8. Графодинамические ассоциативные машины вывода

В данном разделе описывается абстрактная графодинамическая ассоциативная машина вывода, обеспечивающая решение задач в рамках формальных теорий, описанных на языке SCL. Отметим, что графодинамическая ассоциативная машина вывода может быть легко интегрирована с навигационно-поисковой машиной описанной в разделе 7.

Данный раздел может быть использован в качестве учебного пособия по дисциплине «Логические основы интеллектуальных систем» для студентов специальности «Искусственный интеллект».

8.1. Семейство легко интегрируемых абстрактных графодинамических ассоциативных машин вывода

Ключевые понятия: графодинамическая ассоциативная машина вывода, scl-операция, цель, и-подцель, или-подцель, микропрограмма.

Открытые языки представления знаний (к числу которых относится и язык SCL) имеют в общем случае открытую нефиксированную операционную семантику. Это значит, что каждому языку представления знаний может соответствовать целое семейство абстрактных машин, поддерживающих разные стратегии решения задач на основе этого языка. Важнейшим средством повышения "интеллектуального" уровня машины переработки знаний (т.е. средством расширения множества решаемых ею задач) является интеграция этой машины с другой машиной переработки знаний, использующей в качестве внутреннего языка тот же язык представления знаний. Поэтому основным требованием, предъявляемым к современным машинам переработки знаний, наряду с открытым характером используемого языка представления знаний (что необходимо для расширения его изобразительных возможностей) является открытый характер операционной семантики указанного языка (что необходимо для неограниченного расширения логических возможностей этой машины).

Будем называть такие **scl-машинами** машины переработки знаний, которые: 1) в качестве внутреннего языка используют язык SCL, 2) в качестве вспомогательных информационных конструкций, необходимых для реализации микропрограмм, используют только sc-конструкции, хранимые в их памяти. Из вышесказанного, из определения sc-машин (см. подраздел 4.7) и из того, что язык SCL является подъязыком языка SC, следует, что все scl-машины относятся к семейству sc-машин. Как было отмечено в подразделе 4.7, все sc-машины имеют открытый характер и легко интегрируются друг с другом. Для интеграции абстрактных sc-машин необходимо интегрировать хранимые в их памяти sc-конструкции, склеив синонимичные sc-узлы, и объединить наборы их операций. Таким образом, абстрактные scl-машины, являющиеся частным видом абстрактных sc-машин, также имеют открытый характер и легко интегрируются друг с другом.

Технология реализации scl-машин должна обеспечивать и поддерживать их открытый, легко расширяемый характер. Поэтому для реализации scl-машин важное значение имеет не только уточнение набора операций scl-машины (такие операции в дальнейшем будем называть **scl-операциями**), которым ставятся в соответствие демонические **микропрограммы scl-машины**, описывающие реализацию этих операций, но и уточнение системы базовых микропрограмм scl-машин, а также уточнение самого языка микропрограммирования, ориентированного на реализацию всего семейства scl-машин. Хорошо продуманная система базовых микропрограмм и язык микропрограммирования, ориентированный на все семейство scl-машин, образуют эффективную инструментальную среду, позволяющую достаточно легко расширять и модифицировать систему операций scl-машины.

В качестве языка микропрограммирования, ориентированного на реализацию scl-машин, предлагается язык SCP (Semantic Code Programming), являющийся графовым языком параллельного процедурного программирования. Язык SCP и абстрактная scp-машина, определяющая операционную семантику этого языка, описаны в разделе 4 книги [411] (***ПрогрВАМ-2001кн***). Интерпретируя scl-машину на scp-машине, осуществляется переход от sc-машины переработки знаний к sc-машине более низкого уровня.

Подробное рассмотрение базовых микропрограмм абстрактной scl-машины приведено в работе [153] (***Голенков В.В..1996мо-БазовПТЯSCL***). В этом подразделе ниже приведено естественно-языковое описание микропрограмм некоторых операций.

Во множестве scl-операций, в частности, можно выделить следующие классы:

1) операции, обеспечивающие выполнение элементарных заданий по преобразованию текущего состояния соответствующего знания, в состав которого указанные задания входят. К числу таких операций, обеспечивающих решение элементарных задач, относятся информационно-поисковые, арифметические, строковые, теоретико-множественные операции, например, **операция эпизодического информационного поиска**;
2) операции, обеспечивающие реализацию процедурных (в том числе продукционных) программ различного вида, а точнее, реализацию различных операторов, составляющих эти программы. Процедурным программам каждого вида соответствует определенный набор операторов;
3) операции, обеспечивающие решение задач в базах знаний путем ассоциативного поиска подходящих программ, интегрированных в базы знаний;
4) набор операций, обеспечивающих сведение неэлементарных заданий к подзаданиям (в конечном счете – к элементарным заданиям), например, **операция трассировки запроса сверху-вниз**, **операция трассировки запроса снизу-вверх**, **операция декомпозиции запроса**;
5) операции, обеспечивающие выполнение неэлементарного задания после выполнения всех заданий, составляющих один из наборов И-подзаданий указанного неэлементарного задания, например, **операция выполнения логических рассуждений**;
6) операции стирания (снятия) всех явных и неявных подзаданий для уже выполненных заданий. При этом удаляются все сопутствующие вспомогательные структуры, в том числе и связки отношения ***subGoal*** ;
7) операции дедуктивного логического вывода, например, **операция реализации продукции**;
8) операции поиска и устранения противоречий в знаниях (например, на основе высказываний об однозначности) ;
9) операции, обеспечивающие поиск синонимичных sc-узлов с последующим их склеиванием, например, **операция склейки идентичных формул**;
10) операции ввода, например, **операция добавления факта** и **операция добавления высказывания**;
11) операции, обеспечивающие вывод сформированных ответов на пользовательские запросы;
12) операции "сборки мусора", обеспечивающие удаление из базы знаний ненужной информации, в частности, информации, которая редко используется и может быть при необходимости логически восстановлена (**операция уничтожения промежуточных конструкций**) .

Операция (логический механизм) , обеспечивающая решение некоторого класса элементарных задач, осуществляет окончательное разрешение некоторой задачной ситуации, т.е. получает окончательный ответ на некоторое задание. Примерами таких операций для scl-машины являются: операции информационного поиска, арифметические операции (операция сложения, операция вычитания, операция умножения и т.д.) , операции стирания.

Операция информационного ассоциативного поиска осуществляет поиск ответа на некоторый вопрос, предполагая, что ответ на это задание непосредственно содержится в текущем состоянии scl-теории.

Операция арифметического сложения инициируется тогда, когда возникает задачная ситуация, содержащая 1) задание на вычисление пока неизвестного значения некоторой числовой константы и 2) связку, обозначающую элемент отношения сложения и указывающую, что числовая константа, имеющая искомое значение, является суммой двух или нескольких других числовых констант, имеющих известные (вычисленные) значения. Результатом выполнения операции сложения является ситуация, в которой запрашиваемое (искомое) значение числовой константы оказывается вычисленным и занесенным в содержимое sc-узла, обозначающего эту числовую константу. Аналогичным образом выполняются и другие арифметические операции.

К числу операций, обеспечивающих разрешение задачных ситуаций путем использования программ, относится операция поиска и инициирования имеющейся в памяти программы (алгоритма) , реализация которой обеспечивает получение ответа на некоторое задание, а также целый ряд операций, обеспечивающих реализацию инициированных программ. Для выполнения операции поиска нужной программы из числа имеющихся необходимо, чтобы каждая из программ, входящих в состав базы знаний, снабжалась обобщенным описанием того класса задачных ситуаций, с которым она справляется. В состав такого обобщенного описания класса задач может входить описание характера (вида) задания и описание исходной дополнительной информации, необходимой для выполнения программы.

Операции сведения задач к подзадачам реализуют различные стратегии целенаправленного логического вывода (различные стратегии решения задач) , предполагающие использование той или

иной информации , содержащейся в базе знаний. В результате выполнения этих операций для исходного задания, определяющего исходную задачную ситуацию, генерируется иерархическая система подзаданий (наводящих вопросов) . Каждое задание в общем случае сводится к нескольким подзаданиям, на которые нужно получить ответы для того, чтобы мог быть сформирован ответ на исходное задание. В этом суть разбиения задач на И-подзадачи. Однако каждое задание может быть разбито на систему И-подзаданий в общем случае несколькими альтернативными способами. В этом суть разбиения задач на ИЛИ-подзадачи.

В качестве примера рассмотрим одну из операций сведения арифметической задачи к подзадачам. Пусть получен запрос на вычисление некоторой числовой константы и пусть известно, что эта числовая константа является суммой двух других констант , одна из которых определена (вычислена) , а другая – нет. Тогда задание на вычисление второй константы будет оформлено как подзадание исходного задания (в результате выполнения операции сведения задачи к подзадачам) .

Рассмотрим также логический механизм сведения задачи к подзадачам, предполагающий использование программ , хранимых в базе знаний. Суть этого механизма заключается в следующем. Если в базе знаний отсутствует программа, обеспечивающая построение ответа на некоторое задание, то ищется такая программа , которая могла бы построить этот ответ, но для выполнения которой недостаточно некоторых исходных данных. Задание на построение этих недостающих исходных данных оформляется как подзадание по отношению к исходному заданию. Механизм такой генерации подзаданий можно трактовать как один из механизмов целенаправленного синтеза требуемых программ из набора имеющихся.

В заключение еще раз подчеркнем, что рассмотренный набор операций абстрактной scl-машины легко расширяется без изменения денотационной семантики языка SCL (т.е. без изменения набора ключевых узлов языка SCL) . Расширение набора операций scl-машины, в частности, может быть направлено на поддержку (реализацию) новых стратегий решения задач.

Условием применения каждой scl-операции является наличие в памяти scl-машины соответствующей sc-конструкции. Разным scl-операциям соответствуют разные инициирующие их sc-конструкции.

8.1.1. Информационные конструкции, описывающие состояние абстрактной графодинамической ассоциативной машины вывода

Ключевые понятия и идентификаторы ключевых узлов: формула, **цель**, ***goals***, **семантически близкие высказывания**, **фактографическое высказывание**, **релевантные sc-конструкции**.

Построение хорошо продуманной системы базовых микропрограмм абстрактной scl-машины требует уточнения и унификации специальных вспомогательных sc-конструкций, используемых этими микропрограммами. Для организации таких sc-конструкций требуется введение sc-узлов, не являющихся ключевыми узлами языка SCL, но являющихся ключевыми узлами самой scl-машины.

Для представления информационных конструкций, описывающих состояние абстрактной графодинамической ассоциативной машины вывода, дополнительно к ключевым узлам языка SCL вводятся следующие ключевые узлы:

- ***goals*** – знак множества множеств целей, факторизированных по принадлежности к различным формальным теориям;
- ***formula*** – знак множества формул;
- ***active_*** – знак множества дуг принадлежности активной цели множеству целей формальной теории;
- ***search***, ***traceDown***, ***traceUp***, ***interprete***, ***produce***, ***deduce***, ***associateSimply***, ***associate***, ***associateExtensivly***, ***findUnassigned***, ***decompose***, ***analyseStructure*** – знаки множеств дуг принадлежности цели множеству целей формальной теории. Такая цель подлежит первоочередной обработке соответствующей операцией;
- ***searched***, ***tracedDown***, ***tracedUp***, ***interpreted***, ***produced***, ***deduced***, ***simplyAssociated***, ***associated***, ***extensivlyAssociated***, ***decomposed*** – знаки множеств дуг принадлежности цели множеству целей формальной теории. Такая цель считается обработанной соответствующей операцией.

Опишем типичные конструкции, которые будут необходимы для функционирования операций scl-машины. Отношение ***associative*** – отношение, соединяющее формулу ***fi*** с ассоциированными в рамках теории ***ti*** с ней формулами в результате действия операции поиска семантически близких формул:

```
associative !; □ (formula !; fi ),(theory !; ti ),s □;
```

Быть атомарным формулой ***fi***, имеющим фактографическую интерпретацию:

```
factual !;fi ;
```

Отношение ***interpretation*** – отношение, между двумя релевантными формулами и их интерпретацией (отношением релевантности) :

```
interpretation =релевантность sc-конструкций =релевантность !;
□(formula !;f1 ),(formula !;f2 ),interpretation_ : Interpretation □;
interpretation_=отношение_;
```

Отношение ***selected*** – отношение, связывающее формулу ***fi*** со множеством уже найденных для него релевантных формул (фактографических высказываний) :

```
selected !; □ (formula !;fi ), ImagesSet □;
```

Отношение, связывающее формулу ***fi*** со множеством ***Set***, множеством свободных для этой формулы переменных:

```
unassigned !; □(formula !; fi ),Set□;
```

Отношение, связывающее формулу f_1 , которая включает по структуре формулу f_2:

```
inclusion !; □ (formula !;f1 ),in_:(formula !;f2 )□;
```

Отношение между основной целью и множеством И-подцелей, используется для формирования ответа для основной цели и для объяснения процесса решения:

```
reasoning !; □ effect_:ge ,expectations_:Expectations ,causes_:Causes□;
```

где ***Causes*** – знак множества дуг принадлежности достигнутых И-подцелей множеству целей ***Goals*** определённой формальной теории, а ***Expectations*** – знак множества дуг принадлежности не достигнутых И-подцелей, ***ge*** – дуга принадлежности основной цели множеству ***Goals***.

Ключевой узел ***explanation*** обозначает отношение, связывающее выполненные (достигнутые) задания (цели) или построенные системы И-подзаданий с объясняющими scl-высказываниями. После того, как решение задачи закончено и пользователя перестают интересовать объяснения результатов решения, автоматически формируется задание на "сборку мусора" для решенной задачи, в результате выполнения которого соответствующие ***explanation***-структуры будут удалены.

Приведём конструкции, описывающие типичные запросы. Здесь и далее ***Goals*** – произвольный элемент множества ***goals***. Запрос на истинность высказывания ***fi***:

```
active_ !; (Goals !; qi );
qi = [theory !; ti !;; fi ];
```

Запрос на ложность высказывания ***fi***:

```
active_ !; ( Goals !; qi );
qi = [theory !; ti !;; _fn !;; _fc !;; fi ;conj !;; _fc ;negExpr !;; _fn ;];
```

Запрос на поиск интерпретации высказывательной формы ***fi***:

```
active_ !; ( Goals !; Set );
unassigned !; □ (formula !; fi ), Set □;
```

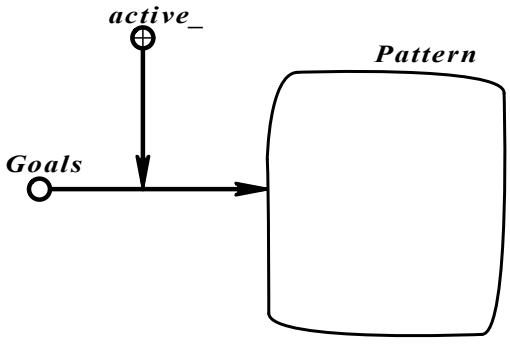
8.1.2. Операции абстрактной графодинамической ассоциативной машины вывода

Ключевые понятия: scl-операции, микропрограмма scl-машины.

Рассмотрим некоторые операции абстрактной графодинамической ассоциативной машины логического вывода. Отметим, что каждая приведённая ниже операция имеет свой абсолютный приоритет выполнения, оцениваемый по шкале от 0 до 1 (см. подраздел 6.1) .

Операция классификации и управления запросами (SCL query classification & control operation) в зависимости от структуры запроса включает его в соответствующее множество приоритетных запросов, обрабатываемых какой-либо из перечисленных ниже специализированных операций.

Условием выполнения операции классификации и управления запросами является наличие конструкции вида:

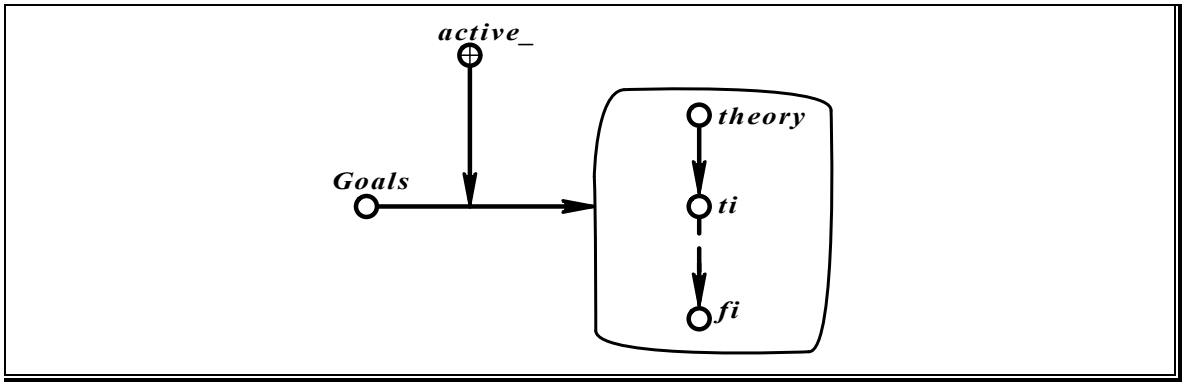


/* необходимое условие запуска операции – наличие дуги из ***active_*** */

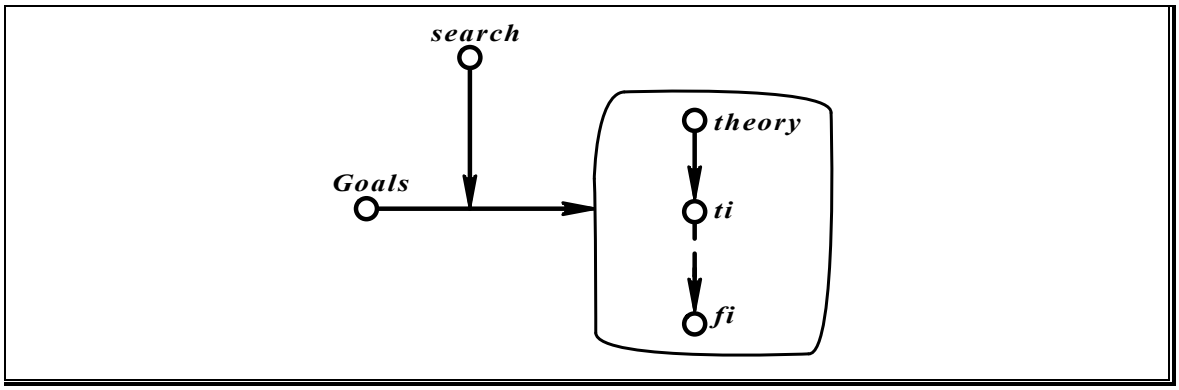
Здесь ***"Pattern"*** – образец конструкции запроса, а ***"Goals"*** – множество целей данной формальной теории.

Результатом выполнения операции классификации и управления запросами могут являться конструкции, описывающие исходный запрос как запрос определённого вида, структура которых включает ключевой узел, являющийся знаком множества sc-элементов, предназначенных для первоочередной обработки соответствующей операцией (такими элементами множества всегда являются дуги вида **(** ***Goals*** !; ***Pattern*** **)**) , а также позитивную константную дугу из ключевого узла в дугу **(** ***Goals*** !; ***Pattern*** **)** . При завершении операции классификации и управления запросами соответствующая дуга из ключевого узла ***active_***, как правило, удаляется.

Приведём пример выполнения данной операции. При наличии в базе знаний конструкции:

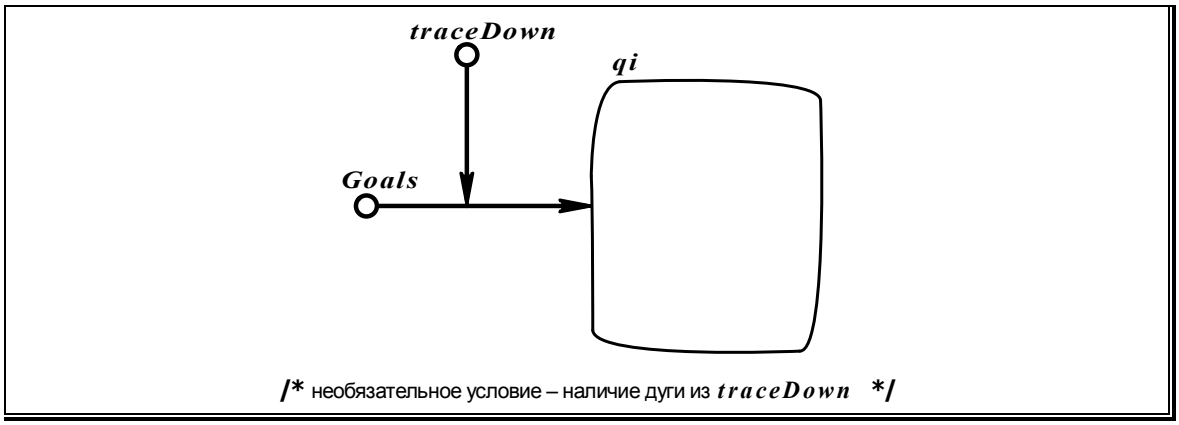


после выполнения операции мы получим, например, следующую конструкцию:



Операция трассировки запроса сверху вниз (SCL query trace-down operation) в зависимости от структуры запроса и типа формулы, к которой относится запрос, формирует запросы для всех формул, входящих в неё.

Условием выполнения операции трассировки запроса сверху-вниз является наличие конструкции вида:

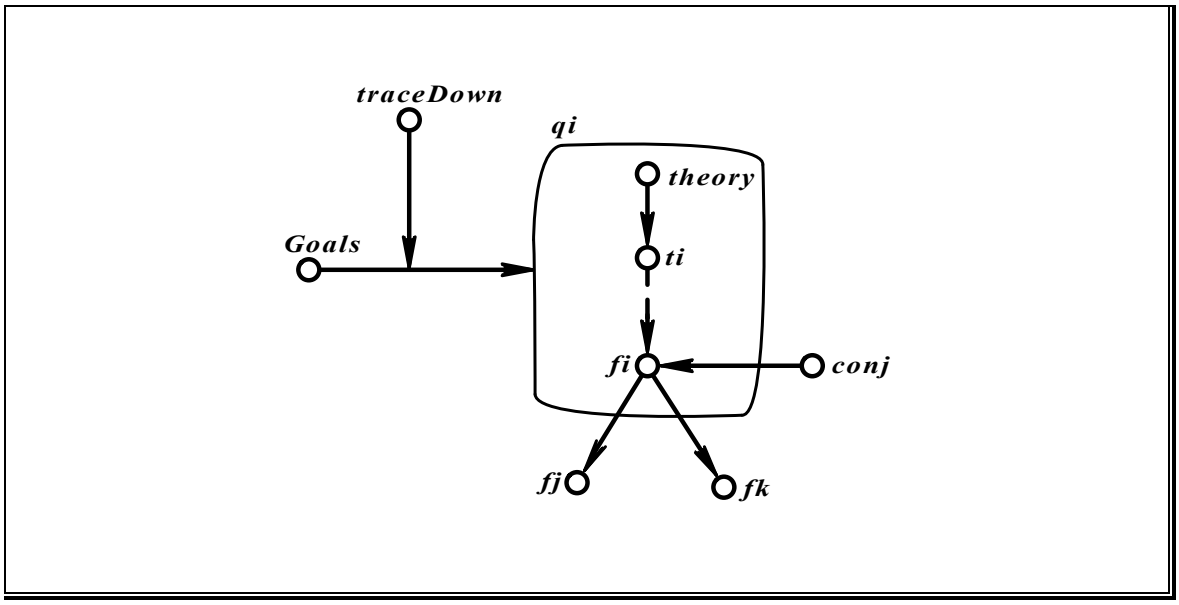


Операция, в зависимости от типа запроса qi – запрос ли это на выяснение истинности, либо ложности исходного высказывания, либо запрос на поиск интерпретации исходной формулы – и типа исходной формулы, к которой относится запрос, формирует все возможные подзапросы к формулам, входящим в исходную, группируя по И-критерию сформированные подзапросы связками отношения $reasoning$.

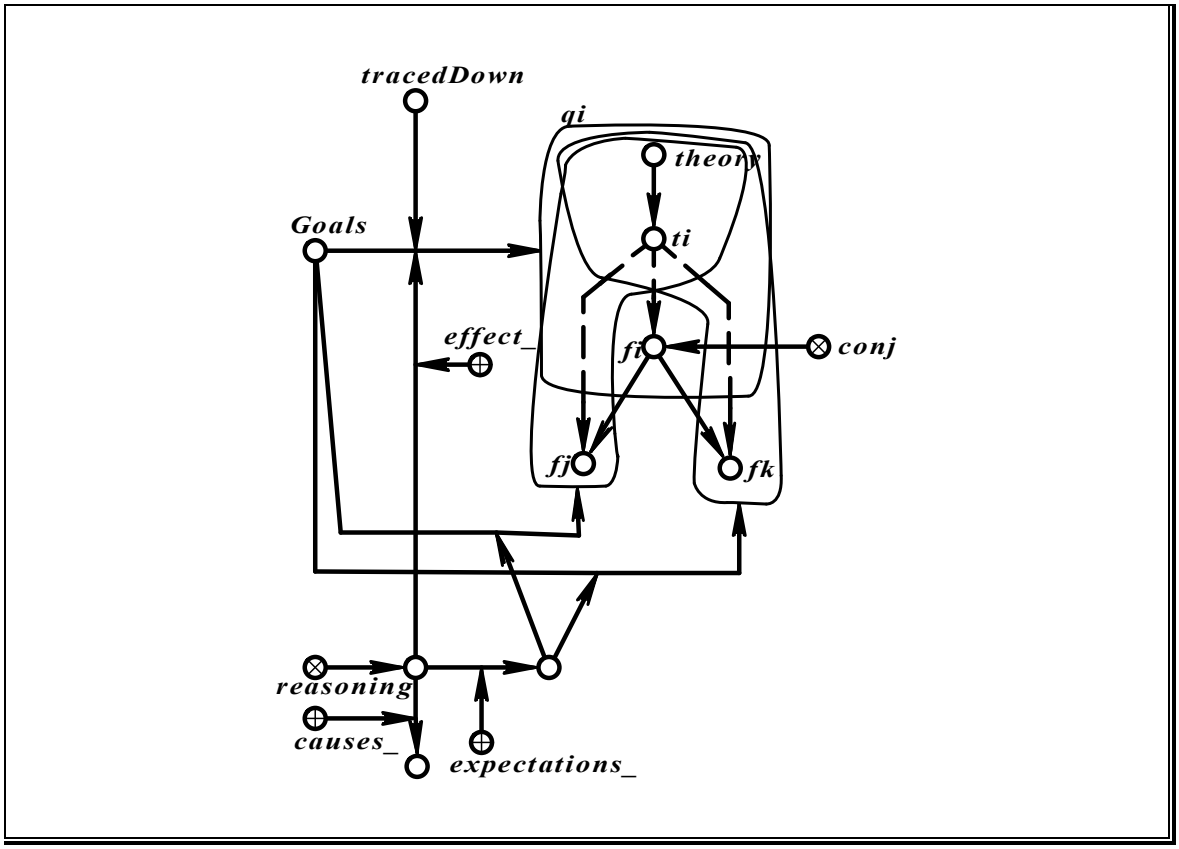
Результатом выполнения операции трассировки запроса сверху вниз в случае успеха является некоторое количество подзапросов сгруппированных связками отношения $reasoning$. При

завершении операции трассировки запроса сверху вниз соответствующая дуга из ключевого узла ***traceDown*** удаляется.

Приведём пример выполнения данной операции. При наличии в базе знаний конструкции:

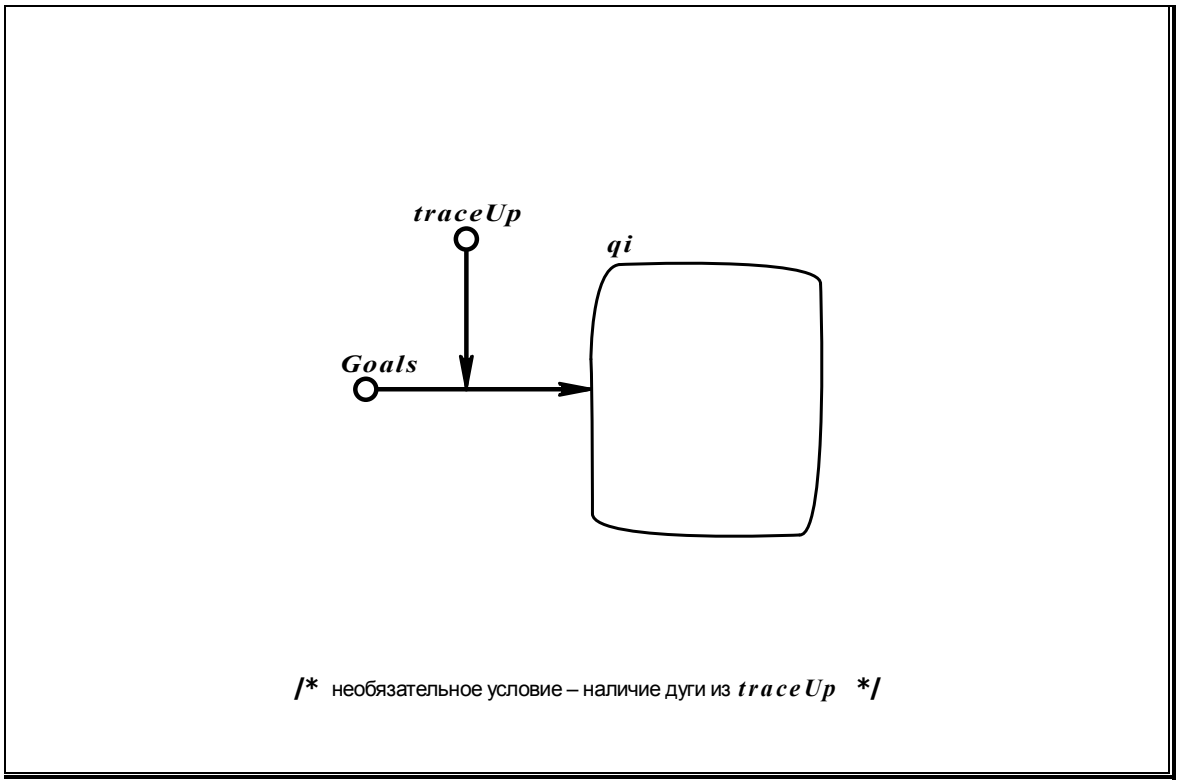


после выполнения операции мы получим следующую конструкцию:



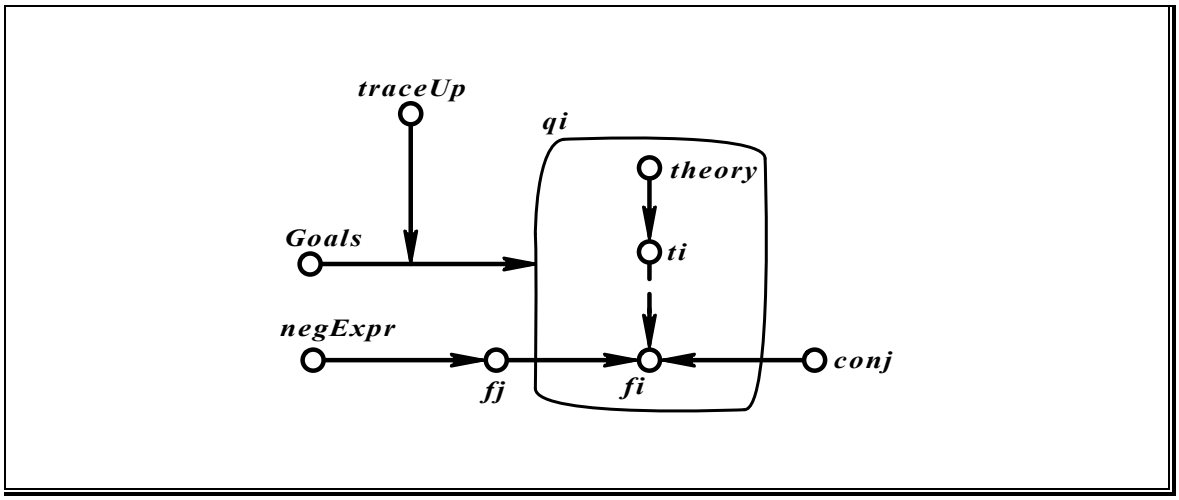
Операция трассировки запроса снизу вверх (SCL query trace-up operation) в зависимости от структуры запроса и типа формулы, к которой относится запрос, формирует запросы для всех формул, включающих её.

Условием выполнения операции трассировки запроса снизу вверх является наличие конструкции вида:

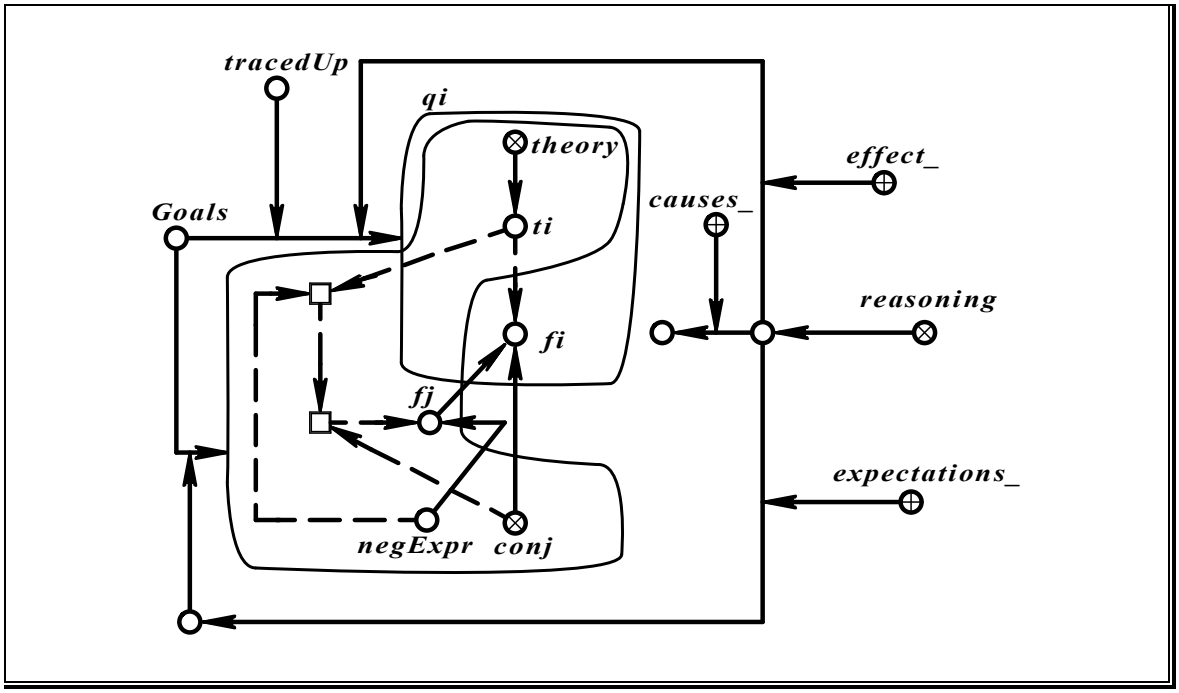


Операция в зависимости от типа запроса – запрос ли это на выяснение истинности, либо ложности исходного высказывания, либо запрос на поиск интерпретации исходной формулы – и типа исходной формулы, к которой относится запрос, формирует все возможные подзапросы к формулам, включающим исходную, и их остальным подформулам, группируя по И-критерию сформированные подзапросы связками отношения ***reasoning***.

Приведём пример выполнения данной операции. При наличии в базе знаний конструкции:

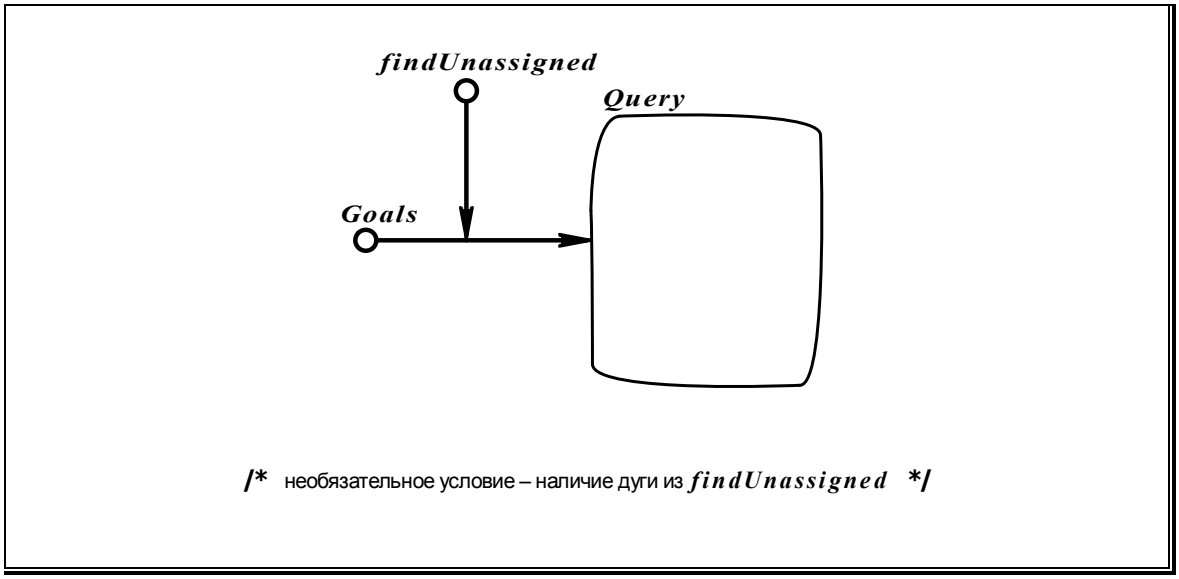


после выполнения операции мы получим следующую конструкцию:



Операция выявления множества свободных или связываемых переменных (SCL free variable locating operation) .

Условием выполнения операции формирования множества свободных или связываемых переменных является наличие конструкции вида:



Операция ищет неатомарную формулу связанную с запросом. Ищет множество свободных переменных: если такое множество есть, то операция завершается безуспешно, иначе она осуществляет поиск свободных переменных (которые имеют вхождения в каждую подформулу, включаемую текущей формулой) и включает эти переменные в соответствующее множество, которое привязывается к текущему высказыванию. При необходимости операция применяется к нахождению свободных переменных в подформулах на нижних уровнях.

Приведём пример выполнения данной операции. При наличии в базе знаний конструкции:

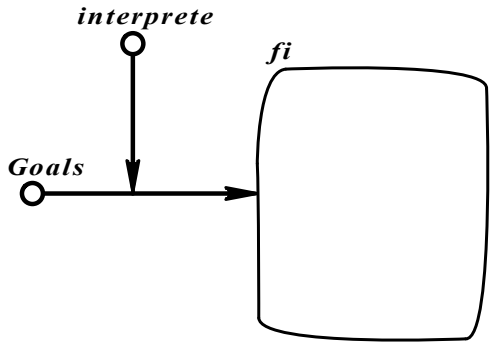
```
findUnassigned !; ( Goals !; qi );
qi = [unassigned !;  ( formula !; fi ), _Set ; ];
allEqExpr !; qi ;
qi !; fk , fj ;
fi = [рефлексивное множество !; _s ; ];
fk = [ _s !;; _s ; ];
existAtExpr !; fj , fk ;
```

после выполнения операции мы получим следующую конструкцию:

```
unassigned !;  ( formula !; fi ), Set ;
Set = [ _s ];
```

Операция формирования интерпретации формулы (SCL interpretation forming operation) .

Условием выполнения операции формирования интерпретации формулы является наличие конструкции вида:

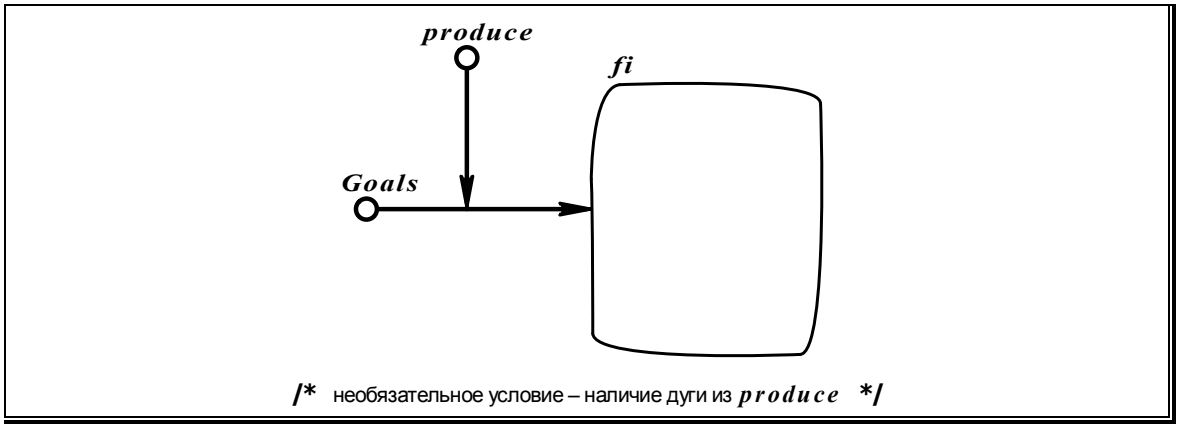


/* необязательное условие – наличие дуги из *interprete* */

Формируется новая, отличная от уже сформированных, интерпретация атомарной формулы ***fi***, используя все подтверждённые связки (связки, в которых компонент с атрибутом ***expectations_*** представляет собой пустое множество) отношения ***reasoning***, на базе существующих интерпретаций И-подцелей исходной цели, выражаемой атомарной формулой ***fi***, подцелей связанных между собой используемыми связками отношения ***reasoning***.

Операция порождения интерпретации атомарной формулы (SCL interpretation producing operation) .

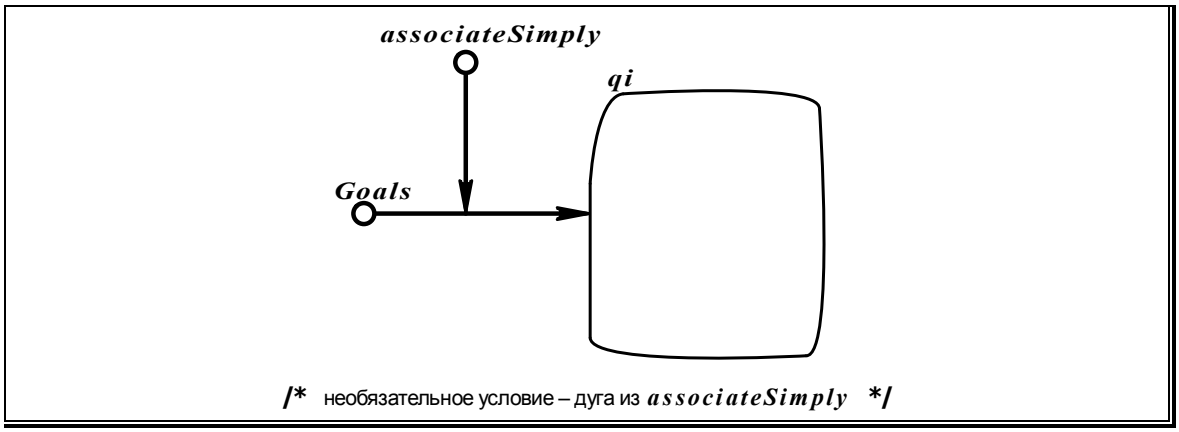
Условием выполнения операции порождения интерпретации атомарной формулы является наличие конструкции вида:



Если атомарная формула ***fi*** является высказыванием, которое обладает истинностным значением, то порождается релевантная ей формула, которая включается в также формируемую связку отношения интерпретации.

Операция поиска семантически близких атомарным формулам формул без учёта значений переменных (SCL simple atom associating operation) .

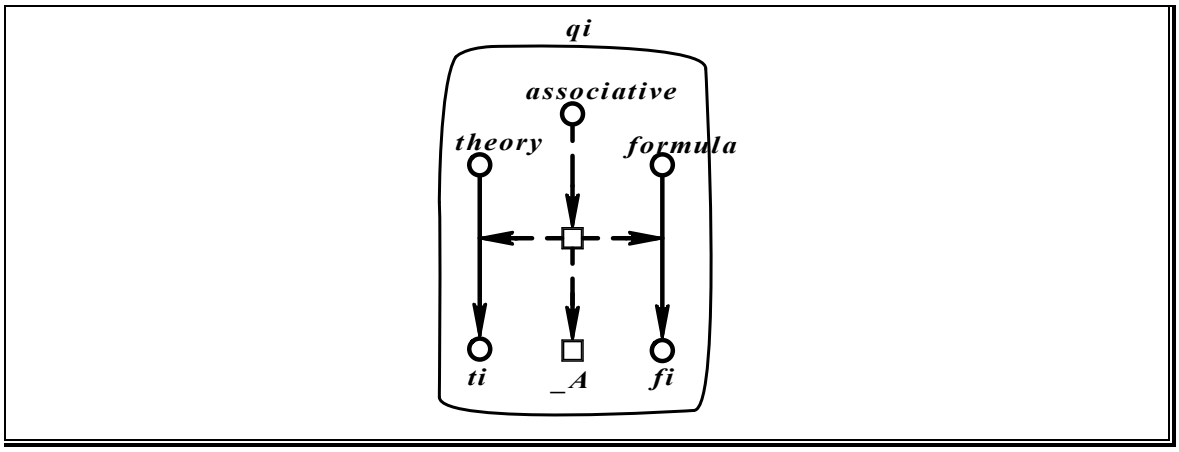
Условием выполнения операции поиска семантически близких атомарным формулам формул без учёта значений переменных является наличие конструкции вида:



Операция осуществляет поиск всех атомарных формул, включающих те же константы, что и исходная. Формируется соответствующая связка отношения ***associative***.

Микропрограмма операции поиска семантически близких атомарным формулам формул без учёта значений переменных имеет следующий вид:

Шаг 1. Вначале операция проверяет структуру запроса: в запросе отыскиваются соответствующая атомарная формула (***fi***) и теория (***ti***) , а в качестве допустимой разрешается структура запроса только следующего вида:



Шаг 2. В ходе проверки типа запроса находится дуга (*Goals* ;! *gq* ;! *qi*) и удаляется входящая в неё дуга из узла *associateSimply*. Если анализ структуры запроса прошёл успешно, то генерируется узел *s* , иначе – операция завершается с возвратом ошибки.

Шаг 3. Формируется множество всех констант атомарной формулы *fi*.

Шаг 4. Осуществляется формирование множества атомарных формул, включающих хотя бы одну константу из множества констант, сформированного на третьем шаге.

Шаг 5. Ищется формула *fj* , входящая во множество атомарных формул, сформированное на четвёртом шаге: если такой формулы нет, то осуществляется переход на шаг 15.

Шаг 6. Найденная формула *fj* исключается из множества формул, которое было сформировано на четвёртом шаге и формируется следующая конструкция:

```
sj , sk !; fj ;
```

Шаг 7. Формируется множество *si* – множество неатомарных формул, включающих хотя бы одну формулу из сформированного множества *sj* , и не включённых во множество *sk*.

Шаг 8. Если во множестве *si* находится формула *ti* , то осуществляется переход на шаг 13.

Шаг 9. Если множество *si* – пустое, то осуществляется переход на шаг 5.

Шаг 10. Элементы множества *sj* добавляются ко множеству *sk*. После чего множество *sj* очищается.

Шаг 11. Элементы множества *si* добавляются ко множеству *sj*. После чего множество *si* очищается.

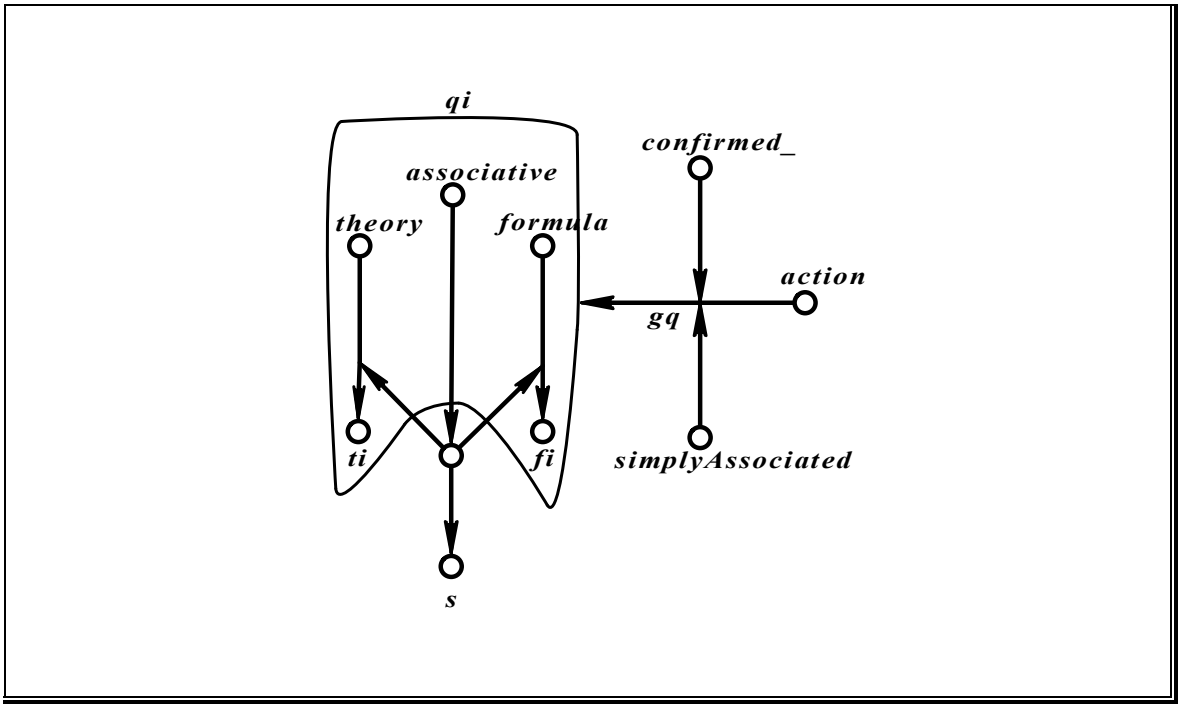
Шаг 12. Осуществляется переход на шаг 7.

Шаг 13. Формула *fj* добавляется во множество *s* – формируется конструкция вида:

```
s !; fj ;
```

Шаг 14. Осуществляется переход на шаг 5.

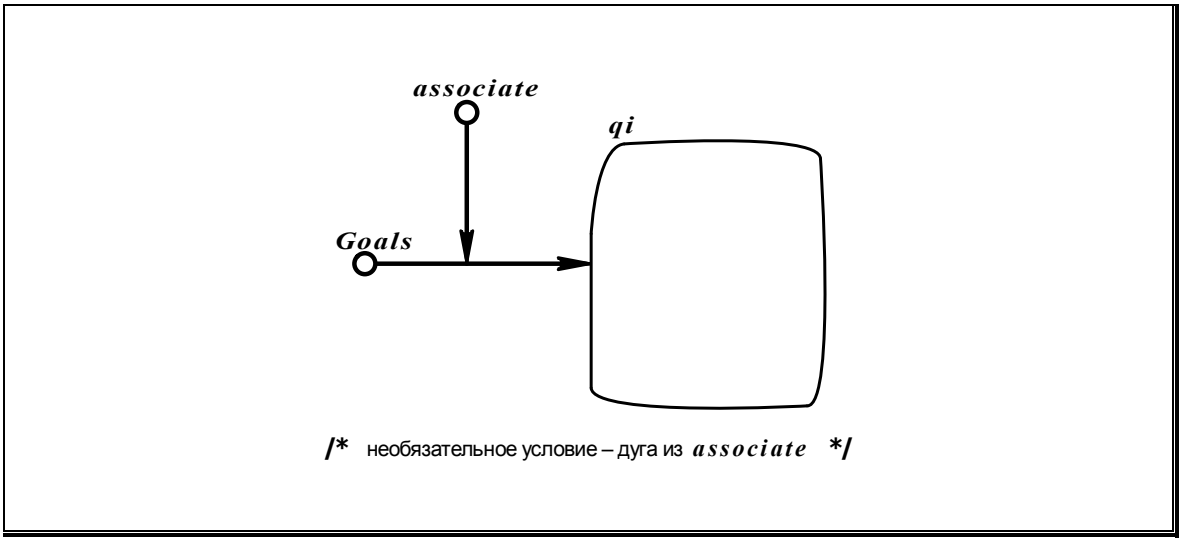
Ш а г 1 5 . Операция успешно завершается с формированием конструкции вида:



К о н е ц м и к р о п р о г р а м м ы .

Операция поиска семантически близких неатомарным формулам формул (SCL molecular associating operation) .

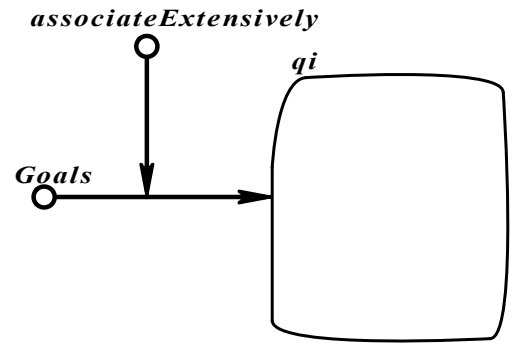
Условием выполнения операции поиска семантически близких неатомарным формулам формул является наличие конструкции вида:



Операция осуществляет поиск всех логически тождественных формул, включающих все те же (идентичные) формулы, что и исходная формула. Все найденные формулы включаются в формируемую связку отношения ***associative***.

Операция поиска семантически близких атомарным формулам формул с учётом значений переменных (SCL extensive atom associating operation) .

Условием выполнения операции поиска семантически близких атомарным формулам формул с учётом значений переменных является наличие конструкции вида:

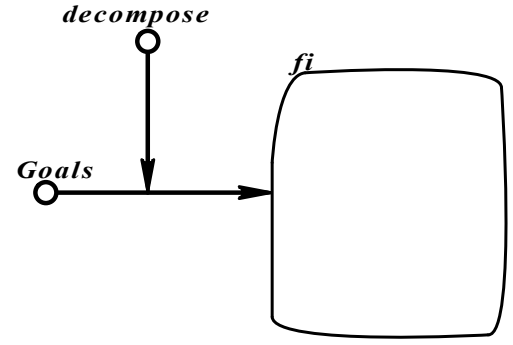


/* необязательное условие – дуга из *associateExtensively* */

Операция осуществляет поиск всех атомарных формул, включающих те же константы и те же значения переменных, что и исходная формула ***fi***, включённая в запрос ***qi***. Формируется соответствующая связка отношения ***associative***.

Операция декомпозиции запроса на вывод атомарной формулы, на запросы к семантически близким атомарным формулам (SCL atom query distributing operation) .

Условием выполнения операции декомпозиции запроса на вывод атомарной формулы, на запросы к семантически близким атомарным формулам является наличие конструкции вида:



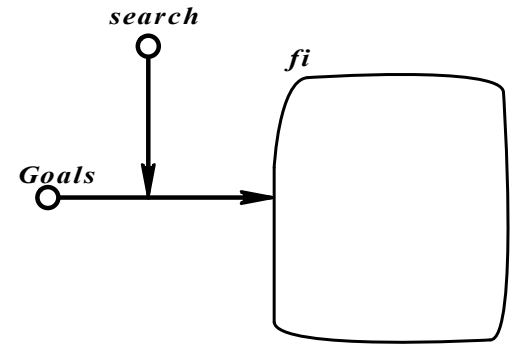
/* необязательное условие – дуга из *decompose* */

Операция ставит соответствующий запрос к подходящим семантически близким формулам в зависимости от типа запроса к исходной атомарной формуле ***fi***. Между исходным и производными запросами формируются связки отношения ***reasoning***. Результатом будет генерация новых запросов и конструкций вида:

```
reasoning !;□effect_:gq1,expectations_:□gx1□,causes_:C1□;
Goals ;! gx1 ;! q1 ;
active_ !;gx1 ;
```

Операция эпизодического информационного поиска (SCL atom episodic confirm_ation-by-facts operation) .

Условием выполнения операции эпизодического информационного поиска является наличие конструкции вида:



/* необязательное условие – дуга из ***search*** ***/**

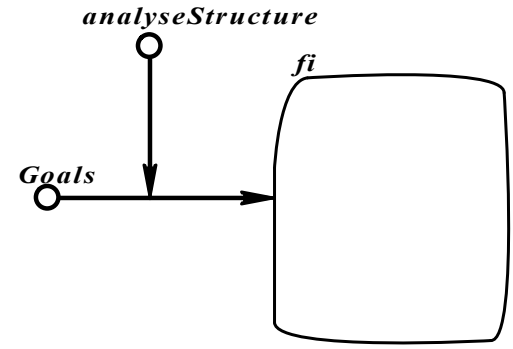
Операция осуществляет информационный поиск конструкции, соответствующей включённой в запрос ***gq*** формуле ***fi*** в указанной области поиска. Поиск прерывается в случае достижения свободной переменной, которая не инцидентна ни одной связанной переменной. При поиске учитываются уже присвоенные значения переменных формулы ***fi*** и найденные интерпретации (отобранные образы) .

Новый образ добавляется к отобранным и формируется запрос на формирование новой интерпретации. Результатом успешного завершения операции является конструкция вида:

factual !; ***fi*** **;**

Операция выявления идентичных, **входящих и конфликтующих атомарных формул** (SCL atom correlating operation) .

Условием выполнения операции выявления идентичных, входящих и конфликтующих атомарных формул является наличие конструкции вида:



/* необязательное условие – дуга из ***analyseStructure*** ***/**

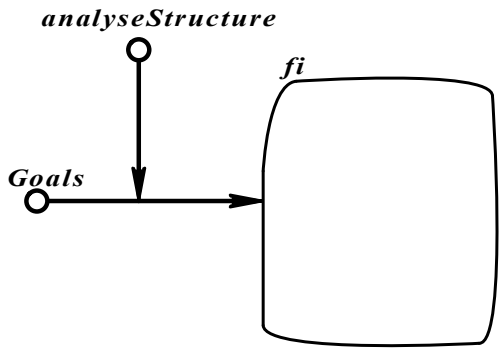
Операция анализирует структуру атомарных формул входящих в связку ***qi*** отношения ***associative*** и формирует связки отношения бинарного структурного включения ***inclusion*** между атомарными формулами. Если две формулы взаимно включают друг друга по структуре, и кванторы, которые навешены на переменные формул совпадают, то между такими формулами формируется связка отношения синонимии.

Результатом применения операции могут быть конструкции вида:

```
inclusion !; □ ( formula !; Formula1 ), in_ : ( formula !; Formula2 ) □ ;
Formula1 = Formula2 ;
```

Операция выявления идентичных, **входящих и конфликтующих неатомарных формул** (SCL molecular correlating operation) . Аналогична предыдущей.

Условием выполнения операции выявления идентичных, входящих и конфликтующих неатомарных формул является наличие конструкции вида:



/* необязательное условие – дуга из *analyseStructure* */

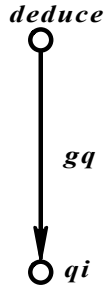
Операция анализирует структуру неатомарных формул входящих в связку *qi* отношения *associative* и формирует связки отношения бинарного структурного включения *inclusion* между формулами. Если две формулы взаимно включают друг друга по структуре, и кванторы, которые навешены на переменные формул совпадают, то между такими формулами формируется связка отношения синонимии.

Результатом применения операции могут быть конструкции вида:

```
inclusion !; □ ( formula !; Formula1 ), in_ : ( formula !; Formula2 ) □ ;
Formula1 = Formula2 ;
```

Операция выполнения логических рассуждений (SCL reasoning operation) .

Условием выполнения операции выполнения логических рассуждений является наличие конструкции вида:



/* необходимое условие – дуга *gq* */

Операция находит элемент *vx* включённой в связку *qi* отношения *reasoning*, помеченный атрибутом *expectations_*, проверяет являются ли все элементы найденного множества *vx* запросами, на

которые получен ответ и если это так, то операция находит ***ga*** – компонент связки ***qi***, помеченный атрибутом ***effect_***. Если ***ga*** включён во множество ***deny_***, то он также включается во множество ***denied_***. Если ***ga*** включён во множество ***confirm_***, то он также включается во множество ***confirmed_***. Если связка ***qi*** включена во множество временных конструкций, то она удаляется.

Микропрограмма операции выполнения логических рассуждений имеет следующий вид:

Шаг 1. Вначале операция проверяет структуру запроса: в качестве допустимой разрешается структура запроса только следующего вида, когда узел запроса является знаком связки отношения ***reasoning***:

reasoning !; ***qi*** **;**

Шаг 2. Если проверка структуры запроса осуществлена безуспешно, то операция завершается с возвратом ошибки, иначе – удаляется дуга ***gq***.

Шаг 3. Отыскивается константная позитивная дуга ***ge***, выходящая из узла ***qi*** и помеченная атрибутом ***effect_***. Если поиск осуществлён безуспешно, то операция завершается с возвратом ошибки.

Шаг 4. Ищется константная позитивная дуга ***ga***, выходящая из узла ***Goals***, входящая в искомый константный узел ***va***, в которую входит дуга ***ge***. Если поиск осуществлён безуспешно, то операция завершается с возвратом ошибки.

Шаг 5. Ищется константная позитивная дуга, выходящая из узла ***qi***, входящая в искомый константный узел ***vx*** и помеченная атрибутом ***expectations_***. Если поиск осуществлён безуспешно, то осуществляется переход на шаг 7.

Шаг 6. Во множестве ***vx*** отыскиваются элементы, которые не входят ни во множество ***denied_***, ни во множество ***confirmed_***. Если найден хотя бы один такой элемент, то операция завершается успешно, иначе – удаляется узел ***vx***.

Шаг 7. Если дуга ***ga***, включена хотя бы в одно из множеств ***denied_*** и ***confirmed_***, то операция завершается успешно.

Шаг 8. Если дуга включена во множество ***confirm_***, то осуществляется генерация конструкции, удовлетворяющей образцу задаваемому узлом ***va***. Генерируемая конструкция включается в формулу, являющуюся соответствующим фактографическим высказыванием (либо квазиатомарной логической формулой), формируется полное соответствие интерпретации между образцом ***va*** и сгенерированной формулой. В дугу ***va*** генерируется дуга из узла ***confirmed_***.

Шаг 9. Если дуга включена во множество ***deny_***, то осуществляется генерация квазиатомарной формулы ***fj***, удовлетворяющей образцу задаваемому узлом ***va***. Генерируемая конструкция включается в формулу ***fj***, формируется полное соответствие интерпретации между образцом ***va*** и сгенерированной формулой ***fj***. В дугу ***va*** генерируется дуга из узла ***denied_***.

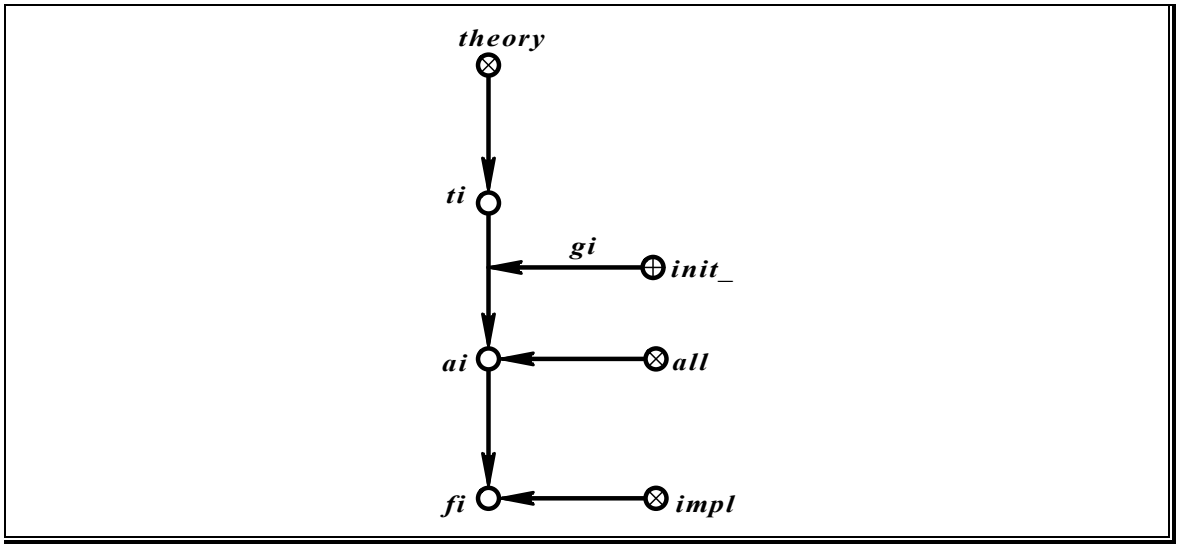
Шаг 10. При наличии константной позитивной дуги из узла ***temporary*** в узел ***qi***. Генерируется константная позитивная дуга из узла ***release*** в узел ***qi***.

Шаг 11. Операция завершается успешно.

Конец микропрограммы.

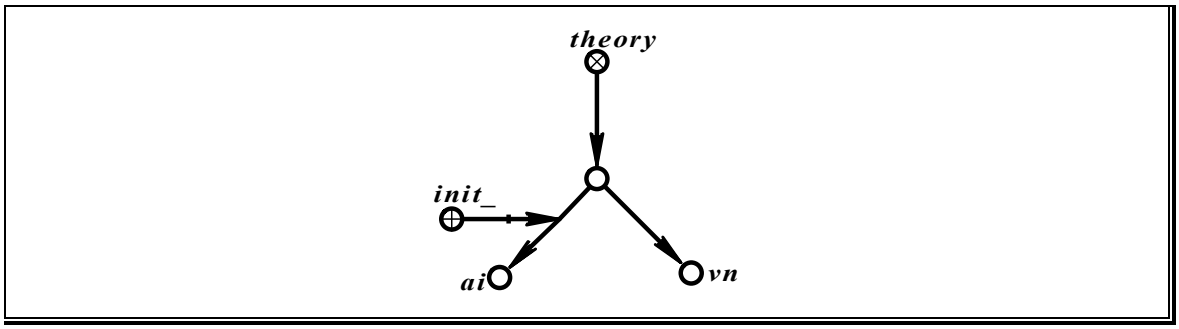
Операция реализации продукции (SCL producing operation).

Условием выполнения операции реализации продукции является наличие sc-конструкции вида:

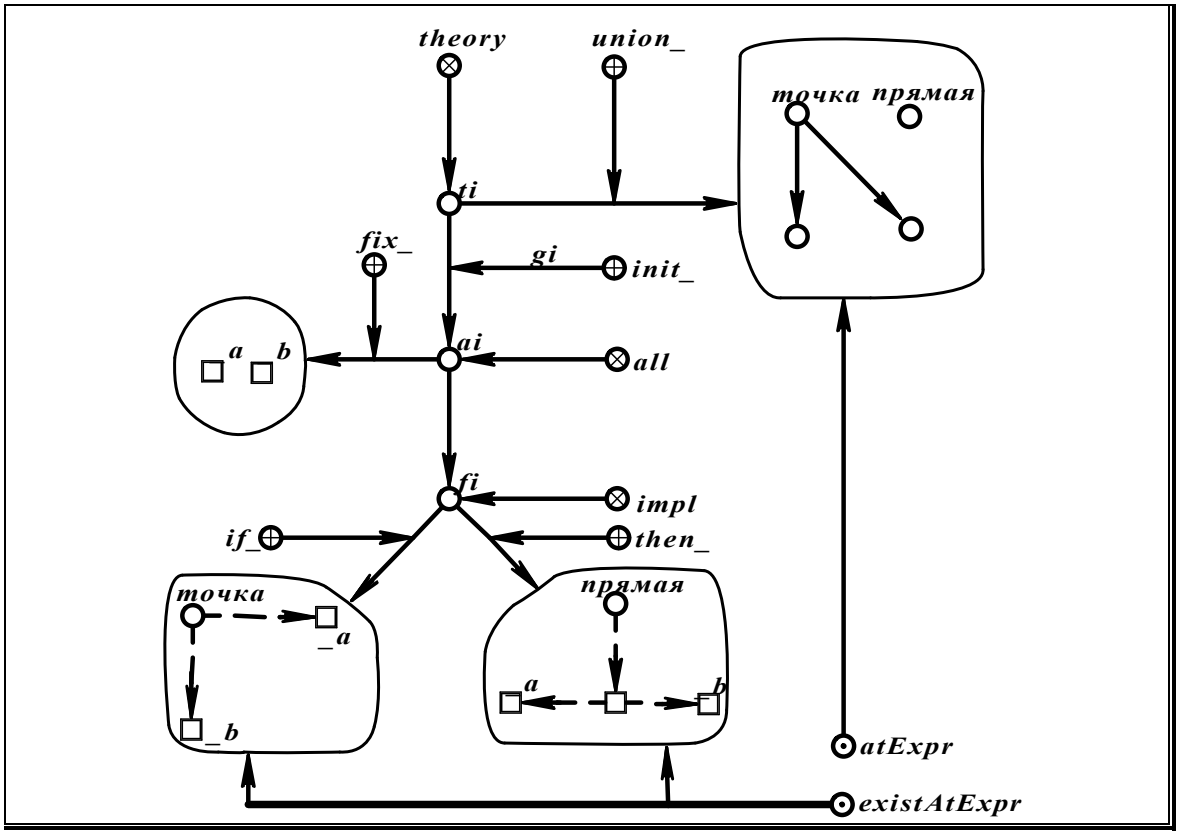


Операция находит левую часть импликативной формулы ***fi*** и ищет формулу ***ri*** релевантную этой части. Если левая часть является атомарной формулой, то поиск новой формулы заключается в поиске структуры, которая является фрагментом обобщённого атомарного (фактографического) высказывания теории ***ti***, структуры, изоморфной этой атомарной формуле с учётом привязки констант, и формировании узла, из которого проводятся константные позитивные дуги во все найденные элементы структуры, являющегося знаком искомой формулы. Если же левая часть является сложной формулой, то осуществляется поиск релевантной ей формулы, которая включена во множество ***ti***. Между левой частью импликации и найденной релевантной ей формулой ***ri*** формируется конструкция отношения релевантности, в рамках которой каждая переменная формулы, являющейся левой частью импликации получает своё значение – константу, включённую в какую-либо атомарную формулу формулы ***ri***. Затем операция находит правую часть формулы ***fi***, которая является атомарной формулой и формирует релевантную ей формулу ***vn***, являющуюся фактографическим высказыванием, структура которого изоморфна правой части импликации с учётом привязки констант. Затем все значения тех переменных из найденных левой и правой частей импликации, которые связаны квантором общности в рамках кванторной формулы ***ai***, склеиваются. Дуга ***gi*** удаляется. Элементы сформированной формулы ***vn*** добавляются в обобщённое атомарное (фактографическое) высказывание теории ***ti***, а сама формула включается в теорию ***ti***.

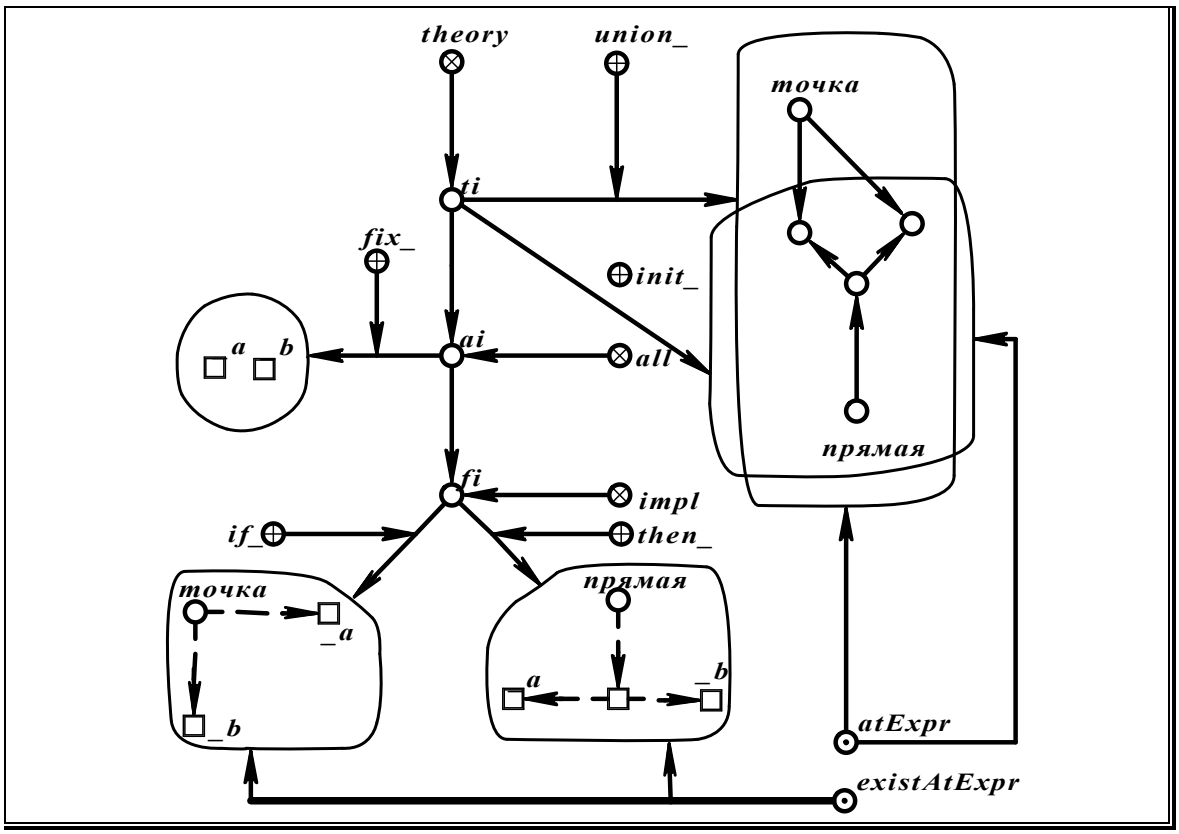
Результатом выполнения операции реализации продукции является следовательно конструкция вида:



Приведём пример выполнения данной операции. При наличии в базе знаний конструкции:



после выполнения операции реализации продукции мы получим следующую конструкцию:



Микропрограмма операции реализации продукции имеет следующий вид:

Шаг 1. Вначале операция осуществляет поиск неизвестных элементов конструкции, являющейся условием выполнения этой операции. Такими неизвестными элементами являются все элементы вышеприведённой конструкции-условия за исключением ключевых узлов: **"*theory*"**, **"*init_*"**, **"*all*"** и **"*impl*"**. В ходе выполнения этого шага операция находит, в частности, такие соответствующие элементы вышеприведённой конструкции-условия как: константную позитивную дугу ***gi***, выходящую из ключевого узла **"*init_*"**, константный узел ***ti***, являющийся знаком формальной теории, константный узел ***ai***, являющийся знаком кванторной формулы, являющейся в рамках теории ***ti*** истинным высказыванием о всеобщности, а также константный узел ***fi***, являющийся знаком импликативной формулы.

Шаг 2. Если хотя бы один из элементов ***gi***, ***ti***, ***ai*** и ***fi*** или хотя бы одна соответствующая константная позитивная дуга из вышеприведённой конструкции не найдена на предыдущем шаге, то операция завершает свою работу, иначе – удаляется дуга ***gi***.

Шаг 3. Осуществляется поиск константного узла ***vc***, в который входит константная позитивная дуга, выходящая из узла ***fi***, которая является элементом ключевого множества **"*if_*"**. Из этого следует, что искомый узел ***vc*** является посылкой импликативной формулы ***fi***. Таким образом формула ***fi*** и её посылка связаны между собой конструкцией вида:

fi !; ***if_*** **:** ***vc*** **;**

Шаг 4. Осуществляется поиск константного узла ***ve***, в который входит константная позитивная дуга, выходящая из узла ***fi***, которая является элементом ключевого множества **"*then_*"**. Из этого следует, что искомый узел ***vc*** является следствием импликативной формулы ***fi***. Таким образом формула ***fi*** и её следствие связаны между собой конструкцией вида:

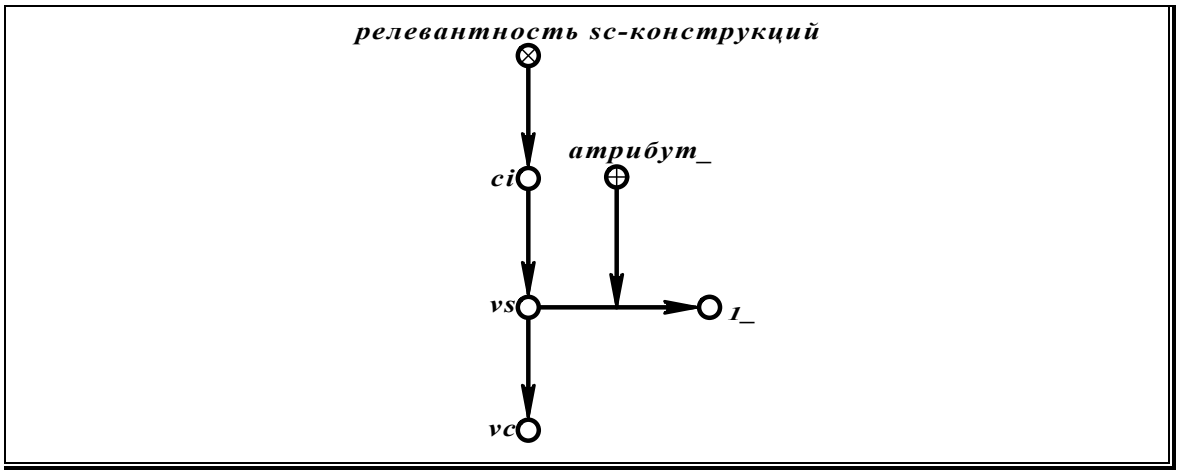
fi !; ***then_*** **:** ***ve*** **;**

Шаг 5. Осуществляется поиск константного узла ***vf***, в который входит константная позитивная дуга, выходящая из узла ***ai***, которая является элементом ключевого множества **"*fix_*"**. Из этого следует, что искомый узел ***vf*** является множеством переменных связываемых квантором общности в рамках кванторного высказывания о всеобщности ***ai***. Таким образом искомое множество связываемых переменных и кванторная формула ***ai*** связаны между собой конструкцией вида:

ai !; ***fix_*** **:** ***vf*** **;**

Шаг 6. Проверяется наличие константной позитивной дуги, выходящей из узла, являющегося одним из двух таких ключевых узлов как: **"*atExpr*"** и **"*existAtExpr*"**, и входящей в узел ***vc***: если такой дуги нет, то осуществляется переход на шаг 30, иначе – наличие такой дуги означает что, посылка ***vc*** является атомарной формулой, либо кванторной формулой о существовании интерпретации атомарной формулы.

Шаг 7. Осуществляется поиск связки ***ci*** отношения **"*релевантность sc-конструкций*"**, которая включает узел ***vs***, являющийся знаком связки, включающего формулу ***vc*** и узел **"*1_*"** под атрибутом **"*атрибут_*"**. Таким образом искомая связка и известные формула ***vc*** и отношение **"*релевантность sc-конструкций*"** связаны конструкцией вида:



Ш а г 8 . Если такая связка не найдена, то осуществляется переход на шаг 14.

Ш а г 9 . Осуществляется поиск узла ***ri*** отличного от узла ***vc***, который включён в одно из таких множеств как: **"*atExpr*"** и **"*existAtExpr*"**, который также включён в искомую пару ***sd*** одновременно с узлом**"*2_*"** под атрибутом **"*атрибут_*"**, в которую проведена константная позитивная дуга, выходящая из связки ***ci*** и не включённая во множество **"*отношение_*"**. Если узел ***ri*** не найден, то осуществляется переход на шаг 14.

Ш а г 1 0 . Осуществляется поиск узла ***ii***, в который проведена константная позитивная дуга из узла ***ci***, в которую входит константная позитивная дуга из ключевого узла **"*отношение_*"**. Если узел ***ii*** найден, то осуществляется переход на шаг 18.

Ш а г 1 1 . Элементы множества ***ri*** и только они включаются во множество, обозначенное узлом ***uf***.

Ш а г 1 2 . Множество ***ri*** очищается, то есть удаляются все выходящие из него константные позитивные дуги.

Ш а г 1 3 . Осуществляется переход на шаг 16.

Ш а г 1 4 . Осуществляется поиск всех элементов, в которые входят константные позитивные дуги из узла ***vc***, если ни один такой элемент не найден, то операция завершается с возвратом ошибки.

Ш а г 1 5 . Осуществляется поиск константного узла ***uf***, в который входит константная позитивная дуга, выходящая из узла ***ti***, которая является элементом ключевого множества **"*union_*"**. Из этого следует, что искомый узел ***uf*** является обобщённым атомарным **(**фактографическим***)*** высказыванием формальной теории ***ti***. Таким образом теория ***ti*** и формула ***uf*** связаны между собой конструкцией вида:

ti !; ***union_*** **:** ***uf*** **;**

Ш а г 1 6 . Осуществляется поиск конструкции, являющейся фрагментом конструкции, включенной во множество ***uf***, – поиск фрагмента изоморфного (с учётом привязки констант) конструкции, которую образуют все элементы, найденные на четырнадцатом шаге. Искомый фрагмент – искомая конструкция будет являться изоморфной с учётом привязки констант в том и только в том случае, если каждому переменному узлу или переменному элементу неопределённого типа, найденному на четырнадцатом шаге соответствует некоторый константный узел или константный элемент соответственно, каждому константному узлу, дуге или элементу неопределённого типа – он сам, каждой переменной дуге – константная дуга аналогичной степени чёткости, выходящая из узла искомой конструкции, соответствующего узлу, из которого выходит переменная дуга, и входящая в элемент искомой конструкции, соответствующий элементу, в который входит переменная дуга, когда других элементов в искомой конструкции нет, причём для каждого элемента, найденного на

четырнадцатом шаге соответствует один единственный элемент искомой конструкции. Все элементы искомой конструкции и только они включаются во множество, обозначенное узлом ***ri***. Каждый элемент искомой конструкции ***es***, соответствующий какому-либо найденному на четырнадцатом шаге переменному элементу ***ei***, связывается с этим переменным элементом ориентированной парой – парой вида:

□ *1_* : ***ei*** , *2_* : ***es*** □;

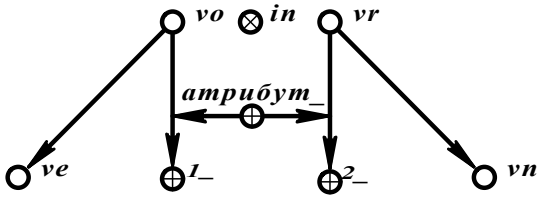
Шаг 17. Если поиск изоморфной конструкции прошёл безуспешно, то осуществляется переход на шаг 28, иначе – все сформированные на предыдущем шаге ориентированные пары и только они включаются во множество, обозначенное узлом ***ii***.

Шаг 18. Если узел ***ci*** не был найден (на седьмом шаге), то генерируется связка отношения **"*релевантность sc-конструкций*"**, в которую включаются узлы ***ri*** и ***vc*** и только. Формула ***ri*** добавляется во множество ***ti***.

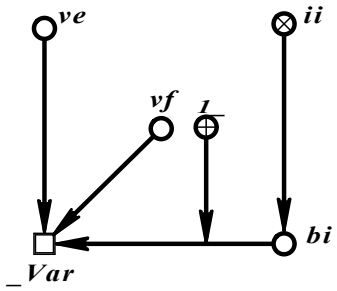
Шаг 19. В случае отсутствия, во множество ***ci*** добавляется узел ***ii*** под атрибутом **"*отношение_*"**.

Шаг 20. Проверяется наличие константной позитивной дуги, выходящей из узла, являющегося одним из двух таких ключевых узлов как: **"*atExpr*"**, **"*existAtExpr*"**, и входящей в узел ***ve***: если такой дуги нет, то осуществляется переход на шаг 28, иначе – наличие такой дуги означает что, посылка ***ve*** является атомарной формулой, либо кванторной формулой о существовании интерпретации атомарной формулы.

Шаг 21. Генерируются узлы ***vn***, ***vr***, ***vo*** и ***in***. Генерируется конструкция вида:



Шаг 22. Во множество ***in*** включается каждая ориентированная пара ***bi***, включённая во множество ***ii*** и имеющая в качестве первого элемента какую-либо переменную ***_Var***, включённую во множества ***vf*** и ***ve***, то есть пара, связанная конструкцией вида:



Шаг 23. Во множество ***vn*** включается каждый элемент ***ev***, являющийся вторым элементом хотя бы одной ориентированной пары ***bi***, включённой во множество ***in***, то есть элемент, связанный конструкцией вида:

in !; ***bi*** !; *2_* : ***ev*** ;

Шаг 24. Каждый константный элемент множества ***ve*** включается во множество ***vn***.

Ш а г 2 5 . Для каждого переменного узла или переменного элемента неопределённого типа, включённого во множество ***ve*** и не включённого во множество ***vf***, генерируется и включается во множество ***vn*** константный узел или константный элемент неопределённого типа соответственно. Для каждого генерируемого константного элемента генерируется включаемая во множество ***in*** ориентированная пара, включающая в качестве второго этот генерируемый элемент, а в качестве первого – переменный элемент, соответствующий генерируемому константному элементу.

Ш а г 2 6 . Для каждой переменной дуги, включённой во множество ***ve*** и не включённой во множество ***vf***, генерируется и включается во множество ***vn*** константная дуга аналогичной степени чёткости, выходящая из узла, лежащего во множестве ***vn***, соответствующего узлу, из которого выходит переменная дуга, и входящая в элемент, лежащий во множестве ***vn***, соответствующий элементу, в который входит переменная дуга. Для каждой генерируемой константной дуги генерируется включаемая во множество ***in*** ориентированная пара, включающая в качестве второго эту генерируемую дугу, а в качестве первого – переменную дугу, соответствующую генерируемой константной дуге.

Ш а г 2 7 . Генерируется связка отношения **“*релевантность sc-конструкций*”**, в которую включаются узлы ***vo*** и ***vr***, а также узел ***in*** под атрибутом **“*отношение_*”** и только. Формула ***vn*** добавляется во множества ***ti*** и **“*atExpr*”**. Все не добавленные элементы множества ***vn*** добавляются в обобщённое атомарное (фактографическое) высказывание ***uf*** формальной теории ***ti***, связанное с теорией ***ti*** конструкцией вида:

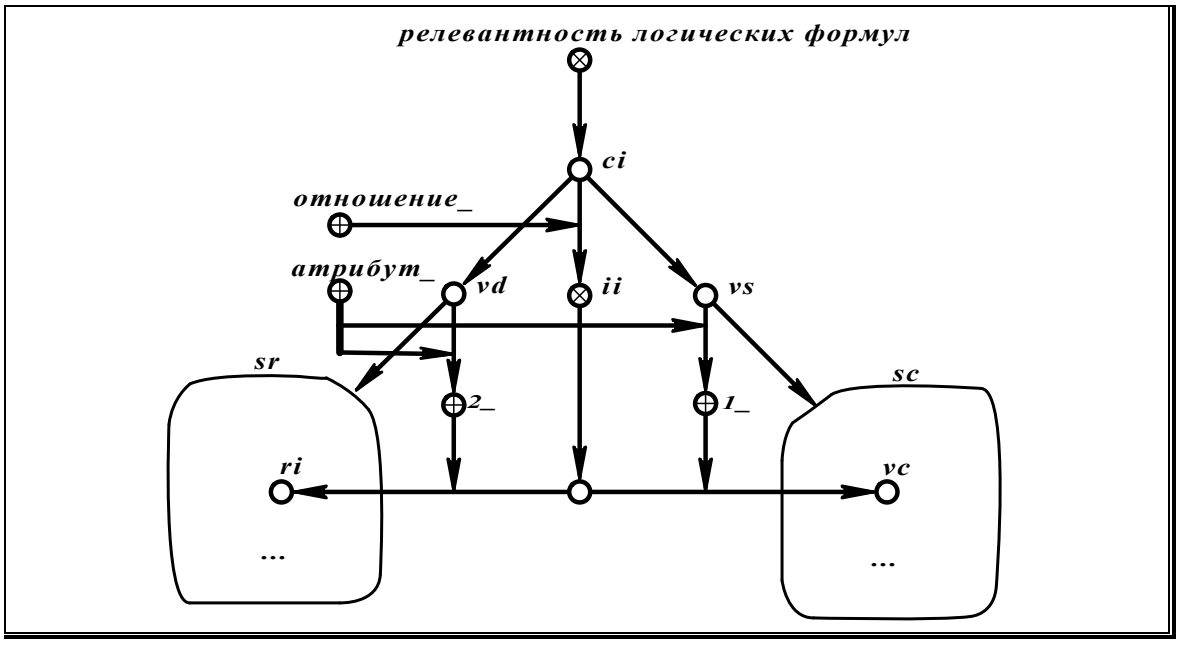
ti !; ***union_*** **:** ***uf*** **;**

Ш а г 2 8 . Осуществляется переход на шаг 34.

Ш а г 2 9 . Если узел ***ri*** был найден на девятом шаге, то операция завершается с возвратом ошибки, иначе – осуществляется переход на шаг 28.

Ш а г 3 0 . Проверяется наличие константной позитивной дуги, выходящей из узла, являющегося одним из двух таких ключевых узлов как: **“*all*”**, **“*allEqExpr*”**, **“*allImpl*”**, **“*alt*”**, **“*conj*”**, **“*disj*”**, **“*exist*”**, **“*eqExpr*”** и **“*impl*”**, и входящей в узел ***ve***: если такой дуги нет, то осуществляется переход на шаг 28.

Ш а г 3 1 . Осуществляется поиск константного узла ***ri***, в который проведена константная позитивная дуга из узла ***ti***. Искомый узел ***ri*** кроме этого должен являться формулой релевантной формуле ***vc***. В процессе поиска формируется связка ***ci*** отношения **“*релевантность логических формул*”**, в которую включены узел ***vs***, являющийся знаком связки, в которую включен узел **“*1_*”** под атрибутом **“*атрибут_*”** и узел ***sc***, включающий всю структуру формулы ***vc***. В связку ***ci*** также должен быть включён узел ***vd***, являющийся знаком связки, в которую включен узел **“*2_*”** под атрибутом **“*атрибут_*”** и узел ***sr***, включающий всю структуру формулы ***ri***. Кроме этого в связку ***ci*** также должен быть включён узел ***ii*** под атрибутом **“*отношение_*”**, являющийся знаком множества, ориентированных пар соответствия релевантности, которые выходят из элементов множества ***sc*** в соответствующие элементы множества ***sr***. Таким образом искомые на данном шаге элементы в итоге будут связаны конструкцией вида:



Ш а г 3 2 . Если такой узел ***ri*** не найден, то осуществляется переход на шаг 28.

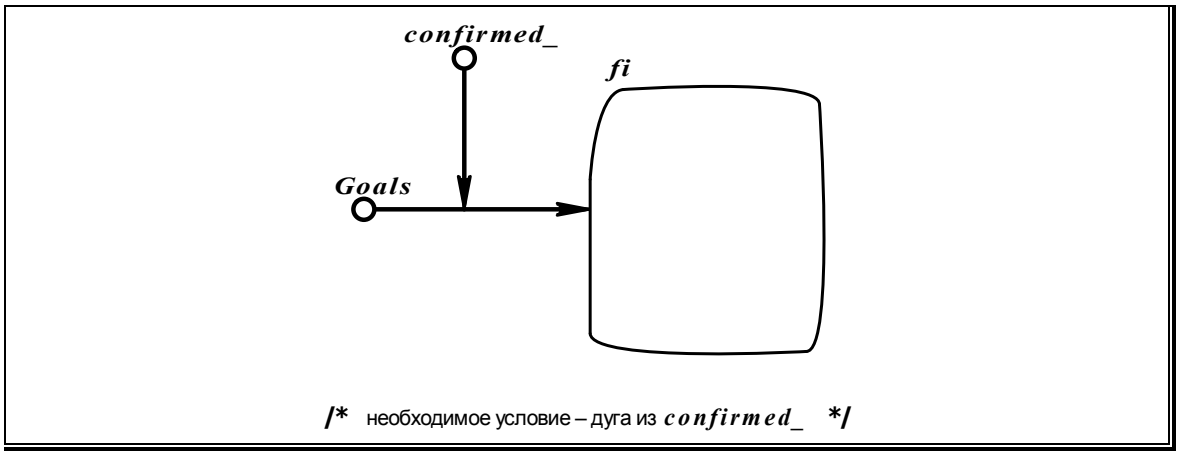
Ш а г 3 3 . Осуществляется переход на шаг 20.

Ш а г 3 4 . Работа операции успешно завершается.

К о н е ц м и к р о п р о г р а м м ы .

Операция обработки положительного ответа на запрос (SCL positive answer processing operation) .

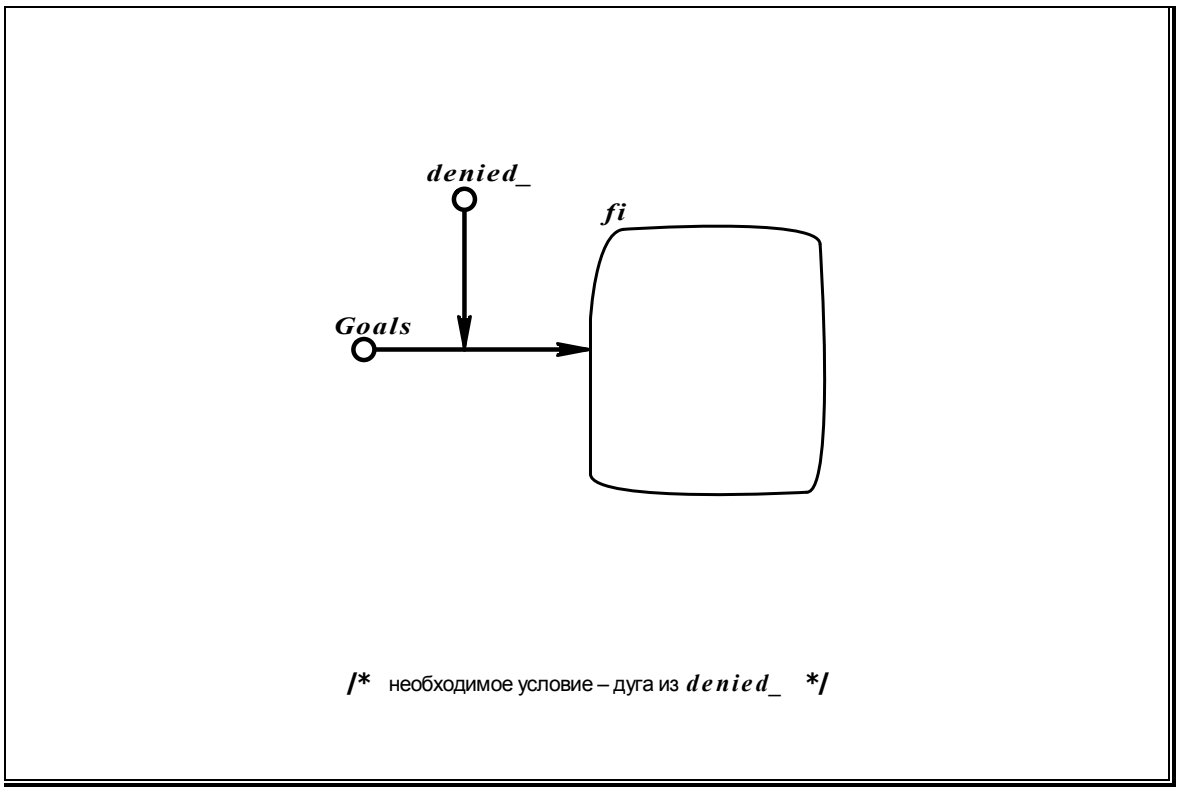
Условием выполнения операции обработки положительного ответа на запрос является наличие конструкции вида:



Операция анализирует: не является подтверждённый запрос какой-либо ожидаемой И-подцелью и если это так, то операция переносит подцель из разряда ожидаемых (из компонента соответствующей связки отношения ***reasoning***, помеченного атрибутом ***expectations_***) в достигнутые (в компонент этой связки, помеченный атрибутом ***causes_***), создавая условия для применения операции выполнения логических рассуждений.

Операция обработки отрицательного ответа на запрос (SCL negative answer processing operation) .

Условием выполнения операции обработки отрицательного ответа на запрос является наличие конструкции вида:



Операция анализирует: не является опровергнутый запрос какой-либо ожидаемой И-подцелью и если это так, то операция переносит подцель из разряда ожидаемых (из компонента соответствующей связки отношения ***reasoning***, помеченного атрибутом ***expectations_***) в достигнутые (в компонент этой связки, помеченный атрибутом ***causes_***), создавая условия для применения операции выполнения логических рассуждений.

Операция уничтожения промежуточных конструкций (SCL structure releasing operation) . Операция уничтожает все конструкции, которые включены во множество удаляемых конструкций.

Операция склейки идентичных формул (SCL formula optimizing operation) . Операция склеивает две синонимичные формулы.

Операция добавления факта (SCL fact adding operation) . Операция добавляет факт в SCL теорию, если он не противоречит другим фактам и высказываниям.

Операция добавления высказывания (SCL expression adding operation) . Операция добавляет высказывание в SCL теорию, если оно не противоречит фактам и другим высказываниям.

Более подробное рассмотрение операций абстрактной scl-машины приведено в работах [148; 150, 153] (***Голенков В.В..1995мо-ОпераЯSCLдОП3 ; Голенков В.В..1995мо-РешенНЯSCL3иОГ ; Голенков В.В..1996мо-БазовПТЯSCL***).

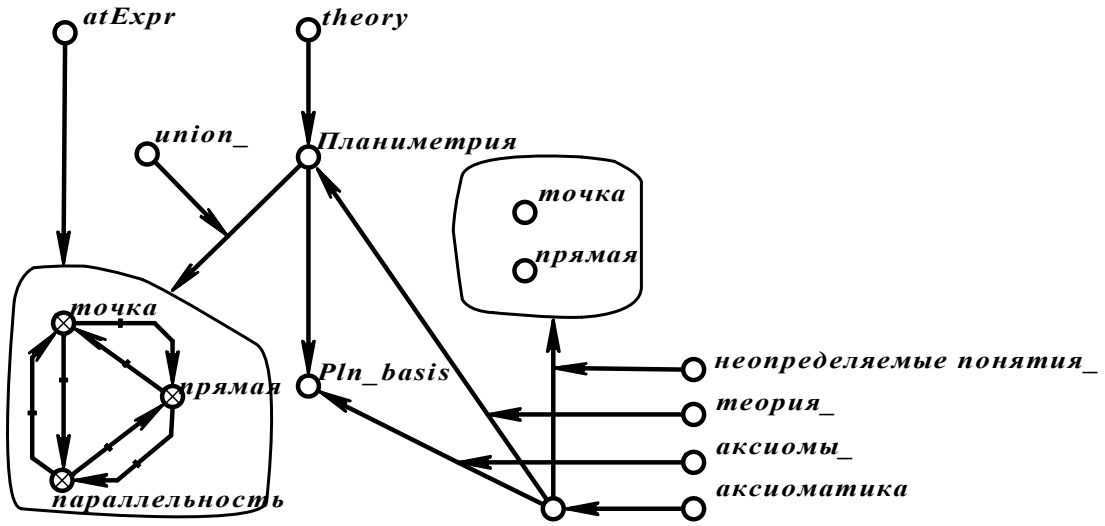
8.2. Решение задач в графодинамических ассоциативных машинах вывода

Ключевые понятия: задача, решение задачи, аксиома, определение, вывод.

Рассмотрим решение задач (см. уточнение понятия задача в подразделе 6.3) на SCL. Пусть нам необходимо доказать, что существуют две параллельные прямые. Мы обладаем следующей исходной информацией.

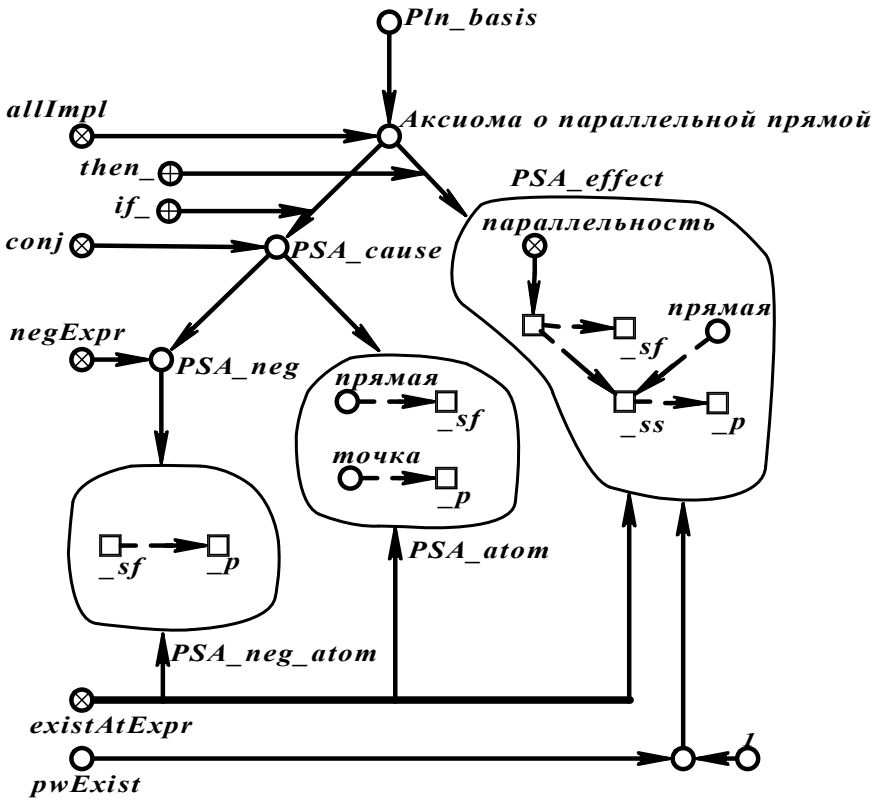
Условие задачи:

/* Описание формальной теории из области планиметрии */

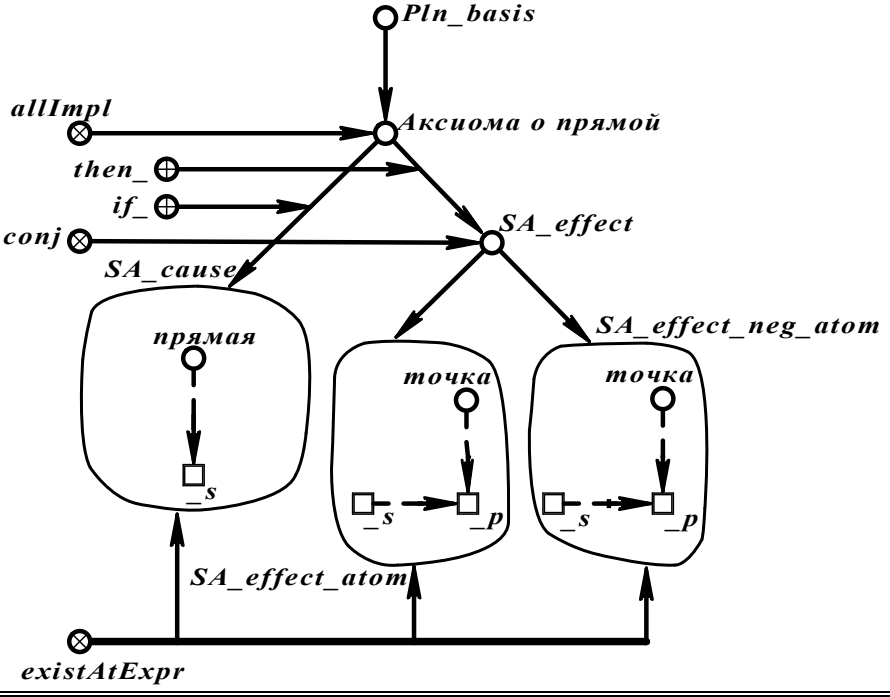


/* Формирование аксиоматики */

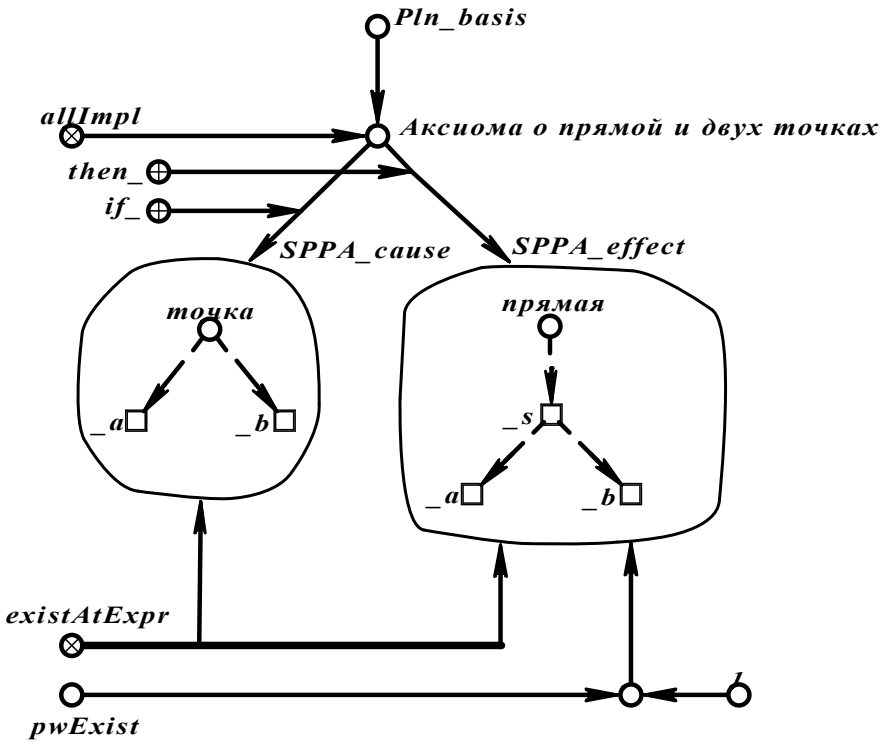
/* "через не лежащую на прямой точку можно провести параллельную этой прямой прямую и только одну" */



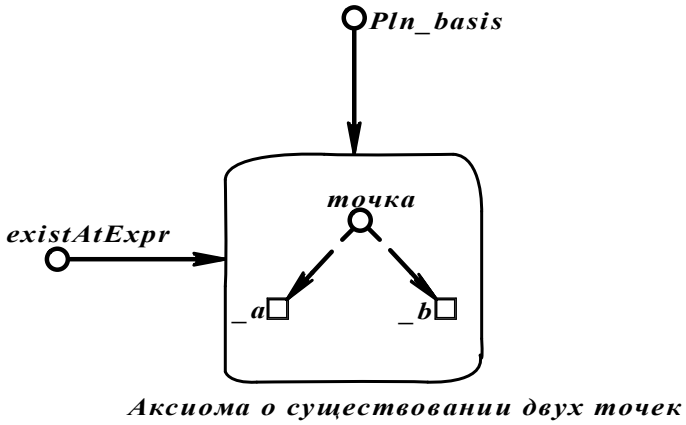
/* "для каждой прямой существуют точки принадлежащие ей и не принадлежащие ей" */



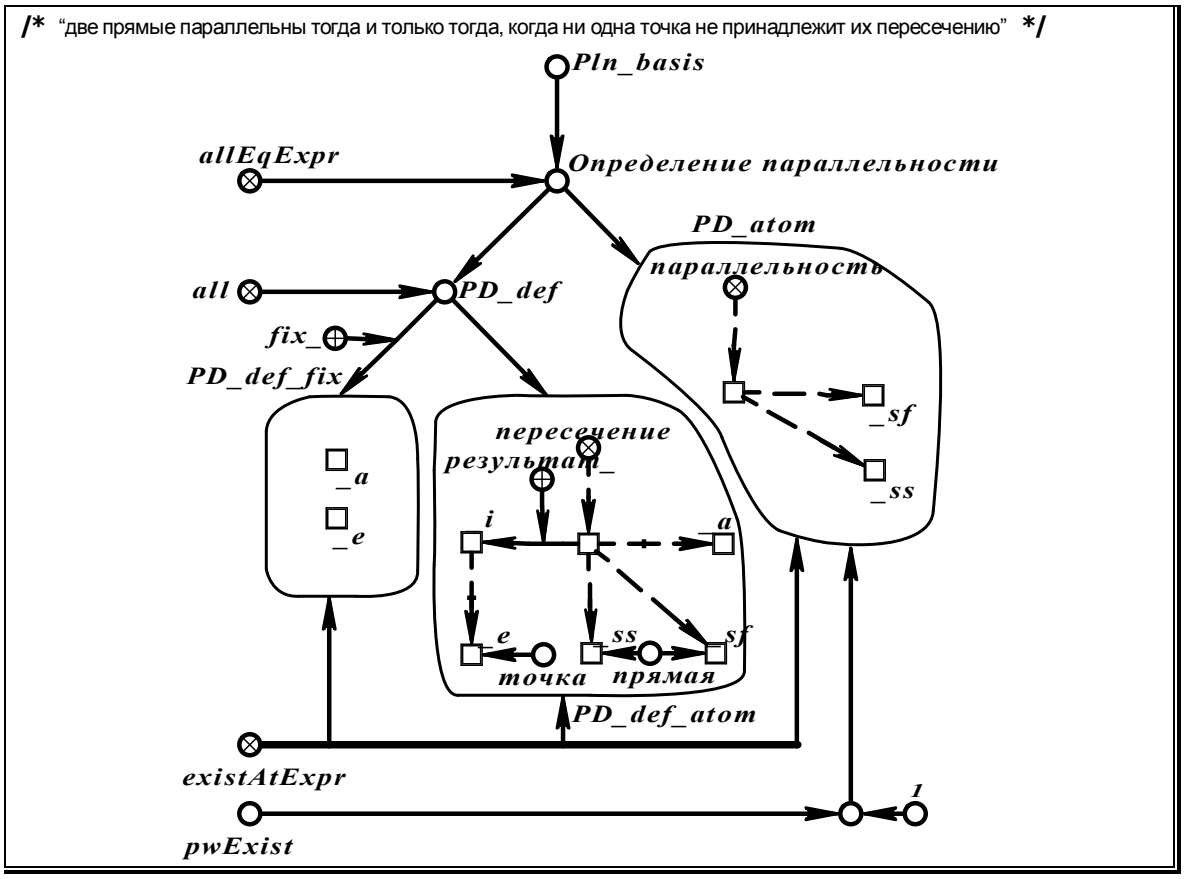
/* "через любые две точки можно провести прямую и только одну" */

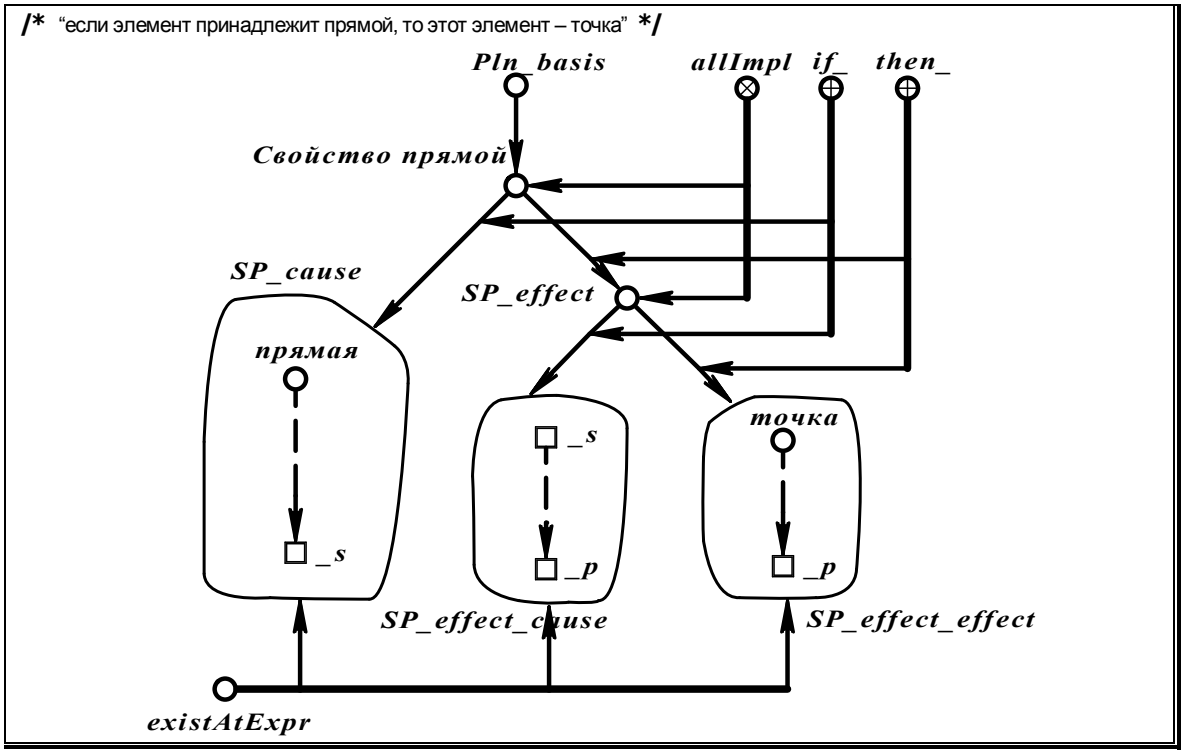


/* "существуют две точки" */



Аксиома о существовании двух точек


/* "две прямые параллельны тогда и только тогда, когда ни одна точка не принадлежит их пересечению" */
Pln_basis
allEqExpr
Определение параллельности
PD_atom
параллельность
all
PD_def
fix_
PD_def_fix
_sf
_ss
_a
_e
пересечение
результат_
i
_a
_e
ss
sf
точка
прямая
PD_def_atom
existAtExpr
1
pwExist


/* "если элемент принадлежит прямой, то этот элемент – точка" */
Pln_basis
allImpl
if_
then_
Свойство прямой
SP_cause
SP_effect
прямая
_s
_s
_p
точка
_p
SP_effect_cause
SP_effect_effect
existAtExpr

```
/* "фигура прямолинейна тогда и только тогда, когда существует прямая включающая все элементы фигуры" */
Pln_basis !; Определение прямолинейной фигуры ;
allEqExpr !; Определение прямолинейной фигуры ;
Определение прямолинейной фигуры !; SFD_def , SFD_atom ;
existAtExpr !; SFD_atom ;
SFD_atom = [ прямолинейная фигура !;; _f ; ];
exist !; SFD_def ;
existAtExpr !; SFD_def ;
SFD_def = [ подмножество !;; □· подмножество_ : _f , множество_ : _s·□ ;
прямая !;; _s ; ];
```

```
/* "пересечением произвольного количества множеств является множество , включающее такие и только такие элементы, каждый из которых принадлежит каждому множеству , образующему пересечение" */
Pln_basis !; Set_basis ;
Set_basis !; Определение пересечения ;
conj !; Определение пересечения ;
Определение пересечения !; DD_forward , DD_backward ;
allImpl !; DD_forward , DD_backward ;
DD_forward !; if_ : DD_def_f , then_ : DD_atom ;
DD_backward !; if_ : DD_atom , then_ : DD_def_b ;
existAtExpr !; DD_atom ;
DD_atom = [ пересечение !;; _c ; ];
exist !; DD_def_f ;
DD_def_f !; fix_ : DD_fix , DD_def_conj ;
exist !; DD_def_b ;
DD_def_b !; fix_ : DD_fix , DD_def_equivalence ;
DD_fix = [ _d ];
allEqExpr !; DD_def_equivalence ;
conj !; DD_def_conj ;
DD_def_conj !; DD_def_equivalence_equivalence , DD_def_atom ;
DD_def_equivalence !; DD_def_equivalence_equivalence , DD_def_atom ;
existAtExpr !; DD_def_atom ;
DD_def_atom = [ _c !;; результат_ :: _d ; ];
allEqExpr !; DD_def_equivalence_equivalence ;
DD_def_equivalence_equivalence !; DD_equivalence_atom , DD_implication ;
existAtExpr !; DD_equivalence_atom ;
DD_equivalence_atom = [ _d !;; _e ; ];
allImpl !; DD_implication ;
DD_implication !; if_ : DD_cause , then_ : DD_effect ;
existAtExpr !; DD_cause , DD_effect ;
DD_cause = [ _c !;; _s ; ];
DD_effect = [ _s !;; _e ; ];
```

```
/* "множество _ss является подмножеством множества _s, если каждый элемент _ss принадлежит множеству _s "
 */
Set_basis !; Определение подмножества ;
conj !; Определение подмножества ;
Определение подмножества !; SSD_forward ,SSD_backward ;
allImpl !; SSD_forward ,SSD_backward ;
SSD_forward !; if_ :SSD_def_f ,then_ :SSD_atom ;
SSD_backward !; if_ :SSD_atom ,then_ :SSD_def_b ;
existAtExpr !; SSD_atom ;
SSD_atom = [подмножество !;; _c ;];
exist !; SSD_def_f ;
SSD_def_f !; fix_ :SSD_fix ,SSD_def_conj ;
SSD_fix = [_ss ,_s ];
conj !; SSD_def_conj ;
allEqExpr !; SSD_def_b ;
SSD_def_b !; SSD_def_atom ,SSD_implication ;
SSD_def_conj !; SSD_def_atom ,SSD_implication ;
existAtExpr !; SSD_def_atom ;
SSD_def_atom = [_c !;; подмножество_ :: _ss ,множество_ :: _s ;];
allImpl !; SSD_implication ;
SSD_implication !; if_ :SSD_cause ,then_ :SSD_effect ;
existAtExpr !; SSD_cause ,SSD_effect ;
SSD_cause = [_ss !;; _e ;];
SSD_effect = [_s !;; _e ;];
```

Далее следует формулировка запроса.

```
/* "существуют параллельные прямые" */
existAtExpr !; query_atom1 ;
query_atom1 = [параллельность !;; □_a ,_c□ ;прямая !;; _a ,_c ;];
```

Приведём описание решения задачи на естественном языке.

Процесс решения состоит из следующих этапов:

1) определение множества формальных теорий описывающих закономерности свойств понятий, обозначенных константными элементами в запросе;
2) для конкретной теории заранее формируется множество целей (запросов) и множество предположений;
3) начинается обработка запроса, определяется тип высказывания запроса;
4) проверяется выводимость отрицания высказывания в рамках формальной теории, перебираются все истинные интерпретации (этапы 5, 6), каждая интерпретация на каждом этапе и только на нём включается во множество предположений;
5) осуществляется поиск семантически близких истинных высказываний формальной теории, при необходимости используются факты формальной теории, в данном примере такими высказываниями изначально являются: ***"Аксиома о параллельной прямой"***, ***"Аксиома о прямой"***, ***"Определение параллельности"***, ***"Свойство прямой"***, ***"Определение прямолинейной фигуры"*** и т. п. ;

6) осуществляется протоколирующийся логический вывод с помощью отобранных на предыдущем этапе высказываний, этапы 5 и 6 повторяются, пока не закончатся семантически близкие высказывания или будет выявлено противоречие, в последнем случае высказывание ложно и при условии возникновения противоречия на каждой истинной интерпретации осуществляется переход на этап 10;
7) проверяется выводимость высказывания в рамках формальной теории, перебираются все ложные интерпретации (этапы 8, 9) , каждая интерпретация на каждом этапе и только на нём включается во множество предположений;
8) осуществляется поиск семантически близких истинных высказываний формальной теории, при необходимости используются факты формальной теории;
9) осуществляется протоколирующийся логический вывод с помощью отобранных на предыдущем этапе высказываний, этапы 8 и 9 повторяются, пока не закончатся семантически близкие высказывания или будет выявлено противоречие, в последнем случае высказывание истинно;
10) на основании результатов полученных на этапах 6 и 9, формируется ответ: истинно ли, ложно или невыводимо высказывание в рамках текущей формальной теории; уничтожается множество предположений, выводится протокол решения;
11) процесс повторяется со второго этапа для каждой найденной формальной теории.

Рассмотрим более детально этот процесс для данного примера.

На первом этапе анализируются все константные элементы, входящие хотя бы в одно из атомарных высказываний запроса. В данном случае это элементы: **“*прямая*”**, **“*параллельность*”**, то есть те элементы, которые также входят во множество, помеченное атрибутом **“*union_*”** в рамках какой-либо формальной теории. На основе этого анализа осуществляется построение множества **“*theory_set*”** всех таких формальных теорий. В данном примере множество **“*theory_set*”** будет содержать элемент **“*Планиметрия*”**.

На втором этапе формулируется запрос к каждой формальной теории:

```
Goals ;! gq1 ;! query1 ;
active_ , confirm_ !; gq1 ;
query1 = [ theory !; Планиметрия !;; query_atom1 ];
```

На третьем этапе начинается обработка запроса (распишем последующие этапы по шагам) :

Шаг 1. К запросу **“*query1*”** применяется операция классификации и управления запросами, которая формирует конструкцию следующего вида:

```
Goals !; search : query1 ;
```

Шаг 2. К запросу **“*query1*”** применяется операция эпизодического информационного поиска. Которая завершается безуспешно:

```
Goals !; active_ : searched : query1 ;
factual ∀; query1 ;
```

Шаг 3. К запросу **“*query1*”** применяется операция классификации и управления запросами, которая формирует конструкцию следующего вида:

```
Goals !; traceUp : query1 ;
```

Шаг 4. К запросу **“*query1*”** применяется операция трассировки запроса снизу-вверх в сложном высказывании, которая завершается безуспешно:

```
Goals !; tracedUp : active_ : query1 ;
```

Шаг 5. К запросу “*query1*” применяется операция классификации и управления запросами, которая формирует конструкцию следующего вида:

```
Goals !; traceDown : query1 ;
```

Шаг 6. К запросу “*query1*” применяется операция трассировки запроса сверху-вниз, которая формирует конструкцию следующего вида:

```
Goals ;! gq1 ;! query1 ;
tracedDown , active_ !; gq1 ;
reasoning !; □effect_ : gq1 , expectations_ : □gx1 □, causes_ : C1 □;
Goals ;! gx1 ;! query_atom1 ;
tracedUp , confirm_ , active_ !; gx1 ;
```

Шаг 7. К запросу “*query_atom1*” применяется операция классификации и управления запросами, которая формирует конструкцию следующего вида:

```
Goals !; search : query_atom1 ;
```

Шаг 8. К запросу “*query_atom1*” применяется операция эпизодического информационного поиска. Которая завершается безуспешно:

```
Goals !; active_ : searched : query_atom1 ;
factual ∀; query_atom1 ;
```

Шаг 9. К запросу “*query_atom1*” применяется операция классификации и управления запросами, которая формирует конструкцию следующего вида:

```
Goals !; associateSimply : □associative !;; □·  (formula !; query_atom1 ), (theory  !;
Планиметрия ), _as ·□;
```

Шаг 10. К атомарному высказыванию “*query_atom1*” применяется операция простого ассоциирования атомарных формул. Среди таких формул будут формулы: “*PSA_atom*”, “*PSA_effect*”, “*SA_cause*”, “*SPPA_effect*”, “*PD_def_atom*”, “*PD_atom*”, “*SP_cause*”, “*SFD_def*”. Будет построена конструкция:

```
associative !; □(formula !; query_atom1 ), (theory !; Планиметрия ),
□PSA_atom , PSA_effect , SA_cause , SPPA_effect , PD_def_atom , PD_atom , SP_c
ause , SFD_def □□;
Goals !; active_ : query_atom1 ;
```

Шаг 11. К атомарному высказыванию “*query_atom1*” применяется операция выявления идентичных, входящих и пересекающихся атомарных формул. Результатом применения операции будут конструкции:

```
inclusion !; □(formula !; PSA_atom ), in_ : (formula !; SP_cause )□;
inclusion !; □(formula !; PSA_atom ), in_ : (formula !; SA_cause )□;
SA_cause = [прямая !;; _s ];      /* qa1_PSAa_intersection */
inclusion !; □(formula !; query_atom1 ), in_ : (formula !; PD_atom )□;
inclusion !; □(formula !; PSA_effect ), in_ : (formula !; PD_atom )□;
inclusion !; □(formula !; PSA_effect ), in_ : (formula !; SP_cause )□;
```

```
inclusion !; □(formula !; PSA_effect ),in_ :(formula !; SA_cause )□;
PD_atom = [параллельность !;; □ _sf , _ss □;];/* qa1_PSAe_intersection */
inclusion !; □(formula !; SPPA_effect ),in_ :(formula !; SP_cause )□;
inclusion !; □(formula !; SPPA_effect ),in_ :(formula !; SA_cause )□;
SA_cause = [прямая !;; _s ];/* qa1_SPPAe_intersctn */
inclusion !; □(formula !; query_atom1 ),in_ :(formula !;
qa1_PDda_intersection )□;
inclusion !; □(formula !; PD_def_atom ),
            in_ :(formula !; qa1_PDda_intersection )□;
qa1_PDda_intersection = [прямая !;; _s1 , _s2 ;];
inclusion !; □(formula !; qa1_PDda_intersection ),in_ :(formula !; SP_cause )□;
inclusion !; □(formula !; qa1_PDda_intersection ),in_ :(formula !; SA_cause )□;
inclusion !; □(formula !; SFD_def ),in_ :(formula !; SP_cause )□;
inclusion !; □(formula !; SFD_def ),in_ :(formula !; SA_cause )□;
 (formula !; SP_cause ) = (formula !; SA_cause );
```

Шаг 12. К атомарным формулам “*SA_cause*” и “*SP_cause*” применяется операция склейки атомарных формул: в результате чего формируется формула “*S__cause*”, а формулы “*SA_cause*” и “*SP_cause*” уничтожаются, а сгенерированные на предыдущем шаге конструкции преобразуются следующим образом:

```
inclusion !; □(formula !; PSA_atom ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; PSA_effect ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; SPPA_effect ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; qa1_PDda_intersection ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; SFD_def ),in_ :(formula !; S__cause )□;
```

Шаг 13. К атомарному высказыванию “*query_atom1*” применяется операция декомпозиции запроса на вывод атомарной формулы на запросы к семантически близким атомарным формулам. Запрос будет выставлен к формулам “*PSA_atom*”, “*PSA_effect*”, “*S__cause*”, “*SPPA_effect*”, “*PD_def_atom*”, “*PD_atom*”, “*SFD_def*”.

```
decomposed ,searched ,confirm_ ,active_ !; gx1 ;
reasoning !; □effect_ :gx1 ,expectations_ :□gx2□,causes_ :C2 □;
Goals ;! gx2 ;! PD_atom ;
searched ,confirm_ ,active_ !; gx2 ;
reasoning !; □effect_ :gx1 ,expectations_ :□gx3□,causes_ :C3 □;
Goals ;! gx3 ;! qa1_PDda_intersection ;
searched ,confirm_ ,active_ !; gx3 ;
reasoning !; □effect_ :gx3 ,expectations_ :□gx4□,causes_ :C4 □;
Goals ;! gx4 ;! S__cause ;
searched ,confirm_ ,active_ !; gx4 ;
reasoning !; □effect_ :gx4 ,expectations_ :□gx5□,causes_ :C5 □;
Goals ;! gx5 ;! PSA_atom ;
searched ,confirm_ ,active_ !; gx5 ;
reasoning !; □effect_ :gx2 ,expectations_ :E6 ,causes_ :C6 □;
```

```
reasoning !; □effect_ : gx4 , expectations_ : E6 , causes_ : C6 □;
E6 !; gx6 ;
Goals ;! gx6 ;! PSA_effect ;
searched , confirm_ , active_ !; gx6 ;
reasoning !; □effect_ : gx4 , expectations_ : □gx7□, causes_ : C7 □;
Goals ;! gx7 ;! SPPA_effect ;
searched , confirm_ , active_ !; gx7 ;
reasoning !; □effect_ : gx3 , expectations_ : □gx8□, causes_ : C8 □;
Goals ;! gx8 ;! PD_def_atom ;
searched , confirm_ , active_ !; gx8 ;
reasoning !; □effect_ : gx4 , expectations_ : □gx9□, causes_ : C9 □;
Goals ;! gx9 ;! SFD_def ;
searched , confirm_ , active_ !; gx9 ;
factual ∀;
PD_atom , qa1_PDda_intersection , S__cause , PSA_atom , PSA_effect , SPPA_effe
ct , PD_def_atom , SFD_def ;
```

Шаг 14. Остальные операции к высказыванию **"*query_atom1*"** не применимы, что выясняется в процессе применения к нему операции классификации и управления запросами.

Шаг 15. Начиная с формулы **"*PSA_cause*"** ко всем входящим формулам высказывания **"*Аксиома о параллельной прямой*"** ставится запрос на применение и применяется операция формирования множества свободных или связываемых переменных. Формируются, в частности, конструкции:

```
unassigned !; □(formula !; PSA_cause ),(theory !; Планиметрия ),
              PSA_cause_uVars □;
PSA_cause_uVars = [ _sf , _p ];
unassigned !; □(formula !; PSA_neg ),(theory !; Планиметрия ),
              PSA_neg_uVars □;
PSA_neg_uVars = [ _sf , _p ];
unassigned !; □(formula !; Аксиома о параллельной прямой ) , PSA_uVars ,
              (theory !; Планиметрия ) □;
PSA_uVars = [ _sf , _p ];
```

Шаг 16. После операции классификации и управления запросами к атомарной формуле **"*PSA_atom*"** применяется операция трассировки запроса снизу-вверх, которая находит формулу **"*PSA_cause*"**. В результате формируется конструкция:

```
reasoning !; □effect_ : gx5 , expectations_ : □gx10□, causes_ : C10 □;
unassigned !; □(formula !; PSA_cause ),(theory !; Планиметрия ),
              PSA_cause_uVars □;
Goals ;! gx10 ;! PSA_cause_uVars ;
searched , confirm_ , active_ !; gx10 ;
tracedUp , active_ !; gx5 ;
```

Шаг 17. Затем к формуле **"*PSA_cause*"** применяется операция трассировки запроса сверху-вниз, которая находит формулу **"*PSA_neg*"**. В результате формируется конструкция:

```
reasoning !; □effect_ : gx10 , expectations_ : □gx5 , gx11 □, causes_ : C11 □;
unassigned !; □(formula !; PSA_neg ),(theory !; Планиметрия ),
              PSA_neg_uVars □;
Goals ;! gx11 ;! PSA_neg_uVars ;
searched , confirm_ , active_ !; gx11 ;
tracedDown , active_ !; gx10 ;
```

Шаг 18. К формуле *"PSA_cause"* применяется операция трассировки запроса снизу-вверх, которая находит априори истинное импликативное высказывание *"Аксиома о параллельной прямой"*, следствием которого является формула *"PSA_effect"*. В результате формируются конструкции:

```
reasoning !; □effect_ : gx14 , expectations_ : □gx12 , gx13 □, causes_ : C12 □;
unassigned !; □(formula !; Аксиома о параллельной прямой ) , PSA_uVars ,
              (theory !; Планиметрия ) □;
Goals ;! gx12 ;! PSA_uVars ;
searched , confirm_ , active_ !; gx12 ;
Goals ;! gx13 ;! PSA_effect ;
deny_ , active_ !; gx13 ;
Goals ;! gx14 ;! [refuse !;; gx10 ];
confirm_ !; gx14 ;
tracedUp , active_ !; gx10 ;
```

Шаг 19. К формуле *"Аксиома о параллельной прямой"* применяется операция трассировки запроса снизу-вверх. В результате формируется конструкция:

```
confirmed_ !; gx12 ;
```

Шаг 20. К формуле *"PSA_neg"* применяется операция трассировки запроса сверху-вниз, которая находит формулу *"PSA_neg_atom"*. В результате формируется конструкция:

```
reasoning !; □effect_ : gx11 , expectations_ : □gx15 □, causes_ : C15 □;
Goals ;! gx15 ;! PSA_neg_atom ;
deny_ , active_ !; gx15 ;
tracedDown , active_ !; gx11 ;
```

Шаг 21. К атомарной формуле *"PSA_effect"* применяется операция эпизодического информационного поиска. Которая завершается безуспешно:

```
factual ∀; PSA_effect ;
searched , active_ !; gx13 ;
```

Шаг 22. К атомарной формуле *"PSA_effect"* применяется операция трассировки запроса снизу-вверх, которая находит априори истинное импликативное высказывание *"Аксиома о параллельной прямой"*, посылкой которого является формула *"PSA_cause"*. В результате формируется конструкция:

```
reasoning !; □effect_ : gx6 , expectations_ : □gx10 □, causes_ : C13 □;
C13 !; gx12 ;
```

```
tracedUp ,active_ !; gx6 ;
```

Шаг 23. К формуле *"PSA_cause"* применяется операция трассировки запроса сверху-вниз, которая находит формулы *"PSA_atom"* и *"PSA_neg"*. В результате формируется конструкция:

```
reasoning !; □effect_ : gx10 ,expectations_ : □gx11 ,gx5 □,causes_ : C14 □;
tracedDown ,active_ !; gx10 ;
```

Шаг 24. К атомарной формуле *"PSA_atom"* применяется операция простого ассоциирования атомарных формул. Будет построена конструкция:

```
associative !;□(formula !;PSA_atom ),(theory !; Планиметрия ),
□query_atom1 ,PSA_effect ,S__cause ,SA_effect_atom ,SA_effect_neg_atom ,SP
PA_cause ,SPPA_effect ,Аксиома о существовании двух
точек ,PD_def_atom ,SFD_def ,SP_effect_effect□□ ;
searched ,confirm_ ,active_ !;gx5 ;
```

Шаг 25. К атомарной формуле *"PSA_atom"* применяется операция выявления идентичных, входящих и пересекающихся атомарных формул. Результатом применения операции будет:

```
inclusion !;□(formula !; PSA_atom ),in_ :(formula !; S__cause )□;
inclusion !;□(formula !; qa1_PDda_intersection ),in_ :(formula !; S__cause )□;
inclusion !;□(formula !; query_atom1 ),in_ :(formula !;
qa1_PDda_intersection )□;
inclusion !;□(formula !; PSA_effect ),in_ :(formula !; S__cause )□;
inclusion !;□(formula !; PSA_atom ),in_ :(formula !; SP_effect_effect )□;
inclusion !;□(formula !; SA_effect_atom ),in_ :(formula !; SP_effect_effect )□;
inclusion !;□(formula !; SA_effect_neg_atom ),
            in_ :(formula !; SP_effect_effect )□;
SP_effect_effect = [точка !;; _p ];      /* PSAa_SAea_intersection */
inclusion !;□(formula !; SPPA_cause ),in_ :(formula !; SP_effect_effect )□;
inclusion !;□(formula !; Аксиома о существовании двух точек ),
            in_ :(formula !; SP_effect_effect )□;
inclusion !;□in_ :(formula !; Аксиома о существовании двух точек ),
            in_ :(formula !; SPPA_cause )□;
inclusion !;□(formula !; SPPA_effect ),in_ :(formula !; S__cause )□;
inclusion !;□(formula !; PD_def_atom ),
            in_ :(formula !; qa1_PDda_intersection )□;
inclusion !;□(formula !; PD_def_atom ),in_ :(formula !; PSA_atom )□;
inclusion !;□(formula !; SFD_def ),in_ :(formula !; S__cause )□;
```

Шаг 26. К атомарному формуле *"PSA_atom"* применяется операция декомпозиции запроса на вывод атомарной формулы на запросы к семантически близким атомарным формулам. В результате будут добавлены следующие конструкции:

```
reasoning !; □effect_ : gx5 ,expectations_ : □gx4□,causes_ : C15 □;
decomposed ,searched ,confirm_ ,active_ !; gx5 ;
reasoning !; □effect_ : gx4 ,expectations_ : □gx3□,causes_ : C16 □;
```

```
searched ,confirm_ ,active_ !; gx4 ;
reasoning !; □effect_ : gx3 ,expectations_ : □gx1□,causes_ : C17 □;
searched ,confirm_ ,active_ !; gx3 ;
reasoning !; □effect_ : gx5 ,expectations_ : □gx18□,causes_ : C18 □;
Goals ;! gx18 ;! SP_effect_effect ;
searched ,confirm_ ,active_ !; gx18 ;
reasoning !; □effect_ : gx18 ,expectations_ : □gx19□,causes_ : C19 □;
Goals ;! gx19 ;! SA_effect_atom ;
searched ,confirm_ ,active_ !; gx19 ;
reasoning !; □effect_ : gx18 ,expectations_ : □gx20□,causes_ : C20 □;
Goals ;! gx20 ;! SA_effect_neg_atom ;
searched ,confirm_ ,active_ !; gx20 ;
reasoning !; □effect_ : gx18 ,expectations_ : □gx21□,causes_ : C21 □;
Goals ;! gx21 ;! SPPA_cause ;
searched ,confirm_ ,active_ !; gx21 ;
reasoning !; □effect_ : gx18 ,expectations_ : □gx22□,causes_ : C22 □;
Goals ;! gx22 ;! «Аксиома о существовании двух точек» ;
searched ,confirm_ ,active_ !; gx22 ;
reasoning !; □effect_ : gx22 ,expectations_ : □gx21□,causes_ : C21 □;
reasoning !; □effect_ : gx21 ,expectations_ : □gx22□,causes_ : C22 □;
reasoning !; □effect_ : gx5 ,expectations_ : □gx8□,causes_ : C23 □;
factual ∀;
SP_effect_effect ,SA_effect_atom ,SA_effect_neg_atom ,SPPA_cause ,Аксиома
о существовании двух точек ;
```

Шаг 27. К атомарной формуле "*PSA_effect*" применяется операция простого ассоциирования атомарных формул. Будет построена конструкция:

```
associative !; □(formula !; PSA_effect ),(theory !;
Планиметрия ),□query_atom1 ,PSA_atom ,S__cause ,SPPA_effect ,PD_def_ato
m ,PD_atom , SFD_def □□ ;
searched ,deny_ ,active_ !; gx13 ;
```

Шаг 28. К атомарной формуле "*PSA_effect*" применяется операция выявления идентичных, входящих и пересекающихся атомарных формул. Результатом применения операции будут найденные конструкции:

```
inclusion !; □(formula !; qa1_PDda_intersection ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; query_atom1 ),in_ :(formula !;
qa1_PDda_intersection )□;
inclusion !; □(formula !; PSA_effect ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; PSA_atom ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; PD_def_atom ),
            in_ :(formula !; qa1_PDda_intersection )□;
inclusion !; □(formula !; PSA_effect ),in_ :(formula !; PD_atom )□;
```

```
inclusion !; □(formula !; SFD_def),in_:(formula !; S__cause)□;
```

Шаг 29. К атомарной формуле *"PSA_effect"* применяется операция декомпозиции запроса на вывод атомарной формулы на запросы к семантически близким атомарным формулам. В результате будут добавлены следующие конструкции:

```
decomposed,searched,deny_,active_ !; gx13;
reasoning !; □effect_:gx13,expectations_:□gx24□,causes_:C24 □;
Goals ;! gx24 ;! S__cause;
searched,deny_,active_ !; gx24;
reasoning !; □effect_:gx13,expectations_:□gx25□,causes_:C25 □;
Goals ;! gx25 ;! PSA_effect;
searched,deny_,active_ !; gx25;
factual ∀; S__cause,PSA_effect;
```

Шаг 30. Ко всем входящим формулам высказывания *"Аксиома о прямой"* ставится запрос на применение и применяется операция формирования множества свободных или связываемых переменных. Формируются, в частности, конструкции:

```
unassigned !; □(formula !; SA_effect),(theory !; Планиметрия),
              SA_effect_uVars □;
SA_effect_uVars = [_s];
unassigned !; □(formula !; Аксиома о прямой),    SA_uVars,
              (theory !; Планиметрия) □;
SA_uVars = [_s];
```

Шаг 31. Применяется операция трассировки запроса снизу-вверх, которая находит априори истинное импликативное высказывание *"Аксиома о прямой"*, следствием которого является конъюнктивная формула *"SA_effect"*. В результате формируется конструкция:

```
reasoning !; □effect_:gx28,expectations_:□gx26,gx27□,causes_:C26 □;
unassigned !; □(formula !; Аксиома о прямой),    SA_uVars,
              (theory !; Планиметрия) □;
Goals ;! gx27 ;! SA_uVars;
searched,confirm_,active_ !; gx27;
Goals ;! gx26 ;! SA_effect_uVars;
searched,deny_,active_ !; gx26;
Goals ;! gx28 ;! [refuse !;; gx4];
confirm_ !; gx28;
tracedUp,active_ !; gx4;
```

Шаг 32. К формуле *"Аксиома о прямой"* применяется операция трассировки запроса снизу-вверх. В результате формируется конструкция:

```
confirmed_ !; gx27;
```

Шаг 33. К формуле *"SA_effect"* применяется операция трассировки запроса сверху-вниз, которая находит формулы: *"SA_effect_neg_atom"* и *"SA_effect_atom"*. В результате формируется конструкция:

```
reasoning !; □effect_ : gx26 , expectations_ : □gx29□ , causes_ : C29 □ ;
Goals ;! gx29 ;! SA_effect_atom ;
searched , deny_ , active_ !; gx29 ;
reasoning !; □effect_ : gx26 , expectations_ : □gx30□ , causes_ : C30 □ ;
Goals ;! gx30 ;! SA_effect_neg_atom ;
searched , deny_ , active_ !; gx30 ;
```

Шаг 34. Для атомарной формулы *"SA_effect_atom"* выполняется операция эпизодического информационного поиска. Которая завершается безуспешно:

```
factual ∀; SA_effect_atom ;
```

Шаг 35. Для атомарной формулы *"SA_effect_neg_atom"* выполняется операция эпизодического информационного поиска. Которая завершается безуспешно:

```
factual ∀; SA_effect_neg_atom ;
```

Шаг 36. Для атомарной формулы *"SA_effect_atom"* выполняется операция трассировки запроса снизу-вверх, которая находит конъюнктивную формулу *"SA_effect"*. В результате формируется конструкция:

```
reasoning !; □effect_ : gx19 , expectations_ : □gx026□ , causes_ : C026 □ ;
Goals ;! gx026 ;! SA_effect_uVars ;
searched , confirm_ , active_ !; gx026 ;
```

Шаг 37. Для атомарной формулы *"SA_effect_neg_atom"* выполняется операция трассировки запроса снизу-вверх, которая находит конъюнктивную формулу *"SA_effect"*. В результате формируется конструкция:

```
reasoning !; □effect_ : gx20 , expectations_ : □gx026□ , causes_ : C027 □ ;
```

Шаг 38. Для атомарной формулы *"SA_effect"* выполняется операция трассировки запроса снизу-вверх, которая находит априори истинное импликативное высказывание *"Аксиома о прямой"*, посылкой которого является формула *"SA_cause"*. В результате формируется конструкция:

```
reasoning !; □effect_ : gx026 , expectations_ : □gx4□ , causes_ : □gx27□ □ ;
```

Шаг 39. К атомарной формуле *"SA_effect_atom"* применяется операция простого ассоциирования атомарных формул. Будет построена конструкция:

```
associative !; □ ( formula !; PSA_effect_atom ) , ( theory !;
Планиметрия ) , □PSA_atom , SA_effect_neg_atom , SPPA_cause , Аксиома о
существовании двух точек , PD_def_atom , SP_effect_effect□□ ;
searched , deny_ , active_ !; gx29 ;
```

Шаг 40. К атомарной формуле *"SA_effect_atom"* применяется операция выявления идентичных, входящих и пересекающихся атомарных формул. Результатом применения операции будут конструкции:

```
inclusion !; □ ( formula !; PSA_atom ) , in_ : ( formula !; SP_effect_effect ) □ ;
```

```
inclusion !; □(formula !; SA_effect_atom ),in_ :(formula !; SP_effect_effect )□;
inclusion !; □(formula !; SA_effect_neg_atom ),
            in_ :(formula !; SP_effect_effect )□;
inclusion !; □(formula !; SPPA_cause ),in_ :(formula !; SP_effect_effect )□;
inclusion !; □(formula !; Аксиома о существовании двух точек ),
            in_ :(formula !; SP_effect_effect )□;
inclusion !; □in_ :(formula !; Аксиома о существовании двух точек ),
            in_ :(formula !; SPPA_cause )□;
inclusion !; □(formula !; PD_def_atom ),in_ :(formula !; PSA_atom )□;
```

Шаг 41. К атомарному формуле "*SA_effect_atom*" применяется операция декомпозиции запроса на вывод атомарной формулы на запросы к семантически близким атомарным формулам. В результате будут добавлены следующие конструкции:

```
decomposed ,searched ,deny_ ,active_ !; gx29 ;
reasoning !; □effect_ :gx29 ,expectations_ :□gx31□,causes_ :C31 □;
Goals ;! gx29 ;! SA_effect_atom ;
Goals ;! gx31 ;! SP_effect_effect ;
searched ,deny_ ,active_ !; gx31 ;
factual ∀; SP_effect_effect ;
```

Шаг 42. Начиная с формулы "*SP_effect*" ко всем входящим формулам высказывания "*Свойство прямой*" ставится запрос на применение и применяется операция формирования множества свободных или связываемых переменных. Формируются, в частности, конструкции:

```
unassigned !; □(formula !; SP_effect ),(theory !; Планиметрия ),
            SP_effect_uVars□;
SP_effect_uVars = [_s ,_p ];
unassigned !; □(formula !; Свойство прямой ) ,    SP_uVars ,
            (theory !; Планиметрия ) □;
SP_uVars = [_s ];
```

Шаг 43. Применяется операция трассировки запроса снизу-вверх, которая находит импликативную формулу "*SP_effect*". В результате формируется конструкция:

```
reasoning !; □effect_ :gx18 ,expectations_ :□gx32 ,gx33□,causes_ :C32 □;
reasoning !; □effect_ :gx31 ,expectations_ :□gx34 ,gx33□,causes_ :C34 □;
unassigned !; □(formula !; SP_effect ) ,SP_effect_uVars ,
            (theory !; Планиметрия ) □;
Goals ;! gx32 ;! SP_effect_uVars ;
searched ,confirm_ ,active_ !; gx32 ;
Goals ;! gx34 ;! SP_effect_uVars ;
searched ,deny_ ,active_ !; gx34 ;
Goals ;! gx33 ;! SP_effect_cause ;
confirm_ ,active_ !; gx33 ;
tracedUp ,active_ !; gx31 ;
```

```
tracedUp ,active_ !; gx18 ;
```

Шаг 44. Применяется операция трассировки запроса снизу-вверх, которая находит априори истинное импликативное высказывание **"Свойство прямой"**, посылкой которого является атомарная формула **"SP_cause"**. В результате формируется конструкция:

```
reasoning !; □effect_ : gx32 ,expectations_ : □gx35 ,gx4□,causes_ : C35 □;
reasoning !; □effect_ : gx34 ,expectations_ : □gx36 ,gx4□,causes_ : C36 □;
unassigned !; □(formula !; Свойство прямой ) ,      SP_uVars ,
                (theory !; Планиметрия ) □;
Goals ;! gx35 ;! SP_uVars ;
searched ,confirm_ ,active_ !; gx35 ;
Goals ;! gx36 ;! SP_uVars ;
searched ,deny_ ,active_ !; gx36 ;
Goals ;! gx4 ;! S__cause ;
tracedUp ,active_ !; gx32 ;
tracedUp ,active_ !; gx34 ;
```

Шаг 45. К атомарной формуле **"S__cause"** применяется операция простого ассоциирования атомарных формул. Будет построена конструкция:

```
associative !; □(formula !; S__cause ),(theory !;
Планиметрия ),□query_atom1 ,PSA_atom ,PSA_effect ,SA_cause ,SPPA_effect
,PD_def_atom ,SFD_def□□ ;
searched ,confirm_ ,active_ !; gx4 ;
```

Шаг 46. К атомарной формуле **"S__cause"** применяется операция выявления идентичных, входящих и пересекающихся атомарных формул. Результатом применения операции будут конструкции:

```
inclusion !; □(formula !; query_atom1 ),in_ :(formula !;
qa1_PDda_intersection )□;
inclusion !; □(formula !; qa1_PDda_intersection ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; PSA_atom ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; PSA_effect ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; SPPA_effect ),in_ :(formula !; S__cause )□;
inclusion !; □(formula !; PD_def_atom ),in_ :(formula !; PSA_atom )□;
inclusion !; □(formula !; SFD_def ),in_ :(formula !; S__cause )□;
```

Шаг 47. К атомарному высказыванию **"S__cause"** применяется операция декомпозиции запроса на вывод атомарной формулы на запросы к семантически близким атомарным формулам. В результате будут добавлены следующие конструкции:

```
decomposed ,searched ,confirm_ ,active_ !; gx4 ;
```

Шаг 48. Ко всем формулам высказывания **"Аксиома о прямой и двух точках"** ставится запрос на применение и применяется операция формирования множества свободных или связываемых переменных. Формируется, в частности, конструкция:

```
unassigned !; □(formula !; Аксиома о прямой и двух точках ) ,
                SPPA_uVars ,
                (theory !; Планиметрия ) □;
SPPA_uVars = [_a , _b ];
```

Ш а г 4 9 . Применяется операция трассировки запроса снизу-вверх, которая находит априори истинное импликативное высказывание **"*Аксиома о прямой и двух точках*"**, которое включает формулу **"*SPPA_cause*"** в качестве посылки. В результате формируется конструкция:

```
reasoning !; □effect_ : gx7 , expectations_ : □gx37 , gx21□, causes_ : C37 □;
unassigned !; □(formula !; SP_effect ), SPPA_uVars ,
                (theory !; Планиметрия ) □;
Goals ;! gx37 ;! SPPA_uVars ;
searched , confirm_ , active_ !; gx37 ;
Goals ;! gx21 ;! SPPA_cause ;
searched , confirm_ , active_ !; gx21 ;
```

Ш а г 5 0 . К атомарной формуле **"*SPPA_cause*"** применяется операция простого ассоциирования атомарных формул. Будет построена конструкция:

```
associative !; □(formula !; SPPA_cause ),(theory !;
Планиметрия ),□PSA_atom , SA_effect_atom , SA_effect_neg_atom , Аксиома о
существовании двух точек , PD_def_atom , SP_effect_effect□□ ;
searched , confirm_ , active_ !; gx21 ;
```

Ш а г 5 1 . К атомарной формуле **"*SPPA_cause*"** применяется операция выявления идентичных, входящих и пересекающихся атомарных формул. Результатом применения операции будут конструкции:

```
inclusion !; □(formula !; PSA_atom ),in_ :(formula !; SP_effect_effect )□;
inclusion !; □(formula !; SPPA_cause ),in_ :(formula !; SP_effect_effect )□;
inclusion !; □(formula !; SA_effect_atom ),in_ :(formula !; SP_effect_effect )□;
inclusion !; □(formula !; SA_effect_neg_atom ),
                in_ :(formula !; SP_effect_effect )□;
inclusion !; □(formula !; Аксиома о существовании двух точек ),
                in_ :(formula !; SP_effect_effect )□;
inclusion !; □in_ :(formula !; Аксиома о существовании двух точек ),
                in_ :(formula !; SPPA_cause )□;
inclusion !; □(formula !; PD_def_atom ),in_ :(formula !; PSA_atom )□;
```

Ш а г 5 2 . К атомарной формуле **"*SPPA_cause*"** применяется операция декомпозиции запроса на вывод атомарной формулы на запросы к семантически близким атомарным формулам. В результате будут добавлены следующие конструкции:

```
decomposed , associatedSimply , searched , confirm_ , active_ !; gx21 ;
reasoning !; □effect_ : gx21 , expectations_ : □gx18□, causes_ : C33 □;
reasoning !; □effect_ : gx18 , expectations_ : □gx5□, causes_ : C39 □;
```

Шаг 53. Для атомарного формулы "*Аксиома о существовании двух точек*" выполняется операция эпизодического информационного поиска. Которая завершается безуспешно:

```
factual ∀; Аксиома о существовании двух точек ;
```

Шаг 54. Применяется операция трассировки запроса снизу-вверх, которая выявляет истинность высказывания "*Аксиома о существовании двух точек*".

```
reasoning !; □effect_ : gx22 , expectations_ : E39 , causes_ : □gx39□□ ;
Goals ;! gx39 ;! query2 ;
confirmed_ !; gx39 ;
query2 = [ theory !; Планиметрия !;; Аксиома о существовании двух точек ];
searched , active_ !; gx22 ;
```

Шаг 55. Применяется операция формирования фиктивной интерпретации атомарной формулы, которая формирует следующую конструкцию:

```
interpretation !; □( formula !; Аксиома о существовании двух
точек ), interpretation_ : □□var_ : _a , value_ : av□, □var_ : _aa , value_ : aav□, □va
r_ : _b , value_ : bv□, □var_ : _ba , value_ : bav□□ , ( formula !; f ) □ ;
f = [ точка ;! aav ;! av ; точка ;! bav ;! bv ; ];
confirmed_ !; gx22 ;
```

где *_aa* и *_ba* соответственно переменные дуги, включённые в высказывание и входящие в *_a* и в *_b*.

Шаг 56. К формуле "*SPPA_cause*" применяется операция формирования интерпретации формулы, которая формирует аналогичную конструкцию:

```
interpretation !;□( formula !; SPPA_cause ), ( formula !;
f ), interpretation_ : □□var_ : _a , value_ : av□, □var_ : _aa , value_ : aav□, □var_ : _b
 , value_ : bv□, □var_ : _ba , value_ : bav□□ ;
confirmed_ !; gx21 ;
```

Шаг 57. К формуле "*SPPA_uVars*" применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□( formula !; SPPA_uVars ), ( formula !;
f2 ), interpretation_ : □□var_ : _a , value_ : av□, □var_ : _b , value_ : bv□□ ;
f2 = [ av , bv ];
confirmed_ !; gx37 ;
```

Шаг 58. К высказыванию "*SPPA_effect*" применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□( formula !; SPPA_effect ), ( formula !;
f3 ), interpretation_ : □□var_ : _a , value_ : av□, □var_ : _b , value_ : bv□, □var_ : _ba
, value_ : bav□, □var_ : _aa , value_ : aav□, □var_ : _s , value_ : sv□, □var_ : _s , value
_ : sav□□□ ;
f3 = [ прямая !; sv ; sv !; av , bv ; ];
confirmed_ !; gx7 ;
```

Шаг 59. К высказыванию *"S__cause"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !;
S__cause ),interpretation_:□□var_:_s ,value_:sv □,□var_:_s ,value_:sav □□ ,
(formula !; f4 )□;
f4 = [прямая !; sv ;];
confirmed_ !; gx4 ;
```

Шаг 60. К высказыванию *"SA_uVars"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !;
SA_uVars ),interpretation_:□□var_:_s ,value_:sv□□ ,(formula !; f5 )□;
f5 = [sv ];
confirmed_ !; gx027 ;
refuse !; gx27 ;
```

Шаг 61. К высказыванию *"SP_uVars"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; SP_uVars ),
interpretation_:□□var_:_s ,value_:sv□□ ,(formula !; f6 )□;
f6 = [sv ];
confirmed_ !; gx35 ;
```

Шаг 62. К высказыванию *"SA_effect_uVars"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !; □(formula !; SA_effect_uVars ) ,(formula !;
f7 ),interpretation_:□□var_:_s ,value_:sv□□□;
f7 = [sv ];
confirmed_ !; gx026 ;
refuse !; gx26 ;
```

Шаг 63. К высказыванию *"SA_effect_neg_atom"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; SA_effect_neg_atom ) ,(formula !;
f8 ),interpretation_:□□var_:_s ,value_:sv□,□var_:_sn ,value_:snv□,□var_:_p
 ,value_:pv□,□var_:_pa ,value_:pav□□□;
f8 = [sv ∀; pv ;точка !; pv ;];
confirmed_ !; gx20 ;
refuse !; gx30 ;
```

Шаг 64. К высказыванию *"SA_effect_atom"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; SA_effect_neg_atom ) ,(formula !;
f8 ),interpretation_ :□□var_ : _s ,value_ :sv□,□var_ : _sa ,value_ :sav□,□var_ : _p
 ,value_ :bv□,□var_ : _pa ,value_ :bav□□□;
f8 = [sv !; bv ;точка !; bv ;];
confirmed_ !; gx19 ;
refuse !; gx29 ;
```

Шаг 65. К высказыванию *"SP_effect_effect"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; SP_effect_effect ),(formula !; f9 ),
interpretation_ :□□var_ : _p ,value_ :pv□,□var_ : _pa ,value_ :pav□□□;
f9 = [точка !; pv ;];
confirmed_ !; gx18 ;
```

Шаг 66. К высказыванию *"PSA_atom"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; SA_effect_neg_atom ),(formula !; f10 ),
interpretation_ :□□var_ : _p ,value_ :pv□,□var_ : _pa ,value_ :pav□□□;
f10 = [точка !; pv ;];
confirmed_ !; gx5 ;
```

Шаг 67. К высказыванию *"PSA_atom"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; PSA_atom ),(formula !; f11 ),
interpretation_ :□□var_ : _p ,value_ :pv□,□var_ : _pa ,value_ :pav□□□;
f11 = [точка !; pv ;];
interpreted !; gx5 ;
```

Шаг 68. К высказыванию *"PSA_cause"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; PSA_cause_uVars ),(formula !; f12 ),
interpretation_ :□□var_ : _p ,value_ :pv□□□;
f12 = [pv ];
interpreted !; gx10 ;
```

Шаг 69. К высказыванию *"PSA_neg"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; PSA_neg_uVars ),(formula !; f13 ),
interpretation_ :□□var_ : _p ,value_ :pv□□□;
f13 = [pv ];
interpreted !; gx11 ;
```

Шаг 70. К высказыванию *"PSA_neg_atom"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; PSA_neg_atom ),(formula !; f14 ),
interpretation_ :□□var_ : _p ,value_ :pv□□□;
f14 = [pv ];
interpreted !; gx15 ;
```

Шаг 71. К высказыванию *"PSA_neg_atom"* применяется операция расширенного ассоциирования атомарных формул. Будет построена временная конструкция:

```
temporary ,associative !;□(formula !; PSA_neg_atom ),
(theory !; Планиметрия ),□SA_effect_neg_atom ,SP_effect_effect□□;
searched ,confirm_ ,active_ !;gx15 ;
```

Шаг 72. К атомарному высказыванию *"PSA_neg_atom"* применяется операция выявления идентичных, входящих и пересекающихся атомарных формул. Результатом применения операции будут генерируемые конструкции:

```
inclusion !;□(formula !; SA_effect_neg_atom ),
in_ :(formula !; PSA_neg_atom )□;
inclusion !;□(formula !; SP_effect_effect ),
in_ :(formula !; PSA_neg_atom )□;
```

Шаг 73. К атомарному высказыванию *"PSA_neg_atom"* применяется операция декомпозиции запроса на вывод простого высказывания на запросы к семантически близким простым высказываниям. В результате будут добавлены следующие конструкция:

```
reasoning !; □effect_ :gx15 ,expectations_ :E40 ,causes_ :□gx19□□;
```

Шаг 74. К высказыванию *"PSA_neg_atom"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; PSA_neg_atom ),(formula !; f15 ),
interpretation_ :□□var_ : _s ,value_ :sv□,
□var_ : _sn ,value_ :snv□,□var_ : _p ,value_ :pv□□□;
f15 = [sv ∀; pv ;];
denied_ !; gx15 ;
```

Шаг 75. К высказыванию *"PSA_neg"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !; PSA_neg_uVars ),(formula !; f13 ),
interpretation_ :□□var_ : _s ,value_ :sv□,□var_ : _p ,value_ :pv□□□;
f13 = [pv ,sv ];
confirmed_ !; gx11 ;
```

Шаг 76. К высказыванию *"PSA_cause"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !;PSA_cause_uVars ),(formula !;f12 ),
interpretation_:□□var_:_s ,value_:sv□,□var_:_p ,value_:pv□□□;
f12 = [pv ,sv ];
interpreted !;gx10 ;
```

Шаг 77. К высказыванию *"PSA_atom"* применяется операция формирования интерпретации формулы, которая на основании формул *"S__cause"* и *"PSA_cause"* формирует конструкцию:

```
interpretation !;□(formula !;PSA_cause_uVars ),(formula !; f11 ),
interpretation_:□□var_:_s ,value_:sav□,□var_:_s ,value_:sv□,
□var_:_pa ,value_:pav□,□var_:_p ,value_:pv□□□;
f11 = [прямая !;sv ;точка !;pv ;];
confirmed_ !;gx5 ;
```

Шаг 78. К высказыванию *"PSA_cause"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !;PSA_cause_uVars ),(formula !; f12 ),
interpretation_:□□var_:_s ,value_:sv□,□var_:_p ,value_:pv□□□;
f12 = [pv ,sv ];
confirmed_ !;gx10 ;
```

Шаг 79. К высказыванию *"Аксиома о параллельной прямой"* применяется операция формирования интерпретации формулы, которая формирует конструкцию:

```
interpretation !;□(formula !;PSA_uVars ),(formula !; f16 ),
interpretation_:□□var_:_s ,value_:sv□,□var_:_p ,value_:pv □□□;
f16 = [pv ,sv ];
confirmed_ !;gx12 ;
```

Шаг 80. К высказыванию *"PSA_effect"* применяется операция формирования интерпретации формулы, которая формирует соответствующую конструкцию.

Шаг 81. К высказыванию *"PD_atom"* применяется операция формирования интерпретации формулы, которая формирует соответствующую конструкцию.

Шаг 82. К высказыванию *"qa1_PDda_intersection"* применяется операция формирования интерпретации формулы, которая формирует соответствующую конструкцию.

Шаг 83. К высказыванию *"query_atom1"* применяется операция формирования интерпретации формулы, которая на основании формул *"S__cause"* и *"PSA_cause"* формирует соответствующую конструкцию.

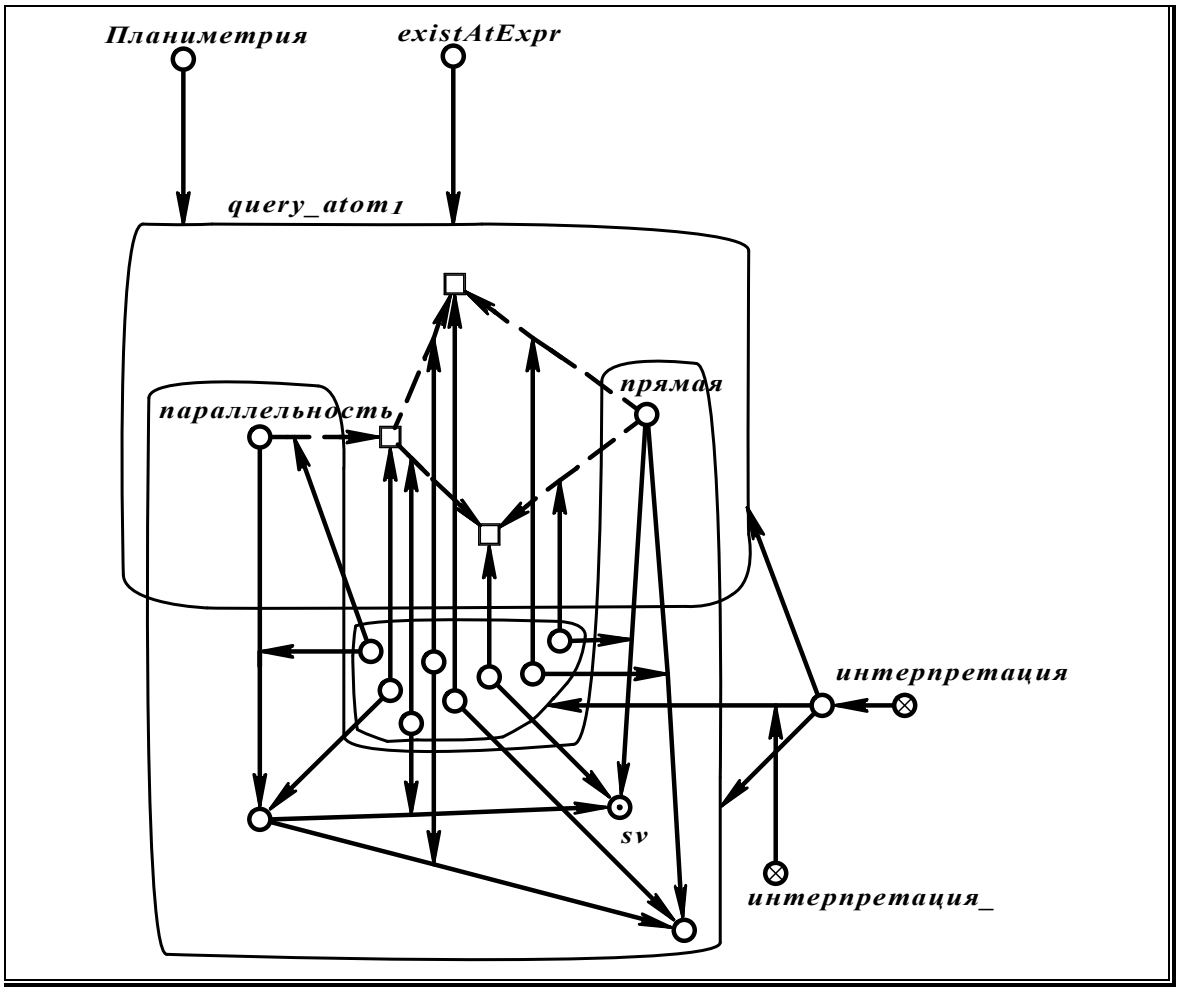
Шаг 84. К высказыванию *"query_atom1"* применяется операция вычисления значения высказывания. Высказывание приобретает истинностное значение *истина*:

```
Планиметрия !;query_atom1 ;
```

Ш а г 8 5 . Ко всем высказываниям входящим во множество ***"temporary"*** применяется операция уничтожения промежуточных конструкций, которая удаляет интерпретации и различные служебные конструкции используемые исключительно для вывода.

К о н е ц р е ш е н и я з а д а ч и

Приведём фрагмент результирующего состояния памяти графодинамической ассоциативной машины вывода, являющийся положительным ответом на поставленный нами запрос.



В процессе решения будет сформирована конструкция, поясняющая результат решения.

Выводы к разделу 8

Рассмотренная в данном разделе графодинамическая ассоциативная машина вывода является основой систем управления базами знаний, в которых информация представляется на языке SCL. Такие системы баз знаний являются неотъемлимой частью любой интеллектуальной системы. Это справедливо для интеллектуальных обучающих систем, систем построения виртуальных организаций и любых прикладных экспертных систем. Система операций машины логического вывода открыта для дополнения, и в случае реализации новых механизмов вывода, такие механизмы легко могут быть интегрированы (особенно в случае использования языка SCP в качестве языка микропрограммирования) с описанной выше графодинамической ассоциативной машиной вывода.