Лекция № 4

Использование языка PROLOG для построения интеллектуальных систем

Using the PROLOG language for building intelligent systems

Понятие экспертных систем

Под **экспертными системами** (ЭС) будет понимать программные системы, которые при решении задач, трудных для эксперта-человека (т.е. квалифицированного специалиста в данной предметной области), получают результаты, не уступающие по качеству и эффективности решениям, получаемым самим экспертом.

Для того, чтобы получать такие решения ЭС, должна обладать знаниями эксперта. Поэтому главным компонентом ЭС является ее база знаний. Поэтому есть еще одно определение ЭС:

ЭС – это класс программных систем, основанных на знаниях. Их вычислительные возможности определяются в 1-ю очередь наращиваемой базой знаний и только во 2-ю очередь используемыми методами.

Понятие неформализованной задачи

Экспертные системы предназначены для решения так называемых неформализованных задач (ill-structured problems).

Согласно А. Ньюэллу и М. Саймону, к неформализованным относятся такие задачи, которые обладают одной или несколькими из следующих характеристик:

- задачи не могут быть заданы в числовой форме;
- цели не могут быть выражены в терминах точно определенной целевой функции;
- не существует алгоритмического решения задачи; либо оно существует, но его нельзя использовать из-за ограниченности ресурсов (времени, памяти).

Следует подчеркнуть, что неформализованные задачи составляют большой и очень важный класс задач, представляющих большое практическое значение.

Особенности и назначение экспертных систем

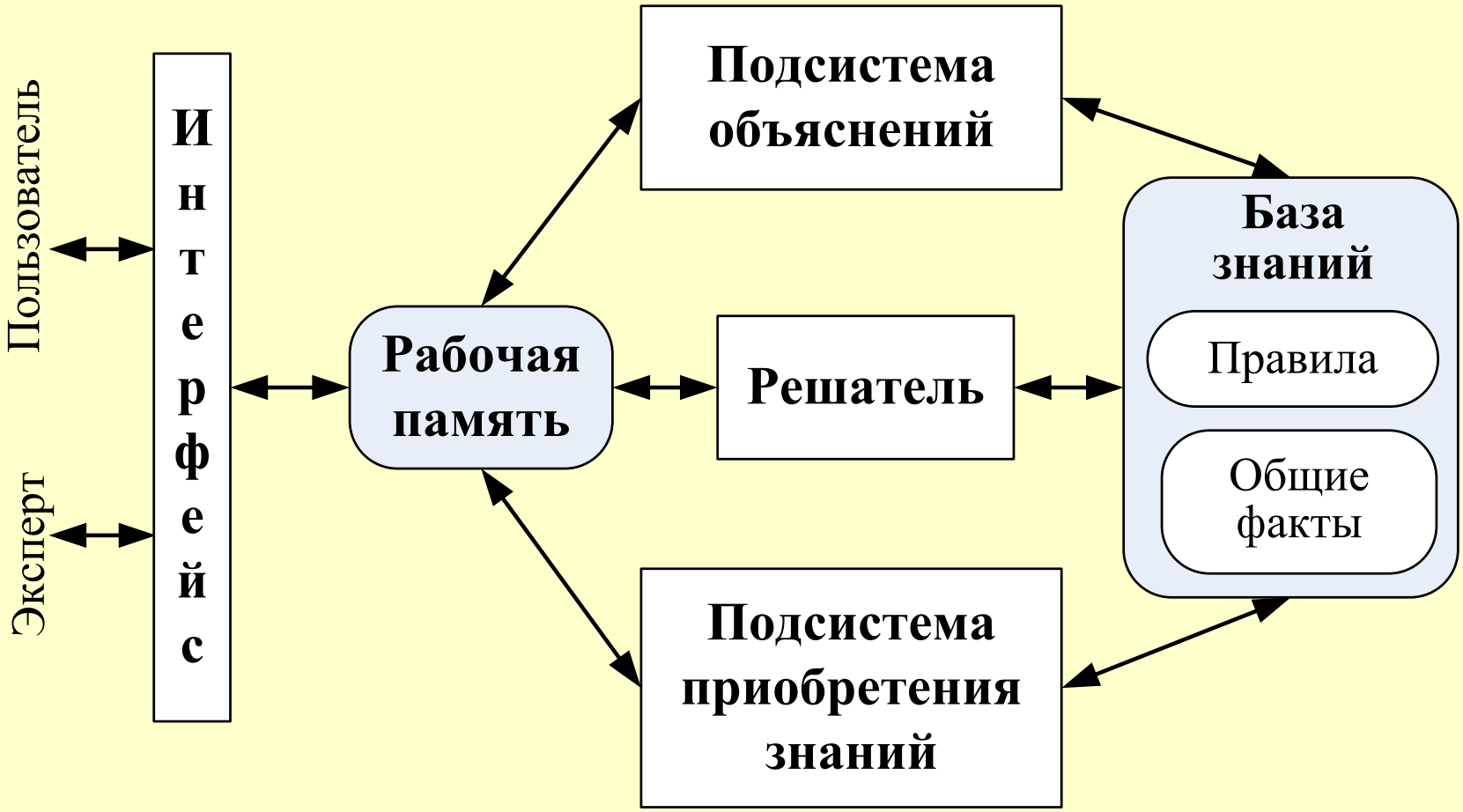
ЭС предназначены для решения неформализованных задач и обладают следующими особенностями:

- **Алгоритм** решения задач **не известен заранее**, а строится самой ЭС с помощью символических рассуждений, базирующихся на эвристических приемах;
- «Прозрачность» (ясность) полученных решений. Т.е. **система** «осознает» в терминах пользователя, как она получила решение, и **может** проанализировать и **объяснить свои действия и знания**;
- **Способность приобретения новых знаний** от пользователя-эксперта и изменения в соответствии с ними своего поведения;
- Обеспечение «дружественного» интерфейса.

Особенности и назначение экспертных систем

Таким образом, экспертные системы, как и другие интеллектуальные системы, отличаются от систем обработки данных тем, что в них в основном используются символьный (а не числовой) способ представления, символьный вывод и эвристический поиск решения (а не простое исполнение известного алгоритма).

Структура типовой экспертной системы



Структура типовой экспертной системы

Рабочая память (РП) предназначена для хранения исходных и промежуточных данных решаемой в текущий момент задачи.

База знаний (БЗ) включает универсальные данные (факты), описывающие рассматриваемую область знаний, и множество правил, описывающих целесообразные преобразования данных в этой области.

Решатель, используя исходные данные из РП и знания из БЗ, формирует такую последовательность правил, которые, будучи примененными к исходным данным, приводят к решению задачи.

Структура типовой экспертной системы

Подсистема приобретения знаний автоматизирует процесс наполнения ЭС знаниями, осуществляемый, как правило, пользователем-экспертом.

Подсистема объяснений объясняет, как система получила решение задачи (или почему она не получила решение) и какие знания она при этом использовала, что облегчает эксперту тестирование системы и повышает доверие пользователя к полученному результату.

Интерфейс обеспечивает дружественное общение с пользователем как в ходе решения задач, так и в процессе приобретения знаний и объяснения результатов работы.

В разработке ЭС участвуют

1) **эксперт** – носитель знаний в проблемной области, задачи которой будет решать ЭС;

2) **инженер знаний** – специалист по представлению знаний;

3) **программист** – специалист по программированию.

Следует заметить, что отсутствие среди участников разработки инженеров знаний (т. е. их замена программистами) либо приводит к неудаче процесс создания ЭС, либо значительно удлиняет его.

Режимы работы ЭС

Экспертная система работает в двух режимах:

1) режиме приобретения знаний и
2) в режиме решения задачи (называемом также режимом консультации или режимом использования ЭС).

В режиме приобретения знаний общение с ЭС осуществляет (через посредничество инженера знаний) эксперт. В этом режиме эксперт, используя компонент приобретения знаний, наполняет систему знаниями, которые позволяют ЭС в режиме решения решать задачи из проблемной области.

Эксперт описывает проблемную область в виде совокупности данных и правил. Данные определяют объекты, их характеристики и значения, существующие в области экспертизы. Правила определяют способы манипулирования с данными, характерные для рассматриваемой области.

В режиме решения задачи общение с ЭС осуществляет конечный пользователь, которого интересует результат решения задачи и, как правило, способ его получения.

Язык Prolog и экспертные системы

Рассмотрим применение языка Prolog для создания экспертных систем, в частности, систем классификации и диагностики.

Напомним, что **экспертные системы** (ЭС) – это программные системы, использующие в своей работе опыт и знания наиболее квалифицированных специалистов (экспертов) в определенной предметной области.

Основными "кирпичиками" экспертных систем являются знания и механизм вывода, управляющий процессами автоматического рассуждения на основе имеющихся знаний.

Организация логического вывода в ЭС

Есть два способа организации логического вывода –
на основе прямой и на основе обратной цепочки рассуждений.

Первый метод (**прямой вывод**) использует рассуждения для нахождения различных предположений, обусловленных имеющимися фактами и правилами.

Второй метод (**обратный вывод**) основан на исследовании заключений, которые представляют интерес и могут быть (а могут и не быть) истинными.

При реализации ЭС на Прологе удобно использовать обратный вывод, так как базовый механизм вывода в Прологе основан на этой стратегии вывода.

Примеры построения экспертных систем

Рассмотрим пример создания программы, реализующей игру-загадку "Птица, зверь или рыба".

Смысл этой игры состоит в том, что играющий задумывает конкретное животное, а система пытается угадать его. Для этого система задает вопросы, запрашивая необходимую для угадывания информацию.

В этой игре возможен следующий диалог:

Вопрос	**Ответ**
Оно покрыто шерстью?	Нет
Оно имеет перья?	Да
Оно летает?	Нет
Оно плавает?	Да
Оно имеет черно-белый цвет?	Да

Примеры построения экспертных систем

Эта программа выбрана в качестве примера по двум причинам:

1) очень легко объяснить ее назначение и логику работы;
2) при кажущейся простоте этой программы использованный в ней подход пригоден также и для создания практически полезных систем классификации и диагностики.

Экспертная система с обратным выводом

Программа содержит группу правил высокого уровня, каждое из которых описывает только одно животное.

Например:

```
animal ("пингвин") :- it_is ("птица"),
                      not (yes ("летает")),
                      yes ("имеет перья"),
                      yes  ("плавает"),
                      yes ("имеет черно-белый цвет"), !.
```

В правой части правил высокого уровня встречаются предикаты двух типов **it_is** и **yes** (с отрицанием или без него).

Экспертная система с обратным выводом

Предикаты с именем **it_is** определяют, к какому классу относится животное, и всегда описываются через другие правила.

```
it_is ("птица") :-  not (it_is ("млекопитающее")),
                          yes ("имеет перья").
it_is ("млекопитающее"):- yes ("вскармливает детенышей молоком").
```

Предикаты с именем **yes** проверяют наличие определенного признака у животного.

```
yes (X) :- db_yes (X), !.

yes (X) :- not (no (X)), !,
           check_if (X).

no (X) :- db_no (X), !.
```

Экспертная система с обратным выводом

Заметим, что здесь используются предикаты базы данных **db_yes/1** и **db_no/1.**

Эти предикаты должны быть объявлены как динамические:
:- dynamic db_yes/1, db_no/1.

Экспертная система с обратным выводом

```
yes (X) :- db_yes (X), !.
yes (X) :- not (no (X)), !,
           check_if (X).
no (X) :- db_no (X), !.
```

Первое правило **yes (X)** непосредственно обращается к базе данных (предикат **db_yes/1**), чтобы получить информацию о наличии признака. Если такой информации там нет, она запрашивается у пользователя вторым правилом.

Второе правило **yes (X)** сначала проверяет, что в базе данных отсутствует и информация о том, что такого признака нет (предикат **db_no/1** представляет отрицание признака), а потом запрашивает эту информацию у пользователя.

Экспертная система с обратным выводом

```
yes (X) :- db_yes (X), !.
yes (X) :- not (no (X)), !,
           check_if (X).
no (X) :- db_no (X), !.
```

Сам запрос информации осуществляется предикатом **check_if/1:**

```
check_if (X) :- write ("Оно "), write (X), writeln (" ?"),
                read (Reply),
                remember (Reply, X).
remember (да, X) :-   asserta (db_yes (X)).
remember (нет, X) :-  asserta (db_no (X)),
                      fail.
```

Здесь предикат **asserta/1** заносит ответы в базу данных.

Экспертная система с обратным выводом

Работа программы инициируется следующей целью:

```
game :- retractall (db_yes(_)),
        retractall (db_no(_)),
        animal (X),
        write ("Задуманное вами животное - "),
        write (X).
```

Вычисляя подцель **animal (X)**, система пытается по очереди установить истинность каждого из правил высокого уровня.

Найдя первое истинное правило, система выдает название животного, которого это правило описывает, и прекращает работу.

Заметим что порядок вопросов будет зависеть от расположения правил в программе, так как они будут выбираться для обработки поочередно в порядке написания. Из-за этого одни диалоги будут длиннее, другие - короче.

Экспертная система с обратным выводом

Перед новым запуском программы необходимо удалить из базы данных все ответы, оставшиеся после предыдущего запуска.

Для этого можно, использовать предикат **retractall (db_yes (_))** и **retractall (db_no (_)).**

ЭС с настраиваемой базой знаний

Описанная выше программа имеет недостаток – она не является настраиваемой, поскольку добавление нового животного требует изменения текста самой программы.

Изменять набор животных, не меняя текста программы, можно, но для этого нужно отделить механизм вывода от самих знаний.

При таком подходе знания хранятся в отдельном файле и при запуске программы считываются в оперативную память.

ЭС с настраиваемой базой знаний

За основу возьмем предыдущую программу. В новой программе правила верхнего уровня заменим на предикаты (факты) базы данных **rule**, содержащие данные о животном, и **property**, хранящие признаки, характеризующие животных.

Предикат **rule** имеет следующий вид:

rule (NumberRule, Category, Properties),

где **NumberRule** – номер правила (целое), **Category** – вид животного (строка), **Properties** – набор признаков (список целых).

Описание предиката **property** имеет вид:

property (NumberAttribute, Specification),

здесь **NumberAttribute** – номер признака (целое), а **Specification** – его содержание (строка).

ЭС с настраиваемой базой знаний

Тогда правило, описывающее пингвина, может быть представлено следующим фактом базы данных:

rule (1, "пингвин", [2, 5, 6]),

а номера признаков заданы такими фактами:

property (1, "имеет шерсть").

property (2, "имеет перья").

property (3, "имеет плавники").

property (4, "летает").

property (5, "плавает").

property (6, "имеет черно-белый цвет").

ЭС с настраиваемой базой знаний

Основные правила механизма вывода имеют следующий вид:

```
animal (X) :-    rule (_, X, Property),
                 check_property (Property).

check_property ([N | Property]) :-    property (N, A),
                                      yes (A),
                                      check_property (Property).

check_property ([ ]).
```

Правило **animal/1** осуществляет выбор из базы данных описания очередного животного и проверяет наличие соответствующих ему признаков (**Property**). При удачной проверке выбранное животное выдается в качестве ответа.

Непосредственная проверка признаков осуществляется предикатом **check_property**, который использует предикаты низкого уровня (**yes**, **no**, **remember** и др.) из первой программы.

ЭС с настраиваемой базой знаний

Работа этой программы инициируется такой же целью, как и первой программы:

```
game :- retractall (db_yes(_)),
        retractall (db_no(_)),
        animal (X),
        write ("Задуманное вами животное - "),
        write (X).
```

Построение ситемы объяснений

Такое представление знаний о предметной области – в виде декларативного описания правил и свойств объектов – позволяет достаточно просто строить объяснения.

В базе данных хранится информация о том, на какие вопросы системы были даны положительные ответы (предикаты **db_yes/1** и **db_no/1**). На их основе можно объяснить, почему был выбран тот или иной объект (печатая выбранные свойства, используя предикат **property/2**).

Можно также отвечать на вопросы пользователя о свойствах объектов (например, признаках животного или симптомах заболевания), описанных в базе знаний системы предикатами **rule/3** и **property/2**.

Код программы

```
:- dynamic db_yes/1, db_no/1. % Вводим предикаты базы данных

rule(1, "пингвин", [2, 5, 6]). % Вводим правила высокого уровня
rule(2, "медведь", [1, 7, 8]).

property(1, "имеет шерсть"). % Вводим описания признаков животных
property(2, "имеет перья").
property(3, "имеет плавники").
property(4, "летает").
property(5, "плавает").
property(6, "имеет черно-белый цвет").
property(7, "имеет бурый цвет").
property(8, "лазает по деревьям").
```

Код программы

```
/* Основные правила механизма вывода */
animal(X) :-  rule(_, X, Property),
              check_property(Property).
animal(_) :-  write("Такого животного я не знаю."),
               fail.
check_property([N | Property]) :-  property(N, A),
                                   yes(A),
                                   check_property(Property).
check_property([]).
/* Правила проверки признаков */
yes(X) :- db_yes(X), !.
yes(X) :- not(no(X)), !,
          check_if(X).
no(X) :- db_no(X), !.
```

Код программы

```
/* Основные правила механизма вывода */
check_if(X) :- write("Оно "), write(X), writeln(" ?"),
                    read(Reply),
                     remember(Reply, X).
remember(да, X)  :- asserta(db_yes(X)).
remember(нет, X) :- asserta(db_no(X)),
                           fail.

/* Целевое правило */
game :-  retractall(db_yes(_)),
             retractall(db_no(_)),
              animal(X),
              write("Задуманное Вами животное - "),
              write(X).
```