# 3. ЛАБОРАТОРНАЯ РАБОТА № 3 «РАЗРАБОТКА И ИССЛЕДОВАНИЕ ЭКСПЕРТНОЙ СИСТЕМЫ»

## 3.1. Цель работы

Разработка экспертной системы продукционного типа на Прологе, исследование базовых принципов организации экспертных систем.

## 3.2. Краткие теоретические сведения

### 3.2.1. Структура экспертных систем

Под *экспертной системой* (ЭС) понимают программную систему, аккумулирующую знания эксперта в определенной области и вырабатывающую решения и рекомендации на уровне эксперта. Перечень типовых задач, решаемых ЭС в самых различных областях, включает [1,9]:

- интерпретацию — извлечение информации из первичных данных (распознавание образов, определение состава вещества и др.);
- диагностику — обнаружение неисправностей и причин их появления в некоторой системе (медицинской, механической, электронной и др.);
- мониторинг — непрерывная интерпретация данных в реальном времени с сигнализацией о выходе тех или иных параметров за допустимые пределы (контроль движения транспорта, наблюдение за состоянием энергетических объектов и др.);
- прогноз — предсказание вероятных последствий на основе прошедших и настоящих событий (предсказание погоды, прогноз ситуаций на финансовых рынках и др.);
- планирование — определение последовательности действий, направленных на достижение заранее поставленных целей (планирование поведения роботов, составление маршрутов движения транспорта и др.);
- проектирование — определение конфигурации системы при заданных ограничениях (синтез электронных схем, оптимальное размещение объектов в ограниченном пространстве и др. );
- отладку и ремонт — выполнение последовательности действий по приведению той или иной системы к требуемым режимам функционирования (помощь при отладке программного обеспечения, ремонт инженерных коммуникаций и др.);
- обучение — интерпретация, диагностика и коррекция знаний и умений обучаемого;
- управление — формирование управляющих воздействий, определяющих поведение сложных систем (управление воздушным транспортом, военными действиями, деловой активностью в сфере бизнеса и др.).

Типовая архитектура ЭС изображена на рисунке 3.1. Взаимодействие с ЭС осуществляется с помощью интерфейса. Кроме обеспечения взаимодействия с различными категориями пользователей, интерфейс (если необходимо) выполня-

ет функции сопряжения с внешними объектами: базами данных, различными датчиками, коммуникационным оборудованием и т.п.

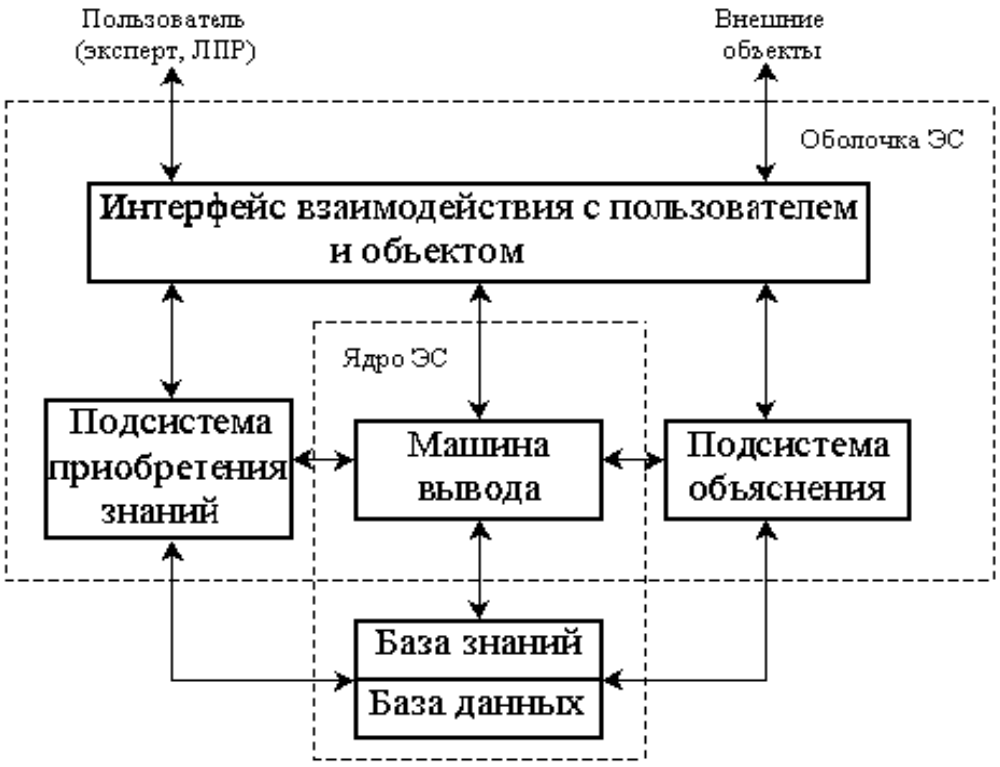


Рисунок 3.1 — Структурная схема ЭС

*Ядро ЭС* образует база данных, база знаний и машина вывода. *База данных* представляет собой рабочую память, в которой хранятся текущие данные, заключения и другая информация, имеющая отношение к анализируемой системой ситуации. *База знаний* обеспечивает хранение знаний, представленных с помощью одной из моделей: логической, продукционной, фреймовой, сетевой и др.

*Машина вывода* (подсистема поиска решений), используя данные и знания, организует управление выводом в соответствии с используемой моделью представления знаний. Целью вывода является получение заключений, согласующихся с той информацией, которая содержится в базе знаний.

*Подсистема объяснения* ЭС позволяет пользователю выяснить, как система получила решение задачи, и какие знания были при этом использованы.

*Подсистема приобретения знаний* используется как с целью автоматизации процесса наполнения ЭС знаниями, так и при корректировке базы знаний, при ее обновлении, пополнении или исключении элементов знаний.

### 3.2.2. Методы вывода в ЭС продукционного типа

При создании ЭС используют различные модели представления знаний: логические, продукционные, фреймовые, сетевые [1,4,5,9]. Наибольшее распространение получили ЭС продукционного типа , в которых база знаний представляется в виде набора правил-продукций: “если *A*, то *B* “. Здесь *A* и *B* могут пониматься как “ситуация — действие”, “причина — следствие”, “условие — заключение” и т.п.

В общем случае продукционная система включает следующие компоненты (рис. 2.2): базу продукционных правил, базу данных (рабочую память) и интерпретатор.

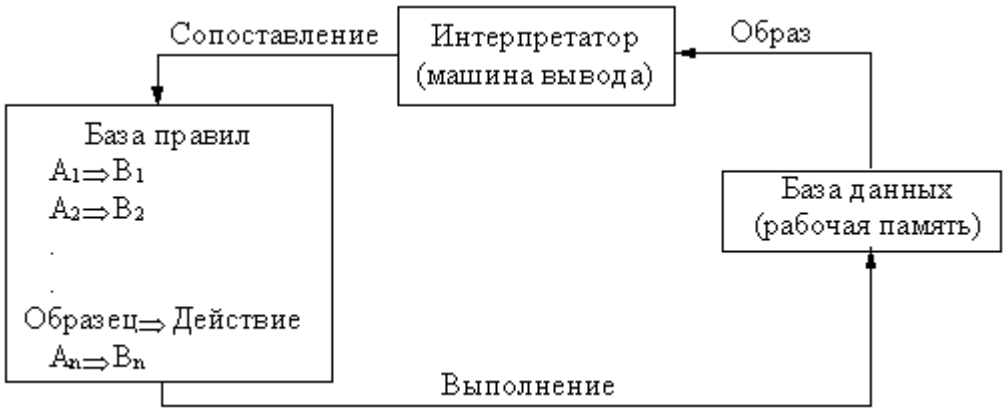


Рисунок 3.2 — Продукционная система

Вывод осуществляется посредством *поиска и сопоставления по образцу*. *Образец* — это некоторая информационная структура, определяющая общие условия, при которых происходит вызов (активизация) правила.

Логический вывод реализуется интерпретатором, который выбирает и активизирует правила. Работу интерпретатора можно описать последовательностью из четырех шагов:

- выбор элементов данных (фактов) из рабочей области;
- сопоставление фактов с образцами правил, если сопоставимы несколько правил, то интерпретатор формирует конфликтный набор правил;
- разрешение конфликтов;
- выполнение выбранного правила.

В продукционных системах применяются два метода вывода — прямой вывод и обратный вывод.

*Прямой вывод* начинается с задания исходных данных решаемой задачи, которые фиксируются в виде фактов в рабочей памяти системы. Правила, предпосылки которых сопоставимы с исходными фактами, обеспечивают генерацию новых фактов, добавляемых в рабочую память. Процесс применения правил к новым фактам продолжается, пока не будет получено целевое состояние рабочей памяти [1,5,10] .

*Обратный вывод*, т.е. вывод управляемый целевыми условиями, начинается с внесения целевого утверждения в рабочую память. Затем отыскивается правило-продукция, заключение которого сопоставимо с целью. Условия применения данного правила помещаются в рабочую память и становятся новой подцелью. Процесс повторяется до тех пор, пока в рабочей памяти не будут найдены необходимые факты, подтверждающие все подцели [1,5,9] .

Отметим, что ситуация, когда в рабочую память заранее вносятся все известные факты, встречается редко. Обычно в рабочей памяти изначально содержится только часть фактов, необходимых для вывода. Остальные сведения система выясняет сама, задавая вопросы пользователю.

### 3.2.3. Формирование объяснений

Для получения объяснений выводов пользователи ЭС могут задавать вопросы двух типов:

- *почему* система сочла необходимым задать пользователю определенный вопрос;
- *как* система пришла к соответствующему заключению.

Выясним механизм формирования ответов на эти вопросы на примере *обратного вывода*. Пусть имеются правила для диагностики неисправностей электрической плиты, изображенные на рис.3.3. Здесь в состав правил включены утверждения, записанные в форме предикатов, например: **лампа(светится), плита(холодная)** и др.

```
правило1 :: если лампа(светится) и плита(холодная)
            то нагреватель(неисправен).
правило2 :: если тока(нет)
            то выключатель(не_включен).
правило3 :: если тока(нет)
            то напряжения(нет).
правило4 :: если плита(холодная) и лампа(не_светится)
            то тока(нет).
правило5 :: если лампа(не_светится) и плита(горячая)
            то лампа(неисправна).
```

Рисунок 3.3. — Правила диагностики неисправностей электрической плиты

Процесс вывода начинается с ввода в рабочую память возможной гипотезы (причины) неисправности, например, **гипотеза(X)**, где **X** – переменная, которая может быть сопоставлена с любой из возможных причин неисправностей, известных системе: **выключатель(не_включен), нагреватель(неисправен), напряжения(нет), лампа(неисправна).**

Если гипотезы проверяются в порядке их записи, то на первом шаге **X** получит значение **выключатель(не_включен)** и система выберет правило 2, которое своим заключением подтверждает эту гипотезу. Далее в соответствии с предпосылками правила 2 система попытается установить справедливость утверждения **тока(нет).** В соответствии с правилом 4 это заключение верно, если имеются факты**: плита(холодная) и лампа(не_светится).** Так как эти факты в рабочей памяти не содержатся, и нет правил, с помощью которых система могла бы установить их значения, то пользователю будут заданы соответствующие вопросы (ответы пользователя набраны курсивом):

**плита(холодная)?**
***:-да.***
**лампа(не_светится)?**
***:-да.***

После этого система сообщает причину неисправности плиты: **решение: выключатель(не_включен).**

Если пользователь желает выяснить, почему система задает тот или иной вопрос, то в качестве ответа на вопрос вводится слово "*почему*". Например:

**лампа(не_светится)?**
***:-почему.***
**пытаюсь доказать лампа(не_светится)**
**с помощью правила: правило4**
**лампа(не_светится)?**
**:-**

В этом случае система демонстрирует пользователю номер текущего правила, с которым был связан заданный вопрос.

Чтобы ответить на вопрос "*как*" в системе сохраняется *дерево решений*, принятых в течение сеанса работы. Просматривая дерево, можно выяснить, достижение каких подцелей привело к данному решению. Для этого после предъявления решения пользователю система может объяснить, как она его получила. Например, возможен следующий сценарий:

**решение: выключатель(не_включен)**
**объяснить ? [цель/нет]:**
***:-выключатель(не_включен).***
**выключатель(не_включен) было доказано с помощью**
**правило2 :: если тока(нет) то выключатель(не_включен)**
**объяснить ? [цель/нет]:**
***:-тока(нет).***
**тока(нет) было доказано с помощью**
**правило4 :: если плита(холодная) и лампа(не_светится) то тока(нет)**
**объяснить ? [цель/нет]:**
***:-плита(холодная).***
**плита(холодная) было введено пользователем**
**объяснить ? [цель/нет]:**
***:-нет.***

Таким образом, формирование ответа на вопрос "как" соответствует просмотру дерева решений в направлении от целевого утверждения до листьев дерева, соответствующих введенным фактам.

Сложнее обстоит дело при прямом выводе, т.е. выводе, управляемом данными. Прямая цепочка рассуждений обеспечивает пользователя менее полезной информацией. Обусловлено это тем, что на промежуточных этапах вывода трудно судить, куда ведет цепочка рассуждений. Так, в ответ на вопрос "почему" пользователю демонстрируется текущее правило. Дальнейшие объяснения не могут быть получены, пока не выполнится следующее правило. Кроме этого, сложно формировать полные ответы на вопрос "как", даже после достижения цели. Информация, которая предоставляется, ограничивается списком выполненных правил.

### 3.3.4. Реализация продукционной ЭС на языке Пролог

*Простейший способ* построения ЭС на Прологе — использование в качестве интерпретатора механизма поиска решений, заложенного в Пролог-системе [1,2,4]. Проиллюстрируем это на примере задачи диагностики неисправности электрической плиты. Представим правила, изображенные на рис. 3.3., в виде предикатов языка Пролог:

```
/* структура правила:  причина:– неисправность */
нагреватель(неисправен):– лампа(светится),  плита(холодная).    %правило 1
выключатель(не_включен):– тока(нет).                            %правило 2
напряжения(нет):– тока(нет).                                    %правило 3
тока(нет):– плита(холодная),  лампа(не_светится).               %правило 4
лампа(неисправна):– лампа(не_светится), плита(горячая).         %правило 5
```

Определить причину неисправности плиты можно, указав перечень гипотетических неисправностей и обеспечив проверку выполнимости каждой из гипотез:

```
лампа(не_светится).                    % признаки неисправности
плита(холодная).

гипотеза (нагреватель(неисправен)).    % гипотезы
гипотеза (выключатель(не_включен)).
гипотеза (напряжения(нет)).
гипотеза (лампа(неисправна)).

найти(X): – гипотеза (X), X.           % проверка гипотез/
```

Для того чтобы узнать, какая из гипотез подтверждается, необходимо задать вопрос: **найти(X).** Получив такой вопрос, Пролог-система выполнит необходимый обратный поиск с помощью встроенного механизма вывода и вернет ответ:

```
X = выключатель(не_включен);
X = напряжения(нет).
```

Недостатком рассмотренной программы является необходимость внесения наблюдаемых признаков неисправностей непосредственно в текст программы в виде фактов. Устранить указанный недостаток можно, запросив значения соответствующих признаков неисправностей у пользователя в процессе выполнения программы:

```
лампа(не_светится): – спроси('Лампа не светится ?').
плита(холодная): – спроси('Плита холодная ?').
лампа(светится): – спроси('Лампа светится ?').
плита(горячая): – спроси('Плита горячая ?').
спроси(Вопрос): – write(Вопрос), nl,read('да').
```

В этом случае система будет задавать вопросы пользователю относительно неизвестных фактов. Так как ответы пользователя не запоминаются, то возможно

повторение вопросов. Для запоминания ответов следует воспользоваться встроенным предикатом **assert (**см. ниже). Иным существенным недостатком простейшего способа реализации ЭС является запись продукционных правил в виде кода на языке Пролог.

*Усовершенствованный вариант* реализации ЭС предполагает запись правил более естественной форме, изображенной на рис. 3.3. Чтобы программа могла интерпретировать правила в такой форме, необходимо определить операторы:

```
определить_операторы: – op(920,  xfy, и),  op(950, xfx, то),
                        op(960, fx, если), op(970, xfx, '::').
: – определить_операторы.
```

Тогда для указанных правил (рис. 3.3) перечень возможных гипотез неисправностей запишется в виде:

```
% гипотезы неисправностей представляются структурой – Имя_факта :: Факт
h1 :: гипотеза(нагреватель(неисправен)).
h2 :: гипотеза(выключатель(не_включен)).
h3 :: гипотеза(напряжения(нет)).
h4 :: гипотеза(лампа(неисправна)).
```

Перечислим также признаки неисправностей, значения которых можно запрашивать у пользователя:

```
% признаки, истинность которых можно выяснить у пользователя
q1 :: признак(лампа(светится)).
q2 :: признак(плита(холодная)).
q3 :: признак(лампа(не_светится)).
q4 :: признак(плита(горячая)).
```

Реализуем интерпретатор ЭС в виде предиката **найти(H),** где **H** — возможная гипотеза, которую требуется подтвердить или опровергнуть. При этом возможны четыре случая:

1) гипотеза **H** подтверждается фактом, уже известным системе;

2) гипотеза **H** соответствует следствию одного из правил, тогда для подтверждения гипотезы необходимо доказать справедливость предпосылок правила;

3) гипотеза **H**, доказываемая на некотором шаге вывода, представляет собой конъюнкцию условий правила, т.е. **H1** и **H2**, тогда необходимо доказать достижимость конъюнкции подцелей: **найти(H1)** и **найти(H2)**;

4) гипотеза **H** сопоставима с одним из признаков, на основе которых устанавливаются причины неисправности, тогда необходимо задать соответствующий вопрос пользователю.

Для того чтобы исключить повтор вопросов, ответы пользователя будем запоминать в базе данных в виде фактов **сообщено(Факт, 'да|нет').** Запоминание соответствующих фактов в базе данных будем выполнять с помощью предиката **assert.** Гипотезы, относительно которых можно задавать вопросы, определяются с помощью предиката **запрашиваемая(H).** Такие гипотезы сопоставимы с призна-

ками. С учетом сказанного, интерпретатор опишется следующей совокупностью правил языка Пролог:

```
найти(Н):- Факт :: Н.                                  % случай 1
найти(Н):- Правило :: если Н1 то Н,  найти(Н1).        % случай2

найти(Н1 и Н2):-найти(Н1), найти(Н2).                  % случай 3
найти(Н):- запрашиваемая(Н),                           % случай 4
           сообщено(Н,да).                             % вопрос уже был
найти(Н):- запрашиваемая(Н),                           % случай 4
           not(сообщено(Н,_)),                         % вопроса  не было
           спроси(Н).

запрашиваемая(Н):- Факт :: признак(Н).

спроси(Н):- nl, write(Н), write('?'), nl,              % вывод вопроса
            read(О), ответ(Н,О).                       % ввод ответа
%обработка ответов
ответ(Н,да):-   assert(сообщено(Н,да)), !.             % добавить ответ в БД
ответ(Н,нет):- assert(сообщено(Н,нет)), !, fail.
ответ(Н,_):- write('правильный ответ: да, нет'), nl,
           спроси(Н).                                  % повторить вопрос
```

Вызов интерпретатора с целью проверки всех гипотез выполняется следующим образом:

```
инициализация:-retractall(сообщено(_,_)).
диагностика(Н):-   инициализация, Факт :: гипотеза(Н), найти(Н).
```

Улучшение разработанной ЭС связано с включением возможности ответов на вопросы типа “почему?” и “как?”. Базовый текст программного кода соответствующей ЭС приведен в приложении В.1. Детальное объяснение программы изложено в [1]. В приложение В.2 включен программный код интерпретатора, позволяющего использовать правила, предпосылки которых могут содержать произвольное количество условий. Для этого предпосылки правил представляются в виде списка условий.

### 3.3. Варианты заданий

Реализовать продукционную экспертную систему в соответствии с номером варианта, указанного в таблице 3.1. При этом количество рассматриваемых объектов предметной области должно быть не менее 10 и характеризующих их атрибутов также — не менее 10. Система должна уметь давать объяснения вывода. Задачу, решаемую ЭС, выбрать самостоятельно с учетом перечня задач, указанного в п.3.2.1, или с помощью каталога ЭС, приведенного в [10].

Таблица 3.1— Предметная область экспертной системы

| Вариант | Предметная область |
|---|---|
| 1, 16 | Базы данных |
| 2, 17 | Транспорт |
| 3, 18 | Обучение |

| | |
|---|---|
| 4, 19 | Компьютерные сети |

Продолжение таблицы 3.1.

| Вариант | Предметная область |
|---|---|
| 5, 20 | Операционные системы |
| 6, 21 | Мобильные устройства |
| 7, 22 | Микропроцессоры |
| 8, 23 | Языки программирования |
| 9, 24 | Компьютерные игры |
| 10, 25 | Математика |
| 11, 26 | Медицина |
| 12, 27 | Метеорология |
| 13, 28 | Сельское хозяйство |
| 14, 29 | Электроника |
| 15, 30 | Юриспруденция |

### 3.4. Порядок выполнения лабораторной работы

3.4.1. Изучить модели представления знаний по лекционному материалу и учебным пособиям [1,5,8 ].

3.4.2. Детально изучить организацию продукционной системы, механизмы прямого и обратного вывода, основы построения подсистемы объяснения, примеры реализации ЭС на Прологе [1,2,4,10].

3.4.3. Выполнить анализ предметной области ЭС в соответствии с вариантом задания:

а) выбрать задачу, решаемую ЭС, и определить цели системы: конечные, промежуточные и вспомогательные;

б) выделить подзадачи, которые следует решить для достижения цели;

в) разработать продукционную базу правил для решения выделенных подзадач.

3.4.4. Ознакомиться с примерами программных кодов, приведенных в приложении В, и, по аналогии, разработать ЭС продукционного типа для решения задачи в заданной предметной области.

3.4.5. Создать в среде программирования Пролог проект ЭС и выполнить его отладку.

3.4.6. Исследовать свойства разработанной системы. Получить протоколы работы системы при доказательстве различных целевых утверждений.

3.4.7. Зафиксировать результаты работы программы в виде экранных копий.

### 3.5. Содержание отчета

Цель работы, вариант задания, описание предметной области, структура базы знаний и описание продукционных правил, описание разработанного интерпретатора продукционных правил, описание машинных экспериментов с ЭС и анализ результатов, выводы.

### 3.6. Контрольные вопросы

3.6.1. Приведите определение экспертной системы.

3.6.2. Перечислите типовые задачи, решаемые с помощью ЭС.

3.6.3. Назовите основные компоненты ЭС и объясните их функции.

3.6.4. Назовите основные этапы разработки ЭС, перечислите задачи, решаемые на каждом из этапов.

3.6.5. Объясните термины “приобретение знаний” и “извлечение знаний”.

3.6.6. Сформулируйте обобщенный алгоритм прямого вывода на правилах-продукциях.

3.6.7. Сформулируйте обобщенный алгоритм обратного вывода на правилах-продукциях.

3.6.8. Объясните суть вопросов “почему” и “как”, используемых для формирования объяснений вывода в ЭС.

3.6.9. Объясните на примере механизм формирования ответа на вопрос типа “как”.

3.6.10. Объясните на примере механизм формирования ответа на вопрос типа “почему”.

3.6.11. Как можно представить базу продукционных правил на языке Пролог?

3.6.12. Приведите пример простейшей реализации ЭС на языке Пролог.

3.6.13. Какие правила положены в основу предиката **найти(H)**, осуществляющего обратный вывод на множестве продукционных правил?

3.6.14. Напишите на языке Пролог простейший вариант реализации предиката **найти(H)**.

3.6.15. Напишете на языке Пролог фрагмент кода, обеспечивающий вывод вопросов, задаваемых ЭС, и анализ ответов.

3.6.16. Напишите на языке Пролог фрагмент кода, обрабатывающий вопросы типа «почему».