Оглавление

[1. Понятие интеллектуальной системы 2](#_Toc154713430)

[2. Этапы развития понятия интеллектуальной системы 3](#_Toc154713431)

[3. Задачи и этапы решения задач 5](#_Toc154713432)

[4. Простая архитектура ИС 5](#_Toc154713433)

[5. Модель многомерной ИС 6](#_Toc154713434)

[6. Знания и их свойства 8](#_Toc154713435)

[7. Достоинства и недостатки ИС 10](#_Toc154713436)

[8. Продукционные модели знаний. Атомарные продукционные системы. Прямой вывод 11](#_Toc154713437)

[9. Механизм обратного вывода для атомарных продукционных систем 15](#_Toc154713438)

[10. Предикатные продукционные системы 19](#_Toc154713439)

[11. Система с исключениями из правил 20](#_Toc154713440)

[12. Иерархическая модель продукций 23](#_Toc154713441)

[13. Сценарные модели продукций 24](#_Toc154713442)

[14. Инварианты продукционного подхода: 26](#_Toc154713443)

[15. Иерархии в отношениях «являться» и «быть частью» 27](#_Toc154713444)

[16. Семантические сети предложений естественного языка 30](#_Toc154713445)

[17. Функциональные сети: прямая задача 32](#_Toc154713446)

[18. Обратная задача для функциональных сетей 34](#_Toc154713447)

[19. Иерархические семантические сети 36](#_Toc154713448)

[20. Обучающие пространства 38](#_Toc154713449)

[21. Формализмы логических программ. Основные понятия 40](#_Toc154713450)

[22. Логические программы с правилом резолюции (MT) 42](#_Toc154713451)

[23. Унификация предикатов. Алгоритм Робинсона 44](#_Toc154713452)

[24. Логические программы с механизмом перехода с возвратом. 47](#_Toc154713453)

[25. Дерево вывода для логической программы (ЛП) 50](#_Toc154713454)

[26. Специальные предикаты языка Prolog 52](#_Toc154713455)

[27. Задача экспертной классификации 53](#_Toc154713456)

# **Понятие интеллектуальной системы**

**Интеллектуальная система** – «(в узком смысле) компьютерная программа». Это система, которая решает задачи (здесь задачи подразумеваются как, например, найдите ошибку, выбросите лишнее, запомнить и тд.) в предметной области, используя для этого формализованное представление содержания этой области в качестве данных для работы системы. «ИС решает задачи предметной области». ИС опирается на знания предметной области, правила, законы, соотношения, ограничения всякие. В БД это триггеры, формулы, замечания, проверки дополнительные.

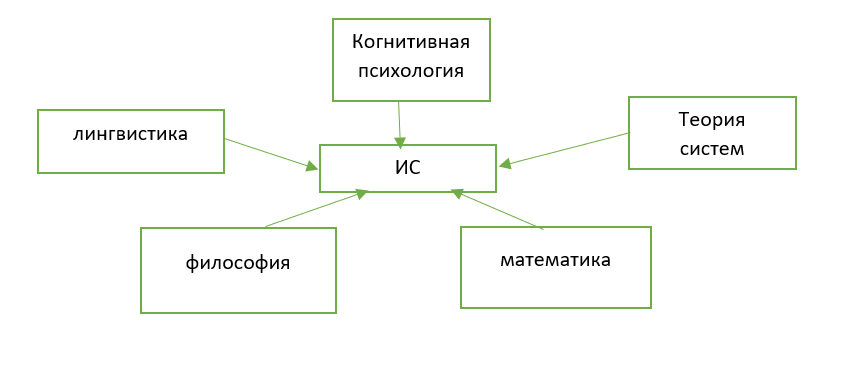
Для решения задач ИС применяют модели, моделирующие структуры памяти и процессы мышления. «Система ведет себя так, как ведет себя человек».

Для работы со знаниями применяются специальные форматы представления и операции обработки знаний. Операции над знаниями, основной операцией, если мыслить, как математик, это математическое рассуждение. В жизни с операциями сложнее, потому что могут быть операции понимания, запоминания (сохранить), ассоциирования знания с ситуацией.

Интеллектуальная система внутрь и наружу.

«наружу»

Формирование концепции ИС осуществляется с использованием понятий и конструктов областей знаний, которые занимаются изучением модели мышления и организации представления знания.



В философии есть два раздела: гносеология – теория познания и теория о том, как выглядят модели – антология.

Антология – система понятий, образующая каркас элементов, из которых собирается содержание предметной области, с помощью комбинаций которых создается описание содержания. Антология – это то, что рассказано на естественном языке.

Гносеология – как познавать, это механизм. Самый общий признак.

Математика: множества, операции над множествами, операции над элементами множеств и тд. Математика дает средства формализации.

Лингвистика – некая формализованная теория языка. Языка прежде всего естественного, разговорного. Язык выдает систему понятий, которыми апеллирует человек. И эти понятия уже можно каким-то образом интегрировать в какие-то структуры более-менее формализованные, применяемые в описаниях знаний и процессов обработки.

Описание знаний – это структурные элементы.

Процессы обработки – название операций, название действий.

Когнитивная психология: как мир познается? Когнитивная психология – психология мышления.

Теория систем занимается исследованием сложных систем, с разными моделями и свойствами поведения. То есть это не одна задача, которую надо решать, а какой-то агрегат, как сложная система. (пожар, как сложная система. Мы должны знать то, что знают пожарные, ДПС, страховщики, за что отвечает медицина, чиновники, электросети, гор. Газ, средства массовой информации и тд.).

# **Этапы развития понятия интеллектуальной системы**

1. Этап универсальных ИС (УИС). «универсальность – значит умеет решать все». В основе подхода лежит применение языка математики (вычисление предикатов).

Механизм решения задач – это правило вывода, применяемое в вычислениях предикатов.

Правило modus ponens (MP): если A, то B.

Правило modus tolles (MT) – от противного: ¬A, A←B, ¬B.

Данный этап закончился неудачей потому, что

* невозможно формализовать содержание больших областей знаний из-за большого объема неявны знаний;
* переборность общих механизмов решения задач.

1. Этап пакета прикладных программ (ППП) – это библиотека алгоритмов решения конкретных задач в заданной предметной области, снабженная механизмом подбора нахождения алгоритма наилучшей для решаемой задачи.
2. Этап экспертных систем 1970г. MYCIN – система диагностики назначения лечения вирусного заболевания крови. Экспертная система «заменяет эксперта».

Экспертная система – это ИС в узкой предметной области, которая решает задачи в этой области, используя ограниченное множество знаний и методы их обработки, применяемые специалистами.

Экспертная система ничего нового не предлагает то, что уже известно предметной области и то, как специалист принимает решение, она берет оттуда, там все расписано. Не надо никаких универсальных механизмов обработки и общих схем.

Каждая экспертная система своя в каждой области знаний. Нет универсальных моделей, единой модели представления знаний и нет единого языка. Она должна быть маленькой.

Недостатком является сложность обобщения и сравнения систем, которые свои для каждой области знаний.

1. Этап семантических информационных систем. Данный этап связан с появлением информационных ресурсов иерархической структуры. Такие ресурсы развивают концепцию SW – Semantic Web, реализованную для решения задач навигации и поиска нужно наделять ресурсы наборами свойств. В рамках этого подхода используются XML (RDF, OWL), которые позволяют создавать иерархические семантические структуры (деревья), листья которых содержат информационные объекты, а вершины – различные отношения и связывания между ресурсами.

Они соответствуют иерархическим БД сложных семантических объектов, позволяющих моделировать знания.

Иерархический формат представления знаний является унифицированным.

# **Задачи и этапы решения задач**

Задачи – это описание ситуации с указанием целей задачи в заданной ситуации. Задача – это всегда конкретное указание на исходные данные.

Зная <исходные данные> найти <цели задачи>

Исходные данные – это обычно набор требований к значениям параметров. Например: возраст > 20; время: 2000

Этапы решения задач:

1. Идентификация области знаний
2. Идентификация начальных данных в базе знаний
3. Извлечение фрагмента базы знаний
4. Нахождение последовательности применений знаний, позволяющей найти решение задачи
5. Применение этой найденной последовательности в конкретных условиях
6. Проверка найденного решения

Программа, решающая задачи предметной области, называется ИС, если она автоматизирует выполнение первых четырех этапов решения.

# **Простая архитектура ИС**

1. Простая архитектура ИС включает в себя:

* СО (система общения) – способ общения с внешней средой.
* МПЗ (механизм приобретения знаний)
* МВ (механизм вывода)
* БД (база данных)
* БЗ (база знаний)

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

# **Модель многомерной ИС**

|  |
| --- |
| Поверхностный **А** |
| Алгоритмический **В** |
| Когнитивный **С** |

Компоненты ИС, которые нужно создавать, связаны с характеристиками знаний, обрабатываемых в этих компонентах. Примером является – измерение:

* Уровня знаний

|  |
| --- |
| Неструктурированный **1** |
| Слабо структурированный **2** |
| Полностью структурированный **3** |

* Структурированность представления знаний

Изображение выглядит как диаграмма, План, Технический чертеж, Прямоугольник

Автоматически созданное описание

Каждая компонента архитектуры ИС имеет собственную память. Обычно организованную в форме иерархии. В каждой компоненте имеется собственная база знаний. Знания о том, как работать со знаниями в этом формате или с этими свойствами. Это знания о том, как работать в этой компоненте.

Есть области, в которых реализуются процессы обработки знаний. Процессы обработки знаний группируются в потоки. Поток – это сложная структура, состоящая в перемещении знаний между компонентами, а перемещать можно между соседними по одной координате, при этом в каждой компоненте связанности в системе выполняется некоторое преобразование знаков, то есть какая-то обработка производится.

Управление целями – это такая организация управления задач, при которой вся система поддерживает нахождение в определенном состоянии «(в состоянии устойчивости)» со способностью осуществления интеллектуальной деятельности. «6 компонентов обычно достаточно».

Еще одним компонентом ИС является процессы обработки знаний компонентов. Обычно они моделируются диаграммами операций (морфизмов) преобразования структур знаний.

Изображение выглядит как круг, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

ИД – исходные данные. F1 – анализ исходных данных на предмет, а нужно ли решать задачу, представляется ли задача актуальной для этих исходных данных. Дальше получается какое-то заключение. Попали в S2, обработали, получили. Дальше F2 – решение задачи. Установление возможности решения, проверили можем ли решать. Решаем, а дальше извлечение ответа.

Компоненты берутся из измерений. Компоненты архитектуры определяют соседство, то есть по одному свойству на одно значение меняется.

# **Знания и их свойства**

1. Знания – это виды ресурсов, применяемых для решения задач. С помощью знаний отображается содержание предметных областей. Знания – это какие-то законы, свойства, понятия, которые есть в предметной области. Определение понятия знания можно – это системный подход.

Можно перечислить свойства:

* Содержание и форма (обычно форма соответствует содержанию с точки зрения эргономичности).

Форма – формат представления знаний ассоциируется с понятием структуры.

Универсальные конструкции позволяют представлять знания в виде алгебраического выражения, в котором знание собирается как композиция других знаний.

((a ○ b) ○ (c ○ a)) ○ r

Изображение выглядит как линия, диаграмма, дизайн

Автоматически созданное описание

Содержание знаний – представление о возможности его использования. В качестве представления содержания применяется предикат вложения.

* Отделимость и связанность.

Каждое знание допускает независимое самостоятельное существование.

Знание допускает связывание в системы знаний и совместную обработку процессов решения задач. Как правило получаются серии знаний.

* Область определения и область значений

Область определения – ситуация, в которой знание может быть применено.

Область значений – результат применения, который выражается в изменении некоторых параметров или свойств содержания области знаний или базы знаний.

* Метазнания

Изображение выглядит как диаграмма, текст, Шрифт, круг

Автоматически созданное описание

Знания можно отнести к XML структуре. Знания относятся к предметной области. Это знания о предметной области. Метазнания – это знание о мире знаний.

* Неточность знаний

1. *Недостоверность*, когда знания имеют вероятностную природу. Исходные данные недостоверные, с какой-то вероятностью это проявляется. Вероятность возникает как в исходных данных, так и в самом правиле. В исходных данных, потому что у нас есть ошибки в измерениях, а в правилах из-за того, что мы не можем перечислить все параметры, которые на закон влияют.
2. *Нечетность* – знания, которые оперируют нечеткими понятиями. Нечеткое понятие – это свойство, которое присутствует всегда. Вот вероятность – это либо есть, либо нет, то есть есть какая-то частотность, а нечетность – это свойство присутствует всегда, но в разной степени.
3. *Многозначность* – ситуация, когда в одних и тех же условиях применяются разные знания, иногда противоречащие друг другу.

# **Достоинства и недостатки ИС**

Достоинства:

1. Высокий уровень решения задач, так как в ИС используется знания и опыт квалифицированных специалистов.
2. Невысокая стоимость
3. Устойчивость решений
4. Универсальность условий применения

Недостатки:

1. Ограниченность. Ограниченность заключается в принятии ограниченного разнообразия знаний как правило в узкой предметной области
2. Есть способность к обобщению. «Обобщение – способ сформулировать уникальный закон»
3. Неспособность к абстрагированию и творчеству
4. Методы решения задач основаны на понятии следствия из имеющихся знаний и поэтому существенно опираются на перебор имеющихся знаний.

# **Продукционные модели знаний. Атомарные продукционные системы. Прямой вывод**

Исторически при построении прикладных ИС применяются близкие (не значит совпадающие) по свойствам модели представления знаний.

1. Продукционные модели (модели правил)
2. Модели, основанные на семантических сетях
3. Логические модели (знания представляются в виде предложений логики первого порядка)

Существуют и другие подходы представления знаний, удобные в определенных случаях

**Атомарные продукционные системы**

Пусть задано множество A = {a1…an}, элементы которого называются атомами. Каждый из атомов может принимать одно из двух значений:

* Истинно
* Ложно

Продукциями называется выражения вида:

Где t1…tk+1 – атомы из A

Содержательно, продукция означает, что если t1…tk объявлены истинными, то истинным объявляется tk+1

Атомы t1…tk – посылки; tk+1 – заключение продукции

Если k = 0, то продукция называется аксиомой

Аксиома содержит атом, являющийся истинным по определению

Атомарной продукционной системой называется всякое конечное множество P = {}

1. Аксиома P обычно представляет начальные данные решаемых задач
2. Постановка задач:
   1. a-? (aA) – требуется установить истинность атома
   2. x-? (xA) – x – неизвестное, решением является множество атомов истинных для данной системы
3. Механизм вывода. Механизм решения задач

Возможны две основные схемы организации постановления задач

1. Прямой вывод
2. Обратный вывод

**Прямой вывод**

1. Пусть задана система атомарных продукций P (там все накидано и аксиомы, как правил это исходные данные, истинность которых известна и просто продукции). Составим два списка

* L1 – содержит атомы, являющиеся заключениями аксиом
* L2 – все остальные продукции

1. Рассмотрим список L2 – до тех пор, пока не встретится первая по порядку продукция, все посылки которой содержатся в L1, а заключения не содержатся в L1.
2. Если такой продукции нет, то механизм вывода завершает свою работу. Если продукция найдена, то добавляем ее заключение в L1.
3. Повторяем действие 2.

Механизм прямого вывода в атомарных продукционных системах всегда завершает свою работу. Это так, потому что на каждом шаге к L1 добавляется новый атом и множество атомов конечно, то количество итераций при работе механизма вывод ограниченно.

После этого задача a – ? получает ответ зависящий от содержимого L1. Если a L1, то a – истинно и выводимо.

Если a ∉ L1, то атом – не выводимый.

Ответом на задачу X – ? является весь список L1 (утверждение об истинности всех атомов из L1).

**Задача** (спросит на экзе)

Проверить, что если существует последовательность применений продукций, позволяющая в конце доказать истинность некоторого атома, то механизм прямого вывода объявляет атом истинным и добавляет в L1 обязательно. «(тут есть тонкость, что, если возьмем какой-то вывод, он может быть не единственным, там разный порядок применения продукций можно доказывать одно и то же. Например, есть какая-то итоговая продукция, которая зависит от двух атомов, мы ее можем доказывать с помощью двух продукций…)»

Замечание

Приведенная схема прямого вывода крайне неэффективна и как правило не применяется, поскольку допускает более эффективную реализацию, эффективную с точки зрения времени.

A = {a1…an} – продукционная система

P = {} – продукции

Представим продукционную систему в виде двудольного графа с двумя множествами вершин A⋃P

Если задана , то ей соответствует фрагмент графа вида:

Изображение выглядит как диаграмма, линия, зарисовка, График

Автоматически созданное описание

* 1. Вершины-атомы. Разметим вершины-атомы соответствующей аксиомы символом T, а каждой вершине продукции припишем количество посылок этой продукции k.
  2. Уменьшим разметки вершин продукций на количество атомов, которые объявлены истинными на шаге 1. (если у продукции один атом объявлен доказанным, то уменьшаем на 1).

Изображение выглядит как линия, диаграмма, График

Автоматически созданное описание

* 1. Составим список L – продукций, размеченных 0 (0 означает, что все посылки доказаны), а заключения не объявлены истинными (то есть L – все продукции, которые можно исполнять).
  2. Возьмем первый элемент списка L, разметим вершину атома заключения выбранной продукции символом T, то есть объявим его истинным. Для всех продукций, у которых aik+1 является посылкой, уменьшаем значение счетчика на 1.

Изображение выглядит как линия, диаграмма, Шрифт, дизайн

Автоматически созданное описание

Из рассматриваемых продукций добавим в L такие продукции, для которых число не доказываемых посылок равно нулю, а заключение еще не доказано, то есть заключение не размечено символом T и тд.

# **Механизм обратного вывода для атомарных продукционных систем**

Прямой вывод плох тем, что если база знаний большая и рассчитана на решение самых разных задач, то прямой вывод решает все задачи одновременно, он не думает о том, что он доказывает, он строит все следствия, которые может построить из исходных данных, то есть доказывает истинность любых атомов, для которых это можно сделать.

В основе механизма обратного вывода лежит конструкция дерева обратного вывода.

X – ? – решать все задачи вида a – ?, где a A

Атом постановки задачи припишем вершине корня дерева

Изображение выглядит как диаграмма, линия

Автоматически созданное описание

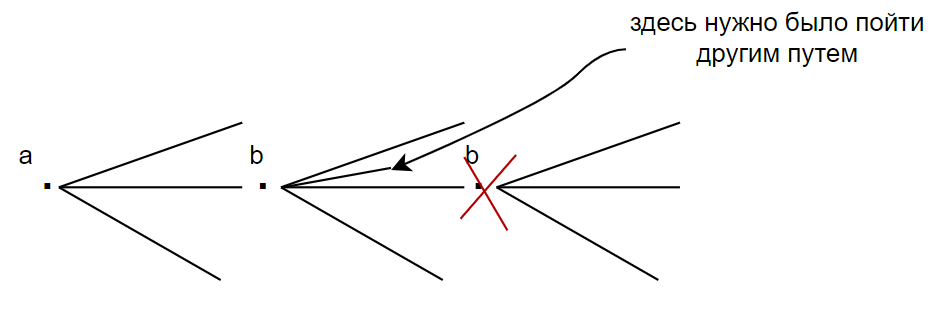
Потомками корневой вершины являются вершины, размеченные продукциями, для которых атом постановки задачи является заключением. Все продукции мы извлекаем и у продукций общее свойство: a – заключение для каждого из них.

Потомками каждой вершины первого яруса являются вершины, размеченные атомами посылками соответствующей продукции, каждая вершина первого яруса, размечена продукцией, а потомками этой вершины являются вершины, размеченные атомами посылок. Из этого следует, что у каждой вершины размеченной продукции потомков столько, сколько посылок и поскольку это дерево, если у двух продукций есть одинаковые посылки, то они представляются двумя разными вершинами, размеченными одинаково.

На третьем ярусе будут продукции (дерево и/или)

Для каждой вершины второго яруса постоим дерево обратного вывода для атома, приписанного этой вершине. Построение дерева заканчивается в вершинах, размеченных атомами, при выполнении одного из трех условий:

1. Атом, приписанный вершине, является аксиомой
2. Атом, приписанный вершине, не является заключением никакой продукции. То есть он не представлен как заключение аксиомы и нет других продукций, в которых этот атом является заключением. «То есть этот атом может быть посылкой, а заключением нет. Как правило аксиомы – исходные данные и когда есть традиционная база знаний, то обычно отдельно продукции и отдельно аксиомы. И когда исходные данные тасуются, тогда по отношению к другим продукциям может оказаться, что посылки и заключения могут быть только как аксиомы, но использоваться в качестве посылок в других продукциях».
3. Атом, приписанный вершине, встречается среди разметок вершин на пути из корня дерева в эту вершину



Дерево обратного вывода является конечным, так как оно содержит конечное множество вершин, которые можно оценить сверху.

Каждая вершина дерева имеет не больше чем max(m, n) потомков, поэтому при переходе с яруса на ярус количество вершин возрастает не более, чем в max(m, n) раз, поэтому, если глубина дерева (глубина – это число ярусов *l*, глубина – максимальная длина пути) *l* с ярусами 0…*l*, то количество вершин такого дерева не превосходит

Значение *l* < 2n, где n – количество атомов в системе.

Почему длина дерева не превосходит 2n? По правилу птичьих гнезд. Вершина продукции, вершины атомов чередуются. Если считать с нулевого яруса, то их будет не более чем 2n+1. Ярусы, размеченные атомами 0, 2, 4, 6… они будут встречаться n+1 раз, и n – размеченные продукциями. Больше не нужно, потому что по правилу птичьих гнезд у нас n разных атомов и n + 1 вершина, размеченная атомами, хотя бы одна точно повторится.

Проверить истинность атомов в корне дерева обратного вывода можно, если выполнить процедуру разметки вершин дерева как выводимых и невыводимых.

Разметим листья дерева как выводимые и невыводимые по следующим правилам:

1. Вершина, размеченная аксиомой – выводимая
2. Остальные – невыводимые

Для проверки истинности атомов в корне дерева обратного вывода, применим специальную процедуру разметки вершин дерева как выводимых и невыводимых. Разметка вершин выполняется по шагам:

1. Вершины – листья объявляются выводимыми, если приписанные им атомы являются заключениями аксиом, остальные листья объявляются невыводимыми (все вершины максимального яруса являются размеченными)
2. Рассматриваем ярус 2*l*-1. Вершины этого яруса размечены продукциями . Если вершина размечается и все потомки выводимы, то вершина также выводима. В противном случае вершина объявляется невыводимой.
3. 2*l*-2 на этом ярусе вершины размечен атомомами.

Остальные вершины объявляются выводимыми, если хотя бы одна вершина потомка объявлена выводимой. Если все потомки вершины яруса 2*l*-2, то вершина, размеченная атомом, является невыводимой. Эти вершины называются или-вершинами.

Продолжаем процесс до тех пор, пока корень не будет объявлен выводимым или невыводимым.

И так далее, продолжаем процесс до тех пор, пока корень дерева обратного вывода не будет объявлен выводимым или невыводимым.

Процесс завершается разметкой корневой вершины, так как на каждом шаге размечаются все вершины обрабатываемого уровня.

Если в полученном дереве какая-то вершина атома размечена как выводимая, то она соответствует атому, который истинный, и эта истинность может быть установлена путем применения некоторых продукций.

Когда объявляем внутреннюю вершину истинной

Для яруса 2*l*-2 все атомы, размеченные как выводимые, являются истинными.

Используя принцип математической индукции, атом в корне выводим, тогда и только тогда, когда он является истинным.

Если невыводима, то доказать нельзя, но это тоже надо доказывать. (Если стоит минус, значит ни одна продукция не работает, => все они не работают, значит у каждой есть какой-то изъян локальный).

**Замечание**

В прикладных алгоритмах, основанных на дереве обратного вывода, реализуется система виртуального обхода дерева в глубину и каждого яруса слева направо и при возврате удается установить выводимость вершин на основе и/или условий

Изображение выглядит как диаграмма, линия, рисунок

Автоматически созданное описание

# **Предикатные продукционные системы**

Пусть задана предметная область, которая оперирует системой параметров изучаемых объектов a1…an,каждый параметр принимает значение из вектора множеств Q1…Qn, и определено семейство предикатов:

P1(x, y)…Pk(x, y)

В продукциях и предикаты применяются P(a, c), где a – один из параметров объектов, которые мы изучаем, и относительно которых мы должны сделать какой-то вывод, а c – значение параметра из Q1…Qn.

Предикатной продукцией называется продукция вида:

t – терм, функциональное выражение, составленное из констант, имен параметров и операциями над значениями.

Лучше (состояние, плохое)

t позволяет что-то вычислять, создавать что-то новое, причем вычислять на основании того, что есть.

Применение продукции состоит в том, чтобы установить истинность предиката посылки, после чего утверждается истинность предиката заключения, если значение t может быть вычислено, т.е. все параметры, входящие в терм заданы своими значениями, поэтому их значения могут быть использованы в функциональных выражениях.

Если Больше (а, 5), а в продукции, которую мы хотим использовать и проверяем из базы знаний написано: Больше (а, 2), то даже не зная явно, что a>2, то этого Больше (а, 5) нам хватит явно, чтобы утверждать что посылка окажется истинной.

Рассмотрим схему прямого вывода.

Пусть задана предикатная продукционная система P = {}

1. Составим список предикатов:

L1 – являющийся заключением продукций аксиом

L2 – составлен остальными продукциями

1. Просматриваем элементы L2 до тех пор, пока не появится первая по порядку продукция, все посылки которой являются следствиями продукций из L1, значение терма t может быть посчитано и получаемое при этом заключение не является следствием предикатов в L1.

Если такая продукция найдена и заключения не являются следствием продукций предикатов в L1, то такое заключение добавляется в L1. После этого из L1 удаляются все предикаты, являющиеся следствием добавленного.

Если подходящая продукция не найдена, то механизм прямого вывода заканчивает свою работу.

# **Система с исключениями из правил**

«Переход от атомарных продукций к предикатным заключается в том, что мы получаем новые аспекты системы не обязательные, но новый вариант или аспект, связанный с возможностью обобщения нескольких фактов в один».

Изображение выглядит как текст, диаграмма, снимок экрана, Прямоугольник

Автоматически созданное описание

Продукции

b и c могут совпадать

Образуют пару: правило – исключение, если выполнимость условий (посылок) блокирует применение продукции .

*Пример*

В общем случае исключения из правила могут иметь собственные исключения и такие исключения выстраиваются в структуры в виде иерархии.

Изображение выглядит как Прямоугольник, диаграмма, линия, План

Автоматически созданное описание

Наползать друг на друга формально не могут, а фактически не знаем. Нужно стремиться, чтобы не наползали.

Необходимо разбить на две части:

* Та, что отменяет первое
* Та, что отменяет второе

Изображение выглядит как диаграмма, линия, зарисовка, дизайн

Автоматически созданное описание

1-7 – это исключения из правил, в которых содержится та область, которая нарисована.

Будем рассматривать «гнезда продукций» с двумя предположениями

1. Исключения из одного и того же правила являются дизъюнктными, то есть не имеют общи случаев.
2. Система правил с исключениями рассчитана на однократное применение, когда в любой ситуации в гнезде продукций, может быть использовано не более одной продукции.

Так может не быть, если исключение – это улучшение, а не отрицание. Это мы рассматривать не будем.

База знаний системы с исключениями из правил выглядит как последовательность «гнезд продукций».

Изображение выглядит как зарисовка

Автоматически созданное описание

Механизм прямого вывода в системах исключений из правил работает так: в каждом гнезде продукции каждое гнездо продукции применяется не более одного раза.

*Пример*

Грамматика русского языка

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

# **Иерархическая модель продукций**

Рассмотрим формат представления правил продукций, в которых посылки разбиты на группы

Q; R; A B, N (буквы берутся любые)

Q – система посылок, идентифицирующая цель

R – система посылок, идентифицирующая ситуацию

A – конкретные обстоятельства

A B – предположение

B – последствие продукции

Изображение выглядит как диаграмма, линия

Автоматически созданное описание

Достоинства иерархической модели продукций:

* Управление процессом формирования базы знаний (база разбивается на части, соответствующие разным началам Q, те разбиваются на части, соответствующие разным продолжениям R, и дальше идут небольшие, компактные группы правил, совершенно конкретной ситуации).
* Обработчик базы знаний допускает оптимизированную реализацию (продукции в некотором поддереве не проверяются до тех пор, пока не будут проверены все посылки на пути в корне этого дерева).

# **Сценарные модели продукций**

Рассмотрим модель продукций для случая, когда решение профессиональной задачи, в которой процесс решения разбивается на этапы, следующие друг за другом.

Каждый этап сценария использует собственную продукционную базу знаний P. Для нее определено понятие результатов R ее работы, а также условие A (предикат завершения этапа).

Изображение выглядит как Прямоугольник, диаграмма, линия, прямоугольный

Автоматически созданное описание

Переход с этапа на этап реализуется при истинности дополнительного предиката (условия) выполнения такого перехода.

A – истинен (т.е завершен) для перехода к следующему этапу

Возможны несколько типов диаграмм

Основными схемами этапов являются:

Изображение выглядит как диаграмма, План, Технический чертеж, схематичный

Автоматически созданное описание

В диаграмме сценария есть начальный этап, который связан с окружением системы.

Сценарная модель продукции называется моделью «классной доски»

Изображение выглядит как текст, снимок экрана, визитная карточка, Шрифт

Автоматически созданное описание

# **Инварианты продукционного подхода:**

1. Посылки продукций (атом, предикат, изображение).
2. Заключение (атом, предикат, изображение, действие).
3. Продукции заключения.

Продукции – это посылки заключения, а посылки – некая комбинация условий, описание ситуации, в которой продукция может быть применена. Заключение продукции – это указание на то, что является результатом применения продукции.

**Посылки**

Структура посылок – это управление логикой принятия решений о применении этой отдельной продукции. До сих пор у нас было, если мы применяем продукцию, то это почти наверняка означает, что у нас конъюнкция посылок выполняется, то есть выполняется каждый элемент этой конъюнкции, дизъюнкцию лучше не использовать.

Организация посылок: конъюнкция условий, дизъюнкция условий, логическая комбинация условий, упорядоченная последовательность (иногда это позволяет ускорить работу механизма вывода).

Иерархии: посылки образуют иерархии.

**Заключения**

Структура заключений: множество посылок – реализуются все заключения, для которых достаточно начальных данных.

1. Структуры множеств продукций

* Иерархия гнезд
* Семейство подсистем БЗ – база знаний распадается на части по этапам процесса решения задач.

1. Механизм вывода: прямой, обратный, исполнение сценариев, диаграмма этапов.

# **Иерархии в отношениях «являться» и «быть частью»**

**Модели семантической сети**

Подход семантических сетей основан на использовании ориентированных, нагруженных графах (графы, у которых вершинам и ребрам приписывается дополнительные сущности), вершины которых размечаются понятиями области знания, а ребра – отношениями между понятиями, размечающие соседние вершины.

Семантические сети (СС) предпочтительнее продукций, потому что в них можно интегрировать сложную систему отношений между сущностями

1. Иерархии в отношениях «являться» и «быть частью»

Пусть A – множество понятий предметной области A={a1,…,an}

На этом множестве определено 2 отношения:

Являться ⊆ AA

Быть частью ⊆ AA

Если a, b ∈ A и a явл b => множество объектов, соответствующих a вложено во множество объектов, соответствующих b.

Поскольку вложение множеств a ⊆ b является отношением порядка, то явл является отношением порядка.

Отношение быть частью (б/ч) называется еще отношением агрегирования, оно определяет архитектуру (структуру организации) остальных понятий.

a, b ∈ A, a б/ч b

Каждый конкретный объект, относящийся к b содержит в своей структуре элемент, относящийся к понятию a

*Пример:*

Колесо б/ч автомобиля

Отношение агрегирования также является отношением порядка (автомобиль не явл частью колеса)

Изображение выглядит как диаграмма, линия, текст, Параллельный

Автоматически созданное описаниеЗадачи, решаемые с помощью иерархии в отношении можно разбить на элементарные и сложные

1. Элементарные

* a явл b ?

Решение: получается, если проверить существование пути, ведущего из a в b вверх по отношению явл.

* a б/ч b ?

Решение: получает ответ «да», если на диаграмме явл и б/ч ∃ путь, ведущий из a в b, который сначала ведет вверх по б/ч, а затем спускается по отношению явл

Обобщение элементарных задач имеет вид:

X явл Y

X б/ч a

a явл X

где X, Y – неизвестные

1. Сложные задачи для иерархии отношений – это логические комбинации элементарных задач T1 & T2

Для сложных задач решаются отдельные задачи, после чего множество решений применятся для конструирования множеств составляющих ответ.

Семантические сети предложения естественного языка – это граф, вершины которого размечены словами предложений или же словами, задающими грамматические свойства понятий приписанных вершин

*«Редкая птица долетит до середины Днепра»*

Изображение выглядит как диаграмма, текст, линия, карта

Автоматически созданное описание

Лк Кати

Применение естественного языка содержит значительные трудности для формализации с помощью семантических сетей, позволяющих в последствии моделировать решение задач и использовать предложения как источник знаний и данных, обеспечивающих ментальную деятельность и разумное поведение в принятии решений.

# **Семантические сети предложений естественного языка**

Представление задач – вопросительное предложение (вопросительное слово и требование к вопр. объекту)

Различают несколько типов вопросов: Кто? Что? Сколько? Где? Когда? Почему?

Изображение выглядит как текст, линия, диаграмма, Шрифт

Автоматически созданное описание

Для решения задач строится семантическая сеть вопроса, которая сопоставляется (накладывается) с семантическими сетями отдельных предложений.

В простейшем случае сеть вопроса накладывается на сеть ответа так, что разметки вершины и ребер совпадают (за исключением вершины вопр. слова) => над вершиной вопроса и будет ответ

Изображение выглядит как текст, линия, Шрифт, снимок экрана

Автоматически созданное описание

Более сложная схема решения задач допускает несовпадение вершин (а также ребер). (задача общая, схема конкретная)

Также разметки должны находиться в отношении «общее» - «частное» (являться) и в качестве формируемого ответа генерируется семантическая сеть, в которой разметки вершины заменены на минимальное из сравниваемых значений в отношении «общее» - «частное»

*Редкий воробей долетит* (воробей – частное; общее – птица)

В более общем случае рассматривается расширение семантических сетей предложением, где несколько предложений объединены в одно (сверхфразовое единство) либо сети вопроса (ответа) дополняются расшифровками отдельных понятий

Это позволяет применят разные по содержанию вопросов и ответов предложения, представленные сетями (разные по составу)

*Общая задача*

В представлении таких описаний отслеживается достаточность фрагментальных совпадений для того, чтобы рассматривать одну или несколько семантических сетей в качестве основы построения.

В общем случае считается, что в естественном языке применяется более чем 250 разных отношений (независимые, разные категории), более 60 из них образуют базу независимых отношений, менее 10 – в реальных системах.

1. Являться – общее – частное
2. Быть частью – агрегирование
3. Свойство
4. Субъект действия
5. Объект действия
6. Место действия
7. Время действия и тд

# **Функциональные сети: прямая задача**

Пусть предметная область оперирует параметрами объекта A={a1,…,an}, каждый параметр принимает значение из множества

Q1…Qn, f1…fr – функциональные зависимости

aj = fi(ai1,…,ais)

каждое такое соотношение представляется фрагментом семантической сети, представляющую собой двудольный граф, вершины которого – a1…an и f1…fr

Приведенная функциональная зависимость отображается:

Изображение выглядит как диаграмма, линия, График, дизайн

Автоматически созданное описание

Обратное отображение может не существовать

Как правило, появляющиеся при этом сети можно разбить на несколько областей:

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

С помощью функциональных сетей решается два основных вида задач:

1. Прямая задача
2. Обратная задача (не то же самое, что прямой и обратный вывод)

Рассмотрим прямую задачу:

Зная ai1 = c1, … , ais = cs – исходные данные; c1,…,cs – менее конкретные представления об aij; найти: bj1, … , bjk – цели, используя f1, … , fr

Содержание задачи:

* Что дано?
* Что найти?
* Что можно использовать?

Общее решение прямой задачи:

Изображение выглядит как диаграмма, круг, линия, зарисовка

Автоматически созданное описание

Для решения прямой задачи можно воспользоваться схемой прямого вывода, когда вычисление bj1, … , bjk осуществляется по шагам и на каждом шаге находится первая по порядку функция, для которой определимы все исходные данные, а вычисляемое значение еще не определено.

Тогда такая функция применяется, и вершине вычисляемого значения приписывается значение.

Процесс продолжается до те пор, пока возможно вычисление новых значений понятий.

Результат решения задачи: значения максимальных параметров, которые были посчитаны. Если значения посчитаны не все, то можно дополнить схему вывода (решения задачи) анализом недостающих параметров исходных данных, добавление которых задано в задаче. Позволяет вычислить все значения цели.

Для этого можно использовать схему обратного вывода, когда по понятию цели устанавливается понятие (параметры) значения цели с помощью заданных функций

Построение аналога дерева обратного вывода позволяет перейти (выйти) к вершинам начальных данных, из которых можно выбрать минимальное множество вершин, значения которых, добавленные в исходные данные позволяет посчитать недостающие значения параметров C.

Рассмотрим обратную задачу (мы знаем, что должно получиться)

Изображение выглядит как диаграмма, круг, линия, текст

Автоматически созданное описание

# **Обратная задача для функциональных сетей**

Зная bj1 = c1, … , bjk = ck, найти ai1, … , ais, по которым с помощью f1, … , fr можно посчитать заданные значения параметров bj1, … , bjk

Основная проблема обратной задачи: невозможность явного применения f1, … , fr, так как обратных функций (отображений), ведущих из bjk в ais может не существовать

Пример: c: a+b

Отсутствие явных схем решения обратной задачи делает возможным переборный принцип решения таких задач.

Пусть

ai1 … ais

Qi1 … Qis – конечные множества

Строится дискретизация (– это из каждого множества выделяется конечное подмножество) множеств: Qi1’ … Qis’

Из дискретизированных множеств составляется пространство состояний начальных данных m = Qi1’ x Qis’ – решетка в S-мерном пространстве

В каждом узле решетки выполняется вычисления bj1, … , bjk по набору узла решетки и оценивается отличие посчитанных значений bj1, … , bjk от заданных c1 … ck

Пример того, как это работает:

10 значений, 10 исходных данных, 210 комбинаций (все перебрать), для каждого случая решить задачу, найти цепочку, затем повторять – считать усредненные отклонения для каждого, затем из всех вариантов выбрать наилучший.

Перебор вариантов является трудоемким процессом, поэтому эффективность переборного алгоритма низкая.

Разработка быстрых алгоритмов решений обратных задач связаны с поиском дополнительных условий f1…fn, при которых можно так организовать обход решетки, при котором на каждом шаге будет выполняться приближение к отыскиваемому решению. Такие методы называются градиентными.

*Пример:*

f(x1…xd) – монотонная по каждой из своих переменных. Это позволяет изменять положения выбираемых наборов, чтобы обеспечить нужное изменение отклонений вычисляемого набора значений параметров bi1…bin, приближающие вычисляемые вектора к векторам исходных данных.

Изображение выглядит как диаграмма, круг, рисунок, шаблон

Автоматически созданное описание

Пусть для некоторого начального набора d значение b больше, чем исходное значение, тогда требуется так выбрать новый набор значений a, b, c, d, e, при котором b уменьшится. Либо увеличим, либо уменьшим e. Для этого уменьшим d. Для этого уменьшим d и увеличим e, с увеличим, а уменьшим.

После этого выполняется изменение a, b, c, d, e по дискретным множествам Q1’…Qs’

Выбранный новый набор используем для вычисления нового значения b, например отклонения, если все еще будет больше, чем нужно, еще раз уменьшаем, если выскочили на другую сторону, начинаем движение в противоположную сторону.

# **Иерархические семантические сети**

Содержательно ИСС – это сеть, отдельные вершины которой могут представляться сетями.

Рассмотрим формализацию этого понятия. Пусть V – алгоритмически перечислимое множество, элементы которого называются вершинами семантических сетей. Перечислимость означает существование алгоритма, который, работая во времени неограниченно, последовательно строит все элементы V.

V разбивается на V1 ∪V2, где V1={*v*11, *v*21, …} – элементарные вершины,

V2={*v*12, *v*22, …} – вершины сети

На V определено множество отношений R={r1…ri,…} ∀i(riV×V)

Относительно отношений r1, r2, … предполагается, что они образуют перечислимое семейство разрешимых множеств пар вершин, то есть есть процедура, которая генерирует эти отношения и такая пара вершин из V за конечное время может сказать принадлежит ли эта пара r1, принадлежит ли она r2 и так далее.

Семантическая сеть ∑ – ориентированный граф

∑=(V’, U’), где V’V, U’ – множество ребер, размеченных семантическим отношением

U’(V’×V’×R)

U’=(a, b, r) => arb

«Одна и та же пара вершин в этом определении может связываться несколькими ребрами, различными, разными отношениями. Можно также сделать, чтобы такого не было, но это придется перестраивать логику сетей, когда дублирующие вторые параллельные связи с другими отношениями между вершинами спрячем»

Обозначим как N – множество всех семантических сетей, это множество является перечислимым, то есть все сети можно переписать с натуральным индексом.

∑1, ∑2, ∑3, ∑4,… →

Такие сети можно перечислить перечислением конечных подмножеств V и перечислением конечных фрагментов отношений r1, r2, r3, …

Введем отображение :V2→N – это отображение, сопоставляющее вершинам сетям представляющее их сети.

Предложенная система понятий позволяет формировать иерархическое представление произвольной сети

Изображение выглядит как зарисовка, рисунок, Штриховая графика, графическая вставка

Автоматически созданное описание

Естественными ограничениями, требованиями к сети является конечность глубины получаемой иерархии. Вот для каждой сети иерархия может иметь свою глубину: 10, 20, 30, 50 и тд, но в каждой сети она должна заканчиваться. Нет ограничения на сложность.

Зацикленность – это когда двигаясь по ярусам сверху вниз мы встречаемся с сетью, которая содержит вершину сеть, встречавшуюся в предыдущих сетях как источник пути вниз.

Последнее свойство достаточно легко обеспечивается, если множества вершин конечны.

# **Обучающие пространства**

Рассмотрим пример применения конструкций ИСС для моделирования процессов адаптивного обучения.

Пусть A – множество A={a1, a2, a3, …}, элементы которого называются элементарными знаниями. Это могут быть: определения понятий, описание свойств, комментарии, замечания, примеры, контрольные вопросы, доказательства, следствия и тд.

Обозначим B(A) – множество конечных подмножеств множества A

:A→B(B(A)) – функция обучения позволяет проструктурировать начальные концепты, которые можно обучить.

Изображение выглядит как зарисовка, рисунок, Штриховая графика, круг

Автоматически созданное описание

Каждое множество из (a) составляет элементарные знания (a’, a’’, …, an) достаточное для того, чтобы обеспечить изучения a, и эти знания собираются в CC, определяющую порядок изучения или обработки и прохождения элементов (a’, a’’, …, an).

Такая сеть определяет общую схему обучения, а также ее варианты.

Изображение выглядит как диаграмма, линия

Автоматически созданное описание

Т. е. сеть включает в себя описание обязательных элементов обучения, то есть какую информацию можно передать и проверить, а также вспомогательных: консультации, подсказки, и все это управляется отношениями, (которые различают обязательные и необязательные), и условиями продолжения обучения. Это вот задачи, которые реализуются на конечных системах объектов, понятий или концептов, из которых собираются сети изучения вот отдельных понятий или сущностей.

Управление процессом обучения осуществляется адаптивно к системе знаний, которыми обладает человек. Это позволяет исключить возможность повторного изучения одного и того же материала.

Выбор схемы прохождения обучения: использовать критерий оптимальности, гарантирующий получение знаний по некоторому предварительно определенному списку целей.

# **Формализмы логических программ. Основные понятия**

**Модели представления знаний. Логические программы.**

Элементы логических программ:

* Константы – отдельные элементы конкретных множеств

В логических программах константы задаются с помощью символьных выражений.

2 1+1 два

С точки зрения понятий человека – это одно и то же, а с точки зрения программы – это три разных константы, до тех пор, пока мы не скажем о другом.

* Переменные – способ обращения к элементам некоторого множества констант X. Переменные задаются именами и в именах первый символ всегда прописной. В константах нет прописанных символов.
* Термы (функциональные выражения)

Каждое функциональное выражение выглядит как f(t1, … ,tn) и

f – функциональный символ

Простейшими термами являются символы переменных и обозначение констант, а в общем случае терма необходимо, чтобы t1, … ,tn корректно применялись в f(t1, … ,tn).

Корректность означает, что те значения, которые можно посчитать, используя функциональное выражение t1 вложено во множество значений первого параметра функции f, значение t2 имеет значение второго параметра, tn – значение n-го параметра.

* Предикаты – основа конструкции логической программы. Предикаты P(x1, … , xn) соответствуют истинностным утверждениям с параметрами, принимающими значения заданных множеств. На каждом наборе значений параметров предикат является истинностным или ложным. В логических программах истинность предикатов доказывается, а ложность никак.

sum(x, y, z) + 2 2 4

- 2 2 5

* Предложения логической программы – это конструкция одного из видов:

p(t1, … , tn) – запись предикатов

p(t1, … , tn)∈B1, … , Br. Такие предложения называются импликациями, где p(t1, … , tn) называются головой предложения, а B1, … , Br – конъюнкция. Если факт содержит символы переменных, то он называется универсальным.

* Логическая программа – множество или последовательность предложений. Предложения завершаются точками.

Символы переменных локализованы в рамках содержащих их предложений. Одинаковые символы переменных в разных предложениях не связаны между собой, а один и тот же символ переменной в одном предложении означает одно и то же.

* Правило вывода (схема рассуждения). Схемы вывода – это некие схемы рассуждения, корректные для логических программ. Правило вывода соответствует схемам рассуждения. В формальной логике правило вывода – это схемы вида:

, где B1,…,Br+1 – шаблоны предложений в программе (шаблоны формул в языке, соответствующем математической теории).

B1,…,Br – посылки при условии применения правил.

Br+1 – заключение, а также это конструкция, собираемая из B1,…,Br.

Из разных схем правил вывода рассмотрим 2:

* Схема правила логического следования (MP)

 – закон следования

 Изображение выглядит как снимок экрана, линия, диаграмма, дизайн

Автоматически созданное описание

* Правило рассуждения от противного (MT)



– истинно => A – ложно => 0 ← B – истинно => B – ложно =>  – истинно.

Изображение выглядит как снимок экрана, линия, диаграмма, дизайн

Автоматически созданное описание

* Постановка задач в логических программах

Задача, решаемая с помощью логической программы (доказываемая цель) представляется в одном из форматов p(t1,…,tn) – ?, если механизм вывода основан на правиле MP и , если механизм вывода основан на правиле MT.

# **Логические программы с правилом резолюции (MT)**

Пусть заданы предложения:



где – родительские предложения,  – резольвента родительских предложений. «Резольвента – это разрешить ситуацию для двух предложений»

В общем случае правило резолюции имеет вид:

 это необходимо проверить самостоятельно.

Особый случай:



Уменьшение числа предикатов в отрицании завершается ситуацией:



Рассмотрим задачу:

p(t1,…,tn)

*sum*(2, 2, *x*)



Когда дойдем до противоречия x будет равен 4.

При реализации исполнения механизма вывода по правилу резолюции используется дополнительная операция. Для этого выполняется уточнение содержания переменных, символы переменных в S1 и S2 заменяются на термы так, чтобы в S1 и S2 совпадали записи предикатов в отрицании и в голове импликации.



Пример:

Давать(x, y, z)

x – кто, y – кому, z – что

Получать(x, y)

S1: Получать(y, z) ← Давать(x, y, z)

S2: Давать(учеба, студент, знания)

Q: Получать(студент, X)



При унификации выполняется замена переменных так, чтобы у S3 и S1 множества переменных были дизъюнктными, поэтому меняем X в постановке задачи на *v.*









Следовательно, доказано, что получать(студент, знания)

Замечание:

В общем случае для решения произвольных задач с помощью правил резолюции справедливо, что если некоторый предикат p(x1,…,xn) для какой-то замены переменных на константы является следствием логической программы, то добавление к программе отрицания p позволяет получить противоречия для заданного следствия из логической программы.

# **Унификация предикатов. Алгоритм Робинсона**

Согласование степени общности. Пусть заданы предикаты:



Где t1,…,tn и  – термы (функциональные выражения).

Будем считать, что символы переменных в обоих предикатах x1,…,xn и множества символов переменных – дизъюнктные.

Подстановкой  называется замена переменных x1,…,xn на термы a1,…,am

Тогда запись – результат замены в записи предиката p(t1,…,tn) всех вхождений x1,…,xm на соответствующие им термы в .

Аналогично определим , тогда  – общий унификатор p(t1,…,tn), если =.

Будем рассматривать только такие унификации, при которых совпадения записей предикатов обеспечивается без дополнительных вычислений значений термов.

Пример:

p(x, 5, x) 

Q(y, y, z)

p(3, y, 4)

p(x, x, x)

Здесь унификация невозможна.

Наиболее общий унификатор

Подстановка , являющаяся общим унификатором предикатам  называется наиболее общим унификатором предикатов, если 

Где z1,…,zr  – все различные символы переменных в термах a1,…,am.

Содержательно наиболее общий унификатор минимально детализирует содержание символов переменных в предикатах.

**Алгоритм Робинсона**

Пусть заданы предикаты , а также x1,…,xn – первые отличные символы переменных, дизъюнктно встречающихся в исходных предикатах.

Образуем начальную подстановку:



Образуем список пар термов:



«Теперь по L будем двигаться и согласовывать изменения в ». Будем просматривать по шагам элементы списка L до тех пор, пока все элементы L не будут просмотрены. Здесь есть проблема: иногда список L удлиняется, тогда возникает проблема завершения.

Пусть выполнены *i-*1 шаг и внесены необходимые изменения в  и в список L.

Пусть на первом шаге рассматривается пара 

Возможны несколько случаев:

1. ti – константа, тогда i
   1.  – константа
   2.  – переменная
   3.  – терм

Если 1.1 и , тогда унификация невозможна, иначе если , то продолжаем процесс, переходим к следующей паре из L, менять ничего не надо, они одинаковые и у нас все хорошо. В случае 1.2 в  символ переменной xi соответствует tiи в остальных термах , содержащих  выполняется такая же замена.

После этого в оставшейся части списка L  заменяется на константу.

1. ti – переменная
   1.  – константа
   2. – переменная
   3. – терм

Здесь 2.1 то же самое, что и 1.2. Заменим ti на везде, где нужно в , затем в оставшихся парах списка L выполним аналогичную замену. В случае 2.3 выполним замену ti на  в  аналогично заменим ti в термах, подставляемых вместо переменных в  и также заменим символы переменной ti в остальных парах списка L.

Пусть 

Если , то унификация невозможна

Если , то добавим в L дополнительные пары термов



Лк Насти

# **Логические программы с механизмом перехода с возвратом.**

Пусть k – максимальное количество переменных у тех функциональных символов, которые применяются в записях термов предикатов. И пусть d – максимальная глубина термов как формул, составленных из функциональных символов, это вложение функции в функцию.

Что есть: n – количество параметров, размерность предиката p



Тогда при переработке первых n пар из начальной версии списка L, то количество добавленных пар можно оценить как n\*k.

Если отследить d этапов процесса, то на последнем этапе будет добавлено не больше n\*kd пар, в которых все термы имеют глубину 0, то есть либо константы, либо символы переменных. После этого случай (3.3) невозможен.



Значит процесс работы алгоритма заканчивается за конечное время, и получаемая в конце подстановка  является наиболее общим унификатором. Это так, поскольку все замены переменных по приведенной схеме реализуются максимально общим способом.

**Механизм вывода в языке Prolog**

Пусть задана логическая программа P и предикат p(t1,…,tn) – цель.

Процесс решения задачи p(t1,…,tn) – найти t1,…,tn, при которых предикат доказуем.

Сформулируем схему работы механизма вывода алгоритма доказательства цели p(t1,…,tn)

1. Пусть требуется доказать p(t1,…,tn). Найдем первое по порядку предложение программы: , то есть предложение, голова которого унифицируется с p(t1,…,tn), если такого нет, то p(t1,…,tn) недоказуемо (недоказуемо с помощью логической программы).

Если такое предложение есть, то пусть  – наиболее общий унификатор p(t1,…,tn) и .

Образуем конъюнкцию целей: B1,…,Bk для предикатов в хвосте найденного предложения. Хвост предложения – это конъюнкция предикатов B1…Bk. «Если k = 0, то конъюнкция пустая и доказывать ничего не надо».

«Цель можно доказать, если мы столкнулись с фактом».

Если k>0, то цели B1,…,Bk доказываются последовательно слева направо. При этом каждая последующая цель конъюнкции производится с учетом результата доказательства предыдущих целей конъюнкции.



Если ничего не меняется, то ’’ – тождественный унификатор и ничего не меняется.

Если доказательство конъюнкции цели закончилось, то следует p(t1…tn)’…k

1. Пусть при доказательстве целей B1,…Bk некоторая цель не может быть доказана.

Некоторая цель Bi’…i-1 не может быть доказана (не доказуема с помощью логической программы)

В таком случае выполняется возврат к ближайшей слева цели, которая может быть передоказана по-другому

Изображение выглядит как диаграмма, дизайн, линия, Шрифт

Автоматически созданное описание



Точка возврата – предложение, которое было использовано для доказательства, с него надо продолжать искать предложения для передоказательсва.

Если такая цель найдена (ближайшая слева, которая может быть передоказана) Bj’…j-1\*j-1…, то, начиная с последней доказанной цели, все последующие цели доказываются заново в виде:

Bj+1’…j-1\*j+1…

Если передоказать предыдущие цели нельзя, то цель p(t1,…,tn) объявляется недоказуемой с помощью предложения 

1. Если цель p(t1,…,tn) нельзя доказать с помощью предложения , голова которого унифицируется с p(t1,…,tn):

Ищется следующее по порядку предложение логической программы P, голова которого унифицируется с целью p(t1,…,tn). Если такого предложения нет, то исходная цель объявляется недоказуемой с помощью логической программы.

# **Дерево вывода для логической программы (ЛП)**

Пусть p(t1,…,tn) – цель. Поставим ей в соответствие дерево с размеченными вершинами и ребрами, которое полезно для разработки (понимания) механизма вывода переход с возвратом.

Образуем корень дерева, который разметим предикатом p(t1,…,tn).

s1…sr – предложения ЛП, головы которых унифицируются с p(t1,…,tn).

Изображение выглядит как линия, диаграмма, текст, График

Автоматически созданное описание

Разметим ребра, ведущие в вершины первого яруса наиболее общими унификаторами, связывающих цель и головы соответствующих предложений. Для каждой вершины первого яруса образуем множество потомков, которое составляют предикаты из конъюнкций целей хвостов предложений. Если ярус размечен предложением, то он может быть заключительным, если само предложение состоит из головы, а хвоста нет.

Четные ярусы – предикаты, возможно предикат, умножаемый на какую-нибудь подстановку; нечетные – предложения логической программы, лист – предикат, если он недоказуем

Для каждой вершины четного яруса строится дерево вывода с целью в этой вершине для предиката цели, приписанного этой вершине.

Механизм вывода в Prolog реализует обход дерева вывода p(t1…tn), основанный на обходе в глубину слева направо.

При этом, при обходе различаются и и или вершины. Или-вершины – это вершины, размеченные предложениями, а и-вершины – вершины, размеченные предикатами целей из конъюнкции целей, которые возникают в процессе обхода.

Или-вершины позволяют решать задачу, для которой достаточно доказать одну из них. И-вершины необходимо доказывать все

Прохождение дерева в глубину прекращается всякий раз, когда удается доказать некоторую цель, а это получается тогда, когда цель унифицируется с фактом.

При этом выполняется возврат

Изображение выглядит как часы, круг, рисунок, линия

Автоматически созданное описание

Если при возврате оказывается, что конъюнкция целей доказана полностью, то доказано предыдущее, из которого выполняется еще один подъем наверх

Изображение выглядит как зарисовка, рисунок, Штриховая графика, белый

Автоматически созданное описание

Если при доказательстве цели приходит возврат не в последний предикат конъюнкции целей, то из предложения, вызвавшего доказанную цель, происходит переход вниз и обход дерева в глубину продолжается, пока опять что-то не докажем.

Изображение выглядит как диаграмма, линия, зарисовка

Автоматически созданное описание

Поэтому для моделирования возврата необходимо запоминать точки возврата, которые позволяют отделить обработанный фрагмент дерева от необработанного.

Изображение выглядит как зарисовка, рисунок, линия, дизайн

Автоматически созданное описание

Запоминание точек возврата позволяет продолжить поиск решений при возврате, которое продолжает найденные решения, если получено новое решение, то дальнейшее прохождение дерева начинается заново.

Особенностью механизма вывод является невозможность решения задачи по программе, в которой достаточно знаний. Эта ситуация соответствует бесконечной ветви в программе.

Изображение выглядит как зарисовка, линия, рисунок, диаграмма

Автоматически созданное описание

# **Специальные предикаты языка Prolog**

fail – предикат неудачи. Доказательство этого предиката завершается неудачей и запускается механизм отладки.

Нужен для реализации поиска всех решений



P позволяет накапливать опыт. G – построит все решения.

Предикат отсечения: !

При решении задачи все дерево не нужно. Например, для решения A<0, если докажем это, то эти случаи:

A>0

A=0 не нужны

С точки зрения семантики: P ← B1,…,Bi-1,!,Bi+1,…,Br

Этот предикат не имеет параметров.

При доказательстве конъюнкции целей хвоста предложения этот предикат доказывается безусловно и при всякой попытке передоказательства мы вернемся к нему

Изображение выглядит как линия, диаграмма, Шрифт, дизайн

Автоматически созданное описание

Программа на языке Prolog различает два вида отсечения: зеленые и красные

Зеленое отсечение – это отсечение, которое можно убрать из программы не нарушая их равенства.

Красное отсечение – это отсечение, которое нельзя убрать из программы.

Красное применяется для моделирования

notP(x) ← P(x), ! fail.

notP(x), если отбросим отсечение, то программа работать не будет.

# **Задача экспертной классификации**

a1…an

Множества Q1…Qn – линейно упорядочены, позволяют определить сравнение параметров 

Пространством состояний называется множество m=Q1×…×Qn – это множество возможных значений параметров объектов, которые мы исследуем.

Диагнозом называется разбиение m= m1…mk

Когда k = 2 говорят о положительном m+, и отрицательномm- диагнозе, т.е. есть заболевание или его нет.

Представление экспертных знаний m+, m-

Экспертное знание формируется в виде конъюнкций сравнений параметров со значениями

1. (a1>c1&a2c2&a3<c3)

Каждое условие эксперта относится к одному из списку:

L+ – список условий попадания в положительный диагноз

L- – список условий попадания в отрицательный диагноз

Изображение выглядит как диаграмма, линия, Прямоугольник, План

Автоматически созданное описание

Задачей экспертной классификации называется задача нахождения двух алгоритмов:

1. Алгоритм опроса эксперта
2. Алгоритм классификации произвольных наборов

**Автоматизация опроса эксперта**

1. При опросе эксперта можно автоматизировать следующие функции
   1. Проверка правильности утверждений эксперта. Можем отследить нет ли противоречий между собой.
   2. Обобщение ответа эксперта

Изображение выглядит как Прямоугольник, линия, диаграмма, прямоугольный

Автоматически созданное описание

* 1. Уточнение неотклассифицированной области
  2. Увеличение размерности пространства состояний