Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №8 по дисциплине "Проектирование экспертных систем"

Тема Алгоритм обратного логичесого вывода на обобщенных правилах продуции

Студент Варламова Е. А.

Группа ИУ7-33М

Оценка (баллы) _____

Преподаватели Русакова З.Н.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – реализовать алгоритм обратного логичесого вывода на обобщенных правилах продуции.

Для достижения поставленной цели потребуется:

— описать используемые структуры данных;

— описать алгоритм обратного логичесого вывода на обобщенных правилах продуции;

— привести реализацию алгоритма;

— привести примеры работы алгоритма.

1 | Используемые структуры данных

Разработаем класс переменной. Поля класса:

— имя;

— тип (переменная).

Разработаем класс константы. Поля класса:

— значение;

— тип (не переменная).

Разработаем класс вершины (атома). Поля класса:

— имя;

— список термов (терм – константа или переменная).

Разработаем класс подстановок. Поля класса:

— словарь переменных (ключ – имя переменной, значение – связанная переменная или константа).

Разработаем класс правила. Поля класса:

— список входных вершин;

— целевая вершина;

— номер правила;

— флаг правила – открыто/закрыто.

2 | Алгоритм обратного логичесого вывода на обобщенных правилах продуции

2.1 Алгоритм поиска по графу в глубину от цели

Вход:

— доказанные вершины;

— целевая вершина.

Выход:

— информация о решении.

2.2 Основной метода поиска

Поместить все доказанные вершины в список закрытых вершин. Добавить целевую вершину в стек открытых вершин. Поа флаги решения истинны, выполняем:

— вызвать метод потоми, который возвращает количество закрытых правил;

— если флаг решение найдено, то выход с сообщением об успешном поиске;

— если количество закрытых правил равно 0 и стек содержит не более одной вершины, то выход с сообщением о неудачном поиске;

— если количество закрытых правил равно 0, то вызывать метод обратного хода (backtracking).

2.3 Метод потомки

В цикле по списку правил выполнить.

— Если правило не открыто, то пропустить его.

— Если выходная вершина текущего правила унифицируется с текущей вершиной стека, то отметить правило как посещённое, добавить его в список открытых правил и записать в стек все вершины левой части правила (записав в стек состояния таблицы подстановок), которые не закрыты (при этом при унификации с закрытыми вершинами переменные получают значения). Если таких (закрытых) нет, то вызывать метод маркирования. Увеличить количество закрытых правил.

Вернуть количество закрытых правил.

2.4 Метод маркирования (label)

— Взять последнее правило из списка открытых правил (удалив его) и поместить его в список закрытых.

— Взять верхний элемент N из стека открытых вершин (удалив его) и поместить его в список закрытых вершин.

— Если вершина N унифицируется с целевой, то изменить флаг наличия решения.

— Получить последний элемент списка открытых правил M (без удаления) и верхний элемент стека открытых вершин K (без удаления). Если целевая вершина правила M унифицируется с вершиной K, то вернуться к первому шагу (так как в таком случае очевидно, что в стеке открытых вершин нет вершин из левой части этого правила, а значит оно доказано и может быть перемещено в закрытые).

2.5 Метод обратного хода (backtracking)

— Взять верхний элемент стека открытых вершин (с удалением) и поместить его в список запрещенных вершин.

— Взять последний элемент списка открытых правил (с удалением) и поместить его список запрещённых правил.

— Удалить из стека открытых вершин все вершины, которые находятся в левой части запрещённого правила.

— Удалить из стека таблицу подстановок и использовать ту, которая на вершине стека.

2.6 Метод унификации

Вход:

— 2 атома;

— подстановка.

Выход:

— информация о возможности унификации.

Унифиация выполняется:

1. Если термы онстанты, то они унифицируемы если совпадают.
2. Если в первом атоме терм переменная, а во втором онстанта, то они унифицируемы и переменная получает значение онстанты.
3. Если терм в первом атоме переменная и во втором тоже, то они унифицируемы и становятся связанными.
4. Если в первом атоме терм переменная, а во втором фунция от переменных, то они унифицируемы и вместо переменной подставляется фунция (x и f(x) не унифицируемы).

3 | Реализация

Листинг 3.1: Структуры данных

```
class Label(Enum):
    OPEN = 0
    CLOSE = 1
    FORBIDDEN = -1
    VIEWED = 2



class Node:
    def __init__(self, name, terminals):
        self.name = name
        self.terminals = terminals

    def __str__(self):
        strterms = ""
        for term in self.terminals:
            strterms += str(term) + ", "
        return self.name + '(' + strterms.strip(", ") + ')'

    def __repr__(self):
        return self.__str__()


class Constant:  #
    def __init__(self, value):
        self.value = value  #
        self.variable = False  #

    def __str__(self):
        return self.value

    def __repr__(self):
        return self.__str__()


class Variable:
    def __init__(self, name):
        self.name = name
```

```
39          self.variable = True
40
41      def __str__(self):
42          return self.name
43
44      def __repr__(self):
45          return self.__str__()
46
47
48  class Table:
49      def __init__(self):
50          self.variables = dict()
51          self.links = dict()
52
53      def reset(self, other):
54          self.variables = other.variables
55          self.links = other.links
56
57      def val(self, var):
58          return self.variables[var.name]
59
60      def var_links(self, var):
61          return self.links[self.variables[var.name]]
62
63      def __str__(self):
64          res = ""
65          for const in self.links.keys():
66              res += str(self.links[const]) + ": " + str(const) + "\n"
67          return res
68
69  class Rule:
70      def __init__(self, number: int, out_node: Node, node_arr: List[Node],
        label=Label.OPEN):
71          self.number = number
72          self.out_node = out_node
73          self.node_arr = node_arr  #
              ,
74          self.label = label  #                  /                /

75      def __str__(self):
76
77          res = str(self.node_arr) + " -> " + str(self.out_node)
78
79          return res
```

Листинг 3.2: класс поиска

```
1 from items import *
2 from stack import Stack
3 import copy
```

```
4
5  class Search:
6      def __init__(self, rule_arr: [Rule]):
7          self.rule_arr = rule_arr  #
8          self.open_node_st = Stack()
9          self.open_rule_lst = []
10         self.close_node_lst = []
11         self.close_rule_lst = []
12         self.prohibited_node_lst = []
13         self.prohibited_rule_lst = []
14         self.tables = Stack()
15
16         self.goal_node = None
17         self.solution_flg = 1
18         self.no_solution_flg = 1
19         self.no_label = 1
20         self.table = Table()
21
22     def run(self, goal_node: Node, in_node_arr: [Node]):
23         self.goal_node = goal_node
24         self.open_node_st.push(goal_node)
25         self.tables.push(copy.deepcopy(self.table))
26         self.close_node_lst = in_node_arr
27
28         while self.solution_flg and self.no_solution_flg:
29             rule_cnt = self.child_search()
30
31             # solution was found
32             if self.solution_flg == 0:
33                 print("Solution was found")
34                 return
35
36             if rule_cnt == 0 and self.open_node_st.length() < 2:
37                 self.no_solution_flg = 0
38                 print("Solution was not found")
39             elif rule_cnt == 0:
40                 print("Backtracking process is going to be launched")
41                 self.backtracking()
42
43     def child_search(self):
44         cnt_rules = 0
45
46         for rule in self.rule_arr:
47             print(f'[Rule {rule.number}] Current rule {rule}')
48
49             current_node = self.open_node_st.peek()
50             print(f'[Node {current_node}]  Current node')
51
52             if rule.label != Label.OPEN: #
53                 print(f'[Rule {rule.number}] was already processed')
```

```
54                print('-' * 128 + '\n')
55                continue
56
57            if unification(self.table, rule.out_node, current_node):
58                print(f'[Rule {rule.number}] has out node that equals goal
                      one')
59
60                rule.label = Label.VIEWED
61                self.open_rule_lst.append(rule)
62                is_new_goal_added = self.add_new_goal(rule.node_arr)
63                if not is_new_goal_added:
64                    print("Label process is going to be launched")
65                    self.label()
66
67                cnt_rules += 1
68                self.print_info(rule)
69                break
70
71            if self.is_prohibited_node_exist(rule.node_arr):
72                self.prohibited_rule_lst.append(rule)
73                rule.label = Label.FORBIDDEN
74
75                self.print_info(rule)
76                continue
77
78            #self.print_info(rule)
79
80        return cnt_rules
81
82    def get_fact(self, node, table):
83        new_terms = []
84        for term in node.terminals:
85            if term.variable:
86                if type(table.variables[term.name]) is str:
87                    print("Error")
88                else:
89                    new_terms.append(table.variables[term.name])
90            else:
91                new_terms.append(term)
92        return Node(node.name, new_terms)
93
94    def label(self):
95        while True:
96            rule = self.open_rule_lst.pop()
97            self.close_rule_lst.append(rule)
98
99            node = self.open_node_st.pop()
100           self.tables.pop()
101           fact = self.get_fact(node, self.table)
102           self.close_node_lst.append(fact)
```

```
103
104             print(f'[Labelling] Rule {rule} was added to close rules')
105             print(f'[Labelling] Node {fact} was added to close nodes')
106
107             if unification(self.table, fact, self.goal_node):
108                 self.solution_flg = 0
109                 return
110
111             current_node = self.open_node_st.peek()
112             current_rule = self.open_rule_lst[-1]
113             if not unification(self.table, current_rule.out_node,
                    current_node):
114                 return
115
116     def backtracking(self):
117         current_goal = self.open_node_st.pop()
118
119         table_prev = self.tables.pop()
120         self.table = self.tables.peek()
121         rule = self.open_rule_lst.pop()
122
123         current_goal.flag = Label.FORBIDDEN
124         self.prohibited_node_lst.append(current_goal)
125
126         rule.label = Label.FORBIDDEN
127         self.prohibited_rule_lst.append(rule)
128         self.print_info(rule)
129         print(f'[Backtrack] Rule {rule} was added to prohibited rules')
130         print(f'[Backtrack] Node {current_goal} was added to prohibited
                nodes')
131         print(f'[Backtrack] Table {table_prev.variables} was changed to {
                self.table.variables}')
132
133         for node in rule.node_arr:
134             print(f'[Backtrack] Node {node} should be removed from opened
                    nodes')
135             self.open_node_st.remove_element(node)
136         print()
137
138     def add_new_goal(self, node_arr: [Node]):
139         new_goal_flg = False
140         for node in node_arr[::-1]:
141             found = False
142             for node_closed in self.close_node_lst:
143                 if unification(self.table, node, node_closed):
144                     found = True
145             if not found:
146                 self.open_node_st.push(node)
147                 copy_table = copy.deepcopy(self.table)
148                 self.tables.push(copy_table)
```

```
149                 new_goal_flg = True
150         return new_goal_flg
151 
152     def is_prohibited_node_exist(self, node_arr: [Node]):
153         for node in node_arr:
154             if node in self.prohibited_node_lst:
155                 return True
156         return False
```

4 | Пример работы

4.1 Программная реализация задачи

```
c_N = Constant('N')
c_M1 = Constant('M1')
c_W = Constant('W')
c_A1 = Constant('A1')

v_x = Variable("x")
v_y = Variable("y")
v_z = Variable("z")
v_x1 = Variable("x1")
v_x2 = Variable("x2")
v_x3 = Variable("x3")
v_x4 = Variable("x4")
v_x5 = Variable("x5")
v_x6 = Variable("x6")
v_x7 = Variable("x7")
v_x8 = Variable("x8")
v_x9 = Variable("x9")
v_x10 = Variable("x10")
v_x11 = Variable("x11")
v_x12 = Variable("x12")


rule_arr = [
    Rule(1, Node("C", [v_x]), [Node("W1", [v_y]),
                               Node("A", [v_x]),
                               Node("S", [v_x, v_y, v_z]),
                               Node("H", [v_z])]),
    Rule(5, Node("W1", [v_x4]), [Node("U", [v_x4])]),
```

```
    Rule(2, Node("S", [c_W, v_x1, c_N]), [Node("M", [v_x1]),
                                          Node("O", [c_N, v_x1])]),
    Rule(3, Node("W1", [v_x2]), [Node("M", [v_x2])]),
    Rule(4, Node("H", [v_x3]), [Node("E", [v_x3, c_A1])]),

]
facts = [Node("O", [c_N, c_M1]),
         Node("M", [c_M1]),
         Node("A", [c_W]),
         Node("E", [c_N, c_A1])]
Search(rule_arr).run(Node("C", [c_W]), facts)
```

Результат:

```
[Rule 1] Current rule [W1(y), A(x), S(x, y, z), H(z)] -> C(x)
[Node C(W)]  Current node
[Rule 1] has out node that equals goal one
[Rule 1] list of opened nodes:     C(W) H(z) S(x, y, z) W1(y)
[Rule 1] list of closed nodes:     O(N, M1) M(M1) A(W) E(N, A1)
[Rule 1] list of prohibited nodes:
[Rule 1] list of opened rules:     1
[Rule 1] list of closed rules:
[Rule 1] list of prohibited rules:
--------------------------------------------------------------------------------

Current table: {'x': W}
[Rule 1] Current rule [W1(y), A(x), S(x, y, z), H(z)] -> C(x)
[Node W1(y)]  Current node
[Rule 1] was already processed
--------------------------------------------------------------------------------

[Rule 5] Current rule [U(x4)] -> W1(x4)
[Node W1(y)]  Current node
[Rule 5] has out node that equals goal one
[Rule 5] list of opened nodes:     C(W) H(z) S(x, y, z) W1(y) U(x4)
[Rule 5] list of closed nodes:     O(N, M1) M(M1) A(W) E(N, A1)
[Rule 5] list of prohibited nodes:
[Rule 5] list of opened rules:     1 5
[Rule 5] list of closed rules:
[Rule 5] list of prohibited rules:
--------------------------------------------------------------------------------

Current table: {'x': W, 'x4': 'y', 'y': 'x4'}
[Rule 1] Current rule [W1(y), A(x), S(x, y, z), H(z)] -> C(x)
[Node U(x4)]  Current node
[Rule 1] was already processed
--------------------------------------------------------------------------------

[Rule 5] Current rule [U(x4)] -> W1(x4)
```

```
[Node U(x4)]  Current node
[Rule 5] was already processed
--------------------------------------------------------------------------------

[Rule 2] Current rule [M(x1), O(N, x1)] -> S(W, x1, N)
[Node U(x4)]  Current node
[Rule 3] Current rule [M(x2)] -> W1(x2)
[Node U(x4)]  Current node
[Rule 4] Current rule [E(x3, A1)] -> H(x3)
[Node U(x4)]  Current node
Backtracking process is going to be launched
[Rule 5] list of opened nodes:     C(W) H(z) S(x, y, z) W1(y)
[Rule 5] list of closed nodes:     O(N, M1) M(M1) A(W) E(N, A1)
[Rule 5] list of prohibited nodes: U(x4)
[Rule 5] list of opened rules:     1
[Rule 5] list of closed rules:
[Rule 5] list of prohibited rules: 5
--------------------------------------------------------------------------------

Current table: {'x': W}
[Backtrack] Rule [U(x4)] -> W1(x4) was added to prohibited rules
[Backtrack] Node U(x4) was added to prohibited nodes
[Backtrack] Table {'x': W, 'x4': 'y', 'y': 'x4'} was changed to {'x': W}
[Backtrack] Node U(x4) should be removed from opened nodes

[Rule 1] Current rule [W1(y), A(x), S(x, y, z), H(z)] -> C(x)
[Node W1(y)]  Current node
[Rule 1] was already processed
--------------------------------------------------------------------------------

[Rule 5] Current rule [U(x4)] -> W1(x4)
[Node W1(y)]  Current node
[Rule 5] was already processed
--------------------------------------------------------------------------------

[Rule 2] Current rule [M(x1), O(N, x1)] -> S(W, x1, N)
[Node W1(y)]  Current node
[Rule 3] Current rule [M(x2)] -> W1(x2)
```

```
[Node W1(y)]  Current node
[Rule 3] has out node that equals goal one
Label process is going to be launched
[Labelling] Rule [M(x2)] -> W1(x2) was added to close rules
[Labelling] Node W1(M1) was added to close nodes
[Rule 3] list of opened nodes:     C(W) H(z) S(x, y, z)
[Rule 3] list of closed nodes:     O(N, M1) M(M1) A(W) E(N, A1) W1(M1)
[Rule 3] list of prohibited nodes: U(x4)
[Rule 3] list of opened rules:     1
[Rule 3] list of closed rules:     3
[Rule 3] list of prohibited rules: 5
--------------------------------------------------------------------------------

Current table: {'x': W, 'x2': M1, 'y': M1}
[Rule 1] Current rule [W1(y), A(x), S(x, y, z), H(z)] -> C(x)
[Node S(x, y, z)]  Current node
[Rule 1] was already processed
--------------------------------------------------------------------------------

[Rule 5] Current rule [U(x4)] -> W1(x4)
[Node S(x, y, z)]  Current node
[Rule 5] was already processed
--------------------------------------------------------------------------------

[Rule 2] Current rule [M(x1), O(N, x1)] -> S(W, x1, N)
[Node S(x, y, z)]  Current node
[Rule 2] has out node that equals goal one
Label process is going to be launched
[Labelling] Rule [M(x1), O(N, x1)] -> S(W, x1, N) was added to close rules
[Labelling] Node S(W, M1, N) was added to close nodes
[Rule 2] list of opened nodes:     C(W) H(z)
[Rule 2] list of closed nodes:     O(N, M1) M(M1) A(W) E(N, A1) W1(M1) S(W, M
[Rule 2] list of prohibited nodes: U(x4)
[Rule 2] list of opened rules:     1
[Rule 2] list of closed rules:     3 2
[Rule 2] list of prohibited rules: 5
--------------------------------------------------------------------------------
```

```
Current table: {'x': W, 'x2': M1, 'y': M1, 'x1': M1, 'z': N}
[Rule 1] Current rule [W1(y), A(x), S(x, y, z), H(z)] -> C(x)
[Node H(z)]  Current node
[Rule 1] was already processed
--------------------------------------------------------------------------------

[Rule 5] Current rule [U(x4)] -> W1(x4)
[Node H(z)]  Current node
[Rule 5] was already processed
--------------------------------------------------------------------------------

[Rule 2] Current rule [M(x1), O(N, x1)] -> S(W, x1, N)
[Node H(z)]  Current node
[Rule 2] was already processed
--------------------------------------------------------------------------------

[Rule 3] Current rule [M(x2)] -> W1(x2)
[Node H(z)]  Current node
[Rule 3] was already processed
--------------------------------------------------------------------------------

[Rule 4] Current rule [E(x3, A1)] -> H(x3)
[Node H(z)]  Current node
[Rule 4] has out node that equals goal one
Label process is going to be launched
[Labelling] Rule [E(x3, A1)] -> H(x3) was added to close rules
[Labelling] Node H(N) was added to close nodes
[Labelling] Rule [W1(y), A(x), S(x, y, z), H(z)] -> C(x) was added to close r
[Labelling] Node C(W) was added to close nodes
[Rule 4] list of opened nodes:
[Rule 4] list of closed nodes:     O(N, M1) M(M1) A(W) E(N, A1) W1(M1) S(W, N
[Rule 4] list of prohibited nodes: U(x4)
[Rule 4] list of opened rules:
[Rule 4] list of closed rules:     3 2 4 1
[Rule 4] list of prohibited rules: 5
--------------------------------------------------------------------------------

Current table: {'x': W, 'x2': M1, 'y': M1, 'x1': M1, 'z': N, 'x3': N}
```

```
Solution was found
```

ЗАКЛЮЧЕНИЕ

В результаты работы цель была достигнута.
Для достижения поставленной цели потребовалось:

— описать используемые структуры данных;

— описать алгоритм обратного логичесого вывода на обобщенных правилах продуции;

— привести реализацию алгоритма;

— привести примеры работы алгоритма.