Представление знаний и операции над ними

(теория экспертных систем)

Раздел.5[1]

---

[1] Введение в методы программных решений. Учебное пособие. / Е.В. Белкин, А.В. Гахов, А.М. Горбань, В.М. Куклин. В.М. Лазурик, А.С. Петренко, М.Ю.Силкин, В.В. Яновский. Под ред. проф. Куклина В.М. – Х.: ХНУ им. В. Н. Каразина, 2010.

[2] Дополнительный материал, не включенный в сборник.

5.1.Простые экспертные системы[3]

5.1.1.Введение

Естественный язык в большинстве своем содержит конструкции, которые кроме простых логических посылок, являются также ключами для инициализации существующего у каждого человека целого спектра понятий и логических структур. Фразы естественного языка способны вызывать отголоски пережитых и зафиксированных ранее в памяти ощущений, могут формировать целостные чувственные представления. При этом мы здесь не обсуждаем вовсе интонацию, жесты, выражения лица говорящего. На первый взгляд создается впечатление, что попытки создать аналог естественного языка формальными методами обречены на провал. Большие трудности возникают при формальном кодировании и декодировании фраз естественного языка.

Проблемы конструирования интеллектуальной системы:

- Трудности понимания запросов на естественном языке (но обойти можно, определив понятный машине язык запросов) [1].

Надо обучать умению задавать вопросы

Человек тоже часто не знает что спросить, не способен даже формулировать вопросы, если его предварительно не подготовить к разговору, если заранее не очертить контуры обсуждаемой проблемы.

- Трудности логического вывода ответа, исходя из хранящейся в базе информации.
- Для понимания запроса и вывода может потребоваться информация, которая не имеется в базе (эту информацию эксперты могли опустить, полагая её общеизвестной).

Консультирующие экспертные системы – системы диагностики, системы, способные формулировать заключения из определенной предметной области. Для этого используют знания экспертов. Эти знания часто являются не полными, не точными и достаточно неопределенными (интуитивными) и, порой, на первый взгляд и не логичными.

Виды задач

Доказательство теорем. Этот класс интеллектуальных задач основан на дедукции на основании гипотез, здесь нужны интуитивные навыки (появляющиеся, увы, только из опыта) выбора набора промежуточных лемм.

При доказательстве теорем виден не только путь решения, но и само решение

Так называемые конструктивные теоремы полезны тем, что после их доказательства виден путь решения, сформулированы логические связи (леммы), указаны значения параметров и переменных. Собственно, именно при выполнении этих условий подобные теоремы могут быть названы конструктивными. Их использование может оказаться полезным на практике.

Роботика (robotics) – научное направление об организации целесообразного поведения робота с сенсорными и исполнительными (эффекторными) механизмами. Рассмотрение построено на просмотре состояний (мира) и описания процессов изменения этих состояний. При этом создаются планы этих изменений, планы последовательности действий, методы управления планами. Планы содержат разный уровень детальности.

Автоматическое программирование (пример - компиляторы, получают спецификацию на входном коде и пишут программу в объектном коде). В системах ИИ это описание на высоком уровне некоторого утверждения (точного в формальном языке -

[3] Для ознакомления с принципами создания интеллектуальных систем можно ограничится подразделом 5.1.

исчисление предикатов, - или не точного в естественном языке, тогда нужно предусмотреть уточняющие запросы), которое за счет автоматического программирования будет воспринято машиной. Сопутствующая задача – ***верификация программы***, - доказательство того, что система достигла результата. ***Отладка*** – стратегия всей (полной) программы.

Логические операции - основа систем искусственного интеллекта

Введение большого числа логических операций, позволяющих при их применении машине фактически выбирать путь решения, в какой-то степени приближает программу к системам искусственного интеллекта. Вообще говоря, переход от жесткой алгоритмической структуры к системам логического вывода лежит сначала в замене множества традиционных математических и конструкторских операций с фактами и отношениями на логические и затем в постепенном отказе от жесткой последовательности их выполнения.

Поиск оптимальных расписаний (например, задача коммивояжера – поиск минимальной длины маршрута, не посещая городов больше одного раза или поиск пути с минимальными затратами по дугам графа, содержащим множество вершин, которые нельзя посещать больше одного раза). К этому классу задач относят задачи размещения объектов на определенных условиях. Простые попытки решения (перебором) порождают ***комбинаторный взрыв*** – взрывной рост вариантов решения, которые исчерпывают возможности вычислительной системы. ***Сложность задачи с ростом объема задачи*** может расти линейно, степенным образом (полиномиально), экспоненциально (например, для т. н. NP - полных задач) и даже быстрее - взрывным образом ($\propto 1/(V - V_0)$).

NP ***- задачи*** – задачи с т.н. неопределенными знаниями, не снабженные эвристическими правилами, упрощающими схемами, заставляющие решать методом перебора вариантов.

Невозможность универсальной схемы решателя

Нельзя решать вообще, каждая задача требует своего подхода, своей эвристики, своих предположений и приоритетов относительно выбора решения, которые опираются на собственный опыт и опыт чужой - привлеченный (изученный и осознанный). Отсюда следствие – нельзя придумать общую технологию решения логических задач принципиально. Можно лишь сформулировать процедуры и интегрировать их в методы с ограниченным применением. А затем подумать, как их рационально использовать в конкретном случае.

Экспертные системы с неопределенными знаниями.

Четыре важные проблемы, которые возникают при проектировании и создании ЭС с неопределенными (не точными) знаниями:

• Как количественно выразить степень определенности при установлении истинности (или ложности) некоторой части данных?

• Как выразить степень поддержки заключения конкретной посылкой?

• Как использовать совместно две (или более) посылки, независимо влияющие на заключение ?

• Как быть в ситуации, когда нужно обсудить цепочку вывода для подтверждения заключения в условиях неопределенности ?

Структуры

Глобальная база данных – база данных экспертной системы.

Система продукций – взаимодействие 3 элементов интеллектуальной системы - базы данных, множества правил продукции и системы управления. ***Модульность*** – изменения в базах данных, наборах правил и системе управления могут быть независимыми. Элементами интеллектуальной системы являются базы данных (знаний), позволяющие представлять знания в эффективном виде, обеспечивать ввод, хранение, извлечение этих знаний. В ряде случаев необходимо для ответа на запрос провести некоторые дедуктивные (от общего, то есть, от аксиом и правил, к частному) операции с

фактами из этих баз данных.

Изменение базы знаний (правил продукции) основа обучения в экспертных системах. Это уже активно используют. Для большей гибкости надо бы изменять и саму систему управления – обычно она задана.

Правило продукции – создается на базе обобщения вычислительного формализма; существует предварительное условие, которому база данных удовлетворяет; использование (применение) правила, вообще говоря, меняет базу данных; использование (применение) правила происходит на основе принятия решения системой управления; самостоятельно между собой правила не взаимодействуют (не «вызывают» друг друга). ***Терминальное условие*** – условие остановки.

Пример: *«игра в восемь»*

Таблица 5.1.

2	*8*	*3*	*ИСХОДНАЯ*	*1*	*2*	*3*
1	*6*	*4*	*ЦЕЛЕВАЯ*	*8*		*4*
7		*5*	*ПОЗИЦИЯ*	*7*	*6*	*5*
			ПОЗИЦИЯ			

Элементы задачи – состояния, ходы, цель. Перебор вариантов – 362 880 состояний (9!). Целевое условие – истинность или ложность. Четыре хода – четыре движения пробела. Дополнительные ограничения – минимальное число ходов (минимальная цена).

Procedure PRODUCTION

1. *DATA –исходная база данных*
2. *until DATA – удовлетворяет терминальному условию, do*
3. *begin*
4. *select –выбрать правило П в множестве правил, которые можно применить к DATA это вопрос!*
5. *DATA ←результат применения П к DATA*
6. *end*

Декларативные знания – данные, включающие в себя факты, ***процедурные*** – представленные в правилах, ***управляющие*** – представлены в стратегии управления.

Основная проблема ИИ – основываясь на задаче, организовать декларативную, процедурную и управляющую компоненты для использования в системах продукции .

Представление состояний, ходов и целевых условий. Основная задача - найти наиболее оптимальное представление. Это и наименьшее число (значимых) состояний, и ограничение по возможности возможных ходов, и упрощение процедур управления. С этого, вообще говоря, следует начинать. Хотя оптимизация представлений – больше искусство, чем технологии.

Вычислительные затраты экспертных систем состоят из 1. затраты на применение правил и 2. на управление. Низкая информированность о задаче приводит к небольшим затратам на управление, но зато к большим затратам на перебор правил.

Энтузиазм или квалификация?

У человека, первый случай - это метод «малоосознанного перебора», работает эффективно когда много времени и здоровья у исполнителя, который стремительно перебирает варианты решения. Второй случай - высокая квалификация (на достижение которой потрачено много усилий), но решение находится быстро.

Высокая информированность о задаче приводит к затратной (и очень дорогостоящей) системе управления (с большим объемом памяти и необходимостью громоздких вычислений), но экономит на потерях при применении правил, решение получаются быстро.

Как мы распознаем образы?

Распознавание у человека происходит на основе совместного действия множества различных, не вполне определенных признаков, которые с высокой надежностью дают ответ (причем только все вместе). Обычно в памяти человека и при его наблюдениях имеется конечное число N объектов, сравнение с которыми может быть проведено. Задача формулируется так: как по неполной, неточной информации по M признакам среди N объектов выделить один объект. Вообще говоря, существует ли критерий на уровень неполноты информации и этих значений (признаков и объектов), позволяющий с уверенностью считать, что выбор однозначный. Кстати, китайцы, которые оказались в первый раз среди массы европейцев, полагают, что европейцы все на одно лицо. Не запоминают, точнее не узнают прежде виденного человека [2].

Стратегии управления

Общие подходы [3]. Для систем продукции ***стратегии управления*** - процесс поиска, где правила используются и проверяются пока некоторая их последовательность не порождает базу данных, удовлетворяющую терминальному условию.

Прямая система продукций – от исходного к целевому состоянию. Если можно выделить все состояния и цели, то можно определить глобальную базу данных для описания этих состояний. ***П-правила*** – правила, которые применяются к описаниям состояний для порождения новых состояний. Этот подход полезно применять, если целевых состояний много, а начальное (исходное) состояние одно.

Обратная система продукций. При этом множество описаний целей задачи может служить глобальной базой данных. ***О-правила*** – правила, которые применяются к описаниям целей для порождения подцелей. Обратное движение от целевого к исходному дает возможность найти ***подцелевое*** состояние (состояние от которого до целевого лишь один шаг). Этот подход полезно применять, если целевое состояние одно, а начальных (исходных) состояний много.

Двусторонняя система продукций. Глобальная база данных содержит начальные состояния и целевые состояния. К начальным (исходным) применяют П-правила, к целевым – О-правила.

Коммутативные системы продукций – каждое из множества правил применимо к измененной (при применении одного из правил) базе данных; ежели целевое условие удовлетворяется базой данных, то оно удовлетворяется измененной (при применении одного из правил) базой данных; базы данных полученных после применения последовательности правил, инвариантны при перестановках в этой последовательности. Преимущества этих систем в том, что можно рассматривать лишь один из возможных путей решения для данного набора правил (свобода выбора очередности применения правил). Правила остаются применимыми к новым порожденным базам данных (что в общем случае не так).

Разложимые системы продукций – исходная база данных разбивается на сегменты, которые можно обрабатывать независимо. Процедура разбиения – декомпозиция базы данных. Обязательное требование – аналогичная декомпозиция терминального условия (терминальное условие – конъюнкция терминальных условий для каждого сегмента).

Например, длинная молекула, каждый сегмент (ген) которой может подвергаться изменениям независимо от соседних сегментов.

Методы

Методы - безвозвратный (применение правила без возможности возвращения к начальному состоянию), пробный (правило используется, но резервируется возможность вернуться к начальному состоянию). Два типа пробных режима - с возвращением, когда определена точка возврата; управление с поиском на графе, где запоминается результаты применения нескольких последовательностей правил.

Безвозвратный режим подходит лишь в том случае, если точно известно, что метод решения (поиска решения), основанный на т.н. ***локальной информации,*** апробирован, надежен и со 100% вероятностью приведет к правильном результату (который, вообще говоря, не известен) – т.н. ***глобальной информации (решения)***. Пример применения локальной информации для построения глобального решения – процесс «***подъема на гору***» - движение в сторону «наибольшей крутизны и/или высоты» пока не достигается нужный результат. Для этого нужно иметь специальную функцию с помощью которой осуществляется стратегия выбора правил.

В «игре в восемь» - эта функция может быть числом фишек не на своем окончательном месте. Уменьшая (или, по крайней мере, не увеличивая) эту функцию можно добиться нужного результата. Однако если по пути достигается локальный минимум вместо нужного глобального, то возникают проблемы. В этом случае должны быть дополнительные программы проверки нужного результата и при несоответствии - механизм выведения («вытряхивания») системы из этого локального минимума.

Режим с возвращением (backtracking). Процедура должна завершить работу, если порождаемая база данных удовлетворяет терминальному условию. Список правил, использованных для порождения этой базы данных печатается в конце (10). база данных совпадает с терминальной (окончательной, верной) список пустой (2), если все применимые правила опробованы, но результата нет, то неудача (4). В рекурсивной процедуре нет остановки (но её можно предусмотреть, вводя ограничения на глубину рекурсии), она может порождать все новые базы данных. Упорядочение правил для их последовательного применения может быть основано на разных соображениях, которые придется формализовать в процедуре упорядочения (3).

ПРИМЕР: Recursive procedure **BACKTRACK** (DATA)

1.**if TERM**(DATA), **return** *NIL*; **TERM -** предикат, принимающий значение «истина» для аргументов, удовлетворяющих терминальному условию системы продукций. При успешном срабатывании на выходе дает пустой список *NIL.*

2.**if DEADEND**(DATA), **return** *FAIL*; **DEADEND-** предикат, принимающий значение «истина» для аргументов, о которых известно, что они не встречаются на пути, ведущем к решению. В этом случае на выходе появляется символ *FAIL*

3. *RULES* ←**APPRULES**(DATA); **APPLRULES-** функция, которая определяет правила, применимые к её аргументу, и упорядочивает их произвольно или согласно их эвристическим качествам.

4. LOOP: **if NULL**(*RULES*), **return** *FALL*; Если не имеется применимых правил, то процедура фиксирует неудачу (LOOP: здесь метка).

5.**R**←**FIRST**(*RULES*); Выбирается лучшее из применимых правил.

6.*RULES* ←**TAIL** (*RULES*); Список применимых правил сокращается – из него удаляется только что выбранное правило

7.*RDATA* ← **R**(*DATA*); Применяется правило **R** и порождается новая база данных.

8.*PATH* ← **BACKTRACK** (RDATA); **BACKTRACK** рекурсивно вызывается для новой базы данных.

9.if PATH =*FAIL*, go LOOP; Если рекурсивный вызов неудачен, пробовать новое правило.

10.**return CONS** (*R,PATH*); На выходе список правил, начинающийся с **R** и обеспечивающий успешное решение.

В таких режимах система обычно не запоминает неудачные пути и множества преобразованных (в результате применения правил) баз данных, хотя можно заставить её запомнить все шаги последнего пути и возникающие на этих шагах базы данных.

5.1.2. Байесовская система логического вывода

Использование байесовской системы логического вывода означает, что информация, обрабатываемая экспертной системой, **не является абсолютно точной, а носит вероятностный характер**. Пользователь не обязательно должен быть уверен в абсолютной истинности или ложности свидетельства, он может отвечать на запросы системы с какой-то степенью уверенности. В свою очередь система выдаёт результаты консультации в виде вероятностей наступления исходов.

Вероятность любого события A является неотрицательной, т.е.
Вероятность всех событий выборочного пространства равна 1.

Если все k событий A1, A2, … , Ak являются взаимно независимыми (т.е. не могут подойти одновременно), то вероятность, по крайней мере, одного из этих событий равна сумме отдельных вероятностей.
Дополнение к A, обозначаемое (¬A), содержит совокупность всех событий за исключением A. Т.к. A и ¬A являются взаимонезависимыми (т.е. $A \cup \neg A = W$) то $p(A) + p(\neg A) = p(A \cup \neg A) = p(W) = 1$. Если $B \in \Omega$ некоторое другое событие. ***Тогда вероятность того, что произойдет A при условии, что произошло B записывается в виде p(A | B) и называется условной вероятностью события A при заданном событии B.***

Вероятность того, что оба события A и B произойдут $p(A \cap B)$ называется совместной вероятностью событий A и B. Условная вероятность p(A|B) равна отношению совместной вероятности $p(A \cap B)$ к вероятности события B, при условии, что она не равна 0.

Вероятность того, что оба события A и B произойдут $p(A \cap B)$ называется совместной вероятностью событий A и B. Условная вероятность p(A|B) равна отношению совместной вероятности $p(A \cap B)$ к вероятности события B, при условии, что она не равна 0.

Теория

Аксиомы:

1.Вероятность любого события A является неотрицательной, т.е.

2.Вероятность всех событий выборочного пространства равна 1.

3.Если все k событий A1, A2, … , Ak являются взаимно независимыми (т.е. не могут подойти одновременно), то вероятность, по крайней мере, одного из этих событий равна сумме отдельных вероятностей.

Определения:

DET1. Дополнение к A, обозначаемое (¬A), содержит совокупность всех событий за исключением A. Т.к. A и ¬A являются взаимонезависимыми (т.е. $A \cup \neg A = W$) то $p(A) + p(\neg A) = p(A \cup \neg A) = p(W) = 1$.

DET2. Если $B \in \Omega$ некоторое другое событие. Тогда вероятность того, что произойдет A при условии, что произошло B записывается в виде $p(A \mid B)$ и называется условной вероятностью события A при заданном событии B.

DET3. Вероятность того, что оба события A и B произойдут $p(A \cap B)$ называется совместной вероятностью событий A и B.

Выводы:

1.Условная вероятность $p(A|B)$ равна отношению совместной вероятности $p(A \cap B)$ к вероятности события B, при условии, что она не равна 0.

$p(A|B) * p(B) = p(A \cap B)$ и аналогично $p(B|A) * p(A) = p(B \cap A)$

так как $p(A \cap B) = p(B \cap A)$, то получим соотношение

$$p(A|B) * p(B) = p(B|A) * p(A)$$ **---- теорема Байеса**

2.Для явно непересекающихся событий $B \cap A$ и $B \cap \neg A$ справедливо $B = (B \cap A) \cup (B \cap \neg A)$, тогда

$$p(B) = p((B \cap A) \cup (B \cap \neg A)) = p(B \cap A) + p(B \cap \neg A) = p(B|A) * p(A) + p(B|\neg A) * p(\neg A)$$

ВЫВОД: P апостериорная = Py * P / (Py * P + Pn * (1 – P))

Структура

База знаний представляет собой текстовый файл (который в дальнейшем может быть зашифрован), включающий три секции со следующей структурой:

--- Описание базы знаний, имя автора, комментарий и т.п . (можно в несколько строк, общая длина которых не должна превышать 10000 символов; данная секция заканчивается после первой пустой строки)

Свидетельство № 0 (любой текст (не более 1000 символов), заканчивающийся переносом строки)

Свидетельство № 1

Свидетельство № 2

...

Свидетельство № N (после последнего свидетельства следует одна пустая строка, и вторая секция заканчивается) Исход № 0, P [, i, Py, Pn]

Исход № 1, P [, i, Py, Pn]

Исход № 2, P [, i, Py, Pn]

...

Исход № M, P [, i, Py, Pn]

Смысл первых двух секции вполне понятен. Последняя секция требует более подробного рассмотрения. В ней перечисляются правила вывода: каждое задаётся в отдельной строке; перечисление заканчивается с концом файла.

Правило вывода

В начале описания правила вывода задаётся исход, вероятность которого меняется в соответствии с данным правилом. Это текст, включающий любые символы, кроме запятых. После запятой указывается априорная вероятность данного исхода (P), т.е. вероятность исхода в случае отсутствия дополнительной информации. После этого через запятую идёт ряд повторяющихся полей из трёх элементов . Первый элемент (i) – это номер соответствующего вопроса (симптома, свидетельства). Следующие два элемента (Py = P(E / H) и Pn = P(E / не H)) – соответственно вероятности получения ответа « Да » на этот вопрос, если возможный исход верен и неверен.
эти данные указываются для каждого вопроса, связанного с данным исходом. (Примечание: P <= 0.00001 считается равной нулю, а P >= 0.99999 – единице, поэтому не указывайте такие значения – исход с подобной априорной вероятностью обрабатываться не будет.)

Пример (три вопроса-симптома):
Грипп, 0.01, 1,0.9,0.01, 2,1, 0.01, 3,0,0.01.
Здесь сказано: существует априорная вероятность P(H) = 0.01 того, что любой наугад взятый человек болеет гриппом. Допустим, программа задает вопрос 1 (симптом 1). Тогда мы имеем P(E / H) = 0.9 и P(E / неH) = 0.01, а это означает, что если у пациента грипп, то он в девяти случаях из десяти ответит «Да» на этот вопрос, а если у него нет гриппа, он ответит «Да» лишь в одном случае из ста (т.е. данный симптом встречается довольно редко при других болезнях (исходах)). Очевидно, ответ «Да» подтверждает гипотезу о том, что у него грипп. Ответ «Нет» позволяет предположить, что человек гриппом не болеет. Для второго симптома имеем запись «2,1,0.01». В этом случае P(E / H) = 1, т.е. если у человека грипп, то этот симптом обязательно должен присутствовать. Соответствующий симптом может иметь место и при отсутствии гриппа (P(E / неH) = 0.01), но это маловероятно.
Вопрос 3 исключает грипп при ответе «Да», потому что P(E / H) = 0. Это может быть вопрос вроде следующего: «Наблюдается ли у Вас данное состояние на протяжении большей части жизни?» – или что-нибудь вроде этого.

Значения P(E / H) и P(E / неH), подставленные в теорему Байеса, позволяют вычислить апостериорную вероятность исхода, т.е. вероятность, скорректированную в соответствии с ответом пользователя на данный вопрос:

$$P(H / E) = P(E / H) * P(H) / (P(E / H) * P(H) + P(E / неH) * P(неH))$$

или

P апостериорная = Py * P / (Py * P + Pn * (1 – P))

Вероятность осуществления некой гипотезы H при наличии определенных подтверждающих свидетельств E вычисляется на основе априорной вероятности этой гипотезы (без подтверждающих свидетельств и вероятностей осуществления свидетельств при условиях, что гипотеза верна или неверна).

Пример актуальной проблемы: Событие: человек оглянулся. Результат: влюбился.

5.1.3. Простой логический вывод

Пример простейшего логического вывода.
База знаний (состоит из двух правил):
Правило1: **Если** "отдых - летом" **и** "человек - активный", **то** "ехать в горы",

$$(отдых\_летом \wedge челов\_акт \rightarrow ехать\_в\_гор)$$

Правило2: **Если** "любит солнце", **то** "отдых летом",

$$(любит\_солн \rightarrow отдых\_летом)$$

Входные данные - "человек активный" и "любит "солнце"

Итак: Правила – это база знаний. База данных содержит факты.
Правило состоит из фактов (литералов) и связок.

Факты в составе правил могут находится в базе данных частично или все. Если они все уже находятся в базе данных, то остановка программы. Это условие – терминальное.
Первое правило сложное, ибо там три литерала, второе – простое, только два литерала.
Рассмотрим действия, изображенные в таблице 4.2.

Таблица 5.2.

Прямой вывод	Обратный вывод
1-й проход. Шаг 1. Пробуем Правило1 - не работает (не хватает данных "отдых - летом"). Шаг 2. Пробуем П2 - работает, в базу поступает факт "отдых - летом". 2-й проход. Шаг 3. Пробуем Правило1 - работает, активируется цель "ехать в горы", которая и выступает как совет, который дает ЭС.	1-й проход. Шаг 1. Цель - "ехать в горы": пробуем Правило1 - данных, "отдых - летом" нет, они становятся новой целью, и ищется правило, где она в правой части. Шаг 2. Цель "отдых - летом": Правило2 подтверждает цель и активирует ее. 2-й проход. Шаг 3. Пробуем П1, подтверждается искомая цель.

Прямой вывод – рассматривается сначала левая часть (посылка) . Если посылка среди фактов, то правило срабатывает и цель (вывод) этого правила, то есть новый факт, добавляется в базу данных. Если в посылке не все факты среди фактов базы данных, правило гне срабатывает и программа переходит ко второму правилу. После использования всех правил (первый проход) база данных модифицируется (в этом случае добавляются новый факт). Второй проход с обновленной базой данных позволяет удовлетворить первому правилу и вывод-факт добавляется в базу данных. Дальше машина может: 1) проверить все задействованные правила, при этом изменений в базе данных нет или 2) все факты в структуре правил оказались в базе данных или 3) еще как-нибудь (?), но как-то надо задать терминальное условие, которое оказывается удовлетворенным и программа останавливается.

Обратный вывод. Стартуют от выводов правил (то есть, целей). Проверяют первую цель (правую часть первого правила.). в левой части два литерала- факта. Один есть в базе данных, в ним все в порядке, а второго - нет. Он активизируется и ищется другое правило базы знаний, где он находится в правой части. Находим такое правило, проверяем левую часть - есть ли она в базе данных. Если есть, то теперь начальная цель получает право на жизнь, её включают в базу данных. При этом в базу данных могут включать и все промежуточные цели этого удачного решения. Проверка всех задействованных правил дает их выполнение и изменений базы данных уже нет – это может служить терминальным условием- условием остановки программы.

Обратный сценарий более конструктивен, ибо если в прямом реализован перебор правил, то во втором формируется дерево целей, правила применяются избирательно. Перебор может увеличивать время решения, формирование дерева сокращает время решения.

Кстати, в обратном выводе тоже можно организовать перебор, аналогичный рассмотренному прямому методу, а в прямом построить схему по типу формирования дерева. Это может быть домашним заданием.

5.1.4. Реляционная база данных[4]

Данные о сущности хранятся в отношениях. Отношение содержит несколько кортежей (строк таблицы). Кортеж состоит (заполняется) из атрибутов. Множество

[4] см. подробнее предыдущий раздел 4.

кортежей данного отношения называют телом отношения. Атрибут – свойство, характеризующее сущность. Набор доменов определяет отношение, а каждый домен – множество возможных значений одного атрибута. Совокупность атрибутов, которые однозначно определяют каждый из кортежей отношения – ключ отношения.

ПРИМЕР:

Таблица 5.3

ФИО	Отдел	Должность	Дата рождения
Волк А.Б.	1	мастер	10 01 70
Заяц В.Г.	2	рабочий	20 02 80
Охотник Д.Е.	3	начальник	30 03 90

Таблица 5.3. в целом – это отношение, первая строка (строка заголовков) – схема отношения, 2-4 строки – кортежи, заголовок столбца – атрибут, запись в ячейке - значение атрибута, домен - набор всех значений атрибута, ключ - набор атрибутов, идентифицирующий кортеж (данную строку). Связывание отношений. Рассмотрим два отношения: ОТДЕЛЫ (**код отдела**, название, число сотрудников); СОТРУДНИКИ (**код сотрудника**, ФИО, *код отдела*, зарплата). *Атрибуты, выделенные жирным шрифтом – внутренние первичные ключи, а курсивом – внешний ключ отношения СОТРУДНИКИ (является первичным ключом отношения ОТДЕЛЫ). Ключи можно индексировать - система индексации фактически автономная база данных. Можно получать новые отношения (вычисляемые) из старых (хранимых). Функциональная зависимость одного (составного) атрибута B от другого (составного) атрибута A состоит в том, что каждому значению A соответствует в точности одно значение B* ($A \to B$) *. Транзитивная зависимость атрибутов A,B,C - если* $A \to B$, *а также* $B \to C$ *, но обратные зависимости, вообще говоря, отсутствуют.*

1.Первичные ключи могут быть индексированы индексами разного ранга, что позволяет формировать множество автономных баз данных.
2.Внешние ключи позволяют связывать различные элементы из разных баз данных.

5.1.5.Решение задач роботом

Решение задач роботом - некоммутативная система продукций.
ПРИМЕР: Игра в кубики. Описание задачи.
Описывают состояния мира : предикаты *CLEAR* – не занята верхняя поверхность кубика, *ON* –находится на кубике, *ONTABLE* - находится на столе.
Описывают состояния руки робота: предикаты *HANDEMPTY* – рука пуста, *HOLDING* - рука держит кубик.
Описывают действия – **pickup** – взять со стола, **putdown** –поставить на стол, **stack** – поставить на другой кубик, **unstuck** – снять с другого кубика.
Описание данного состояния: *CLEAR(B) ; CLEAR(C) ; ON(C,A); HANDEMPTY; ONTABLE(A); ONTABLE(B) – кубики A и B лежат на столе, кубик C – на кубике A.*
Описание целей: *ON (B,C)* $\wedge$ *ON(A,B) – кубик A находится на кубике B, а кубик B, в свою очередь, на кубике C.*

Действия робота. При действиях робота П-правила должны допускать вычеркивание выражений, которые могут перестать быть справедливыми. П-правила (систем типа STRIPS) явно указывают ***списки вычеркивания***. Напомним, что к начальным (исходным) состояниям применяют П-правила, к целевым – О-правила.

Применение правила – моделирование реального действия.

Форма П-правила (в STRIPS) состоит из (1) ***формулы предварительного условия***, подобного условию «если» («если» = антецедент, «то» = консеквент) в импликации, должно следовать из фактов в описании состояния, форма – конъюнкция литералов,

переменные относятся к квантору существования. ***Подстановка соответствия*** - набор унификаторов, который обеспечивает ***соответствие*** П-правил фактам (таких подстановок соответствия может быть несколько). (2) Список вычеркивания (список литералов возможно со свободными переменными) – вторая часть П-правила. Подстановка соответствия применяется к списку вычеркивания и полученные основные частные случаи (когда все переменные заменены на константы) вычеркиваются из описания прежнего состояния. (3) ***Формула добавлений*** (состоит из конъюнкции литералов) – третья часть П-правила., - подобна части «то» в импликации. Подстановка соответствия применяется к этой формуле добавлений и результирующий частный случай добавляется в прежнему описанию состояния.

ПРИМЕР: Модель действия: взятие кубика со стола

Предусловия – кубик на столе, рука робота не занята, сверху на кубике ничего нет. Результат – рука держит кубик.

pickup *(x)*

P (precondition – предусловие): ONTABLE (x) $\wedge$ HANDEMPTY $\wedge$ CLEAR (x).

D (delete list - список вычеркиваний): ONTABLE (x), HANDEMPTY , CLEAR (x).

A (add formula - формула добавлений): HOLDING (x).

Подстановка соответствия в этом случае дает - **pickup** *(x)* может быть применено только если *x = B*. Тогда новое описание состояния: *CLEAR (C), ON(C,A), ONTABLE (A), HOLDIBG (B).* В формулу добавлений можно включать отрицания вычеркиваемых литералов.

Обычные условия – все формулы являются конъюнкциями литералов, поэтому формулы предусловий и добавлений можно представить в виде списка литералов.

Проблема фрейма – какие ППФ описания состояния следует изменять, а какие – нет. Существует состояние мира (фон) и моделируемые действия (носят локальный характер). При более точном рассмотрении надо учитывать изменения фона, то есть существует иерархия планов (действий), которые учитывают связи между компонентами мира, активируемые данным простым действием. Существуют также трудности описания и правил (действий) в случае аномальных условий (которых может оказаться слишком много).

ПРИМЕР: Описание действий: брать и ставить кубик на стол, брать и ставить один кубик на другой, снимать поставленный кубик с нижнего. Здесь формула предусловия и список вычеркивания совпадают.

1. **pickup** *(x)*
   P&D: ONTABLE (x), CLEAR (x), HANDEMPTY
   A: HOLDING (x)
2. **putdown** *(x)*
   P&D: HOLDING (x)
   A: ONTABLE (x), CLEAR (x), HANDEMPTY
3. **stack** *(x,y)*
   P&D: HOLDING (x), CLEAR (y)
   A: HANDEMPTY, ON(x,y), CLEAR (x)
4. **unstuck**
   P&D: HANDEMPTY, CLEAR (x), ON(x,y)
   A: HOLDING (x), CLEAR (y)

Внутренний мир интеллектуальных систем

Все обсуждаемые системы накопления, хранения и использования знания, как, впрочем, и человеческий разум содержат свой мир, где все операции, образы и действия заложены ранее. Вызов информации в значительной степени связан с характером его внесения в этот мир и часто невольно выполняется в обратном порядке, хотя возможны исключения. Ассоциативность появления информации, выделения образов и проведения действий в разных интеллектуальных системах (лучше использовать этот термин для систем логического вывода, нежели

определение «искусственный интеллект») различна, но имеет место везде, что позволяет говорить о наличии элементов эвристичности. Хотя это вовсе не ограничивает нас в применении иных схем подобного рода.

5.2.Элементы формальной логики

5.2.1. Силлогизм
5.2.1. Силлогизм[5]

Силлогизм - это вид логических рассуждений, впервые описанный Аристотелем, при котором "... если определенные вещи установлены, то некоторые другие неизбежно следуют из справедливости первых".
Суждение = квантор + субъект + связка → предикат
Силлогизм = правила получения суждений на основе четырех допустимых форм (логика классов, но не содержит дополнения классов):
все* S есть P
никакой* из S не есть P
некоторые** из S являются P
некоторые из S не являются P
(*универсальные и **экзистенциальный кванторы, есть еще ∀, ∃).

Высказывания – обобщение суждения, которому можно приписать значения истина или ложь.

modus ponendo ponens P→Q, if P- tru, then Q - tru
modus tollendo tollens P→Q, if P- fal, then Q -fal
modus ponendo tollens P ≠ Q, if P- tru, , then Q -fal
modus tollendo ponens P or Q – tru, if P- fal, then Q – tru (дизъюнктивный силлогизм)

modus – правило, мера.

5.2.2. Начала формальной логики

Булева алгебра[6] - новая форма логики, где выполнялись : коммутативность, ассоциативность и дистрибутивность для множеств и истинностных значений, где знак суммирования + это «или», знак умножения * - это «и».
Новое, что было введено в традиционную логику:
Дополнение - унарная операция с множеством или истинностным значением.
Правила де Моргана

$$\sim(x+y)= \sim x * \sim y \ , \sim(x*y)= \sim x + \sim y.$$

Импликация (следование)
if x then y , а для истинностных значений[7] можно переписать ~x+y.

5.2.3. Логика высказываний

Дальнейшее развитие формальной логики, где вводились новые понятия:

---

[5] По тонкому замечанию Клея Ширки, силлогизмы «описывают мир гораздо более простой, нежели наш с вами» ибо для применения силлогизмов «нужен мир, где язык - это просто набор математических операций над словами».
[6] Из импликации и отрицания можно вывести все соединители (Фреже).
[7] Если x ложно, то всегда результат истина (!).

Правильно построенные формулы (ППФ) обозначают пропозициональными переменными p, q, r. Следующие ф-лы тоже ППФ: ~p, p&q, pVq, p→q, p←→q. Здесь &- «и», V – «или».
Истина – единица, ***ложь*** – нуль. Тогда & имеет смысл умножения, V – сложения. Импликация p→q соответствует ~pVq, эквивалентность p←→q (p&q)V(~p&~q).
Тавтология = тождественно истинна = AV ~A;
Непоследовательная ППФ = противоречие = тождественно ложна = A&~A.
Теория – полное множество атомарных ППФ для данной предметной области, каждая такая ППФ – ***аксиома***.
Модель – интерпретация, при которой каждая аксиома истинна.
Доказательство того, что данная ППФ является следствием аксиом теории. Аксиомы считаются истинными для всех отдельных интерпретаций.

Методы логических построений

1.Семантический метод - Нужно показать, что при всех интерпретациях данная ППФ – истина. Если есть модель, при которой ППФ будет ложной, то ППФ – не следствие теории.
2.Синтаксический метод – ищут полное (если можно получить любую тавтологию, скомбинированную из пропозициональных обозначений) множество правил вывода. Аксиомы = предпосылки, результат употребления правил – заключение. Заключение в конце всех шагов- данная ППФ. Можно использовать т.н. аксиоматические схемы и одно (два или больше) правило, например, modus ponens.
3.Алгоритм - позволяет ответить на вопрос, является ли данная последовательность шагов доказательством того, что ППФ – следствие заданных аксиом. Вопрос является ли ППФ следствием аксиом, алгоритмически разрешим только если ППФ является таким следствием (***исчисление высказываний*** - полное множество правил вывода).

Новые понятия:
Формальный язык – объектный язык.
Метаязык описывает все детали и особенности применения объектного языка (часто не формализованные). Аналитически - выделение тавтологий, противоречий, полноты; предписывающее - установление истинностных значений.
Непротиворечивость (синтаксическая последовательность) ***теории*** – из аксиом нельзя вывести противоречие.
Полнота теории – каждую истинную ППФ можно доказать на основании аксиом теории.
Рекурсивное определение множества отношений – соблюдение отношения в одном случае опирается на соблюдение отношения в предыдущем случае.
Рекурсивный алгоритм – правила вывода и рекурсивные аксиомы.
Рекурсивная перечислимость – образования множества путем рекурсивного алгоритма.
Процедура опровержения – (доказательство от противного) – добавление в теорию отрицания ППФ = A, то есть ~A, что переводит эту теорию в непоследовательную, то есть следует довести доказательство вплоть до A&~A.

5.3.Сущность языка - исчисления предикатов

5.3.1. Основная терминология

Синтаксис – алфавит символов и допустимые выражения (в исчислении предикатов – это правильно построенные формулы - ППФ).
Символы – отношений (предикатные), переменных, констант, функциональные.
Атомные формулы – осмысленные выражения, состоят из предикатных символов и термов.

СУПРУЖЕСТВО[отец(*ДЖОН*), мать(*ДЖО*]*Н*)

Термы – константы, переменные, функции.

Связки $\wedge$ (и), $\vee$ (или) $\Rightarrow$(***импликация***).

Отрицание *НЕ.*

**)Напомним, что , НЕ F1* $\vee$ *F2 имеет то же значение истинности, что и F1* $\Rightarrow$ *F2*

Конъюнкция (состоит из конъюнктов)

ЖИВЕТ(ДЖОН, ДОМ1) $\wedge$ *ЦВЕТ(ДОМ1, ЖЕЛТЫЙ)* *Джон живет в желтом доме*

Дизъюнкция (состоит из дизъюнктов)

ЕСТ(ДЖОН, КОЛБАСА) $\vee$ *ЕСТ(ДЖОН, СЫР)* *Джон есть или колбасу или сыр*

Импликация (если, …то, если = антецедент, то = консеквент)

ВЛАДЕТЬ (ДЖОН, МАШИНА1) $\Rightarrow$ *ЦВЕТ(МАШИНА1, ЗЕЛЕНЫЙ)* *Если машина принадлежит Джону, то она - зеленая.*

Таблица 5.4.

X1	***X2***	***X1*** $\vee$ ***X2***	***НЕ X1*** $\vee$ ***X2***	***X1*** $\wedge$ ***X2***	***X1*** $\Rightarrow$ ***X2***	***НЕ X1***
T	***T***	***T***	***T***	***T***	***T***	***F***
F	***T***	***T***	***T***	***F***	***T***	***T***
T	***F***	***T***	***F***	***F***	***F***	***F***
F	***F***	***F***	***T***	***F***	***T***	***T***

Кванторы - $\exists$ существования, $\forall$ общности.

$\forall x$ [*СЛОН (x),* $\Rightarrow$ *ЦВЕТ*(*x, СЕРЫЙ*] *Все слоны серые*

Предикаты первого порядка – не допускается квантификации по предикатным и функциональным символам.

Правила вывода 1. создание *A2* из *A1* и *A1* $\Rightarrow$ *A2 (модус поненс*)

2. создание *Ф(А) из (*$\forall x$*) Ф(x) (специализация)*

Унификация – поиск подстановок термов на место переменных. Алфавитный случай - замена одной переменной на другую.

Основной частный случай – подстановка вместо переменных - констант. Обозначение $s = \{t / a\}$ - подстановка *s:* вместо переменной *t* подставлена постоянная *a*. Сравнение унифицированных литералов часто называют «сравнением с образцом».

Удовлетворимость – если каждая ППФ из множества имеет значение Т при одной и той же интерпретации, то эта интерпретация удовлетворяет этому множеству ППФ.

Описание состояния - описание конкретной ситуации («описание мира»). Любая конечная конъюнкция формул описывает семейство (конкретизируемых за счет добавления новых формул, то есть уточнения) состояний, каждое состояние может рассматриваться как интерпретация.

Интерпретация – 1.Сопоставление высказываниям и предикатам значения истинности или ложности. 2. Предписывает соответствие между элементами языка, отношениями, элементами и функциями в указанной области рассуждения (определяет семантику языка исчисления предикатов), тогда можно приписать некоторым значениям формул значения Т или F.

Система дедукции = доказательство теорем – показывает, что данная ППФ (целевая ППФ) является теоремой, выводимой из множества формул (описания состояния).

Неудовлетворимое множество ППФ – множество ППФ, которое не может удовлетворяться при любой интерпретации.

Выполнимость – если для всех возможных интерпретаций ППФ имеет значение Т (истина).

Тавтология – выполненная ППФ.

Полуразрешимость исчисления предикатов – при применении кванторов не всегда

ППФ выполняются. Нет универсального метода установления факта выполненности квантифицированных выражений. Но существует процедура установления выполненности, если изначально известно, что ППФ выполнима.

Логическое следование – если данная ППФ следует из множества М других ППФ в каждой из интерпретаций, которые удовлетворяют и М, и данной ППФ.

Логичность системы правил вывода (sound) – если каждая теорема, выводимая из множества ППФ логически следует из этого множества.

Полнота системы правил вывода – все ППФ, которые логически следуют из какого-либо множества ППФ, являются также теоремами, выводимыми из этого множества.

Необходимость расширения базы знаний

Прогнозирование, как форма решения задач, во всех интеллектуальных системах, включая естественный разум, опирается лишь на созданный прежде внутренний мир образов, событий и представлений. Интеллектуальные системы заперты в этом мире и не способны выйти из него в своих решениях и прогнозах. Единственным способом вырваться из созданного внутреннего мира может быть только получение дополнительной информации о мире внешнем. То есть, возможность развития связана с созданием эффективных систем мониторинга внешнего окружения. Для развития искусственных интеллектуальных систем нужно заметно большее внимание уделять интерактивным структурам, обеспечивающим связь между базой знаний и данных (то есть, глобальной базой данных) с одной стороны, и внешним окружением, экспертной средой с другой стороны.

Предложение (clause) – ППФ, состоящая из дизъюнкции литералов: $A \vee B \vee C$.

Резолюции процесс – применяется к двум предложениям и порождает выведенное предложение в исчислении предикатов.

5.3.2. Процедуры преобразования ППФ в исчислении предикатов [3]

Рассмотрим технику преобразований правильно построенных формул в набор простых выражений для организации коммутативной базы данных.

$$(\forall x)\{P(x) \Rightarrow \{(\forall y)[P(y) \Rightarrow P(f(x,y))] \wedge \sim(\forall y)[Q(x,y) \Rightarrow P(y)]\}\}$$

-------- ***исходная ППФ***

1.***Исключение символов импликации*** ($\sim X1 \vee X2$ *вместо* $X1 \Rightarrow X2$)

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge \sim(\forall y)[\sim Q(x,y) \vee P(y)]\}\}$$

2.***Ограничение области действия отрицания*** (применение не более чем к одной атомной формуле). Используем ***законы Моргана*** ($\sim(X1 \vee X2)$ *эквивалентно* $\sim X1 \wedge \sim X2$; $\sim(X1 \wedge X2)$ *эквивалентно* $\sim X1 \vee \sim X2$), ***эквивалентности выражений с кванторами*** ($\sim(\exists x)P(x)$ *эквивалентно* $(\forall x)[\sim P(x)]$; $\sim(\forall x)[P(x)]$ *эквивалентно* $(\exists x)[\sim P(x)]$) и другие эквивалентности.

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge (\exists y)[Q(x,y) \wedge \sim P(y)]\}\}$$

3.***Разделение переменных.*** В пределах области действия квантора переменная, связываемая этим квантором, представляет собой немую переменную. Ее повсюду можно заменить любой другой (не встречающейся) переменной в пределах области действия квантора, при этом значение истинности этой ППФ не изменится. Стандартизация

переменных в пределах ППФ означает переименование немых переменных с той целью, чтобы каждый квантор имел свою, свойственную только ему, немую переменную. Так, вместо $(\forall x)[P(x) \Rightarrow (\exists x)Q(x)]$ получим $(\forall x)[P(x) \Rightarrow (\exists y)Q(y)]$.

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge (\exists w)[Q(x,w) \wedge \sim P(w)]\}\}$$

4.***Исключение кванторов существования.*** Можно перейти от $(\forall y)[(\exists x)P(x,y)]$ к $(\forall y)[P(g(y),y)]$ с помощью, так называемой, сколемовской функции g(y). Это возможно, если область определения (действия) квантора существования и область значений скалемовской функции (которая к тому же должна существовать) совпадают. В случае, если квантор существования не входит в область действия кванторов общества, то применяют сколемовскую функцию без аргументов, то есть константу A: $(\exists x)P(x)$ заменяется на $P(A)$.

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge [Q(x,g(x)) \wedge \sim P(g(x))]\}\}$$

5.***Преобразование в префиксную форму.*** Перенос квантора общности в начало формулы (тогда она будет состоять из префикса = цепочки кванторов и бесквантворной формулы = матрицы).

$$(\forall x)(\forall y)\{\sim P(x) \vee \{[\sim P(y) \vee P(f(x,y))] \wedge [Q(x,g(x)) \wedge \sim P(g(x))]\}\}$$

6.***Приведение к конъюнктивной нормальной форме.*** Любая матрица может быть записана как конъюнкция конечного множества дизъюнкций литералов. Используются, например, эквивалентности $A \vee (B \wedge C) = A \vee B \wedge A \vee C$, а также $A \wedge (B \vee C) = A \wedge B \vee A \wedge C$.

$$(\forall x)(\forall y)\{[\sim P(x) \vee \sim P(y) \vee P(f(x,y))] \wedge [\sim P(x) \vee Q(x,g(x))] \wedge [\sim P(x) \vee \sim P(g(x))]\}$$

7.***Исключение кванторов общности.*** Предположение, что можно не показывать явно кванторы общности приводит к упрощению. Тогда все будет представлено в точное соответствие с конъюнктивной нормальной формой.

$$[\sim P(x) \vee \sim P(y) \vee P(f(x,y))] \wedge [\sim P(x) \vee Q(x,g(x))] \wedge [\sim P(x) \vee \sim P(g(x))]$$

8.***Исключение символов*** $\wedge$**.** Так как матрица представлена в конъюнктивной нормальной форме, то можно освободиться от явного присутствия символов конъюнкции $\wedge$. Запись множеством ППФ, которые есть дизъюнкции литералов, то есть множеством предложений.

$$\sim P(x) \vee \sim P(y) \vee P(f(x,y))$$
$$\sim P(x) \vee Q(x,g(x))$$
$$\sim P(x) \vee \sim P(g(x))$$

9.***Переименование переменных.*** Переменные должны быть разными в различных переменных списка.

$$\sim P(x1) \vee \sim P(y) \vee P(f(x1,y))$$
$$\sim P(x2) \vee Q(x2,g(x2))$$
$$\sim P(x3) \vee \sim P(g(x3))$$

10. ***Резолюция.*** Резолюция двух предложений (одной и той же константой или переменной) получается дизъюнкцией двух взаимно противоположных литералов P и ~P. Например, $R \vee P$ и $\sim P \vee Q$ получим $R \vee P \vee \sim P \vee Q$, удалим $P \vee \sim P$ (тавтологию) и окончательно резольвента - это $R \vee Q$. Но только если область определения переменных является общей для предложений.

Примеры: простейшие резолюции

Таблица 5.5.

ИСХОДНЫЕ ПРЕДЛОЖЕНИЯ	РЕЗОЛЬВЕНТЫ	ОБСУЖДЕНИЕ
P и $\sim P \vee Q$ $(P \Rightarrow Q)$	Q	Модус поненс
$P \vee Q$ и $\sim P \vee Q$	Q	$Q \vee Q$ это Q т.н. слияние
$P \vee Q$ и $\sim P \vee \sim Q$	$Q \vee \sim Q$, $P \vee \sim P$	Две тавтологии
P и ~P	*NIL*	Пустое множество
$\sim P \vee Q$ $(P \Rightarrow Q)$ $\sim Q \vee R$ $(Q \Rightarrow R)$	$\sim P \vee R$ $(P \Rightarrow R)$	Цепочка

Общий случай резолюций - если в составе двух предложений с переменными выделятся возможные при некоторой подстановке (например, $x = y$) дополнительные литералы (то есть, *P(y)* и *~P(x)* превращаются в *P(x)* и *~P(x)*), то в рамках этой подстановки можно применить правило резолюции и получить резольвенту.

5.3.3.Доказательства теорем и методы получения решений

Применение исчисления предикатов – (**1**) все выражения базы данных преобразовываются в предложения в нормальной форме – ППФ; (**2**) система управления на основе применения резолюций (причем берется отрицание целевой ППФ и доказывается противоречие, целевая ППФ – теорема, которую нужно доказать); (**3**) система продукций является коммутативной, что позволяет применять безвозвратный режим управления (где порядок применения не имеет значения, а повторы не опасны, проблема лишь в сокращении времени решения).

Стратегии выбора применяемых предложений - формирование дерева выбора – вспомогательного графа; поиска в ширину (полного перебора) и т.д.

Упрощения –

(**1**) исключение тавтологий,

(**2**) оценка отдельных литералов (для основных частных случаев) за счет т.н. присоединенных процедур (если литерал получает оценку *T*, то такое предложение исключают, а если F, то можно его оставить, но исключить этот литерал),

(**3**) выявление включения одного из предложений в состав второго при определенной подстановке переменных (то есть, первое предложение до подстановки является более общим видом соответствующей части второго), при этом говорят о том, что первое предложение ***подсуммирует*** второе (тогда эту часть второго предложения можно исключить, ибо исключаемая часть не влияет на неудовлетворимость оставшейся части).

Конструктивные доказательства – Поиск и определение значений (частных случаев) переменных в формулах при доказательстве теорем в исчислении предикатов. То есть, недостаточно доказать существование решения в области определения переменных, нужно найти конкретные значения этих переменных (само решение).

ПРОЦЕСС ИЗВЛЕЧЕНИЯ ИЗ ФОРМАЛЬНОГО РЕШЕНИЯ (ЗДЕСЬ ЭТО ОПРОВЕРЖЕНИЕ НА ОСНОВЕ РЕЗОЛЮЦИИ) ЗНАЧЕНИЯ ПЕРЕМЕННОЙ, ОТНОСЯЩЕЙСЯ К КВАНТОРУ СУЩЕСТВОВАНИЯ.

Пример 1 . Задача: Если Жора ходит в те же самые места, куда ходит Коля, а Коля в школе, то где же Жора?

S : $\forall(x)[B(KOLIA,x) \Rightarrow B(GORA,x)];\ B(KOLIA,SCHOOL)$.

На основной вопрос задачи «Где Жора?» можно ответить, если существует решение, иначе говоря, $(\exists x)B(GORA,x))$ есть следствием ППФ начальной базы знаний (***аксиом***). Здесь $(\exists x)B(GORA,x))$ - целевая ППФ. Отрицание целевой ППФ - *(∀x)[~B(GORA,x)]* . Поэтому выясним, есть ли решение.

Опровержение на основе резолюции:

Таблица 5.6.

	$\sim B(KOLIA,y) \vee B(GORA,y)$ аксиома 1	$B(KOLIA,SCHOOL)$ аксиома 2
~ B(GORA, x) отрицание целевой ППФ	$\sim B(KOLIA,x)$	*NIL*

Решение существует, но где все же Жора? Добавим к каждому предложению, возникающему из отрицания целевой ППФ его собственное отрицание (получим тавтологии), и, выполнив ***те же самые резолюции,*** получим в корневой вершине ответ.

Опровержение на основе резолюции:

Таблица 5.7.

	$\sim B(KOLIA,y) \vee B(GORA,y)$	$B(KOLIA,SCHOOL)$
~ B(GORA, x) ∨ B(GORA, x) начальная тавтология	$\sim B(KOLIA,x) \vee B(GORA,x)$	$B(GORA,SCHOOL)$

Итак – это преобразование дерева опровержения (с пустым множеством в корневой вершине) в дерево доказательства с некоторым утверждением в корневой вершине, которое может служить ответом.

Некоторые тонкости (know-how)

1.Как обойти квантор существования?

Пример 2. Задача. Для всех x и y , если x является родителем y, а y родителем z , то x является дедушкой или бабушкой z. У каждого есть родитель.

S: $(\forall x)(\forall y)\{[P(x,y) \wedge P(y,z)] \Rightarrow G(x,z)\}$ - аксиома 1 ; $(\forall y)(\exists x)P(x,y)$ - аксиома 2.

Целевая ППФ: $(\exists x)(\exists y)G(x,y)$ - существуют такие лица *x* и *y,* что *x* – дедушка или бабушка *y.*

Дерево опровержения.

Таблица 5.8.

	$\sim P(x,y) \vee \sim P(y,z) \vee G(x,z)$	$P(f(w),w)$	$P(f(w),w)$
~ G(u, v)	$\sim P(u,v) \vee \sim P(y,v)$	$\sim P(u,f(v))$	***NIL***

Модификация дерева опровержения.

Таблица 5.9.

	$\sim P(x,y)\vee \sim P(y,z)\vee G(x,$	$P(f(w),w)$	$P(f(w),w)$
$\sim \mathrm{G(u,v)}\vee \mathrm{G(u,v)}$	$\sim P(u,v)\vee \sim P(y,v)\vee G(u,v)$	$\sim P(u,f(v))\vee G(u,v)$	$G(f(f(v)),v)$

Сколемовская функция $f(x)$, которая вводится для исключения квантора существования в аксиоме 2, причем она указывает имя родителя для своего аргумента.

2. В процессе унификации переменные заменяются на другие и могут возникнуть ситуации, когда размещенные в определенном порядке переменные могут оказаться функциями самих себя. Но это проблема постановки задачи или надо искать смысл в таком ответе.

3.При решении может быть несколько опровержений и несколько ответов, при этом один может быть достаточно общим. Но заранее нет способа выяснить является ли данных ответ подходящим или общим.

4.Если в процессе унификации переменные заменяются на константы и эти константы никак не определены, то можно снова в ответе перейти к переменным.

5.3.4.Практические методы логического вывода

Стратегии управления для метода резолюции. В этом разделе рассмотрены случаи использования глобальной базы данных, представленной только в форме ППФ, где исключена импликация (утверждение). При этом все предложения относятся к одному классу, невозможно отделить правила и факты. Это, вообще говоря, имеет свои достоинства - все равно с какого предложения начинать процесс вывода, - но и недостатки – эффективность системы оказывается ниже, теряются некоторые эвристические составляющие, свойственные импликации.

Рассмотрим стратегии поиска решения на примере следующей задачи:

Дано

1.Кто читает- грамотный $(\forall x)$ [Ч (x) $\Rightarrow$ Г(x)]

2.Дельфины не грамотны $(\forall x)$ [Д (x) $\Rightarrow \sim$ Г(x)]

3.Некоторые дельфины обладают интеллектом $(\exists x)$ [Д (x) $\wedge$ И(x)]

Надо доказать,

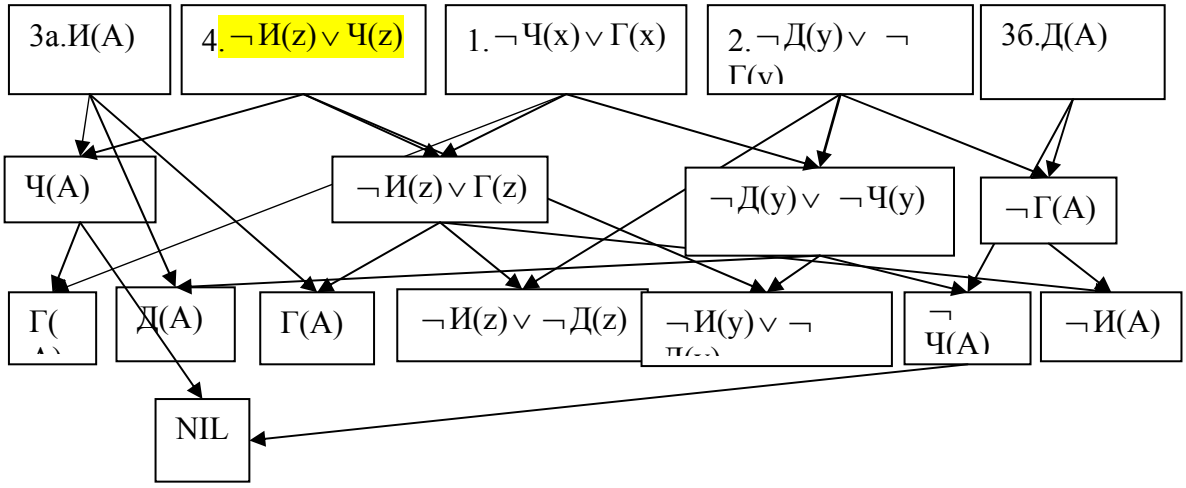
4.Некоторые из тех, кто обладает интеллектом, не могут читать $(\exists x)$ [И (x) $\wedge \sim$ Ч(x)]

Пусть x=A. Вместо Д (A) $\wedge$ И(A) можно записать Д (A) и И(A),

Отрицание целевой ППФ здесь $\neg$ И(z) $\vee$ Ч(z) - тогда все сводится к доказательству теоремы, и отысканию пустого множества на определенной глубине поиска.

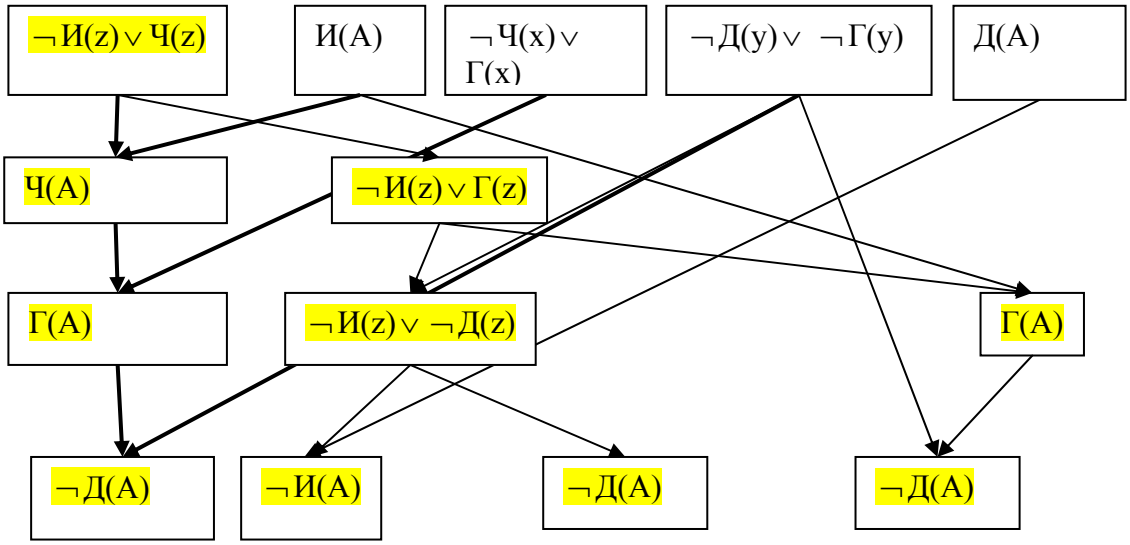
> В логике имеет место двойственное отношение между утверждениями (фактами) и целями в системах доказательства теорем. В системах опровержения на основе резолюции целевые ППФ заменялись их отрицанием и преобразовывались в форму предложений, то есть в ту же форму, что и факты. То есть, отрицания цели можно было представить как факты.

Стратегия поиска в ширину. Сначала вычисляют все резольвенты первого уровня (верхняя строка предложений - базовое множество, из которого формируются резольвенты первого уровня), затем резольвенты второго уровня (которые формируются из резольвент первого уровня) и т.д. Стратегия полная и потому неэффективная.



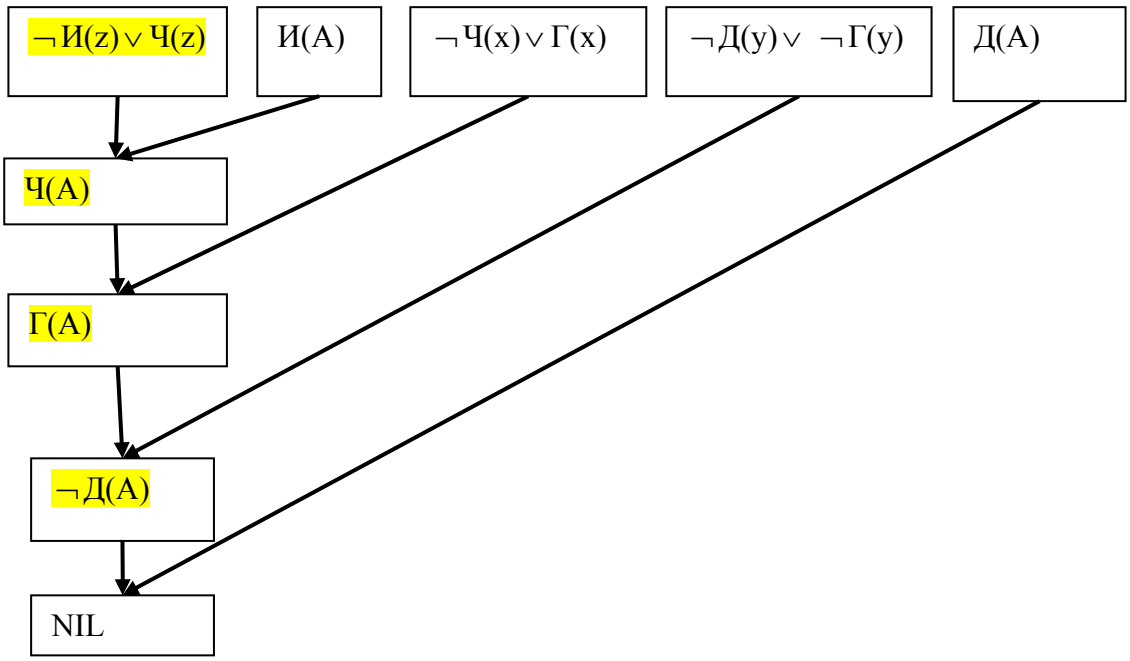
.К стратегии поиска в ширину

Стратегия опорного множества. Опорное множество – это предложения, полученные отрицанием целевой ППФ или их потомков. Опорное множество создает меньше предложений в сравнении с неограниченной резолюцией в ширину (см. выше). Это позволяет ослабить комбинаторный взрыв. Однако такая стратегия может увеличить глубину поиска пустого множества. На рисунке выше (иллюстрирующем неограниченную резолюцию в ширину) количество предложений на втором уровне заметно больше, чем на втором уровне см. рис. ниже (отвечает стратегии опорного множества). Но если пустое множество возникает на верхнем рисунке на третьем уровне, то на подобном уровне нижнего рисунка его еще нет.



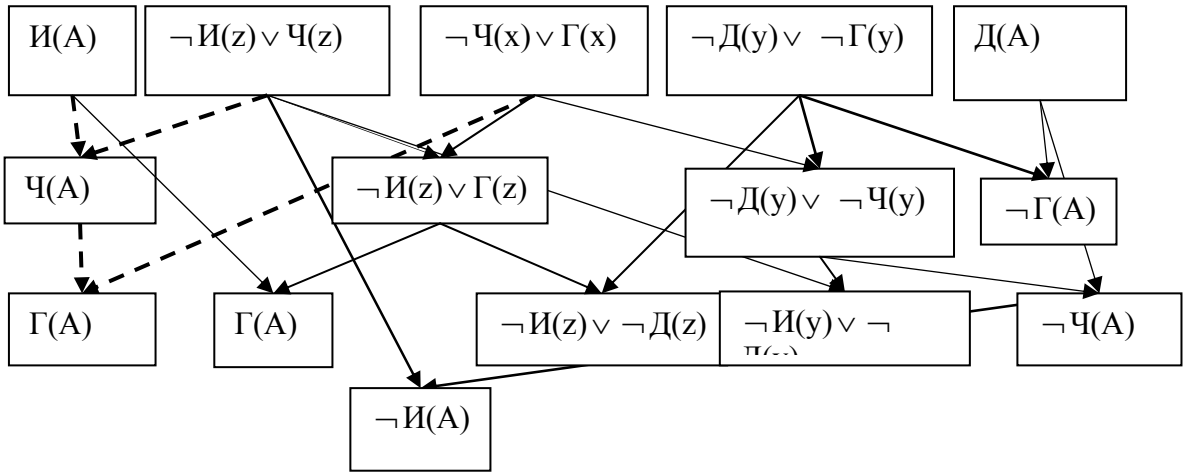
К стратегии опорного множества

Стратегия предпочтения одночленам. В этом случае для родительских вершин предпочтительнее выбирать одночлены. Но желательно стартовать с отрицания целевой ППФ



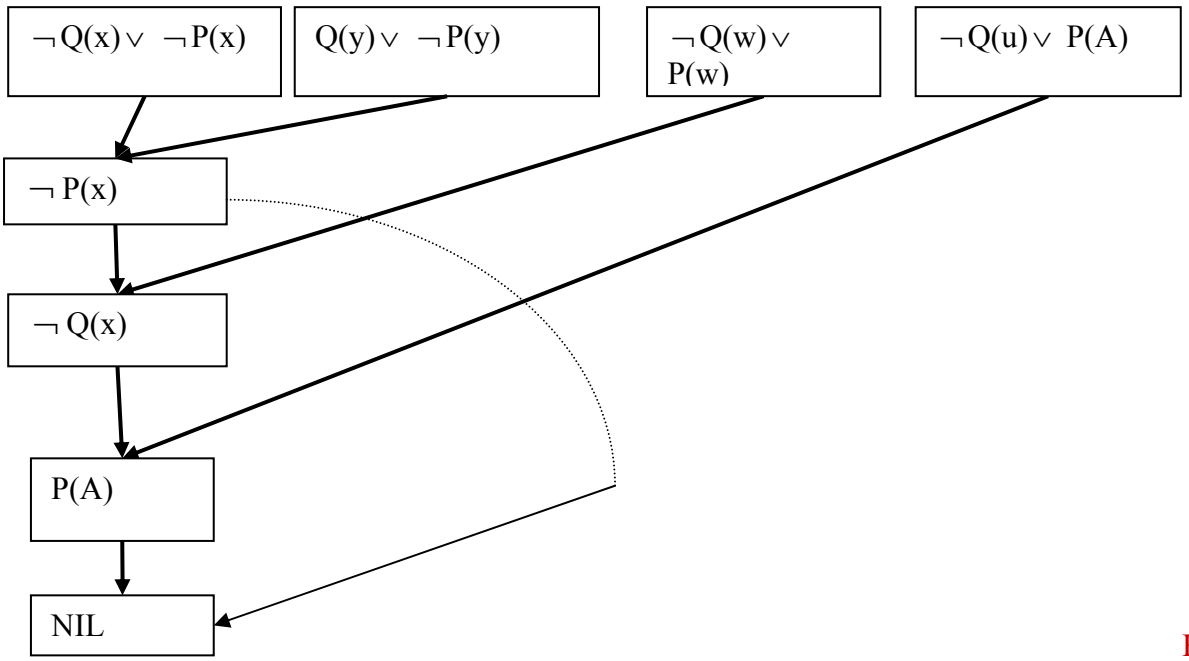
К стратегии предпочтения одночленам.

Линейная по входу стратегия. Линейное по входу опровержение – опровержение, когда одно родительское предложение в резольвенте принадлежит базовому множеству.



Линейная по входу стратегия

Эта стратегия не отличается полнотой, и, в частности, не дает решения, если в базовом наборе нет однолитерального предложения (для порождения пустого предложения в этой стратегии нужно, чтобы было однолитеральное предложение из базового набора и однолитеральное, как следствие решения). Если исключить пунктирную кривую на следующем рисунке, это утверждение станет очевидным.



К

К стратегии с учетом предшествующих вершин.

Стратегия с учетом предшествующих вершин. Здесь требуется, чтобы одно из родительских предложений для резольвенты были или из базового множества (это то же самое, что и предыдущей, линейной по входу стратегии), или было одним из потомков другого родительского предложения (что расширяет возможности стратегии и позволяет добиться полноты). Именно пунктирная кривая связи на рис.5. в этом случае позволяет получить пустое предложение и доказать теорему.

Стратегия упрощения за счет исключения тавтологий. В это случаеиз базового набора и при последующих операциях могут быть отброшены предложения, содержащие тавтологии.

Пример: полностью исключаются из рассмотрения следующие предложения

P (x) ∨ ~ P(x),

Q (x) ∨ P(y) ∨ ~ P(y)].

Стратегия упрощения за счет оценки литерала в предложении. Оценка некоторых литералов может быть проведена независимо за счет т.н. ***присоединенных процедур***.

Пример: Если Q (x) ∨ P(y) ∨ E(x=3, y=5), где E(x=3, y=5) истина при x=y и ложь в ином случае, то можно ввести процедуру (присоединенную к предикатному символу E) **EQUALS**(x, y) и если при проведении процедуры для x=3, y=5 получим «ложь», то и литералу E(x=3, y=5) тоже следует присвоить значение «ложь». Только при этом вместо Q (x) ∨ P(y) ∨ E(x=3, y=5) можно записать Q (x) ∨ P(y).

Исключение при подсуммировании. При решениях допустимо исключать предложения, которые при определенной подстановке являются частью другого предложения (или с ним совпадают). При этом подстановка должна быть учтена в дальнейшем. Говорят, что предложения, которые при данной подстановке являются частью другого предложения, ***подсуммируют*** это другое предложение.

Пример: предложение P (x) ∨ ~ Q(A) подсуммирует R (x) ∨ P(y) ∨ ~ Q(z)] и может быть отброшено при x=y и z=A.

5.4.Графы

5.4.1.Сетевые представления

Для структурирования (и локализации структуры базы данных или сбора информации по одному объекту) можно использовать графическое представление в форме ***концептуальных графов***. Этот граф представляет собой логическую формулу. Имена и аргументы предикатов представлены двумя типами узлов. Дуги соединяют имена предикатов с их аргументами.

Концептуальный граф. Логика предикатов – здесь интерпретация в терминах области рассуждений (экспертизы). Логические формулы – фразы метаязыка. Аргументы предикатов – атрибуты, события, состояния. Имена предикатов указывают способ связей (правила грамматики, правила соединения, процедуры) между понятиями.

> ***Сори́т*** (от греч. σωρός — «куча») — цепь силлогизмов, в которых заключение является одной из посылок следующего за ним, а одна из посылок при этом не выражается в явной форме.
> Например:
>
> Англичане — мужественный народ.
> Мужественный народ свободен.
> Свободный народ счастлив.
> Следовательно, англичане счастливы

Семантические сети состоят из множества концептуальных графов. Это представление позволяет визуализировать множество отношений между концептуальными графами и составляющими отдельных графов.

5.4.2. Поиск на графе

Поиск на графе – формирование разных применений набора правил из общего их множества и порожденных этими применениями баз данных образуют ***дерево поиска.*** В ***корне*** дерева – описание исходной конфигурации. Правила – ***дуги***, которые ведут к вершинам - преемникам (потомкам).

Здесь вершины соответствуют базам данных, а дуги – правилам. Дуги направлены от родителей к преемникам (***направленный граф***). ***Дерево*** – частный случай графа, каждая вершина которого имеет не более одного родителя. Если две вершины одновременно преемники друг друга, то пара дуг заменяется ***ребром***. Вершина, не имеющая родителя – ***корневая***, а не имеющая преемников – ***концевая***. Корневой вершине соответствует нулевая глубина, ***глубина*** любой другой вершины равна глубине предшественника плюс единица. Последовательность вершин, каждая следующая из которых является предшественником предыдущей, называется ***путем***, с ***длиной*** равной количеству вершин в последовательности. Для одного пути можно ввести понятия ***потомка*** и ***предка***, а также определить ***достижимость*** одной вершины из другой. Каждой дуге можно определить значения различных характеристик, например, ***стоимости***. Это позволяет в системах управления рассматривать, например, ***суммарную стоимость*** пути и требовать её оптимизации. Множество вершин, представляющих базы данных терминального условия называют ***целевым***, а каждую вершину из этого множества ***целевой***. Граф может быть задан ***явно*** или ***неявно*** (если порождается самой стратегией управления, то есть задается этой стратегией явно). Можно ввести представление ***оператора построения преемников*** - применение этого оператора к отдельной вершине выявляет всех преемников (процесс применения называют ***раскрытием*** этой вершины).

5.4.3.Стратегия управления

Стратегию управления с поиском на графе можно также рассматривать как процесс выявления части ***неявного графа***, содержащей целевую вершину. Дерево (скорее куст) растет до тех пор, пока одна из его ветвей не удовлетворит терминальному условию, причем, вообще говоря, такая стратегия весьма не эффективна. Таким образом, в отличие от предыдущего случая (то есть, режима с возвращением) система помнит все

предыдущие пути, отдельные шаги каждого пути и сформированные в результате этих шагов модифицированные базы данных. Достоинство такой ситуации в том, что система может стартовать не с исходной позиции, а с той модифицированной базы данных, которая оказалась бы наиболее близкой к результату задачи, если в процедуре поиска существуют критерии близости промежуточного состояния к результирующему состоянию. Полезно с другой стороны найти условия (вообще говоря, эвристические), которые все же ограничивали бы количество ветвей, делали дерево (куст) более узким.

ПРИМЕР: Порождение части неявного графа - procedure **GRAPHSEARCH**

1.Создать граф поиска ***G***, состоящий только из начальной вершины **s**. Занести **s** в список, определенный как *OPEN*.
2.Создать список, названный *CLOSED*. который вначале пуст.
3.LOOP: если *OPEN* пуст, то неудача, окончание работы.
4.Выбрать первую вершину в *OPEN* , убрать её из *OPEN* и поместить в *CLOSED*. Обозначить эту вершину через ***n***.
5. Если ***n*** – целевая вершина - успех, окончание работы с решением, полученным в результате отслеживания пути в ***G*** от ***n*** к **s** (указатели определены на шаге 7).
6. Раскрыть вершину ***n*** , порождая множество ***M*** её преемников, не являющихся предками ***n***. Поместить в ***G*** эти элементы множества ***M*** как преемники ***n***.
7. Ввести указатель к ***n*** от тех элементов ***M***, которые еще не были в ***G*** (т.е. ни в *OPEN* , ни в *CLOSED*). Добавить эти элементы в *OPEN*. *Для каждого элемента из **M**, который уже был в OPEN или CLOSED, решить, нужно ли переориентировать указатель на **n**. Для каждого элемента из **M**, уже находящегося в CLOSED, принять решение относительно каждого его потомка в **G** – нужно ли переориентировать его указатель.*
8. Переупорядочить список *OPEN* в соответствии либо с некоторой произвольной схемой, либо с эвристической значимостью.
9.Перейти к метке LOOP.

ПОЯСНЕНИЯ: Процедура порождает граф ***G***, который называют ***графом поиска***. Существует ***T*** - подмножество ***G***, которое называемое ***деревом поиска***. Дерево поиска ***T*** определено указателями (7), причем каждая вершина имеет только одного родителя. Все возможные указатели заданы в ***G*** и их следует брать оттуда и использовать, причем родителей отдельной вершины в ***G*** может быть несколько. Ни одна вершина в ***G*** не является своим собственным предком (6). Упорядочение в ***G*** является частичным. Вершины в списке *OPEN* являются концевыми вершинами дерева поиска, в вершины в списке *CLOSED* – не концевыми. На шаге 3 процедуры, в списке *OPEN* концевые вершины которые еще не выбирались для раскрытия. В списке *CLOSED* – это не концевые, или концевые, которые выбраны для раскрытия. Выбор вершины для раскрытия на шаге 4 происходит на основании переупорядочения на шаге 8.

5.4.4.Развитие системы

Неинформированные процедуры – процедуры, не имеющие эвристической информации для упорядочения вершин и оценки характеристик дуг при процессах формирования неявного графа. Для неинформированных процедур (как правило, они редко используются в системах ИИ) существуют методы: поиск в глубину с введением ограничений на глубину *проходит каждый раз до предельной глубины, потом возвращается в самую верхнюю точку ветвления и снова на самую глубину и т.д.* (обычно вместо этого используют режим с возвращением – он менее ресурсоемкий) и поиск в ширину *проходит слоями на каждом уровне глубины.*

Эвристические методы – методы эвристического поиска – использование дополнительной информации для сокращения поиска. Основаны на минимизации комбинации стоимости пути к цели (сумме величин, характеризующих элементы-дуги на пути к цели) и стоимости (объема вычислений) поиска этого пути. Для этого вводятся т.н. ***оценочные функции***, на основании которых некоторые вершины дерева поиска выбрасываются из рассмотрения.

Двунаправленный поиск – два фронта формирования поиска от исходной и от

целевой вершин в случае неинформированных процедур встречаются быстрее и формируют наиболее короткий путь. В случае эвристических процедур происходит искажение фронтов – они могут встретиться далеко на периферии или в случае ограничений на глубину поиска вообще не встретиться.

Поэтапные поиски – поиски «с выбираемой плавающей точкой». Используют в случае ограниченных ресурсов или в случае обширных графов для случаев эвристического поиска. Информация первого этапа запоминается и старт второго происходит с наиболее приемлемой (с позиций эвристики, оптимального значения оценочной функции) достигнутой на первом этапе вершины.

Характеристики качества работы - направленность поиска $P = L / T$, где L – длина пути к цели, T- общее число порожденных вершин. ***Показатель эффективности ветвления*** – среднее число преемников отдельной вершины дерева. Оценить можно из уравнения $B + B^2 + ... + B^L = T$ или, суммируя, получим неявный вид $T = B(B^L - 1)/(B - 1)$.

5.4.5.Версия исчисления предикатов

Концептуальные графы и семантические сети могут представлять собой ***сетевые версии исчисления предикатов***.

Графы содержат (здесь) прямоугольники для представления аргументов и круги для имен предикатов. Соединены входящими и выходящими стрелками. Наиболее простым является концептуальный граф бинарного предиката.

Рис.5.1. slot-assertion

Вообще говоря, многие предикаты можно представить как произведение (конъюнкцию) нескольких бинарных предикатов. При этом используют следующие соглашения: каждая функция прежнего предиката становится именем бинарного предиката, первый аргумент которого - имя начального предиката, а второй является значением аргумента данной функции, эта пара называется слотом. Иначе, слот состоит из своего имени (имя этой функции) и ***ключа*** (значения соответствующего аргумента):

slot, slot - name, slot – value.

Итак, предикат вида

функция _ j (предикатное_ имя, значение _ j)

называют - ***slot - assertion notation***.

По договоренности все стрелки n-арного предиката направлены к кругу (то есть к имени предиката), а последняя n – ная стрелка от круга к последнему n –ному аргументу.

Концепты в семантических сетях могут определять свойства аналитические (свойства типа) и синтетические (свойства множества).

Некоторые определения

Иерархия типов - Надтипы и подтипы (надтип > подтип) – надтип - обобщающий образ подтипа. Универсальный надтип U = надтип всех типов и абсудный тип A = подтип всех типов. Для двух разных меток (типов) существует наибольший общий подтип и наименьший общий надтип. Они всегда находятся между U и A.

Денотат типа B – множество всех сущностей, которые являются конкретизациями некоторого концепта типа B.

Функция «тип» - отображает множество концептов на множество «меток (ярлыков) типа».

Прототип – конкретизация типа (свойства истинные в типичном и необязательные в реализации).

Узлы индивиды или узлы-ссылки (метод Совы):

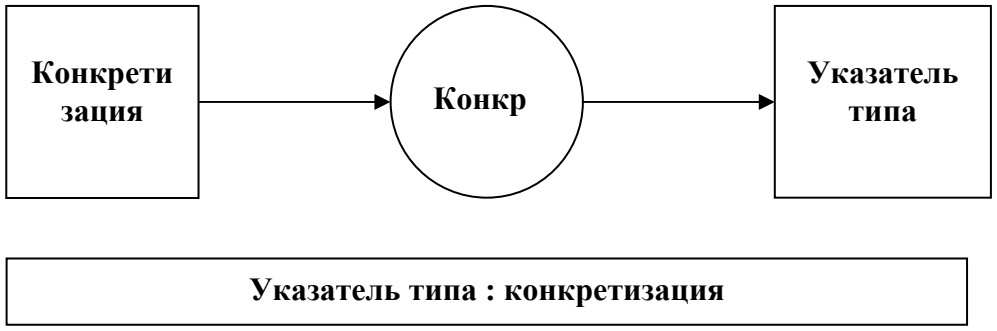


Рис.5.2. Структура бинарного предиката

Обычно используют бинарные предикаты, тогда узел-круг, указывающий имя предиката является ***связывающим узлом,*** а функция, которую он представляет называется ***концептуальным отношением,*** остальные узлы, соответствующие прямоугольникам - аргументам называют ***узлы- концепты.***

ПРИМЕРЫ:

Преобразования унарного предиката в бинарный:
Имя _ совокупности (имя _ индивидуума)
преобразуется в
Конкр(имя _ индивидуума, имя _ совокупности).

Преобразование 3-арного предиката в бинарный:
фраза: «Вася посылает конфетку Маше», которая записывается в форме предиката
Посылка(Вася _ 1, Маша _ 10, конфетка _10) и преобразуется в три бинарных предиката
1.Отправитель(Посылка, Вася _1);
2.Получатель (Посылка, Маша _10);
3.Объект (Посылка, конфетка _10).

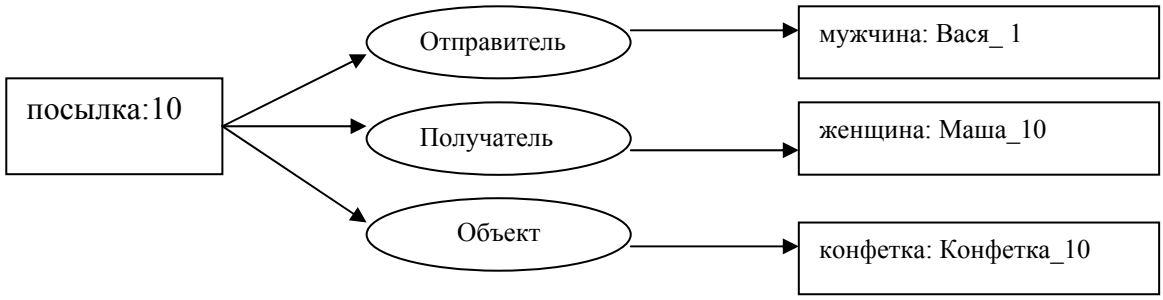


Рис.5.3. 3-арный предикат

Кстати, представление [книга : x] означает просто объект типа книга (поэтому можно заменить на [книга]), а [книга : Книга_2] – вполне конкретную книгу.

Правила конъюнкции - если узел- концепт c_1 в G1 и узел- концепт c_2 в G2

идентичны, то узел- концепт c_2 удаляют, а все его связи замыкают на узел- концепт c_1.

Правило упрощения **–** если есть два идентичных связывающих узла, соединенных одними и теми же узлами концептами, то один связывающий узел можно удалить.

Действие кванторов. Концептуальный граф делят на иерархическое множество зон действия отдельных кванторов. Требование конкретизации оформлены стрелкой от выделенного сектора с нужной переменной к связывающему узлу Конкр. И далее к прямоугольнику – узлу-концепту названному, например, ∀ Кван_форм. Действие квантора на переменную символично отражено связующим узлом со знаком ∀.

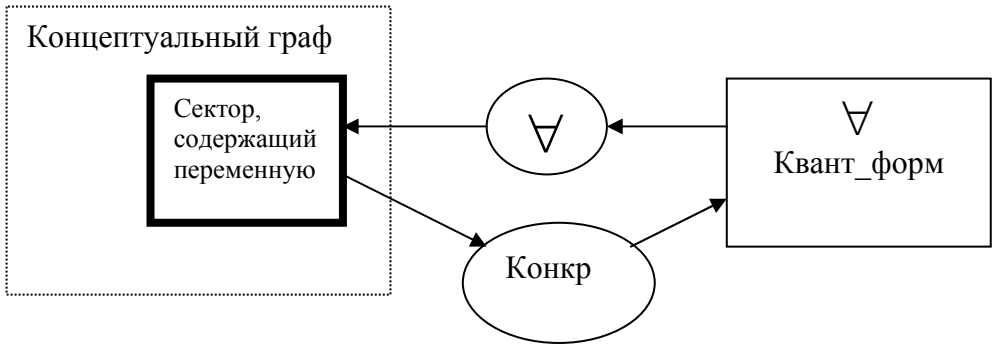


Рис.5.4. Действие квантора на граф

Канононические графы - семантически корректные - такие, которые не содержат в себе нереальных или невозможных ситуаций.

Схемы – дальнейшее ограничение после каноничности – специфические знания об области рассуждений (экспертизы), представлено все правдоподобное. Концепт (данного типа) можно определить «множеством его возможных применений», можно ввести понятие ***кластера*** или ***набора схем***, каждая из которых дает способ применения данного концепта. Набор всех возможностей применения типа называется ***схематическим кластером,*** который для типа B есть множество $\{\lambda x_1 F_1 , \lambda x_2 F_2 \ldots, \lambda x_n F_n \}$ λ выражений, причем каждый $x_к$ принадлежит к типу B и каждое выражение $\lambda x_к F_к$ - схема для типа B. Заметим, что λxF ***-λ выражение, представляющее схему***, - F – логическая формула (или представляющий её граф), формальный параметр x представляет концепт того типа, который нуждается в определении, а тело F укажет на один из способов применения этого типа.

Схемы соответствуют ***созвездиям, фреймам (кадрам), сценариям (предписаниям).***

Унаследованные свойства – свойства, которые неявно, но с определенностью следуют из уже представленной информации.

Сети Петри – представление процессов в форме графов - ориентированные графы, вершины которых представляют собой «позиции» (представлены окружностями) или «переходы» (поперечные отрезки). Позиции содержат «жетоны», разрешающие дальнейшее перемещение по сети (жетон при этом передается по цепочке, то есть в предыдущей позиции жетона уже не оказывается). Сеть включает в себя систему управления в виде жетонов и заложенных в переходах и позициях правил.

Представлениие знаний в виде графа. Графы типа И/ИЛИ

Использование графов в исчислении предикатов и организации логического вывода часто позволяет упростить и сократить процедуры. Но следует помнить, что представление в форме графов часто не позволяют переименовывать переменные, как это было возможным в прежде.

В этом разделе мы откажемся от перевода знаний в форме импликации в предложения, исключающие импликацию. Тогда все ППФ, разделяются на правила и

факты. ППФ в форме импликаций – это общие знания о предметной области, и они используются в качестве порождающих правил. Факты, которые не имеют импликативной формы – это специфические знания, относящиеся к данному конкретному случаю.

Предварительное преобразование: Форма (безымпликационная) ППФ, при необходимости после сколемовского преобразования, переводится в префиксную форму, переменные в пределах действия кванторов переименовываются, кванторы общности опускаются [3] .

Пример предварительного преобразования

Рассмотрим выражение $(\exists u)(\forall v)\{Q(v,u) \wedge \sim[[R(v) \vee P(v)] \wedge S(u,v)]\}$, которое нетрудно преобразовать сначала в $Q(v, A) \wedge [[\sim R(v) \wedge \sim P(v)] \vee \sim S(A,v)]$, а затем, чтобы переменные отличались, в форму И/ИЛИ (то есть с включением конъюнкций и дизъюнкций).

$$Q(w, A) \wedge [[\sim R(v) \wedge \sim P(v)] \vee \sim S(A,v)].$$

Схема построения графа И/ИЛИ (часто такой граф относят к гиперграфам) в обсуждаемой выше правильной форме достаточно проста, конъюнкция представляется в виде обычных связей, а дизъюнкция в виде к- связок, то есть объединенных связок.

Пример построения графа И/ИЛИ

Важно отметить, что здесь встречается конъюнкция и дизъюнкция. Построим граф И/ИЛИ:

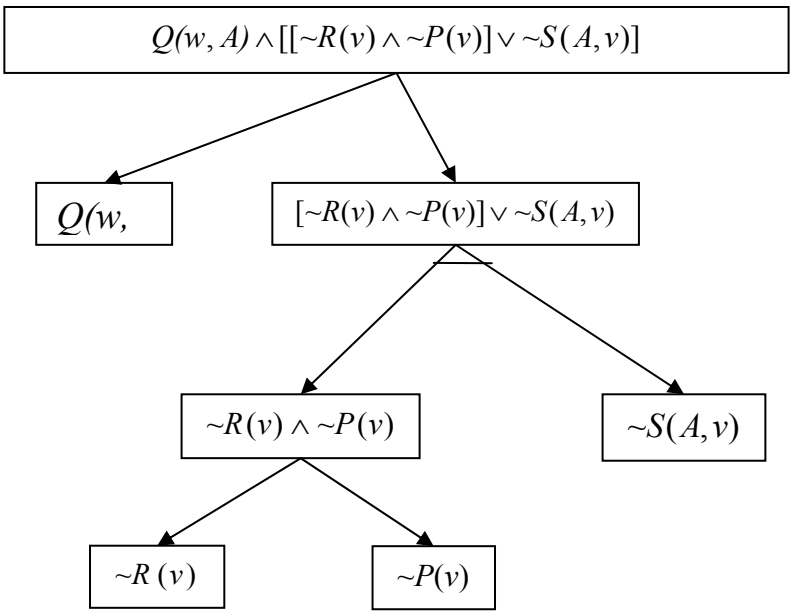


Рис. 5.5.Гиперграф утверждений И/ИЛИ

Дизъюнктивно связанные выражения $\sim R(v) \wedge \sim P(v)$ и $\sim S(A,v)$ в описании графа содержат, так называемую, к-связку (здесь к = 1). Множество предложений, в которые эта ППФ может быть преобразована, представляет собой множеством литералов (разумеется, конъюнктивно связанных между собой) на концевых вершинах графа, то есть

$$Q(w, A),$$
$$\sim R(v) \vee \sim S(A,v),$$
$$\sim P(v) \vee \sim S(A,v),$$

Отметим, что представление знаний в виде графа является менее общим, чем представление в виде этих трех предложений. Из-за существующего в последнем случае

произвола для переменной v в различных предложениях. То есть, последнее выражение, например, позволительно (в последнем случае коммутативной системы представлений знаний) записать иначе $\sim P(w) \vee \sim S(D, w)$.

Примеры применения правил и получения решений на графе

Тип доказательства: здесь будет использоваться непосредственное доказательство (дедукция, основанная на правилах), а не система опровержения, как ранее в разделах 1, 2. Прямая система доказательств - применяются П-правила для достижения условия остановки, включающей целевую ППФ. Обратная система – использование О-правил до достижения условия остановки, которое включает начальные условия- факты.

Прямая система продукций

Посмотрим как можно после применения к данному гиперграфу правила, например, такого

$$\sim S(x, y) \Rightarrow R(x) \wedge Q(y),$$

получить решение. Подставим правило в граф утверждений. При этом следует применить соответствующие унификаторы для переменных.

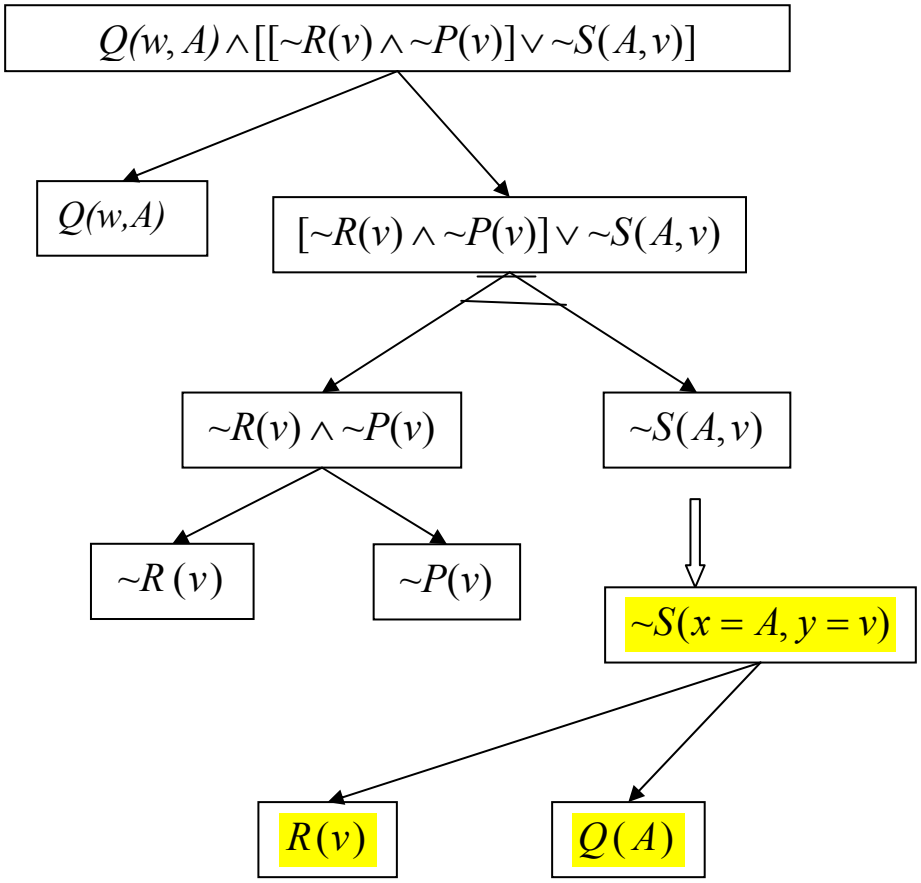


Рис. 5.6. Присоединение правила к графу фактов

Ответом, то есть ***разрешением графа фактов на правиле S***, будет набор дизъюнкций корневых выражений $\sim R(v) \vee Q(A)$, $\sim P(v) \vee Q(A)$, $\sim R(v) \vee R(v)$, $R(v) \vee \sim P(y)$. Получение ответа при применении подобной процедуры.

Задания:

1. Какое решение из полученных выше нужно исключить.
2. Покажите, что произойдет при применении правила $\sim S(x, y) \Rightarrow \sim R(x) \wedge Q(y)$.

Использование целевой ППФ для остановки. Если предположить, что целевая ППФ есть $\sim P(v) \vee Q(A)$, то при применении правила одним из решений может быть именно это предложение, что и остановит программу.

Обратная система продукций

Здесь будем использовать графы И/ИЛИ, где к-связки используем для разделения дизъюнктивно связанных предложений, представляющих из себя отдельные литералы или их конъюнкции, например $L_1 \wedge L_2$ [8].

1.Так как правила имеют вид $W \Rightarrow L$, *то при таком описании* $W \Rightarrow L_1 \wedge L_2$ *все равно сводится к* $W \Rightarrow L_1$ *и* $W \Rightarrow L_2$, *что облегчает анализ.*

2.Можно приводить правила к его отрицанию и исключению импликации $W \Rightarrow L_1$ *то есть замены ее на* $\sim W \vee L_1$ *и отрицанию, что дает* $W \wedge \sim L_1$, *что также сохраняет принятую конъюнктивную форму предложений.*

Рассмотрим пример ***обратной системы продукции***.

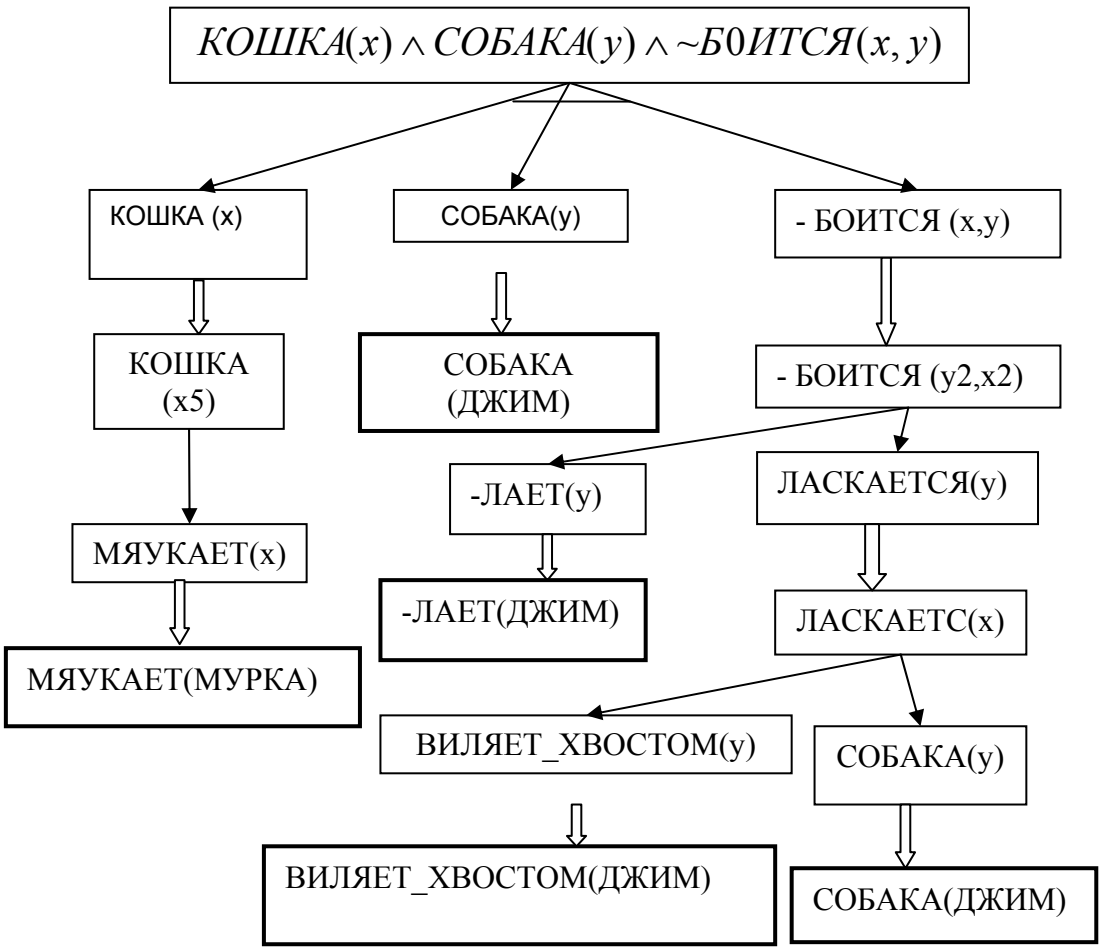
Факты: СОБАКА(ДЖИМ);
~ ЛАЕТ(ДЖИМ);
ВИЛЯЕТ_ХВОСТОМ(ДЖИМ);
МЯУКАЕТ(МУРКА).

Правила:

[ВИЛЯЕТ_ХВОСТОМ(x) $\wedge$ СОБАКА(x)] $\Rightarrow$ ЛАСКАЕТСЯ(x1)
[ЛАСКАЕТСЯ (x2) $\wedge \sim$ ЛАЕТ(x2)] $\Rightarrow \sim$ БОИТСЯ (y2,x2)
СОБАКА(x3) $\Rightarrow$ ЖИВОТНОЕ (x3)
КОШКА(x4) $\Rightarrow$ ЖИВОТНОЕ (x4)
МЯУКАЕТ(x5) $\Rightarrow$ КОШКА (x5)

$(\exists x)(\exists y)[КОШКА(x) \wedge СОБАКА(y) \wedge \sim БОИТСЯ(x, y)]$ - это вопрос (есть ли такие кошка и собака, причем кошка не боится собаки?)

[8] Понятно, что переименовывать переменные можно лишь в разных предложениях, представляющих собой конъюнкции.



Граф решения для обратной системы, которую следует проверить на согласование, выделены вершины, соответствующие фактам.

Согласованные графы решений. Если задан вопрос, о возможности существования участников определенного действия или события, и относительно этих участников что-то известно, то данная задача имеет смысл. Причем если возможно согласование всех аргументов (или значений переменных), нет противоречий, то такое решение представляет практический интерес. Нужно только отметить, что оно может быть иной раз вероятностным, потребовать для полной определенности дополнительных условий. В таких задачах основной акцент сделан на согласовании (после унификации) аргументов.

Оказывается, существуют унифицирующие подстановки {МУРКА/ x}, {МУРКА/ x5}; {МУРКА/ y2}; {ДЖИМ/ y}; {ДЖИМ/ x2};{ДЖИМ/ x}, при котором все согласовано и ответ тоже: КОШКА (МУРКА) ∧ СОБАКА(ДЖИМ) ∧ ~ БОИТСЯ (МУРКА, ДЖИМ).

Условие остановки. Условием остановки является факт согласования такого графа, который должен непременно заканчиваться в вершинах фактов, которые на рисунке выделены.

5.5.Система фреймов

Система фреймов - как модель представления знаний - хорошо отражает концептуальную основу организации памяти человека, а также обладает гибкостью и наглядностью.

Фрейм (англ. frame - каркас или рамка) предложен М.Минским в 70-е гг. как структура знаний для восприятия пространственных сцен. Сейчас под фреймом понимается абстрактный образ или ситуация.

5.5.1. Структура фрейма

Имя фрейма: (имя 1-го слота: значение 1-го слота), (имя 2-го слота: значение 2-го слота),…(имя N-го слота: значение N-го слота).

Представление в виде таблицы 4.10, которая дополнена двумя столбцами.

Таблица 5.10

Имя слота	значение слота	способ получения значения	присоединённая процедура

Дополнительные столбцы предназначены для описания способа ***получения слотом его значения*** и возможного ***присоединения к тому или иному слоту специальных процедур***, что допускается в теории фреймов. В качестве значения слота может выступать имя другого фрейма; так образуют ***сети фреймов***.
Фреймы-образцы, или прототипы, хранятся в базе знаний,
Фреймы-экземпляры создаются для отображения реальных ситуаций на основе поступающих данных.

Процедура сравнения (сопоставления). Слоты, заданные по умолчанию могут корректироваться (менять значения) и не исключаются из процесса сравнения. Если нет соответствия, то переходят к другому фрейму. Связанные (в каждом слоте есть задано условие, которое должно выполняться при установлении соответствия между значениями) фреймы – они сами являются отношениями, - формируют фреймовую систему. *Такие системы уже используют для решения задач формирования отчетов и расчетов, а также для обработки изображений…*

Модель фрейма является достаточно универсальной, поскольку позволяет отобразить все многообразие знаний о мире через:

фреймы-структуры, для обозначения объектов и понятий (заем, залог, вексель);
фреймы-роли (менеджер, кассир, клиент);
фреймы-сценарии (банкротство, собрание акционеров, празднование именин);
фреймы-ситуации (тревога, авария, рабочий режим устройства) и др.

Важнейшим свойством теории фреймов является заимствованное из теории семантических сетей наследование свойств. И во фреймах, и в семантических сетях наследование происходит по АКО-связям (A-Kind-Of = это). Слот АКО указывает на фрейм более высокого уровня иерархии, откуда неявно наследуются, т.е. переносятся, значения аналогичных слотов.

Специальные языки представления знаний в сетях фреймов FRL (Frame Representation Language)
Широко известны такие фреймо-ориентированные экспертные системы, как ANALYST, МОДИС.
Теория фреймов пока больше соответствует теории постановки задач, чем результативной теории.

5.5.2. Описание знаний с помощью фреймов

1.***Взаимопроникновение структур*** данных и структур процедур.
2.Законы вывода из-за этого ***лишаются формальной строгости*** правил вывода (подобных логике предикатов).
3.Достигнута ***зато*** большая ***эффективность исчисления*** (за счет ослабления требований полноты и др.).

Объектное представление можно получить из логического или сетевого

представлений. Из логических формул, содержащих одни и те же конкретизации собирают фреймы. Это можно сделать собрав, например, все бинарные предикаты, которые относятся к рассматриваемому объекту. Это реализовано в семействе объектно-ориентированных языков.

Сцепка *(unit)* –множество всех фактов о данном концепте. Напомним, концепт (данного типа) можно определить «множеством его возможных применений».

Фреймы *(slot–and-filler notation)* – сцепки, где все фразы выражены бинарными предикатами.

Слот – пара (атрибут, значение) фрейма.

Явный фрейм (*case-frame*) – фрейм, где представлены конкретные значения аргументов и явные имена формул.

Функциональный фрейм - используем функциональные представления:

ПРИМЕР:

Фраза: «Вася посылает конфетку Маше», которая записывается в форме фрейма:

Посылка_10		
элем	:	*(элем_из посылок)*
отправитель	:	*(Вася_1)*
получатель	:	(Маша_10)
объект	:	*(Конфетка_10)*

Здесь элем – это множество (посылок).

Рис. 5.7.Вид явного фрейма

$\forall$ ***квантификация переменных:***

Предложение «Вася посылает конфетку каждой женщине» описывается бинарными предикатами:

$\forall x \exists y \exists z$ [Отправитель (z, Вася_1) $\wedge$ Получатель(z,x) $\wedge$ Объект (z,y) $\wedge$ Элем (z, посылки)

Конкр (y, конфета) $\wedge$ Конкр (x, женщина)

Здесь Элем – это множество (посылок).

Функциональный фрейм в области действия квантора общности принимает вид:

z (x)		
элем	:	*(элем_из посылок)*
отправитель	:	*(Вася_1)*
получатель	:	*x*
объект	:	*y(x)*

Рис.5.8. Вид функционального фрейма

Паросочетание (унификация) двух объектных представлений (matching operation) **-** существуют подстановки для переменных, делающие логические формулы двух объектных представлений идентичными.

Для использования объектного представления в качестве БД для запросной системы (query system), для построения языка запросов (query language). Речь идет о том, чтобы логическая формула «объекта-цели» унифицировалась с одним из сомножителей «объекта-факта» (напомним, последний представлен в виде конъюнкции гипотез и аксиом).

ПРИМЕР: Подстановка {(*z*, Посылка_10), (x, Маша_10), (y, Конфетка_10)} унифицирует или паросочетает приведенные выше функциональные фреймы.

Атрибут – конкретизирует некоторые данные или функционально связывает эти данные (последующего фрейма) с иными результатами предыдущих исчислений (предыдущего фрейма). Во втором случае – это ***функциональный атрибут***.

ПРИМЕР: В глобальной базе данных есть два (явных) функциональных фрейма.

а) явный фрейм

Посылка_10		
элем	:	*(элем_из посылок)*
отправитель	:	*(Вася_1)*
получатель	:	(Маша_10)
объект	:	*(Конфетка_10)*

б) функциональный фрейм

Посылка_111		
элем	:	*(элем_из посылок)*
отправитель	:	*получатель (Посылка _10)*
получатель	:	(Петя_111)
объект	:	*(Конфетка_10)*

Рис.5.9.Глобальная база данных: явный фрейм (а) соответствует известному Факту и функциональный фрейм (б) соответствует Правилу.

Здесь очевидно, что Маша_10 и есть *получатель (Посылка _10),* но вопрос может быть поставлен иначе - Посылала ли Маша что-нибудь кому-нибудь? Этот вопрос может быть представлен уже известным третьим функциональным фреймом:

z (x)		
элем	:	*(элем_из посылок)*
отправитель	:	*(Маша_10)*
получатель	:	*x*
объект	:	*y(x)*

Рис. 5.10. Добавление в глобальную базу данных нового функционального фрейма, эквивалентного запросу

Подстановка *{(x, Петя_111), (y, Конфетка_10) }* – «Маша отправила конфетку Пете».

Информация об объектах

Информация об объектах (инвариантная репрезентация) с расширением и усложнением внутреннего мира интеллектуальных систем будет становиться все более обширной и адекватной, но пока мы далеки от возможностей человеческого разума. Поэтому, опять таки пока, все образы, которыми мы оперируем в интеллектуальных системах достаточно схематичны и очень далеки от реальных. Но это поправимо и мы быстро будем наращивать мощности, объемы доступной памяти, развивать системы распознавания и т.д.

5.6.Нечеткая логика

5.6.1.Элементы нечетких множеств

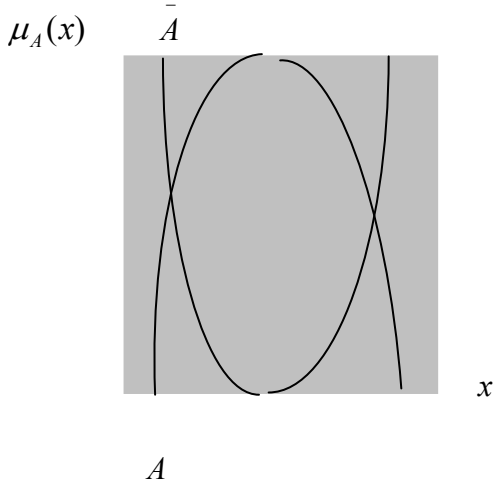
Нечеткая логика применяется для задач слабо формализованных и задач с ненадежными условиями и с неопределенными выражениями; располагается между формальным и естественным описаниями; описывает – аппроксимирует любую математическую модель (теорема FAT = Fuzzy Approximation Theorem, B. Kosko, 1993); но в действительности исходный набор данных часто не только неполный, а и слегка противоречив, введенные экспертами переменные входа и выхода могут оказаться не вполне адекватно

описывающими реальность; потому с необходимостью от систем требуют адаптивности, то есть они должны допускать коррекцию знаний и параметров в процессе решения задач [4].

Характеристическая функция = функция принадлежности элемента к множеству, - принимает значения от нуля до единицы или до иной величины – верхней границы (в четких множествах только 0 и 1), то есть $\mu_A(x)$ – это характеристическая функция, принимающая значения на множестве *M*.

Нечеткие (fuzzy) множества (L. Zadeh, 1965) – множества элементов, описываемых характеристическими функциями.

Пусть *E* – универсальное множество элементов *x*, а *G* – некоторое свойство. $A = \{\mu_A(x)/x\}$ – это подмножество *E*, где $\mu_A(x)$ – характеристическая функция, принимающая значения на множестве *M*, определяющие наличие (долю, степень) свойства G у элемента *x*. Если $M = \{0,1\}$ - это четкое множество, то $M = [0,1]$ - нечеткое (нормализованное) множество.



$$A \cap \bar{A} \neq \varnothing$$

$$A \cup \bar{A} \neq E$$

Рис.5.11. Представление характеристических функций нечеткого множества и его дополнения.

Высота нечеткого множества - Величина верхней границы M то есть $\sup_{x \in A} \mu_A(x)$ - высота нечеткого множества (если высота =1 – это множество нормально, а если меньше – субнормально).

Нормализация нечеткого множества - процедура вида $\mu(x) = \mu_A(x) / \sup_{x \in A} \mu_A(x)$.

Унимодальность элемента x_0 - $\mu_A(x_0) = 1$.

Носитель нечеткого множества A – его подмножество для которого $\mu_A(x) > 0$.

Точки перехода множества A – элементы для которых справедливо $\mu_A(x) = 1/2$.

Таблица 5.11.

Логические операции

1.Включение - $\forall x \in E$, $\{\mu_A(x) \le \mu_B(x)\}$, то A содержится в B (B доминирует A) или $A \subset B$

2.Равенство - $\forall x \in E \quad \mu_A(x) = \mu_B(x)$, то $A = B$.

3.Дополнение – $M = [0,1]$, A и B на М, $\forall x \in E$, $\mu_A(x) = 1 - \mu_B(x)$, то

$B = \bar{A}$ или $A = \bar{B}$

4.Пересечение - $\forall x \in E, \{\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))\}$ - наибольшее подмножество содержащееся и в А и в В или

$A \cap B$

5.Объединение - $\forall x \in E$, - $\{\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))\}$ - наибольшее подмножество содержащееся и в А и в В или

$A \cup B$

6.*Разность* – это А–В = $A \cap \bar{B}$ $\{\mu_{A \cup B}(x) = \mu_{A-B} = \max(\mu_A(x), 1 - \mu_B(x))\}$,

$A - B = A \cap \bar{B}$

7.Дизъюнктивная сумма $A \oplus B = (A - B) \cup (B - A) = (A \cap \bar{B}) \cup (B \cap \bar{A})$ или

$\{\mu_{A \oplus B}(x) = \max\{[\min(\mu_A(x), 1 - \mu_B(x))], \min[(1 - \mu_A(x), \mu_B(x))]\}\}$,

$A \oplus B = (A - B) \cup (B - A) = (A \cap \bar{B}) \cup (B \cap \bar{A})$

Нечеткие переменные - <α,X,A> - где α - наименование нечеткой переменной, X - область определения переменной (универсальное множество), А – нечеткое множество, с функцией принадлежности $\mu_A(x)$.

Лингвистическая переменная <α,T, X,G,M> , где α - наименование лингвистической переменной, X - область определения переменной (универсальное множество), Т – (базовое) терм - множество (множество значений = наименований) лингвистической переменной, *G* – синтаксическая процедура, позволяющая генерировать новые термы (значения), причем $T \cup G(T)$ - расширенное терм- множество лингвистической переменной, где $G(T)$ - сгенерированное множество термов; *M* - семантическая процедура, превращающая новые значения термов в элементы нечетких множеств.

Таблица 5.12.

Алгебраические операции

1.Алгебраическое произведение - $\forall x \in E$, $\mu_{A \cdot B}(x) = \mu_A(x) \cdot \mu_B(x)$

$A \cdot B$

2.Алгебраическая сумма - $\forall x \in E$, $\mu_{A+B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)$

$A + B$

для которых

выполняются свойства коммутативности (положение сомножителей неважно);

ассоциативности (порядок двух последовательных одинаковых операций неважны);

$A \cdot E = A$, $A + E = E$,

умножение на нуль и добавление нуля – обычные,

справедливы формулы де Моргана: $\overline{A \cdot B} = \bar{A} + \bar{B}$, $\overline{A + B} = \bar{A} \cdot \bar{B}$,

а также $A \cdot \bar{A} = \varnothing$, $A + \bar{A} = E$.

Не справедливы идемпотентность (то есть, $A \cdot A = A$, $A + A = A$, которые для логических напротив справедливы) и дистрибутивность (порядок двух разных операций).

Справедливы дистрибутивность одной из алгебраических и одной из логических ($\cup, \cap$)операций.

Можно возводить в степень, если степень больше единицы – это уплотнение =

концентрация, а меньше единицы - растяжение.

3. Умножение на число – *p (*при $p \sup_{x \in A} \mu_A(x) \le 1$*).* Множество *pA* есть множество для которого $\mu_{pA}(x) = p\mu_A$.

4.Выпуклая комбинация n множеств {A}- определяется функцией принадлежности $\mu(x) = \sum_{i=1}^{n} \omega_i \mu_{Ai}(x)$.

5.Прямое (декартово) произведение множеств – произведение n множеств {A} определенные на n универсальных множествах {E} определяется функцией принадлежности $\mu = \min\{\mu_{A1}(x_1), \mu_{A2}(x_2), ..., \mu_{An}(x_n),\}$ на множестве $E_1 \times E_1 \times ... \times E_n$.

6.Оператор увеличения нечеткости - $H(A,K) = \bigcup_{x \in E} \mu_A(x) K(x)$ где *K(x)* – подмножества универсального множества *E* (их совокупность – ядро оператора увеличения нечеткости H).

7.Четкое множество α уровня – множество α уровня нечетного множества *A* универсального множества *E* - четкое подмножество $A_\alpha = \{x / \mu_A(x) \ge \alpha\}$ универсального множества *E,* где $\alpha \le 1$, то есть только те *x,* которым в нечетком множестве *A* соответствовали $\mu_A(x) \ge \alpha$.

Для операций над нечеткими числами порой используют обозначения

$$\vee_x = \max_x, \wedge_x = \min_x$$

хотя допустимы и иные представления

Нечеткие числа – нечеткое множество A на множестве действительных чисел R с функцией принадлежности $\mu_A(x) \in [0,1], \quad x \in R$.

Нормальные числа A – если $\max_x \mu_A(x) = 1$.

Выпуклые числа A – если для любых $x \le y \le z$ $\mu_A(x) \ge \mu_A(y) \cap \mu_A(z)$.

Множество α уровня нечеткого числа A - это $A_\alpha = \{x / \mu_A(x) \ge \alpha\}$

Подмножество $S_A = \{x / \mu_A(x) \ge 0\} \subset R$ ***- носитель нечеткого числа A.***

Нечеткий нуль - выпуклое нечеткое число A для которого справедливо $\mu_A(0) = \sup_x \{\mu_A(x)\}$.

Положительное нечеткое число A если при $\forall x \in S_A, x > 0$.

Нечеткое n-арное отношение – нечеткое подмножество *R* на прямом произведении универсальных множеств $E = E_1 \times E_1 \times ... \times E_n$ принимающее значения на множестве принадлежностей *M*.

Пример: Для случая *n* = 2, нечеткое отношение определяется функцией *R: (X,Y)* на [0,1], для которого каждому набору *(x,y)* из *(X,Y)* ставиться в соответствие $\mu_R(x,y)$. Обозначается нечеткое отношение на $X \times Y$ в виде: $x \in X, y \in Y : xRy$. В случае *X=Y*, используют обозначениее нечеткого отношения на множестве X $R : X \times X \to [0,1]$.

Обычное отношение, ближайшее к нечеткому $\underline{R}$ определяется как

$$\underline{R}: \mu_R(x,y) = 0..if...\mu_R(x,y) < 0.5;...1..if...\mu_R(x,y) > 0.5;...0...or...1,..if...\mu_R(x,y) = 0.5,$$

Композиция (свертка) = *(max-min композиция, max-min свертка)* – $R_1 \bullet R_2$ то есть $\mu_{R1\bullet R2}(x,z) = \bigvee_y [\mu_{R1}(x,y) \wedge \mu_{R2}(y,z)]$, если заданы два нечетких отношения $R_1 : X \times Y \to [0,1] \quad R_2 : Y \times Z \to [0,1]$.

Логика Zadeh. Было введено понятие «расплывчатых» множеств - как множеств, имеющих различные градации степени принадлежности к данному классу. Здесь можно говорить о случайности лишь отчасти, разве что лишь с позиций неопределенности относительного уровня принадлежности к данному классу. Вводились разные методы расчетов, например вида $\alpha + b \Rightarrow \max(a,b)$ или $\alpha \cdot b \Rightarrow \min(a,b)$.Расплывчатое множество A в X вводилось как совокупность пар $A = \{x, \mu_A(x)\}$ где $x \in X$, то есть $\mu_A(x)$ функция, отображающая X в M, где M-пространство принадлежности $M = [0,1]$. Задача оценки (определения) $\mu_A(x)$ в множестве пар $\{x, \mu_A(x)\}$- задача распознавания образов. Оператор размытия (в оригинале fuzzifier) размывает границы множества, создает не вполне определенные образы. Вместо « x меньше 1 и больше 2» используем «x приблизительно находится между 1 и 2».
Пересечение множеств может быть определено в жестком смысле, например, $\mu_{A\cap B} = \min\{\mu_A(x), \mu_B(x)\}$, то есть или $\mu_A(x)$, или $\mu_B(x)$ в зависимости что оказалось меньше. Возможно пересечение в более мягком смысле $\mu_{A\cap B} = \mu_A(x) \cdot \mu_B(x)$, где остается взаимное участие множеств $\mu_A(x)$, и $\mu_B(x)$.
В структуре принятия решений обычно рассматривается множество альтернатив и множество ограничений. Для альтернатив вводится функция предпочтительности (например, выигрыш или проигрыш при ее выборе). Обыкновенно стараются создавать симметрию по отношению к альтернативам и ограничениям, что упрощает принятие решений. Расплывчатое решение можно считать расплывчатым множеством в пространстве альтернатив, которое получается пересечением альтернатив и ограничений. Способы перехода к четкости- обычно это взвешенное среднее всех $\mu_A(x)$.

5.6.2.Формальные схемы нечеткого логического вывода

Рассмотрим чисто формально следующие примеры.
Алгоритм Mamdani – 1) С помощью экспертов вводится нечеткость в форме нечетких множеств = функции принадлежности $A_1(x)$, $A_2(x)$, $B_1(y)$, $B_2(y)$, $C_1(z)$, $C_2(z)$;
2). С помощью экспертов вводятся правила:
П1:Если x есть A_1 и у есть B_1, то z есть C_1;
П2:Если x есть A_2 и у есть B_2, то z есть C_2;
Информация представляется в виде предложений, содержащих конъюнкцию и импликацию. Ответ подразумевает дизъюнкцию полученных результатов. То есть, надо построить нечеткий ответ в виде соответствующего значения характеристической функции $\mu_\Sigma(z) = A_1(x_0) \wedge B_1(y_0) \wedge C_1(z) \vee A_1(x_0) \wedge B_1(y_0) \wedge C_2(z)$
а потом перейти к четкости. Рассмотрим все операции последовательно:

Итак, первая половина правил может быть представлена формально как
$A_1(x_0) \wedge B_1(y_0) = \alpha_1$ и
$A_1(x_0) \wedge B_1(y_0) = \alpha_2$
вторая как
$\alpha_1 \wedge C_1(z)$
$\alpha_2 \wedge C_2(z)$
где (x_0, y_0) - конкретные входные переменные;

3).Так как $\wedge = \min$, то находят уровни «отсечения» $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$ и $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$.

4). Теперь находят функции принадлежности *C*

$\alpha_1 \wedge C_1(z)$

$\alpha_2 \wedge C_2(z)$

5). Композиция (операция $\vee = \max$)

$\mu_\Sigma(z) = \alpha_1 \wedge C_1(z) \vee \alpha_2 \wedge C_2(z)$

6). Приведение к четкости

Центроидный метод: $z_0 = \dfrac{\int z\mu_\Sigma(z)dz}{\int \mu_\Sigma(z)dz}$ - центр тяжести = четкое значение

Алгоритм Tsukamoto -1) С помощью экспертов вводится нечеткость в форме нечетких множеств = функции принадлежности $A_1(x)$, $A_2(x)$, $B_1(y)$, $B_2(y)$, но здесь $C_1(z)$, $C_2(z)$ – это четкие, определенные функции, позволяющие найти четкие значения *z* по каждому из правил П1 и П2.

2). Находят уровни «отсечения» $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$ и $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$.

3).Из уравнений $\alpha_1 = C_1(z_1)$ и $\alpha_2 = C_2(z_2)$ находят четкие значения (ответы по применению отдельных правил) z_1, z_2.

4).Определяют ответ, например, как взвешенное среднее

Таким образом, начальные знания представлены в виде предложений с использованием конъюнкции (их используют на этапе получения уровней отсечения) и импликации (следования). Причем импликации могут не преобразовываться и использоваться непосредственно (на этом этапе определяются значения характеристических функций у нечетких множеств С). Далее, так как все предложения (для набора предложений, где использована конъюнкция) связаны дизъюнкцией, её и применяют для получения окончательного решения в нечеткой форме. Переход к четкой форме может различным, в частности таким, как предложен в приведенных традиционных примерах (алгоритмах).

Проблемы расширения возможностей интеллектуальных систем

Большая сложность в создании систем, способных справиться с вариативными и некорректными задачами. Для их решения понадобятся большие объемы информации, даже плохо структурированной, представляющей собой клудж образов и связей между ними. Следует только добиваться при наполнении такой информацией глобальных баз знаний максимальной добросовестности. Кроме того, видимо, следует ослабить ограничения на уровень несовместимости содержательной информации и правил. Полезно остерегаться информационного шума, который с развитием сетевой структуры, ускорения и увеличения объемов информационных потоков станет большой проблемой для информационных технологий и интеллектуальных систем в особенности, не исключая и самого человека.

За счет расширения и усложнения систем обратных связей надо добиваться создания в искусственной интеллектуальной среде развития механизмов, аналогичных способностям человека проводить мысленный эксперимент. Такое внутреннее воображение, когда система использует собственные информационные ресурсы в качестве начальных данных, должно активно развивать её внутренний мир, создавая новое знание, структурируя его и выискивая неточности и нестыковки. Не стоит бояться того, что система порой будет «впадать в задумчивость». Кроме того, обнаруженные нестыковки способны дать пищу для размышлений человеку, такому неловкому создателю интеллектуальных систем.

5.6.3.Нейронная сеть в представлении нечеткой логики

Норма и конорма. Определим следующие операции $\min(\mu_A, \mu_B)$; $\mu_A \cdot \mu_B$; $\max(0, \mu_A + \mu_B - 1)$. Они относятся к так называемой *треугольной норме (t-норма)*, если все – $\mu_A(x)$ находятся в интервале x [0,1].

Свойства нормы:

1. $T(0,0) = 0; T(\mu_A, 1) = T(1, \mu_A) = \mu_A$; - ограниченность;

2.монотонность;

3. $T(\mu_A, \mu_B) = T(\mu_B, \mu_A)$; коммутативность

4. $T(\mu_A, T(\mu_B, \mu_C)) = T(T(\mu_A, \mu_B), \mu_C)$; ассоциативность.

Для операций $\max(\mu_A, \mu_B)$; $\mu_A + \mu_B - \mu_A \cdot \mu_B$; $\min(1, \mu_A + \mu_B)$ полезно ввести понятие треугольной конормы (*t*-конорма).

Свойства конормы:

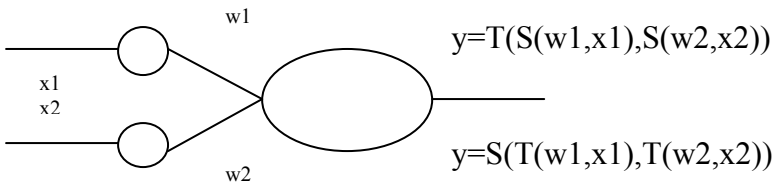
1. $S(1,1) = 1; S(\mu_A, 0) = S(0, \mu_A) = \mu_A$; - ограниченность;

2.монотонность;

3. $S(\mu_A, \mu_B) = S(\mu_B, \mu_A)$; коммутативность

4. $S(\mu_A, T(\mu_B, \mu_C)) = S(T(\mu_A, \mu_B), \mu_C)$; ассоциативность.

Нечеткие нейроны. Пусть x- входные сигналы, а w – синаптические веса нейрона.



Определим нейрон «И» как y=T(p1, p2) =T(S(w1,x1),S(w2,x2)), p1=S(w1,x1).

Определим нейрон «ИЛИ» как y=S(p1, p2) =S(T (w1,x1),T(w2,x2)), p1=T(w1,x1).

Представление нейронной сети с нечеткими нейронами (реализующей алгоритм Tsukamoto).

П1: если x1 естьL1, если x2 естьL2, если x3 естьL3, то y есть G,
П2: если x1 естьH1, если x2 естьH2, если x3 естьL3, то y есть P,
П3: если x1 естьH1, если x2 естьH2, если x3 естьH3, то y есть R.

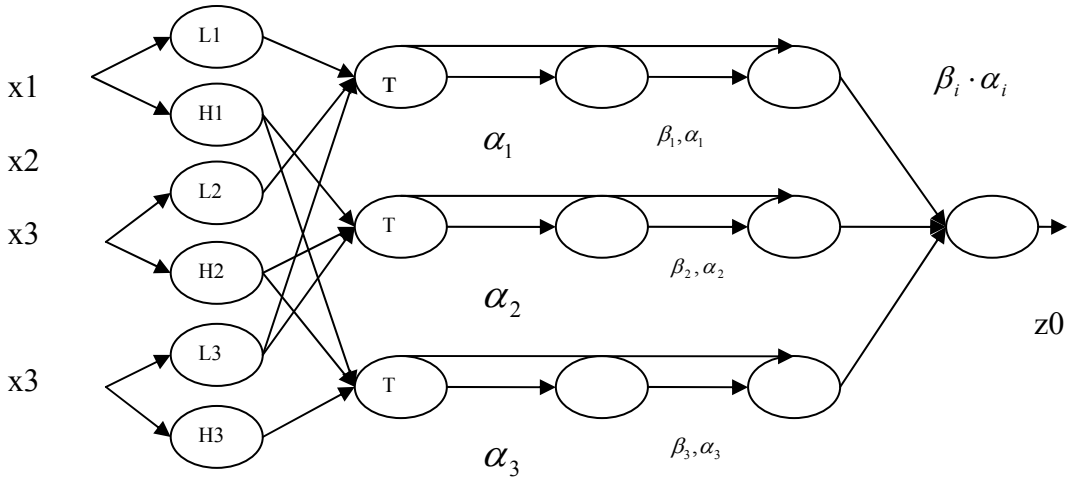
Определим уровни отсечения как (второй слой сети)

$$\alpha_1 = L_1 \wedge L_2 \wedge L_3,$$

$$\alpha_2 = H_1 \wedge H_2 \wedge L_3,$$

$$\alpha_3 = H_1 \wedge H_2 \wedge H_3.$$

Здесь уровни отсечения α – это есть результат действия нейронов «И», например минимум всех переменных L или H. Сами значения переменных L или H есть результат действия нейронов типа «ИЛИ», что в условиях этой задачи не конкретизировано.



Определим также отображения (четвертый слой сети)

$$z_1 = G^{-1}(\alpha_1)$$
$$z_2 = P^{-1}(\alpha_2)$$
$$z_3 = R^{-1}(\alpha_3)$$

где, например, G^{-1} - есть функция обратная функции G.
А также приведение к четкости

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3}{\alpha_1 + \alpha_2 + \alpha_3},$$

для чего полезно определить величины вида $\beta_i = \frac{\alpha_i}{\alpha_1 + \alpha_2 + \alpha_3}$ (третий слой сети)

5.7.Логический вывод для робота

5.7.1.Процедуры составления программы действий робота в системе STRIPS

Изучение этого вопроса рациональней начинать рассматривать на конкретных примерах.

Рассмотрим достаточно простую ситуацию, когда кубик С стоит на кубике А, а поверхность кубика В свободна, причем кубики А и В стоят на столе. Рука робота свободна. Таким образом? база данных следующая:
HANDEMPTY , CLEAR(B), CLEAR(C), ONTABLE (B), ONTABLE (A), ON(C,A).

П-правило. Напомним, что такое П-правило, которое использовалось ранее в разделе 5. 1.

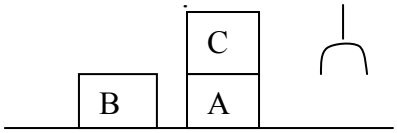
unstack(x,y)
P&D HANDEMPTY,CLEAR(x) ,ON(x,y) P= prediction, D = delete list A
HOLDING(x) , CLEAR (y) A= add formula

Это правило означает, что надо «взять x, который стоит на y (или на позиции y)». Первая строка – это предусловия применения правила (то есть условия, которые являются необходимыми для его выполнения): HANDEMPTY,CLEAR(x) ,ON(x,y). Вторая строка – это последствия применения этого правила: HOLDING(x) , CLEAR (y).

Технология решения задачи (система STRIPS)
с помощью П-правил

Рассмотрим ситуацию, когда кубик С стоит на кубике А, а поверхность кубика В свободна, причем кубики А и В стоят на столе. Рука свободна.

HANDEMPTY , CLEAR(B), CLEAR(C), ONTABLE (B), ONTABLE (A), ON(C,A)



Цель для робота поставить кубик С на кубик В, а кубик А сверху на кубик С:

$$ON(C,B) \wedge ON(A,C)$$

Построим ***последовательность выбора стратегии***:

1.Цель явно составная, поэтому она разбивается сразу же на подцели (связанные разумеется). Первая подцель ON(C,B), а вторая подцель ON(A,C).

2.Выбор с чего начинать, вообще говоря, случаен, но, допустим, начали с первой подцели.

3.Проверка предусловий.Проверяются соответствие правил из набора, данной подцели. Правило применяется (активируется), если его можно выполнить. Для выяснения возможности выполнить проверяются предусловия (выполнения) правила. Например, предусловиями (выполнения) правила **unstack** (x,y) – «взять x, который стоит на y (или на позиции y)»:

$$HANDEMPTY \wedge CLEAR(x) \wedge ON(x,y)$$

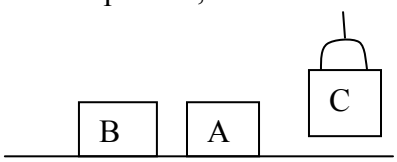
4. Поиск унификаций. База данных разрешает выполнение правила **unstack** (C,A) при следующей унификации x=C, y=A. Рука робота свободна, поверхность кубика чистая, он стоит на кубике A,:

$$HANDEMPTY \wedge CLEAR(C) \wedge ON(C,A)$$

5.Последствия применения правила **unstack** (x,y) приводят к тому, что рука после этого занята: HOLDING (x), а поверхность, на которой стоял x-очищена: CLEAR(y) . Эти изменения должны быть непременно согласованы с выполнением правила обязательно.

6.Сразу же отметим, что Операторы HOLDING (x) и HANDEMPTY конкурируют (противоречат друг другу) и тот, кто последний, стирает предыдущий. Точно также, операторы ON(C,A) и CLEAR(A) не могут существовать одновременно, остается только последний.

7.Возникает новое состояние (новая база данных). Новые элементы HOLDING (C), CLEAR(A), и старые CLEAR(B), ONTABLE (B), ONTABLE (A). Здесь же отметим, что те из элементов базы данных, которые не участвовали в операциях и переменные которых не упоминались в процедуре применения правила, остаются без изменения.



8.Теперь нужно искать новое правило, предусловия которого не противоречат новой базе данных и определяются первой подцелью. Например, для правила **stack** (x,y) – « x, который держит рука, поставить на y (или на позицию y)»

требуется, чтобы то, что нужно поставить на y было именно x, а поверхность y была свободна. Другими словами, в общем виде предусловие правила **stack** (x,y)

$$\text{HOLDING}(x) \wedge \text{CLEAR}(y)$$

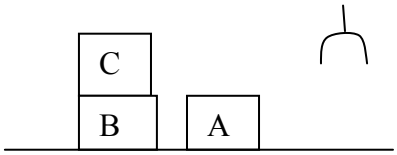
Это предусловие должно быть связано с применением соответствующего правила обязательно.

9.В случае унификации x=C, y=B, предусловия выполняются и правило **stack**(C,B) можно применять.

10.Последствия применения правила: из-за взаимного несоответствия HOLDING (x) и HANDEMPTY, остается только HANDEMPTY, а CLEAR(B) заменяется на ON(C,B) согласно этому правилу.

11. Возникает снова новое состояние (новая база данных)

Новые элементы HANDEMPTY, ON(C,B), подтверждается CLEAR(C), и остаются прежние CLEAR(A), ONTABLE (B), ONTABLE (A).



12. Важно, что в новой базе данных появляется первая подцель ON(C,B). Факт ее появления (локальное терминальное условие) останавливает все процедуры для разрешения первой подцели.

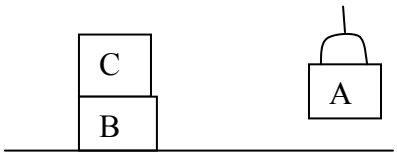
13. Теперь рассматривается вторая подцель ON(A,C). Для ее реализации нужно иметь свободную поверхность кубика C и свободную поверхность кубика A. То есть, CLEAR(A) и CLEAR(C). Но кубик A стоит на столе.

14.Теперь начинается поиск нового правила, то есть согласование предусловий правил с новой базой данных. Понятно, что подходит новое правило **pickup**(x)-«взять x со стола», причем с унификацией x=A.

15.Применяем **pickup**(A).

16. Последствия применения правила: из-за взаимного несоответствия HOLDING (x) и HANDEMPTY, остается только HOLDING (A), а CLEAR(A), ONTABLE (A) удаляются, согласно этому правилу.

17.Получаем новую базу данных. Новые элементы HOLDING (A), и остаются ON(C,B), CLEAR(C), ONTABLE (B).



18. Теперь ищется правило, предусловие которого соответствует второй подцели и последней базе данных. Согласно второй подцели, нужно поставить кубик A на кубик C, для этого нужно держать кубик A в руке и иметь свободную поверхность на кубике C. Есть такое правило – **stack** (x,y) – «x, который держит рука, поставить на y (или на позицию y)», ранее им пользовались. Кроме того, из базы данных следует, что предусловия этого правила при унификации x=A, y=C тоже выполнены.

19. Применяем **stack** (A,C).

20 Новая база данных: Новые элементы HANDEMPTY , ON(A,C), и остаются ON(C,B), ONTABLE (B). Здесь после действия правила появились новые элементы и удалено CLEAR(C), что отвечает результатам его применения. Появление в базе данных сразу двух подцелей, которые неявно связаны

конъюнкцией, отвечает полному терминальному условию. То есть, цель достигнута.

Таким образом, построен, так называемый, стек целей (последовательность применения правил)

{ **unstack** (C,A), **stack**(C,B), **pickup**(A), **stack** (A,C)}

Форма записи процедур обычно следующая

Таблица 5.13

HANDEMPTY ∧ CLEAR(C) ∧ ON(C,A) ∧ CLEAR (B)
unstuck(C,A)
HOLDING(C) CLEAR (A) CLEAR (B) HOLDING(C) ∧ CLEAR (B) ∧ CLEAR (A)
stack(C,B)
ON(C,B) ON(A,C) ON(C,B) ∧ ON(A,C)

*Перед первым правилом приведены все предусловия его применения (по отдельности и вместе). Записываются только основные литералы, те, которые не используются, не пишутся.

** Под применяемым правилом выписан результат его применения и предусловия применения следующего правила.

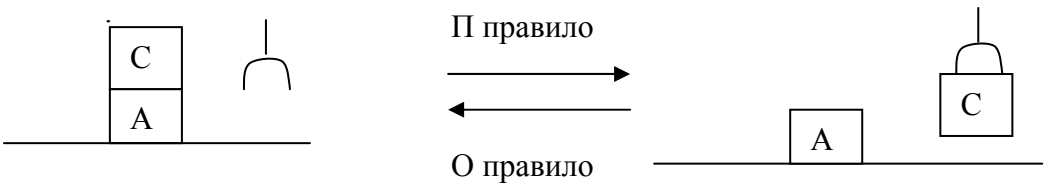
*** Внизу выписываются все подцели и цель полностью. В одну группу выделяются компоненты цели и составные цели.

Вообще говоря, можно заметить, что вызываются все правила, предусловия которых отвечают базе данных и первой подцели. Попробуйте сами составить полный стек целей в схеме решения, которая представлена в этом разделе.

5.7.2.Конструирование программы действий робота с помощью О-правил

Прямой метод перебора правил, рассмотренный выше, приемлем, если их немного, если же база данных и правил обширна, то можно поступить иначе - ввести т.н. О-правила.

О-правило. Можно ввести понятие О-правила, Если, в рамках того же примера, задана цель CLEAR(A) и следует найти П-правило, применение которого привело к этой цели. Нетрудно видеть, что это П-правило может быть **unstack(x,A).**Таким образом, О-правило – это правило, обратное **unstack(x,A).**



Процедура поиска О- правила. Если задана цель CLEAR(A), проверяются все П-правила, которые содержат в списке добавлений CLEAR(x), например, **unstack(x,y).** Его предусловия HANDEMPTY ∧ CLEAR(x) ∧ ON(x,y). При унификации y= A, предусловия будут иметь вид HANDEMPTY ∧ CLEAR(x) ∧ ON(x,A), где x – неопределено. Если есть ситуация, когда на кубике A стоит нечто x, подходит О- правило, обратное **unstack(x,A).**

Регрессия. Процедура, позволяющая провести поиск О-правил, называется регрессией. Регрессия проверяет, соответствуют ли возникающему текущему состоянию

базы данных определенные подцели или условия.

Пример регрессии

Рассмотрим, что такое регрессия на прежнем примере. Пусть кубик С стоит на кубике А, а кубики А и В стоят на столе, при этом рука робота свободна: HANDEMPTY , CLEAR(B), CLEAR(C), ONTABLE (B), ONTABLE (A), ON(C,A)- это начальная база данных.

Рассмотрим правило

***unstack**(x,y)*

P&D HANDEMPTY,CLEAR(x) ,ON(x,y)

A HOLDING(x) , CLEAR (y)

*Регрессия для HANDEMPTY (согласованная с базой данных) на правиле **unstack**(x,y), – это F (ложь). То есть, если дописать HANDEMPTY снизу, например, в качестве предусловия для удовлетворения следующего правила, то оно не будет согласовано с правилом **unstack**(x,y), точнее с последствиями его применения на базе данных.*

Регрессия ONTABLE (C) (согласованная с базой данных) – это F (ложь). То есть, и в невозмущаемой применением правила части начальной базы данных и в последствиях применения правила это не реализуется.

Регрессия для CLEAR (A) – это T (истина). Так как кубик С снимается с кубика А.

Регрессия для CLEAR (B) – это CLEAR (B). Так как кубик В не трогают, он продолжается стоять так, как стоял.

Механизм формирования программы действий робота с использованием регрессии. О-правила используются системой для создания макета (непроверенного списка, структуры) программы действий робота, который затем уточняется и проверяется.

1.Цель разбивается на подцели (связанные конъюнкцией).

2.К первой полцели последовательно применяются подходящие О-правила, (используя истинность регрессии каждой конкретной цели на рассматриваемом правиле) пока система не придет в соответствие с начальной базой данных.

3.Потом включается процедура проверок и согласования.

4. Затем все повторяется для следующей подцели…

Первые два пункта процедуры, выписанной выше, представлены ниже в таблице 5.14.

Таблица 5.14

HANDEMPTY ∧ CLEAR(B) ∧ **CLEAR(C)** ∧ ∧ ∧ ONTABLE (B) ∧ ONTABLE (A) ∧ **ON(C,A)**
CLEAR(B) ∧ ∧ **HANDEMPTY** ∧ **CLEAR(C)** ∧ **ON(C,A)**
unstuck(C,A)
CLEAR(A) ∧ ∧ **HOLDING(C)** ∧ CLEAR (B)
stack(C,B)
ON(C,B) ∧ HANDEMPTY
ON(C,B) ON(A,C) ON(C,B) ∧ ON(A,C)

В верхней строке таблицы приведена начальная база данных, в нижней строке -

подцели и полная цель задачи. Здесь выделена подцель **ON(C,B)**[9]. Подцель ON(C,B) - это часть результата применения правила, в данном случае подобранного по этому критерию **stack**(C,B), то есть ON(C,B) ∧ HANDEMPTY, а предусловие его выполнения HOLDING(C) ∧ CLEAR (B). Регрессия ON(C,B) на правиле **stack**(C,B) дает T, то есть такое правило подходит.

HOLDING(C) - это результат действия другого правила, например, **unstack**(C,A). Регрессия HOLDING(C) на правиле **unstack**(C,A) дает тоже T, а регрессия еще одной необходимой части предусловия CLEAR (B) на этом же правиле дает CLEAR (B)[10]. То есть, и в этом случае применение такого правила оправданно.

5.7.3. Технология решения задачи в системе RSTRIPS

При выборе второй подцели, решение с удалением элементов базы данных приведет к очевидным трудностям, заводя систему в тупик. Именно такие проблемы и привели к модернизации системы **STRIPS.** Такой модернизированной системой стала **RSTRIPS.**

Рассмотрим основные особенности этой системы.

Маркер и сохранение предистории. Форма записи процедур перед применением первого правила немного изменяется

Таблица 5.15

HANDEMPTY ∧ CLEAR(C) ∧ ON(C,A)
unstuck(C,A)
*HOLDING(C)_____ CLEAR (B) HOLDING(C) ∧ CLEAR (B)
stack(C,B)
ON(C,B) ON(A,C) ON(C,B) ∧ ON(A,C)

*Под условием HOLDING(C) размещают маркер _____. Маркер показывает, что только что выполнено HOLDING(C).

**Все правила выше маркера были применены.

***Условия ниже маркера проверяются прямо сейчас.

****Маркер, входя в зону действия правила, оставляет вверху достигнутые цели, а внизу цели, которые предстоит достигнуть. Важно, что достигнутая подцель *HOLDING(C), помечена звездочкой. Система запрещает её вычеркивать или делать ложной, то есть, применяет защиту этой цели. *Если применение следующего правила коррелирует с защищенной подцелью, то зашита снимается*, а если не коррелирует (то есть, отмеченная звездочкой подцель не зависит от применения правила), то защита остается.

Ранее в **STRIPS** применялась процедура вычеркивания, то есть замена предыдущей базы данных другой БД , обновленной в результате применения правил. Здесь, в системе

---

[9] Выбор этой подцели может быть случайным или определенный дополнительным уточняющим условием (которое следует из полной цели) типа ONTABLE (B), то есть кубик B должен стоять на столе, поэтому все перемещения начинаются с него, значит выбирается та подцель, которая подразумевает позицию кубика B внизу, то есть **ON(C,B).**

[10] Вообще говоря, применение регрессии к правилу добавляет результат регрессии к списку предусловий этого правила явно или неявно. Здесь регрессия HOLDING(C) подтверждает список предусловий и ничего записывать не надо, а регрессия CLEAR(B) и равна CLEAR(B), потому требует записи перед этим правилом. Однако, если этот литерал записан в той части начальной базы данных, которая не изменилась перед применением правила, то такая запись вообще говоря, не обязательна, ибо подразумевается по умолчанию.

RSTRIPS, нет необходимости вычеркивания, ибо последовательность применения операций определяется положением маркера. Кроме того, система «помнит» все предшествующие операции и прежние базы данных.

Система должна проверить два типа условий. Первое условие (шаг назад) – соответствуют ли предусловия[11] подобранного системой нового правила состоянию, которое получается после применения предыдущего, уже выполненного только что правила[12]. При необходимости рассматривают и второе условие (шаг вперед) - выбранное новое правило не должно нарушать ранних защищенных подцелей.

Процедура, позволяющая провести проверку выполнения таких условий, - это регрессия. Регрессия проверяет, соответствуют ли возникающему текущему состоянию базы данных определенные подцели или условия. Пока достаточно отметить, что проведение регрессии с шагом назад и шагом вперед позволяют перейти к применению следующего правила.

Прямая процедура создания программы в системе RSTRIPS. Теперь рассмотрим полностью процедуру решения. Цель ON(C,B) $\wedge$ ON(A,C) разбивается на подцели: первая подцель - ON(C,B), вторая подцель ON(A,C). Первая подцель ON(C,B) из базы данных HANDEMPTY, CLEAR(B), CLEAR(C), ONTABLE (B), ONTABLE (A), ON(C,A) находит предусловия правила **unstack**(x,y), то есть HANDEMPTY, CLEAR(C), ON(C,A). При унификации x=C, y=A. Это правило становится во главе списка. Маркер после результата действия правила перемещается ниже результата HOLDING(C) – этот литерал помечается звездочкой, что означает его защиту.

Система ищет новое правило, предусловие которого включает HOLDING(C) и литерал с переменной B, то есть CLEAR(B) и ONTABLE (B). Таким образом, выполнены предусловия правила **stack**(x,y) при унификации x=C, y=B.

Однако для применения этого правила, следует прежде найти регрессию CLEAR (B) на предыдущем правиле **unstack**(C,A). Регрессия равна CLEAR (B), то есть явно не ложь, что разрешает применение следующего правила **stack**(C,B).

Таблица 5.16

HANDEMPTY $\wedge$ CLEAR(C) $\wedge$ ON(C,A)
unstack(C,A)
*HOLDING(C) CLEAR (B) HOLDING(C) $\wedge$ CLEAR (B)
stack(C,B)
ON(C,B) ON(A,C) ON(C,B) $\wedge$ ON(A,C)

Одновременное выполнение всех предусловий HOLDING(C) $\wedge$ CLEAR (B), двигает маркер ниже и включает правило **stack**(C,B). Маркер после применения правила перемещается и останавливается ниже результата применения этого правила ON(C,B). Защита с HOLDING(C) снимается. Теперь подцель - ON(C,B) достигнута и защищена – перед ней ставят звездочку.

Таблица 5.17

HANDEMPTY $\wedge$ CLEAR(C) $\wedge$ ON(C,A)
unstack(C,A)

[11] Речь идет о предусловиях, которые не есть следствия применения предыдущего – выполненного только что, - правила, а взяты из ранней (или даже начальной, в той ее части, которая не изменилась) базы данных.
[12] Если это условие нарушено, система будет искать правила, действие которых введет нужное предусловие.

HOLDING(C) CLEAR (B) HOLDING(C) ∧ CLEAR (B)
stack(C,B)
*ON(C,B) HANDEMPTY ∧ CLEAR (A) ∧ ONTABLE (A)
pickup(A)
*HOLDING(A) CLEAR (C) HOLDING(A) ∧ CLEAR (C)
stack(A,C)
ON(A,C) ON(C,B) ∧ ON(A,C)

Теперь первая подцель выполнена. Система выбирает вторую подцель ON(A,C). Ищутся правила, предусловия которых отвечают базе данных и содержащие переменные A и C. Одно из них **pickup**(x) при унификации x=A имеет все необходимые и достаточные предусловия

HANDEMPTY ∧ CLEAR (A) ∧ ONTABLE (A)

В результате его действия маркер двигается дальше и опускается ниже результата применения правила HOLDING(A), на которых ставят защиту. Но защита с ON(C,B) не снимается, ибо правило **pickup**(A) не коррелирует (нет общих переменных) с ON(C,B). Система ищет правило для предусловий которого имеет место защищенное HOLDING(A) и те компоненты базы знаний, которые имеются в наличии. Это может быть CLEAR (C), и тогда общее предусловие HOLDING(A) ∧ CLEAR (C). Оно соответствует правилу **stack**(x,y) при унификации x = A, y = C. Система его выполняет и маркер опускается ниже результата применения правила **stack**(A,C), то есть ON(A,C).

Одновременное выполнение всех условий сдвигает маркер ниже ON(C,B) ∧ ON(A,C). Так как дальнейшего продолжения нет - все цели выполнены, - формирование программы закончено.

5.7.4.Решение проблемы взаимодействия целей

Рассмотрим то же начальное состояние, когда кубик C стоит на кубике A, а поверхность кубика B свободна, причем кубики A и B стоят на столе. Рука свободна.
HANDEMPTY , CLEAR(B), CLEAR(C), ONTABLE (B), ONTABLE (A), ON(C,A).
Новая цель для робота поставить кубик A на кубик B, а кубик B сверху на кубик C:

ON(A,B) ∧ ON(B,C)

Аналогично предыдущему легко построить стек целей [13]

{**unstack**(C,A), **putdown**(C), **pickup** (A), **stack** (A,B)},

при этом подцель ON (A,B) достигнута и защищена, аналогично тому, как это было сделано в предыдущем разделе. Маркер опустился ниже литерала ONTABLE (B), также защищенного.

Таблица 5.18

1	*ONTABLE (A) ∧ ONTABLE (B) ∧ CLEAR(B) ∧* *∧* **HANDEMPTY ∧ CLEAR(C) ∧ ON(C,A)**
2	**unstuck***(C,A)*
3	*ONTABLE (A) ∧ ONTABLE (B) ∧ CLEAR(B) ∧ CLEAR(A) ∧*

[13] Жирным шрифтом выделены основные литералы, которые входят в результат и в предусловия правил. Причем курсивом те из них, которые не являются предусловиями выполнения последующего правила.

	∧ **HOLDING(C)**
4	**putdown**(C)
5	*ONTABLE (B)* ∧ *CLEAR(B)* ∧ *ONTABLE (C)* ∧ *CLEAR(C)* ∧ ∧ **ONTABLE (A)** ∧**CLEAR(A)** ∧**HANDEMPTY**
6	**pickup**(A)
7	*ONTABLE (B)* ∧ *ONTABLE (C)* ∧ *CLEAR(C)* ∧ ∧ **HOLDING(A)** ∧CLEAR(B)
8	**stack**(A,B)
9	*ONTABLE (C)* ∧ *CLEAR(C)* ∧*CLEAR(A)* ∧ ∧ ***ON (A,B)*** ∧ **ONTABLE (B)* ∧ ∧ **HANDEMPTY** ∧CLEAR(z) ∧ON(z,B)
10	**unstuck**(z,B)
11	*ONTABLE (C)* ∧ *CLEAR(C)* ∧ **ONTABLE (B)* ∧ ***CLEAR (B)*** ∧ **HOLDING(z)**
12	**putdown** (z)
13	*ONTABLE (C)* ∧ *CLEAR(C)* ∧*CLEAR (z)* ∧ *ONTABLE (z)* ∧ ∧ ONTABLE (B) ∧CLEAR (B) ∧ **HANDEMPTY**
14	**pickup**(B)
15	*ONTABLE (C)* ∧*CLEAR (z)* ∧ *ONTABLE (z)* ∧ ∧ **HOLDING(B)** ∧CLEAR (C)
16	**stack**(B,C)
17	*ONTABLE (C)* ∧*CLEAR (z)* ∧ *ONTABLE (z)* ∧ ***CLEAR (B)*** ∧ ∧ **ON(B,C)** ON(A,B) ∧ ON(B,C)

Вплоть до верхних строк 9 раздела программы система действует уже известным из прежнего рассмотрения образом. Так как первая цель достигнута, система начинает добиваться второй подцели ON(B,C), для чего ей нужно освободить поверхность кубика B. Но при этом нарушается защищенная подцель**ON (A,B)*.* Таким образом, вторая подцель ON(B,C) - это подцель, нарушающая защищенность. Система столкнулась с явлением ***взаимодействия целей***.

Это значит, что для продолжения работы, необходимо разрушить часть прежних достижений. Именно поэтому рационально сначала создать план работы, его согласовать, очистить от ненужных действий , сократить, оптимизировать, а только потом выполнять. Поэтому часто создание программы для робота называют **созданием планов**.

Возвращаясь к STRIPS

Если использовать унификацию z = A, то дальнейшие действия системы вполне логичны и это полностью отвечает принципам действия системы **STRIPS.** Понятно, что маркер и защита в этом случае отсутствуют. Можно увидеть, что отказываясь от достигнутой первой цели система **STRIPS** переключается на выполнение второй. Добившись реализации в разделе 17 второй цели система **STRIPS** может вернуться к выполнению первой цели, причем её она уже достигнет без особого труда. Читателю предлагается самостоятельно выполнить эту часть задачи.

Прежде чем обсудить особенности поведения системы **RSTRIPS** при взаимодействии целей, обратим внимание на ряд важных деталей. 8 и 10 разделы определяют выполнение двух операций прямой и обратной, то есть **stack**(A,B) и **unstuck**(z,B), если использовать унификацию z = A. Соответственно при этой унификации

база данных до применения правила **stack**(A,B) (раздел7) и база данных, полученная после применения правила **unstuck**(A,B) совпадают. Вообще говоря, эти операции можно исключить, убрав из списка разделы 8-11. Но это возможно, только при данной унификации. Точно также, база данных до применения правила **pickup**(A) (раздел 5) совпадает при той же унификации z = A с базой данных полученной после применения обратного правила **putdown** (z) (раздел 13). И в этом случае результат не изменится, если исключить из списка разделы 6-13. Вообще говоря, при данной унификации можно разными программными способами обеспечить такую процедуру удаления комбинаций прямых и обратных операций (при необходимости убеждаясь в неизменности базы данных после подобных исключений), если система к ним приходит.

Система **RSTRIPS** эту операцию выполняет следующим образом. Отвлекаясь от важных процедурных вопросов, убеждаемся в том, что система в 9 разделе сталкивается с проблемой решения второй подцели и проводит регрессию защищенной цели *ON (A,B) на новом правиле **unstuck**(z,B). Невозможность выполнения защищенной цели при этом убирает не только новое правило, но и предыдущее **stack**(A,B), удаляя из списка разделы 8-11. Защита вместе с маркером переносится в обратном направлении, то есть вверх.

Теперь защита приложена к литералу HOLDING(A) **.** Проверяя регрессию этого литерала на следующем правиле **putdown** (z) при той же унификации система убеждается в невозможности этого выполнения и убирает оба правила- прямое **pickup**(A) и обратное **putdown** (z), удаляя из списка разделы 6-13.

Таким образом, список сокращается и движению маркера вниз ничего не препятствует:

Таблица 5.19

1	*ONTABLE (A) ∧ ONTABLE (B) ∧ CLEAR(B) ∧* **HANDEMPTY ∧ CLEAR(C) ∧ ON(C,A)**
2	**unstuck***(C,A)*
3	*ONTABLE (A) ∧ ONTABLE (B) ∧ CLEAR(B) ∧ CLEAR(A)* **HOLDING(C)**
4	**putdown**(C)
5	*ONTABLE (B) ∧ CLEAR(B) ∧* **ONTABLE (C) ∧ *CLEAR(C) ∧* **ONTABLE (A) ∧ CLEAR(A) ∧ HANDEMPTY**
6(14)	**pickup**(B)
7(15)	*ONTABLE (C) ∧ CLEAR (z) ∧ ONTABLE (z)* **HOLDING(B)** ∧ CLEAR (C)
8(16)	**stack**(B,C)
9(17)	*ONTABLE (C) ∧ CLEAR (z) ∧ ONTABLE (z) ∧* ***CLEAR (B)*** **ON(B,C)** ON(A,B) ON(A,B) ∧ ON(B,C)

Дальнейшие процедуры формирования списка предоставляем читателю.

Что лучше: сразу делать, а потом переделывать, или сначала подумать…
Первая система, как мы видим, решает задачу и если что-то не так, просто переделывает. Действительно, **STRIPS** сначала добивается первой подцели, потом переходит к достижению второй подцели. Но при этом нарушает первую подцель, чтобы потом, после достижения второй подцели, вернуться к разрешению первой. Вторая система - **RSTRIPS**, - сначала создает программу действий (план), потом ее корректирует, прежде чем начинать выполнять действия.

5.7.5. Представление программы в виде графа. Декомпозиции графа.

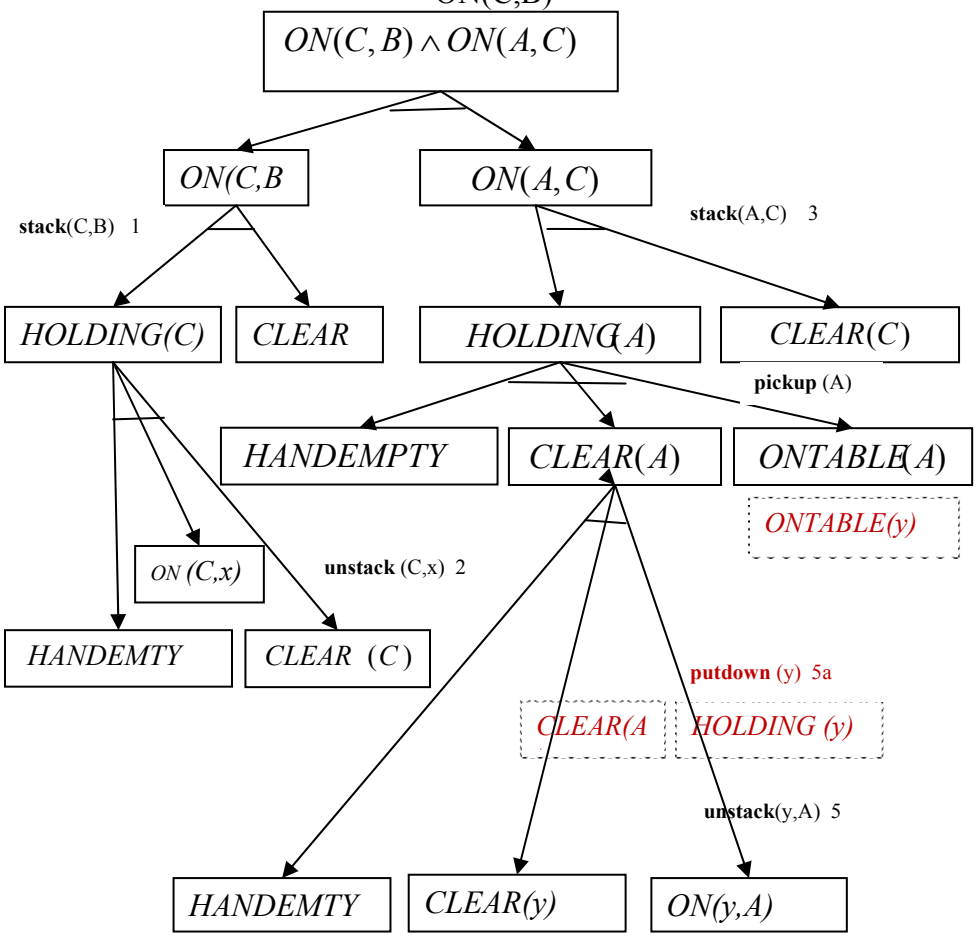
Система DCOMP

Эта система создает «временное решение в ***виде графа типа И/ИЛИ***. Потом этот граф корректируется для снятия проблем со взаимодействиями. Рассмотрим уже известную ситуацию, когда кубик С стоит на кубике А, а поверхность кубика В свободна, причем кубики А и В стоят на столе. Рука свободна.

HANDEMPTY , CLEAR(B), CLEAR(C), ONTABLE (B), ONTABLE (A), ON(C,A).

Цель для робота поставить кубик С на кубик В, а кубик А сверху на кубик С:

ON(C,B) ∧



ON(A,C)

Рис. 5.12. Построение графа «временного» решений. Пунктиром вставлена «заплатка».

О- правила применяют к целевым вершинам, а концевые вершины соответствуют начальным условиям. Полученное «временное» решение нуждается в коррекции.

Заплатки. Прежде всего, обратим внимание, что в структуре такого временного решения возможны нарушения. Часть графа, которая формируется из правой подцели, может игнорировать операции (действия), которые не имеют отношения в частной программе решения. Таким образом, следствием действия 5 **unstack** (y,A), являет процедура очистки поверхности кубика А от постороннего предмета, который не интересует решающую систему.

Но тогда возникает противоречие, определяемое процедурой регрессии, которое состоит в том, что HANDEMPTY дает ложь при регрессии на правиле 5 **unstack** (y,A), которое в качестве П-правила действует снизу вверх. Напомним, что регрессия проверяет соответствует ли HANDEMPTY во второй строчке снизу рисунка последствиям применения правила **unstack**(y,A) к нижней строчке правой части рисунка. Ранее рука была свободна, после применения правила она должна быть занята, а этого нет.

Решением проблемы является введение дополнительного правила 5а **putdown** (y) ,

которое освобождает руку робота и снимает это противоречие. При этом регрессия HANDEMPTY на правилах оказывается истинной. Таким образом, система DCOMP способна везде во «временных» планах наложить «заплатки».

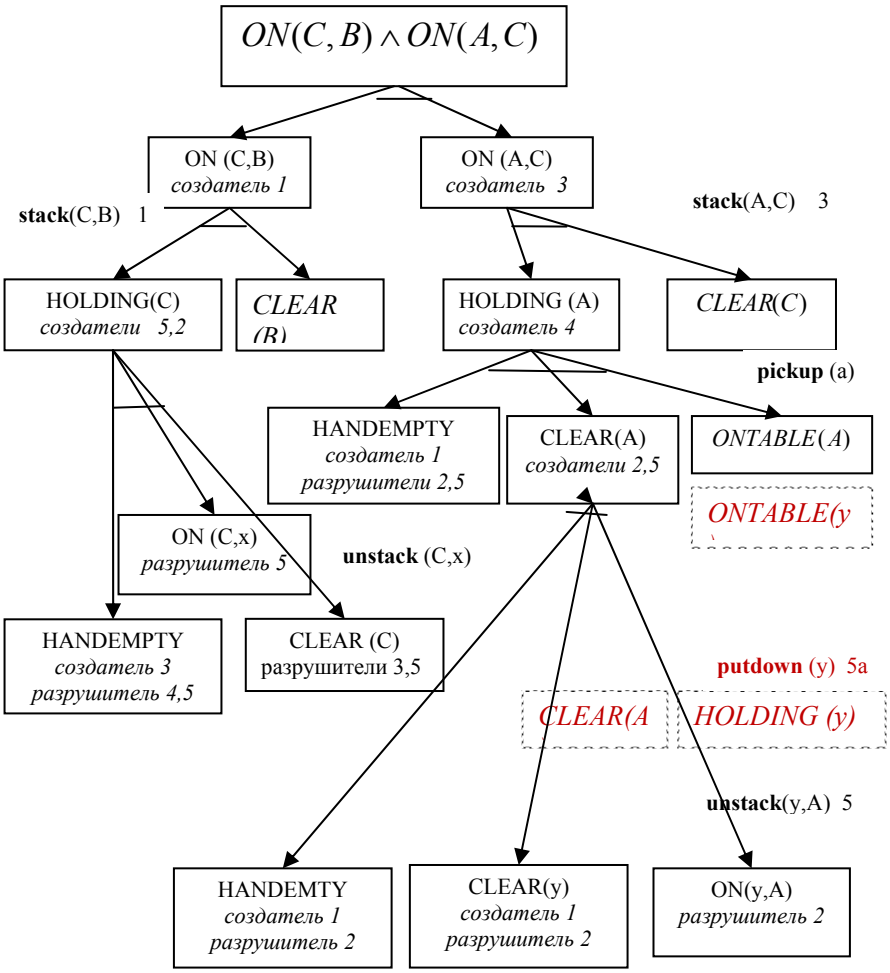


Рис. 5.13. Формирование множеств правил «создателей» и «разрушителей» для декомпозиции графа.

Выбор последовательности действий. Другой важной особенностью системы DCOMP является способность выбора последовательности действий. Для этого в системе предусмотрено создание двух множеств П-правил, отвечающих литералам C_{ij}, используемым для предусловий правил. Определим C_{ij} как *i*-тый литерал предусловия *j*-того правила. Одно множество «разрушителей» (для *i*-того литерала, который входит в предусловие *j*-того правила) и другое – «создателей». ***Разрушитель*** способен нарушить предусловие этого (*j*-того) правила. ***Создатели*** добавляют C_{ij} и не являются ни предшествующими *j*-тому правилами, ни самим *j*-тым правилом.

Следует обратить внимание на важную особенность. Если правило 2 (которое является разрушителем для литерала CLEAR (C), и которое есть предусловием для правила 2) будет предшествовать в программе правилу 3, то правило 3 не будет вычеркивать этот литерал (условие).

То есть, на этой основе можно создать метод формирования последовательности применения правил.

Если вставить правило 2 (при x=A) перед правилом 4, то CLEAR(A) выполняется, и правило 5 вообще можно исключить (цель 5 правила – очистка поверхности кубика A). Но выполнять правило 4 после правила 2 нельзя, регрессия показывает ложные значения для литерала HANDEMPTY на правиле 2. Поэтому нужно воспользоваться правилом -

создателем HANDEMPTY, вставляя его в виде «заплатки» между правилом 2 и правилом 4. Эта «заплатка» - правило 1 **stack**(C,B). Теперь все проблемы решены. Последовательность действий: выполнение подряд правил 2,1,4,3.

Вопросы для самоконтроля:

1.Как можно избежать «комбинаторного взрыва»? Предложите приемлемые концепции.
2. Изложите преимущества и недостатки коммутативных систем продукций.
3. В чем слабости Байесовской системы логического вывода?
4. Изложите принцип (схему работы) Байесовской системы логического вывода.
5. Как организована система действий робота?
6. В действиях робота видна ли интеллектуальная сторона деятельности, или это просто вид алгоритма?
7. Почему все (правильно составленные) предложения в исчислении предикатов содержат только дизъюнкции? Или конъюнкции?
8. Проблемы конструктивных и неконструктивных решений в исчислении предикатов.
9. Применение кванторов и сколемовские функции. Почему их нельзя просто назвать функциями?
10. Поясните принцип резолюции на конкретных примерах. Что делать, если переменные в одинаковых литералах разные?
11. Как записываются высказывания на графе?
12. Как переписать все виды предикатов в бинарные?
13. Как переписать сложное предложение на языке предикатов в виде функционального фрейма?
14. Смысл характеристической функции в системах с нечеткой логики. Как измениться эта функция при переходе к логике предикатов?
15. Поясните разницу между базовыми процедурами конъюнкции и дизъюнкции в логике предикатов и в системе нечеткой логики.
16.Как построить логический вывод (схему) на основе нечеткой логики?
17. Процедуры приведения к четкости. Что они означают и насколько они корректны?

5.8.Структурированные объекты

5.8.1.Общие представления

Блок - структура, объединяющая множество элементов описания, предикатов (структура представления) и некоторые сведения о характере представления и особенностях вычислений, относящихся к одной области приложения.

Это позволяет

- Упростить доступ. При необходимости вся группа сразу является доступной для операций.
- Упрощение ввода информации целыми блоками.
- Упрощение вычислений.

Рассмотрим блок информации в форме обычного языка

Евгений дал Маше книгу
Евгений – программист
Маша – юрист
Адрес Евгения: улица Сумская 37

Эту информацию можно представить в виде двух блоков

Евгений
ДАТЬ (Евгений, Маша, книга)
ЗАНЯТИЕ(Евгений, программист)
АДРЕС(Евгений, улСумская37)

Маша
ДАТЬ (Евгений, Маша, книга)
ЗАНЯТИЕ(Маша, юрист)

Трехаргументный предикат ДАТЬ (Евгений, Маша, книга) можно записать в виде бинарных предикатов:

$(\exists x)$ [ЭЛЕМЕНТ(x,ПЕРЕДАЧИ) ^ ДАЮЩИЙ(x, Евгений) ^ ПОЛУЧАЮЩИЙ(x, Маша) ^
^ ОБЪЕКТ(x, книга)];

Если представить событие передачи как ПЕР1, (это может внести в описание время, определяя номера передач, располагая по порядку их с течением времени) то исключив квантор,

[ЭЛЕМЕНТ(ПЕР1,ПЕРЕДАЧИ) ^ ДАЮЩИЙ(ПЕР1, Евгений) ^ ПОЛУЧАЮЩИЙ(ПЕР1, Маша) ^
^ ОБЪЕКТ(ПЕР1, книга)];

Переход от предикатов к функциям. Или записать не в виде предикатов, а в виде функций над множеством ПЕРЕДАЧИ (предикат РАВ - это «равенство»):

ЭЛ (ПЕР1, ПЕРЕДАЧИ) ^
^ РАВ[*дающий* (ПЕР1), Евгений] ^
^ РАВ[*получающий*(ПЕР1), Маша] ^
^ РАВ[*объект*(ПЕР1), Книга] ^

Из-за временного характера все описания ниже представлены в терминах событий и ситуаций. Введем предикаты ПМ - «первое множество есть подмножеством второго», Эл- «нечто является элементом данного множества», Род занятий – РЗ, адрес - АДР.

ПЕР1
ЭЛ(ПЕР1, ПЕРЕДАЧИ)
РАВ[*дающий* (ПЕР1), Евгений]
РАВ[*получающий*(ПЕР1), Маша]
РАВ[*объект*(ПЕР1), Книга]

Р31

ЭЛ(Р31,РОД_ЗАНЯТИЙ)

РАВ[*работающий* (Р31), Евгений]

РАВ[*профессия(*Р31), Программист]

Р32

ЭЛ(Р32,РОД_ЗАНЯТИЙ)

РАВ[*работающий* (Р32), Маша]

РАВ[*профессия(*Р32), Юрист]

АДР1

ЭЛ(АДР1бАДРЕСА)

РАВ[человек (АДР1), Евгений]

РАВ[место(АДР1), улСумская37]

Можно упростить, например первое выражение

ПЕР1

дающий: Евгений

получающий: Маша

объект: Книга

Здесь «дающий» - имя ячейки, «Евгений»- значение ячейки этого блока. Значением может быть функциональное выражение, которое может относиться к имени ячейки другого блока.

Рассмотрим два предложения: «Евгений дал Маше книгу» и « Боря дал ручку тому лицу, которому Евгений дал книгу». Функции позволяют вводить функцию от функции (как предикаты второго и выше порядков, их стараются не использовать).

ПЕР1

ЭЛ(ПЕР1, ПЕРЕДАЧИ)

дающий: Евгений

получающий: Маша

объект: Книга

ПЕР2

ЭЛ(ПЕР2, ПЕРЕДАЧИ)

дающий: Боря

получающий: получающий (ПЕР1)

объект: Ручка

Сколемовские константы и функции, псевдозначения. Обычно стараются упростить все формы записи. Предложение «Некто дал Маше книгу» содержит предикат, например, $(\exists x)$ РАВ[*дающий (ПЕР3), x*]. Можно использовать сколемовскую переменную РАВ[*дающий (ПЕР3), S*]. Этот некто может быть человеком, тогда нужно это отметить: $(\exists x)$ {РАВ[*дающий (ПЕР3), x*] $\wedge$ ЭЛ(x,ЛЮДИ) }, а можно проще: ЭЛ(*дающий (ПЕР3)*,ЛЮДИ).

Можно ввести псевдозначение (*элемент множества ЛЮДИ*) - это будет считаться неопределенным значением ячейки.

Можно использовать ячейку «сам», где объект сам по себе элемент некоторого множества.

ПЕР1

сам:(*элемент множества* ПЕРЕДАЧИ)

дающий :Евгений

получающий: Маша

объект: Книга

Р31

сам:(*элемент множества* РОД_ЗАНЯТИЙ)

работающий: Евгений

профессия: Программист

Р32

сам:(*элемент множества* РОД_ЗАНЯТИЙ)

работающий : Маша

профессия: Юрист

АДР1

сам:(*элемент множества* АДРЕСА)

человек: Евгений

место: ул Сумская37]

Остальные элементы: Евгений сам:(*элемент множества* ЛЮДИ)

Маша сам:(*элемент множества* ЛЮДИ)

Книга сам:(*элемент множества* ФИЗИЧ_ОБЪЕКТЫ)

Программист сам:(*элемент множества* РАБОТЫ)

Юрист сам:(*элемент множества* РАБОТЫ)

улСумская37 сам:(*элемент множества* МЕСТОЖИТЕЛЬСТВО)

ЛЮДИ сам:(*подмножество множества* ЖИВОТНЫЕ)

Пример

«Евгений каждому что-то дал»

$(\forall x)\ (\exists y)\ (\exists z)$ [ЭЛ(y,ПЕРЕДАЧИ) ^ РАВ (дающий (y), Евгений) ^ РАВ (обьект(y), z) ^

^ РАВ(получающий (y), x)];

g(x) сам:(*элемент множества* ПЕРЕДАЧИ)

дающий :Евгений

получающий: x

объект: sk(x)

где sk(x)- сколемовская функция.

Семантические (смысловые) связи. Если акцентировать внимание на организации связей, то следует добавить функции дополнения, -до, пересечение, объединение, например:

Пример *множество-из* (Евгений, Маша, Боря)

Пример

Высказывания: «Евгений купил автомобиль», «Это был «Форд» или «Шевроле», «Он был с откидным верхом»

ПОК1 сам:(*элемент множества* ПОКУПКИ)

покупатель: Евгений

куплено : (элемент- множества пересечение (объединение(ФОРДЫ, ШЕВРОЛЕ),

дополнение-до(С_ОТКИДНЫМ_ВЕРХОМ))).

Пример

«Евгений дал книгу Боре или Маше»

ПЕР1

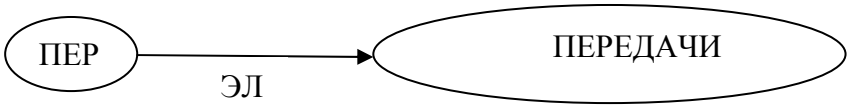
сам:(*элемент множества* ПЕРЕДАЧИ)

дающий :Евгений

получающий: (элемент множества множество-из(Боря, Маша))
объект: Книга

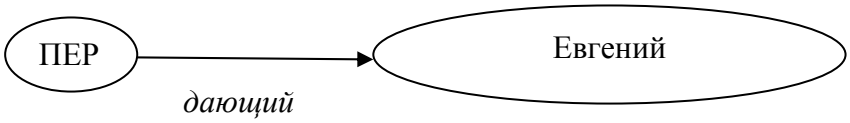
5.8.2.Семантические сети

Выражение ЭЛ(ПЕР1, ПЕРЕДАЧИ) можно представить как



Для выражения РАВ[*дающий* (ПЕР1), Евгений] или в упрощенном виде для блока ПЕР1

дающий: Евгений



Для сколемовского представления функции sk(x)

g(x) сам:(*элемент множества* ПЕРЕДАЧИ)
дающий :Евгений
получающий: x
объект: sk(x)

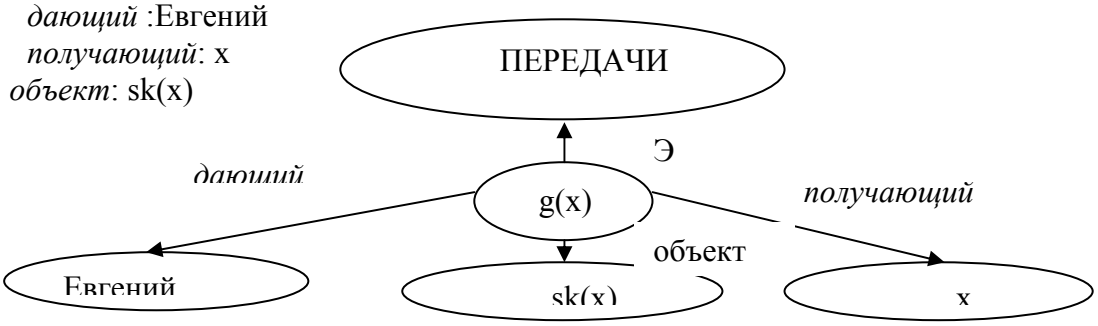


Рис.5.14. Сеть с вершинами, которые представляют сколемовские функции.

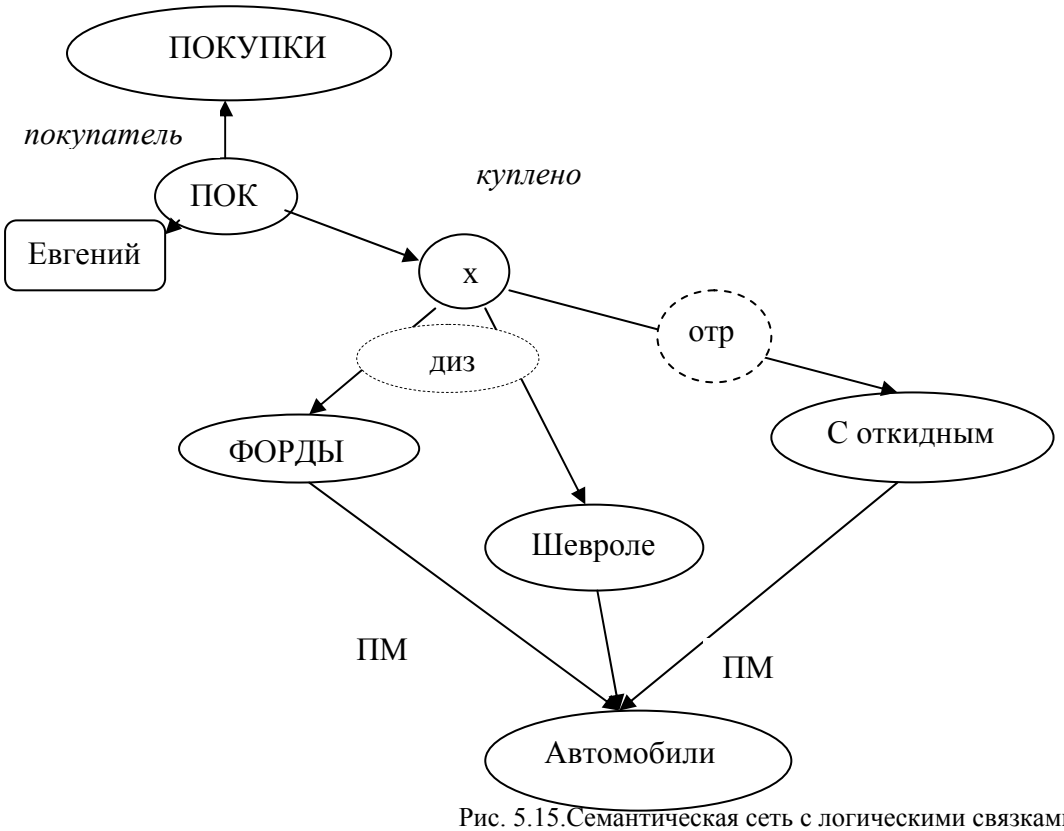


Рис. 5.15.Семантическая сеть с логическими связками

Эта сеть может быть представлена в виде сложно сочиненного предложения:
$\wedge$ РАВ[куплено(ПОК1),Х] $\wedge \sim$ ЭЛ(Х, С_ОТКИДНЫМ_ВЕРХОМ) $\wedge$
$\wedge$ [ЭЛ(Х, Форды) $\vee$ (Х, Шевроле)] $\wedge$
$\wedge$ ПМ(Форды, Автомобили) $\wedge$ ПМ(Шевроле, Автомобили) $\wedge$
$\wedge$ ПМ(С_ОТКИДНЫМ_ВЕРХОМ, Автомобили).

5.8.3. Унификация vs соответствие

Унификация достаточно сильное требование, соответствие несколько более слабое свойство и, главное, несимметричное.

Таблица 5.20

<table>
<tr><td colspan="2">БР1
сам: (элемент-множества Браки)
мужчина : Евгений
женщина : Маша
или
ЭЛ(БР1, Браки($\wedge$ РАВ[мужчина(БР1), Евгений] $\wedge$ РАВ[женщина(БР1), Маша]</td></tr>
<tr><td>БР1
сам: (элемент-множества Браки)
мужчина : Евгений</td><td>БР1
сам: (элемент-множества Браки)
мужчина : Евгений
женщина : Маша
продолжительность: 10</td></tr>
</table>

Верхний блок находится в соответствии с нижним правым блоком, но не с нижним левым.

Рассмотрим эти ситуации в представлении семантических сетей

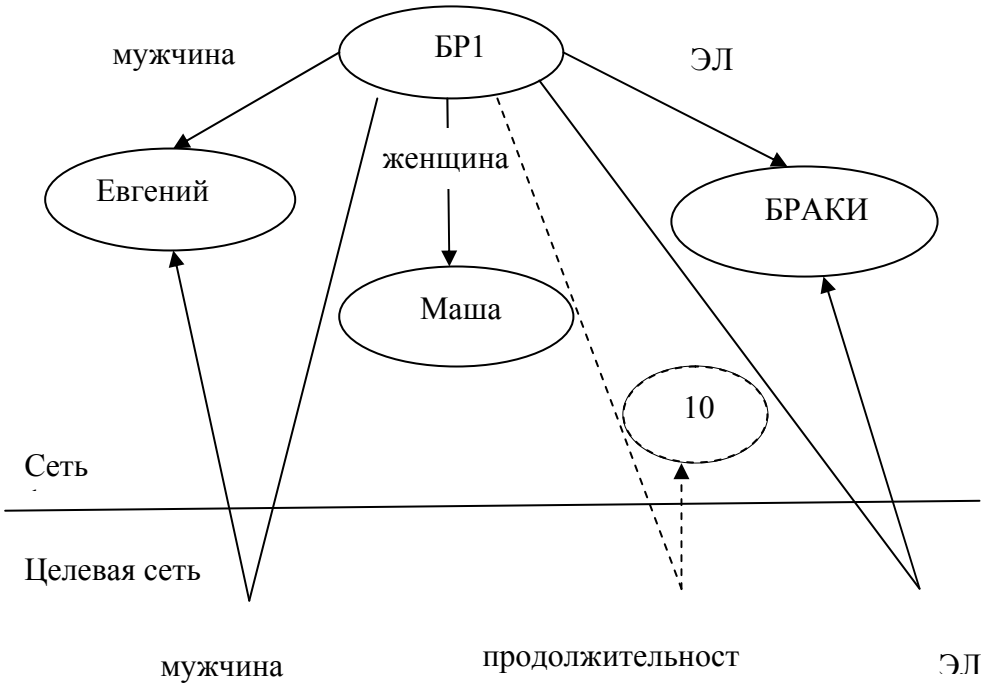


Рис. 5.16. Пример соответствия целевой сети сети фактов, при исключении пунктирных элементов, и несоответствия при включении пунктирных элементов.

Если исключить пунктирную вставку, то две сети: целевая находится в соответствии с сетью фактов, но при добавлении пунктирных деталей, это соответствие нарушается.

При альтернативных представлениях высказываний семантические конструкции

могут вообще не соответствовать друг другу. Высказывания «Евгений женат на Маше» «У Евгения супруга Маша» можно представить следующим образом

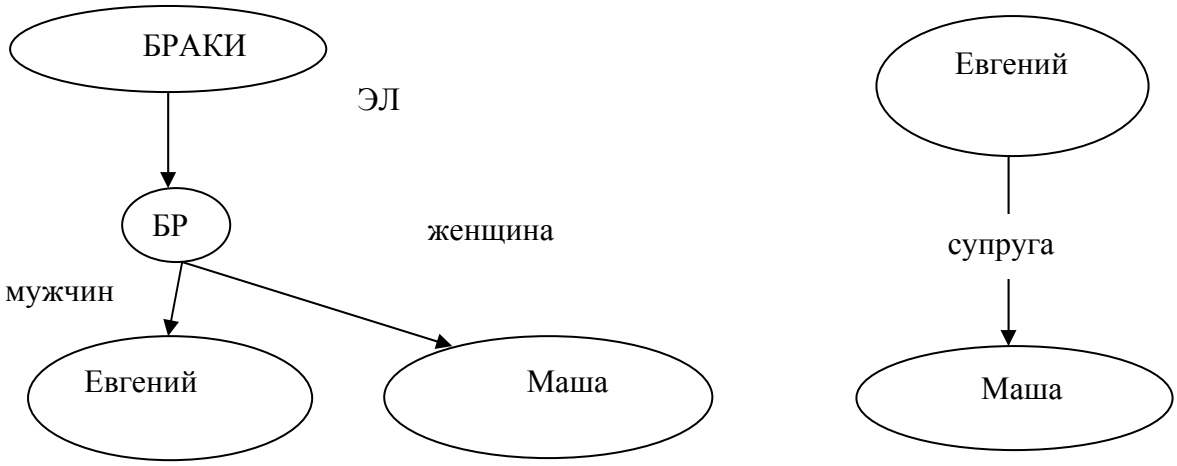


Рис.5.17. Не соответствующие альтернативные представления.

Пример запроса.

Вопрос: «Боря дал Маше ручку?» Это целевой блок:

x
сам: (элемент-множества ПЕРЕДАЧИ)
дающий: Боря
получающий: Маша
объект: ручка

Блоки фактов следующие:

ПЕР1
сам: (элемент-множества ПЕРЕДАЧИ)
дающий: Евгений
получающий: Маша
объект: книга

ПЕР2
сам: (элемент-множества ПЕРЕДАЧИ)
дающий: Боря
получающий: получающий (ПЕР1)
объект: ручка

Целевой блок соответствует блоку ПЕР1, но при правильной унификации.

5.8.4.Дедуктивные операции над структурированными объектами

Представление импликации. Рассмотрим правило «Все студенты ФКН учатся на выпускном курсе»

(а) ЭЛ(x, КН_Студенты) $\Rightarrow$РАВ[курс(x), Выпускной]

Блоки импликации будем представлять блоком описаний, где x- видовая переменная , относящаяся к квантору общности. Имя множества, относящегося к квантору общности будем записывать после переменой x , отделяя её вертикальной чертой:

(б)
x| КН_Студенты
специализация: КН
курс: Выпускной

Представление в сети

(в)

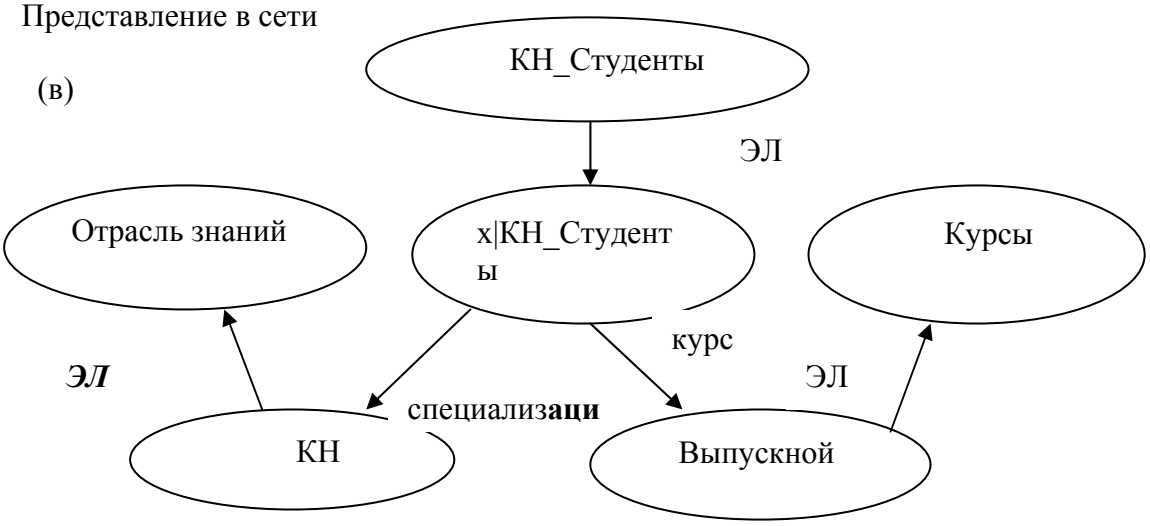


Рис.5.18. Использование блока описания в сети.

Теперь

Факт: ЭЛ(Евгений, КН_Студенты)

Правило: ЭЛ(x, КН_Студенты) $\Rightarrow$РАВ[курс(x), Выпускной]

Цель: РАВ[курс(Евгений), Выпускной]

Блок фактов

Евгений

сам:(элемент-множества КН_Студенты)

Цель:

Евгений

Курс: Выпускной

1.Блок описаний x| КН_Студенты соответствует блоку факта Евгений, если унифицировать переменную x= Евгений, тогда

2. Добавляя к блоку описаний функции *специализация*: КН и *курс*: Выпускной, убеждаемся: блок для факта соответствует целевому блоку.

Абстракция элемента множества. При необходимости, если введенные элементы множества не способны нести смысл его абстрактного элемента, то такой объект может быть введен в семантическую сеть.

Пример: в семантической сети во множестве «автомобили» выделены конкретный автомобиль, и блок описаний, представляющий каждый (любой) автомобиль, однако нам понадобится автомобиль, который был изобретен в 1892 году. Тогда придется ввести еще объект = «абстрактный автомобиль», для построения новой семантической связи об изобретении 1892 года. Ибо все предыдущие объекты не способны такую связь организовать.

Пример: Ранее мы использовали понятия-объекты «Юрист», «Программист»- это тоже примеры абстрактных объектов.

Наследование свойств. При дедукции приходится переходить к объектам, которые представляют собой множества, включающие в себя данный объект. При этом объект наследует свойства этих обобщающих множеств, что позволяет продолжать процедуры.

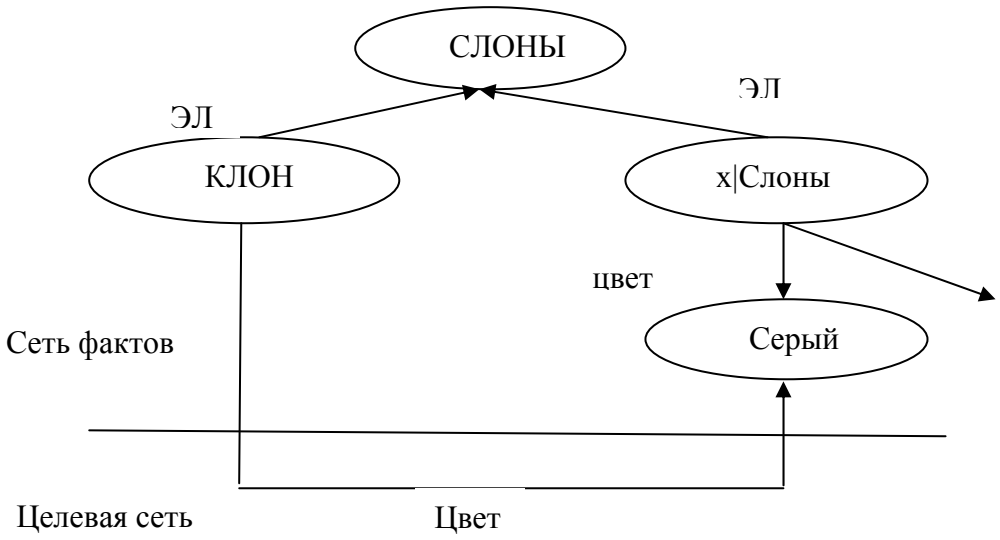


Рис.5.19. Пример доказательства за счет наследования свойств объектом КЛОН.

Программа автоматически наследует все необходимые свойства множеств и супермножеств, в которые объект входит.

Из вершины КЛОН в вершину Серый нет прямой дуги «цвет», то следует пониматься в обобщения=множества типа СЛОНЫ и через импликацию в форме блока описаний приходим к дуге «цвет» и далее к целевой вершине.

Переменные целевые вершины - это обычно константные вершины, причем константы сколемовские, то есть, такие, которые следует определить. ***Константные вершины*** - это реальные определенные объекты. ***Стратегия*** решения задач следующая:

1.Поиск константных вершин в сети фактов

(а) ищутся константные вершины в сети фактов, которые находятся в цепи, соединяющей переменную целевую вершину или

(б) поднимаясь во множество, откуда выделен переменный целевой объект (переменная целевая вершина связана с множеством, которое позволяет операцию наследования) и затем опускаясь по другим направлениям, ищутся константные вершины.

2.Поиск соответствия сети целевой и сети фактов. В первом случае (а), нужно установить соответствие между переменной целевой вершиной и вершинами сети фактов (можно позволить программе поиска соответствия использовать описания при поиске соответствия). **Пример (а)**.

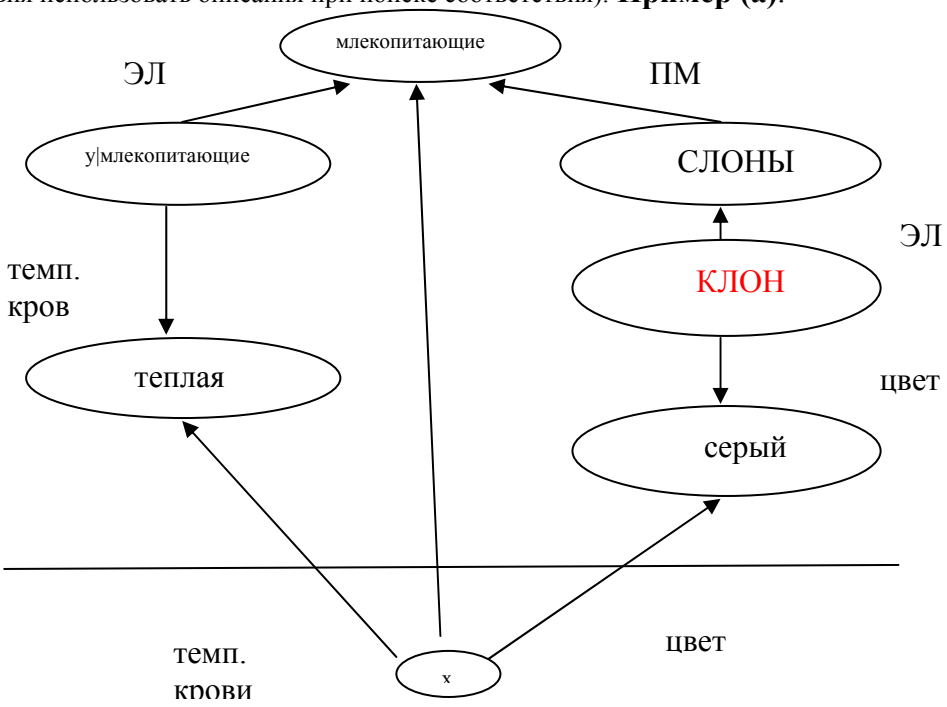


Рис.5.20. Поиск соответствия сети целевой с переменной целевой вершиной и сети фактов с явной константной вершиной.

Требуется установить объект из множества «млекопитающие», такой, что он серый и что кровь у него теплая. Рассмотрим сеть фактов (выше горизонтальной линии) и целевую сеть (ниже горизонтальной линии). Константная вершина только одна - КЛОН. Нужно найти

- дугу «температура крови» в вершину «теплая» из вершины КЛОН и
- дугу «цвет» в вершину «серый» из вершины КЛОН.
- дугу ЭЛ из КЛОН в вершину «млекопитающие».

Если мы это сделаем, то целевая сеть соответствует сети фактов при подстановке x=КЛОН.

В случае (б) константная вершина «спрятана» в сети фактов. Это вариант «неявной» константной вершины. Поэтому для ее поиска нужно подняться в вершину «млекопитающие» (обычно программа при неявных константных вершинах стартует по дугам ЭЛ или ПМ, которые обеспечивают наследование, чтобы потом опускаться по вееру цепей) и пройтись по всем цепям, которые выходят из этой вершины. Покажите, как достигается соответствие сети целевой сети фактов этом случае.

Пример (б).

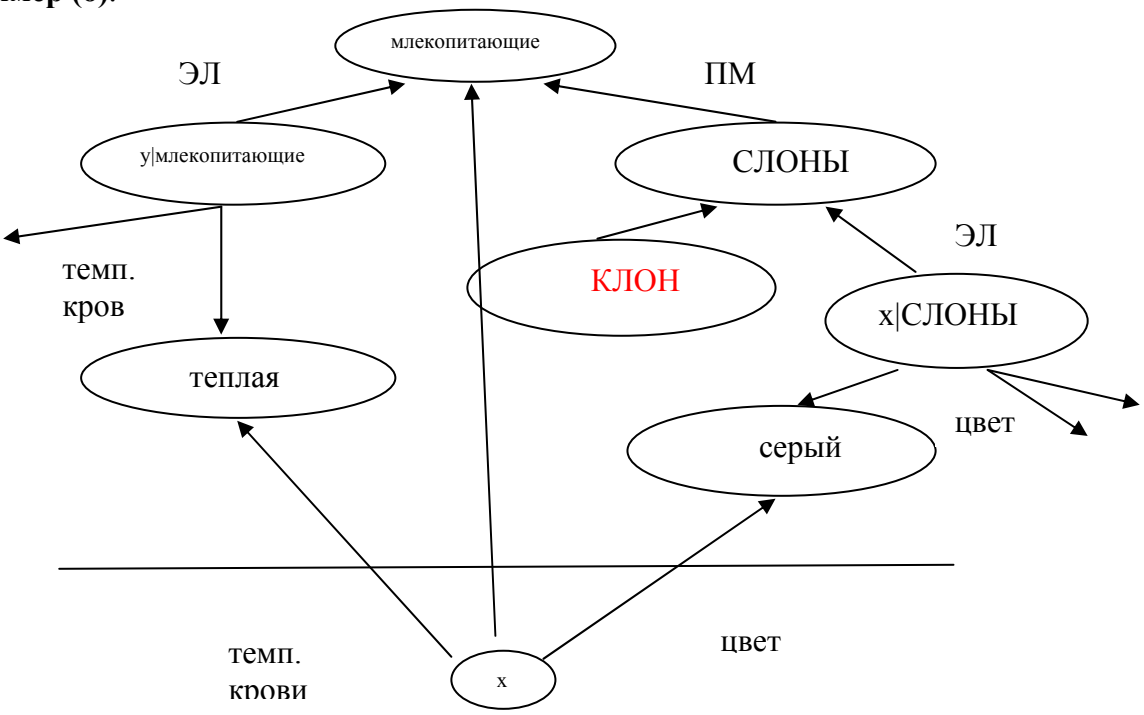


Рис.5.21. Поиск соответствия сети целевой с переменной целевой вершиной и сети фактов с неявной константной вершиной.

Присоединенные процедуры. Это программы, которые вызываются при необходимости .

Представим целевой блок

z

сомножитель1: 3
сомножитель2: 6
произведение: x

для получения значения произведения введем блок описания умножения

x| УМН

сомножитель1: (элемент множества «натуральные числа»)
сомножитель2: (элемент множества «натуральные числа»)
произведение: ПЕРЕМНОЖИТЬ[сомножитель1(x), сомножитель2(x)]

Здесь ПЕРЕМНОЖИТЬ[сомножитель1(x), сомножитель2(x)] – это присоединенная процедура, (программа умножения). При подстановке (унификации):
{ПЕРЕМНОЖИТЬ(3, 6)/x}, получим ответ.

Блочные правила. Блочное правило содержит два списка блоков - Антецедент (*АНТЕ*) и Консеквент (*КОНСЕ*). Применяются эти правила при импликации.

Если все блоки в *АНТЕ* (целевые) сопоставлены с блоками-фактами, то после применения блочного правила, блоки КОНСЕ можно также присоединять к блокам-фактам.

*Если блоки *АНТЕ* считать целевыми, то переменные относятся к квантору существования.

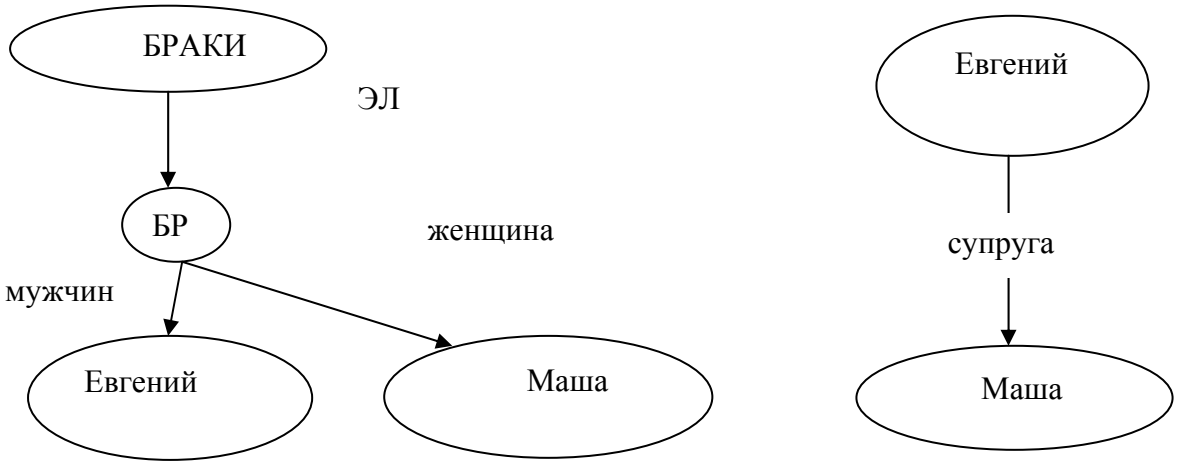
**Если добавляемые блоки уже имеются среди блоков-фактов, то при присоединении добавляют только свойства, которые фигурируют в *КОНСЕ*.

Пример:

П1: Глава(x,y) $\Rightarrow$ Работает_в(x,y)
или
{ЭЛ(x,ОТДЕЛЫ) $\wedge$ РАВ[ГЛАВА(x),y]} $\Rightarrow$ РАВ[работает_в(y), x]
или
П1:
АНТЕ: x
сам: (элемент-множества ОТДЕЛЫ)
глава: y
КОНСЕ:y
работает_в x

П2:[РАБОТАЕТ_В(x,y) $\wedge$ Глава(x,z)] $\Rightarrow$ БОСС(y,z)
или
{РАВ[работает_в(y),x] $\wedge$ РАВ(глава(x),z]} $\Rightarrow$ [РАВ(босс(y),z]
или
П2
АНТЕ: y
работает_в x
x
глава:z
КОНСЕ:y
босс:z

Пример:



Повторение: Рис.5.17. Не соответствующие альтернативные представления.

Тождественность двух соотношений:

РАВ[у,супруга(х)] ≡ (∃ z){ЭЛ(z, БРАКИ) ∧ РАВ[х, мужчина(z)] ∧ РАВ[х, женщина(z)] }

*Вообще говоря, вместо W1 ≡ W2 нужно было бы записать [W1 ⇒ W2] ∧ [W2 ⇒ W1].

Сколемовское преобразование:

РАВ[у,супруги(х)] ⇒){ЭЛ(бр(х,у), БРАКИ) ∧

∧ РАВ[х, мужчина(бр(х,у))] ∧

∧ РАВ[х, женщина(бр(х,у))]

Блочное правило:

П_БР

АНТЕ: х

супруга: у

КОНСЕ: бр(х,у)

сам: (элемент-множества БРАКИ)

мужчина:х

женщина: у

Процедура применения - *установить соответствие между блоком АНТЕ и некоторым блоком-фактом, а потом создать константный блок, соответствующий блоку КОНСЕ.*

Сетевые правила

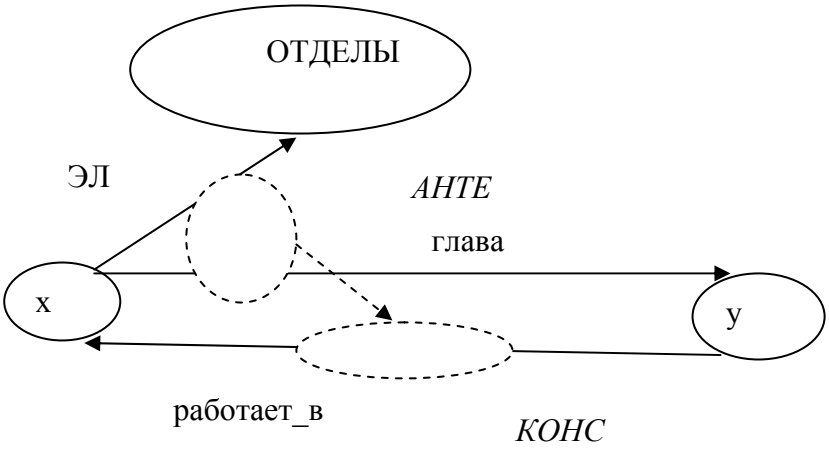


Рис.5.22. Импликация:
{ЭЛ(х,ОТДЕЛЫ) ∧ РАВ[ГЛАВА(х),у]} ⇒ РАВ[работает_в(у), х]

Прямое правило: структура АНТЕ, как целевая. Показываем, что она соответствует сети фактов. После этого к сети фактов присоединяют структуру КОНСЕ, что и есть применение импликации.

Неточные описания. Если КЛОН является слоном альбиносом, то возникает коллизия. В детальном описании может быть приведена информация о том, что он белый. А в сети есть выход в иерархически высшую вершину СЛОНЫ, а затем уж к вершине «серые». Как поступит машина? С одной стороны на детальном уровне ясно, что слон это белый, а через иерархически высшую вершину получаем – серый. Поэтому должна быть создана схема решения, когда ***только при отсутствии детальной (низшей иерархически) информации, нужно подниматься вверх к более высокой и общей иерархической вершине, наследовать её характеристики и двигаться по сети в поисках ответа.*** Можно ввести т.н. неточные описания, то есть вершины - описания,

которые имеют низший уровень приоритетов[14], в отношении очередности их выполнения. А можно следовать приведенному выше правилу.

Добавление рекомендаций. Рассмотрим эту ситуацию на приведенном выше примере двух правил.

П1:
АНТЕ:x
сам: (элемент-множества ОТДЕЛЫ)
глава: y
КОНСЕ:y
работает_в x

П2
АНТЕ: y
работает_в x
x
глава:z
КОНСЕ: y
босс:z

Рекомендация 1:
u|СЛУЖАЩИЕ
босс: (элемент_множества СЛУЖАЩИЕ)
〈 при заполнении П2〉
работает_в (элемент множества ОТДЕЛЫ)
Рекомендация 2:
r|ОТДЕЛЫ
глава(элемент_множества СЛУЖАЩИЕ)
〈 при заполненной П1〉

В Рекомендации1, обозначение 〈при заполнении П2〉 указывает, что П2 должно применяться в обратном направлении всякий раз, когда в блоке есть какое-то значение ячейки «босс», при условии, что прямого соответствия с блоком фактов установить нельзя.

В Рекомендации2 , обозначение 〈при заполненной П1〉, говорит о том, что П1 должно применяться когда у некоторого блока фактов ячейка « сам» содержит элемент множества ОТДЕЛЫ и ячейке «глава» присвоено какое-то значение.

Пример:

ВАСЯ
сам: (элемент_множестваСЛУЖАЩИЕ)
работает_в: ВПК
ВПК
сам: (элемент-множества ОТДЕЛЫ)
глава: Евгений

1 вариант: Пусть предъявлен второй блок, то описание (рекомендация2) r|ОТДЕЛЫ укажет на то, что П1 применяется в прямом направлении (элемент множества Отделы конкретный- это ВПК и происходит уточнение значения переменной x , которая относилась к квантору существования) что породит

Евгений
работает_в: ВПК

[14] Часто используют вместо термина «неточная»- default, что может иметь смысл «за неимением точной» или «по умолчанию».

2 вариант: вопрос «Кто является боссом Васи?» То есть это блок вида

ВАСЯ

босс:u

Прямого соответствия с блоком-фактом не установить. Но согласно Рекомендации 1 , так как упомянули ячейку «босс» рекомендуется применить П2 в обратном направлении. Откуда

ВАСЯ

работает-в: x

x

глава:u

сопоставить с блоком-фактом ВАСЯ можно при x= ВПК, тогда второй блок можно сопоставить в блоком-фактом ВПК при подстановке u= Евгений, что и требовалось выяснить.

Литература к разделу 5

Основная

1.P. Payr, S. Payr . Speaking Minds: Interviews with Twenty Eminent Cognitive Scientists / Princeton, N. J.: Princeton University Press, 1995.

2. J. Hawkins with S. Blakeslee. On Intelligence/ New York: Times Books. 2005.

3. Н. Нильсон Принципы искусственного интеллекта / М.- Радио и связь.- 1985.

4. В.В. Круглов, В.В. Борисов. Искусственные нейронные сети./ – М.: Горячая линия. – Телеком, 2002.

5. Заде Л.А. Основы нового подхода к анализу сложных систем и процессов принятия решений// Математика сегодня. – М.: Знание, 1974. – С.5-49.

Дополнительная (классические работы)

6. Р. Ковальский. Логика в решении проблем./ М. Наука, 1990.

7. Представление и использование знаний. / Под ред. Х.Уэно, М. Исидзука. М.: Мир, 1989.

8.Логический подход к искусственному интеллекту: от классической логики к логическому программированию: Пер. с франц. / Тейз А., Грибомон П., Луи Ж. и др. – М. Мир., 1990.

9.Дж. Малпас. Реляционный язык ПРОЛОГ и его применение./ М. «Наука», 1990.

10. Д. Гроп. Методы идентификации систем. М.: Мир, 1979.

11.Ж.-Л. Лорен. Системы искусственного интеллекта. М.Мир, 1991.

12.Zadeh L.A. Thinking Machines – a New Field in Electrical Engineering// Columbia Eng,. –1950. – № 3.

13.Заде Л.А. Понятие лингвистической переменной и его применение к принятию приближенных решений. – М.: Мир, 1976.