**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 №6 по дисциплине "Проектирование экспертных систем"

**Тема** Метод резолюции и унификации

**Студент** Варламова Е. А.

**Группа** ИУ7-33М

**Оценка (баллы)** _____

**Преподаватели** Русакова З.Н.

Москва — 2024 г.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Цель работы – реализовать метод резолюции и унификации.
Для достижения поставленной цели потребуется:

— описать идею метода резолюций;

— описать используемые структуры данных;

— описать алгоритм метода резолюций;

— описать алгоритм метода унификации;

— привести реализацию алгоритма;

— привести примеры работы алгоритма.

# 1 | Идея метода резолюций

Доказательство теорем сводится к доказательству того, что некоторая формула $G$ (гипотеза теоремы) является логическим следствием множества формул $F_1, \ldots, F_k$. То есть сама теорема может быть сформулирована следующим образом: «если $F_1, \ldots, F_k$ истинны, то истинна и $G$ ».

Для доказательства того, что формула $G$ является логическим следствием множества формул $F_1, \ldots, F_k$, метод резолюций применяется следующим образом. Сначала составляется множество формул $\{F_1, \ldots, F_k, \neg G\}$. Затем каждая из этих формул приводится к КНФ (конъюнкция дизъюнктов) и в полученных формулах зачеркиваются знаки конъюнкции. Получается множество дизъюнктов $S$. И, наконец, ищется вывод пустого дизъюнкта из $S$. Если пустой дизъюнкт выводим из $S$, то формула $G$ является логическим следствием формул $F_1, \ldots, F_k$. Если из $S$ нельзя вывести, то $G$ не является логическим следствием формул $F_1, \ldots, F_k$.

# 2 | Используемые структуры данных

Разработаем класс переменной. Поля класса:

— имя;

— тип (переменная).

Разработаем класс константы. Поля класса:

— значение;

— тип (не переменная).

Разработаем класс атома. Поля класса:

— имя;

— список термов (терм – константа или переменная);

— знак.

Разработаем класс подстановок. Поля класса:

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

— словарь ссылок (ключ – значение константы, значение – список переменных, имющих знчение этой константы).

Разработаем класс выражения (дизъюнкта). Поля класса:

— список атомов.

# 3 | Метод резолюций

Вход:

— список аксиом (список выражений);

— отрицание заключения .

Выход:

— информация о решении.

Атом – это один из элементов, соединённых дизъюнкцией.
Таким образом, алгоритм метода резолюций выглядит следующим образом:

1. Принять отрицание залючения.
2. Привести все формулы посыло или асиом и отрацания цели КНФ.
3. С помощью алгоритма полного перебора определить: если существует пара дизъюнтов, содержащая онтрарные атомы (то есть унифицируемые атомы с противоположными знаками), то эти дизъюнты объединяются с ударением онтрарной пары и получаем новый дизъюнт (резольвенту), оторая является их логичесим следствием и добавляется исходному множеству дизъюнтов.
4. Если на аом-то шаге мы получим 2 дизъюнта, в отором один атом с разными знаами, а резольвента будет пустым дизъюнтом или ложь, то в этом случае показано противоречие, а, следовательно, залючение является логическим следствием предпосылок.
5. Если на каком-то шаге после просмотра всех пар дизъюнктов, их множество не пополнится, то доказано, что залючение не является логическим следствием предпосылок.

# 4 | 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).

# 5 | Реализация

Листинг 5.1: Структуры данных

```
class Atom:
    def __init__(self, name, terminals, sign):
        self.name = name
        self.terminals = terminals
        self.sign = sign

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

    def __repr__(self):
        a = self.__str__()
        return f"{self.sign}{a}"

    def print(self):
        t = ""
        if self.sign == -1:
            t = "-"
        a = self.__str__()
        print(f"{t}{a}", end = " ")

    def __eq__(self, other):
        if isinstance(other, Atom):
            return self.__hash__() == other.__hash__() # unification(table,
                self, other)
        return False

    def __hash__(self):
        objs = []
        for val in self.terminals:
            if type(val) is Constant:
                objs.append(val.value)
            else:
                objs.append(val.name)
        return hash((self.name, self.sign, *objs))

```

```
38
39 class Constant:
40     def __init__(self, value):
41         self.value = value
42         self.variable = False
43
44     def __str__(self):
45         return self.value
46
47     def __repr__(self):
48         return self.__str__()
49     def __hash__(self):
50         return hash((self.value))
51
52 class Variable:
53     def __init__(self, name):
54         self.name = name
55         self.variable = True
56
57     def __str__(self):
58         return self.name
59
60     def __repr__(self):
61         return self.__str__()
62     def __hash__(self):
63         return hash((self.name, self.variable))
64
65
66 class Table:
67     def __init__(self):
68         self.variables = dict()
69         self.links = dict()
70
71     def reset(self, other):
72         self.variables = other.variables
73         self.links = other.links
74
75     def val(self, var):
76         return self.variables[var.name]
77
78     def var_links(self, var):
79         return self.links[self.variables[var.name]]
80
81     def __str__(self):
82         res = ""
83         for const in self.links.keys():
84             res += str(self.links[const]) + ": " + str(const) + "\n"
85         return res
86
87 class Clause:
```

```
88      def __init__(self, atoms: list):
89          self.atoms = atoms
90          self.seen = []
91      def add_seen(self, seen_id):
92          self.seen.append(seen_id)
93      def get_seen(self):
94          return self.seen
95      def get_atoms(self):
96          return self.atoms
97      def print(self):
98          l = len(self.atoms)
99          print("(", end = "")
100         for i in range(l):
101             self.atoms[i].print()
102             if i != l - 1:
103                 print("+", end = " ")
104         print(")", end = "")
105     def __eq__(self, other):
106         if isinstance(other, Clause):
107             return set(self.atoms) == set(other.atoms)
108         return False
```

Листинг 5.2: Класс метода резолюций

```
1  def unification(table, p1, p2):
2      if p1.name != p2.name:
3          #print('                                    ')
4          return False
5
6      if len(p1.terminals) != len(p2.terminals):
7          #print('                                    ')
8          return False
9
10     original = copy.deepcopy(table)
11     for t1, t2 in zip(p1.terminals, p2.terminals):
12         if t1.variable:
13             if t2.variable:
14                 y = True
15
16                 if t1.name not in table.variables and t2.name not in table.
                       variables:
17                     table.variables[t1.name] = t2.name
18                     table.variables[t2.name] = t1.name
19
20                 elif t1.name not in table.variables:
21                     table.variables[t1.name] = table.variables[t2.name]
22
23                 elif t2.name not in table.variables:
24                     table.variables[t2.name] = table.variables[t1.name]
25
```

```
26                elif set([t1.name, table.variables[t1.name]]) != set([table.
                      variables[t2.name], t2.name]):
27                    y = False
28
29                if y == False:
30                    #print("                    ", t1.name, "
                                                  ", t2.name, ": ", table.val(t1),
31                    #      " != ", table.val(t2), sep='')
32                    table.reset(original)
33                    return False
34
35            else:
36                y = True
37                if t1.name in table.variables and type(table.variables[t1.
                      name]) is not str:
38                    if table.variables[t1.name].value != t2.value:
39                        y = False
40
41                if t1.name not in table.variables:
42                    table.variables[t1.name] = t2
43
44                if type(table.variables[t1.name]) is str:
45                    k = table.variables[t1.name]
46                    table.variables[t1.name] = t2
47                    table.variables[k] = t2
48                    table.links[t2.value] = {k}
49
50                if t2.value not in table.links:
51                    table.links[t2.value] = {t1.name}
52                else:
53                    table.links[t2.value].add(t1.name)
54
55                if y == False:
56                    #print("
                                                : ", t1.name, " = ", table.val(t1),
57                    #      "                    ", t2.value)
58                    table.reset(original)
59                    return False
60        else:
61            if t2.variable:
62                y = True
63
64                if t2.name in table.variables and type(table.variables[t2.
                      name]) is not str:
65                    if table.variables[t2.name].value != t1.value:
66                        y = False
67
68                if t2.name not in table.variables:
```

```
69                      table.variables[t2.name] = t1
70
71                  if type(table.variables[t2.name]) is str:
72                      k = table.variables[t2.name]
73                      table.variables[t2.name] = t1
74                      table.variables[k] = t1
75                      table.links[t1.value] = {k}
76
77                  if t1.value not in table.links:
78                      table.links[t1.value] = {t2.name}
79                  else:
80                      table.links[t1.value].add(t2.name)
81
82                  if y == False:
83                      #print("
                                                    :", t2.name, "=", table.val(t2),
84                      #       "                      ", t1.value)
85                      table.reset(original)
86                      return False
87              else:
88                  if t1.value != t2.value:
89                      #print("
                                                        :", t1.value, "!=", t2.
                            value)
90                      table.reset(original)
91                      return False
92      return True
93
94  class KNF:
95      def __init__(self, clauses: list, label="KNF: "):
96          self.clauses = clauses
97          self.label = label
98      def print(self):
99          print(self.label, "")
100         for c in self.clauses:
101             c.print()
102         print()
103 var_count = 5
104 class Resolution:
105     @staticmethod
106     def resolve(c1: Clause, c2: Clause) -> Clause:
107         new_atoms = []
108         n_c1 = c1.atoms
109         n_c2 = c2.atoms
110         change = False
111         for atom1 in c1.atoms:
112             for atom2 in c2.atoms:
113                 table = Table()
114                 if unification(table, atom1, atom2) and atom1.sign != atom2.
```

```
                    sign:
115                  n_c1_ = []
116                  for a in n_c1:
117                      if a != atom1:
118                          new_terms = []
119                          for term in a.terminals:
120                              if term.variable:
121                                  if type(table.variables[term.name]) is
                                        str:
122                                      new_terms.append(Variable(table.
                                            variables[term.name]))
123                                  else:
124                                      new_terms.append(table.variables[
                                            term.name])
125                              else:
126                                  new_terms.append(term)
127                          n_c1_.append(Atom(a.name, new_terms, a.sign))
128                  n_c2_ = []
129                  for a in n_c2:
130                      if a != atom2:
131                          new_terms = []
132                          for term in a.terminals:
133                              if term.variable:
134                                  if type(table.variables[term.name]) is
                                        not str:
135                                      new_terms.append(table.variables[
                                            term.name])
136                                  else:
137                                      new_terms.append(Variable(term.name)
                                            )
138                              else:
139                                  new_terms.append(term)
140                          n_c1_.append(Atom(a.name, new_terms, a.sign))
141                  global var_count
142
143                  atoms = list(set(n_c1_ + n_c2_))
144
145                  replaced = {}
146                  clause = []
147                  for a in atoms:
148                      new_terms = []
149                      for term in a.terminals:
150                          if term.variable:
151                              if term.name not in replaced:
152                                  replaced[term.name] = Variable(f"x{
                                        var_count}")
153                                  var_count += 1
154                              new_terms.append(replaced[term.name])
155                          else:
156                              new_terms.append(term)
```

```
157                         clause.append(Atom(a.name, new_terms, a.sign))
158
159                     return Clause(clause)
160         return None
161
162     @staticmethod
163     def run_full(axioms: list, target: list):
164         clauses = [Clause(list(set(a.get_atoms()))) for a in target + axioms
                ]
165
166         new_clauses = [Clause(list(set(a.get_atoms()))) for a in target +
                axioms]
167         KNF(new_clauses, "                    : ").print()
168         iters = 200
169         while True and iters:
170             found = False
171
172             for i in range(len(clauses)):
173                 if found:
174                     break
175                 for j in range(i + 1, len(clauses)):
176                     if i in clauses[j].get_seen() or j in clauses[i].
                            get_seen():
177                         continue
178                     resolvent = Resolution.resolve(clauses[i], clauses[j])
179                     if not resolvent:
180                         continue
181                     if not resolvent.atoms:
182                         KNF([clauses[i]], "                              : ")
                                .print()
183                         KNF([clauses[j]], "                              : ")
                                .print()
184                         KNF([resolvent], "                         : ").print()
185                         print("
                                                  ")
186                         return
187                     if resolvent in clauses:
188                         continue
189                     KNF([clauses[i]], "                                 : ").
                            print()
190                     KNF([clauses[j]], "                                 : ").
                            print()
191                     KNF([resolvent], "                          : ").print()
192                     clauses[j].add_seen(i)
193                     clauses[i].add_seen(j)
194                     new_clauses = clauses + [resolvent]

                        # clauses[:i] + clauses[i + 1:j] + clauses[j + 1:] +
                            [resolvent] #
195                     found = True
```

```
196                 break
197             print("————————————————————————————————————————")
198             KNF(new_clauses, "                          : ").print()
199             if new_clauses == clauses:
200                 print("                                                  .")
201                 return
202             clauses = new_clauses
203             iters -= 1
```

# 6 | Пример работы

## 6.1 Задача

— Петя, Коля, Лена члены луба.

— Каждый член луба или лыжни или альпинист или является и тем и другим.

— Нет альпинистов оторые любят дождь.

— Все лыжнии любят снег.

— Лена не любит то, что любит Петя и любит то, что Петя не любит.

— Петя любит дождь

— Петя любит снег

Доазать, есть ли в клубе человек, являющийся альпинистом и не являющийся лыжнием.

## 6.2 Математическое представление задачи

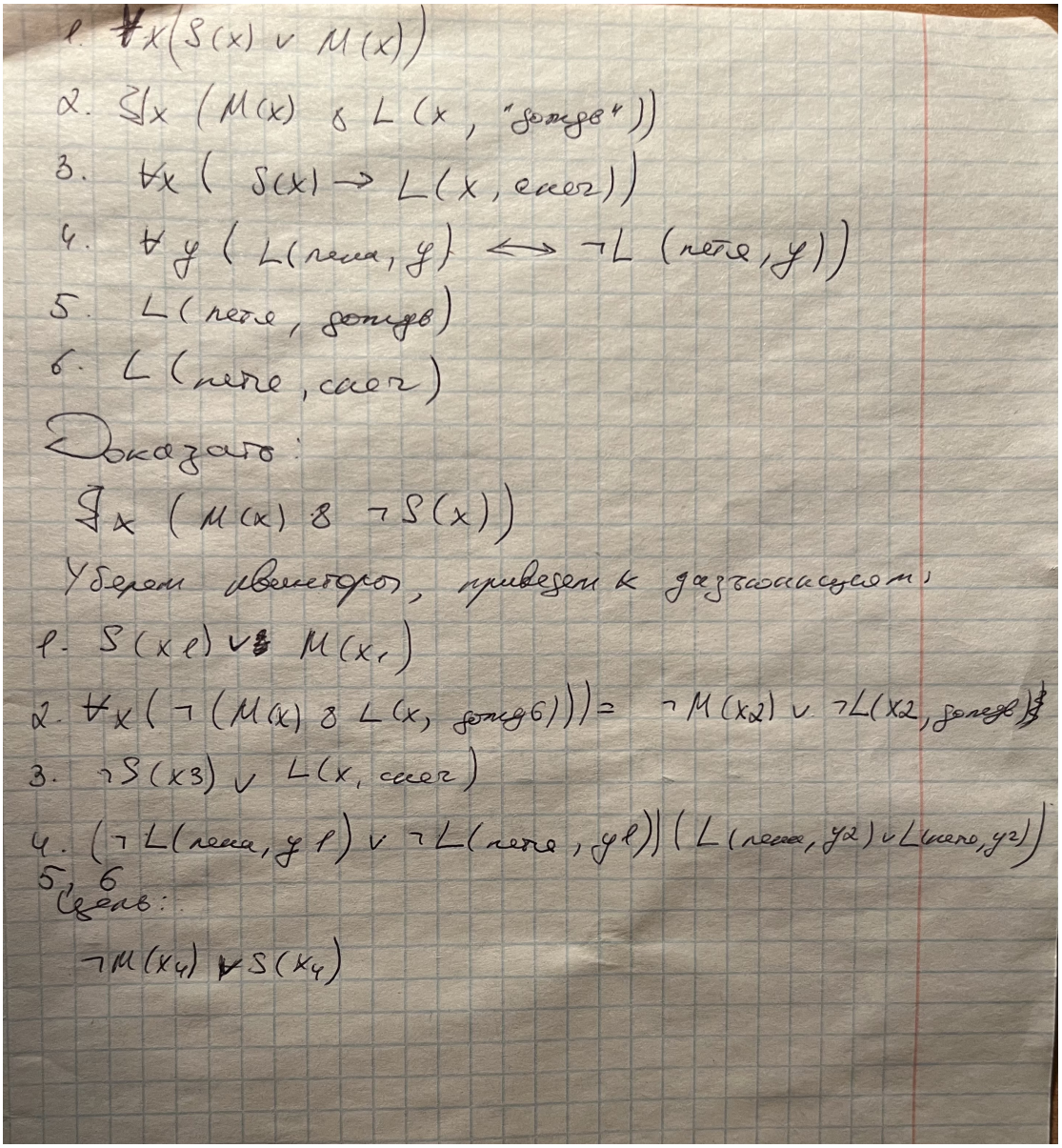


Рис. 6.1: Математическое представление задачи

## 6.3 Программное описание задачи

Листинг 6.1: Программное описание задачи

```
1
2     c_lena = Constant('LENA')
3     c_rain = Constant('RAIN')
4     c_snow = Constant('SNOW')
5     c_petya = Constant('PETYA')
6
7     v_x1 = Variable("x1")
8     v_x2 = Variable("x2")
9     v_x3 = Variable("x3")
10    v_x4 = Variable("x4")
11    v_y1 = Variable("y1")
12    v_y2 = Variable("y2")
13
14
15    axioms = [
16        Clause([Atom("L", [c_petya, c_rain], 1)]),
17        Clause([Atom("L", [c_petya, c_snow], 1)]),
18        Clause([Atom("S", [v_x1], 1), Atom("M", [v_x1], 1)]),
19        Clause([Atom("M", [v_x2], -1), Atom("L", [v_x2, c_rain], -1)]),
20        Clause([Atom("S", [v_x3], -1), Atom("L", [v_x3, c_snow], 1)]),
21        Clause([Atom("L", [c_lena, v_y1], -1), Atom("L", [c_petya, v_y1],
              -1)]),
22        Clause([Atom("L", [c_lena, v_y2], 1), Atom("L", [c_petya, v_y2], 1)
              ]),
23    ]
24
25
26    target = [Clause([Atom("M", [v_x4], -1), Atom("S", [v_x4], 1)])]
```

## 6.4 Результат работы программы

```
дизъюнкты:
(-M(x4) + S(x4) )
(L(PETYA, RAIN) )
(L(PETYA, SNOW) )
(M(x1) + S(x1) )
(-L(x2, RAIN) + -M(x2) )
(-S(x3) + L(x3, SNOW) )
(-L(PETYA, y1) + -L(LENA, y1) )
(L(LENA, y2) + L(PETYA, y2) )
```

```
первый дизъюнкт:
(-M(x4) + S(x4) )

второй дизъюнкт:
(M(x1) + S(x1) )

резольвента:
(S(x5) )

-------------------------------------
дизъюнкты:
(-M(x4) + S(x4) )
(L(PETYA, RAIN) )
(L(PETYA, SNOW) )
(M(x1) + S(x1) )
(-L(x2, RAIN) + -M(x2) )
(-S(x3) + L(x3, SNOW) )
(-L(PETYA, y1) + -L(LENA, y1) )
(L(LENA, y2) + L(PETYA, y2) )
(S(x5) )

первый дизъюнкт:
(-M(x4) + S(x4) )

второй дизъюнкт:
(-S(x3) + L(x3, SNOW) )

резольвента:
(-M(x6) + L(x6, SNOW) )

-------------------------------------
дизъюнкты:
(-M(x4) + S(x4) )
(L(PETYA, RAIN) )
(L(PETYA, SNOW) )
(M(x1) + S(x1) )
(-L(x2, RAIN) + -M(x2) )
(-S(x3) + L(x3, SNOW) )
```

```
(-L(PETYA, y1) + -L(LENA, y1) )
(L(LENA, y2) + L(PETYA, y2) )
(S(x5) )
(-M(x6) + L(x6, SNOW) )

первый дизъюнкт:
(L(PETYA, RAIN) )

второй дизъюнкт:
(-L(x2, RAIN) + -M(x2) )

резольвента:
(-M(PETYA) )

-------------------------------------
дизъюнкты:
(-M(x4) + S(x4) )
(L(PETYA, RAIN) )
(L(PETYA, SNOW) )
(M(x1) + S(x1) )
(-L(x2, RAIN) + -M(x2) )
(-S(x3) + L(x3, SNOW) )
(-L(PETYA, y1) + -L(LENA, y1) )
(L(LENA, y2) + L(PETYA, y2) )
(S(x5) )
(-M(x6) + L(x6, SNOW) )
(-M(PETYA) )

первый дизъюнкт:
(L(PETYA, RAIN) )

второй дизъюнкт:
(-L(PETYA, y1) + -L(LENA, y1) )

резольвента:
(-L(LENA, RAIN) )

-------------------------------------
```

```
дизъюнкты:
(-M(x4) + S(x4) )
(L(PETYA, RAIN) )
(L(PETYA, SNOW) )
(M(x1) + S(x1) )
(-L(x2, RAIN) + -M(x2) )
(-S(x3) + L(x3, SNOW) )
(-L(PETYA, y1) + -L(LENA, y1) )
(L(LENA, y2) + L(PETYA, y2) )
(S(x5) )
(-M(x6) + L(x6, SNOW) )
(-M(PETYA) )
(-L(LENA, RAIN) )

первый дизъюнкт:
(L(PETYA, SNOW) )

второй дизъюнкт:
(-L(PETYA, y1) + -L(LENA, y1) )

резольвента:
(-L(LENA, SNOW) )

-------------------------------------
......
-------------------------------------
дизъюнкты:
(-M(x4) + S(x4) )
(L(PETYA, RAIN) )
(L(PETYA, SNOW) )
(M(x1) + S(x1) )
(-L(x2, RAIN) + -M(x2) )
(-S(x3) + L(x3, SNOW) )
(-L(PETYA, y1) + -L(LENA, y1) )
(L(LENA, y2) + L(PETYA, y2) )
(S(x5) )
(-M(x6) + L(x6, SNOW) )
(-M(PETYA) )
```

```
(-L(LENA, RAIN) )
(-L(LENA, SNOW) )
(S(x7) + -L(x7, RAIN) )
(S(PETYA) )
(L(x8, SNOW) + M(x8) )
(S(x9) + L(x9, SNOW) )
(S(x10) + L(x10, SNOW) )
(-M(LENA) + L(PETYA, RAIN) )
(S(LENA) + L(PETYA, RAIN) )
(-L(x11, RAIN) + L(x11, SNOW) )
(-M(LENA) + -M(PETYA) )
(S(LENA) + -M(PETYA) )
(S(PETYA) + S(LENA) )
(-S(PETYA) + -L(LENA, SNOW) )
(-L(LENA, SNOW) + -M(PETYA) )
(M(PETYA) + -L(LENA, SNOW) )
(S(PETYA) + -L(LENA, SNOW) )
(-L(PETYA, RAIN) + -L(LENA, SNOW) )
(L(x12, SNOW) )
(-S(LENA) )
(-M(LENA) )
(M(LENA) )
(S(LENA) )
(L(x13, SNOW) + -L(x13, RAIN) )
(L(x14, SNOW) )
(L(x15, SNOW) )
(L(PETYA, RAIN) + L(LENA, SNOW) )
(-M(PETYA) + L(LENA, SNOW) )
(S(PETYA) + L(LENA, SNOW) )
(S(LENA) + L(PETYA, SNOW) )
(-S(PETYA) + -S(LENA) )
(-S(LENA) + -M(PETYA) )
(M(PETYA) + -S(LENA) )
(S(PETYA) + -S(LENA) )
(S(PETYA) + -M(LENA) )
(M(LENA) + M(PETYA) )
(M(PETYA) + S(LENA) )
(M(LENA) + S(PETYA) )
```

(-S(LENA) + -L(PETYA, RAIN) )
(-L(PETYA, RAIN) + -M(LENA) )
(M(LENA) + -L(PETYA, RAIN) )
(-L(PETYA, RAIN) + S(LENA) )
(M(PETYA) + -L(LENA, RAIN) )
(S(PETYA) + -L(LENA, RAIN) )
(-L(PETYA, RAIN) + -L(LENA, RAIN) )
(-L(LENA, SNOW) + L(PETYA, SNOW) )
(L(LENA, SNOW) )
(L(LENA, SNOW) + L(PETYA, SNOW) )
(-S(LENA) + L(PETYA, SNOW) )
(-M(LENA) + L(PETYA, SNOW) )
(M(LENA) + L(PETYA, SNOW) )
(-L(LENA, RAIN) + L(PETYA, SNOW) )
(M(PETYA) + L(LENA, SNOW) )
(-L(PETYA, RAIN) + L(LENA, SNOW) )
(L(LENA, x16) + -L(LENA, x16) )
(-L(LENA, RAIN) + -M(LENA) )
(-L(LENA, RAIN) + S(LENA) )
(-S(LENA) + L(LENA, SNOW) )
(-M(LENA) + L(LENA, SNOW) )
(M(LENA) + L(LENA, SNOW) )
(S(LENA) + L(LENA, SNOW) )
(-L(LENA, RAIN) + L(LENA, SNOW) )
(-L(PETYA, SNOW) + -M(PETYA) )
(S(PETYA) + -L(PETYA, SNOW) )
(-S(PETYA) + -M(PETYA) )
(-S(PETYA) + S(PETYA) )
(S(PETYA) + -M(PETYA) )
(M(PETYA) + S(PETYA) )
(S(PETYA) + -L(PETYA, RAIN) )
(-L(PETYA, SNOW) + L(PETYA, SNOW) )
(-S(PETYA) + L(PETYA, SNOW) )
(-M(PETYA) + L(PETYA, SNOW) )
(M(PETYA) + L(PETYA, SNOW) )
(S(PETYA) + L(PETYA, SNOW) )
(-L(PETYA, RAIN) + L(PETYA, SNOW) )
(S(LENA) + -L(LENA, SNOW) )

```
(-L(LENA, SNOW) + L(LENA, SNOW) )
(-L(PETYA, SNOW) )

первый дизъюнкт:
(L(PETYA, SNOW) )

второй дизъюнкт:
(-L(PETYA, SNOW) )

резольвента:
()

Доказана истинность  предположения
```

# ЗАКЛЮЧЕНИЕ

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

— описать идею метода резолюций;

— описать используемые структуры данных;

— описать алгоритм метода резолюций;

— описать алгоритм метода унификации;

— привести реализацию алгоритма;

— привести примеры работы алгоритма.