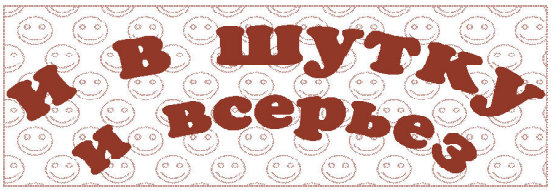
ЛОГИЧЕСКИЙ ДЕТЕКТИВ



ЛОГИЧЕСКИЙ ДЕТЕКТИВ

Данилов Константин Дмитриевич

ПРОИСШЕСТВИЕ ПЕРВОЕ. СЛУЧАЙ С СЕЙФОМ

От редакции

Уважаемые читатели! В этом году мы знакомим Вас с новой рубрикой, которая называется «Логический детектив». Возможно, многие из вас знакомы с замечательными книгами – сборниками оригинальных логических задач автора Смаллиан (), а те, кто постарше, наверное, помнят знаменитого инспектора Варнике из журнала «Наука и жизнь», который радовал читателя детективными историями с подсказками-уликами на картинках, связанных с сюжетом. ???

Для нашего журнала очень интересна тематика искусственного интеллекта: в данном случае мы хотим показать, как можно расследовать детективно-логические истории с помощью искусственного разума.

Группа студентов факультета компьютерных технологий и информатики Санкт-Петербургского государственного электротехнического университета «ЛЭТИ» под руководством Константина Данилова разработала динамический сайт http://xomak.net/d/ , на котором читатели практически могут познакомиться с автоматизацией логического вывода на примерах «логических детективов».

Логическим детективом мы будем называть «длинную» логическую задачу со сравнительно большим числом логических утверждений (логических переменных), которые связаны между собой контекстом задачи и «уликами» на прилагающейся к сюжету картинке. Улики опровергают некоторые из утверждений, которые рассматриваются как свидетельские показания в задаче. В процессе общения с программой автоматизации раскрытия «преступления» пользователь отбирает свидетельские показания, опровергая некоторые из них уликами и, после того как по его мнению этих данных достаточно, он вызывает «электронного помощника», который осуществляет логический вывод и находит «преступника» либо информирует о том, что заданная информация не позволяет однозначно его определить.

Разумеется, вы можете попробовать разгадать логический детектив и по представленному ниже описанию сюжета и картинке с уликами. Однако в таком случае на помощь электронного детектива надеяться не приходится.

ИСТОРИЯ

Был вскрыт сейф, сигнализация не сработала. Доступ к отключению сигнализации утром имеет только Алекс. Поэтому если сигнализация работала и преступление произошло не ночью, то Алекс преступник. Брюс всегда включает сигнализацию, когда уходит

с работы, в отличие от остальных - остальные постоянно забывают это делать. Вчера на работе были Карл, Дилан и Брюс. Сегодня утром на работу пришли Алекс, Карл и Дилан. Вчера было чётное число, по чётным числам Брюс уезжает раньше всех в типографию. По нечетным происходит наоборот. Ночью камеры не работали и сейф не контролировался. Если Карл ушёл последним и пришёл утром, то он преступник. Если Брюс ушёл последним и преступление произошло ночью, то он преступник. Если Дилан ушёл последним и пришёл утром, то он преступник.

НЕМНОГО О МЕТОДАХ АВТОМАТИЧЕСКОГО ДОКАЗАТЕЛЬСТВА

Надеемся, что к данному моменту внимательные читатели уже раскрыли дело о сейфе.

Поэтому перейдём к тому, как это может делать компьютер, точнее, как работают программы, автоматизирующие логический вывод. Первым шагом является формализация условия задачи. Заметим, что этот этап неформальный. Разговорный язык не является строгим и часто одни и те же предложения на естественном языке могут быть переведены в формальные логические выражения по-разному, в зависимости от контекста задачи.

Поэтому, на сайте логического детектива приводится формализация задачи:

A – Алекс – преступник,
B – Брюс – преступник,
C – Карл – преступник,
D – Дилан – преступник,
LB – Брюс ушёл из офиса последним,
LC – Карл ушёл из офиса последним,
LD – Дилан ушёл из офиса последним,
N – Преступление произошло ночью,
MA – Алекс пришёл на работу утром,
MC – Карл пришёл на работу утром,
MD – Дилан пришёл на работу утром,
S – Сигнализация работала.

Также читатель может для каждой части условия увидеть соответствующую ему логическую формулу (здесь восклицательный знак обозначает логическую операцию НЕ, амперсанд (&) соответствует логической операции И, знак вертикальной черты (|) обозначает ИЛИ, а импликация, которая связана с высказыванием типа «ЕСЛИ A ТО B» обозначается привычным образом – стрелкой (=>) (см. табл. 1).

Табл. 1

Свидетельские показания (часть из них опровергается уликами на картинке)	**Логические выражения**
если сигнализация работала и преступление произошло не ночью, то Алекс преступник.	(S & !N) => A
Брюс всегда включает сигнализацию, когда уходит с работы	LB => S
остальные постоянно забывают это делать (включать сигнализацию)	!LB => !S
Вчера на работе были Карл, Дилан и Брюс.	LB | LC | LD
Сегодня утром на работу пришли Алекс, Карл и Дилан.	MC & MD & MA
Вчера было чётное число, по чётным числам Брюс уезжает раньше всех в типографию.	!LB & (LC | LD)
Ночью камеры не работали и сейф не контролировался	N
Если Карл ушёл последним и пришёл утром, то он преступник.	(LC & MC) => C
Если Брюс ушёл последним и преступление произошло ночью, то он преступник.	(LB & N) => B
Если Дилан ушёл последним и пришёл утром, то он преступник.	(LD & MD) => D

Использование улик с картинки

Стоит обратить внимание на выражение, в котором говорится, что Брюс уезжает по чётным числам раньше всех в типографию. Видим, что в утверждении явно говорится о том, что вчера было чётное число, но почему-то авторами задачи не выделена отдельная переменная под данное утверждение. Обратим внимание, что в тексте сюжета далее следует фраза «по нечётным происходит наоборот». Именно благодаря этой фразе мы понимаем, что имеем право просто инвертировать выражение, если число было нечётным. В общем случае, без этой фразы, мы бы так сделать не смогли. Ну, а раз здесь можно провернуть такой трюк, то и зачем нам лишняя переменная?

Логический вывод

Все читатели наверняка читали Шерлока Холмса и знают, что такое «метод дедукции». Это способ из одних истинных утверждений получать другие. Оказывается, таких приёмов много, называются они силлогизмами. Самый известный силлогизм называется MODUS PONENS и формулируется так:

если верно A и верно, что A=>B, то верным является и B.

Однако в системах автоматического вывода по техническим соображениям удобно использовать другое правило, которое называется резольвентой.

Вот как оно определяется:

если верны посылки A ИЛИ D и B ИЛИ НЕ D, то верно A ИЛИ B.

Формально она записывается так:

резольвента (A | D; B | !D) = A | B.

На первый взгляд правило кажется малопонятным, но это только для людей, для компьютера оно оказалось более удобным, чем MODUS PONENS. Это объясняется тем, что математический аппарат работы с операциями НЕ, И и ИЛИ не сложен и хорошо развит.

Впрочем, для людей резольвенту можно представить более привычным образом – через импликацию:

резольвента (!D => A; D => B) = A | B, что можно прочитать так:

если при верности утверждения D верным является и утверждение B, а при неверности D верным будет утверждение A, то, по крайней мере, одно из утверждений A и B будет верным. Заметим, что эта переформулировка использована в программе для представления текста решения пользователям.

Как же программа использует резольвенту для логического вывода?

На первом шаге все формулы приводятся к некоторому стандартному виду, который называется КНФ (конъюнктивная нормальная форма). В некотором смысле – это разложение формулы на множители.

Например, !LB & (LC | LD) уже имеет требуемый вид, а формула (LB & N) => B преобразуется следующим образом:

!(LB & N) | B = !LB | !N | B.

Если формула состоит из нескольких множителей, как, например, MC & MD & MA («Сегодня утром на работу пришли Алекс, Карл и Дилан»), то ее можно разбить на три разных посылки (свидетельства):

MC – «Сегодня утром на работу пришёл Алекс»,

MD – «Сегодня утром на работу пришёл Карл»,

MA – «Сегодня утром на работу пришёл Дилан».

Далее, после того как сложные высказывания разбиты на более простые, начинает применяться резольвента к посылкам (свидетельствам) и отрицанию того, что мы хотим установить. Поочередно нам придётся подозревать каждого участника происшествия до тех пор, пока мы не приходим к противоречию (резольвента даёт значение ЛОЖЬ), что означает невиновность подозреваемого. Если после полного перебора резольвент окажется, что ЛОЖЬ не появилась, значит, доказать невиновность по этим посылкам мы не можем.

Например, если мы хотим доказать, что Брюс преступник, то к свидетельствам добавляем !B и начинаем делать резольвенты с другими формулами. Однако, в определении резольвенты у нас присутствуют два операнда (A | D или B | !D), а здесь операнд один. Как обойти эту проблему? Ответ прост: вместо !B мы запишем F | !B, где F – обозначает ЛОЖЬ. Эта добавка ничего не меняет, но формально резольвента уже применима. Другой проблемой применения резольвенты может стать избыточное число операндов. Например, в формуле !LB | !N | B их три. В этом случае первые два из них мы можем обозначить другой буквой, например, Q = !LB | !N и формула примет вид Q | B.

Теперь можно применить резольвенту:

резольвента (F | !B; Q | B) = F | Q = Q = !LB | !N

Этот формальный вывод можно попробовать перевести на естественный язык:

Мы знаем, что «Если Брюс ушёл последним и преступление произошло ночью, то он преступник», и предположили, что «Брюс – НЕ преступник». Тогда «если он не уходил последним, то преступление НЕ произошло ночью» (или, что то же самое, «если преступление произошло ночью, то Брюс НЕ ушёл последним»).

Как можно заметить, это суждение не кажется очевидным, хотя его правильность можно проверить простым перебором всех вариантов.

В заключение приведём текст, который сгенерировала программа после того, как мы выделили правильные высказывания и опровергли уликами ложные

Итак, приступим к расследованию.

Вы собрали важную информацию для расследования, повторим её:

- НЕ ((Брюс ушёл из офиса не последним И (Карл ушёл из офиса последним ИЛИ Дилан ушёл из офиса последним))).
- Преступление произошло не ночью.
- (ЕСЛИ (сигнализация работала И преступление произошло не ночью), ТО Алекс – пре-

ступник).

- (ЕСЛИ Брюс ушёл из офиса последним, ТО сигнализация работала).
- (ЕСЛИ Брюс ушёл из офиса не последним, ТО сигнализация не работала).
- (Брюс ушёл из офиса последним ИЛИ (Карл ушёл из офиса последним ИЛИ Дилан ушёл из офиса последним)).
- (Карл пришёл на работу утром И (Дилан пришёл на работу утром И Алекс пришёл на работу утром)).
- (ЕСЛИ (Карл ушёл из офиса последним И Карл пришёл на работу утром), ТО Карл – преступник).
- (ЕСЛИ (Брюс ушёл из офиса последним И преступление произошло ночью), ТО Брюс – преступник).
- (ЕСЛИ (Дилан ушёл из офиса последним И Дилан пришёл на работу утром), ТО Дилан – преступник).

Теперь можем преобразовать эти высказывания (получим КНФ):

Рассмотрим высказывание: НЕ ((Брюс ушёл из офиса не последним И (Карл ушёл из офиса последним ИЛИ Дилан ушёл из офиса последним))).

Внесем все отрицания внутрь скобок, пользуясь законами де Моргана: (Брюс ушёл из офиса последним ИЛИ (Карл ушёл из офиса не последним И Дилан ушёл из офиса не последним)).

По свойству дистрибутивности получим: ((Карл ушёл из офиса не последним ИЛИ Брюс ушёл из офиса последним) И (Дилан ушёл из офиса не последним ИЛИ Брюс ушёл из офиса последним))

Разобьем высказывания на группы:

- Группа 1:
  - Карл ушёл из офиса не последним.
  - Брюс ушёл из офиса последним.
- Группа 2:
  - Дилан ушёл из офиса не последним.
  - Брюс ушёл из офиса последним.

Рассмотрим высказывание: преступление произошло не ночью.

Разобьем высказывания на группы:

- Группа 1:
  - Преступление произошло не ночью.

Рассмотрим высказывание: (ЕСЛИ (сигнализация работала И преступление произошло не ночью), ТО Алекс – преступник).

Избавимся от импликации и эквивалентности, получим:(НЕ ((сигнализация работала И преступление произошло не ночью)) ИЛИ Алекс – преступник).

Внесем все отрицания внутрь скобок, пользуясь законами де Моргана: ((сигнализация не работала ИЛИ преступление произошло ночью) ИЛИ Алекс – преступник).

Разобьем высказывания на группы:

- Группа 1:
  - Сигнализация не работала.
  - Преступление произошло ночью.
  - Алекс – преступник.

Рассмотрим высказывание: (ЕСЛИ Брюс ушёл из офиса последним, ТО сигнализация работала).

Избавимся от импликации и эквивалентности, получим: (Брюс ушёл из офиса не последним ИЛИ сигнализация работала).

Разобьем высказывания на группы:

- Группа 1:
  - Брюс ушёл из офиса не последним.
  - Сигнализация работала.

Рассмотрим высказывание: (ЕСЛИ Брюс ушёл из офиса не последним, ТО сигнализация не работала).

Избавимся от импликации и эквивалентности, получим: (Брюс ушёл из офиса последним ИЛИ сигнализация не работала).

Разобьем высказывания на группы:

- Группа 1:
  - Брюс ушёл из офиса последним.
  - Сигнализация не работала.

Рассмотрим высказывание: (Брюс ушёл из офиса последним ИЛИ (Карл ушёл из офиса последним ИЛИ Дилан ушёл из офиса последним)).

Разобьем высказывания на группы:

- Группа 1:
  - Брюс ушёл из офиса последним.
  - Карл ушёл из офиса последним.
  - Дилан ушёл из офиса последним.

Рассмотрим высказывание: (Карл пришёл на работу утром И (Дилан пришёл на работу утром И Алекс пришёл на работу утром)).

Разобьем высказывания на группы:

- Группа 1:
  - Карл пришёл на работу утром.
- Группа 2:
  - Дилан пришёл на работу утром.
- Группа 3:
  - Алекс пришёл на работу утром.

Рассмотрим высказывание: (ЕСЛИ (Карл ушёл из офиса последним И Карл пришёл на работу утром), ТО Карл – преступник).

Избавимся от импликации и эквивалентности, получим: (НЕ ((Карл ушёл из офиса последним И Карл пришёл на работу утром)) ИЛИ Карл – преступник).

Внесем все отрицания внутрь скобок, пользуясь законами де Моргана: ((Карл ушёл из офиса не последним ИЛИ Карл пришёл на работу не утром) ИЛИ Карл – преступник).

Разобьем высказывания на группы:

- Группа 1:
  - Карл ушёл из офиса не последним.
  - Карл пришёл на работу не утром.
  - Карл – преступник.

Рассмотрим высказывание: (ЕСЛИ (Брюс ушёл из офиса последним И преступление произошло ночью), ТО Брюс – преступник).

Избавимся от импликации и эквивалентности, получим: (НЕ ((Брюс ушёл из офиса последним И преступление произошло ночью)) ИЛИ Брюс – преступник).

Внесем все отрицания внутрь скобок, пользуясь законами де Моргана: ((Брюс ушёл из офиса не последним ИЛИ преступление произошло не ночью) ИЛИ Брюс – преступник).

Разобьем высказывания на группы:

- Группа 1:
  - Брюс ушёл из офиса не последним.
  - Преступление произошло не ночью.
  - Брюс – преступник.

Рассмотрим высказывание: (ЕСЛИ (Дилан ушёл из офиса последним И Дилан пришёл

на работу утром), ТО Дилан – преступник).

Избавимся от импликации и эквивалентности, получим:(НЕ ((Дилан ушёл из офиса последним И Дилан пришёл на работу утром)) ИЛИ Дилан – преступник).

Внесем все отрицания внутрь скобок, пользуясь законами де Моргана: ((Дилан ушёл из офиса не последним ИЛИ Дилан пришёл на работу не утром) ИЛИ Дилан – преступник).

Разобьем высказывания на группы:

- Группа 1:
  - Дилан ушёл из офиса не последним.
  - Дилан пришёл на работу не утром.
  - Дилан – преступник.

Выберем нужные нам выражения и преобразуем их:

- Если Карл ушёл из офиса последним, то Брюс ушёл из офиса последним.
- Если Дилан ушёл из офиса последним, то Брюс ушёл из офиса последним.
- Если сигнализация работала, то преступление произошло ночью или Алекс – преступник.
- Если сигнализация не работала, то Брюс ушёл из офиса не последним.
- Если Дилан ушёл из офиса не последним, то Брюс ушёл из офиса последним или Карл ушёл из офиса последним.

Теперь распишем, как всё было:

Известно, что если Дилан ушёл из офиса последним, то Брюс ушёл из офиса последним. Если же Дилан ушёл из офиса не последним, то Брюс ушёл из офиса последним или Карл ушёл из офиса последним. Делаем вывод, что если Карл ушёл из офиса не последним, то Брюс ушёл из офиса последним.

Знаем, что если Карл ушёл из офиса последним, то Брюс ушёл из офиса последним. Также нам известно, что если Карл ушёл из офиса не последним, то Брюс ушёл из офиса последним. Следовательно, Брюс ушёл из офиса последним.

Известно, что если сигнализация работала, то преступление произошло ночью или Алекс – преступник. Также мы знаем, что если сигнализация не работала, то Брюс ушёл из офиса не последним. Следовательно, если преступление произошло не ночью, то Алекс – преступник или Брюс ушёл из офиса не последним.

Известно, что если преступление произошло не ночью, то Алекс – преступник или Брюс ушёл из офиса не последним. Известно, что преступление произошло не ночью. Следовательно, если Брюс ушёл из офиса последним, то Алекс – преступник.

Известно, что если Брюс ушёл из офиса последним, то Алекс – преступник. Факты говорят о том, что Брюс ушёл из офиса последним. Заключаем, что Алекс – преступник.

«ЭЛЕКТРОННЫЙ ДЕТЕКТИВ»: КТО ТАКОЙ И КАК РАБОТАЕТ?

Мы думаем, что любопытного читателя не до конца устроило наше объяснение работы детектива. «Метод резолюции понятен, но как же устроен детектив внутри? Как компьютер делает эти действия?» – спросите Вы.

Программная часть написана на языке JavaScript. Вся задача описывается человеком-составителем с использованием языка разметки HTML. Именно человек указывает, как трактовать ту или иную переменную, а также устанавливает связь между частями художественного текста, описывающего сюжет, и логическими выражениями. Приведём примеры.

Есть некая переменная A, она соответствует утверждению «Алекс – преступник». Допустим, в ходе расследования наш детектив получил её отрицание и ему нужно вывести свой вывод на русском языке, то есть сказать, что Алекс не преступник. «Что сложного?» – спросите вы. А вот что: компьютер не умеет говорить на русском языке, он не может просто и красиво сделать отрицание этой фразы, ему нужна помощь. Именно поэтому составитель

задачи указывает трактовку на русском языке как для случая, когда утверждение истинно, так и для случая, когда ложно. В подтверждение слов покажем, как детективу задана информация о переменной A:

```
<span data-name="A" data-inversion="Алекс не виновен" data-final="true">Алекс - преступник
</span>
```

Обратим внимание на атрибут data-final, который означает, что наша переменная «финальная». Что же это такое?

Если посмотреть на описание метода резолюций и на решение, которое вывел наш детектив, можно заподозрить самого детектива в обмане: в выводе не была получена ложь! Но если присмотреться, то у нас и нет подозреваемых, как того требует метод. Среди данных, собранных для детектива нет отрицания того, что мы хотим установить. Детектив считает свою работу выполненной тогда, когда получил в выводе «финальную» переменную и только её. В задаче, где только одно такое выражение является истинным, такое поведение является корректным, в ином случае детектив остановится, когда поймёт, что проверил все возможные варианты. Стоит заметить, что, с точки зрения метода резолюций, наш «метод финальных переменных» абсолютно корректен. Допустим, мы получили выражение A. Если бы мы заподозрили A, то передали бы детективу !A. Очевидно, что метод резолюций для A и !A даст ложь, ту самую, что требуется методом.

С переменными всё ясно, идём дальше. Выражения связаны с предложениями в тексте примерно таким же образом, сразу же приведём пример:

```
<span data-expression="(S & !N) => A"> если сигнализация работала и преступление произошло
не ночью, то Алекс преступник.</span>
```
[1]

Как видим, всё очень просто, но читатель, знакомый с HTML, воскликнет: «Как же выражение, которое является лишь атрибутом тега, занимает правильное место на странице, да ещё и динамически связывается с фрагментами текста, например, подсвечиваясь при наведении?» Об этом чуть далее. Ведь мы ещё не поняли, как детектив различает объекты на рисунке к сюжету.

Всё очень просто. Области улик задаются привычным образом:

```
<div class="expPicture" data-x="45" data-y="18" data-width="139" data-height="131"></div>
```

Здесь мы сообщили координаты x и y, а также ширину и высоту области с уликой. Вот мы и задали все условия задачи и подходим к разгадке, как всё это превращается в то, что мы видим.

При загрузке страницы специальный модуль просматривает объекты, расположенные на странице и формирует для них соответствующие внутренние объекты (которые и используются при преобразовании логических выражений), а также модифицирует соответствующим образом содержимое страницы (создает видимые объекты, содержащие логические выражения; назначает обработчиков событий наведения мыши на изображение и т. д.). Таким образом, задание задачи – совсем не сложный процесс, доступный человеку с начальными знаниями HTML.

ВНУТРЕННЕЕ УСТРОЙСТВО

Теперь поговорим о внутренних объектах, обеспечивающих выполнение логических операций над выражениями. В рамках проекта была написана библиотека, обеспечивающая автоматизацию при работе с выражениями. В библиотеке описан класс «логическое выражение», а также его соответствующие подклассы: переменная, логическое И, логическое ИЛИ и т. д. Создаются такие объекты с помощью вызова функции с параметрами, соот-

[1] На момент выхода статьи модуль, преобразующий выражения на привычном математическом языке в выражения, понятные системе, находился в стадии разработки.

ветствующими её аргументам, например, `LogicAND(объект_выражение_1, объект_выражение_2)`, где `объект_выражение` – объект, порождённый одной из таких функций. Отдельно скажем о переменных – они задаются следующим образом: `LogicVariable(строка_название, строка_описание_истинное, строка_описание_ложное)`. Возвращаемый объект является таким же логическим выражением, а значит, его можно использовать в качестве аргумента при конструировании логического выражения. Приведём пример:

```
var A = LogicVariable("A", "Утверждение A истинно", "Утверждение A ложно");
var B = LogicVariable("B", "Утверждение B истинно", "Утверждение B ложно");
var C = LogicVariable("C", "Утверждение C истинно", "Утверждение C ложно");
var A_and_B = LogicAND(A, B); //Построили выражение A & B
var final = LogicImplication(A_and_B, C); //Построили выражение (A & B) => C
```

Объекты класса «выражение» имеют набор методов, который позволяет производить с ними различные операции: получить трактовку выражения на естественном языке, получить выражение в логическом базисе И-ИЛИ-НЕ (избавиться от импликаций и эквиваленций), получить выражение с внесёнными под скобки отрицаниями и т. д. С их помощью мы можем получить КНФ (конъюнктивную нормальную форму – произведение элементарных дизъюнкций) для выражения, а также построить резольвенту для двух выражений, что используется в методе резолюций.

Эти методы достаточно сложны, и их реализация имеет массу нюансов, о некоторых из которых будет сказано далее. Однако в основе большинства из них лежит рекурсия: метод, преобразующий выражение, запускает такой же метод для частей выражения. Отметим, что в зависимости от типа текущего выражения, реализация метода может отличаться. Поясним на примере метода, вносящего отрицания под скобки.

Пусть имеется выражение: !(A & !(B | C)). Рассмотрим код:

```
//Пусть переменные A, B, C уже заданы
var expression=LogicNOT(LogicAND(A, LogicNOT(LogicOR(B, C))));
expression.getMath(); //Вернёт !(A & !(B | C))
var newExpression = expression.getWithoutComplexInversion();
newExpression.getMath(); //Вернёт (!A | (B | C))
```

Фактически мы вызываем метод `getWithoutComplexInversion()` у объекта класса LogicNOT. В данном классе этот метод сначала анализирует свой операнд (что отрицает отрицание), затем выполняет какие-либо действия в зависимости от класса операнда. Так, если это логическое И или ИЛИ, производит замену И на ИЛИ[2] (и наоборот, согласно закону де Моргана) и заменяет операнды этих функций на отрицание, запускает для новых операндов тот же метод `getWithoutComplexInversion()`. Если же операнд – просто переменная, то метод вернёт сам объект, для которого был вызван. Наша функция вернёт `LogicOR(LogicNOT(A).getWithoutComplexInversion()`, `LogicNOT(LogicNOT(LogicOR(B, C))).getWithoutComplexInversion())`. Обратим внимание на то, что больше нет внешнего `LogicNOT`. для отрицания переменной A рекурсивно будет вызван тот же метод, который вернёт отрицание переменной A. А что же произойдёт со вторым операндом? Он лишится двойного отрицания, поскольку конструктор функции `LogicNOT` «умный» и при создании устраняет двойные отрицания. Таким образом, второй операнд превратится в `LogicOR(B, C)`, для которого вызов `getWithoutComplexInversion()` вернёт его же. Теперь, когда мы поняли, как совершаются операции над выражениями, можем перейти непосредственно к методу резолюций.

РЕАЛИЗАЦИЯ МЕТОДА РЕЗОЛЮЦИЙ. ЛОГИЧЕСКИЕ ЦЕПОЧКИ

[2] В данной библиотеке введено упрощение: методы для преобразований работают только в базисе И-ИЛИ-НЕ.

Метод резолюций предполагает, что мы должны искать резольвенты для выражений, до тех пор пока не получим ложь. Но ведь каждая найденная резольвента – это ещё одно выражение, с помощью которого нужно попытаться построить резольвенту с остальными выражениями. Строить резольвенту за резольвентой можно долго, но неизвестно, какой путь приведёт нас к решению. Так легко запутаться. Для «электронного детектива» ситуация осложняется тем, что он должен дать развернутый рассказ, как он получил свой вывод, а не просто сказать: «Вот я получил ложь, а как именно – не помню». Мы должны иметь информацию, откуда возникла та или иная резольвента, а если и она возникла на основе другой резольвенты? Тут вводим понятие «логическая цепочка».

«Логическая цепочка» – это такой объект, который хранит цепочку действий, которые породили её текущее выражение. Начальное выражение каждой цепочки задаётся при создании (каждая цепочка – одна из посылок при выводе формул). У каждой цепочки может быть вызван метод `makeResolution(другая_цепочка)`, который вернёт новую цепочку, но при условии, что из текущих выражений цепочек (вызываемой и переданной) получается резольвента. Получившаяся цепочка будет содержать полную историю получения своего выражения. Обратим внимание на то, что запоминается переменная, по которой была построена резольвента, это поможет детективу в будущем. Если выражение в получившейся цепочке оказывается конечным, то детектив сможет проанализировать цепочку и расписать ход расследования.

Итак, мы передали нашему детективу сложные формулы, описывающие свидетельские показания. Как же они станут решением? Сначала для каждого выражения получается КНФ (используются вспомогательные методы: вноса отрицания под скобки, применения формулы поглощения, применения свойства дистрибутивности и т. д.) Выделяем из каждой КНФ дизъюнкты – это и есть наши посылки. На основе каждой посылки создаём логическую цепочку. Теперь запускаем вложенные циклы, которые пытаются получить резольвенты. На момент написания статьи применялась стратегия насыщения уровней (не самая оптимальная) с некоторой модификацией (вычёркиваются одинаковые резольвенты). Предлагаем ознакомиться со стратегиями метода резолюций самостоятельно.

После получения правильной цепочки запускается метод построения текста расследования. Метод достаточно хитрый: он работает с конца, а не с начала цепочки. Чтобы понять, зачем это сделано, попробуйте преобразовать выражение «A истинно или B истинно или C ложно» в импликацию (ранее мы говорили, как это можно сделать). Оказывается, что это можно сделать тремя способами: «Если A ложно, то B истинно или C ложно», «Если B ложно, то A истинно или C ложно», «Если C истинно, то B истинно или A истинно». Наш детектив на очередном шаге имеет лишь выражение вида «A или B или …» и должен преобразовать его всего в одну импликацию. Очевидно, что наше расследование – это цепочка. Сначала мы получаем факты, которые будем использовать далее. Вспомним, что получение резольвенты можно записать как (!D => A | C, D => B | E) = A | B | C | E. Здесь мы видим, что неоднозначность трактовки исключается (в качестве посылки берём переменную, которая даёт резольвенту). Таким образом, в какой-то момент в будущем детектив узнает, как же он должен был преобразовать выражение в начале. Теперь понятно, зачем детектив начинает с конца цепочки! Он сопоставляет с выражениями вида «A или B или …» одно единственное выражение «Если …., то …». Благодаря этому возможны цепочки вида: «получили, что если A, то B.» и где-то далее «Знаем, что если A, то B. В то же время знаем, что A, следовательно, B».

ЗАКЛЮЧЕНИЕ

О некоторых приёмах, которые помогли ответить на часть вопросов, мы уже поговорили. Как было сказано ранее, методы для работы с выражениями оказываются достаточно

громоздкими. Для того чтобы построить КНФ выражения, необходимо провести большое число промежуточных с виду простых манипуляций: убрать дубликаты переменных (A | A = A), найти константы (A & !A = FALSE), внести отрицания под скобки (!(A & B) = !A | !B), воспользоваться законом дистрибутивности (A | (B & C) = (A | C) & (A | B)) и т. д. Все эти операции достаточно просты для человека, но требуют достаточно основательного программирования.

Напоследок рассмотрим с виду тривиальную задачу, решение которой оказывается сложным.

Чтобы не порождать лишние резольвенты, мы должны сравнить создаваемую резольвенту с уже существующими. Например, A | B | C и B | A | C. Кажется, что задача элементарна. Однако вспомним, что функция ИЛИ в нашей реализации – функция двух переменных, и данные выражения могут записываться так: (A | (B | C)), ((B | A), C). Очевидно, что простое сравнение операндов не поможет. Здесь применяем уже ранее описанный подход: создадим метод `GetDisjuncts()`, вызываемый рекурсивно. В результате его вызова для выражений из нашего примера получим массивы, состоящие из одинаковых элементов. Задача сводится к сравнению массивов. Но что делать, если нужно сравнить, например: (A | (B & (C &D))) и (A | (C & (B &D)))? Просто создадим функцию `compare()`, которая будет вызываться рекурсивно, а её реализация будет зависеть от типа объекта. Например, если сравниваем два отрицания, то рекурсивно запускаем функцию сравнения для операнда обоих отрицаний.

Кстати, заметили ли вы, что детектив старается говорить разными словами? Дело в том, что он использует генератор случайных значений (синонимов), на основе которого и выбираются различные фразы в тексте расследования.

На этом наше первое знакомство с «электронным детективом» заканчивается. Надеемся, что детектив станет хорошим проводником в мир автоматических доказательств, а работа с ним доставит Вам положительные эмоции.

Данилов Константин Дмитриевич
студент факультета компьютерных технологий и информатики СПбГЭТУ «ЛЭТИ».