В.Н. Вагин, К.Ю. Хотимчук

Абдукция в задачах планирования работ в сложных объектах[1]

Аннотация. Приводится описание систем поддержки истинности, основанных на предположениях, и базовых понятиях для работы с этими системами. Разработан алгоритм абдуктивного вывода AAA. Предложен эвристический метод для этого алгоритма. Результаты экспериментов, проведенных на примере задачи составления расписаний для энергохранилищ, подтвердили эффективность алгоритма.

Ключевые слова: абдукция, абдуктивный вывод, ATMS, AAA\M\H.

Введение

В настоящее время в интеллектуальных системах растёт актуальность правдоподобных форм рассуждения, таких, как индукция и абдукция. Индуктивный вывод – это вывод от частного к общему. Абдуктивный вывод заключается в выводе причины из наблюдаемого события и является выводом от частного к частному. Схематично абдуктивный вывод можно представить следующим образом. Пусть имеется правило $A \to B$ и известно, что имеет место событие B. Тогда, по абдукции делаем заключение о том, что, быть может, имеет место и A. Точнее, при заданной теории и наблюдении, предложенном для объяснения, абдуктивный вывод должен определить одно или более наилучших объяснений наблюдения на основе заданной теории. Термин “абдукция” впервые ввел американский философ Пирс [1]. На сегодняшний день вывод по абдукции успешно применяется для решения задач диагностики [2], понимания естественного языка [3], распознавания, накопления знаний, составления расписаний и, несомненно, является очень важной частью интеллектуальных систем.

В настоящей работе рассмотрим алгоритм абдуктивного вывода с использованием Assumption-based Truth Maintenance Systems (ATMS), а так же эвристику для этого алгоритма. На примере задачи составления расписаний для энергохранилищ проведено сравнение работы алгоритма AAA без эвристики и с эвристикой.

1. Основные понятия ATMS

Большое количество самых различных задач искусственного интеллекта не может быть решено с использованием лишь вывода из посылок, поэтому при их решении большую важность приобретает возможность добавлять в базу данных факты, от части которых можно будет отказаться позже. Кроме этого, часто возникает необходимость выбрать среди взаимно несовместимых фактов какой-нибудь один. В таких случаях необходимо осуществить некоторый выбор, но при этом иметь возможность отказаться от него позже.

Эти соображения привели к созданию систем поддержки истинности (Truth-Maintenance System, TMS) [4-18], значительный вклад в разработку и исследование которых внесли Дж. Дойл [5] (одна из самых ранних TMS), Дж. ДеКлир [4, 11-13] (ввел термин “основанная на предположениях”, представив соответствующую систему поддержки истинности), Д. МакАллестер [6-8], Р. Рейтер [9, 13], Дж. Мартинс [10], К. Форбус [4, 14]. TMS, основанные на предположениях ATMS, являются семейством TMS, представляющим только хорновские дизъюнкты.

[1] Работа выполнена при финансовой поддержке РФФИ (проект № 08-07-00212а).

К достоинствам ATMS [11-18] следует отнести тот факт, что она сохраняет все выведенные промежуточные результаты, что значительно повышает эффективность поиска решений с возвратами. Принципы работы с обоснованиями являются наиболее простыми среди методов данной группы: обработка обоснований сводится к операциям над множествами. ATMS ориентирована на логику предикатов первого порядка. Выводимым заключениям сопоставляются качественные характеристики, составленные из списков обоснований, причем в ATMS каждое обоснование содержит не только список посылок правила, исходя из которого было получено заключение, но и список предположений, принятие которых позволило его вывести, т.е. окружение, в котором выведено заключение. Основные функции ATMS заключаются в построении обоснований заключений, исходя из обоснований посылок применяемых правил, а также в анализе обоснований противоречий. Вывод противоречия говорит о несовместности предположений, в рамках которых сделан вывод.

Рассмотрим основные определения, используемые в ATMS [11]. Основной структурой данных ATMS является *вершина (node)*, которая имеет формат <данное, метка, обоснования>. Возможны следующие типы вершин ATMS:

- *посылка* – определяется как факт, не требующий обоснования;
- *предположение* – обозначает допущение о некотором факте, принятое за истину, но которое может стать ложным и отвергнуто из дальнейшего рассмотрения;
- *выведенная вершина* – некоторое утверждение, выведенное решающей системой;
- *противоречие* – используется решающей системой для обозначения обнаруженных противоречий: ATMS гарантирует, что никакая вершина не будет рассматриваться из множества предположений, если вершина противоречия также следует из этого множества предположений.

С каждым утверждением связывается *обоснование*, состоящее из трех частей: обосновываемой вершины, называемой заключением, списка обосновывающих вершин и описания обоснования решающей системой. Обоснования имеют следующий вид: $x_1, x_2, ..., x_k \Rightarrow n$, где $x_1, x_2, ..., x_k$ – родительские вершины, а n – заключение. Данными вершин обоснования выступают атомы. Таким образом, обоснованию можно сопоставить дизъюнкт $\neg X_1 \vee \neg X_2 ... \vee \neg X_k \vee N$, где $X_1, X_2, ..., X_k, N$ – это данные вершин $x_1, x_2, ..., x_k, n$ соответственно. С обоснованиями можно сопоставить только хорновские дизъюнкты.

Множество предположений ATMS называется *окружением*. Также вводится противоречивое окружение, представляющее собой противоречивую конъюнкцию предположений.

Говорят, что вершина n содержится в окружении E, если n может быть получена из E и текущего множества обоснований J. Выводимость определяется в терминах исчисления высказываний: $E, J \models n$, где E рассматривается как конъюнкция атомов, и J – множество дизъюнктов.

Окружение противоречиво, если из него выводима ложь ($\perp$): $E, J \models \perp$.

Меткой вершины называется минимальное множество окружений, из которых выводима данная вершина. Метка описывает предположения данных и в отличие от обоснований строится самой ATMS. В то время как обоснование описывает, как данные получены из непосредственно предшествующих вершин, метка окружения характеризует, как данные зависят от предположений.

ATMS отслеживает непротиворечивость, состоятельность, полноту и минимальность относительно обоснований каждой метки каждой вершины [11]. Метка непротиворечива, если все ее окружения непротиворечивы. Метка для вершины n состоятельна, если n выводима в каждом окружении E данной метки: $J \models E \rightarrow n$.

Метка вершины n полна, если любое непротиворечивое окружение E, для которого $J \models E \rightarrow n$, является надмножеством некоторого окружения E' метки для вершины n: $E' \subseteq E$.

Метка минимальна, если никакое окружение метки не является надмножеством других окружений, т.е. для любой метки с окружением E не должно существовать другой метки с окружением E', такой, что $E' \subseteq E$.

Есть три основные операции ATMS: создание вершины с данными решающей системы (**AddNode**), создание предположения (**AddAssumption**) и добавление обоснования к вершине (**AddJustification**). Эти операции проиллюст-

рированы на Рис.1 - Рис. 3. На Рис. 1 создаются вершины с данными A, B и C, на Рис. 2 – предположения D и E. На Рис. 3 к вершине A добавляется обоснование $B \wedge C$, к вершинам B и C – обоснования D и E соответственно. Здесь овалом (кругом) обозначается выводимая вершина, а прямоугольником (квадратом) – вершина-предположение.

Также важна операция вычисления метки вершины. которая может быть описана с помощью операций над множествами или при помощи логических операций. В терминах множеств метка вершины *n* вычисляется как

$$\bigcup_k \left\{ x \mid x = \bigcup_i x_i, \text{где } x_i \in j_{ik} \right\},$$

здесь j_{ik} – метка i-й вершины k-го обоснования вершины *n*.

Определение 1. Будем говорить, что конъюнкция литер *f* поглощает конъюнкцию литер *g*, если существуют наиболее общий унификатор σ и конъюнкция литер *r*, такая что $f \wedge r = \sigma g$, *g* будем называть поглощённой конъюнкцией литер.

Пример 1. Пусть $f = P(A,B)$, $g = P(x,y) \wedge Q(s,t)$. Тогда σ = {*A*/*x*, *B*/*y*}, r = $Q(s,t)$. Поэтому *f* поглащает *g*.

Если метки рассматривать как пропозициональные выражения в дизъюнктивной нормальной форме, то метка вершины *n* вычисляется как $\bigvee_k \bigwedge_i j_{ik}$. После удаления противоречивых и поглощённых конъюнкций литер получаем состоятельную и минимальную метку.

В абдуктивном выводе метка вершины есть множество минимальных абдуктивных объяснений литеры на этой вершине. Далее будет показано, как описанные выше операции **AddNode**, **AddAssumption**, **AddJustification** применяются в организации абдуктивного вывода с помощью ATMS.

2. Алгоритм абдуктивного вывода, использующий ATMS

Назовем алгоритм абдуктивного вывода, использующий ATMS, как AAA (ATMS-based Abduction Algorithm) [19]. На вход алгоритма поступает исходное множество дизъюнктов и наблюдаемое событие, которое необходимо объяснить. На выходе получается множество абдуктивных объяснений этого события. Сна-

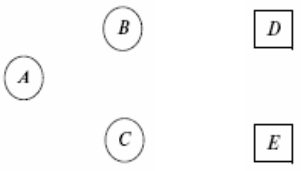

Рис.1. Рис.2. Рис.3.

чала создается вершина *GOAL* (операция **AddNode**), а наблюдаемое событие добавляется в ATMS как обоснование *GOAL* (операция **AddJustification**). Процедура **Assume** добавляет для вершины, помеченной некоторой литерой, вершину-предположение, содержащую эту же литеру (**AddAssumption**). Таким образом, реализуется допущение об истинности литеры для вершины. Далее, начиная с *GOAL*, в ход вступают процедуры **ProcessNode** и **ProcessJustification**, рекурсивно вызывающие друг друга. В процедуре **ProcessNode** для каждого обоснования текущего узла запускается механизм **ProcessJustification**. В процедуре **ProcessJustification** происходит собственно добавление новых вершин в ATMS: для каждой вершины *n* текущего обоснования, помеченной литерой *N*, ищется дизъюнкт $\neg X_1 \vee \neg X_2 \vee ... \vee \neg X_k \vee N$, и к вершине *n* добавляется обоснование $x_1, x_2, ..., x_k \Rightarrow n$, где $x_1, x_2, ..., x_k$ – это вершины, помеченные литерами $X_1, X_2, ..., X_k$ соответственно (операция **AddJustification**). Если в процессе унификации аргументы текущего обоснования изменяются, то обоснование копируется, и унификация и добавление новых узлов происходят с копией обоснования. Это необходимо для того, чтобы не потерять текущее обоснование как часть решения. После того, как дерево построено, вычисляется метка на узле *GOAL*, которая представляет собой множество минимальных абдуктивных объяснений.

Пример 2. Пусть исходное множество дизъюнктов имеет вид

$\neg p(x) \vee \neg p(A) \vee q(x)$,

$\neg q(x) \vee \neg r(x,y) \vee s(x,B)$,

$\neg m(x) \vee l(x)$.

Наблюдается конъюнкт $s(x,y) \wedge l(y)$. Требуется найти множество его абдуктивных объяснений. На Рис.4 приведён пример дерева, построенного алгоритмом AAA. Здесь прямоугольниками обозначены предположения, овалами – выводимые вершины. Начальная вер-

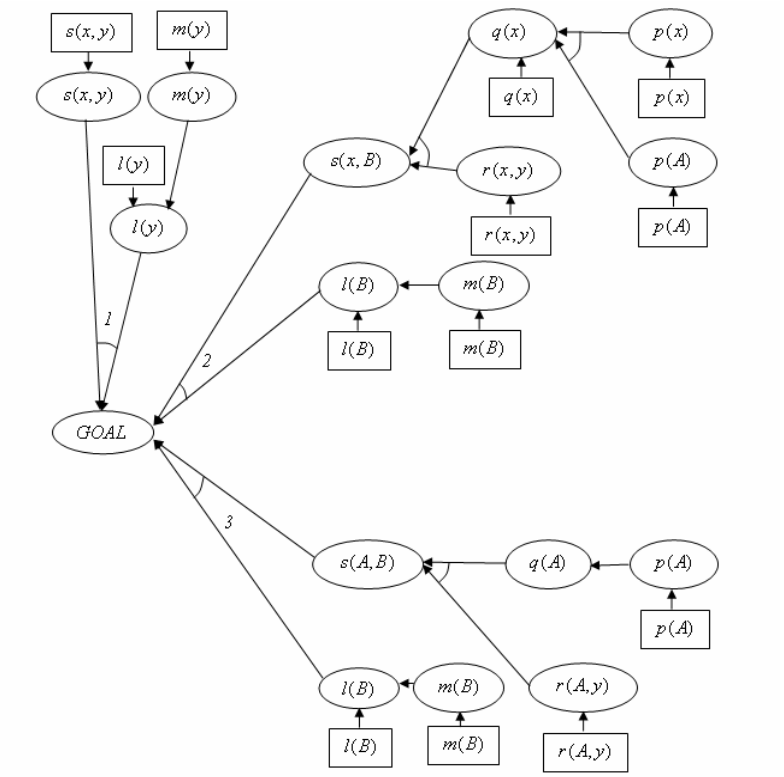

Рис.4.

шина – *GOAL*. Метка на вершине *GOAL* является множеством минимальных абдуктивных объяснений.

Рассмотрим фрагмент вычисления метки для поддерева *1*. Рассматриваем обоснование вершины GOAL: $s(x,y) \wedge l(y)$. *Label*(*GOAL* по поддереву *1*) = *Label*($s(x,y)$) $\wedge$ *Label*($l(y)$), где *Label*($s(x,y)$) – метка на вершине $s(x,y)$, и *Label*($l(y)$) – метка на вершине $l(y)$. Вычислим метку на вершине $s(x,y)$. Так как у этой вершины есть единственное обоснование, состоящее из единственного предположения $s(x,y)$, получаем *Label*($s(x,y)$) = $s(x,y)$. Вычисляем, далее, метку на вершине $l(y)$. У этой вершины два обоснования: вершина-предположение $l(y)$ и вершина $m(y)$. Поэтому *Label*($l(y)$) = $l(y)$ $\vee$ *Label*($m(y)$), где *Label*($m(y)$) – метка на вершине $m(y)$. Вычисляем *Label*($m(y)$). У этой вершины единственное обоснование, состоящее из единственной вершины-предположения $m(y)$, поэтому *Label*($m(y)$) = $m(y)$. Возвращаясь последовательно назад, получаем: *Label*($l(y)$) = $l(y)$ $\vee$ *Label*($m(y)$) = $l(y)$ $\vee$ *Label*($m(y)$) = $l(y)$ $\vee$ $m(y)$. *Label*(*GOAL* по поддереву *1*) = *Label*($s(x,y)$) $\wedge$ *Label*($l(y)$) = $s(x,y)$ $\wedge$ ($l(y)$ $\vee$ $m(y)$) = $s(x,y)$ $\wedge$ $l(y)$ $\vee$ $s(x,y)$ $\wedge$ $m(y)$. Аналогично вычисляем метку на *GOAL* полностью.

Получаем Label(*GOAL*) =

$s(x,y) \wedge l(y) \vee$

$s(x,y) \wedge m(y) \vee$

$p(x) \wedge p(A) \wedge r(x,y) \wedge l(B) \vee$

$p(x) \wedge p(A) \wedge r(x,y) \wedge m(B) \vee$

$q(x) \wedge r(x,y) \wedge l(B) \vee$

$q(x) \wedge r(x,y) \wedge m(B) \vee$

$p(A) \wedge r(A,y) \wedge l(B) \vee$

$p(A) \wedge r(A,y) \wedge m(B).$

Ответ:

$s(x,y) \wedge l(y),$

$s(x,y) \wedge m(y),$

$p(x) \wedge p(A) \wedge r(x,y) \wedge l(B),$

$p(x) \wedge p(A) \wedge r(x,y) \wedge m(B),$

$q(x) \wedge r(x,y) \wedge l(B),$

$q(x) \wedge r(x,y) \wedge m(B),$

$p(A) \wedge r(A,y) \wedge l(B),$

$p(A) \wedge r(A,y) \wedge m(B).$

3. Модифицированный алгоритм AAA\M

Задача планирования заключается в нахождении поведения объекта или системы в будущем, причём такое поведение должно привести к заранее заданной цели и удовлетворять заданным ограничениям. Результатом решения задачи планирования является множество действий, которые позволяют достичь требуемой цели. Абдуктивный вывод успешно применяется при решении задачи планирования, где в качестве наблюдаемого события выступает цель планирования, а в качестве правил – описание и ограничения предметной области. Искомое множество действий вычисляется как причина наблюдаемого события.

Большинство задач планирования могут формулироваться как нахождение/вычисление решения-кандидата с последующей его проверкой на различные условия предметной области. Очень часто процесс проверки на условие заключается в просмотре имеющихся дизъюнктов в базе знаний для того, чтобы удостовериться, что выводимое решение не противоречит этим условиям. Мы адаптировали алгоритм AAA под такого рода особенности предметной области.

Введём ряд определений.

Определение 2. Под обратной резолюцией будем понимать набор инструкций/шагов алгоритма абдуктивного вывода, т.е. вывод посылки, когда известны заключение и правило.

Например, для алгоритма AAA обратная резолюция выражается в добавлении обоснования для вершины таким образом, чтобы составленный из литер на вершинах обоснования и заключения дизъюнкт унифицировался с одним или несколькими дизъюнктами из базы правил.

Приведём пример для алгоритма AAA. Пусть есть дизъюнкт $\neg A \vee B$, и в ATMS имеется вершина b, помеченная литерой B. Тогда добавление в ATMS обоснования $a \Rightarrow b$, где вершина a помечена литерой A, является обратной резолюцией вершины b с дизъюнктом $\neg A \vee B$.

Определение 3. Будем говорить, что обратная резолюция литеры l с дизъюнктом d непротиворечива, если полученные в её результате формулы не противоречивы.

Перефразируем это определение для алгоритма AAA. Пусть вершина n помечена литерой l. Обратная резолюция литеры l с дизъюнктом d непротиворечива, если добавленные к вершине n обосновывающие вершины не обосновываются вершиной-противоречием.

Заметим, что сделать заключение о непротиворечивости обратной резолюции для вершины n можно только после того, как абдуктивным алгоритмом обработаны все поддеревья вывода вершины n.

Определение 4. Литеру, для которой должна быть выполнена хотя бы одна непротиворечивая обратная резолюция, будем называть !-литерой и в конце её названия приписывать символ '!'.

Пример 3. Пусть база правил состоит из следующих дизъюнктов:

$$allowed(1),$$
$$allowed(2),$$
$$allowed(5),$$
$$\neg find(x) \vee \neg allowed!(x) \vee result(x),$$
$$\neg N(x) \vee find(x).$$

Здесь предикат $result(x)$ означает, что x – искомое число, $find(x)$ означает, что найдено значение-кандидат x. $allowed(x)$ означает, что x может быть решением поставленной задачи. Предикат $N(x)$ означает, что x – натуральное число. Дизъюнкт $\neg find(x) \vee \neg allowed(x) \vee result(x)$ задаёт следующую схему нахождения решения. Сначала аргумент x означивается в результате резолюции с дизъюнктом $\neg N(x) \vee find(x)$. После этого проверяется условие на ограничение элементов решения. И если это условие не выполняется, то решение-кандидат x отбрасывается.

В приведённом примере x должно быть результатом, если удалась хотя бы одна резолюция по литере $allowed$. Например, если $x = 5$, то получаем результатом $x = 5$.

Если же $x = 6$, то подходящих дизъюнктов для резольвирования нет, поэтому в качестве обоснования $allowed(6)$ ставится вершина-противоречие, так что литера $result(6)$ не попадает ни в один результирующий дизъюнкт.

Понятие !-литеры имеет смысл для конкретного дизъюнкта и указывает на функциональную роль литеры в данном дизъюнкте. В других дизъюнктах та же литера может не являться !-литерой.

В рассмотренном примере решение удовлетворяло условию, если по определённой литере возможно осуществить хотя бы одну резолюцию. Нередки так же предметные области, для выполнения некоторого условия в которых необходимо произвести все подходящие обратные резолюции и убедиться в непротиворечивости новых вершин дерева вывода. Если же хотя бы одна обратная резолюция оказывается противоречивой, текущая ветвь решения полностью становится противоречивой.

Определение 5. Литеру, для которой должны быть выполнены все возможные обратные резолюции, причём все полученные в её результате формулы не противоречивы, будем называть *-литерой и в конце её названия приписывать символ '*'.

Определение 6 (для ААА). Литеру, для которой должны быть выполнены все возможные обратные резолюции, причём ни одна из вершин, которая обосновывает вершину, помеченную этой литерой, не должна обосновываться вершиной-противоречием, будем называть *-литерой и в конце её названия приписывать символ '*'.

Модифицируем пример 3:

$x \in [0;5] \vee allowed(x)$,

$x \in [10;13] \vee allowed(x)$,

$x \in [501;507] \vee allowed(x)$,

$\neg find(x) \vee \neg allowed*(x) \vee result(x)$,

$\neg N(x) \vee find(x)$.

Здесь x должно быть результатом, если из возможных обратных резолюций для вершины, помеченной литерой $allowed$, ни одна не окажется противоречивой. В этом примере смысл такого ограничения состоит в том, чтобы выводимые решения не попадали ни в промежуток [0; 5], ни в промежуток [10; 13], ни в промежуток [501; 507].

Если $x = 6$, то получаем результат $x = 6$.

Если же $x = 501$, то получаем противоречие.

Заметим, что так же как и в случае !-литеры, понятие *-литеры имеет смысл для конкретного дизъюнкта и указывает на функциональную роль литеры в данном дизъюнкте. В других дизъюнктах та же литера может не являться *-литерой.

Определение 7. Процесс нахождения литеры для непротиворечивого обратного резольвирования с вершиной, помеченной !-литерой, будем называть подтверждением, а найденную литеру – подтверждающей. Процесс нахождения литеры для противоречивого обратного резольвирования с вершиной, помеченной *-литерой, будем называть опровержением, а найденную литеру – опровергающей.

Первая модификация алгоритма ААА заключается в возможности оперирования !-литерами и *-литерами, а также во введении условий останова процессов подтверждения и опровержения, которые формулируются как:

- если для вершины, помеченной !-литерой, найдена подтверждающая литера, то оставшиеся пары дизъюнкт-подстановка для этой вершины не рассматривать и перейти к рассмотрению следующей вершины, находящейся в том же обосновании, что и рассматриваемая вершина.
- если для вершины, помеченной *-литерой, найдена опровергающая литера, то оставшиеся пары дизъюнкт-подстановка для этой вершины не рассматривать, добавить в качестве обоснования этой вершины противоречие, а также добавить противоречие как обоснование всех вершин, находящихся во всех обоснованиях на пути от рассматриваемой вершины к вершине, непосредственно обосновывающей вершину верхнего уровня *GOAL*.

Назовём эту модификацию как ААА\М.

4. Эвристический метод выбора начального порядка в исходных дизъюнктах

Так как в ААА\М, рассмотренном ранее, !-литеры и *-литеры предполагают сокращение ненужных вычислений, хотелось бы максимально приблизить момент нахождения подтверждающей литеры для процесса подтверждения и опровергающей литеры для процесса опровержения.

Определение 8. Будем называть литеру математической, если она представляет математический оператор: *add*, *substract*, *not_crossing*.

Например, литера *add*(x, y, z) принимает значение ‘Истина’, когда z является суммой x и y. Литера *not_crossing* принимает значение ‘Истина’, если отрезки, составленные из первой пары чисел и из второй пары чисел, не пересекаются.

Пример 4. not_crossing(1, 4, 10, 15) принимает значение ‘Истина’, *not_crossing*(1, 4, 3, 15) при-

нимает значение ‘Ложь’. В первом случае отрезки [1; 4] и [10; 15] не пересекаются, во втором случае отрезки [1; 4] и [3; 15] пересекаются.

Определение 9. Введём функцию Args, которая возвращает множество аргументов данной литеры, то есть $Args(l(x_1, x_2, ..., x_N)) = \{x_1, x_2, ..., x_N\}$.

Определение 10. Будем говорить, что литера l обосновывается математической литерой, если база правил содержит дизъюнкт $\neg l_m \vee \neg l_1 \vee ... \vee \neg l_N \vee l$, где l_m – математическая литера.

Предлагается эвристический алгоритм **ReArrangeLiteras** выбора начального порядка в исходных дизъюнктах для алгоритма AAA\M. Алгоритм перемещает в дизъюнкте математические литеры на позиции перед последней литерой дизъюнкта. Оставшиеся литеры упорядочиваются по мере зависимости своих аргументов. Таким образом, литеры оказываются расположенными в таком порядке, что аргументы, от значений которых зависят значения других аргументов, означивались как можно раньше.

Алгоритм ReArrangeLiteras

Исходные данные:

ruleBase – база правил.

Требуется:

Переставить литеры в дизъюнктах *ruleBase* для того, чтобы сократить процесс опровержения (т.е., построить как можно меньше вершин).

Начало.

Шаг 1. Найти в *ruleBase* все дизъюнкты, содержащие хотя бы одну *-литеру. Обозначим их множество как *StarClauses*. Пусть N – число таких дизъюнктов. Рассмотрим i – номер текущего дизъюнкта. Присвоим $i = 1$.

Шаг 2. Если $i > N$, то Конец. Иначе рассмотрим *clause* – i-й дизъюнкт в *StarClauses*. Пусть литера *lit* – *-литера в этом дизъюнкте. Найти все дизъюнкты в *ruleBase*, в которых *lit* встречается без отрицания. Обозначим найденное множество как *S*. Пусть *NS* – число таких дизъюнктов. Рассмотрим j – номер текущего дизъюнкта. Присвоим $j = 1$.

Шаг 3. Если $j > NS$, то присваиваем $i = i + 1$ и переход к шагу 2. Иначе рассмотрим *clause* – j-й дизъюнкт в *S*. Выполнить процедуру **AnalyzeMath** с параметрами *clause*, *ruleBase*. Выполнить процедуру **AnalyzeArgumentCover** с параметром *clause*. Присваиваем $j = j + 1$ и переход к шагу 3

Конец

Алгоритм **AnalyzeMath**, используемый алгоритмом **ReArrangeLiteras**, находит в дизъюнкте математическую литеру и переставляет её в в конец вплоть до положительной литеры.

Алгоритм AnalyzeMath

Исходные данные:

clause – дизъюнкт.

Требуется:

Переставить математические литеры в дизъюнкте *clause* в конец вплоть до положительной литеры.

Начало.

Шаг 1. Пусть N – число отрицательных литер в дизъюнкте *clause*. Рассмотрим i – номер текущей литеры. Присвоим $i = 1$.

Шаг 2. Если $i > N$, то Конец. Иначе рассмотрим *lit* – i-ю литеру в *clause*. Если это математическая литера, то поменять её местами с N-й литерой в дизъюнкте *clause*, присвоить $N = N - 1$ и к шагу 2.
Иначе присвоить $i = i + 1$ и к шагу 2.

Конец

Алгоритм **AnalyzeArgumentCover**, используемый алгоритмом **ReArrangeLiteras**, моделирует порядок рассмотрения литер в дизъюнкте, и накапливает *Arguments* – множество означенных аргументов. На начальные позиции помещается литера, которая имеет максимальное пересечение по аргументам с *Arguments*. Постепенно множество *Arguments* заполняется всеми аргументами рассматриваемого дизъюнкта.

Алгоритм AnalyzeArgumentCover

Исходные данные:

clause – дизъюнкт вида $\neg l_1 \vee \neg l_2 \vee ... \vee \neg l_{N_l} \vee \neg m_1 \vee \neg m_2 \vee ... \vee \neg m_{N_m} \vee l$, где $l_1, l_2, ..., l_{N_l}$ – литеры, не являющиеся математическими, $m_1, m_2, ..., m_{N_m}$ – литеры, являющиеся математическими.

Требуется:

Переставить литеры $l_1, l_2, ..., l_{N_l}$ в дизъюнкте *clause* так, чтобы аргументы, от значений которых зависят значения других аргументов, означивались как можно раньше, и самой левой литерой в дизъюнкте была литера с максимальным пересечением по аргументам с положительной литерой l.

Начало.

Шаг 1. Пусть *Arguments* – множество рассматриваемых аргументов. Присвоим $Arguments = Args(l) \setminus \bigcup_{j=1}^{N_m} Args(m_j)$. Рассмотрим i – текущая позиция литеры. Присвоим $i = 1$.

Шаг 2. Если $i > N_l - 1$, то Конец. Иначе найти такую литеру *litera* среди $l_i, l_2, ..., l_{N_l}$, что $|Arguments \cap Args(litera)| \to \max$. Присвоить $Arguments = Arguments \cup Args(litera)$. Поменять местами в дизъюнкте *clause* литеры *litera* и l_i. Присвоить $i = i + 1$ и к шагу 2.

Конец.

Будем называть модификацию алгоритма ААА\М, которая перед работой самого алгоритма осуществляет перестановку литер в исходных дизъюнктах согласно алгоритму **ReArrangeLiteras**, как ААА\М\Н.

Проиллюстрируем работу алгоритма **ReArrangeLiteras** на примере усеченной версии задачи о составлении расписаний для энергохранилищ [20].

Есть компания, которая занимается поставками электричества. В распоряжении этой компании имеется сеть электростанций, каждая из которых содержит несколько элементов хранения энергии. Эти элементы необходимо хранить в энергохранилищах в течение года. Имеется список хранилищ: $maintenance_1$, $maintenance_2$ … $maintenance_N$. Задача состоит в составлении расписания работы каждого энергохранилища.

Введем предикат $start(M, B, E)$, означающий, что работа хранилища M начинается на неделе B и заканчивается на неделе E. Таким образом, наблюдение записывается в виде следующего конъюнкта:

$start(maintenance_1, B_1, E_1) \wedge start(maintenance_2, B_2, E_2) \wedge ... \wedge start(maintenance_N, B_N, E_N)$, где переменные B_i,E_i находятся путем означивания при унификации.

Далее на решение накладываются два ограничения.

Ограничение 1 (предстартовое условие).

Предикат *maint(M)* означает, что объект M является энергохранилищем.

Предикат $duration(M, D)$ ставит в соответствие энергохранилищам M длительность D их работы.

Предикат $week(W)$ означает, что объект W – номер недели в промежутке 1…52.

Предикат $startCondition(M, B, E)$ показывает, что для объекта M выполнено предстартовое условие на протяжение периода с недели B по неделю E включительно.

Предстартовое условие для энергохранилища выполнено, если согласованы начальная неделя, конечная неделя и продолжительность хранения:

$$\forall M, B, E, D : maint(M) \wedge week(B) \wedge week(E) \wedge duration(M, D) \wedge (E = B + D - 1) \to startCondition(M, B, E).$$

Ограничение 2 (ограничение на возможность хранения элементов энергии).

Предикат $prohibited(U, B_p, E_P)$ означает, что хранение элемента энергии U на период $[B_p; E_P]$ не разрешено.

Предикат $maint\_for\_unit(M, U)$ показывает, что в энергохранилище M может храниться элемент энергии U.

Предикат $prohibitedCondition(M, B, E)$ показывает, что для энергохранилища M выполнено ограничение на возможность хранения элементов энергии на протяжение периода с недели B по неделю E включительно.

Ограничение на возможность хранения элементов энергии выполнено, если хранение элемента энергии в энергохранилище производится за пределами ограничивающих промежутков:

$$\forall U, B_p, E_P, M, B, E : prohibited(U, B_p, E_P) \wedge maint\_for\_unit(M, U) \wedge not\_crossing(B_p, E_P, B, E) \to prohibitedCondition(M, B, E).$$

Если выше перечисленные условия выполнены, то энергохранилище работает на протяжении [B; E]:

$$\forall M, B, E : startCondition(M, B, E) \wedge prohibitedCondition(M, B, E) \to start(M, B, E).$$

Множество дизъюнктов для случая пяти энергохранилищ имеет вид:

maint(M1), *maint*(M2), *maint*(M3), *maint*(M4), *maint*(M5), *duration*(M1,4), *duration*(M2,10), *duration*(M3,8), *duration*(M4,35), *duration*(M5,2), $\neg maint(m) \vee \neg week(b) \vee \neg week(e) \vee \neg dura-$

tion(m,d) ∨ ¬*add*(b,d,a) ∨ ¬*substract*(a,1,e) ∨ *startCondition*(m,b,e), *maint_for_unit*(M1, U1), *maint_for_unit*(M2, U2), *maint_for_unit*(M3, U3), *maint_for_unit*(M4, U4), *maint_for_unit*(M5, U5), *prohibited*(U1, 4, 7), *prohibited*(U1, 13, 38), *prohibited*(U1, 41, 49), *prohibited*(U2, 4, 14), *prohibited*(U2, 25, 38), *prohibited*(U2, 41, 46), *prohibited*(U3, 2, 29), *prohibited*(U3, 38, 39), *prohibited*(U3, 41, 47), *prohibited*(U4, 3, 5), *prohibited*(U4, 41, 42), *prohibited*(U4, 44, 45), *prohibited*(U5, 1, 30), *prohibited*(U5, 32, 42), *prohibited*(U5, 44, 49), ¬*prohibited*!(u, bp, ep) ∨ ¬*maint_for_unit*!(m, u) ∨ ¬*not_crossing*(b, e, bp, ep) ∨ *prohibitedCondition*(m, b, e), ¬*startCondition*(m,b,e) ∨ ¬*prohibitedCondition**(m,b,e) ∨ *start*(m,b,e), ¬*diapason*(w,1,52) ∨ *week*(w).

Исходное множество дизъюнктов для пяти энергохранилищ обозначим (I).

Находим множество *StarClauses*, представляющее собой дизъюнкты, которые содержат хотя бы одну *-литеру.

StarClauses = {¬*startCondition*(m,b,e) ∨ ¬*prohibitedCondition**(m,b,e) ∨ *start*(m,b,e)}. $N = 1$. Присваиваем $i = 1$ и рассматриваем первый и единственный дизъюнкт в этом множестве: ¬*startCondition*(m,b,e) ∨ ¬*prohibitedCondition**(m,b,e) ∨ *start*(m,b,e). Литера *prohibitedCondition* является *-литерой в этом дизъюнкте. Находим, далее, множество S - все дизъюнкты, в которых *prohibitedCondition* встречается без отрицания. S = {¬*prohibited*!(u, bp, ep) ∨ ¬*maint_for_unit*!(m, u) ∨ ¬*not_crossing*(b, e, bp, ep) ∨ *prohibitedCondition*(m, b, e)}. $NS = 1$. Присваиваем $j = 1$ и рассматриваем первый и единственный дизъюнкт в этом множестве:

¬*prohibited*!(u, bp, ep) ∨ ¬*maint_for_unit*!(m, u) ∨ ¬*not_crossing*(b, e, bp, ep) ∨*prohibitedCondition*(m, b, e).

Число отрицательных литер в этом дизъюнкте $N = 3$. Выполняем процедуру **AnalyzeMath** с этим дизъюнктом. Рассматриваем литеру *prohibited*. Она не является математической. Следующая литера *maint_for_unit* так же не является математической. На очереди литера *not_crossing*, которая является математической. Меняем её местами с N-й литерой, а $N = 3$. На этом процедура **AnalyzeMath** завершает свою работу. И начинается процедура **AnalyzeArgumentCover**. $N_1 = 2$ – число литер, не являющиеся математическими, $N_2 = 1$ – число литер, являющиеся математическими. Вычисляем первоначальное значение *Arguments*. *Arguments* = Args(*prohibitedCondition*) \ Args(*not_crossing*) = {m, b, e}\{b, e, bp, ep}={m}. Последовательно для литер *prohibited* и *maint_for_unit* вычисляем $|\ Arguments \cap Args(litera)\ |$. Для литеры prohibited: $|\ Arguments \cap Args(\ prohibited\ )| = |\ \{m\} \cap \{u, bp, ep\}\ | = |\ \varnothing\ | = 0$. Для литеры *maint_for_unit*: $|\ Arguments \cap Args(maint\_for\_unit)| = |\ \{m\} \cap \{m, u\}\ | = |\ \{m\}\ |$. Максимальное значение 1 мы получаем для литеры *maint_for_unit*. Присваиваем $Arguments = Arguments \cup Args(litera) = \{m\} \cup \{m, u\} = \{m, u\}$. Меняем местами литеру *maint_for_unit* и *prohibited*. На этом работа процедуры **AnalyzeArgumentCover** завершается. На выходе получаем дизъюнкт ¬*maint_for_unit*!(m, u) ∨ ¬*prohibited*!(u, bp, ep) ∨ ¬*not_crossing*(b, e, bp, ep) ∨ *prohibitedCondition*(m, b, e). Работа алгоритма **ReArrangeLiteras** завершается. Выходное множество дизъюнктов выглядит следующим образом:

maint(M1), *maint*(M2), *maint*(M3), *maint*(M4), *maint*(M5), *duration*(M1,4), *duration*(M2,10), *duration*(M3,8), *duration*(M4,35), *duration*(M5,2), ¬*maint*(m) ∨ ¬*week*(b) ∨ ¬*week*(e) ∨ ¬*duration*(m,d) ∨ ¬*add*(b,d,a) ∨ ¬*substract*(a,1,e) ∨ *startCondition*(m,b,e), *maint_for_unit*(M1, U1), *maint_for_unit*(M2, U2), *maint_for_unit*(M3, U3), *maint_for_unit*(M4, U4), *maint_for_unit*(M5, U5), *prohibited*(U1, 4, 7), *prohibited*(U1, 13, 38), *prohibited*(U1, 41, 49), *prohibited*(U2, 4, 14), *prohibited*(U2, 25, 38), *prohibited*(U2, 41, 46), *prohibited*(U3, 2, 29), *prohibited*(U3, 38, 39), *prohibited*(U3, 41, 47), *prohibited*(U4, 3, 5), *prohibited*(U4, 41, 42), *prohibited*(U4, 44, 45), *prohibited*(U5, 1, 30), *prohibited*(U5, 32, 42), *prohibited*(U5, 44, 49), ¬*maint_for_unit*!(m, u) ∨ ¬*prohibited*!(u, bp, ep) ∨ ¬*not_crossing*(b, e, bp, ep) ∨ *prohibitedCondition*(m, b, e), ¬*startCondition*(m,b,e) ∨ ¬*prohibitedCondition**(m,b,e) ∨ *start*(m,b,e), ¬*diapason*(w,1,52) ∨ *week*(w).

Для тестирования разработанного программного комплекса используется множество дизъюнктов (I). Ставился следующий эксперимент. Число энергохранилищ изменялось от 1 до 50. В зависимости от этого числа приведенное выше множество дизъюнктов пополнялось соответствующими правилами для новых энергохранилищ. Запрос при количестве энергохранилищ, равном одному, имеет вид:

start(M1, b1, e1) *answer*(b1, e1),

при двух энергохранилищах

start(M1, b1, e1) *start*(M2, b2, e2) *answer*(b1, e1, b2, e2) и т.д. до 50 энергохранилищ. Требовалось найти все минимальные абдуктивные объяснения вводимых запросов, а означенные переменные предиката *answer* в полученных минимальных абдуктивных объяснениях являются искомыми параметрами задачи составления расписаний энергохранилищ.

Эксперимент проводился на машине Core 2 Duo 2,20 GHz.

Запрос для случая пяти энергохранилищ выглядит следующим образом:

start(M1, b1, e1) *start*(M2, b2, e2) *start*(M3, b3, e3) *start*(M4, b4, e4) *start*(M5, b5, e5) *answer*(b1, e1, b2, e2, b3, e3, b4, e4, b5, e5)

В результате, для варианта без эвристики (AAA\M) мы получаем 32 решения с общим числом выводимых вершин, равным 12952, за время 0,5 сек:

answer(8, 11, 15, 24, 30, 37, 6, 40, 50, 51), *answer*(8, 11, 15, 24, 30, 37, 6, 40, 51, 52), *answer*(8, 11, 15, 24, 30, 37, 7, 41, 50, 51), *answer*(8, 11, 15, 24, 30, 37, 7, 41, 51, 52), *answer*(8, 11, 15, 24, 31, 38, 6, 40), *answer*(8, 11, 15, 24, 31, 38, 6, 40), *answer*(8, 11, 15, 24, 31, 38, 7, 41), *answer*(8, 11, 15, 24, 31, 38, 7, 41), *answer*(8, 11, 16, 25, 30, 37, 6, 40), *answer*(8, 11, 16, 25, 30, 37, 6, 40), *answer*(8, 11, 16, 25, 30, 37, 7, 41), *answer*(8, 11, 16, 25, 30, 37, 7, 41), *answer*(8, 11, 16, 25, 31, 38, 6, 40), *answer*(8, 11, 16, 25, 31, 38, 6, 40), *answer*(8, 11, 16, 25, 31, 38, 7, 41), *answer*(8, 11, 16, 25, 31, 38, 7, 41), *answer*(9, 12, 15, 24, 30, 37, 6, 40), *answer*(9, 12, 15, 24, 30, 37, 6, 40), *answer*(9, 12, 15, 24, 30, 37, 7, 41), *answer*(9, 12, 15, 24, 30, 37, 7, 41), *answer*(9, 12, 15, 24, 31, 38, 6, 40), *answer*(9, 12, 15, 24, 31, 38, 6, 40), *answer*(9, 12, 15, 24, 31, 38, 7, 41), *answer*(9, 12, 15, 24, 31, 38, 7, 41), *answer*(9, 12, 16, 25, 30, 37, 6, 40), *answer*(9, 12, 16, 25, 30, 37, 6, 40), *answer*(9, 12, 16, 25, 30, 37, 7, 41), *answer*(9, 12, 16, 25, 30, 37, 7, 41), *answer*(9, 12, 16, 25, 31, 38, 6, 40), *answer*(9, 12, 16, 25, 31, 38, 6, 40), *answer*(9, 12, 16, 25, 31, 38, 7, 41), *answer*(9, 12, 16, 25, 31, 38, 7, 41).

Для варианта с использованием эвристики (AAA\M\H) мы получаем те же самые решения с общим числом выводимых вершин, равным 9925, за время 0,39 сек. Таким образом, для рассмотренного примера получен выигрыш в 1,3 раза по числу вершин, и в 1,28 по времени.

При увеличении объёма задачи также получаем стабильный выигрыш во времени и по числу вершин. На Рис.5 представлен график зависимости времени выполнения программы от числа энергохранилищ. Для 50 энергохранилищ алгоритм AAA\M\H по сравнению с алгоритмом AAA\M даёт выигрыш в 1,3 раза по времени. При этом общее число выводимых вершин алгоритма AAA\M составило 140524, а общее число выводимых вершин алгоритма AAA\M\H – 104537, что означает выигрыш по числу вершин в 1,34 раза.

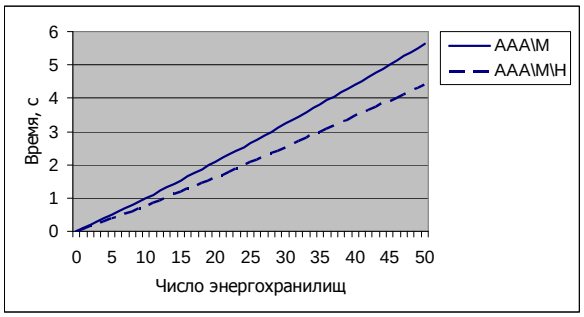

Рис. 5

Заключение

В данной работе рассмотрено применение систем поддержки истинности, основанных на предположениях, в организации абдуктивного вывода. Предложены алгоритм AAA, использующий ATMS, и алгоритм AAA\M\H, использующий эвристический метод выбора начального порядка в исходных дизъюнктах. Проведено сравнение этих двух алгоритмов. Это сравнение показало, что на машине Core 2 Duo 2.20 GHz для решения задачи составления расписаний при 50 энергохранилищах алгоритм AAA\M\H по сравнению с алгоритмом AAA во времени даёт выигрыш 1,28 и 1,34 по числу вершин. Таким образом, алгоритм AAA\M\H работает эффективнее, чем алгоритм AAA.

Литература

1. Peirce C.S., Philosophical writings. N.Y.: Dover Publications, Inc, 1955. P. 386.
2. Cox P.T. and Pietrzykowski T. General diagnosis by abductive inference // Proc. IEEE Sympos. Logic Programming. San Francisco. 1987. P. 183-189.
3. Hobbs J., Stickel M.E., Appelt D. et al. Interpretation as abduction // Artificial Intelligence. 1993. 63(1 – 2). P. 69-142.
4. Forbus K.D., de Kleer J. Building Problem Solvers. Cambridge, Massachusetts, London, England: The MIT Press, 1993. P. 716.
5. Doyle J. A Truth Maintenance System // Artificial Intelligence. 1979. V.12. P. 231-272.

6. McAllester D. A Three-valued Truth Maintenance System. Cambridge: MIT, 1978. P.31.
7. McAllester D. An Outlook on Truth Maintenance. Cambridge: MIT. 1980. P. 44.
8. McAllester D. Truth Maintenance // Proc. AAAI-90. Boston. 1990. P. 1109-1116.
9. Reiter R. A Logic for Default Reasoning // Artificial Intelligence. 1980. V. 13. P. 81-132.
10. Martins J.P. Reasoning in Multiple Belief Spaces. N.Y.: State University of New York, 1983. P. 370.
11. de Kleer J. An Assumption-based TMS // Artificial Intelligence. 1986. V. 28. P. 127-162.
12. de Kleer J. Extending the ATMS // Artificial Intelligence. 1986. V.28. P. 163-196.
13. Reiter R., de Kleer J. Foundations of Assumption-based Truth Maintenance Systems // Proc. AAAI-87, Washington. 1987. P. 183-189.
14. de Kleer J., Forbus K., McAllester D. Tutorial Notes on Truth Maintenance System // Proc. IJCAI-89, Detroit, MI. 1989. P. 81-97.
15. de Kleer J. Problem Solving with the ATMS // Artificial Intelligence. 1986. V. 28. P. 197-224.
16. Castro J.L., Zurita J.M. A Generic ATMS // International Journal of Approximate Reasoning. 1996. V. 14. P. 259-280.
17. Dechter R., Dechter A. Structure Driven Algorithms for Truth Maintenance // Artificial Intelligence. 1996. V. 82. P.1-20.
18. Вагин В.Н., Головина Е.Ю., Загорянская А.А., Фомина М.В. Достоверный и правдоподобный вывод в интеллектуальных системах. – 2е изд., испр. и доп. / Под ред. Вагина В.Н. и Поспелова Д.А. М.: ФИЗМАТЛИТ, 2008. P. 712.
19. Hwee Tou Ng, Mooney R.J. An Efficient First-Order Horn-Clause Abduction System Based on the ATMS // Proc. AAAI-91. Anaheim, CA. 1991. P. 494-499.
20. Denecker M., Kakas A. Abduction in Logic Programming // Computational Logic: Logic Programming and Beyond, Essays in Honour of Kowalski R.A., P. I, London, UK: Springer-Verlag. 2002. V. 2407. P. 402-437.

Вагин Вадим Николаевич. Профессор Московского энергетического института (Технического университета). Окончил Московский энергетический институт (Технический университет) в 1963 году. Доктор технических наук, профессор. Автор 250 печатных работ и 3-х монографий. Область научных интересов: достоверный и правдоподобный вывод в интеллектуальных системах. E-mail: Vagin@appmat.ru.

Хотимчук Кирилл Юрьевич. Младший научный сотрудник Московского энергетического института (Технического-университета). Окончил Московский энергетический институт (Технический университет) в 2008 году. Автор 12 печатных работ. Область научных интересов: абдуктивный вывод в интеллектуальных системах.
E-mail: KhotimchukKY@mail.ru.