МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
“ЛЭТИ” ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра алгоритмической математики

Демонстрация различных методов резолюций

Выполнили студенты группы 9308:
Соболев Матвей;
Степовик Виктор.

Распределение работы (команда 9):
Соболев М.С. -- Создание программной реализации и разбор алгоритмов
Степовик В.С. -- Разбор теории, примеров и создание презентации

Ссылки на работы:
Ссылка на документ с теорией [настоятельно рекомендуем ознакомиться перед прочтением данного]
Ссылка на презентацию
Ссылка на Git-репозиторий с программой, инструкцией и файлами тестов
Ссылка на книгу Ченя и Ли
Ссылка на учебник С.Н. Позднякова и С.В. Рыбина

Стратегии метода резолюции для логики высказываний

Введение

При работе с методом резолюций рано или поздно встаёт вопрос: а каким образом лучше всего его использовать? Какие дизъюнкты выбирать первыми, а какие и вовсе лучше не использовать? На этот вопрос дают ответ стратегии метода резолюции. Мы попробуем представить их и описать из алгоритм, чтобы его можно было использовать на практике, а также представим их тестирование (демонстрацию) с помощью программы.

Стратегия насыщения уровней

Введение

Самый очевидный способ, который в первую очередь следует проверить, это полный перебор до того момента, как мы найдём пустой дизъюнкт. Эта идея заложена в стратегию насыщения уровней -- по сути -- полный перебор всех возможных комбинаций резольвент. Именно поэтому эта стратегия так и называется -- мы насыщаем новые “уровни” новыми дизъюнктами, которые потенциально могут привести к пустому дизъюнкту.

Алгоритм

Выписываем имеющиеся дизъюнкты D по порядку в список M:

```
M = {D_1, D_2,D_3…D_n}
i = 1;
j = 2;
```

ПОКА D_n != □ (пустому дизъюнкту) ВЫПОЛНЯТЬ

```
-Вычисляются резольвенты C путем сравнения каждого дизъюнкта D_i,
с дизъюнктом D_j из списка M, где i != j.
ПОКА не найдена пара дизъюнктов, дающая резольвенту ВЫПОЛНЯТЬ:
        ЕСЛИ D_i и D_j дают резольвенту,ТО
                Записываем резольвенту в список M
        ЕСЛИ j !=n ТО  j++ ;
        ИНАЧЕ  i++ и  j = i +1;
-Записать вычисленную резольвенту в список M (n = n+1, M{} = M{} + C)
```

КЦ.

Пример

Применим стратегию насыщения уровней для выражения из логики высказываний. Построим опровержение следующего множества дизъюнктов используя стратегию насыщения уровней:

S = {D_1 = X ∨ Y, D_2 = !X ∨ !Y, D_3 = X ∨ Z, D_4 = !X ∨ Z, D_5 = !Z}

Итерация 1. Строим уровень S_1:

D_6 = [D_1, D_2] = Y ∨ !Y = 1;
D_7 = [D_1, D_2] = X ∨ !X = 1;
D_8 = [D_2, D_3] = !Y ∨ Z;
D_9 = [D_1, D_4] = Y ∨ Z;
D_10 = [D_3, D_4] = Z;
D_11 = [D_3, D_5] = X;
D_12 = [D_4, D_5] = !X.

Итерация 2. Строим уровень S_2:

D_13 = [D_1, D_6] = X ∨ Y;
D_14 = [D_2, D_6] = !X ∨ !Y;
D_15 = [D_1, D_7] = X ∨ Y;
D_16 = [D_2, D_7] = !X ∨ !Y;
D_17 = [D_3, D_7] = X ∨ Z;
D_18 = [D_4, D_7] = !X ∨ Z;
D_19 = [D_1, D_8] = X ∨ Z;
D_20 = [D_5, D_8] = !Y;
D_21 = [D_6, D_8] = !Y ∨ Z;
D_22 = [D_2, D_9] = !X ∨ Z;
D_23 = [D_6, D_9] = Y ∨ Z;
D_24 = [D_8, D_9] = Z,
D_25 = [D_5, D_10] = □.

Программный тест

```
---------- FILE INPUT ----------
Please write file name(<file name>):
/Users/matmanbj/Documents/eclipse-workspace/Alternative exam/examples/example_theory.txt
Current file input:
Input 1: X + Y
Input 2: !X + !Y
Input 3: X + Z
Input 4: !X + Z
Input 5: !Z

---------- CONSOLE OUTPUT ----------
№1 (id 1): (0, 0, 0) X + Y
№2 (id 2): (0, 0, 0) !X + !Y
№3 (id 3): (0, 0, 0) X + Z
№4 (id 4): (0, 0, 0) !X + Z
№5 (id 5): (0, 0, 0) !Z
№6 (id 6): (1, 2, X) 1 (~ true)
№7 (id 7): (1, 2, Y) 1 (~ true)
№8 (id 8): (1, 4, X) Y + Z
№9 (id 9): (2, 3, X) !Y + Z
№10 (id 10): (3, 4, X) Z
№11 (id 11): (3, 5, Z) X
№12 (id 12): (4, 5, Z) !X
№13 (id 13): (1, 9, Y) X + Z
№14 (id 14): (1, 12, X) Y
№15 (id 15): (2, 8, Y) !X + Z
№16 (id 16): (2, 11, X) !Y
№17 (id 17): (3, 12, X) Z
№18 (id 18): (4, 11, X) Z
№19 (id 19): (5, 8, Z) Y
№20 (id 20): (5, 9, Z) !Y
№21 (id 21): (5, 10, Z) □ (~ false)
Do you want to input something else? Press 0 to NO, press else to YES:
```

Программная реализация выполняет последовательный перебор возможных резольвент, поэтому она может немного отличаться от разобранного примера.

Вывод

Плюсы	Минусы
Простая форма алгоритма -- реализуется, по сути, последовательный полный перебор	В результате работы алгоритма и программы создаётся множество ненужных дизъюнктов, которые резко увеличивают сложность алгоритма на большом количестве величин
Хорошо подходит для эмпирического выбора дизъюнктов, когда мы можем оценить для каких из них мы применим метод резолюции, так как их достаточно много	Плох для программной реализации, так как ЭВМ не может “оценить”, и работает по заданному алгоритму

Стратегия предпочтения

Введение

Очевидно, что полный перебор хоть и теоретически эффективен (охватывая и проверяя все возможные комбинации), но практически происходит, во-первых, порождение огромного количества резольвент, во-вторых, происходит перебор неупорядоченных множеств, где высока вероятность при выборе двух случайных дизъюнктов либо провести ненужную резолюцию (и породить ещё больший дизъюнкт), либо не провести вовсе (отсутствие контрарных переменных). Поэтому, сделаем ограничение: мы будем выбирать наиболее короткие дизъюнкты, чтобы не порождать большие резольвенты и чтобы быстрее прийти к пустому дизъюнкту (отсюда и название -- стратегия предпочтения).

Алгоритм

Упорядочиваем имеющиеся дизъюнкты по возрастанию количества атомов
Выписываем имеющиеся дизъюнкты D по порядку в список M_1:

M_1= {D_1, D_2, D_3…D_n}, где D_1 - дизъюнкт с наименьшим количеством атомов, а D_n - с наибольшим.

i = 1;

j = 2;

ПОКА D_n != □ (пустому дизъюнкту) ВЫПОЛНЯТЬ

-Вычисляются резольвенты C путем сравнения каждого дизъюнкта D_i, с дизъюнктом D_j из списка M, где i != j.

ПОКА не найдена пара дизъюнктов, дающая резольвенту ВЫПОЛНЯТЬ:

ЕСЛИ D_i и D_j дают резольвенту,ТО

Записываем резольвенту в список M_2

ЕСЛИ j != n ТО j++ ;

ИНАЧЕ i++ и j = i +1;

КЦ

Слияние списков M_1 и M_2 в M_1 (n = n + m , M_1{} = M_1{} + M_2{});

Упорядочивание нового M_1 списка;

КЦ.

Пример

S = {D_1 = X ∨ Y, D_2 = !X ∨ !Y, D_3 = X ∨ Z, D_4 = !X ∨ Z, D_5 = !Z}

M_1:

(1) X ∨ Y,
(2) !X ∨ !Y,
(3) X ∨ Z,
(4) !X ∨ Z,
(5) !Z

S_2:

(6) X (из (3) и (5)),
(7) !X (из (4) и (5)),
(8) Y ∨ !Y (из (1) и (2)),
(9) X ∨ !X (из (1) и (2)),

S_3:

(10) !Y ∨ Z (из (2) и (3)),
(11) Y ∨ Z (из (1) и (4)),
(12) Z (из (3) и (4)),
(13) !Y (из (2) и (6)),
(14) Z (из (2) и (6)),
(15) Y (из (1) и (7)),
(16) Z (из (3) и (7)),
(17) □ (из (6) и (7)).

Программный тест

```
---------- FILE INPUT ----------
Please write file name(<file name>):
/Users/matmanbj/Documents/eclipse-workspace/Alternative exam/examples/example_theory.txt
Current file input:
Input 1: X + Y
Input 2: !X + !Y
Input 3: X + Z
Input 4: !X + Z
Input 5: !Z

---------- CONSOLE OUTPUT ----------
№1 (id 5): (0, 0, 0) !Z
№2 (id 1): (0, 0, 0) X + Y
№3 (id 2): (0, 0, 0) !X + !Y
№4 (id 3): (0, 0, 0) X + Z
№5 (id 4): (0, 0, 0) !X + Z
№6 (id 8): (1, 2, X) 1 (~ true)
№7 (id 9): (1, 2, Y) 1 (~ true)
№8 (id 6): (5, 3, Z) X
№9 (id 7): (5, 4, Z) !X
№10 (id 12): (3, 4, X) Z
№11 (id 10): (1, 4, X) Y + Z
№12 (id 11): (2, 3, X) !Y + Z
№13 (id 13): (5, 12, Z) □ (~ false)
Do you want to input something else? Press 0 to NO, press else to YES:
```

Вывод

Плюсы	Минусы
Так как мы выбираем наиболее короткие дизъюнкты, вероятность того, что мы придём к пустому дизъюнкту увеличивается, ведь мы можем миновать лишние дизъюнкты	В программной реализации необходимо дополнительно тратить время и ресурсы ЭВМ для того, чтобы упорядочить дизъюнкты и выбрать короткий
При порождении огромного количества дизъюнктов происходит игнорирование	При использовании предпосылок, во-первых, не всегда удаётся

длинных дизъюнктов, использование которых не требуется, поэтому это укорачивает работу алгоритма	использовать сразу кратчайшие дизъюнкты, во-вторых, не всегда есть возможность провести резолюцию по ним

Стратегия вычёркивания

Введение

Мы убедились, что полный перебор, а также предпочтение при выборе дизъюнктов, так или иначе может приводить к потенциально большому множеству ненужных, которые затрудняют работу алгоритму (тем более, работу алгоритма на ЭВМ). Поэтому нам необходимо сделать более строгое ограничение, нежели просто предпочтение в выборе дизъюнктов, которое будет намного эффективнее. Так как во время работы алгоритма порождается много лишних (а также одинаковых) дизъюнктов, мы будем их удалять. При этом, мы будем также удалять их “расширения”, то есть те дизъюнкты, которые не просто являются “тавтологией” (повторением) уже существующих, а содержат их!

Можно также добавить и предпочтение в выборе дизъюнктов, но этот алгоритм предотвращает их массовое разрастание и порождение, поэтому это ограничение [на выбор более коротких дизъюнктов] не сыграет большой роли. Кстати, отсюда и название алгоритма -- мы “вычёркиваем” повторяющиеся или расширяющиеся дизъюнкты.

Алгоритм

Выписываем имеющиеся дизъюнкты по порядку D_1,D_2,D_3...D_n в список M{}
ПОКА не найден пустой дизъюнкт (D_n != □) ВЫПОЛНЯТЬ
    -Вычисляются резольвенты C путем сравнения каждого дизъюнкта D_i,
    с дизъюнктом D_j из списка M, где i!=j.
    -Записать вычисленную резольвенту в список M (n=n+1, M{}=M{}+C)
    ЦИКЛ по k от 1 до n-1:
        ЕСЛИ (*) D_n содержит D_k или явл. его повторением, ТО
            Вычеркиваем D_n (n=n+1, M{}=M{}-D_n)
    КЦ;
КЦ.

(*) -- D_n содержит все формулы (переменные) из D_k

Интерпретация №2
Шаг 1: Сперва выписываются имеющиеся дизъюнкты по порядку C_1; C_2; C_3 … C_n
Шаг 2: Вычисляются резольвенты путем сравнения каждого дизъюнкта C_i,
с дизъюнктом C_j, i != j.
Шаг 3: Когда резольвента вычислена, она записывается в конец списка (n = n+1), как

только она порождается, если она не тавтология (не является повторением имеющихся дизъюнктов) и не поглощается каким-нибудь дизъюнктом из списка (т.е. никакой дизъюнкт из списка выше не содержится в данной). В противном случае она вычеркивается.

Пример

Применим стратегию вычеркивания ко множеству,описанному в стратегии насыщения:

S = {D_1 = X ∨ Y, D_2 = !X ∨ !Y, D_3 = X ∨ Z, D_4 = !X ∨ Z, D_5 = !Z}

Итерация 1. Строим уровень S_1:

D_6 = [D_2, D_3] = !Y ∨ Z;
D_7 = [D_1, D_4] = Y ∨ Z;
D_8 = [D_3, D_4] = Z;
D_9 = [D_3, D_5] = X;
D_10 = [D_4, D_5] = !X.

Итерация 2. Строим уровень S_2:

D_11 = [D_5, D_6] = !Y;
D_12 = [D_5, D_7] = Y
D_13 = [D_5, D_8] = □.

Гораздо короче, чем пример в стратегии насыщения уровней, не правда ли? Поэтому в данном случае лучшим является именно эта стратегия. Несмотря на то, что мы удаляем расширения дизъюнктов, этот алгоритм, если пустой дизъюнкт можно построить, обязательно его построит!

Программный тест

```
---------- FILE INPUT ----------
Please write file name(<file name>):
/Users/matmanbj/Documents/eclipse-workspace/Alternative exam/examples/example_theory.txt
Current file input:
Input 1: X + Y
Input 2: !X + !Y
Input 3: X + Z
Input 4: !X + Z
Input 5: !Z

---------- CONSOLE OUTPUT ----------
№1 (id 1): (0, 0, 0) X + Y
№2 (id 2): (0, 0, 0) !X + !Y
№3 (id 3): (0, 0, 0) X + Z
№4 (id 4): (0, 0, 0) !X + Z
№5 (id 5): (0, 0, 0) !Z
№6 (id 8): (1, 4, X) Y + Z
№7 (id 9): (2, 3, X) !Y + Z
№8 (id 10): (3, 4, X) Z
№9 (id 11): (3, 5, Z) X
№10 (id 12): (4, 5, Z) !X
№11 (id 14): (1, 12, X) Y
№12 (id 16): (2, 11, X) !Y
№13 (id 21): (5, 10, Z) □ (~ false)
Do you want to input something else? Press 0 to NO, press else to YES:
```

Можно заметить, что в программной реализации максимальный id, который получается в ходе построения дизъюнктов, равен 21, хотя всего их 13. Это говорит о том, что вновь созданные дизъюнкты удаляются по алгоритму, что сокращает их общее количество.

Вывод

Плюсы	Минусы
Не порождается большого количества дизъюнктов, что упрощает работу алгоритма в целом (и работу алгоритма на ЭВМ)	На ЭВМ каждую итерацию (каждое) приходится удалять ненужные дизъюнкты
Относительно предыдущих алгоритмов имеет меньшее время работы на ЭВМ	

Метод лок-резолюции

Введение

Далее мы рассмотрим специальные методы резолюции, которые подходят для реализации узконаправленных задач. Например, можно выделить лок-резолюцию. Её суть заключается в том, что мы нумеруем каждую формулу (переменную, или предикат для логики предикатов) по возрастанию. Причём нумерация уникальна, то есть не может быть двух дизъюнктов, где формула будет под одним номером. И мы можем проводить резолюцию только по наименьшим номерам в дизъюнктах. Очевидно, что иногда мы не сможем прийти к пустому дизъюнкту. Но лок-резолюция (от английского слова “lock” -- заблокировать) блокирует, то есть фиксирует текущее положение дизъюнктов и исходит из них.

Когда мы точно знаем, как лучше проводить резолюцию, мы можем использовать метод лок-резолюции, чтобы зафиксировать их положение и ограничиться новыми порождениями.

Алгоритм

Основные правила лок-резолюции
1)Индексация литер в дизъюнкте упорядочивает его (высокий индекс - высокий приоритет)

Пример правильного порядка P_1 \/ Q_2 \/ R_3, где 3 - индекс высш приоритета
2)Разрешается отрезать только литеры с наименьшим индексом

Резольвента P_1 \/ Q_2 и ~P_3 \/ Q_4 = Q_2 \/ Q_4

Новзможно получить резольвенту от дизъюнктов P_1 \/ Q_2 и Q_3 \/ ~P_4
3)Если литера при склейке может унаследовать более одного индекса, то ей присваивается наименьший:
Q_2 \/ Q_4 = Q_2 - назовём это “Лок-склейка”

Инициализация алгоритма

```
// Каждой литере L в имеющихся посылках(утверждения записанные через дзъюнкты
// D_i) в множестве дизъюнктов S даем свой уникальный индекс(целое число)
S = {D_1;D_2;D_3...D_n} ;
j = 1;
i = 1;
ПОКА  i != n ВЫПОЛНЯТЬ
        C = D_i;
        ПОКА в C есть не индексированные литеры ВЫПОЛНЯТЬ
                //по своему усмотрению упорядочиваем литеры перед индексированием
                //для примера возьмём лексикографический порядок
                L = указатель на первую литеру в дизъюнкте C
                ЕСЛИ L не имеет индекса И по алфавиту идёт перед неиндексированными
                                                                литерами, ТО:
                        ПИСВОИТЬ_ИНДЕКС(L) //присваеваем текущей литере индекс
                        УПОРЯДОЧИТЬ(C) //сначала индексированные, потом безиндексные
                        j++;
                        L = следующая по порядку литера;
                ИНАЧЕ:
                        L = следующая по порядку литера;
        КЦ.
        i++;
КЦ.

i = 1;
j = i+1;
ПОКА S не содержит пустой дизъюнкт ВЫПОЛНЯТЬ
      ЕСЛИ  D_i и D_j имеют одинаковые литеры при наименьших в дизъюнкте индексах
                                                                ТО
                //ищем резольвенту D_i и D_j
                R = РЕЗОЛЬВЕНТА(D_i,D_j);
                //Выполняем лок-склейку дублирующихся литер
                //если таких литер нет -- лок-склейка ничего не изменит
                ЛОК-СКЛЕЙКА(R);
                S= S+R; //R = D_(n+1)
                n++;
                ЕСЛИ j!=n ,ТО: j++;
                ИНАЧЕ: i ++; j = i+1;
    ИНАЧЕ:
                ЕСЛИ j!=n ,ТО: j++;
                ИНАЧЕ: i ++; j = i+1;
КЦ.
```

Пример

Рассмотрим следующее множество дизъюнктов S:

1_P + 2_Q = D_1
3_P + 4_!Q = D_2
6_!P + 5_Q = D_3
8_!P + 7_!Q = D_4

Обратим внимание на то, как расставлены индексы в лок-резолюции. От них зависит полнота этого примера (точнее, получение пустого дизъюнкта). Индексы, обозначенные слева от атомов -- это и есть нумерация вхождения. Резолюцию мы можем проводить только по наименьшим индексам в формуле, поэтому можно получить только одну лок-резольвенту:

D_5 = [D_3, D_4, по Q] = 9_!P (можно, по сути, ставить любой номер, он влиять не будет, например, можно проиндексировать этот новый дизъюнкт как “6” или “8”)

Далее, получим только 2 резольвенты:

D_6 = [D_1, D_5] = 10_Q
D_7 = [D_2, D_5] = 11_!Q

Проведём последнюю резолюцию:

D_8 = [D_6, D_7] = □

Программный тест

```
---------- FILE INPUT ----------
Please write file name(<file name>):
/Users/matmanbj/Documents/eclipse-workspace/Alternative exam/examples/example_theory.txt
Current file input:
Input 1: X + Y
Input 2: !X + !Y
Input 3: X + Z
Input 4: !X + Z
Input 5: !Z

---------- CONSOLE OUTPUT ----------
№1 (id 1): (0, 0, 0) X + Y
№2 (id 2): (0, 0, 0) !X + !Y
№3 (id 3): (0, 0, 0) X + Z
№4 (id 4): (0, 0, 0) !X + Z
№5 (id 5): (0, 0, 0) !Z
№6 (id 8): (1, 4, X) Y + Z
№7 (id 9): (2, 3, X) !Y + Z
№8 (id 10): (3, 4, X) Z
№9 (id 11): (3, 5, Z) X
№10 (id 12): (4, 5, Z) !X
№11 (id 14): (1, 12, X) Y
№12 (id 16): (2, 11, X) !Y
№13 (id 21): (5, 10, Z) □ (~ false)
Do you want to input something else? Press 0 to NO, press else to YES:
```

Вывод

Плюсы	Минусы
Существенно упрощает сложность алгоритма, сводя его к O(n^2) (перебор: одно множество с другим), так как мы будем использовать только 1 атом из дизъюнкта, а значит, в дизъюнкте не придётся искать другие контрарные атомы	Если с определённой нумерацией лок-резолюция невозможна, то придётся делать перебор, который увеличит сложность НЕ НА, А В O(k*n) раз, где n -- количество дизъюнктов, а k -- среднее количество атомов во всех дизъюнктов (это без учёта итерации, когда резолюция возможна), соответственно, применение алгоритма приравнивается к остальным методам
Применение алгоритма очень уместно, когда мы точно знаем, по каким дизъюнктам нам следует получать новые резольвенты	

Метод семантической резолюции

Введение

Рассмотрим ещё один вариант упрощения. Как писали в своей книге Чень и Ли о том, что хоть и можно использовать стратегию вычёркивания, чтобы выбросить лишние дизъюнкты из имеющегося набора, но на их создание будет потрачено время. Поэтому встаёт похожий вопрос на тот, что мы рассматривали в предыдущей главе: каким образом НЕ УДАЛЯТЬ, А ОГРАНИЧИТЬ создание ненужных дизъюнктов? Чень и Ли имели ввиду механизм, который изначально предотвратит это. Он существует и называется “семантическая резолюция”. Он назван так, поскольку нам необходимо ввести интерпретацию (понятие, связанное с семантикой), которая разделит исходное множество дизъюнктов на две независимые группы.

Алгоритм

Для упрощения понимания алгоритма необходимо ввести несколько новых понятий

Упорядоченный дизъюнкт - последовательность различных литер,, расположенных по своему приоритету (например, лексикографический порядок в дизъюнкте)

Положительный дизъюнкт -- дизъюнкт без отрицаний составляющих его литер
Неположительный дизъюнкт -- дизъюнкт имеющий отрицания над литерами (но не над всеми)

В нашем алгоритме отрицательные литеры помещаются после положительных!

Инициализация алгоритма:

//Имеем множество упорядоченных дизъюнктов S (впредь: дизъюнкт = упорядоченный дизъюнкт)

M = множество положительных дизъюнктов S;

N = множество НЕ положительных дизъюнктов S;

j = 1;

A_0 = пустое множество;

B_0 = N;

ПОКА <A_i> НЕ содержит пустой дизъюнкт ВЫПОЛНЯТЬ:

i = 0;

ЕСЛИ <B_i> НЕ пусто, ТО:

//Вычисляем множество упорядоченных резольвент W

C_1 = упорядоченный дизъюнкт из M (отрезаемая в резольвенте литера имеет наибольший приоритет);

C_2 = упорядоченный дизъюнкт из B_i (отрезаемая в резольвенте литера имеет наименьший приоритет);

W_(i+1) = множество всевозможных резольвент из дизъюнктов типа C_1 и C_2;

//Добавляем новые/(переобъявляем) переменные A_(i+1) и B_(i+1)

A_i = множество положительных дизъюнктов из W;

B_i = множество НЕположительных дизъюнктов из W;

i++;

ИНАЧЕ:

$$T = \sum_{k=0}^{j} A\_k;$$

M = M+T;

j++

//Вычисляем множество упорядоченных резольвент R_(i+1):

C_1 = упорядоченный дизъюнкт из T (отрезаемая в резольвенте литера имеет наибольший приоритет)

C_2 = упорядоченный дизъюнкт из N (отрезаемая в резольвенте литера НЕ ОБЯЗАТЕЛЬНО имеет наименьший приоритет)

R = множество всевозможных резольвент из дизъюнктов типа C_1 и C_2

//Переобъявляем переменные A_0 и B_0:

A_0 = множество положительных дизъюнктов из R

B_0 = множество НЕположительных дизъюнктов из R

КЦ.

В описанном алгоритме B_i при каждом j в конце концов станет пустым, так как наибольшее число отрицательных литер в любом упорядоченном дизъюнкте из B_i убывает на единицу при увеличении i на единицу. Все дизъюнкты в каждом A_i являются положительными резольвентами. Нетрудно видеть, что если S противоречиво, то пустой дизъюнкт можно породить с помощью описанного алгоритма. Стратегия вычеркивания также может быть включена в этот алгоритм без потери свойства полноты, т. е. для множеств T и M, полученных на шаге “ИНАЧЕ” алгоритма, любой дизъюнкт в T или M, поглощаемый другими дизъюнктами из T или M, может быть выброшен. (Отметим,что в T ∪ M нет тавтологий, так как все дизъюнкты из T ∪ M положительны.) В примере ниже, однако, мы не используем стратегию вычеркивания.

Пример

Есть множество дизъюнктов:

!P + !Q + R
P + R
Q + R
!R

Пусть интерпретация I = {!P, !Q, !R}. Тогда, если дизъюнкты верны согласно интерпретации, то разделим дизъюнкты на 2 множества -- на то, которое будет тождественно давать 1 по интерпретации, и на то, которое будет тождественно давать 0 по интерпретации:

D_1 = !P + !Q + R = 1 + 1 + 0 = 1
D_2 = P + R = 0 + 0
D_3 = Q + R = 0 + 0
D_4 = !R = 1

Согласно этим данным отнесём дизъюнкты в разные группы:

S_1 = {!P + !Q + R; !R}

S_2 = {P + R, Q + R}

Проведём резолюцию только МЕЖДУ этими множествами:

D_5 = [D_1, D_2] = !Q + R
D_6 = [D_1, D_3] = !P + R
D_7 = [D_4, D_2] = P
D_8 = [D_4, D_3] = Q

Аналогично, мы разобьём новое множество полученных дизъюнктов по нашей исходной интерпретации. Таким образом добавим к полученным на итерации 1 дизъюнктам новые. Продолжим:

	1	0
Исходные дизъюнкты	R + !P + !Q	P + R
	!R	Q + R
Итерация 1	R+!Q	P
	R+ !P	Q

Повторяем (напомним, что мы можем составлять резольвенту из дизъюнктов из разных множеств):

D_9 = [D_1, D_7] = !Q + R
D_10 = [D_1, D_8] = !P + R
D_11 = D_12 = [D_3, D_5 // D_2, D_6] = R

Разбиваем:

	1	0
Исходные дизъюнкты	!P + !Q + R	P + R
	!R	Q + R
Итерация 1	!Q + R	P
	!P + R	Q
Итерация 2	!Q + R	R
	!P + R	

D_13 = [D_4, D_11] = □

Программный тест

```
---------- CONSOLE INPUT ----------
Repeated and same disjuncts will be deleted!
How many disjuncts do you want?
3
Input them:
!A + B
Input 1: !A + B
A + B
Input 2: A + B
!B
Input 3: !B
Current console input:
№1 (id 1): (0, 0, 0) !A + B
№2 (id 2): (0, 0, 0) A + B
№3 (id 3): (0, 0, 0) !B

+---------------------------------------+---------------------------------------+
|TRUE INTERPRETATION                    |FALSE INTERPRETATION                   |
+---------------------------------------+---------------------------------------+
+---------------------------------------+---------------------------------------+
|ITERATION 1                                                                    |
+---------------------------------------+---------------------------------------+
|№1 (id 1): (0, 0, 0) !A + B            |№1 (id 3): (0, 0, 0) !B                |
|---------------------------------------|---------------------------------------|
|№2 (id 2): (0, 0, 0) A + B             |                                       |
+---------------------------------------+---------------------------------------+
|ITERATION 2                                                                    |
+---------------------------------------+---------------------------------------+
|№3 (id 5): (2, 3, B) A                 |№2 (id 4): (1, 3, B) !A                |
+---------------------------------------+---------------------------------------+
|ITERATION 3                                                                    |
+---------------------------------------+---------------------------------------+
|№4 (id 8): (2, 4, A) B                 |№3 (id 9): (5, 4, A) □ (~ false)       |
+---------------------------------------+---------------------------------------+
|ITERATION 4                                                                    |
+---------------------------------------+---------------------------------------+
---------- CONSOLE OUTPUT ----------
Do you want to input something else? Press 0 to NO, press else to YES:
```

```
Current console input:
№1 (id 1): (0, 0, 0) !P + !Q + R
№2 (id 2): (0, 0, 0) P + R
№3 (id 3): (0, 0, 0) Q + R
№4 (id 4): (0, 0, 0) !R

+---------------------------------------+---------------------------------------+
|TRUE INTERPRETATION                    |FALSE INTERPRETATION                   |
+---------------------------------------+---------------------------------------+
+---------------------------------------+---------------------------------------+
|ITERATION 1                                                                    |
+---------------------------------------+---------------------------------------+
|№1 (id 1): (0, 0, 0) !P + !Q + R       |№1 (id 4): (0, 0, 0) !R                |
|---------------------------------------|---------------------------------------|
|№2 (id 2): (0, 0, 0) P + R             |                                       |
|---------------------------------------|---------------------------------------|
|№3 (id 3): (0, 0, 0) Q + R             |                                       |
+---------------------------------------+---------------------------------------+
|ITERATION 2                                                                    |
+---------------------------------------+---------------------------------------+
|№4 (id 6): (2, 4, R) P                 |№2 (id 5): (1, 4, R) !P + !Q           |
|---------------------------------------|---------------------------------------|
|№5 (id 7): (3, 4, R) Q                 |                                       |
+---------------------------------------+---------------------------------------+
|ITERATION 3                                                                    |
+---------------------------------------+---------------------------------------+
|№6 (id 11): (2, 5, P) !Q + R           |№3 (id 13): (6, 5, P) !Q               |
|---------------------------------------|---------------------------------------|
|№7 (id 12): (3, 5, Q) !P + R           |№4 (id 14): (7, 5, Q) !P               |
+---------------------------------------+---------------------------------------+
|ITERATION 4                                                                    |
+---------------------------------------+---------------------------------------+
|№8 (id 17): (2, 14, P) R               |№5 (id 19): (6, 14, P) □ (~ false)     |
+---------------------------------------+---------------------------------------+
|ITERATION 5                                                                    |
+---------------------------------------+---------------------------------------+
---------- CONSOLE OUTPUT ----------
Do you want to input something else? Press 0 to NO, press else to YES:
```

В программном тесте использовалась интерпретация, в которой все атомы (переменные) были истинны, когда принимали истинное значение.

Вывод

Плюсы	Минусы
Первая итерация фактически существенно упрощает поиск пустого	Для подготовки интерпретации требуется эвристика: некоторые из интерпретаций

дизъюнкта, давая необходимый набор	будут давать сразу очень хороший результат, а некоторые невозможно будет отличить от обычного полного перебора
После первой итерации осуществляется переход к стратегии вычёркивания, тем самым продолжая наиболее эффективный поиск	Если “неформально” включить только раздел множества по интерпретации, алгоритм увеличивает временную сложность (ведь ЭВМ всё равно придётся делать сортировку по интерпретации)
Формально, часть разделения на множества с помощью интерпретаций не входит в алгоритм, поэтому, формально, это можно считать частичным уменьшением временной сложности	

Метод линейной резолюции

Введение

Заключительным методом резолюции является “линейная резолюция”. Она получила своё название из-за схемы своего применения: мы последовательно применяем резолюцию к одному дизъюнкту, получаем новый, а затем применяем к нему.

Мы вибираем некоторый дизъюнкт из исходного множества дизъюнктов, а к нему в свою очередь и к другому дизъюнкту (по сути, также из исходного множества) применяется правило резолюции. К полученной резольвенте, а также опять к какому-либо другому дизъюнкту опять применяем правило резолюции, получаем следующий дизъюнкт и повторяем действие. Процесс повторяется до тех пор, пока не будет получен пустой дизъюнкт, или для текущего дизъюнкта будет невозможно применить правило резолюции ни с одним другим дизъюнктом. В последнем случае осуществляет “откат” к тому дизъюнкту, где был возможен множественный выбор дизъюнкта для резольвирования.

Алгоритм

Данный алгоритм можно интерпретировать как поиск в глубину дерева резольвент от какого-то дизъюнкта(т.е. это дизъюнкт из начального мн-ва дизъюнкт будет корнем дерева). В таком случае нам необходимо определить понятие “глубины” дизъюнкта

Пусть корневой дизъюнкт C_0 = D_1 из начального множества дизъюнктов. Глубина C_0 = 0. C_1 = резольвента от C_0 и D_k, где k != 1. Глубина C_1 = 1. Таким образом, глубина C_i (= резольвенте от C_(i-1) и D_k) = i;

Специфика резолюции в данной стратегии(РЕЗОЛЬВЕНТА2.0):

Если раньше контрарные литеры по которым происходила резолюция отрезались, то в данной стратегии мы не отрезаем литеру от первого дизъюнкта (условно C_0 =), а помечаем её, причём эта пометка позволяет игнорировать данную литеру при последующем взятии резольвенты от полученного дизъюнкта.

Разберём на примере: имеем два дизъюнкта C = {P ∨ Q} и D_1 = {~R ∨ ~Q}

1) Резольвента(по Q) раньше: R = P ∨ ~R
2) Резольвента(по Q) сейчас : R = P ∨ Q ∨ ~R

Данная специфика в методе не случайна: после первой итерации алгоритма мы получим некоторые новые дизъюнкты, которые уже на следующей итерации незамедлительно будут использованы для вычисления новых резольвент. В чём трюк: мы выделяем литеры,которые УЖЕ ПОЛУЧАЛИ в качестве резольвенты на ЭТОЙ ВЕТКЕ(будет дана иллюстрация), А ЗНАЧИТ в один прекрасный момент мы можем получить ситуацию когда в данной ветке найдётся дочерний дизъюнкт с контрарными литерами, которые убираются резолюцией с прародителем.

Редукция дизъюнкта - это это операция уничтожения контрарных литер в дизъюнктах, получившихся в ходе НОВОЙ резолюции

Пример: вычислим резольвенту от R (из примера выше) и D_2= {R ∨ ~Q}

R_r = P ∨ Q ∨ ~Q

вот тут как раз и наступает момент, когда мы вспоминаем о выделенной литере т.к. Q и ~Q - контрарны, ТО: R_r = P ∨ Q ∨ ~Q = P

Инициализация алгоритма

```
S = {D_1; D_2; D_3 … D_n}
d* = максимальная глубина спуска
C_0 = D_1 //не существенно, может быть любой другой D_i
//инициализация списка пар, который можно интерпретировать как дерево
CLIST = пустое множество
ЦИКЛ по i = 2 … n :
        ЕСЛИ  R и D_i дают резольвенту, ТО :
                PAIR = {C_0,D_i};
                CLIST = CLIST + PAIR     //т.е. после этого CLIST = {PAIR_1,PAIR_2,...}
                i++
КЦ.
ПОКА CLIST != пустое множество И flag == false ВЫПОЛНЯТЬ
        i  = 1;
        //Вынимаем из CLIST пару PAIR_i = {C,D}
        XPAIR = CLIST[i];
        CLIST = CLIST - CLIST[i];
        RLIST = пустое множество;
        //вычисляем все резольвенты пары относительно разных литер
        ЕСЛИ глубина C<d* ,ТО:
            ПОКА возможна резолюция <XPAIR> ВЫПОЛНЯТЬ:
                ЕСЛИ i>1,ТО :
                        R = РЕЗОЛЬВЕНТА2.0(PAIR); //по новому способу
```

R = РЕДУКЦИЯ(R);
ИНАЧЕ: R = РЕЗОЛЬВЕНТА(PAIR) //по классич. определению
RLIST = RLIST + R;
КЦ;

ЕСЛИ RLIST содержит пустой дизъюнкт, ТО:
flag = true;// доказательство найдено
ИНАЧЕ:

ЦИКЛ по q = 1...m :
R_q = RLIST[q];
TempLIST = пустое множество;
ЕСЛИ R_q не содержится в
ЦИКЛ по j = 1 … n :
//ищем подходящие для резолюции дизъюнкты из S
ЕСЛИ R_q и D_j дают резольвенту, ТО :
PAIR = {R_q,D_j};
TempLIST = PAIR+TempLIST
j++;
КЦ;
// если TempLIST не пуст - добавляем его в начало CLIST
ЕСЛИ TempLIST !=пустое множество, ТО: CLIST = TempLIST + CLIST
q++;
КЦ;

---

i++;
КЦ.

КОММЕНТАРИЙ: вообще при небольших d* алгоритм может и не дать ответа вовсе, но запускать ещё один цикл -нет смысла, поэтому стоит запускать его несколько раз при разных d* с фиксированным шагом, чтобы получить оптимальное решение

Пример

S = {D_1= P ∨ Q, D_2 = ~P ∨ Q , D_3 = P ∨ ~Q, D_4 = ~P ∨ ~Q}
C_0 = D_1
Резолюция C_0 возможна только с D_3 и D_4
CLIST = {{C_0,D_3},{C_0,D_4}}
ПЕРВАЯ ИТЕРАЦИЯ
XPAIR = {C_0,D_3} , CLIST = {{C_0,D_4}}
R = резолюция(C_0,D_3) = P
RLIST = R = {P}
для R есть дизъюнкты D_4 и D_2 из S дающие резольвенту
TempLIST = {{{P},D_2},{{P},D_4}}
CLIST = {{{P},D_2},{{P},D_4},{C_0,D_4}}
ВТОРАЯ ИТЕРАЦИЯ
XPAIR = {{P},D_2} , CLIST = {{{P},D_4},{C_0,D_4}}

R = резольвента2.0(C_0,D_3) = P ∨ Q
RLIST = R = {P ∨ Q}
для R есть дизъюнкты D_4 и D_3 из S дающие резольвенту
TempLIST = {{{P ∨ Q},D_4},{{P ∨ Q},D_3}}
CLIST = {{{P ∨ Q},D_4},{{P ∨ Q},D_3},{{P},D_4},{C_0,D_4}}

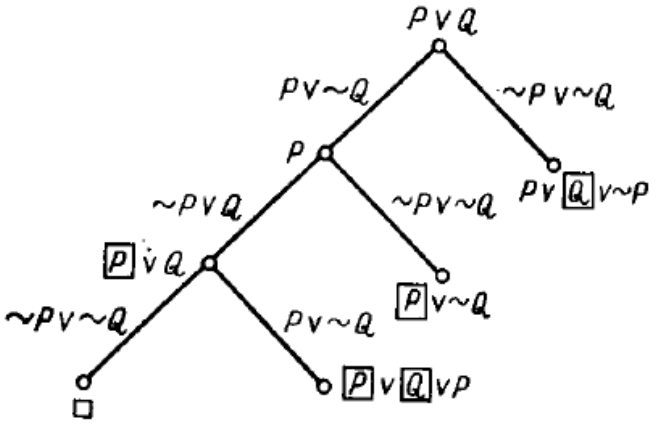
ТРЕТЬЯ ИТЕРАЦИЯ

XPAIR = {{P ∨ Q},D_4} , CLIST = {{{P ∨ Q},D_3},{{P},D_4},{C_0,D_4}}
R = резольвента2.0({P ∨ Q},D_4) = P ∨ ~P
данный дизъюнкт можно ПРОРЕДУЦИРОВАТЬ
R = РЕДУКЦИЯ(R) = □
RLIST = R = □
RLIST содержит □, следовательно: flag = true и алгоритм завершает работу

ДОКАЗАТЕЛЬСТВО НАЙДЕНО!

Приведённый выше пример не затрагивает всех аспектов работы алгоритма, однако,если бы мы изъяли на третьей итерации из CLIST пару {{P ∨ Q},D_3}, действий было бы гораздо больше, но установленная ранее d* спасла бы нас от рекурсии

ИЛЛЮСТРАЦИЯ ПРИМЕРА



Программный тест

// ВРЕМЕННО ОТСУТСТВУЕТ

Вывод

Плюсы	Минусы
В лучшем случае сложность алгоритма становится линейной: O(n)!	Аналогично семантической резолюции, требуется эвристика в поиске дерева линейной резолюции: не с каждым дизъюнктом можно её провести!

Программная реализация

Программная реализация описанных стратегий метода резолюции описана в работе Соболева Матвея: [ссылка на работу]