ВВЕДЕНИЕ

Логическое программирование (в широком смысле) представляет собой семейство таких методов решения задач, в которых используются приемы логического вывода для манипулирования знаниями, представленными в декларативной форме.

Пролог — язык и система логического программирования, основанные на языке предикатов математической логики дизъюнктов Хорна, представляющей собой подмножество логики предикатов первого порядка.

Задачи курсовой работы:

‐ изучить классификация экспертных систем;

‐ изучить классификация программных средства для разработки экспертных систем;

‐ разработать ЭС «Подбор жанров кинофильмов»;

‐ сформировать дерево решения задачи, правила, таблицу «атрибут-предикат-значение» и листинг;

‐ создание базы данных;

‐ сформировать ориентированный граф.

Цель курсовой работы разработать программное обеспечения с помощью языка Пролог на тему «Подбор жанров кинофильмов».

1 КЛАССИФИКАЦИЯ ЭКСПЕРТНЫХ СИСТЕМ

Класс "экспертные системы" сегодня объединяет несколько тысяч различных программных комплексов, которые можно классифицировать по различным критериям. Полезными могут оказаться следующие классификации представленной на рисунке 1.

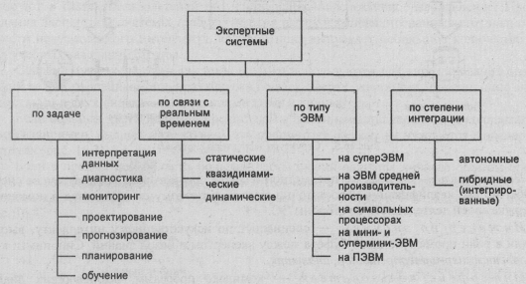


Рисунок 1 – Классификация экспертных систем.

Классификация по решаемой задаче:

‐ интерпретация данных – это определение смысла данных, результаты которого должны быть согласованными и корректными. Обычно предусматривается многовариантный анализ данных;

‐ диагностика – это обнаружение неисправности в некоторой системе, отклонения от нормы. Такая трактовка позволяет с единых теоретических позиций рассматривать и неисправность оборудования в технических системах, и заболевания живых организмов, и всевозможные природные аномалии. Важной спецификой является необходимость понимания функциональной структуры диагностирующей системы;

‐ мониторинг. Основная задача мониторинга – непрерывная интерпретация данных в реальном масштабе времени и сигнализация о выходе тех или иных параметров за допустимые пределы. Главные проблемы - "пропуск" тревожной ситуации и инверсная задача "ложного" срабатывания. Сложность этих проблем в размытости симптомов тревожных ситуаций и необходимость учета временного контекста;

‐ проектирование. Проектирование состоит в подготовке спецификаций на создание "объектов" с заранее определенными свойствами. Под спецификацией понимается весь набор необходимых документов – чертеж, пояснительная записка и т.д. Основные проблемы здесь – получение четкого структурного описания знаний об объекте и проблема "следа". Для организации эффективного проектирования и, в еще большей степени, перепроектирования необходимо формировать не только сами проектные решения, но и мотивы их принятия. Таким образом, в задачах проектирования тесно связываются два основных процесса, выполняемых в рамках соответствующей ЭС: процесс вывода решения и процесс объяснения;

‐ прогнозирование. Прогнозирующие системы логически выводят вероятные следствия из заданных ситуаций. В прогнозирующей системе обычно используется параметрическая динамическая модель, в которой значения параметров "подгоняются" под заданную ситуацию. Выводимые из этой модели следствия составляют основу для прогнозов с вероятностными оценками;

‐ планирование. Под планированием понимается нахождение планов действий, относящихся к объектам, способным выполнять некоторые функции. В таких ЭС используются модели поведения реальных объектов с тем, чтобы логически вывести последствия планируемой деятельности.

‐ обучение. Системы обучения диагностируют ошибки при изучении какой-либо дисциплины с помощью ЭВМ и подсказывают правильные решения. Они аккумулируют знания о гипотетическом "ученике" и его характерных ошибках, затем в работе способны диагностировать слабости в знаниях обучаемых и находить соответствующие средства для их ликвидации. Кроме того, они планируют акт общения с учеником в зависимости от успехов ученика с целью передачи знаний.

Классификация по связи с реальным временем:

‐ статические ЭС разрабатываются в предметных областях, в которых база знаний и интерпретируемые данные не меняются во времени. Они стабильны;

‐ квазидинамические ЭС интерпретируют ситуацию, которая меняется с некоторым фиксированным интервалом времени;

‐ динамические ЭС работают в сопряжении с датчиками объектов в режиме реального времени с непрерывной интерпретацией поступаемых данных.

Классификация по типу ЭВМ:

‐ ЭС для уникальных стратегически важных задач на суперЭВМ (Эльбрус, CRAY, CONVEX и др.);

‐ ЭС на ЭВМ средней производительности (типа ЕС ЭВМ, mainframe);

‐ ЭС на символьных процессорах и рабочих станциях (SUN, APOLLO);

‐ ЭС на мини- и супермини-ЭВМ (VAX, micro-VAX и др.);

‐ ЭС на персональных компьютерах (IBM PC, MAC II и подобные).

Классификация по степени интеграции с другими программами:

‐ автономные ЭС работают непосредственно в режиме консультаций с пользователем для специфически "экспертных" задач, для решения которых не требуется привлекать традиционные методы обработки данных (расчеты, моделирование и т.д.);

‐ гибридные ЭС представляют программный комплекс, агрегирующий стандартные пакеты прикладных программ (например, математическую статистику, линейное программирование или системы управления базами данных) и средства манипулирования знаниями. Это может быть интеллектуальная надстройка над ППП или интегрированная среда для решения сложной задачи с элементами экспертных знаний.

2 КЛАССИФИКАЦИЯ ПРОГРАММНЫХ СРЕДСТВА ДЛЯ

РАЗРАБОТКИ ЭКСПЕРТНЫХ СИСТЕМ

В настоящее время имеется ряд средств, ускоряющих создание экспертных систем. Эти средства называются инструментальными (ИС), или инструментарием. Использование ИС сокращает время разработки ЭС в 3-5 раз. По своему назначению и функциональным возможностям инструментальные программы, применяемые при проектировании экспертных систем, можно разделить на четыре достаточно большие категории:

‐ оболочки экспертных систем;

‐ языки программирования высокого уровня;

‐ среда программирования, поддерживающая несколько парадигм;

‐ дополнительные модули.

Оболочки экспертных систем.

Системы этого типа создаются, как правило, на основе какой-нибудь экспертной системы, достаточно хорошо зарекомендовавшей себя на практике. При создании оболочки из системы-прототипа удаляются компоненты, слишком специфичные для области ее непосредственного применения, и оставляются те, которые не имеют узкой специализации. Примером может служить система EMYCIN, созданная на основе прошедшей длительную «обкатку» системы MYCIN. В EMYCIN сохранен интерпретатор и все базовые структуры данных – таблицы знаний и связанные с ними механизм индексации. Оболочка дополнена специальным языком, улучшающим читабельность программ, и средствами поддержки библиотеки типовых случаев и заключений, выполненных по ним экспертной системой.

Языки программирования высокого уровня

Инструментальные средства этой категории избавляют разработчика от необходимости углубляться в детали реализации системы – способы эффективного распределения памяти, низкоуровневые процедуры доступа и манипулирования данными. Одним из наиболее известных представителей таких языков является OPS5. Этот язык прост в изучении и предоставляет программисту гораздо более широкие возможности, чем типичные специализированные оболочки. Следует отметить, что большинство подобных языков так и не было доведено до уровня коммерческого продукта и представляет собой скорее инструмент для исследователей.

Среда программирования, поддерживающая несколько парадигм

Средства этой категории включают несколько программных модулей, что позволяет пользователю комбинировать в процессе разработки экспертной системы разные стили программирования. Среди первых проектов такого рода была исследовательская программа LOOP, которая допускала использование двух типов представления знаний: базирующегося на системе правил и объектно-ориентированного. На основе этой архитектуры во второй половине 1980-х годов было разработано несколько коммерческих программных продуктов, из которых наибольшую известность получили KEE, KnowledgeCraft и ART. Эти программы предоставляют в распоряжение квалифицированного пользователя множество опций и для последующих разработок, таких как КАРРА и CLIPS, и стали своего рода стандартом. Однако освоить эти языки программистам далеко не так просто, как языки, отнесенные к предыдущей категории.

Дополнительные модули

Средства этой категории представляют собой автономные программные модули, предназначенные для выполнения специфических задач в рамках выбранной архитектуры системы решения проблем.

Экспертные системы выполняются на ЭВМ следующих типов:

‐ общего назначения;

‐ ПЭВМ;

‐ интеллектуальные рабочие станции;

‐ последовательные символьные ЭВМ типа Лисп-машин и Пролог-машин;

‐ параллельные символьные ЭВМ.

Программные инструментальные средства определяются следующими характеристиками:

‐ назначение;

‐ стадия существования;

‐ тип ИС;

‐ тип используемых методов и знаний;

‐ универсальность;

‐ основные свойства;

‐ среда функционирования.

Назначение позволяет определить, для работы в каких проблемных областях и для создания какой стадии ЭС предназначено ИС.

По степени проработанности ИС обычно выделяют три стадии существования: экспериментальная, исследовательская, коммерческая. Экспериментальные ИС создаются для решения узких специфических задач и редко проверяются на других задачах, обычно они работают медленно и неэффективно. Средства, достигшие в исследовательской стадии, обычно тщательно проверены, имеют документацию и поддерживаются разработчиком, однако они тоже могут действовать медленно и неэффективно. Высшей стадией существования ИС является коммерческая. Этой стадии достигают те ИС, которые всесторонне и тщательно проверены, хорошо документированы, сопровождены разработчиком, являются быстрыми и обладают удобным интерфейсом.

По типу ИС классифицируются следующим образом:

‐ символьные языки программирования, ориентированные на создание ЭС и систем ИИ (например, LISP, INTERLISP, SMALLTALK);

‐ языки инженерных знаний, т.е. языки высокого уровня, ориентированные на построение ЭС (например, OPS-5, LOOPS, KES, ПРОЛОГ);

‐ системы, автоматизирующие разработку ЭС (например, KEE, ART, TEIRESIAS, AGE, TIMM), их часто называют окружением для разработки систем ИИ, ориентированных на знания;

‐ оболочки ЭС – ЭС, не содержащие знаний ни о какой проблемной области (например, ЭКСПЕРТИЗА, EMYCIN, ЭКО, ЭКСПЕРТ).

В приведенной классификации ИС перечислены в порядке убывания трудозатрат, необходимых на создание с их помощью конкретной ЭС. Инструментальные средства можно классифицировать и по классам ЭС:

‐ ИС для создания простых ЭС;

‐ ИС для создания сложных ЭС.

В настоящее время, как правило, ИС первого типа разрабатываются на ПЭВМ, а второго – на символьных ЭВМ, интеллектуальных рабочих станциях и ЭВМ общего назначения.

По типу используемых методов и знаний ИС делятся, так же как и ЭС, на традиционные, использующие только методы и способы представления инженерии знаний, и гибридные, сочетающие подходы инженерии знаний с подходами, развитыми в традиционном программировании при представлении данных и использовании подпрограмм.

Универсальность задаётся совокупностью двух параметров:

‐ универсальностью представления знаний;

‐ универсальностью функционирования.

Универсальность представления характеризует способ представления знаний в ИС и принимает следующие значения: единое представление – ИС использует одну модель; интегральное представление – ИС допускает интегральное использование нескольких моделей; универсальное – ИС допускает интегральное использование всех основных моделей представления. К основным моделям представления относятся: правила; фреймы или семантические сети; логические модели (исчисление предикатов). Примерами ИС, в которых используется единое представление, является ПРОЛОГ, интегральное представление – CENTAUR, а универсальное – KEE, ART.

Универсальность функционирования характеризует механизмы, определяющие, как в ИС задается поведение системы, и принимает следующие значения: единый механизм функционирования – ИС использует один механизм функционирования; интегральное функционирование – ИС допускает интегральное использование нескольких механизмов функционирования; универсальное функционирование – ИС допускает интегральное использование всех основных механизмов. К основным механизмам относятся: процедурное программирование; обратное программирование; программирование, ориентированное на данные; программирование, ориентированное на правила.

Процедурное программирование наиболее широко распространено среди существующих сегодня языков программирования (например, Си). Здесь явно выделяют два вида различных сущностей: процедуры, являющиеся тем, что задает поведение программы и выполняющие активную роль; данные, являющиеся тем, что обрабатывается способом, предписанным процедурами, и выполняющие пассивную роль. Способность составлять процедуры из команд и вызывать их – ключ функционирования, использующего данную парадигму. Особенностью её являются боковые эффекты, возникающие в тех случаях, когда различные процедуры, использующие общие данные, независимо их изменяют.

Парадигма объектного программирования в отличие от процедурной не разделяет программу на процедуры и данные. Здесь программа организуется вокруг сущностей, которые включают локальные процедуры и локальные данные. Поведение в этой парадигме организуется путём пересылки сообщений между объектами. Объект, получив сообщение, осуществляет его локальную интерпретацию, основываясь на локальных процедурах и данных. Объектная парадигма удобна в тех приложениях, где взаимодействие сущностей сведено к унифицированным протоколам. Важным свойством языков, использующих данную парадигму, является сеть наследований, которая существенно упрощает определение новых объектов, почти подобных известным.

В обеих рассмотренных парадигмах активная роль в организации поведения отводится процедурам, а не данным. Причем процедура активизируется либо её вызовом, либо посылкой ей сообщения. Подобные способы задания поведения удобны для описания детерминированной последовательности действий либо одного процесса, либо нескольких, но строго взаимосвязанных процессов.

В программировании, ориентированном на данные, активную роль играют данные, а не процедуры. Здесь со структурами активных данных связывают некоторые процедуры, которые активизируются тогда, когда осуществляется доступ к этим данным. Программирование, ориентированное на данные, позволяет организовать поведение независимых процессов, что трудно реализовать в процедурной и объектной парадигмах. Малая зависимость процессов означает, что они могут рассматриваться и программироваться отдельно. Однако при использовании парадигмы, управляемой данными, эти независимо запрограммированные процессы, могут взаимодействовать между собой без их изменения.

В программировании, ориентированном на правила, поведение определяется множеством правил вида условие – действие. Условие задаёт образ данных, при возникновении которого действие правила может быть выполнено. Правила в данной парадигме играют такую же роль, как и подпрограммы в процедурной парадигме. Однако если в процедурной парадигме поведение задаётся детерминированной последовательностью процедур, не зависящей от значений этих данных, то в парадигме, ориентированной на правила, поведение задаётся не заранее предписанной последовательностью правил, а формируется на основе тех данных и их значений, которые в текущий момент обрабатываются программой. Формирование поведения осуществляется по следующей схеме. Условия правил сопоставляются с текущими данными, и те правила, условия которых удовлетворяются значениями текущих данных, становятся претендентами на выполнение. Затем по определенному критерию осуществляется выбор одного правила среди претендентов и выполнение его. Если система допускает параллельную обработку, то все правила-претенденты могут выполняться одновременно. Таким образом, правила выполняются в порядке, предписываемом теми образцами данных, которые, как правило, независимы друг от друга. Поэтому подход, ориентированный на правила, удобен для описания поведения, гибко и разнообразно реагирующего на большое многообразие состояний данных.

Основные свойства определяют особенности ИС с точки зрения реализации компонентов ЭС. Среда функционирования определяет тип ЭВМ, на которой реализовано ИС, тип операционной системы, в среде которой ИС работает, и используемый язык программирования.

Инструментальные средства разработки экспертных систем представлена на рисунке 2.



Рисунок 2 – Инструментальные средства разработки экспертных систем представлена.

3 РАЗРАБОТАТЬ ЭС «ПОДБОР ЖАНРОВ КИНОФИЛЬМОВ»

3.1 ФОРМАЛИЗАЦИЯ ЗАДАЧИ – ДЕРЕВО РЕШЕНИЯ ЗАДАЧИ

В кинотеатр звонит клиент и спрашивает у консультанта какой кинофильм он может предложить из-за множества разных кинофильмов консультант должен узнать интересующий жанр кинофильмов что бы определить, что можно предложить клиенту, с помощью нескольких вопрос консультант получить требуемые ему сведенья, необходимые для решения вопроса.

Один из вариантов такой консультации может быть представлен в графической форме, представленной на рисунке 3.

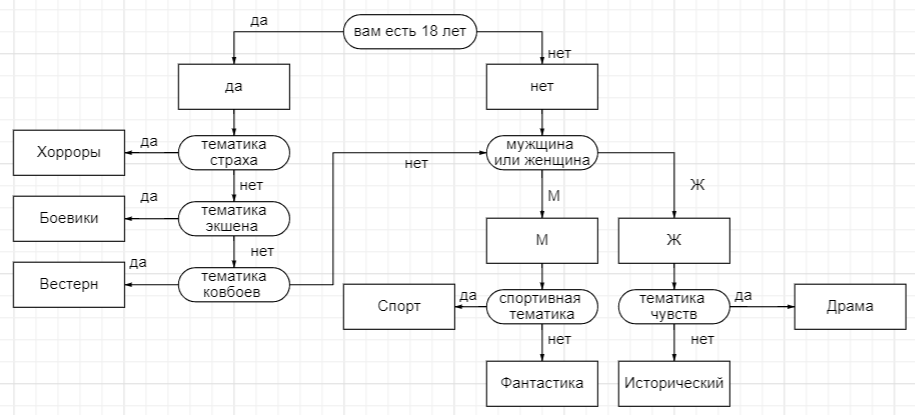


Рисунок 3 – Дерево решения задачи.

3.2 ФОРМИРОВАНИЕ ПРАВИЛ – СХЕМА

Правила в Прологе - это правила логического вывода. Слева в правилах записывается следствие, а справа - предусловие.

В правилах со сложносоставными определениями запятая также играет роль логической связки и, объединяя условия, образующие определение. Такого рода правила позволяют создавать самые слож­ные и изощренные базы знаний по самым различным предметным областям и применениям.

На рисунке 4 представлено формирование правил который является переводом «дерева решений» из вопросов-ответов в утверждения-факты.

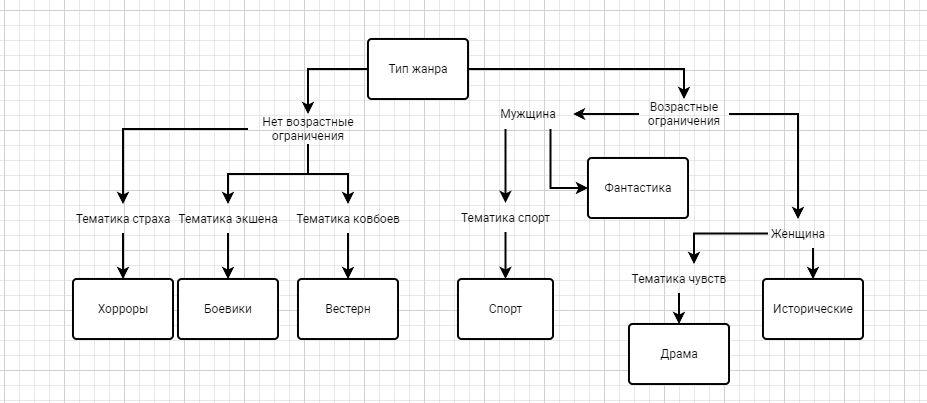


Рисунок 4 – Формирование правил.

3.3 ТАБЛИЦА «АТРИБУТ-ПРЕДИКАТ-ЗНАЧЕНИЕ»

Общим для них является то, что все утверждения представляются предложениями, которые состоят из <атрибута>, <предиката>, <значения>.

Атрибут - это ключевое слово или фраза, описывающая некоторое качество, о котором мы пытаемся найти информацию.

Значение - это описание, назначенное атрибуту.

Предикат - это элемент, указывающий на отношения между атрибутом и его значением.

На таблице 1 представлено Множество фактов для «дерева решений»

Таблица 1 Множество фактов для «дерева решений».

|  |  |  |
| --- | --- | --- |
| Атрибут | Предикат | Значение |
| Тип жанра | IS | Хорроры  Боевики  Вестерн  Фантастика  Спорт  Драма  Исторические |
| Возрастные ограничения | IS | Присутствуют  Отсутствуют |
| Половой признак | IS | Мужчина  Женщина |
| Тематика | IS | Страха  Экшен  Ковбоев  Спорта  Чувств |

4 СОЗДАНИЕ БАЗЫ ДАННЫХ

В базе данных на тему ЭС «подбор жанров кинофильмов» будет состоять из следующих фактов:

- agelimit – ограничения на возраст просмотра жанров кинофильмов имеющее 2 предиката, первый это жанр, второй наличия ограничения или его отсутствие;

- gender – предпочтения жанров по половому признаку, имеющей 2 предиката, первый это жанр, второй половой признак (М, Ж);

- theme – сопоставление интересов к жанру кинофильмов, имеющей 2 предиката, первый это жанр, второй интерес.

Ниже представлен код создания базы данных на тему ЭС «подбор жанров кинофильмов» написанный на языке prolog.

agelimit(sport, not).

agelimit(fantastic, not).

agelimit(drama, not).

agelimit(historical, not).

agelimit(horror, yes).

agelimit(fighters, yes).

agelimit(western, yes).

gender(sport, man).

gender(fantastic, man).

gender(historical, woman).

gender(drama, woman).

theme(sport, sport).

theme(drama, sensation).

theme(horror, horror).

theme(fighters, fighters).

theme(western, cowboys).

Запросы prolog и результаты представлены ниже.

?- agelimit(X, not) , gender(X, man)

X = [sport, fantastic]

?- agelimit(X, yes) ; gender(X, man)

X = [sport, fantastic, horror, fighters, western]

?- agelimit(X, yes) ; gender(X, woman)

X = [historical, drama, horror, fighters, western]

?- agelimit(X, yes) , theme(X, western).

X = [western]

?- agelimit(X, not) , theme(X, western).

X = []

5 ОРИЕНТИРОВАННЫЙ ГРАФ

Графы — это абстракция для представления отношений между сущностями. С ее помощью можно описать дорожную карту, сеть из нескольких модулей, программу и пр. Сам граф описывается как точки и соединяющие их отрезки либо с помощью математических формул.

Ориентированный граф (кратко орграф) — (мульти) граф, рёбрам которого присвоено направление. Направленные рёбра именуются также дугами, а в некоторых источниках и просто рёбрами. Граф, ни одному ребру которого не присвоено направление, называется неориентированным графом или неорграфом.

На рисунке 5 представлен ориентированный граф.

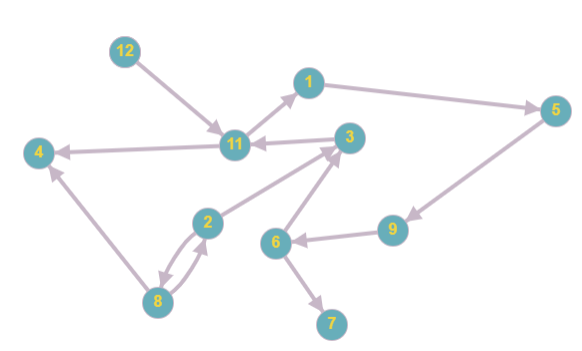


Рисунок 5 - Ориентированный граф.

Ниже представлен код prolog описывающий ориентированный граф.

Path(1,5).

Path(2,8).

Path(2,3).

Path(3,11).

Path(5,9).

Path(6,3).

Path(6,7).

Path(8,4).

Path(8,2).

Path(9,6).

Path(12,11).

Path(11,4).

Path(11,1).Запросы кода prolog

-? Path(9,X).

X = [6]

-? Path(2,X).

X = [3,8]

-? Path(11,X).

X = [1,4]

-? Path(11,X); Path(6, X).

X = [1,3,4,7]

-? Path(12,X)

X = []