Правый берег")|\_]).

%Движение с левого берега на правый

move(state(M1, K1, "Левый берег"), state(M2, K2, "Правый берег"), state(M, K, "На правый берег")) :-

M = (M1 - M2),

K = (K1 - K2).

%Движение с правого берега на левый

move(state(M1, K1, "Правый берег"), state(M2, K2, "Левый берег"), state(M, K, "На левый берег")) :-

M = (M2 - M1),

K = (K2 - K1).

%Перебор всех возможных вариантов размещения каннибалов и миссионеров на лодке (0-1,1-0,1-1,2-0,0-2)

possible\_move(M, K):-

member(M, [0, 1, 2]),

member(K, [0, 1, 2]),

(M + K) <= 2,

(M + K) > 0.

/\*Проверка того, что на берегу осталось каннибалов не больше , чем миссионеров и наоборот\*/

possible\_state(X, Y) :-

X = Y;

X = 0;

Y = 0.

%Новое состояние при отправлении на правый берег

new\_state(state(Missioner\_1, Cannibal\_1, "Левый берег"), state(Missioner1\_2, Cannibal\_2, "Правый берег")) :-

possible\_move(Missioner, Cannibal),

Missioner <= Missioner\_1,

Cannibal <= Cannibal\_1,

Missioner1\_2 = (Missioner\_1 - Missioner),

Cannibal\_2 = (Cannibal\_1 - Cannibal),

possible\_state(Missioner1\_2, Cannibal\_2).

%Новое состояние при отправлении на левый берег

new\_state(state(Missioner\_1, Cannibal\_1, "Правый берег"), state(Missioner1\_2, Cannibal\_2, "Левый берег")) :-

possible\_move(Missioner, Cannibal),

Missioner1\_2 = (Missioner\_1 + Missioner),

Cannibal\_2 = (Cannibal\_1 + Cannibal),

Missioner1\_2 <= 3,

Cannibal\_2 <= 3,

MR = (3 - Missioner1\_2),

KR = (3 - Cannibal\_2),

possible\_state(MR, KR).

%Определение принадлежности списку

member(Head,[Head|\_]).

member(Head,[\_|Tail]):- member(Head,Tail).

GOAL

start(Begin, Moves),print\_lst\_col(Moves),nl.