Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра интеллектуальных информационных технологий

ЛОГИЧЕСКИЕ ОСНОВЫ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия по выполнению курсового проекта для специальности 1-40 03 01 «Искусственный интеллект»

Минск БГУИР 2015

УДК 004.8:164(076)
ББК 32.813я7+22.12я7
Л69

Авторы:
В. В. Голенков, Н. А. Гулякина, Н. В. Гракова, И. Т. Давыденко,
И. И. Жуков, Д. В. Шункевич

Рецензенты:
кафедра интеллектуальных систем
Белорусского государственного университета
(протокол №11 от 01.04.2014);

старший научный сотрудник государственного научного учреждения
«Объединенный институт проблем информатики Национальной академии
наук Беларуси», кандидат технических наук, доцент Н. А. Кириенко;

заведующий кафедрой экономической информатики учреждения образования
«Белорусский государственный университет информатики
и радиоэлектроники», кандидат технических наук, доцент В. Н. Комличенко

Логические основы интеллектуальных систем : пособие /
Л69 В. В. Голенков [и др.]. – Минск : БГУИР, 2015. – 63 с. : ил.
ISBN 978-985-543-097-2.

Сформулированы основные положения, касающиеся выполнения курсового проектирования, даны рекомендации по решению типовых задач в рамках курсового проекта, а также рассмотрены типичные ошибки и проблемы, возникающие в процессе выполнения.

УДК 004.8:164(076)
ББК 32.813я7+22.12я7

ISBN 978-985-543-097-2

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При изучении дисциплины «Логические основы интеллектуальных систем» студенты получают навыки решения логических задач и использования логического аппарата для представления и обработки информации, изучают основные виды классических и неклассических логик в качестве формальной основы для разработки инструментальных средств проектирования прикладных интеллектуальных систем, а также теоретические основы разработки баз знаний и создания прикладных интеллектуальных систем для трудноформализуемых предметных областей.

Целью выполнения курсового проекта по данной дисциплине является закрепление навыков использования неклассических и прикладных логик в интеллектуальных системах, моделей правдоподобных рассуждений для решения задач искусственного интеллекта, традиционных программных инструментальных средств для решения логических задач, а также использования логических моделей представления и переработки информации.

Для достижения цели необходимо решить следующие задачи:

1) спроектировать набор операций логического вывода в рамках некоторой предметной области;
2) разработать алгоритмы работы для набора операций, указанного в первой задаче;
3) освоить принципы работы с необходимым программным обеспечением;
4) разработать тестовые фрагменты базы знаний для отладки и верификации разработанных операций;
5) реализовать и отладить спроектированный набор операций при помощи предлагаемого инструментария.

Пособие включает в себя:

1) описание программной оболочки py_ui, ее основных команд и принципов работы с ней;
2) примеры добавления пунктов меню и операций обработки знаний в систему;
3) типовые примеры поисковых операций для некоторой предметной области.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Логика – наука, предметом изучения которой являются общие схемы верного человеческого мышления. Математическая логика – часть логики, использующая математические методы и модели для изучения предмета логики. Математическая логика изучает логические формы мышления и такие понятия, как истина и ложь, с помощью логических констант, логических переменных, логических связок, логических функций и операций. В математической логике выделяют классическую логику, которая включает логику предикатов и логику высказываний, т. е. упрощения (частный случай логики предикатов). Другие модели математической логики имеют название неклассических логических моделей. Являясь основой символьно-логического подхода, одного из главных в искусственном интеллекте наравне с нейро-бионическим, математическая логика также совмещает такие различные подходы, как содержательно-алгебраический и формальный. Однако, как можно выявить, противопоставление этих подходов зачастую условное и относительное. Поскольку логические принципы, которые исследуются математической логикой, являются не только основой человеческого и, в частности, строгого математического мышления, но и архитектурным базисом подавляющей части современных вычислительных систем, все результаты, расчеты на этих системах получены благодаря логике. В силу этого данная дисциплина является одной из фундаментальных в науке, и этим обосновывается ее постоянная актуальность.

В рамках курсового проекта основной задачей является разработка набора операций (или агентов) логического вывода.

Любой агент представляет собой открытую систему, помещенную в некоторую среду, причем эта система обладает собственным поведением, удовлетворяющим некоторым экстремальным принципам. Таким образом, агент считается способным воспринимать информацию из внешней среды с ограниченным разрешением, обрабатывать ее на основе собственных ресурсов, взаимодействовать с другими агентами и действовать на среду в течение некоторого времени, преследуя свои собственные цели.

В рамках предлагаемого в курсовом проекте подхода любая машина обработки знаний, в том числе машина логического вывода, представляет собой графодинамическую sc-машину (память в качестве модели представления знаний использует семантическую сеть), состоящую из двух частей:

1) графодинамической sc-памяти;
2) системы sc-операций.

Условием инициирования каждой из операций является появление в памяти системы некоторой определенной конструкции [1–3].

В качестве модели представления знаний в интеллектуальных системах, разрабатываемых по предлагаемой технологии, используется семантическая сеть.

Семантическая сеть – это ориентированный граф, вершины которого – понятия, а дуги – отношения между ними [4].

Для того чтобы отлаживать реализованную систему операций логического вывода, необходимо также разработать набор тестовых фрагментов базы знаний по выбранной предметной области.

База знаний – совокупность знаний предметной области, записанная на машинный носитель в форме, понятной эксперту и пользователю (обычно на некотором языке, приближенном к естественному). Параллельно такому «человеческому» представлению существует база знаний во внутреннем «машинном» представлении [4].

Принципы проектирования баз знаний на основе предлагаемых средств описаны в работах [1, 5, 6].

При разработке операций логического вывода рекомендуется ориентироваться на классические дедуктивные методы логического вывода, рассматриваемые в рамках данного курса.

В разрабатываемой системе понятие программного агента в целом конкретизируется до понятия абстрактного sc-агента. Под абстрактным sc-агентом понимается некоторый класс функционально эквивалентных sc-агентов, разные экземпляры (т. е. представители) которого могут быть реализованы по-разному.

Каждый абстрактный sc-агент имеет соответствующую ему спецификацию. В спецификацию каждого абстрактного sc-агента входят:

1) указание ключевых sc-элементов этого sc-агента, т. е. тех sc-элементов, хранимых в sc-памяти, которые для данного sc-агента являются «точками опоры»;

2) формальное описание условий инициирования данного sc-агента,

3) т. е. тех ситуаций в sc-памяти, которые инициируют деятельность данного sc-агента;

4) формальное описание первичного условия инициирования данного sc-агента, т. е. такой ситуации в sc-памяти, которая побуждает sc-агента перейти в активное состояние и начать проверку наличия своего полного условия инициирования;

5) строгое, полное, однозначно понимаемое описание деятельности данного sc-агента, оформленное при помощи каких-либо понятных, общепринятых средств, не требующих специального изучения, например на естественном языке;

6) описание результатов выполнения данного sc-агента.

Абстрактные sc-агенты делятся на неатомарные и атомарные абстрактные sc-агенты. Под неатомарным абстрактным sc-агентом понимается абстрактный sc-агент, который декомпозируется на коллектив более простых абстрактных sc-агентов, каждый из которых в свою очередь может быть как атомарным абстрактным sc-агентом, так и неатомарным абстрактным sc-агентом.

Под неатомарным абстрактным sc-агентом понимается абстрактный sc-агент, который декомпозируется на коллектив более простых абстрактных sc-

агентов, каждый из которых в свою очередь может быть как атомарным абстрактным sc-агентом, так и неатомарным абстрактным sc-агентом. При этом в каком-либо варианте декомпозиции абстрактного sc-агента дочерний неатомарный абстрактный sc-агент может стать атомарным абстрактным sc-агентом и реализовываться соответствующим образом.

Отношение декомпозиции абстрактного sc-агента трактует неатомарные абстрактные sc-агенты как коллективы более простых абстрактных sc-агентов, взаимодействующих через sc-память.

Другими словами, декомпозиция абстрактного sc-агента на абстрактные sc-агенты более низкого уровня уточняет один из возможных подходов к реализации этого абстрактного sc-агента путем построения коллектива более простых абстрактных sc-агентов.

Под атомарным абстрактным sc-агентом понимается абстрактный sc-агент, для которого уточняется платформа его реализации, т. е. существует соответствующая связка отношения «программа sc-агента».

Понятие абстрактного sc-агента разбивается на два подкласса:

– платформенно-независимый абстрактный sc-агент;
– платформенно-зависимый абстрактный sc-агент.

К платформенно-независимым абстрактных sc-агентам относят атомарные абстрактные sc-агенты, реализованные на базовом языке программирования технологии OSTIS, т. е. на языке scp.

При описании платформенно-независимых абстрактных sc-агентов под платформенной независимостью понимается платформенная независимость с точки зрения технологии OSTIS, т. е. реализация на специализированном языке программирования, ориентированном на обработку семантических сетей (языке scp), поскольку атомарные sc-агенты, реализованные на указанном языке, могут свободно переноситься с одной платформы интерпретации sc-моделей на другую. При этом языки программирования, традиционно считающиеся платформенно-независимыми, в данном случае не могут считаться таковыми.

К платформенно-зависимым абстрактным sc-агентам относят атомарные абстрактные sc-агенты, реализованные ниже уровня sc-моделей, т. е. не на языке scp, а на каком-либо другом языке описания программ.

Существуют sc-агенты, которые принципиально должны быть реализованы на платформенно-зависимом уровне, например, собственно sc-агенты интерпретации sc-моделей или рецепторные и эффекторные sc-агенты, обеспечивающие взаимодействие с внешней средой.

Под sc-агентом понимается конкретный экземпляр (с теоретико-множественной точки зрения – элемент) некоторого атомарного абстрактного sc-агента, работающий в какой-либо конкретной интеллектуальной системе.

При этом каждый sc-агент инициируется при появлении в sc-памяти конкретной ситуации, соответствующей условию инициирования атомарного абстрактного sc-агента, экземпляром которого является заданный sc-агент. В данном случае можно провести аналогию между принципами объектно-

ориентированного программирования, рассматривая атомарный абстрактный sc-агент как класс, а конкретный sc-агент – как экземпляр, конкретную имплементацию этого класса.

Взаимодействие sc-агентов осуществляется только через sc-память. Как следствие, результатом работы любого sc-агента является некоторое изменение состояния sc-памяти, т. е. удаление либо генерация каких-либо sc-элементов.

В общем случае один sc-агент может явно передать управление другому sc-агенту, если этот sc-агент априори известен. Для этого каждый sc-агент в sc-памяти имеет обозначающий его sc-узел, с которым можно связать конкретную ситуацию в текущем состоянии базы знаний, которую инициируемый sc-агент должен обработать.

Однако далеко не всегда легко определить того sc-агента, который должен принять управление от заданного sc-агента, в связи с чем описанная выше ситуация возникает крайне редко. Более того, иногда условие инициирования sc-агента является результатом деятельности непредсказуемой группы sc-агентов, равно как и одна и та же конструкция может являться условием инициирования целой группы sc-агентов.

При этом общаются через sc-память не программы sc-агентов, а сами описываемые данными программами sc-агенты.

Под активным sc-агентом понимается sc-агент интеллектуальной системы, который реагирует на события, соответствующие его условию инициирования, и, как следствие, его первичному условию инициирования. Не входящие во множество активных sc-агентов sc-агенты не реагируют ни на какие события в sc-памяти.

Если связки отношения «ключевые sc-элементы sc-агента» связывают между собой sc-узел, обозначающий абстрактный sc-агент, и sc-узел, обозначающий множество sc-элементов, которые являются ключевыми для данного абстрактного sc-агента, то данные sc-элементы явно упоминаются в рамках программ, реализующих данный абстрактный sc-агент.

Связки отношения «программа sc-агента» связывают между собой sc-узел, обозначающий атомарный абстрактный sc-агент, и sc-узел, обозначающий множество программ, реализующих указанный атомарный абстрактный sc-агент.

В случае платформенно-независимого абстрактного sc-агента каждая связка отношения «программа sc-агента» связывает sc-узел, обозначающий указанный абстрактный sc-агент с множеством scp-программ, описывающих деятельность данного абстрактного sc-агента. Данное множество содержит одну агентную scp-программу и произвольное количество (может быть, и ни одной) scp-программ, которые необходимы для выполнения указанной агентной scp-программы.

В случае платформенно-зависимого абстрактного sc-агента каждая связка отношения «программа sc-агента» связывает sc-узел, обозначающий указанный абстрактный sc-агент, с множеством файлов, содержащих исходные тексты

программы на некотором внешнем языке программирования, реализующей деятельность данного абстрактного sc-агента.

Связки отношения «первичное условие инициирования» связывают между собой sc-узел, обозначающий абстрактный sc-агент, и бинарную ориентированную пару, описывающую первичное условие инициирования данного абстрактного sc-агента, т. е. такой ситуации в sc-памяти, которая побуждает sc-агента перейти в активное состояние и начать проверку наличия своего полного условия инициирования.

Первым компонентом данной ориентированной пары является знак некоторого подмножества понятия sc-событие, например sc-событие добавления выходящей sc-дуги, т. е. по сути конкретный тип события в sc-памяти.

Вторым компонентом данной ориентированной пары является произвольный в общем случае sc-элемент, с которым непосредственно связан указанный тип события в sc-памяти, т. е., например, sc-элемент, из которого выходит либо в который входит генерируемая либо удаляемая sc-дуга, либо sc-ссылка, содержимое которой было изменено.

После того как в sc-памяти происходит некоторое sc-событие, активизируются все активные sc-агенты, первичное условие инициирования которых соответствует произошедшему sc-событию.

Связки отношения «условие инициирования и результат» связывают между собой sc-узел, обозначающий абстрактный sc-агент, и бинарную ориентированную пару, связывающую условие инициирования данного абстрактного sc-агента и результаты выполнения данного экземпляра данного sc-агента в какой-либо конкретной системе.

Указанную ориентированную пару можно рассматривать как логическую связку импликации, при этом на sc-переменные, присутствующие в обеих частях связки, неявно накладывается квантор всеобщности, а на sc-переменные, присутствующие либо только в посылке, либо только в заключении, неявно накладывается квантор существования.

Первым компонентом указанной ориентированной пары является логическая формула, описывающая условие инициирования описываемого абстрактного sc-агента, т. е. конструкции, наличие которой в sc-памяти побуждает sc-агента начать работу по изменению состояния sc-памяти. Данная логическая формула может быть как атомарной, так и неатомарной, в которой допускается использование любых связок логического языка.

Вторым компонентом указанной ориентированной пары является логическая формула, описывающая возможные результаты выполнения описываемого абстрактного sc-агента, т. е. описание произведенных им изменений состояния sc-памяти. Данная логическая формула может быть как атомарной, так и неатомарной, в которой допускается использование любых связок логического языка.

Sc-описание поведения sc-агента представляет собой sc-окрестность, описывающую деятельность sc-агента до какой-либо степени детализации, однако такое описание должно быть строгим, полным и однозначно понимаемым. Как любая другая sc-окрестность, sc-описание поведения sc-агента может быть протранслировано на какие-либо понятные, общепринятые средства, не требующие специального изучения, например на естественный язык.

Описываемый абстрактный sc-агент входит в sc-описание поведения sc-агента под атрибутом ключевого sc-элемента.

Под sc-событием понимается элементарное изменение состояния sc-памяти, которое может стать первичным условием инициирования какого-либо sc-агента.

Класс sc-событий в целом разбивается на следующие подклассы:

- sc-событие добавления выходящей sc-дуги;
- sc-событие добавления входящей sc-дуги;
- sc-событие добавления инцидентного sc-ребра;
- sc-событие удаления выходящей sc-дуги;
- sc-событие удаления входящей sc-дуги;
- sc-событие удаления инцидентного sc-ребра;
- sc-событие удаления sc-элемента;
- sc-событие изменения содержимого sc-ссылки.

Рассмотрим ряд принципов, соблюдать которые рекомендуется при проектировании операций логического вывода.

Каждая операция должна быть по возможности предметно-независимой, т. е. по возможности меньше опираться на константы, имеющие отношение непосредственно к рассматриваемой предметной области. Исключение составляют понятия, которые могут использоваться в различных предметных областях. Данное правило может также быть нарушено в случае, если операция является вспомогательной и ориентирована на обработку какого-либо конкретного класса объектов (например, арифметические операции могут напрямую работать с конкретными отношениями «сложение» и «умножение» и т. п.).

Операция должна по возможности меньше ориентироваться на фиксированную форму представления фрагментов базы знаний, на работу с которыми она ориентирована. Степень глубины формализации и другие аспекты базы знаний определяются инженером по знаниям и не должны влиять на корректность работы операции. При обнаружении некорректности в представлении рассматриваемого фрагмента базы знаний операция, как и вся система, не должна терять управления и по возможности сообщить пользователю о некорректности приведенных знаний.

Одна операция может состоять из ряда подпрограмм на выбранном языке программирования. Не стоит путать понятия «операция» и «программа» («подпрограмма»). Подпрограмма не является агентом и должна вызываться

другой подпрограммой. Операция сопоставляется с как минимум одной подпрограммой, которая имеет особый формат входных и выходных данных, автоматически реагирует на состояние sc-памяти и при необходимости запускает другие подпрограммы. При проектировании подпрограмм следует учитывать возможность использования различными операциями одних и тех же подпрограмм.

Каждая операция должна самостоятельно проверять полноту соответствия условия инициирования конструкции, имеющейся в памяти системы на данный момент. В процессе работы системы может возникнуть ситуация, когда на появление одной и той же конструкции среагировали несколько операций. В таком случае выполнение продолжает только та операция, условие инициирования которой полностью соответствует сложившейся ситуации.

Если в процессе работы операция генерирует в памяти какие-либо временные конструкции, то при завершении работы она обязана удалять всю информацию, использование которой в системе более нецелесообразно. Исключение составляют ситуации, когда подобная информация необходима нескольким операциям для решения одной общей задачи, однако позже информация становится бесполезной или избыточной и требует удаления. В данном случае ни одна из операций не может оказаться в состоянии удалить информационный мусор. В таком случае возникает необходимость говорить о специализированных вспомогательных операциях, задачей которых является уничтожение информационного мусора.

При необходимости операции можно объединять в группы для решения более сложных задач. Очевидно, что такого рода задачи могут рассматриваться в рамках практически любой предметной области. Подобная группа операций является в некотором смысле самостоятельной подсистемой в рамках целостной системы операций. При объединении операций в группы рекомендуется проектировать их таким образом, чтобы они могли быть использованы не только в рассматриваемой группе, и, будучи отделенными от группы, не теряли смысл.

Инициатором запуска какой-либо операции может быть как непосредственно пользователь системы, так и другая операция. При этом за счет организации взаимодействия через общую память это никак не отражается в алгоритме работы самой операции. Необходимость вывода (трансляции) какого-либо фрагмента памяти конечному пользователю отслеживается компонентами пользовательского интерфейса самостоятельно.

Язык взаимодействия операций через sc-память описан в [7]. В вопросе должна указываться только критически важная информация, т. е. параметры запроса, все остальные знания операция способна найти самостоятельно, анализируя память в семантической окрестности вопроса. Это позволяет уменьшить сложность восприятия принципов работы с операцией потенциальным пользователем и унифицировать форматы вопросов у большого числа различных операций.

При разработке алгоритмов работы проектируемых операций необходимо учитывать ряд факторов:

– операции должны быть как можно более универсальными, т. е. использоваться при решении как можно большего числа задач, что позволит избежать повторной реализации одних и тех же фрагментов рассуждений и уменьшит количество хранимых в системе знаний;

– операции должны быть по возможности антропоморфными, т. е. одна операция должна моделировать некий единый законченный акт мыслительной деятельности человека. Не следует искусственно увязывать ряд действий в одну операцию и наоборот, расчленять одно самодостаточное действие на поддействия. Это вызовет сложности восприятия принципов работы операции разработчиками и не позволит использовать операцию в ряде систем (например, в обучающих системах, которые должны объяснять ход решения пользователю).

Таким образом, в процессе разработке системы операций можно выделить следующие этапы:

– определение необходимого набора операций;

– определение ключевых узлов языка вопросов для связи операций посредством графодинамической памяти;

– разработка алгоритма работы каждой из операций;

– реализация и тестирование коллектива операций.

Приведем краткую классификацию существующих методов логического вывода, рассматриваемых в литературе по искусственному интеллекту.

1. *Классический дедуктивный вывод* является наиболее популярным при построении автоматических решателей задач, т. к. всегда дает достоверный результат. Дедуктивный вывод включает в себя прямой и обратный и логический выводы (принцип резолюции, процедуру Эрбрана и др.), все виды силлогизмов и т. д. Основной проблемой дедуктивного вывода является невозможность его использования в ряде случаев, когда отсутствуют достоверные правила вывода.

2. *Индуктивный вывод* предоставляет возможность в процессе решения использовать различные предположения, что делает его удобным для использования в слабоформализованных и трудноформализуемых предметных областях, например при построении систем медицинской диагностики.

3. *Абдуктивный вывод*. Под абдуктивным выводом в искусственном интеллекте, как правило, понимается вывод наилучшего абдуктивного объяснения, т. е. объяснения некоторого события, ставшего неожиданным для системы. Причем «наилучшим» считается такое объяснение, которое удовлетворяет специальным критериям, определяемым в зависимости от решаемой задачи и используемой формализации.

4. *Нечеткие логики*. Теория нечетких множеств и соответственно нечетких логик, также применяется в системах, связанных с трудноформализуемыми предметными областями.
5. *Логика умолчаний* применяется в том числе для того, чтобы оптимизировать процесс рассуждений, дополняя процесс достоверного вывода вероятностными предположениями в тех случаях, когда вероятность ошибки крайне мала.
6. *Темпоральная логика*. Применение темпоральной логики является очень актуальным для нестатичных предметных областей, в которых истинность того или иного утверждения меняется со временем, что существенно влияет на ход решения какой-либо задачи. Следует отметить, что используемый в данной работе язык представления знаний предоставляет все необходимые возможности для описания таких динамических предметных областей.

1. ЗНАКОМСТВО С ПРОГРАММНЫМИ СРЕДСТВАМИ

Для начала работы необходимо следующее:

- архив с sc-core. В зависимости от разрядности системы используются разные sc-core:

32x – https://www.dropbox.com/sh/7zi37thai67badz/fFZBNe6a4w,
64x – https://www.dropbox.com/sh/xojxciamrxcpi4v/BiU7arw5l1;

- файл m4scp.m4 (https://www.dropbox.com/s/1alqgadjp6n959k/m4scp.m4);
- python-2.6.2c1;
- архив Python26 (питон со всеми установленными библиотеками https://www.dropbox.com/s/ic9wgcorhzhcn9p/Python26.rar);
- архив с py_ui (https://www.dropbox.com/s/0o639i3gynn3as9/py_ui.rar).

1.1. Установка необходимого программного обеспечения

1.1.1. *Установка sc-core*

1. Скачать архив, соответствующий вашей системе. Архивы называются соответственно: **sc-core-32.rar** и **sc-core-64.rar**. Также скачать файл **m4scp.m4**.
2. Распаковать архив. Заменить скачанным файлом **m4scp.m4** файл, лежащий по следующему пути: sc-core-32/share/sc-core/m4scp.m4.
3. Создать переменную окружения SC_CORE_HOME, в которой прописать путь к папке, где лежит sc-core (рис. 1.1).

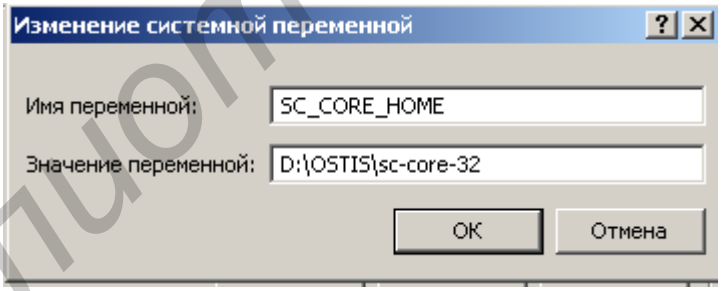

Рис. 1.1. Системная переменная SC_CORE_HOME

4. sc-core установлен.

1.1.2. *Установка Python*

1. Скачать и запустить программу установки python-2.6.2c1.msi. Для корректной работы программы необходима именно эта версия.
2. Добавить путь к папке с установленным Python в переменную окружения PATH.
3. Для работы py-ui необходимо некоторое количество библиотек, которых нет в стандартной комплектации Python. Скачать архив

Pyton26.rar и распаковать его в папку, где стоит Python. При появлении сообщения о замене файлов подтвердить замену.

1.1.3. *Установка py_ui*

1. Скачать и распаковать архив **py_ui.rar**. Полученная папка py_ui_geom и есть система для работы.
2. Запустить start.py. Должна появиться консоль, а через некоторое время – окно с настройками (рис. 1.2).

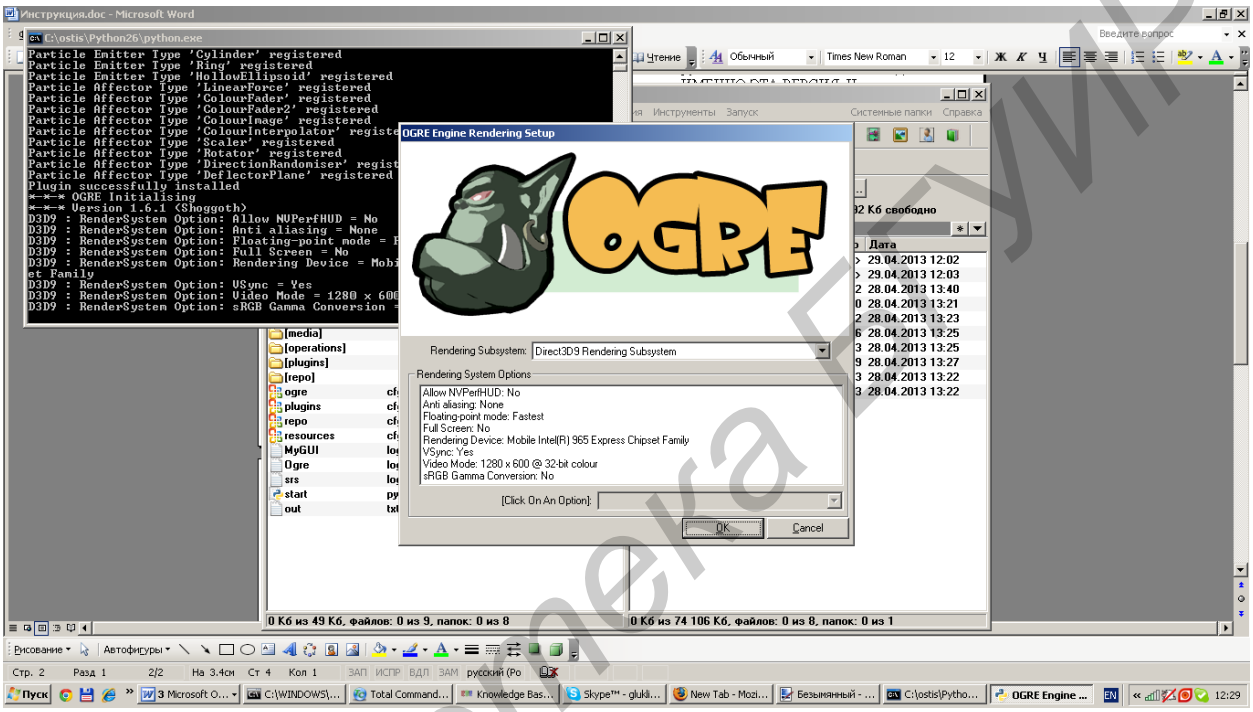

Рис. 1.2. Окно настроек py-ui

3. В выпадающем меню Rendering Subsystem нужно выбрать Direct3D9 Rendering Subsystem. В меню Rendering Device указать свой видеоадаптер. Нажать OK, должен появиться прогресс-бар загрузки, а потом – окно программы (рис. 1.3).

Рис. 1.3. Окно графического интерфейса py_ui

1.2. Краткие сведения о работе с интерфейсом py_ui

1.2.1. *База знаний*

Исходные тексты базы знаний системы лежат в папке repo/fs_repo_src. Основные элементы:

py_ui_geom\repo\fs_repo_src\etc\questions_src\questions_keynodes.gwf – описание ключевых узлов вопросов;

py_ui_geom\repo\fs_repo_src\include\etc_questions.scsy – файл для описания сокращенных имен узлов вопросов;

py_ui_geom\repo\fs_repo_src\include\com_keynodes.scsy – файл для описания сокращенных имен узлов отношений и других ключевых узлов базы знаний;

py_ui_geom\repo\fs_repo_src\operation – в этой папке находятся созданные scp-операции;

py_ui_geom\repo\fs_repo_src\seb\planimetry_src – исходные тексты базы знаний;

py_ui_geom\repo\fs_repo_src\ui\menu – папка содержит описание структуры главного меню программы;

py_ui_geom\repo\fs_repo_src\startup.scs – в данном файле регистрируются реализованные поисковые операции, чтобы они могли реагировать на возникновение событий в памяти.

После внесения любых изменений в исходные тексты базы знаний запустить файл **py_ui_geom\repo\rule_builder.py**. Запуск данного скрипта пересобирает базу знаний. Весь процесс сборки отображается в консоли. В ней же отображаются сообщения об ошибках, в частности, синтаксических ошибках в исходных текстах scp-программ.

1.2.2. *Графический интерфейс*

Для создания узла использовать правую клавишу мыши (ПКМ).

Для создания дуги последовательно нажать ПКМ, сначала на узле, из которого проведена дуга, потом на узле, в который проведена дуга.

Для выбора узла использовать левую клавишу мыши (ЛКМ).

Для изменения идентификатора узла выбрать узел при помощи ЛКМ и нажать клавишу I. Появится окно ввода идентификатора.

Для изменения содержимого узла выбрать узел и нажать кнопку C. Далее необходимо выбрать тип содержимого и задать его.

Для просмотра содержимого узла использовать клавишу H.

Для изменения типа узла выбрать узел и нажать T. После в появившемся окне выбрать тип узла.

Для удаления узла выбрать узел и нажать Delete.

Перед тем как осуществлять запросы к созданным узлам или использовать их в качестве аргументов операций, созданные объекты необходимо поместить в память. Это осуществляется сочетанием клавиш **Ctrl+Enter**. После этого объекты, которые уже находились в памяти (т. к. описаны в базе знаний или были помещены в память ранее), меняют свой цвет на синий, а вновь созданные объекты – на зеленый. На рис. 1.4 узел test_node уже находился в памяти, а безымянный узел был создан.

Рис. 1.4. Внешний вид узлов после загрузки листа в память

После того как содержимое листа было загружено в память, можно осуществлять вызов поисковых операций для этих объектов. В качестве примера рассмотрим поиск всех позитивных выходящих дуг.

Пример:

1. Создать в базе знаний структуру, которую будем использовать для демонстрации. Для этого в папке **py_ui_geom\repo\fs_repo_src\seb\planimetry_src** создать gwf-файл, содержащий конструкцию следующего вида (рис. 1.5):

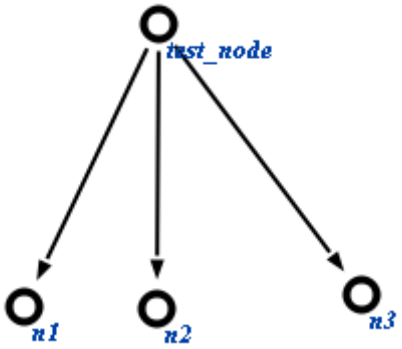

Рис. 1.5. Конструкция базы знаний для демонстрации поиска выходящих дуг

2. Запустить скрипт **rule_builder.py** и, дождавшись окончания его работы, убедиться, что база собрана без ошибок (рис. 1.6).

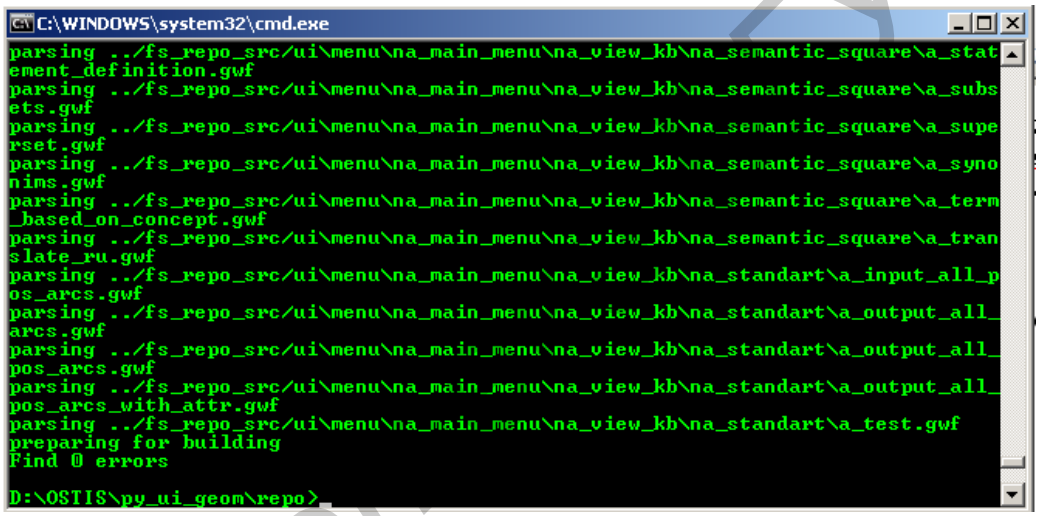

Рис. 1.6. Консоль сборки базы знаний

3. Запустить пользовательский интерфейс, создать узел и назвать его test_node (рис. 1.7). Погрузить содержимое листа в память.

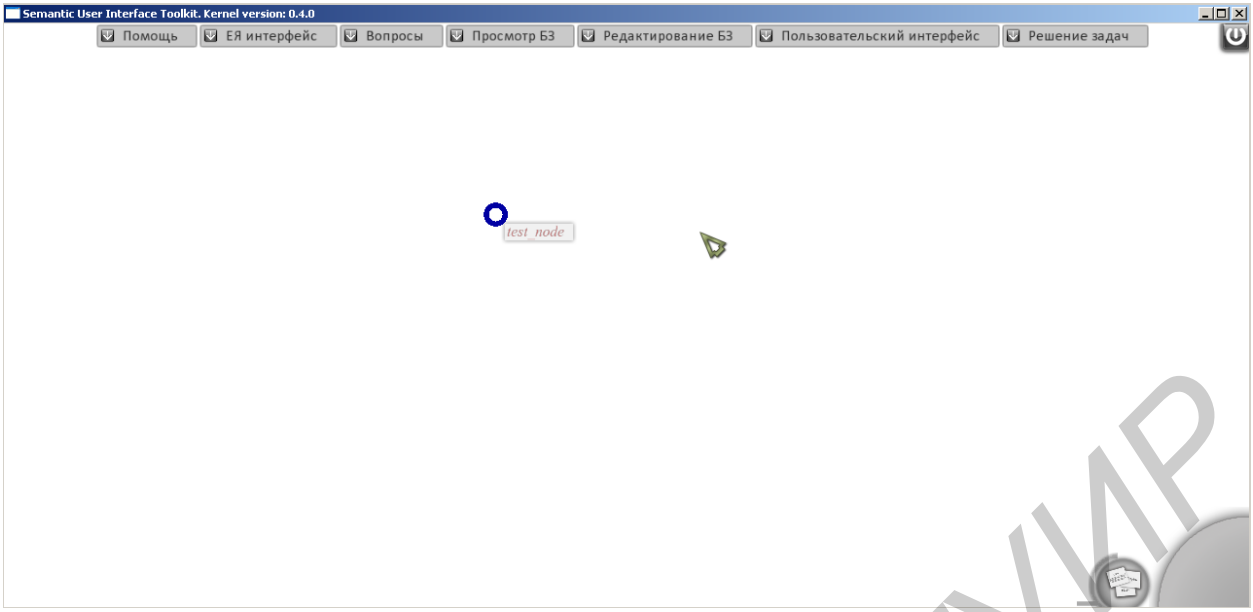

Рис. 1.7. Созданный узел test_node

4. Указать созданный узел как аргумент поисковой операции. Для этого кликнуть по нему ЛКМ, удерживая нажатой клавишу Alt. Если возле узла появилась «1», то узел будет использован в качестве первого аргумента поисковой операции (рис. 1.8).

Рис. 1.8. Узел, переданный в качестве аргумента

5. Запустить операцию при помощи команды меню: «Просмотр БЗ – Стандартные – Поиск всех позитивных выходящих дуг». По окончании работы получится следующая комбинация (рис. 1.9).

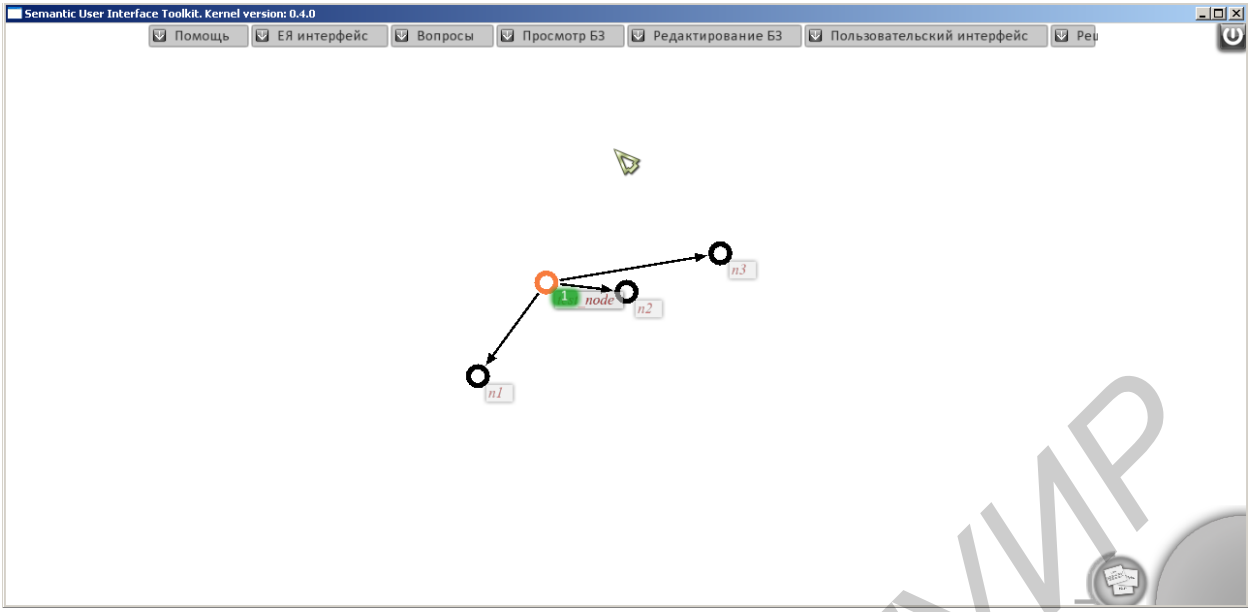

Рис. 1.9. Найденные выходящие дуги

2. ПЕРВАЯ ПРОГРАММА

Необходимо создать простую операцию, чтобы продемонстрировать последовательность шагов, которую необходимо выполнить при создании поисковой операции. Далее будет показано создание пункта меню, подключение операции к системе, получение заданного аргумента поисковой операции.

Реализованная операция будет выводить информацию об узле, переданном в качестве аргумента.

2.1. Создание команды меню

Поисковые операции в графическом интерфейсе вызываются через команды меню. Команды делятся на атомарные (которые, собственно, вызывают операции) и неатомарные (которые открывают подменю).

Создание неатомарной команды меню

Для операций, которые будут реализованы, необходимо создать неатомарное меню «Мои операции», в котором будут созданы атомарные команды для запуска реализованных операций.

1. Перейти в каталог **py_ui_geom\repo\fs_repo_src\ui\menu\na_main_menu**. В нем находится набор папок и gwf-файлов. Файлы имеют такие же названия, как и папки. Каждый файл содержит описание пункта меню, а папка – его содержимое.
2. Создать папку **na_my**. Пока она пуста, позже туда будут складываться атомарные команды.
3. Создать gwf-файл с именем **na_my**. **Имена всех неатомарных команд начинаются с na**. В этом файле поместить описание команды меню. Узел назвать так же, как папку, в которую будут помещены команды данного меню. Через отношение идентификации задать отображаемое имя пункта меню, через отношение пояснения – всплывающую подсказку. Через отношение декомпозиции будут задаваться атомарные команды и подменю. На рис. 2.1 изображено описание неатомарной команды меню.

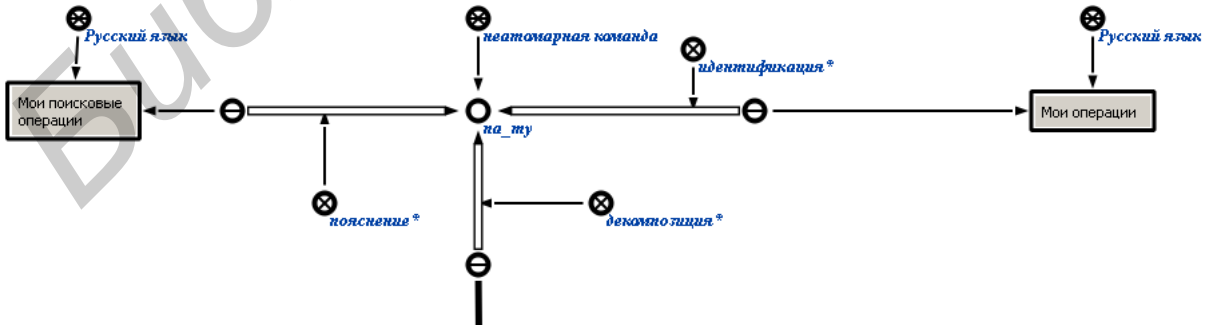

Рис. 2.1. Описание неатомарной команды меню

4. Подключить созданную неатомарную команду к главному меню. Для этого открыть файл **py_ui_geom\repo\fs_repo_src\ui\menu\na_main_menu.gwf** и добавить узел na_my в отношение декомпозиции (рис. 2.2).

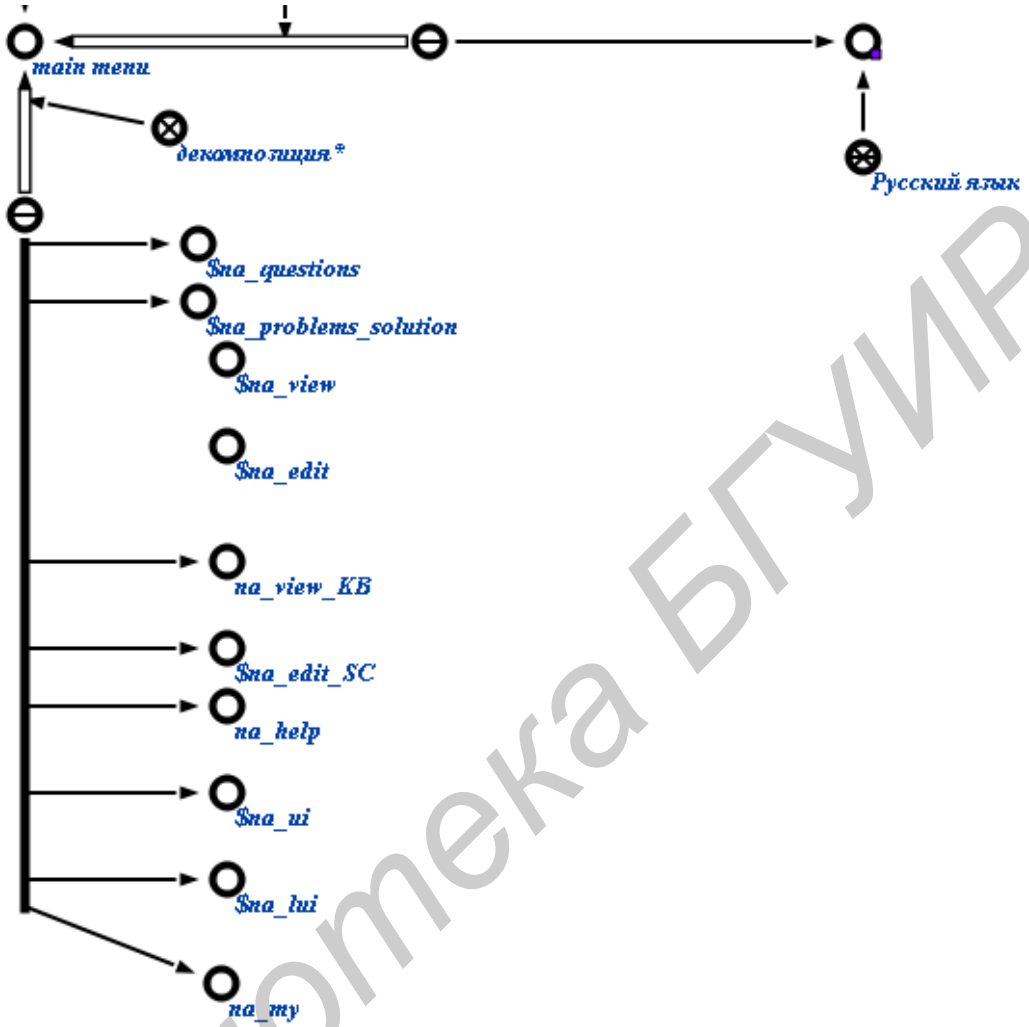

Рис. 2.2. Фрагмент базы знаний, описывающий главное меню

Пересобрать базу знаний и запустить графический интерфейс. В меню должен появиться новый пункт «Мои операции», при нажатии на который ничего не происходит. Теперь добавить в него атомарную команду, вызывающую операцию.

Создание атомарной команды меню

1. Перейти в каталог **py_ui_geom\repo\fs_repo_src\ui\menu\na_main_menu\na_my**. Данный каталог хранит содержимое меню «Мои операции», созданного ранее. Создать здесь gwf с названием **a_test**. **Все, что относится к атомарным командам, начинается с префикса a_!**
2. В созданный файл поместить следующую конструкцию (рис. 2.3).

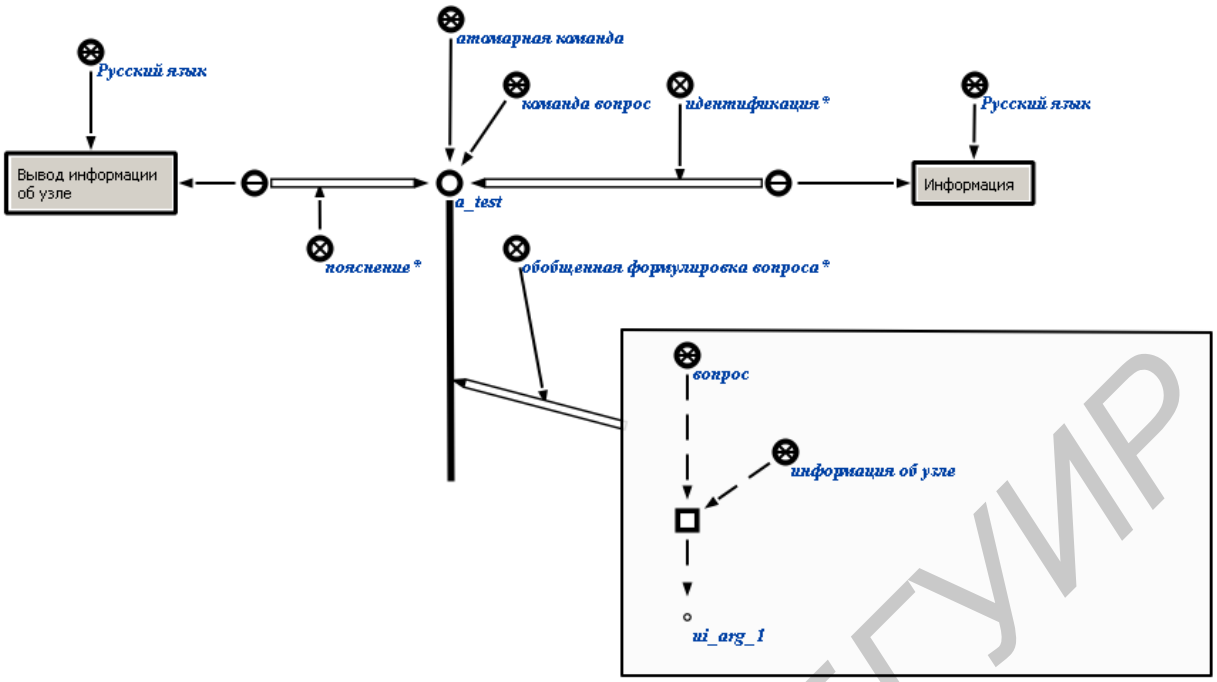

Рис. 2.3. Описание атомарной команды меню

Узел, обозначающий команду меню, именуется с использованием префикса a_. Использование отношений идентификации и пояснения такое же, как и для неатомарных команд. Новым является отношение обобщенной формулировки вопроса. В контуре изображается структура, которая появится в памяти после того, как выбирается эта команда меню. Узел «информация об узле» показывает, какой именно вопрос будет задан, т. е. описывает поисковую операцию. Узел ui_arg_1 показывает, что у данной операции будет один входной параметр (узел, информация о котором запрашивается).

Теперь если собрать базу знаний, запустить графический интерфейс и зайти в меню «Мои операции», то там будет кнопка «Информация», которая будет вызывать нашу первую операцию.

2.2. Написание кода операции

Создать файл, содержащий исходный код нашей поисковой операции. Поместить его в папку **py_ui_geom\repo\fs_repo_src\operation\my**. Назвать файл **hello_world.m4scp**. Исходный код данной операции:

```
#include "scp_keynodes.scsy"
#include "etc_questions.scsy"
#include "com_keynodes.scsy"
#include "lib_search.scsy"
#include "lib_check.scsy"
#include "lib_gen.scsy"
#include "lib_answer.scsy"
#include "lib_set.scsy"

// программа, представляющая поисковую операцию
```

```
program(init_op,
[[
     // Подпрограмма, вызываемая для выполнения операции
     hello_world;
     // Ключевой узел, обозначающий инициированный вопрос
     q_initiated;
     //  Событие,  на  которое  реагирует  обработчик(проведение
выходящей дуги из узла)
     catch_output_arc;
]],
[{
}],
{[
]}
)

// Установка обработчика события на проведение дуги из узла
// «Инициированный запрос»
sys_set_event_handler([
     1_: fixed_: catch_output_arc,
     2_: fixed_: hello_world,
     3_: fixed_: {1_: q_initiated}
])

return()

end

// Процедура, вызывающаяся при срабатывании обработчика события
procedure(hello_world,
[[

     // Ключевой узел, обозначающий запрос
     q_test;

     // Ключевой узел, обозначающий инициированный вопрос
     q_initiated;

]],
[{
     // переменные для представления входных параметров
     handler, element,
     arcFromQuestion,
     questionLink,
     // дуга между узлом, обозначающим вид вопроса, и узлом
// вопроса
     arcFromRequest,
     // дуга между узлом вопроса и аргументом
     arcVar,
     // аргумент поисковой операции
     object,
     // переменные для установки сегмента памяти, в котором
```

```
// происходит работа
     location, segments
}],
{[
     1_: in_: handler,
     2_: in_: element,
     3_: in_: arcFromQuestion,
     4_: in_: questionLink
]}
)

// Получение сегмента, в котором находится узел вопроса
sys_get_location([
     1_: fixed_: questionLink,
     2_: assign_: location
])

// Установка найденного сегмента как основного
sys_set_default_segment([
     1_: fixed_: location
])

// Разворачивание установленного сегмента
sys_spin_segment([
     1_: fixed_: location,
     2_: assign_: segments
])
// Проверяем, относится ли вопрос к множеству «Информация об узле»
searchElStr3([
     1_: fixed_: q_test,
     2_: assign_: const_: pos_: arc_: arcFromRequest,
     3_: fixed_: questionLink
], , finishOperation)


// Находим объект, информацию о котором мы хотим получить
searchElStr3([
     1_: fixed_: questionLink,
     2_: assign_: const_: pos_: arc_: arcVar,
     3_: assign_: const_: node_: object
], ,finishOperation)

// Выводим информацию об объекте
printEl([
     1_: object
])

label(finishOperation)

return()
end
```

Узел, обозначающий запрос (в примере он называется q_test), – это тот узел, который в описании команды обозначает множество, к которому принадлежит вопрос, т. е. «Информация об узле». Необходимо установить соответствие между этим узлом и используемым обозначением q_test.

Для этого сначала необходимо открыть файл **py_ui_geom\repo\fs_repo_src\etc\questions_src\questions_keynodes.gwf** и скопировать в него узел «Информация об узле» (рис. 2.4).

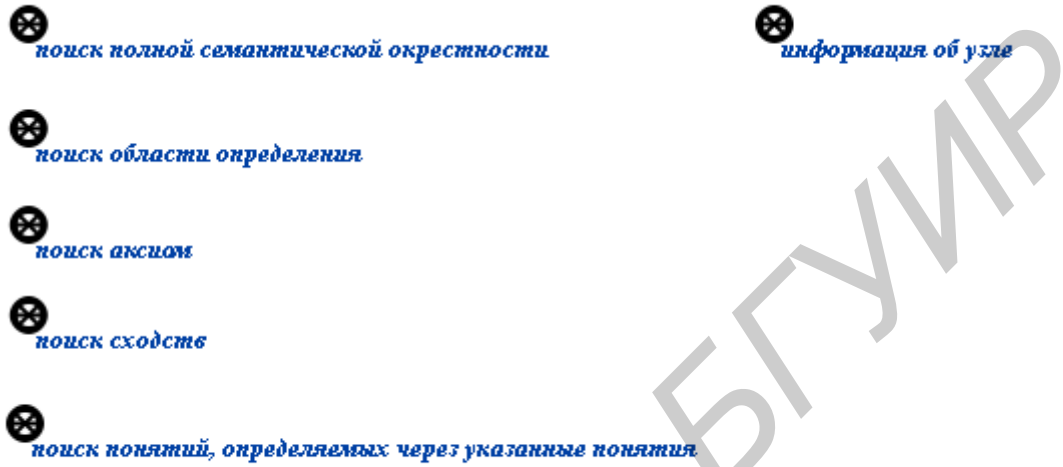

Рис. 2.4. Фрагмент файла questions.keynodes.gwf

Потом открыть файл **py_ui_geom\repo\fs_repo_src\include\etc_questions.scsy** и дописать в него следующую строку:

q_test = "/etc/questions/информация об узле";

Эта строчка и устанавливает соответствие узла вопроса и его сокращенного имени.

Финальным шагом необходимо сделать так, чтобы при запуске системы устанавливался обработчик событий для нашей операции. Для этого необходимо указать путь к программе, содержащей поисковую операцию, т. е. в файл **py_ui_geom\repo\fs_repo_src\startup.scsy** добавить следующую строку:

"/operation/my/hello_world/init_op".

2.3. Запуск операции

Теперь необходимо собрать базу знаний, запустить графический интерфейс, создать узел с идентификатором test_node, загрузить его в память и вызвать на нем написанную нами операцию с помощью пункта меню. После этого в консоли будет отображена информация об этом узле, а также о дугах, входящих и выходящих их него (рис. 2.5).

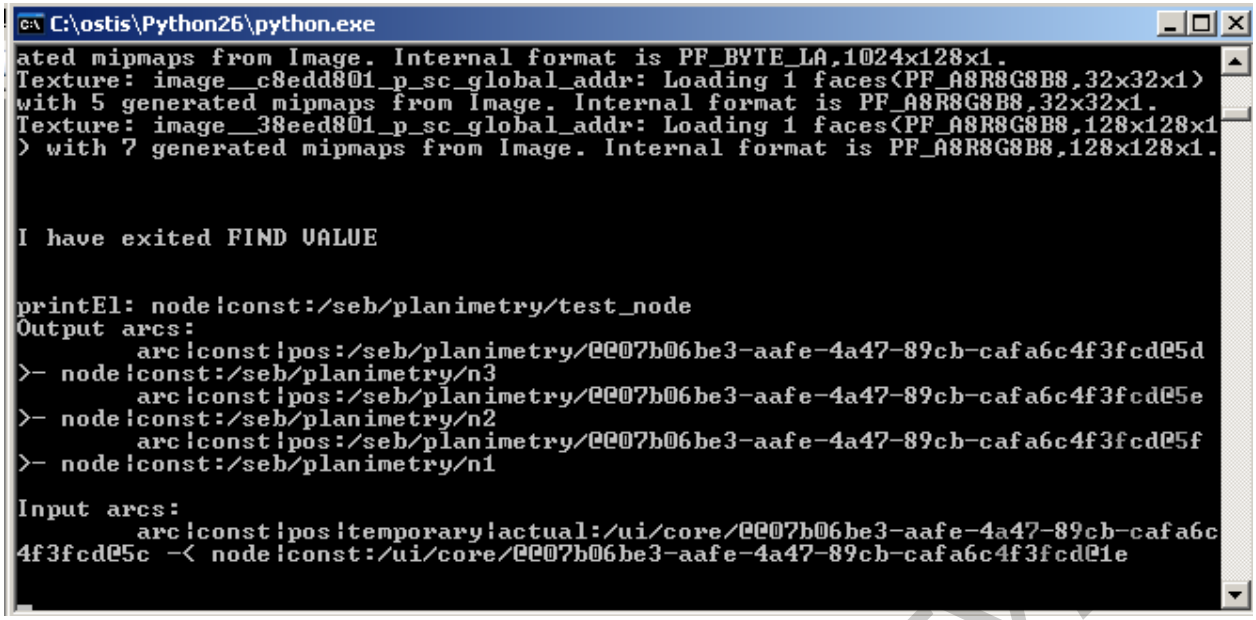

Рис. 2.5. Результат вывода информации об узле

3. ОПЕРАЦИЯ ПОИСКА ХАРАКТЕРИСТИКИ ОБЪЕКТА ПО ЗАДАННОМУ ОТНОШЕНИЮ

Рассмотрим пример операции поиска нечисловой характеристики по некоторому отношению (поиск радиуса окружности) (рис. 3.1). Тестовый пример реализован в базе знаний в файле **py_ui_geom\repo\fs_repo_src\seb\planimetry_src\Исходные данные.gwf**.

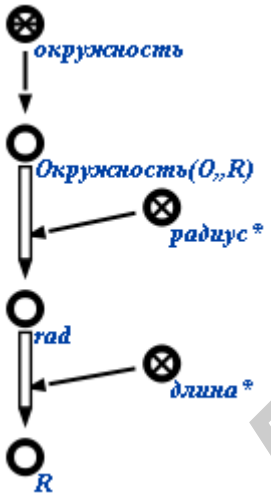

Рис. 3.1. Конструкция для проверки работы операции

Создать атомарную команду меню для запуска операции по приведенному выше алгоритму. Описание команды приведено на рис. 3.2.

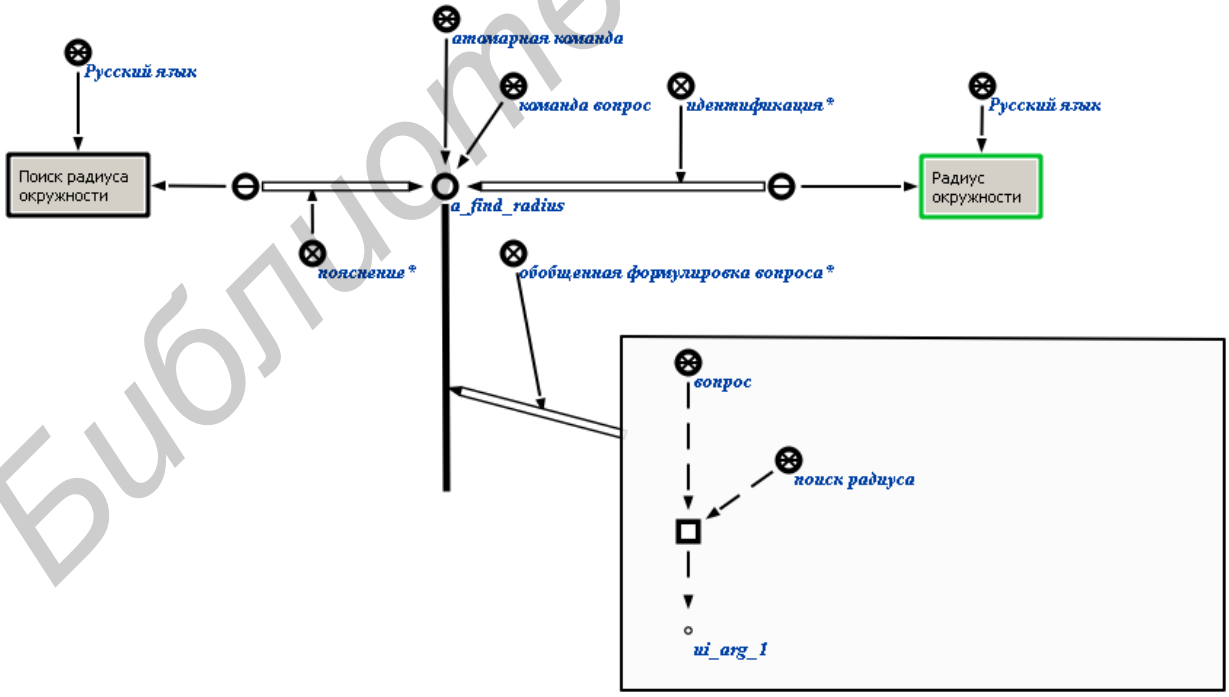

Рис. 3.2. Команда меню для запуска операции

Добавить узел a_find_radius в файл na_my.gwf (рис. 3.3).

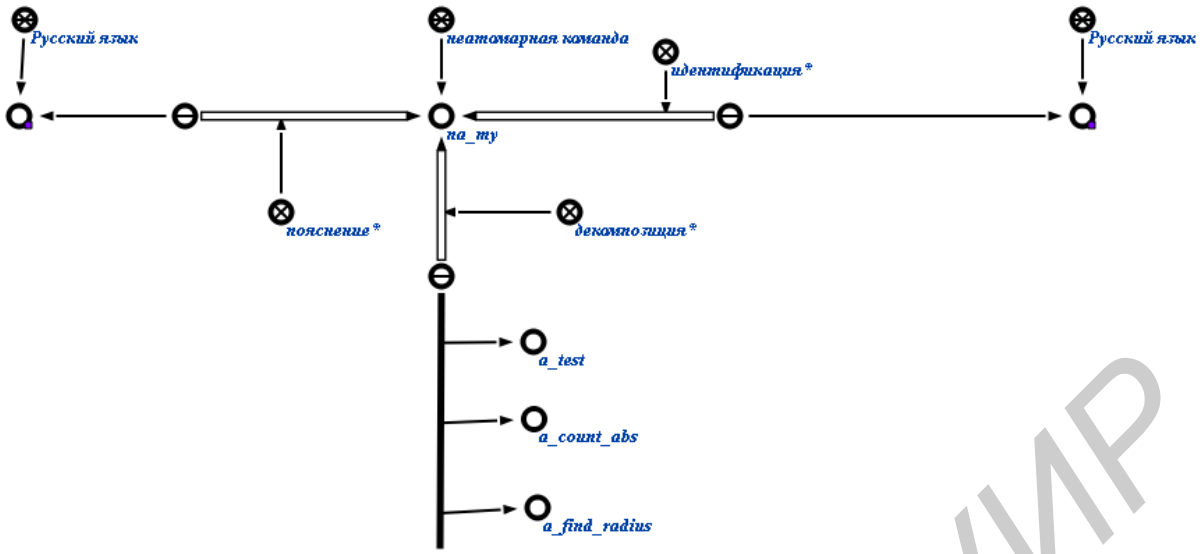

Рис. 3.3. Подключение написанного пункта меню к меню «Мои операции»

Собрать базу знаний, запустить интерфейс и убедиться, что появился новый пункт меню.

Для работы операции необходим узел типа вопроса: «поиск радиуса» и узел отношения «радиус*».

Добавить узел «поиск радиуса» в файл **py_ui_geom\repo\fs_repo_src\etc\questions_src\questions_keynodes.gwf**, а также внести в файл **py_ui_geom\repo\fs_repo_src\include\etc_questions.scsy** следующую строку:

q_radius = "/etc/questions/поиск радиуса";

Скопировать узел отношения «радиус*» в файл **py_ui_geom\repo\fs_repo_src\etc\com_keynodes_src\com_keynodes.gwf**.

После этого в файл **py_ui_geom\repo\fs_repo_src\include\com_keynodes.scsy** добавить следующую строчку:

nrel_radius = "/etc/com_keynodes/радиус*";

Теперь возможно работать с узлом отношения через имя nrel_radius.

В папке с исходными текстами операций создать файл find_radius.m4scp со следующей операцией:

```
#include "scp_keynodes.scsy"
#include "etc_questions.scsy"
#include "com_keynodes.scsy"
#include "lib_search.scsy"
#include "lib_check.scsy"
#include "lib_gen.scsy"
#include "lib_answer.scsy"
#include "lib_set.scsy"

// Программа, представляющая операцию поиска радиуса окружности
program(init_op,
[[
     // Подпрограмма, вызываемая для выполнения операции
     find_radius;
```

```
     // Ключевой узел, обозначающий инициированный вопрос
     q_initiated;
     //  Событие,  на  которое  реагирует  обработчик(проведение
выходящей дуги из узла)
     catch_output_arc;
]],
[{
}],
{[
]}
)

//  Установка  обработчика  события  на  проведение  дуги  из  узла
«Инициированный запрос»
sys_set_event_handler([
     1_: fixed_: catch_output_arc,
     2_: fixed_: find_radius,
     3_: fixed_: {1_: q_initiated}
])

return()

end

// Процедура, вызываемая при срабатывании обработчика события
procedure(find_radius,
[[

     // Ключевой узел, обозначающий запрос
     q_radius;

     // Ключевой узел, обозначающий инициированный вопрос
     q_initiated;

     // Ключевой узел, обозначающий отношение «радиус*»
     nrel_radius;

]],
[{
     // переменные для представления входных параметров
     handler, element,
     arcFromQuestion,
     questionLink,
     //  дуга  между  узлом,  обозначающим  вид  вопроса,  и  узлом
вопроса
     arcFromRequest,
     // дуга между узлом вопроса и аргументом
     arcVar,
     // аргумент поисковой операции
     object,
     //  переменные  для  установки  сегмента  памяти,  в  котором
происходит работа
```

```
     location, segments,

     radius,
     answer
}],
{[
     1_: in_: handler,
     2_: in_: element,
     3_: in_: arcFromQuestion,
     4_: in_: questionLink
]}
)

// Получение сегмента, в котором находится узел вопроса
sys_get_location([
     1_: fixed_: nrel_radius,
     2_: assign_: location
])

// Установка найденного сегмента как основного
sys_set_default_segment([
     1_: fixed_: location
])

// Разворачивание установленного сегмента
sys_spin_segment([
     1_: fixed_: location,
     2_: assign_: segments
])
// Проверяем, относится ли вопрос к множеству «Поиск радиуса»
searchElStr3([
     1_: fixed_: q_radius,
     2_: assign_: const_: pos_: arc_: arcFromRequest,
     3_: fixed_: questionLink
], , finishOperation)

// Находим объект, который был передан в качестве аргумента
searchElStr3([
     1_: fixed_: questionLink,
     2_: assign_: const_: pos_: arc_: arcVar,
     3_: assign_: const_: node_: object
], ,finishOperation)

// находим значение радиуса для указанного объекта
callReturn([
     1_: fixed_: search_bin_pair_end_proc,
     2_: fixed_: {[
          1_: object,
          2_: nrel_radius,
          3_: radius
     ]}
], , , finish_operation)
```

```
// проверяем, найден ли узел, содержащий значение радиуса
// если узел найден, генерируем ответ, иначе завершаем операцию
ifVarAssign([
     1_: radius
],,finishOperation)


label(generateAnswer)
// Генерируем ответ
// Генерируем узел ответа
genElStr3([
     1_: assign_: node_: const_: answer,
     2_: assign_: pos_: const_: arcVar,
     3_: fixed_: radius
])

// Вызов процедуры построения ответа
callReturn([
     1_:  fixed_: answer_make,
     2_:  fixed_: {[
               1_: questionLink,
               2_: answer
          ]}
])

label(finishOperation)

return()
end
```

Эта операция значительно сложнее приведенной ранее. Рассмотрим некоторые ее фрагменты более детально.

После того как был определен нужный сегмент памяти и найден объект, переданный в качестве аргумента, мы ищем узел, обозначающий модуль числа. Как известно, он связан с самим числом отношением «радиус*». В данной ситуации нельзя воспользоваться поиском пятиэлементной конструкции, т. к. бинарное отношение в памяти представляется иначе. Существует готовая процедура поиска второго элемента бинарной пары search_bin_pair_end_proc. Следует вызывать ее, чтобы найти узел модуля.

```
// находим значение модуля для указанного объекта
callReturn([
     1_: fixed_: search_bin_pair_end_proc,
     2_: fixed_: {[
          1_: object,
          2_: nrel_radius,
          3_: radius
     ]}
], , , finish_operation)
```

В качестве первого параметра передается начало дуги отношения, второго – узел отношения, третьего – переменная, содержащая искомый конец отношения.

После завершения работы процедуры проверить, нашла ли она второй элемент:

```
ifVarAssign([
     1_: radius
],,finishOperation)
```

Если радиус не найден, завершить работу программы. В противном случае создать узел, который будет обозначать множество объектов, являющихся результатом работы операции (которые будут показаны пользователю). Добавить в множество, обозначаемое этим узлом, найденный радиус.

```
genElStr3([
     1_: assign_: node_: const_: answer,
     2_: assign_: pos_: const_: arcVar,
     3_: fixed_: radius
])
```

После этого вызывать процедуру answer_make, в которую передать узел вопроса и множество результирующих объектов.

```
callReturn([
     1_:  fixed_: answer_make,
     2_:  fixed_: {[
               1_: questionLink,
               2_: answer
          ]}
])
```

Эта процедура строит ответ и показывает его пользователю.

Для того чтобы операция запустилась, зарегистрировать ее в файле py_ui_geom\repo\fs_repo_src\startup.scs:

"/operation/my/find_abs/init_op".

Собрать базу знаний, запустить графический интерфейс и проверить результат работы программы. Создать узел с идентификатором «Окружность (O,,R)». Загрузить его в память при помощи комбинации клавиш Ctrl+Enter. Нажать клавишу Alt и, удерживая ее, нажать по этому узлу. Так узел выбирается в качестве аргумента для операции. Вызвать операцию «Поиск радиуса» при помощи созданного ранее меню. На экране будет изображен найденный узел, обозначающий радиус.

4. ОПЕРАЦИЯ ПОИСКА ЧИСЛОВОЙ ХАРАКТЕРИСТИКИ ОБЪЕКТА

Рассмотрим операцию поиска числовой характеристики объекта. В качестве примера предметом поиска будет модуль комплексного числа.

Добавить к базе знаний gwf со следующим содержимым (рис. 4.1).

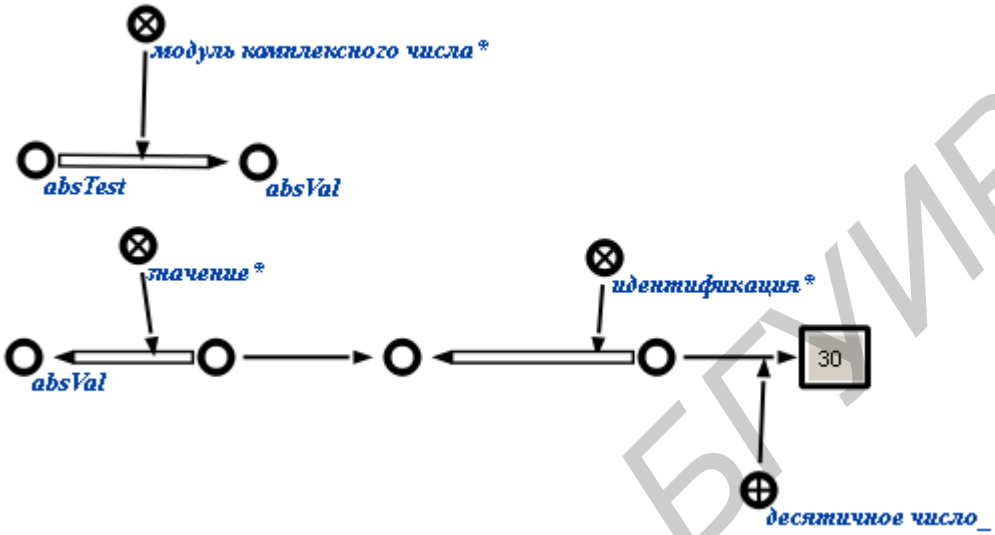

Рис. 4.1. Конструкция базы знаний для проверки операции

Данный фрагмент базы знаний показывает, что у узла absTest существует модуль, значение которого в десятичной системе равно 30. Файл с данным исходным текстом находится в папке **py_ui_geom\repo\fs_repo_src\seb\planimetry_src\.**

По описанному выше алгоритму создать атомарную команду меню. Ее описание будет выглядеть следующим образом (рис. 4.2).

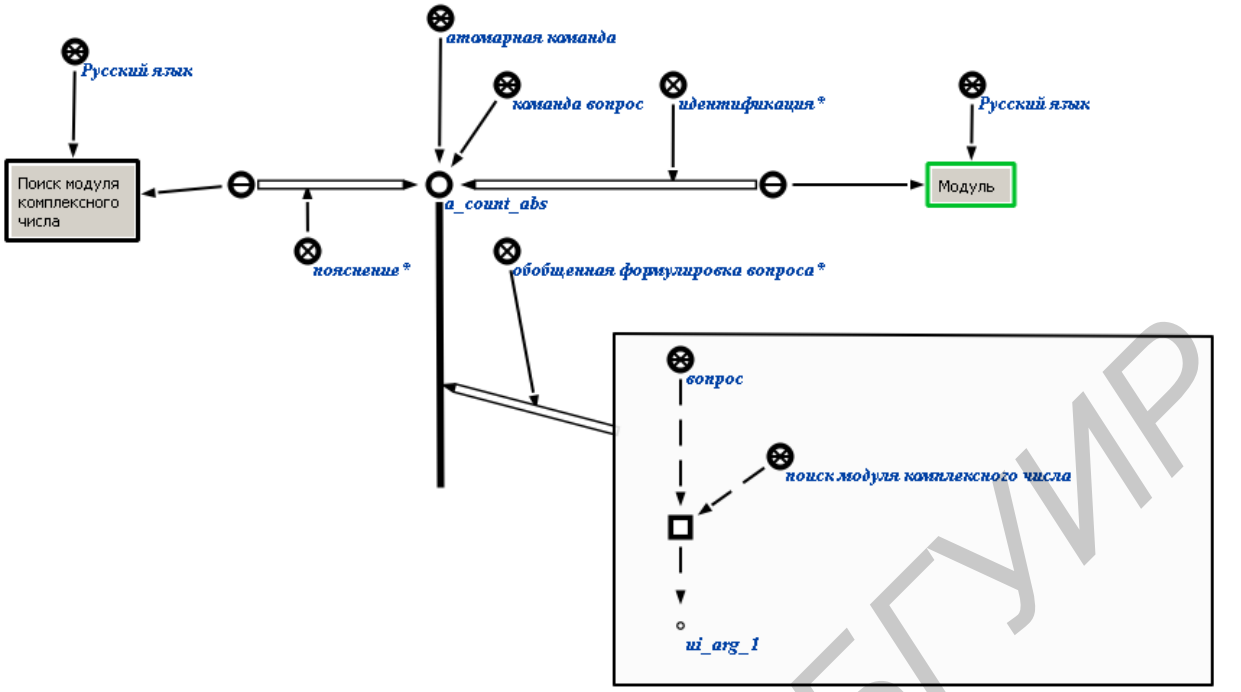

Рис. 4.2. Команда меню для запуска операции

Описание пункта меню почти не отличается от приведенного в предыдущем разделе. Добавить узел a_count_abs в файл na_my.gwf (рис. 4.3).

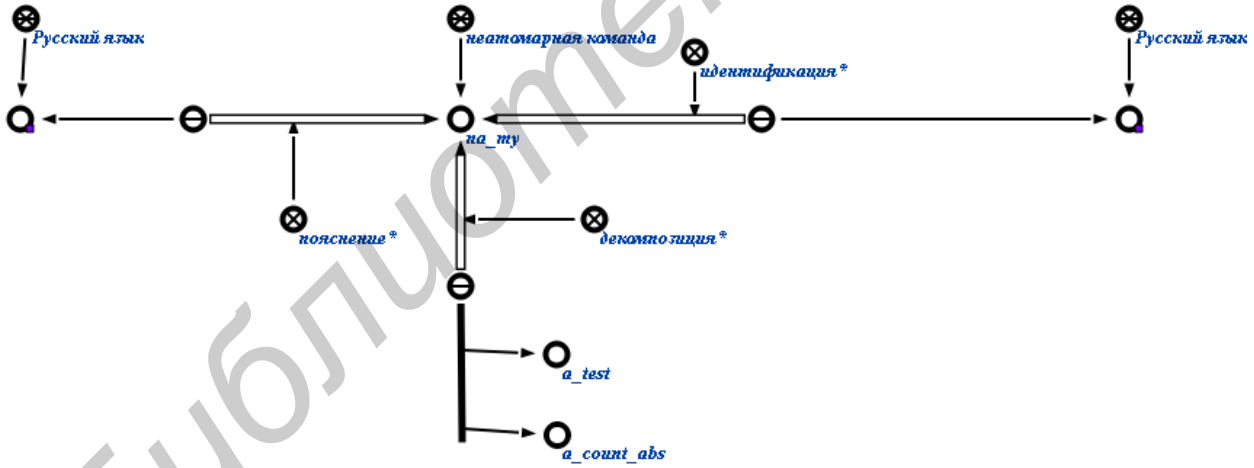

Рис. 4.3. Подключение написанного пункта меню к меню «Мои операции»

Для поиска модуля комплексного числа будет необходимо работать с отношением «модуль комплексного числа*». Необходимо задать для него краткое имя. Для этого следует скопировать узел отношения в файл **py_ui_geom\repo\fs_repo_src\etc\com_keynodes_src\com_keynodes.gwf**.

После этого в файл **py_ui_geom\repo\fs_repo_src\include\com_keynodes.scsy** нужно добавить следующую строчку:

nrel_abs = "/etc/com_keynodes/модуль комплексного числа*";

Теперь возможна работа с узлом отношения через имя nrel_abs.

Также надо задать короткое имя для узла «поиск модуля комплексного числа». Добавить узел в файл **py_ui_geom\repo\fs_repo_src\etc\questions_src\questions_keynodes.gwf**, а также внести в файл **py_ui_geom\repo\fs_repo_src\include\etc_questions.scsy** следующую строку:

q_abs = "/etc/questions/поиск модуля комплексного числа";

В папке, в которой находятся созданные исходные тексты операций, создать файл find_abs.m4scp со следующим текстом:

```
#include "scp_keynodes.scsy"
#include "etc_questions.scsy"
#include "com_keynodes.scsy"
#include "lib_search.scsy"
#include "lib_check.scsy"
#include "lib_gen.scsy"
#include "lib_answer.scsy"
#include "lib_set.scsy"

// программа, представляющая операцию поиска модуля комплексного
числа
program(init_op,
[[
     // Подпрограмма, вызываемая для выполнения операции
     find_abs;
     // Ключевой узел, обозначающий инициированный вопрос
     q_initiated;
     //  Событие,  на  которое  реагирует  обработчик(проведение
выходящей дуги из узла)
     catch_output_arc;
]],
[{
}],
{[
]}
)

// Установка обработчика события на проведение дуги из узла
«Инициированный запрос»
sys_set_event_handler([
     1_: fixed_: catch_output_arc,
     2_: fixed_: find_abs,
     3_: fixed_: {1_: q_initiated}
])
return()

end

// Процедура, вызывающаяся при срабатывании обработчика события
procedure(find_abs,
[[
     // Ключевой узел, обозначающий запрос
```

```
     q_abs;

     // Ключевой узел, обозначающий инициированный вопрос
     q_initiated;

     // Ключевой узел, обозначающий отношение «модуль*»
     nrel_abs;

     // Ключевой узел, обозначающий запрос значения величины
     q_var_value;

     // Ключевой узел, обозначающий вопрос
     question;
     // Атрибут присутствия ответа
     rrel_answer_is_present;

]],
[{
     // переменные для представления входных параметров
     handler, element,
     arcFromQuestion,
     questionLink,
     //  дуга  между  узлом,  обозначающим  вид  вопроса,  и  узлом
вопроса
     arcFromRequest,
     // дуга между узлом вопроса и аргументом
     arcVar,
     // аргумент поисковой операции
     object,
     //  переменные  для  установки  сегмента  памяти,  в  котором
происходит работа
     location, segments,

     absValue,
     questionNode, attributeArc,
     result, answer
}],
{[
     1_: in_: handler,
     2_: in_: element,
     3_: in_: arcFromQuestion,
     4_: in_: questionLink
]}
)

// Получение сегмента, в котором находится узел вопроса
sys_get_location([
     1_: fixed_: nrel_abs,
     2_: assign_: location
])
// Установка найденного сегмента как основного
sys_set_default_segment([
```

```
     1_: fixed_: location
])

// Разворачивание установленного сегмента
sys_spin_segment([
     1_: fixed_: location,
     2_: assign_: segments
])
// Проверяем, относится ли вопрос к множеству «Информация об узле»
searchElStr3([
     1_: fixed_: q_abs,
     2_: assign_: const_: pos_: arc_: arcFromRequest,
     3_: fixed_: questionLink
], , finishOperation)

// Находим объект, который был передан в качестве аргумента
searchElStr3([
     1_: fixed_: questionLink,
     2_: assign_: const_: pos_: arc_: arcVar,
     3_: assign_: const_: node_: object
], ,finishOperation)

// находим значение модуля для указанного объекта
callReturn([
     1_: fixed_: search_bin_pair_end_proc,
     2_: fixed_: {[
          1_: object,
          2_: nrel_abs,
          3_: absValue
     ]}
], , , finish_operation)

// проверяем, найден ли узел, содержащий значение модуля
//  если найден, переходим сразу к вычислению значения, иначе
генерируем узел
ifVarAssign([
     1_: absValue
],have_value)



// генерируем узел значения модуля
genEl([
     1_: assign_: const_: node_: absValue
])

// генерируем отношение между объектом и узлом значения модуля
callReturn([
     1_: fixed_: gen_bin_pair,
     2_: fixed_: {[
          1_: object,
          2_: absValue,
```

```
          3_: nrel_abs
     ]}
], , , finish_operation)

label(have_value)
genEl([
          1_: assign_: const_: node_: questionNode
])

// Добавляем в вопрос найденную величину периметра
genElStr3([
     1_: fixed_: questionNode,
     2_: arc_: const_: pos_: actual_: arcVar,
     3_: fixed_: absValue
])

// Инициируем выполнение запроса значения величины
// Проводим дугу из запроса значения величины
genElStr3([
     1_: fixed_: q_var_value,
     2_: assign_: arc_: const_: pos_: actual_: arcVar,
     3_: fixed_: questionNode
])

// Проводим дугу из узла «Вопрос»
genElStr3([
     1_: fixed_: question,
     2_: assign_: arc_: const_: pos_: actual_: arcVar,
     3_: fixed_: questionNode
])

// Проводим дугу из узла «Инициированный вопрос»
genElStr3([
     1_: fixed_: q_initiated,
     2_: assign_: arc_: const_: pos_: actual_: arcVar,
     3_: fixed_: questionNode
])

// Ждем, пока процесс работы решателя завершится
label(waiting)
sys_wait([
     1_: fixed_: catch_output_arc,
     2_: fixed_: {1_: rrel_answer_is_present},
])

// Проверяем, что дуга проведена именно к этому вопросу
searchElStr5([
     1_: fixed_: q_var_value,
     2_: assign_: arc_: const_: pos_: actual_: arcVar,
```

```
     3_: fixed_: questionNode,
     4_: assign_: arc_: const_: pos_: actual_: attributeArc,
     5_: fixed_: rrel_answer_is_present
], , waiting)


// Находим значение модуля
callReturn([
     1_: fixed_: search_quantity_value,
     2_: fixed_: {[
          1_: absValue,
          2_: result
     ]}
])

// Генерируем ответ
// Генерируем узел ответа
genElStr3([
     1_: assign_: node_: const_: answer,
     2_: assign_: pos_: const_: arcVar,
     3_: fixed_: result
])

// Вызов процедуры построения ответа
callReturn([
     1_:  fixed_: answer_make,
     2_:  fixed_: {[
               1_: questionLink,
               2_: answer
          ]}
])
label(generateAnswer)

label(finishOperation)

return()
end
```

После того как определен нужный сегмент памяти и найден объект, переданный в качестве аргумента, необходимо найти узел, обозначающий модуль числа. Как известно, он связан с самим числом отношением «модуль комплексного числа*». В данной ситуации нельзя воспользоваться поиском пятиэлементной конструкции, т. к. бинарное отношение в памяти представляется иначе. Можно воспользоваться готовой процедурой поиска второго элемента бинарной пары search_bin_pair_end_proc (вызывается для того, чтобы найти узел модуля).

```
// находим значение модуля для указанного объекта
callReturn([
     1_: fixed_: search_bin_pair_end_proc,
     2_: fixed_: {[
          1_: object,
```

```
          2_: nrel_abs,
          3_: absValue
     ]}
], , , finish_operation)
```

В качестве первого параметра передается начало дуги отношения, второго – узел отношения, третьего – переменная, содержащая искомый конец отношения.

После завершения работы процедуры проверить, нашла ли она второй элемент:

```
ifVarAssign([
     1_: absValue
],have_value)
```

Если нет, то искусственно создать узел, означающий модуль, и связать его с аргументом отношением «модуль комплексного числа*» при помощи процедуры gen_bin_pair.

```
genEl([
     1_: assign_: const_: node_: absValue
])

// генерируем отношение между объектом и узлом значения модуля
callReturn([
     1_: fixed_: gen_bin_pair,
     2_: fixed_: {[
          1_: object,
          2_: absValue,
          3_: nrel_abs
     ]}
], , , finish_operation)
```

Далее начинается процесс поиска численного значения величины периметра. Фрагмент базы знаний, описывающий численное значение, имеет достаточно сложную структуру. Однако существуют готовые поисковые операции. Операция search_quantity_value выполняет поиск значения величины. Однако для ее запуска необходимо сгенерировать запрос (структуру, которая описывается в описании пункта меню для операции). Это и осуществляется следующим фрагментом.

```
genEl([
          1_: assign_: const_: node_: questionNode
])

// Добавляем в вопрос найденную величину периметра
genElStr3([
     1_: fixed_: questionNode,
     2_: arc_: const_: pos_: actual_: arcVar,
     3_: fixed_: absValue
])

// Инициируем выполнение запроса значения величины
// Проводим дугу из запроса значения величины
genElStr3([
```

```
     1_: fixed_: q_var_value,
     2_: assign_: arc_: const_: pos_: actual_: arcVar,
     3_: fixed_: questionNode
])

// Проводим дугу из узла «Вопрос»
genElStr3([
     1_: fixed_: question,
     2_: assign_: arc_: const_: pos_: actual_: arcVar,
     3_: fixed_: questionNode
])

// Проводим дугу из узла «Инициированный вопрос»
genElStr3([
     1_: fixed_: q_initiated,
     2_: assign_: arc_: const_: pos_: actual_: arcVar,
     3_: fixed_: questionNode
])
```

После этого нужно подождать, пока не отработает операция поиска:

```
sys_wait([
     1_: fixed_: catch_output_arc,
     2_: fixed_: {1_: rrel_answer_is_present},
])
```

Запустить процедуру поиска значения:

```
callReturn([
     1_: fixed_: search_quantity_value,
     2_: fixed_: {[
          1_: absValue,
          2_: result
     ]}
])
```

В переменной result будет находиться искомый узел. Для того чтобы вывести его на экран, используется процедура answer_make. На вход она принимает узел вопроса и множество, представляющее ответ (answer). Добавить найденное значение в множество answer, после чего вызвать процедуру генерации ответа.

```
genElStr3([
     1_: assign_: node_: const_: answer,
     2_: assign_: pos_: const_: arcVar,
     3_: fixed_: result
])

// Вызов процедуры построения ответа
callReturn([
     1_:  fixed_: answer_make,
     2_:  fixed_: {[
               1_: questionLink,
               2_: answer
          ]}
])
```

Для того чтобы операция запустилась, зарегистрировать ее в файле py_ui_geom\repo\fs_repo_src\startup.scs:

"/operation/my/find_abs/init_op"

Собрать базу знаний, запустить графический интерфейс. Создать узел, назвать его absTest, загрузить в память. Вызвать для него написанную операцию (рис. 4.4). Должен появиться узел с содержимым (рис. 4.5). Для просмотра содержимого этот узел необходимо выделить и нажать клавишу H.

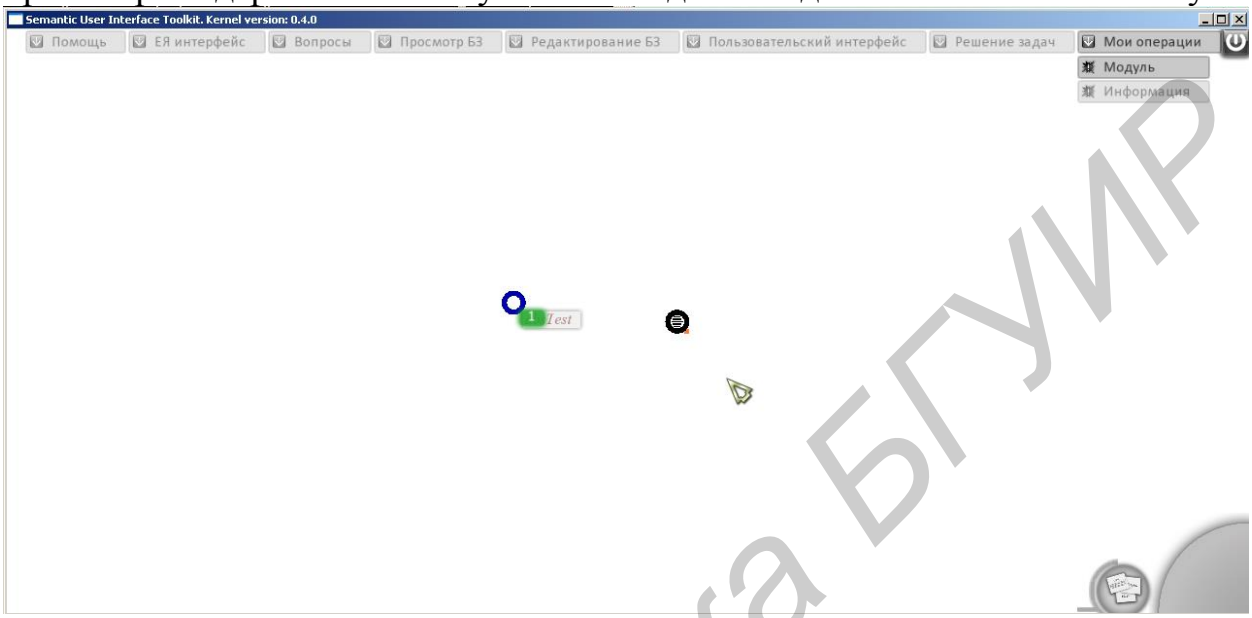

Рис. 4.4. Результат работы операции (найденный узел)

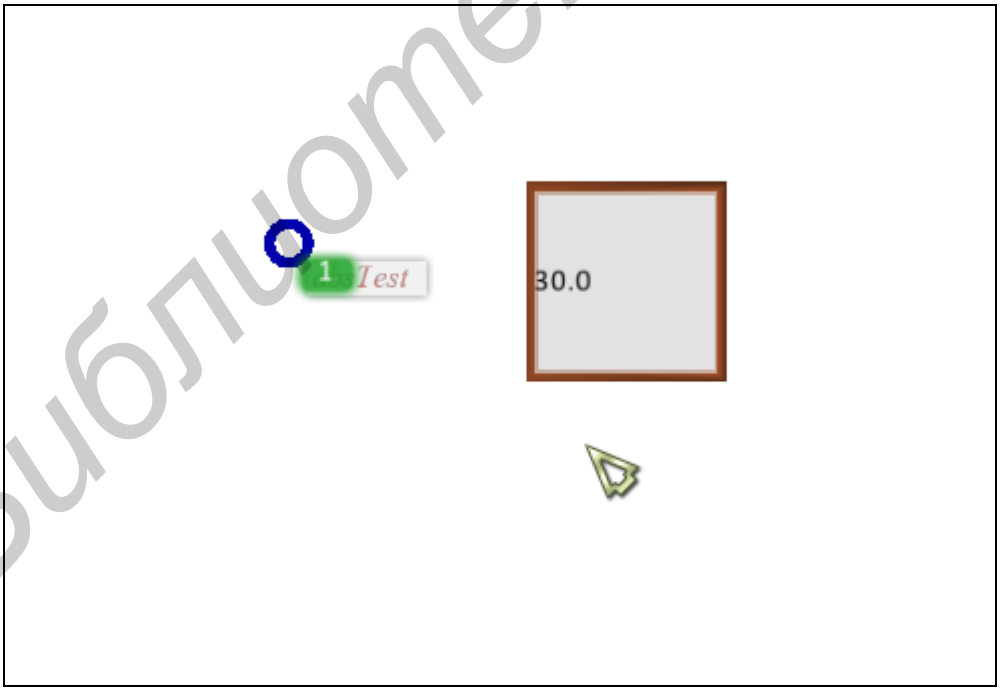

Рис. 4.5. Содержимое найденного узла

5. ОПЕРАЦИЯ ПОИСКА ХАРАКТЕРИСТИКИ ОБЪЕКТА, ЗАДАННОЙ МНОЖЕСТВОМ

Рассмотрим операцию поиска характеристики, заданной множеством. В качестве примера создать в папке py_ui_geom\repo\fs_repo_src\seb\planimetry_src\ gwf-файл со следующим фрагментом базы знаний (рис. 5.1).

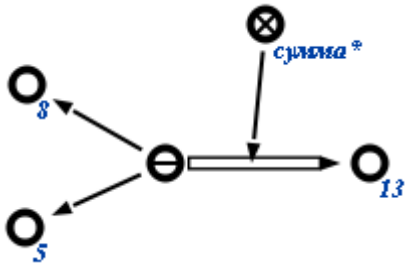

Рис. 5.1. Фрагмент базы знаний для демонстрации работы операции

Создать поисковую операцию, которая по узлу, содержащему сумму, ищет множество слагаемых.

Для начала повторить процедуру создания атомарного пункта меню и добавления его в существующий неатомарный (рис. 5.2).

py_ui_geom\repo\fs_repo_src\ui\menu\na_main_menu\na_my\na_find_summands.gwf

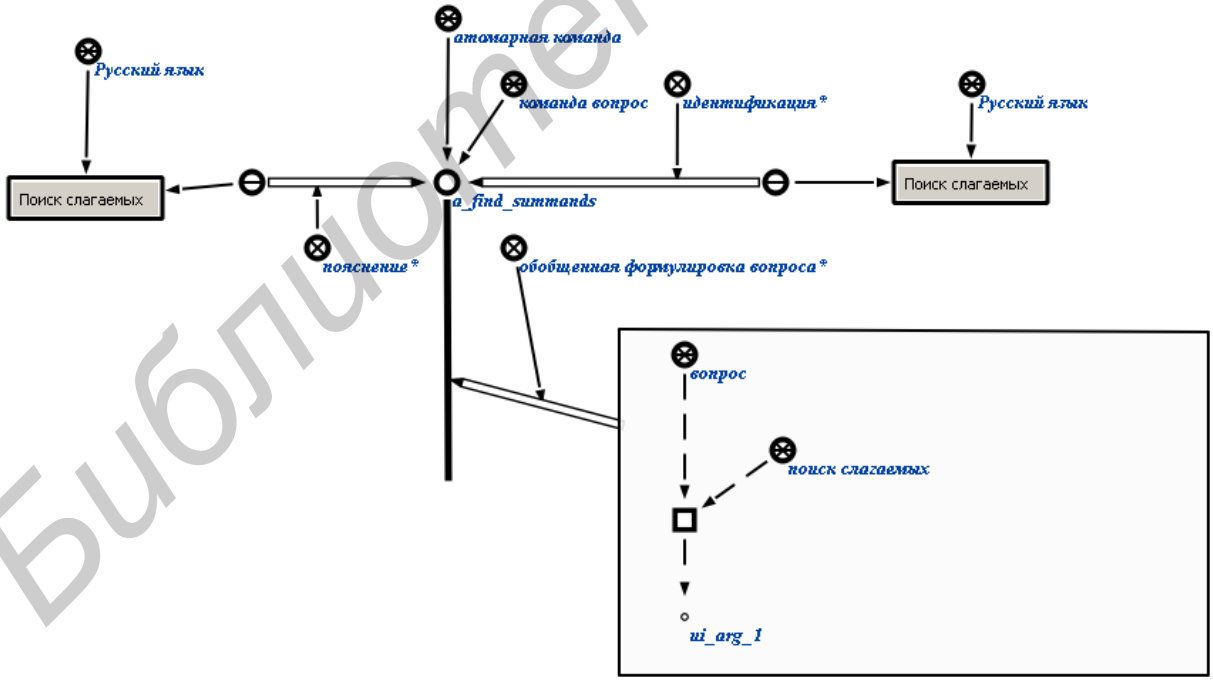

Рис. 5.2. Неатомарная команда меню для запуска операции

py_ui_geom\repo\fs_repo_src\ui\menu\na_main_menu\na_my.gwf

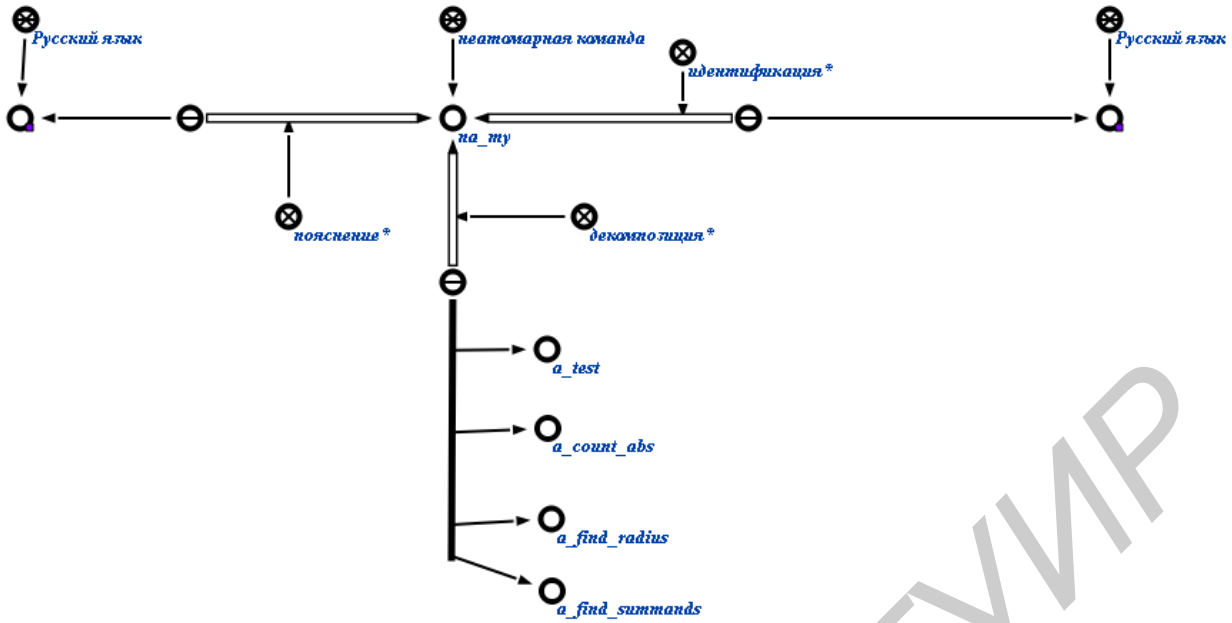

Рис. 5.3. Подключение написанного пункта меню к меню «Мои операции»

Для работы операции необходим узел типа вопроса: «поиск слагаемых» и узел отношения «сумма*».

Добавить узел «поиск слагаемых» в файл **py_ui_geom\repo\fs_repo_src\etc\questions_src\questions_keynodes.gwf**, а также внести в файл **py_ui_geom\repo\fs_repo_src\include\etc_questions.scsy** следующую строку:

q_summands = "/etc/questions/поиск слагаемых";

Скопировать узел отношения «сумма*» в файл **py_ui_geom\repo\fs_repo_src\etc\com_keynodes_src\com_keynodes.gwf**.

После этого в файл **py_ui_geom\repo\fs_repo_src\include\com_keynodes.scsy** нужно добавить следующую строку:

nrel_sum = "/etc/com_keynodes/сумма*";

Теперь возможна работа с узлом отношения через имя nrel_sum.

В папке, в которой находятся созданные исходные тексты операций, создать файл find_summands.m4scp со следующим текстом:

```
#include "scp_keynodes.scsy"
#include "etc_questions.scsy"
#include "com_keynodes.scsy"
#include "lib_search.scsy"
#include "lib_check.scsy"
#include "lib_gen.scsy"
#include "lib_answer.scsy"
#include "lib_set.scsy"

// программа, представляющая операцию поиска слагаемых
program(init_op,
[[
     // Подпрограмма, вызываемая для выполнения операции
```

```
     find_summands;
     // Ключевой узел, обозначающий инициированный вопрос
     q_initiated;
     // Событие, на которое реагирует обработчик(проведение
выходящей дуги из узла)
     catch_output_arc;
]],
[{
}],
{[
]}
)

// Установка обработчика события на проведение дуги из узла
«Инициированный запрос»
sys_set_event_handler([
     1_: fixed_: catch_output_arc,
     2_: fixed_: find_summands,
     3_: fixed_: {1_: q_initiated}
])

return()

end

// Процедура, вызываемая при срабатывании обработчика события
procedure(find_summands,
[[

     // Ключевой узел, обозначающий запрос
     q_summands;

     // Ключевой узел, обозначающий инициированный вопрос
     q_initiated;

     // Ключевой узел, обозначающий отношение «сумма*»
     nrel_sum;

]],
[{
     // переменные для представления входных параметров
     handler, element,
     arcFromQuestion,
     questionLink,
     // дуга между узлом, обозначающим вид вопроса, и узлом
вопроса
     arcFromRequest,
     // дуга между узлом вопроса и аргументом
     arcVar,
     // аргумент поисковой операции
     object,
```

```
     // переменные для установки сегмента памяти, в котором
происходит работа
     location, segments,

     summands, loop, sumArc,
     tempNode, tempArc,
     answer
}],
{[
     1_: in_: handler,
     2_: in_: element,
     3_: in_: arcFromQuestion,
     4_: in_: questionLink
]}
)

// Получение сегмента, в котором находится отношение «сумма*»
sys_get_location([
     1_: fixed_: nrel_sum,
     2_: assign_: location
])

// Установка найденного сегмента как основного
sys_set_default_segment([
     1_: fixed_: location
])

// Разворачивание установленного сегмента
sys_spin_segment([
     1_: fixed_: location,
     2_: assign_: segments
])

// Проверяем, относится ли вопрос к множеству «поиск слагаемых»
searchElStr3([
     1_: fixed_: q_summands,
     2_: assign_: const_: pos_: arc_: arcFromRequest,
     3_: fixed_: questionLink
], , finishOperation)

// Находим объект, который был передан в качестве аргумента
searchElStr3([
     1_: fixed_: questionLink,
     2_: assign_: const_: pos_: arc_: arcVar,
     3_: assign_: const_: node_: object
], ,finishOperation)


// находим множество слагаемых
callReturn([
     1_: fixed_: search_bin_pair_begin_proc,
     2_: fixed_: {[
```

```
          1_: object,
          2_: nrel_sum,
          3_: summands
     ]}
], , , finish_operation)

// проверяем, найден ли узел, представляющий множество слагаемых
// если узел найден, готовим ответ, иначе завершаем операцию
ifVarAssign([
     1_: summands
],,finishOperation)

// генерируем узел ответа
genElStr3([
     1_: assign_: node_: const_: answer,
     2_: assign_: pos_: const_: arcVar,
     3_: fixed_: summands
])

// теперь необходимо пройти по всем элементам множества summands
// и добавить их в результирующее множество

// создаем копию найденного множества
searchSetStr3([
     1_: fixed_: summands,
     2_: assign_: const_: pos_: arcVar,
     3_: assign_: node_: tempNode,
     set3_: assign_: loop
])

label(next_iter)
// выбираем элемент множества, если таких нет, генерируем ответ
searchElStr3([
     1_: fixed_: loop,
     2_: assign_: const_: pos_: arc_: arcVar,
     3_: assign_: const_: node_: tempNode
],,generateAnswer)

// добавляем найденный узел в результирующее множество
genElStr3([
     1_: fixed_: answer,
     2_: assign_: const_: pos_: arc_: tempArc,
     3_: fixed_: tempNode
])

// ищем дугу, связывающую найденный узел с множеством слагаемых
searchElStr3([
     1_: fixed_: summands,
     2_: assign_: const_: pos_: sumArc,
     3_: fixed_: tempNode
])
// добавляем дугу в результирующее множество
```

```
genElStr3([
     1_: fixed_: answer,
     2_: assign_: const_: pos_: arc_: tempArc,
     3_: fixed_: sumArc
])

// переходим на следующую итерацию
eraseEl([
     1_: fixed_: arcVar
],next_iter,next_iter)


label(generateAnswer)
// удаляем вспомогательные структуры
eraseEl([
     1_: fixed_: loop
])
// Вызов процедуры построения ответа
callReturn([
     1_:  fixed_: answer_make,
     2_:  fixed_: {[
               1_: questionLink,
               2_: answer
          ]}
])

label(finishOperation)

return()
end
```

В предыдущем примере рассматривался поиск узла, находящегося на конце дуги бинарного отношения, а теперь искомый узел находится в начале. Поэтому используется другая процедура, которая ищет начало дуги:

```
// находим множество слагаемых
callReturn([
     1_: fixed_: search_bin_pair_begin_proc,
     2_: fixed_: {[
          1_: object,
          2_: nrel_sum,
          3_: summands
     ]}
], , , finish_operation)
```

Следующей отличительной особенностью данной операции является то, что в качестве ответа пользователю выводятся несколько объектов, среди которых есть и узлы, и дуги. Это достигается простым добавлением этих объектов в множество answer, которое затем передается процедуре построения ответа.

Еще одним нюансом данной операции является организация циклов. Язык scp не поддерживает циклы, поэтому для итерации по элементам множества приходится использовать нетривиальный подход.

Сначала скопировать множество слагаемых в множество loop. Это необходимо для того, чтобы при итерации не изменить исходное состояние базы знаний.

```
searchSetStr3([
     1_: fixed_: summands,
     2_: assign_: const_: pos_: arcVar,
     3_: assign_: node_: tempNode,
     set3_: assign_: loop
])
```

После этого выполнить нижеописываемые действия.

Найти любой элемент множества loop. В случае, если таких элементов нет, перейти на метку, находящуюся за циклом. В начале блока операторов, которые «крутятся» в цикле, также ставится метка.

```
searchElStr3([
     1_: fixed_: loop,
     2_: assign_: const_: pos_: arc_: arcVar,
     3_: assign_: const_: node_: tempNode
],,generateAnswer)
```

Элемент итерируемого множества, выбранный на данной итерации, находится в tempNode. Выполнить требуемые действия над этим элементом (в данном примере – добавление элемента и дуги принадлежности множеству слагаемых в множество ответов).

```
genElStr3([
     1_: fixed_: answer,
     2_: assign_: const_: pos_: arc_: tempArc,
     3_: fixed_: tempNode
])

// ищем дугу, связывающую найденный узел с множеством слагаемых
searchElStr3([
     1_: fixed_: summands,
     2_: assign_: const_: pos_: sumArc,
     3_: fixed_: tempNode
])

// добавляем дугу в результирующее множество
genElStr3([
     1_: fixed_: answer,
     2_: assign_: const_: pos_: arc_: tempArc,
     3_: fixed_: sumArc
])
```

После этого удалить дугу принадлежности arcVar и перейти в начало блока цикла:

```
eraseEl([
     1_: fixed_: arcVar
],next_iter,next_iter)
```

В конце операции вызывать процедуру построения ответа.

Сохранить файл. Добавить в startup.scs следующую строку:

"/operation/my/find_summands/init_op".

Далее запустить сборку базы знаний. После завершения сборки запустить графический интерфейс, создать узел с идентификатором 13, загрузить его в память, выбрать в качестве аргумента и вызвать операцию при помощи команды меню (рис. 5.4).

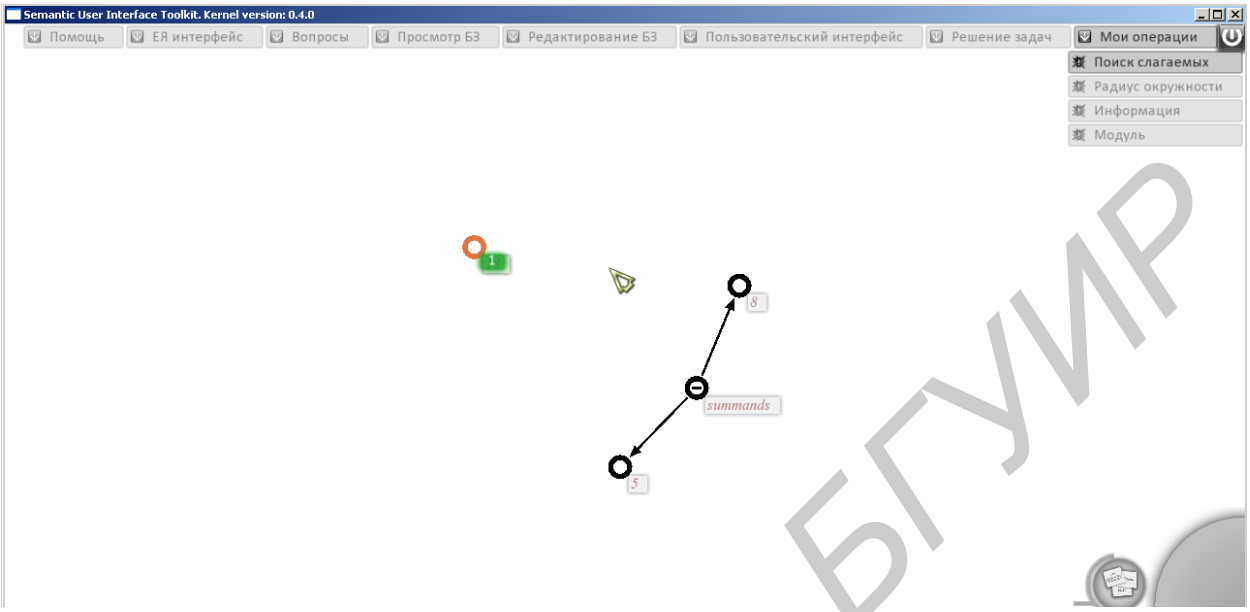

Рис. 5.4. Результат работы операции

6. СЕМАНТИЧЕСКИЙ УНИФИЦИРОВАННЫЙ ЯЗЫК ОПИСАНИЯ ВОПРОСОВ

Язык описания вопросов (или просто язык вопросов) предназначен для формулирования вопросов системе сторонними субъектами, каковыми могут быть как пользователи системы, так и сторонние интеллектуальные системы. Также и в рамках самой системы операции обработки знаний могут формулировать вопросы, предназначенные для других подобных операций.

Каждый вопрос в памяти системы описывается некоторой вопросной конструкцией, содержащей сам узел вопроса и дополнительную информацию, необходимую для поиска ответа на заданный вопрос – тип вопроса, аргументы и т. д.

Ниже приводятся описания ключевых узлов языка описания вопросов в рамках технологии, предлагаемой для решения задач, поставленных в рамках курсового проекта.

- Ключевой узел ***вопрос*** – знак множества вопросов произвольного вида, без уточнения конкретной семантики вопроса (рис. 6.1).

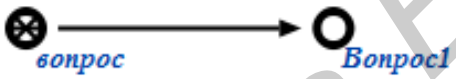

Рис. 6.1. Вопрос

- Семейство знаков множеств частных вопросов.

Примерами знаков множеств частных вопросов могут служить ***запрос значения величины***, ***запрос истинности*** и т. д.

Принадлежность какому-либо классу частных вопросов уточняет семантику данного конкретного вопроса. Классы вопросов можно разделить на предметно-зависимые (запрос яркости небесного тела) и предметно-независимые (запрос значения величины). Однако такое деление достаточно условно, потому как, например, запрос температуры объекта или запрос скорости объекта могут быть использованы далеко не только в системе по физике.

Любой класс частных вопросов является подмножеством множества вопросов (рис. 6.2).

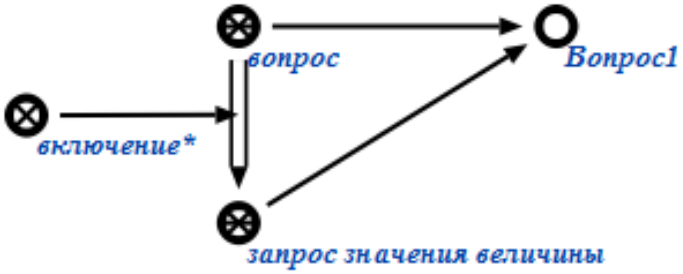

Рис. 6.2. Частный вопрос

- Аргументы вопроса.

Аргументами вопроса могут быть любые элементы базы знаний системы. В зависимости от конкретного класса вопроса их количество может варьироваться от нуля до десятка (рис. 6.3). Аргумент вопроса с теоретико-множественной точки зрения является его элементом, т. е. любой вопрос представляет собой множество аргументов (иногда – пустое множество).

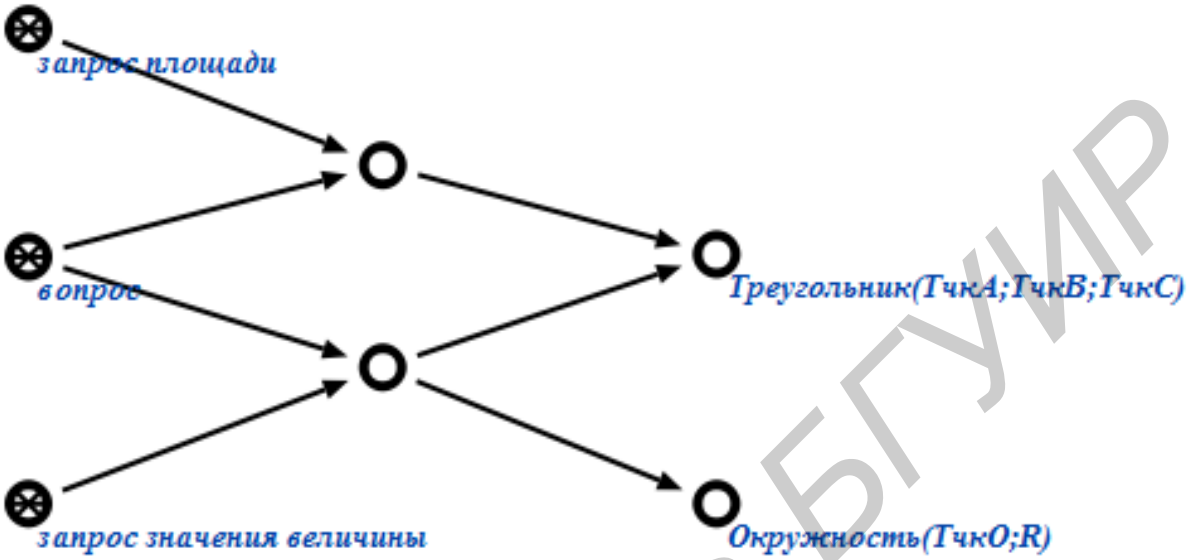

Рис. 6.3. Аргументы вопроса

При необходимости роль каждого аргумента может уточняться при помощи соответствующих ролевых отношений (рис. 6.4).

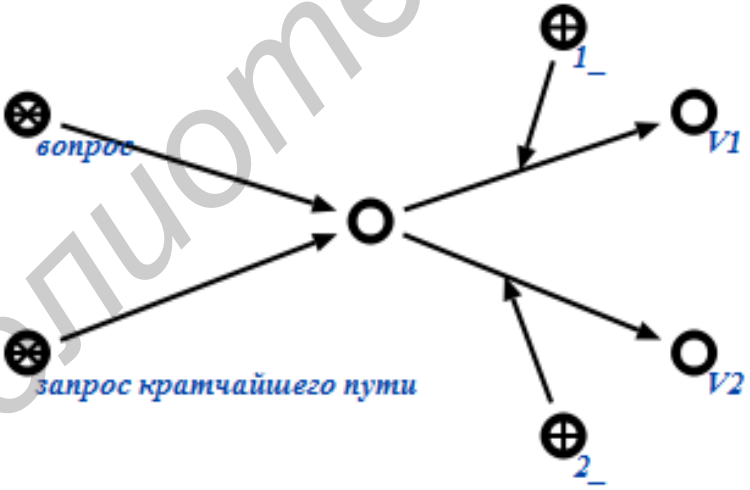

Рис. 6.4. Роль аргументов вопроса

- Ключевой узел ***автор****.

Является знаком отношения, связывающего вопрос автора данного вопроса (т. е. субъекта, сгенерировавшего в памяти соответствующую вопросную конструкцию).

Указание автора является необходимым в ряде случаев, например, когда вопрос задан пользователем через интерфейс, и ответ также должен быть выдан в нужное окно пользовательского интерфейса (рис. 6.5).

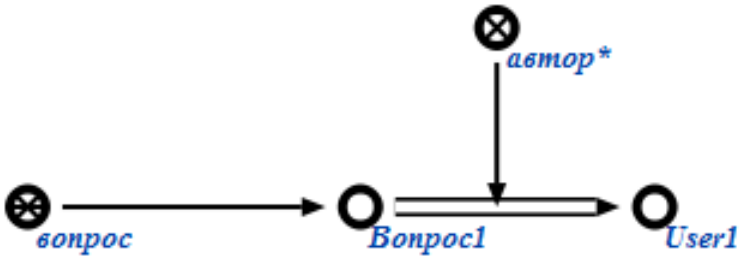

Рис. 6.5. Автор вопроса

- Ключевой узел ***ответ****.

Является знаком отношения, связывающего конкретный вопрос и некоторую конструкцию, являющуюся ответом на данный вопрос.

Вид этой конструкции определяется конкретным классом вопроса. Наличие ответа на вопрос (конечно, при условии возможности его получения) является необходимым независимо от автора, класса вопроса и прочих параметров (рис. 6.6).

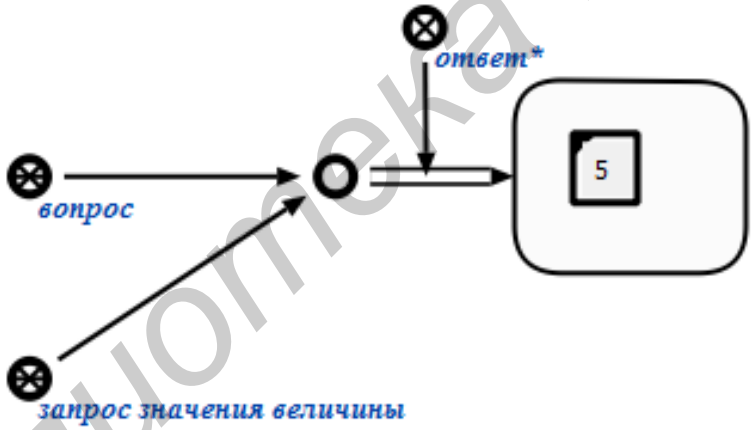

Рис. 6.6. Ответ на вопрос

- Ключевой узел ***решение****.

Является знаком отношения, связывающего конкретный вопрос с решением. Решение представляет собой набор связок, каждая из которых содержит информацию о том, какое утверждение было использовано на данном шаге решения, для какого объекта и какая конструкция получена в итоге.

На указанных связках задано отношение строгого порядка, обеспечивающее возможность определения последовательности шагов решения конкретной задачи.

В некоторых случаях решение может отсутствовать даже в случае успешного ответа на вопрос, к примеру, в том случае, если решение задачи ограничено простым поиском (рис. 6.7).

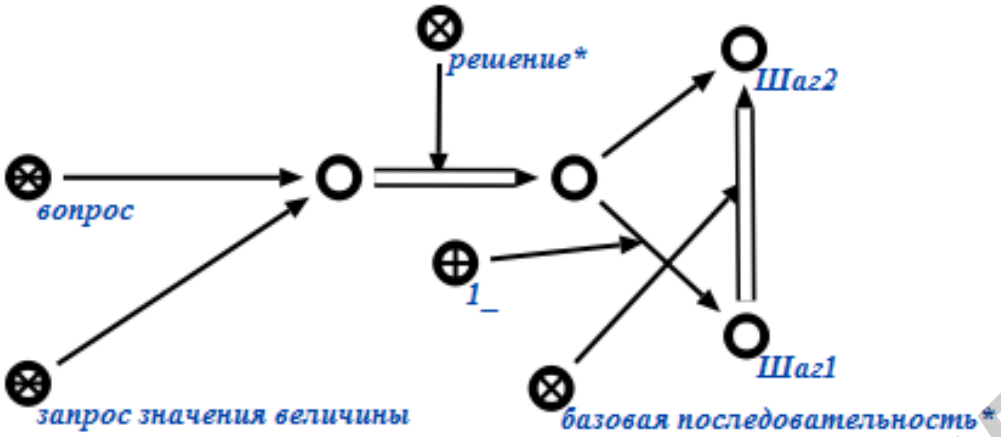

Рис. 6.7. Решение задачи

- Ключевой узел ***инициированный вопрос***.

Является знаком множества инициированных вопросов (рис. 6.8). Тот факт, что вопрос инициирован, свидетельствует о том, что вопросная конструкция полностью сформирована и машина обработки знаний может приступать к поиску ответа на поставленный вопрос. Таким образом, генерация дуги принадлежности вопроса множеству инициированных вопросов должна являться завершающим шагом построения любой вопросной конструкции.

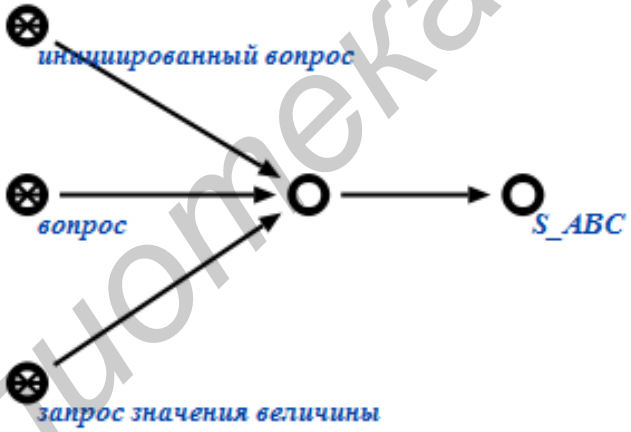

Рис. 6.8. Инициированный вопрос

- Ролевое отношение ***присутствует ответ'***.

Используется для указания того факта, что системе удалось найти ответ на поставленный вопрос. Роль указывается для вопроса в рамках некоторого класса частных вопросов (рис. 6.9).

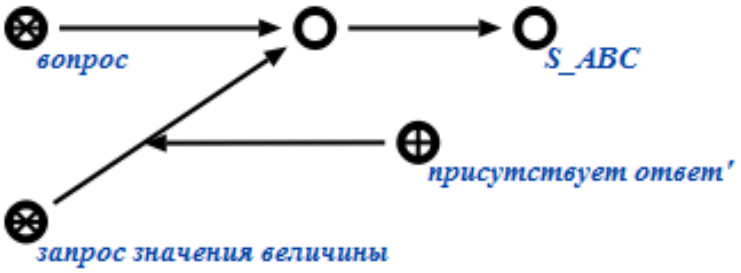

Рис. 6.9. Присутствие ответа на вопрос

Предполагается, что факт присутствия ответа указывается после того, как ответная конструкция полностью сформирована и сгенерирована связка отношения ***ответ****.

- Ролевое отношение ***отсутствует ответ'***.

Используется для указания того факта, что ответ на поставленный вопрос в настоящий момент найден быть не может (рис. 6.10). Роль указывается для вопроса в рамках некоторого класса частных вопросов.

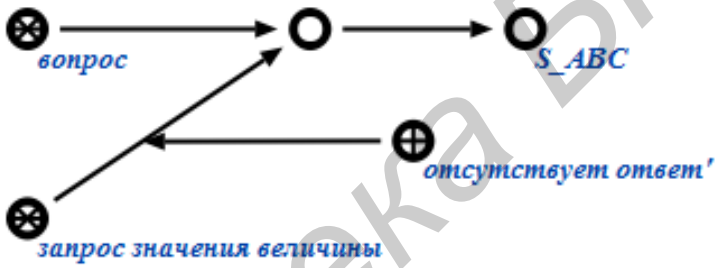

Рис. 6.10. Отсутствие ответа на вопрос

Для дополнительной синхронизации между агентами машины обработки знаний факт отсутствия ответа может уточняться более частными ролевыми отношениями, например, ***ответ отсутствует в явном виде'***.

7. ПРИМЕР ОПИСАНИЯ АЛГОРИТМА ВЫПОЛНЕНИЯ ОПЕРАЦИИ

В качестве примера, описывающего особенности выполнения алгоритма, была выбрана операция получения значения продукции. Данная операция является одной из ключевых в системе операций прямого логического вывода.

Описание алгоритма работы операции:

1. Ищем узел связки конкретной теории.

1.1. Если узел связки конкретной теории не найден, то переходим к шагу 2.

1.2. Если узел связки конкретной теории найден, то получаем объект из запроса продукции.

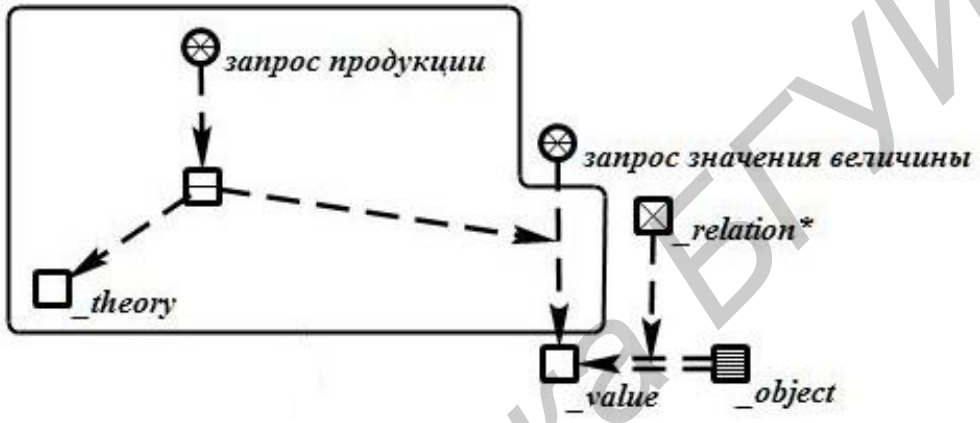

Рис. 7.1. Пояснение шага 1.2

1.2.1. Если объект из запроса продукции не найден, то переходим к шагу 2.

1.2.2. Если объект из запроса продукции найден, то получаем условие (в конкретной теории находится под атрибутом 1_) и следствие (в конкретной теории находится под атрибутом 2_) из конкретной теории.

1.2.2.1. Если условие и следствие не были получены, то переходим к шагу 2.

1.2.2.2. Если условие и следствие получены, то получаем связанные узлы конкретной теории.

1.2.2.2.1. Если связанные узлы конкретной теории не получены, то переходим к шагу 2.

1.2.2.2.2. Если связанные узлы конкретной теории не получены, то происходит поиск фрагмента базы знаний по шаблону (условие теории) с сохранением множества пар результатов поиска.

1.2.2.2.3. Среди множества пар соответствия из шага 1.2.2.2.2 определяем ту, где под атрибутом 2_ имеем объект из шага 1.2.

1.2.2.2.4. Из полученной пары из шага 1.2.2.2.3. получаем узел под атрибутом 1_.

1.2.2.2.5. Ищем конкретный фрагмент базы знаний для объекта из шага 1.2 с сохранением множества пар результатов поиска.

1.2.2.2.6. Формируем множество связок, где под атрибутом 1_ находится связанная переменная из условия теории, которая была найдена в шаге 1.2.2, а под атрибутом 2_ находится соответствующий узел из базы знаний.

1.2.2.2.7. Для найденных пар из шага 1.2.2.2.5 ищем, где под атрибутом 1_ находятся связанные переменные из шага 1.2.2 и добавляем во множество связок из шага 1.2.2.2.6 связку, где под атрибутом 1_ находится узел из вышенайденной пары под атрибутом 1_, под атрибутом 2_ находится узел из найденной пары под атрибутом 2_.

1.2.2.2.8. Генерируем по следствию из теории константный контур, где заменяем связанные переменные из шага 1.2.2 на конкретные с помощью сформированного множества пар из шагов 1.2.2.2.6 и 1.2.2.2.7.

1.2.2.2.9. Если множество связок к шагу 1.2.2.2.7 не сформировано, то переходим к шагу 2.

1.2.2.2.10. Генерируем условие успешного завершения операции (рис. 7.2).

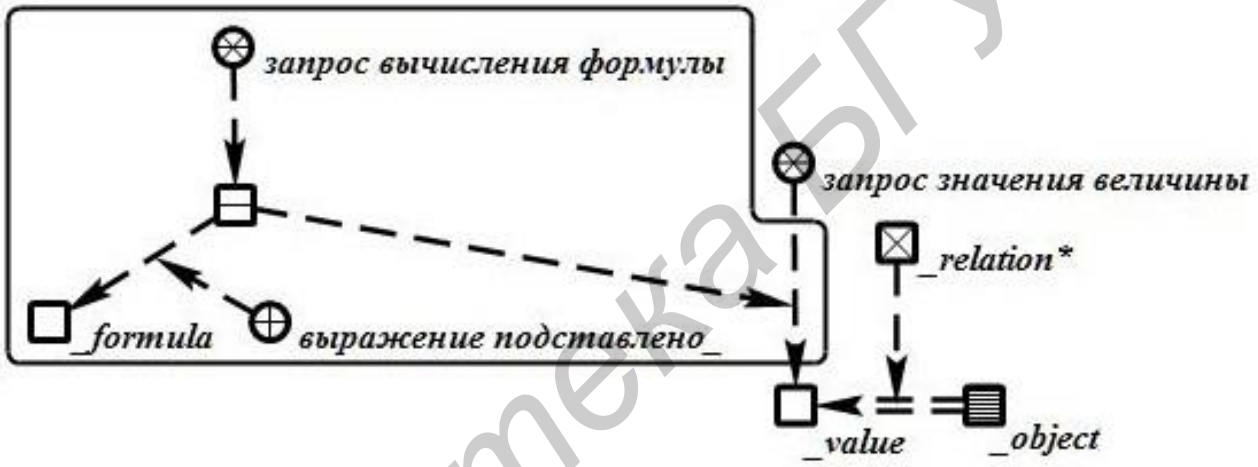

Рис. 7.2. Результат успешного завершения работы операции

1.2.2.2.11. Переходим к шагу 3.

2. Генерируем условие неудачного завершения операции (рис. 7.3).

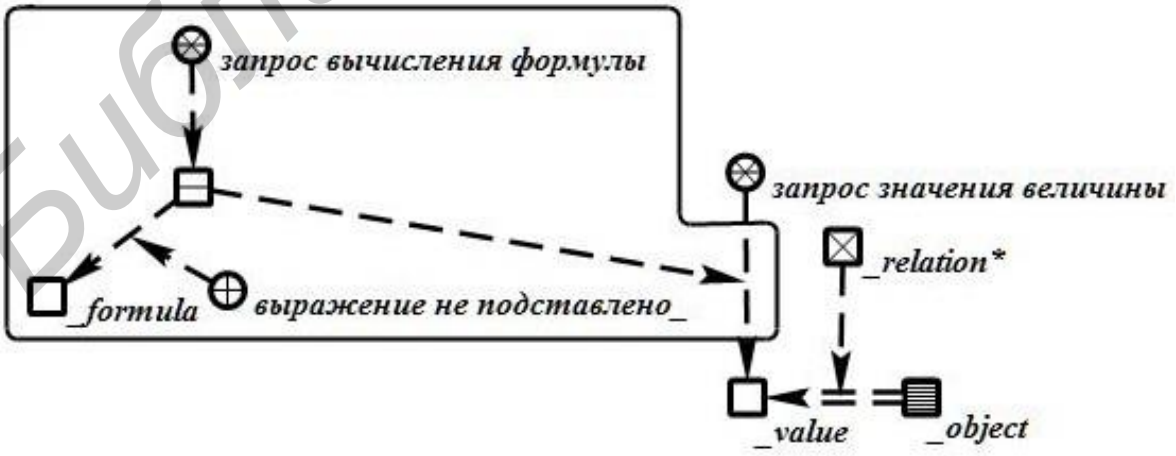

Рис. 7.3. Результат неудачного завершения работы операции

3. Завершение работы операции.

8. СОДЕРЖАНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Пояснительная записка содержит описание процесса проектирования интеллектуальной справочной системы, построенной с помощью семантических технологий проектирования интеллектуальных систем либо фрагмента такой системы.

Пояснительная записка должна содержать следующие разделы проекта интеллектуальной справочной системы.

Введение

Во введении, как правило, освещаются основные направления в области разработки и внедрения интеллектуальных систем, обосновываются актуальность темы и степень новизны, формулируются цель и задачи курсового проекта.

Технико-экономическое обоснование проектируемой интеллектуальной справочной системы

В разделе обосновывается актуальность разработки интеллектуальной справочной системы по выбранной предметной области; приводится описание пользователя системы (указание категории пользователей системы – исследователь, преподаватель, студент, экспериментатор, школьник, эксперт, турист и т. д.; указание функциональных возможностей предоставляемых системой конкретной категории пользователей); приводятся аналоги систем, решающие проблемы в заданной предметной области, их сравнение с разрабатываемой системой по различным критериям (функциональные возможности, многообразие поддерживаемых видов знаний, обработка знаний и т. д.); указываются преимущества (достоинства) разрабатываемой прикладной системы по отношению к аналогам; обосновывается использование семантической технологии проектирования интеллектуальных систем; оцениваются трудозатраты по реализации (изготовлению) системы.

База знаний интеллектуальной справочной системы

В разделе описываются фрагменты базы знаний, разработанные для отладки реализованных операций машины обработки знаний проектируемой системы. Сюда входят собственно исходные тексты базы знаний проектируемой интеллектуальной системы, оформленные на каком-либо из внешних вариантов записи внутреннего языка представления знаний.

Машина обработки знаний интеллектуальной справочной системы

В данном разделе описывается собственно разработанная в рамках курсового проекта система операций машины обработки знаний интеллектуальной системы. В данном разделе можно выделить ряд подразделов.

Спецификация машины обработки знаний интеллектуальной справочной системы

В данном пункте выделяются предметные задачи, которые должна уметь решать проектируемая интеллектуальная справочная система. Предметные задачи необходимо разбить на некоторые типы, т. е. провести классификацию предметных задач. «Интеллект» системы для данной версии определяется

многообразием предметных задач и их нетривиальностью решения (т. е. решение задач, для которых явно отсутствуют алгоритмы решения).

Далее приводится список используемых в операциях ip-компонентов. В данном пункте приводится перечень уже реализованных компонентов, необходимых для решения предметных задач интеллектуальной справочной системы. Здесь необходимо отметить как сами операции, так и программы, их реализующие и включенные библиотеки ip-компонентов.

В заключение подраздела приводятся декомпозиция операций на подпрограммы и содержательная структура библиотеки программ специфицированных операций. Декомпозиция операций на подпрограммы позволяет перейти на уровень программирования операций. В итоге необходимо указать перечень необходимых подпрограмм для реализации операций и определить структуру библиотеки операций.

Алгоритмы и исходные тексты программ, реализующие операции машины обработки знаний

Приводятся разработанные алгоритмы и тексты scp-программ и шаблоны для информационного поиска, описываются способы их вызовов.

Верификация и отладка программ специфицированных операций

В данном подразделе приводятся тестовые наборы, на которых тестировались разработанные операции. В случае выявленных ошибок приводится протокол тестирования с указанием типа ошибки, фрагмента и другой служебной информации.

Пользовательский интерфейс интеллектуальной справочной системы

В данном разделе рассматривается взаимодействие между пользователем системы и самой интеллектуальной справочной системой. Пользовательский интерфейс представляет собой специфическую интеллектуальную систему, построенную по семантическим технологиям, которая включает базу знаний пользовательского интерфейса и машину обработки знаний пользовательского интерфейса. Поэтому и база знаний пользовательского интерфейса и машина обработки знаний пользовательского интерфейса проектируется в соответствии с методикой проектирования этих компонентов.

Интеграция разработанной системы с другими системами

В результате такой интеграции может получиться новое качество интегрированной системы, когда на новые вопросы пользователей проинтегрированная система дает ответ, а две прикладные системы в отдельности – нет.

Направления дальнейшего развития разработанной системы

В данном разделе указываются направления дальнейшего развития прикладной интеллектуальной системы по различным направлениям: добавление новых видов знаний, наполнение базы знаний новыми знаниями, разработка операций обработки знаний с учетом специфики предметной области, разработка пользовательских интерфейсов.

СПИСОК ВОЗМОЖНЫХ ВАРИАНТОВ ТЕМ КУРСОВОГО ПРОЕКТА

1. Набор операций логического вывода интеллектуальной справочной системы по геометрии.
2. Набор операций логического вывода интеллектуальной справочной системы по физике.
3. Набор операций логического вывода интеллектуальной справочной системы по астрономии.
4. Набор операций логического вывода интеллектуальной справочной системы по теории графов.
5. Набор операций логического вывода интеллектуальной справочной системы по лингвистике.
6. Набор операций логического вывода интеллектуальной справочной системы по теории множеств.
7. Набор операций логического вывода интеллектуальной справочной системы по географии.
8. Набор операций логического вывода интеллектуальной справочной системы по числовым моделям.
9. Набор операций логического вывода интеллектуальной справочной системы по фармакологии.
10. Набор операций логического вывода интеллектуальной справочной системы по музыке.
11. Набор операций логического вывода интеллектуальной справочной системы по химии.
12. Набор операций логического вывода интеллектуальной справочной системы по истории.
13. Набор операций логического вывода интеллектуальной справочной системы по изобразительному искусству.
14. Набор операций логического вывода интеллектуальной справочной системы по автодиагностике.
15. Набор операций логического вывода интеллектуальной справочной системы по кулинарии.
16. Набор операций логического вывода интеллектуальной справочной системы по правилам дорожного движения.

СПИСОК ЛИТЕРАТУРЫ

1. Представление и обработка знаний в графодинамических ассоциативных машинах / В. В. Голенков [и др.] ; под ред. В. В. Голенкова. – Минск : БГУИР, 2001.

2. Голенков, В. В. Семантическая технология проектирования интеллектуальных решателей задач на основе агентно-ориентированного подхода / В. В. Голенков, Д. В. Шункевич, И. Т. Давыденко // Программные системы и вычислительные методы. – 2013. – №1.

3. Тарасов, В. Б. От многоагентных систем к интеллектуальным организациям / В. Б. Тарасов. – М. : Изд-во УРСС, 2002.

4. Базы знаний интеллектуальных систем: учебник / Т. А. Гаврилова [и др.]. – СПб. : Изд-во «Питер», 2001.

5. Гулякина, Н. А. Методика проектирования семантической модели интеллектуальной справочной системы, основанная на семантических сетях / Н. А. Гулякина, И. Т. Давыденко, Д. В. Шункевич // Программные системы и вычислительные методы. – 2013. – №1.

6. Давыденко, И. Т. Технология компонентного проектирования баз знаний на основе унифицированных семантических сетей / И. Т. Давыденко // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2013) : материалы III Междунар. науч.-техн. конф. (Минск, 21–23 февраля 2013 г.). – Минск : БГУИР, 2013.

7. Шункевич, Д. В. Модели и средства компонентного проектирования машин обработки знаний на основе семантических сетей / Д. В. Шункевич // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2013) : материалы III Междунар. науч.-техн. конф. (Минск, 21–23 февраля 2013 г.). – Минск : БГУИР, 2013.

Св. план 2014, поз. 29

Учебное издание

Голенков Владимир Васильевич
Гулякина Наталья Анатольевна
Гракова Наталья Викторовна и др.

ЛОГИЧЕСКИЕ ОСНОВЫ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

ПОСОБИЕ

Редактор *Е. Н. Чайковская*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *Е. Д. Степусь*

Подписано в печать 16.04.2015. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 3, 84. Уч.-изд. л. 4,0 Тираж 150 экз. Заказ 146.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6