МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах

Кафедра «Информационные технологии и компьютерные системы»

РАСЧЕТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ

Методические указания
к выполнению расчетно-графического задания
по дисциплине «Логическое программирование»
для студентов направления подготовки
09.03.01 «Информатика и вычислительная техника»

Севастополь
СевГУ
2019

УДК 004.056.5

Р е ц е н з е н т:
С.Г. Лелеков – доцент кафедры «Информационные технологии и компьютерные системы», канд. физ.-мат. наук, доцент

Составители: Брюховецкий А.А., Ткаченко К.С.

Методические указания для выполнения расчетно-графического задания по дисциплине «Логическое программирование» для бакалавров по направлению 09.03.01 «Информатика и вычислительная техника» / Сост. А.А. Брюховецкий, К.С. Ткаченко. – Электрон. дан. – Севастополь: СевГУ, 2019 г. – Режим доступа: свободный после авторизации. – Загл. с экрана. – 16 с.

Целью методических указаний является оказание методической помощи бакалаврам при выполнении РГЗ по дисциплине «Логическое программирование» при разработке программ на LISP и PROLOG.

Методические указания содержат:
- краткие теоретические сведения;
- постановку задачи для выполнения работы;
- пример выполнения РГЗ;
- содержание отчета;
- список литературы;
- индивидуальные задания для выполнения работы.

УДК 004.056.5

Методические указания рассмотрены и утверждены на заседании кафедры «Информационные технологии и компьютерные системы» (протокол № 4 от 23 января 2019 г.)

Текстовое (символьное) издание (1,5 Мб).

Системные требования: Intel, 3,4 GHz; 150 Мб; Windows XP/Vista/7; DVD-ROM; 1 Гб свободного места на жестком диске; программа для чтения pdf-файлов: Adobe Acrobat Reader, Foxit Reader.

Дата подписания к использованию: 23.01.2019.

РИИЦМ СевГУ. 299015, г. Севастополь, ул. Курчатова, 7.

СОДЕРЖАНИЕ

Введение

В современном мире рост производительности программиста практически достигается только в тех случаях, когда часть интеллектуальной нагрузки берут на себя компьютеры. Одним из способов достигнуть максимального прогресса в этой области является искусственный интеллект, когда компьютер не только берет на себя однотипные, многократно повторяющиеся операции, но и сам может обучаться. Кроме того, создание полноценного искусственного интеллекта открывает перед человечеством новые горизонты развития.

РГЗ рассматривает базовые понятия экспертных систем, методику построения ЭС, а также все этапы построения ЭС: идентификацию, концептуализацию, формализацию, выполнение задачи, тестирование. На практике построение ЭС ведется с использованием средств языков обработки знаний: Lisp и Prolog. Дается разъяснение конкретных путей, которые существуют в настоящее время для повышения интеллектуальности систем проектирования в любой из отраслей производства промышленной продукции и пояснении роли ЭС в современном проектировании.

Задание №1. Работа с динамическими базами данных средствами Пролог

Цель работы: изучить основные приемы реализации динамических баз данных в языке Пролог, разработать структуру БД в соответствии с индивидуальным заданием и программу, реализующую основные типовые операции с БД.

1.1. Теоретические сведения

Реляционная модель данных предполагает, что база данных – это есть описание некоторого множества отношений. Пролог–программу можно рассматривать как такую базу данных, здесь отношения между данными представлены в виде фактов и правил. В Прологе есть специальные средства для модифицирования этой базы данных, то есть добавлять и удалять новые отношения из файла. Для этих целей служат встроенные предикаты.

Чтобы понять, как в Прологе реализуется обращение к БД, рассмотрим запрос на примере БД игроков футбольной команды:

```
dplayer("Bernie Kosar",Team,Pos).
```

В этом утверждении Team и Pos есть переменные, значения которых нужно найти. Когда этот запрос (цель) испытывается, процедуры Пролога просматривают утверждения БД игроков (см.ниже) на предмет сопоставления с утверждением, содержащим Bernie Kosar. Так как в базе данных такое утверждение присутствует, то переменной Team присваивается значение Cleveland Browns, а переменной Pos - QB.

Если трактовать dplayer как предикат БД Пролог, то отсюда следует с необходимостью такое его описание

```
database
    dplayer(name,team,position)
```

Раздел **database** в Прологе предназначен для описания предикатов базы данных, таких как dplayer. Все различные утверждения этого предиката составляют динамическую базу данных Пролога. База данных называется динамической, так во время работы программы из нее можно удалять любые содержащиеся в ней утверждения, а также добавлять новые. В этом состоит ее отличие от "статических" баз данных, где утверждения являются частью кода программы и не могут быть изменены во время счета. Другая важная особенность динамической базы данных состоит в том, что такая база может быть записана на диск, а также считана с диска в оперативную память.

Иногда бывает предпочтительно иметь часть информации базы данных в виде утверждений статической БД; эти данные заносятся в динамическую БД сразу после активизации программы. Для этой цели используются предикаты **asserta и assertz,** которые будут рассмотрены ниже. В общем, предикаты статической БД имеют другое имя, но ту же самую форму представления данных, что и предикаты динамической. Предикат статической БД, соответствующий предикату dplayer динамической базы данных, есть

```
predicates
    player(name,team,position)
clauses
    player("Dan Marino","Miami Dolphins","QB").
    player("Richart Dent","Chicago Bears","DE").
    player("Bernie Kosar","Cleveland Browns","QB").
    player("Doug Cosbie","Dallas Cowboy","TE").
    player("Mark Malone","Pittsburgh Steelers","QB").
```

Заметим, что все отличие предиката dplayer по сравнению с player заключается лишь в одной лишней букве терма. Добавление латинской буквы d - обычный способ различать предикаты динамической и статической баз данных. Правилом для занесения в динамическую БД информации из утверждений предиката player служит

```
assert_database :-
  player(Name,Team,Number),
  assertz( dplayer(Name,Team,Number) ),
  fail.
assert_database :- !.
```

В этом правиле применяется уже знакомый вам метод отката после неудачи, который позволяет перебрать все утверждения предиката player.

Таким образом, можно указать следующее соответствие понятий:

база данных Пролога – реляционная база данных,
предикат БД – отношение,
объект – атрибут,
отдельное утверждение – элемент отношения,
количество утверждений – мощность.

Пример использования динамической базы данных в Прологе.

При создании динамической базы данных Пролога будут очень полезны многие уже известные вам вещи. Так, здесь применим почти весь материал, касающийся предикатов и утверждений Пролога. Можно создавать правила для работы с информацией БД, можно также использовать введенные предикаты, осуществляющие обращение к файлам, файлам БД, записанным на диск.

В Прологе имеются и специальные встроенные предикаты для работы с динамической базой данных. Таковыми являются **asserta, assertz, retract, save, consult, и findall.**

Предикаты **asserta, assertz и retract** позволяют занести факт в заданное место динамической БД и удалить из нее уже имеющийся факт.

Предикат **asserta** заносит новый факт в базу данных, располагающуюся в оперативной памяти компьютера (резидентная БД). Новый факт помещается перед всеми уже внесенными утверждениями данного предиката. Этот предикат имеет такой синтаксис:

```
asserta(Clause).
```

Таким образом, чтобы поместить в БД утверждение

```
dplayer("Bernie Kosar","Cleveland Browns","QB").
```

перед уже имеющимся там утверждением

```
dplayer("Doug Cosbie","Dallas Cowboy","TE"),
```

стоящим в настоящий момент в базе данных на первом месте, необходимо следующее предикатное выражение:

```
asserta(dplayer("Bernie Kosar","Cleveland Browns","QB")).
```

Теперь БД содержит два утверждения, причем утверждение со сведениями о Kosar предшествует утверждению со сведениями о Cosbie:

dplayer("Bernie Kosar","Cleveland Browns","QB").
dplayer("Doug Cosbie","Dallas Cowboy","TE").

Крайне важно отдавать себе отчет в том, что в динамической базе данных могут содержаться только факты (не правила).

Предикат **assertz** так же, как и **asserta**, заносит новые утверждения в базу данных. Однако он помещает новое утверждение за всеми уже имеющимися в базе утверждениями того же предиката. Синтаксис предиката столь же прост:

assertz(Clause).

Для добаления к двум уже имеющимся в БД утверждениям третьего

dplayer("Mark Malone","Pittsburgh Steelers","QB")

потребуется следующее предикатное выражение:

assertz(dplayer("Mark Malone","Pittsburgh Steelers","QB")).

после чего БД будет содержать третьего утверждения

dplayer("Bernie Kosar","Cleveland Browns","QB").
dplayer("Doug Cosbie","Dallas Cowboy","TE").
dplayer("Mark Malone","Pittsburgh Steelers","QB").

Третье, новое, только что занесенное утверждение, следует за двумя старыми.

Предикат **retract** удаляет утверждение из динамической БД (еще раз напомним, что динамическая БД содержит факты, но не правила.) Его синтаксис таков:

retract(Existing_clause).

Предположим, что вы хотите удалить из базы данных второе утверждение. Для этого необходимо написать выражение

retract(dplayer("Doug Cosbie","Dallas Cowboy","TE")).

и БД будет состоять уже только из двух утверждений:

dplayer("Bernie Kosar","Cleveland Browns","QB").
dplayer("Mark Malone","Pittsburgh Steelers","QB").

Так же, как **asserta и assertz, retract** применим только в отношении фактов.

Предикаты **save и consult** применяются для записи динамической БД в файл на диск и для загрузки содержимого файла в динамическую БД.

Предикат **save** сохраняет находящуюся в оперативной памяти базу данных в текстовом файле. Синтаксис этого предиката

save(DOS_file_name),

где DOS_file_name есть произвольное допустимое в MS DOS или PC DOS имя файла.

Для того, чтобы сохранить содержимое футбольной БД в файле с именем football.dba, требуется предикат

save("football.dba").

В результате все утверждения находящейся в оперативной памяти динамической БД будут записаны в файл football.dba.

Файл БД может быть считан в память (загружен) при помощи предиката **consult**, систаксис которого таков:

consult(DOS_file_name).

Для загрузки файла футбольной БД требуется выражение

consult("football.dba").

Предикат **consult** неуспешен, если файл с указанным именем отсутствует на диске, или если этот файл содержит ошибки, как, например, в случае несоответствия синтаксиса предиката из файла описаниям из раздела программы **database**, или если содержимое файла невозможно разместить в памяти ввиду отсутствия места.

Предикат **findall** позволяет собрать все имеющиеся в базе данные в список, который может быть полезен при дальнейшей работе. Так, findall можно использовать для получения списка имен всех игроков.

После того, как успешным будет предикат

findall(Name,dplayer(Name,_,_),Name_list)

переменная Name_list содержит список имен всех игроков.

Пример программы добавления, вывода и удаления из базы данных фактов и значений.

```
Add :– write('Введите имя мужчины: '),
      read_string(S), assertz(man(S)).
Out :– write('Список имен мужчин:\n'), man(S),write(S),nl.
Del :– write('Введите имя мужчины: '),
      read_string(S), retractall(man(S)).
Main :– consult('db.pro'),add,out,del,out.
```

Файл db.pro первоначально содержит следующие факты:

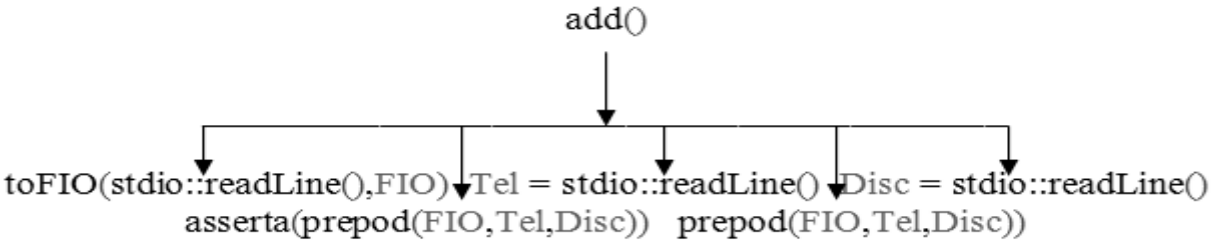
```
man(oleg). wom-
an(zina).
man(alexandr).
man(petr).
```

1.2. Пример выполнения работы

Допустим, в базе данных необходимо хранить следующие данные: ФИО преподавателя, его контактный номер телефона и название преподаваемой дисциплины. Базе данных можно задавать любые вопросы (запросы), механизмы пролога обеспечат обработку любых вопросов. Добавим операции добавления и удаления факта, а так же вывода состояния базы данных в терминал.

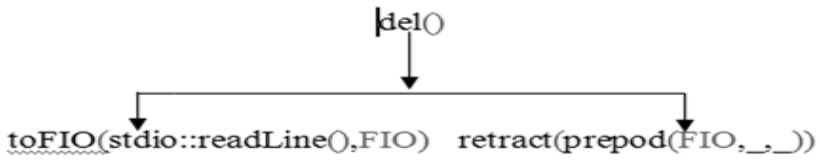
Фрагмент программы добавления факта в базу данных:

```
add():- stdio::write("Добавляем новую запись\n"),
   stdio::write("Введите фамилию преподавателя: "), toFIO(stdio::readLine(),FIO),
   stdio::write("Введите номер телефона: "), Tel = stdio::readLine(),
   stdio::write("Введите преподаваемую дисциплину: "), Disc = stdio::readLine(),
    asserta(prepod(FIO,Tel,Disc)),
    stdio::write("Добавлен преподаватель: ",prepod(FIO,Tel,Disc)), stdio::nl(),
succeed(),!;succeed().
```



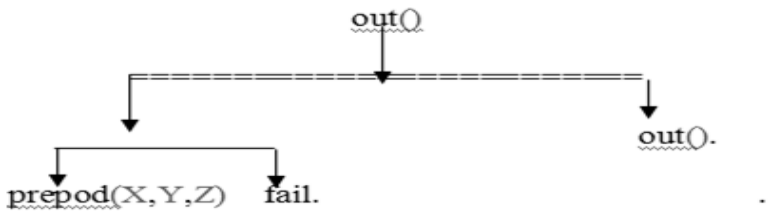
Операция удаления факта из базы данных:

```
del():- stdio::write("Удаляем запись\n"),
          stdio::write("Введите фамилию преподавателя: "),
          toFIO(stdio::readLine(),FIO),
          retract(prepod(FIO,_,_)),
          stdio::write("Удален преподаватель: ",prepod(FIO,Tel,Disc)), stdio::nl(),
   succeed(),!;succeed().
```



Операция вывода базы данных:

```
out():- prepod(X,Y,Z), stdio::write(X,"\n",Y,"\n",Z,"\n\n"),fail.
out().
```



Текст программы

```
    implement main
   open core

constants
   className = "main".
   classVersion = "".
domains
   fio = fio(string,string,string).
 %   prepod = prepod(fio,string,string).
class facts
   prepod : (fio,string,string).
class predicates
  %  prepod : (fio FIO, string Tel, string Discp) determ.
   add : () procedure.
   out : ()procedure.
   del : ()procedure.
   toFIO : (string,fio) determ (i,o).
   fio : (string*, fio)determ (i,o).
clauses
   classInfo(className, classVersion).
      toFIO(In, FIO):- L=string::split(In," "), fio(L,FIO).
   fio([X|[Y|[Z|[]]]], FIO):- FIO=fio(X,Y,Z).

   prepod(fio("Ярков","Игнат","Федорович"),"0-100-500","Компьютерная  логика").
   prepod(fio("Селезнёв","Михаил","Вадимович"),"1-311-404","Компьютерная электроника").
   prepod(fio("Брюлов","Дмитрий","Алексеевич"),"3-333-13","Основы  проектирования
ЭВМ").

   add():- stdio::write("Добавляем новую запись\n"),
           stdio::write("Введите фамилию преподавателя: "), toFIO(stdio::readLine(),FIO),
           stdio::write("Введите номер телефона: "), Tel = stdio::readLine(),
           stdio::write("Введите преподаваемую дисциплину: "), Disc = stdio::readLine(),
            asserta(prepod(FIO,Tel,Disc)),
            stdio::write("Добавлен преподаватель: ",prepod(FIO,Tel,Disc)), stdio::nl(),
```

```
    succeed(),!;succeed().

  out():- prepod(X,Y,Z), stdio::write(X,"\n",Y,"\n",Z,"\n\n"),fail.
  out().

  del():- stdio::write("Удаляем запись\n"),
          stdio::write("Введите фамилию преподавателя: "), toFIO(stdio::readLine(),FIO),
     %     stdio::write("Введите номер телефона: "), Tel = stdio::readLine(),
     %     stdio::write("Введите преподаваемую дисциплину: "), Disc = stdio::readLine(),
          retract  (prepod(FIO,_,_)),
          stdio::write("Удален преподаватель: ",prepod(FIO,Tel,Disc)), stdio::nl(),
  succeed(),!;succeed().

  run():-console::init(),      add(),out(),      del(),out(),      succeed().
end implement main
goal
  mainExe::run(main::run).
```

1.3. Индивидуальное задание и методика выполнения работы

В ходе выполнения работы требуется:

Составить программу, реализующую следующие операции с базой данных:

1. Добавление записей
2. Удаление конкретной записи по заданному значению.
3. Поиск заданной записи.
4. Вывод данных из базы данных.

Выполнить вариант для заданной предметной области: I=(N MOD 10)+1.

1.Отдел кадров	6.Авиакасса
2.Библиотека	7.Магазин
3.Отдел договоров	8.Склад
4.Деканат	9.Автотранспортное предприятие
5.Кафедра	10. Фирма по продаже компьютеров

1.4. Содержание отчета о выполнении индивидуального задания

1. Постановка задачи.
2. Представление структур данных и структуры программ в виде деревьев И/ИЛИ в соответствии с заданным вариантом.
3. Примеры обработки запросов. Представление доказательства цели в виде дерева И/ИЛИ.
4. Анализ результатов работы программ в режиме трассировки.
5. Выводы по работе.

1.5. Контрольные вопросы

1. Назовите способы представления баз данных в ПРОЛОГ.
2. Какие есть типы отношений между термами в ПРОЛОГ.
3. В чем отличие структуры программы для работы с базой данных.
4. Назовите основные предикаты при работе с базами данных.
5. Какие типовые операции над базами данных Вам известны.

1.6. Библиографический список

1. Методические указания по дисциплине «Логическое программирование» для студентов направлению подготовки 09.03.01 "Информатика и вычислительная техника" направленности 09.03.01.01 «ЭВМ, системы и сети»./ А.А.Брюховецкий, Д.Ю.Воронин, П.Нюнькина – Севастополь: Изд-во СевГУ, 2017. – 59с.
2. Братко И. Программирование на языке Пролог для искусственного интеллекта. – М.: Мир, 1990.
3. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. – М.: Мир, 1990.
4. Метакидес Г., Нероуд А. Принципы логики и логического программирования. – М.: Факториал, 1998
5. Хоггер К. Введение в логическое программирование. – М.: Мир, 1988.
6. Малпас Дж. Реляционный язык Пролог и его применение. /Малпас Дж. ; Пер. с англ.; — М.: Наука, 1990. –574с.

Задание №2. Разработка экспертной системы средствами Лисп-подобного языка.

2.1. Теоретические сведения

Имеется, по крайней мере, две точки зрения на то, что следовало бы считать искусственным интеллектом. Первую можно назвать нейробионической. Ее сторонники ставят перед собой цель воспроизвести искусственным образом те процессы, которые протекают в мозгу человека, — это путь изучения естественного мозга, выявление способов его работы, создания технических средств для повторения биологических структур и протекающих в них процессов.

Вторая точка зрения, доминирующая в проблеме искусственного интеллекта, может быть названа информационной. Сторонники информационного исхода считают, что основной целью работ в искусственном интеллекте является не построение технического аналога биологической системы, а создание средств для решения задач, традиционно считающихся интеллектуальными.

Информационная точка зрения в свою очередь неоднородна. В ней можно выделить три направления.

Часть специалистов считает, что можно найти свой способ ее решения на ЭВМ, который даст либо результат, подобный человеческому, либо даже лучший. Специалисты этого направления неоднократно демонстрировали свое искусство по созданию программ такого рода. Достаточно назвать, например, программы для игры в шахматы, которые играют в эту игру лучше подавляющего большинства людей, проводящих время за шахматной доской. Но делают это программы совсем не так, как люди.

Другая часть специалистов считает, что искусственный интеллект должен имитировать не решение отдельных (пусть и весьма творческих) задач. Ибо естественный интеллект человека – это его способность при необходимости обучаться тому или иному виду творческой деятельности, значит, и программы, создаваемые в искусственном интеллекте, должны быть ориентированы не на решение конкретных задач, а на создание для автоматического построения необходимых программ решения конкретных задач, когда в этом возникает необходимость. Именно эта группа исследователей сейчас определяет лицо искусственного интеллекта, составляя основную массу специалистов этого профиля.

Третья часть специалистов – это программисты, чьими руками делают программы для решения задач искусственного интеллекта. Они склонны рассматривать область своей деятельности как новый виток развития программирования. Они считают, что средства, разрабатываемые для написания программ решения интеллектуальных задач, в конце концов, есть

средства, позволяющие по описанию задачи на профессиональном естественном языке построить нужную программу на основании тех стандартных программных модулей, которые хранятся в памяти машины. Все метасредства, которые предлагают те, кто рассматривает искусственный интеллект как способ разобраться на информационном уровне, какие функции реализует естественный интеллект, когда он решает задачу, программисты видят сквозь призму своей цели — создания интеллектуального программного обеспечения (по существу, комплекса средств, автоматизирующих деятельность самого программиста).

Система называется интеллектуальной, если в ней реализованы следующие основные функции:

- накапливать знания об окружающем систему мире, классифицировать и оценивать их с точки зрения прагматической полезности и непротиворечивости, инициировать процессы получения новых знаний, осуществлять соотнесение новых знаний с ранее хранимыми;
- пополнять поступившие знания с помощью логического вывода, отражающего закономерности в окружающем систему мире в накопленных ею ранее знаниях, получать обобщенные знания на основе более частных знаний и логически планировать свою деятельность;
- общаться с человеком на языке, максимально приближенном к естественному человеческому языку;
- получать информацию от каналов, аналогичных тем, которые использует человек при восприятии окружающего мира;
- уметь формировать для себя или по просьбе человека (пользователя) объяснение собственной деятельности;
- оказывать пользователю помощь за счет тех знаний, которые хранятся в памяти, и тех логических средств рассуждений, которые присущи системе.

Наиболее значительные успехи в настоящее время достигнуты в таком классе интеллектуальных систем, как экспертные системы (ЭС).

Принципы проектирования экспертных систем. Первые ЭС были статического типа. Типичная статическая ЭС должна включать следующие компоненты: базу знаний (БЗ); базу данных (БД, рабочую память); решатель (интерпретатор); систему объяснений; компоненты приобретения знаний; интерфейс с пользователем. БЗ ЭС предназначена для хранения долгосрочных данных, описывающих рассматриваемую область, и правил, описывающих целесообразные преобразования данных этой области. БД ЭС служит для хранения текущих данных решаемой задачи. Решатель формирует последовательность применения правил и осуществляет их обработку, используя данные из рабочей памяти и знания из БЗ. Система объяснений показывает, каким образом система получила решение задачи, и какие знания при этом использовались. Это облегчает тестирование системы и повышает доверие пользователя к полученному результату. Компоненты приобретения знаний необходимы для заполнения ЭС знаниями в диалоге с пользователем-экспертом, а также для добавления и модификации заложенных в систему знаний.

К разработке ЭС привлекаются специалисты из разных предметных областей, а именно: эксперты той проблемной области, к которой относятся задачи, решаемые ЭС; инженеры по знаниям, являющиеся специалистами по разработке интеллектуальных информационных систем (ИИС); программисты, осуществляющие реализацию ЭС. Любая ЭС должна иметь, по крайней мере, два режима работы: режим приобретения знаний; режим консультаций. В режиме приобретения знаний эксперт наполняет ЭС знаниями, которые позволят ЭС в дальнейшем решать конкретные задачи из описанной проблемной области. Эксперт описывает проблемную область в виде совокупности данных об объектах и правил, определяющих взаимные связи между данными, и способы манипулирования данными. В режиме консультаций пользователь ЭС сообщает системе конкретные данные о решаемой задаче и стремится получить с её помощью результат. При этом входные данные о задаче поступают в рабочую память. Решатель на основе данных из БД и правил из БЗ формирует решение.

Динамические ЭС, наряду с компонентами статических ЭС, должны содержать: подсистему моделирования внешнего мира; подсистему связи с внешним окружением. Подсистема

моделирования необходима для прогнозирования, анализа и адекватной оценки состояния внешней среды. Изменения окружения решаемой задачи требуют изменения хранимых в ЭС знаний, для того чтобы отразить временную логику происходящих в реальном мире событий.

CLIPS, (от англ. C Language Integrated Production System) – программная среда для разработки экспертных систем [1]. CLIPS является продукционной системой. Реализация вывода использует алгоритм эффективный алгоритм сопоставления с образцом для продукционных систем, экспертных систем и баз знаний. Продукционная модель знания – модель, основанная на правилах, позволяет представить знание в виде предложений типа «Если (условие), то (действие)». Продукционная модель – фрагменты Семантической сети, основанные на временных отношениях между состояниями объектов. Продукционная модель обладает тем недостатком, что при накоплении достаточно большого числа (порядка нескольких сотен) продукций они начинают вследствие необратимости дизъюнкций противоречить друг другу. В этом случае разработчики начинают усложнять систему, включая в неё модули нечёткого вывода или иные средства разрешения конфликтов, – правила по приоритету, правила по глубине, эвристические механизмы исключений, возврата и т. п.

При реализации экспертная система проверяет применимость каждого правила вывода к каждому факту базы знаний, при необходимости выполняет его и переходит к следующему правилу, возвращаясь в начало при исчерпании всех правил. Экспертная система строит специальный граф или префиксное дерево, узлам которого соответствуют части условий правил. Путь от корня до листа образует полное условие некоторой продукции. В процессе работы каждый узел хранит список фактов, соответствующих условию. При добавлении или модификации факта он прогоняется по сети, при этом отмечаются узлы, условиям которых данный факт соответствует. При выполнении полного условия правила, когда система достигает листа графа, правило выполняется.

2.2. Пример выполнения работы

Первый шаг к выяснению того, как может быть реализовано в языке CLIPS обучающееся дерево решений, состоит в создании наиболее подходящего варианта представления знаний. Поскольку дерево решений должно обеспечивать обучение, по-видимому, целесообразно представить дерево в виде фактов, а не правил, в связи с тем, что факты могут быть легко добавлены и удалены для обновления дерева в ходе его обучения. Для перехода по дереву решений в ходе реализации алгоритма Solve_Tree_and_Learn с использованием подхода на основе правил может применяться множество правил CLIPS.

Каждый узел дерева решений будет представлен в виде факта. Для представления и узлов ответов, и узлов принятия решений будет использоваться следующая конструкция deftemplate:

```
(deftemplate node (slot name)
(slot type)
(slot question)
(slot yes-node)
(slot no-node)
(slot answer))
```

В этом определении слот name содержит уникальное имя узла, а слот type обозначает тип узла и содержит значение answer или decision. Слоты question, yes-node и no-node используются только в узлах принятия решений. Слот question содержит вопрос, который должен быть задан при переходе к узлу принятия решений. Слот yes-node указывает на узел, к которому должен быть выполнен переход, если ответ на вопрос является положительным, а слот no-node указывает на узел, к которому должен быть выполнен переход, если ответ на вопрос является отрицательным. Слот answer используется только в узлах ответа и представ-

ляет собой ответ, формируемый деревом решений после перехода к узлу ответа.

Поскольку эта программа идентификации животных должна обучаться, то возникает необходимость сохранять информацию о том, что было усвоено программой в процессе обучения от одного прогона к другому. В данном случае для представления структуры дерева решений используется коллекция фактов, поэтому было бы удобно сохранять эти факты в файле с применением формата команды load-facts, затем вносить их в список фактов с помощью команды load-facts в начале работы программы и снова сохранять с использованием команды save-facts после завершения работы программы. Факты, предназначенные для этой программы, хранятся в файле animal.dat. Файл animal.dat должен содержать текст, приведенный ниже. Обратите внимание на то, что корневой узел (root) обозначен как таковой, а каждому из прочих узлов присвоено уникальное имя. Кроме того, следует отметить, что некоторые слоты (такие как decision, yes-node и no-node для узлов ответа) остаются незаданными, поскольку им должны присваиваться значения, предусмотренные по умолчанию (а для разрабатываемой программы не важно, какие значения будут помещены в эти слоты).

```
(node (name root) (type decision)
(question "Is the animal warm-blooded?")
(yes-node nodel) (no-node node2))
(node (name nodel) (type decision)
(question "Does the animal purr?")
(yes-node node3) (no-node node4))
(node  (name  node2) (type  answer)  (answer  snake))
(node  (name  node3) (type  answer)  (answer  cat))
(node  (name  node4) (type  answer)  (answer  dog))
```

Теперь необходимо разработать правила, применяемые для прохождения по дереву решений. Инициализация программы обучения дерева решений осуществляется с помощью следующего правила:

```
(defrule initialize
(not (node (name root)))
=>
(load-facts "animal.dat")
(assert (current-node root)))
```

Запуск этого правила initialize происходит, если в списке фактов отсутствует факт, соответствующий корневому узлу. Действия, предусмотренные в правиле initialize, обеспечивают загрузку представления дерева решений в список фактов и внесение в список фактов того факта, который указывает, что текущим узлом, представляющим интерес, является корневой узел.

Пусть имеется дерево принятия решений :

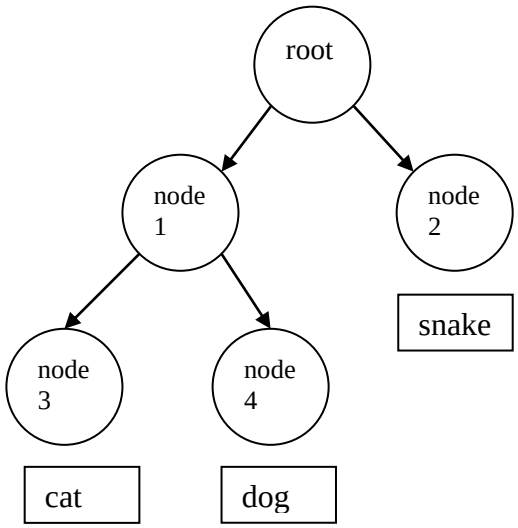


Рис. 2.1.

Вершины дерева типа decision: root, node1
Вершины дерева типа answer: node2, node3, node4

Содержимое файла ANIMAL.DAT:

(node (name root) (type decision) (question "Is the animal warm-blooded?") (yes-node node1) (no-node node2) (answer nil))
(node (name node1) (type decision) (question "Does the animal purr?") (yes-node node3) (no-node node4) (answer nil))
(node (name node2) (type answer) (question nil) (yes-node nil) (no-node nil) (answer snake))
(node (name node3) (type answer) (question nil) (yes-node nil) (no-node nil) (answer cat))
(node (name node4) (type answer) (question nil) (yes-node nil) (no-node nil) (answer dog))

2.3. Индивидуальное задание и методика выполнения работы

Эта часть РГЗ предназначена для построения студентом экспертной системы по выбору цифрового оборудования для последующего приобретения им.

В рамках РГЗ студент должен выполнить решение следующих задач на встроенном языке S-выражений ANIMAL.CLP (надстройка системы CLIPS [1]):

1. Построение дерева принятия решений для базового варианта ANIMAL.DAT.
2. Для предметной области (по варианту):
2.1. Перечислить и описать вершины дерева типа «decision».
2.2. Перечислить и описать вершины дерева типа «answer».
2.3. Построить дерево принятия решений на основе вершин типа «decision»и типа «answer».
2.4. Формирование нового дерева ANIMAL.DAT:
* для вершин дерева типа decision в формате:
(node (name root|node1) (type decision) (question "Вопрос?")
(yes-node node1) (no-node node2) (answer nil))
* для вершин дерева типа answer в формате:
(node (name node3) (type answer) (question nil)
(yes-node nil) (no-node nil) (answer ответ))

Для имен узлов, вопросов и ответов использовать транслитерацию, имена узлов и ответы содержат только малые английские буквы.

Оформить отчет о проделанной работе.

Вариант содержит предметную область, номер варианта - N выбирается по списку группы: N=(I mod 6) + 1.

№ варианта	Предметная область
1.	Ноутбук
2.	Смартфон
3.	Интернет-планшет
4.	Принтер
5.	Монитор
6.	ПК

Критерий оценивания результатов выполнения РГЗ:

№	Оценка	Количество вершин дерева типа decision	Количество вершин дерева типа answer
1.	Удовлетворительно	2–3	3
2.	Хорошо	4	4–5
3.	Отлично	5–6	8–12

2.4. Содержание отчета о выполнении индивидуального задания

Оформить отчет, содержащий:

Титульный лист.

Цель работы.

1. Постановка задачи (по варианту).

2. Ход работы.

2.1. Построенного дерева принятия решений для базового варианта ANIMAL.DAT.

2.2. Описание вершин дерева типа «decision».

2.3. Описание вершин дерева типа «answer».

2.4. Построение дерева принятия решений на основе вершин типа «decision» и типа «answer».

2.5. Текст программы описания правил ANIMAL.CLP, т.е. разработанный студентом ANIMAL.DAT.

2.6. Провести тестирование разработанной программы.

Вывод (отразить Ваш вклад в разработанную надстройку)

2.5. Контрольные вопросы

1. Что понимают под экспертной системой?
2. Назовите особенности экспертных систем.
3. Приведите структуру экспертной системы.
4. Какими качествами должна обладать ЭС?
5. Назовите трудности, возникающие при разработке ЭС.
6. Что называют метазнаниями?
7. Что дают пользователю экспертные системы?
8. Назовите типы задач, решаемых ЭС.
9. Какие действия выполняет диалоговый компонент ЭС?
10. Какие функции выполняет решатель?
11. Какие действия выполняет объяснительный компонент?
12. Опишите режимы работы ЭС.
13. Что означает термин «знание» в искусственном интеллекте?
14. Покажите различие между алгоритмическим и эвристическим методами.
15. Опишите режим консультации ЭС.

16. Что входит в компонент приобретения знаний?
17. Что такое система, основанная на знаниях?

2.6. Библиографический список

1. Джарратано, Джозеф, Райли, Гари. Экспертные системы: принципы разработки и программирования, 4-е издание. М.: Вильямс, 2007 – 1152 с.
2. Хендерсон П. Функциональное программирование. –М.:Мир,1983.– 347 с.
3. Э.Хювенен, Й.Сеппянен «Мир Лиспа» т.1 и т.2 М. 1990.- М.: Мир, 1982.-584 с.
4. Л.В. Городняя «Основы функционального программирования». – Новосибирск 2004. – 126 с.