Министерство образования и науки Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

А.В. ГАВРИЛОВ

Интеллектуальные системы и основы теории интеллектуального управления

Методические указания к лабораторным работам
для студентов факультета МТФ по направлению подготовки
«Автоматизация технологических процессов и производств»

Новосибирск
2012

Лабораторная работа №1
Знакомство с логическим программированием на языке Prolog в среде SWI-Prolog.

Цель работы

1) Изучить основы синтаксиса языка Пролог.

2) Выработать навыки работы с интерактивной системой SWI- Prolog.

3) Научиться оформлять отношения между данными на языке Пролог на примере выбора станков для обработки детали.

Задание

Разработать программу, в которой в виде предикатов фактов представлены некоторые знания об обработке разных деталей на различных станках. В процессе интерпретации целевого предиката программа должна отвечать на вопросы о том, на каком станке можно обработать заданную деталь и какие детали можно обрабатывать на заданном станке.

Порядок выполнения работы:

1. Запустить программу SWI-Prolog.
2. Создать с использованием встроенного редактора файл с текстом программы для вычисления факториала (см. методические указания ниже).
3. Загрузить файл в интерпретатор с помощью команды *consult*.
4. Задать точку перехвата в отладчике (см. методические указания ниже). Запустить программу вычисления факториала и изучить, как она выполняется, используя отладчик и точку перехвата.
5. Во встроенном редакторе создать файл с программой, включающей в себя предикаты-факты, описывающие то, на каком станке обрабатывается та или иная деталь, тип обработки для деталей, и правила вывода, доказывающие можно ли обработать деталь на заданном станке (см. методические указания).
6. Загрузить программу и поэкспериментировать с ней, задавая ей различные варианты запросов (с неопределенными переменными и со значениями вместо них).
7. Оформить файл с отчетом по работе.

   Содержание отчета:
   - титульный лист;
   - задание;
   - исходный текст;
   - выводы по работе.

Методические указания

Описание программы SWI-Prolog

Исполняемый файл программы SWI-Prolog размещается в **..\bin\plwin.exe**. Для загрузки программы следует вызвать пункт меню “File /

Consult…” и выбрать файл с текстом программы, который может быть создан в текстовом редакторе Notepad. Расширение файла по умолчанию .PL. Файл можно создавать и в оболочке программы SWI-Prolog пунктом меню “File / New…”. По умолчанию вызывается редактор Notepad (Блокнот).

В составе пакета SWI-Prolog имеется более продвинутый редактор (PCE_EMAX), анализирующий синтаксис Пролог-программы, позволяющий устанавливать точки прерываний и т.п. Чтобы включить редактор PCE_EMAX, необходимо в файле pl.ini удалить комментарии в строке

:- set_prolog_flag(editor, pce_emacs).

Сделать это можно любым текстовым редактором, либо вызвав пункт меню “*Settings / User init file*…”.

После запуска программы на экране появится приглашение для ввода запросов:

?-

Запрос (вопрос) вводится после приглашения и обязательно заканчивается точкой, например,

?- 5+4<3.

No

Пролог анализирует запрос и выдает ответ Yes (Да) в случае истинности утверждения и No (Нет) в противном случае или когда ответ не может быть найден.

Теперь рассмотрим работу оболочки SWI-Prolog на примере программы вычисления факториала:

```
fact(0,1).
fact(N,F) :- N1 is N-1,
      fact(N1,F1),
      F is N*F1.
```

Введем в основном окне программы цель: fact(4,F). Программа выдаст ответ: F = 24. В случае ошибок на этапе трансляции или выполнения Пролог в этом же окне выдает диагностические сообщения. Пользоваться таким режимом неудобно.

Лучше воспользоваться отладчиком. Для этого необходимо включить опцию графического отладчика “*Debug / Graphical debugger*”, затем с помощью пункта меню “*Debug / Edit spy points*…” включить точку перехвата на интересующий нас предикат, например, в нашем случае на предикат fact, и нажать кнопку с изображением шпиона.

После этого нужно в окне Пролога задать цель, и появится окно отладки. Панель Bindings отображает текущие значения переменных, Call Stack – состояние стека, т.е. глубину вложенности рекурсий, наконец, нижняя панель – текст программы, где зеленым цветом выделяется выполняемый предикат. Для пошагового выполнения программы нужно нажимать на кнопку со стрелкой вправо.

После этого Пролог начинает выполнять собирать факториал из запомненных в стеке значений 1!, 2! и 3!.

Таким образом, мы можем отслеживать логику работы предикатов и

выявлять ошибки.

Хранят программы на языке Пролог в текстовых файлах, чаще всего имеющих расширение pl, например, example1.pl. Для того чтобы Пролог мог оперировать информацией, содержащейся в файле, его нужно загрузить в интерпретатор. Это можно сделать несколькими способами. При использовании первого варианта в квадратных скобках записывается имя файла (без pl), например,

?- [example1].

В случае удачного завершения этой операции будет выдано сообщение, аналогичное следующему:

% example1 compiled 0.00 sec, 612 bytes

Yes

В противном случае будет выдан список ошибок (ERROR) и/или предупреждений (Warning).

Второй способ состоит в вызове встроенного предиката consult, которому в качестве аргумента передается имя файла (также без расширения), например:

?- consult(example1).

Расширение **pl** часто используется для файлов, содержащих программы на языке программирования Perl, поэтому можно встретить и другие расширения для файлов с программами на Прологе. Для загрузки файла с расширением, отличным от pl, имя файла следует обязательно заключать в апострофы:

?- consult('example2.prolog').

?- ['example2.prolog'].

Обе эти команды добавляют факты и правила из указанного файла в базу данных Пролога. Можно загружать несколько файлов одновременно. В этом случае они перечисляются через запятую, например,

?- [example1, 'example2.prolog'].

Важно помнить, что все запросы должны заканчиваться точкой. Если вы забудете ее поставить, то Пролог выведет символ '|' и будет ожидать дальнейшего ввода. В этом случае надо ввести точку и нажать клавишу Enter.

Программирование на языке Пролог состоит из следующих этапов:

1) объявления некоторых *фактов* об объектах и отношениях между ними,

2) определения некоторых *правил* об объектах и отношениях между ними,

3) формулировки *вопросов* об объектах и отношениях между ними.

1. Факты.

Предположим, надо сформулировать на Прологе факт, что «Деталь обрабатывается на станке». Этот факт включает в себя два объекта «деталь» и «станок», связанные отношением «обрабатывается». В языке Пролог используется стандартная форма записи фактов:

обрабатывается(деталь,станок).

Важно соблюдать при записи фактов следующие правила:

– Имена всех отношений и объектов должны начинаться со строчной буквы. Например, обрабатывается, деталь, станок.

– Сначала записывается имя отношения. Затем в круглых скобках через запятую записываются имена объектов.

– Каждый факт должен заканчиваться точкой.

Определяя с помощью фактов отношения между объектами, необходимо учитывать, в каком порядке перечисляются имена объектов внутри круглых скобок. Этот порядок может быть произвольным, но, выбрав один раз определенный порядок, необходимо следовать ему и дальше. Например, в приведенном ниже примере на первом месте стоит объект, который обрабатывается, а на втором - который обрабатывает. Таким образом, **обрабатывается(деталь,станок)** и **обрабатывается(станок,деталь)** не одно и тоже.

Имена объектов, список которых в каждом факте заключен в круглые скобки, называют аргументами. Имя отношения, которое записывается непосредственно перед скобками, называется именем предиката. Получаем, что «обрабатывается» - это предикат с двумя аргументами.

Совокупность данных в Прологе называется базой данных.

2. Вопросы.

Имея некоторую совокупность фактов, мы можем обращаться к Прологу с вопросами о них. В Прологе вопрос записывается почти так же, как и факт, за исключением того, что перед ним ставится специальный символ « **?-** ». Например:

?- обрабатывается(деталь,станок).

Обращение к Прологу с вопросом инициирует процедуру поиска в базе данных, ранее введенной в систему. Пролог ищет факты, сопоставимые с фактом в вопросе. Два факта сопоставимы (или соответствуют один другому), если их предикаты одинаковы (побуквенное совпадение) и их соответствующие аргументы попарно совпадают. Если Пролог находит факт, сопоставимый с вопросом, то он отвечает **true**. Если в базе данных такого факта не существует – то **false**.

Для того, чтобы обеспечить поиск ответов на более сложные вопросы, например «Что обрабатывается на станке?» в Прологе используются *переменные*.

3. Переменные.

В Прологе можно не только присваивать имена конкретным данным, но и использовать имена, подобные **X** (неизвестное), для обозначение объектов, которые должны быть определены системой. Имена такого типа называются *переменными*. В Прологе используется соглашение, согласно которому каждое имя, начинающееся с прописной буквы, рассматривается как переменная.

При поиске ответа на вопрос Пролог организует просмотр всех фактов в базе данных, чтобы обнаружить объект, который эта переменная могла бы обозначать. Так при вопросе «Обрабатывается ли на станке X?», программа

просматривается все известные ему факты для обнаружения тех вещей, которые могут обрабатываться на станке.

Такая переменная, как **X**, сама по себе не является именем какого-то конкретного объекта, но она может быть использована для обозначения объектов, которым не получается пока что дать имя.

Пролог позволяет использовать переменные и задавать вопросы в виде:

?- обрабатывается(X,станок).

Возможны и другие варианты обозначения переменной. Например:

?- обрабатывается(Что-то, на станке).

Рассмотрим следующую базу данных:

обрабатывается(фланец,станок_1).
обрабатывается(втулка,станок_1).
обрабатывается(колесо,станок_1).

и запрос к Прологу:

?- обрабатывается(X,станок_1).

Смысл этого запроса – «Какие детали обрабатываются на станке ‘станок_1’»

В ответ Пролог предложит вариант:

X=фланец

А затем будет ждать дальнейших приказов. Если этот вариант ответа устраивает, то после нажатия кнопки Enter, он прекратит поиск в базе данных. Если вместо этого нажать «;», то поиск продолжится, и будут выведены следующие варианты значения переменной.

X=фланец;
X=втулка;
X=колесо.

4. Правила.

Предположим, что есть утверждение, что «на данном станке могут обрабатываться все детали, полученные литьем». Один из способов сделать это заключается в записи для каждой детали отдельного факта. Но это не рационально.

Другой способ выразить факт, что на станке могут обрабатываться все детали, при условии, что они получены литьем. Здесь этот факт представлен в форме правила для определения того, какая деталь может обрабатываться на станке, а не прямого перечисления всех деталей.

В Прологе правила используются в том случае, когда необходимо выразить зависимость некоторого факта от группы других фактов.

Правило – это некоторое общее утверждение об объектах и отношениях между ними.

В Прологе правило состоит из заголовка и тела правила. Заголовок и тело соединяются с помощью символа « :- ». Пример, приведенный выше, запишется следующим образом:

обрабатывается(X,станок_1) :- литье(X).

Правила также как и факты, заканчиваются точкой. Заголовком этого правила является предикат **обрабатывается(X,станок)**. Заголовок правила описывает тот факт, для доказательства которого предназначено правило. Тело правила, в данном случае **литье(X)**, описывает цель, которая должна быть последовательно согласованна с базой данных, для того чтобы заголовок правила был истинным (был доказан).

В качестве примера рассмотрим следующую базу данных:

```
литье(втулка).
литье(колесо).
штамповка(диск).
штамповка(фланец).
```

В данном случае используем правило, означающее, что на станке может обрабатываться только деталь, полученная литьем. Это правила можно записать в следующем виде:

```
будет_обрабатываться(X, станок_1) :-    литье(X).
```

В результате получаем:

```
?- будет_обрабатываться(X,станок_1).
X = втулка ;
X = колесо.
```

Контрольные вопросы

1. Как можно загрузить программу из файла?
2. Как запускается программа на Прологе?
3. Как отображаются в программе на Прологе свойства сущностей?
4. Как отображаются в программе на Прологе отношения между сущностями?
5. Как отобразить в программе на Прологе проверку конъюнкции двух предикатов?
6. Что такое унификация в Прологе?
7. Что такое правило в Прологе?

Лабораторная работа №2
Рекурсия и работа со списками в языке Prolog

Цель работы

Изучить принципы построения рекурсии и работы со списками.

Задание

Составить список фрезерных станков с их параметрами, осуществить поиск станка в данном списке и вывести его параметры. Основной параметр — это ширина стола. Для данных станков ширина стола имеет следующие размеры: 200, 250, 320, 400, 500.

В таблице 1 приведены виды станков с параметрами.

Таблица 1. Перечень станков.

Тип станка	*Название*	*Размер стола*
вертикально-фрезерный	6М10	200 мм
вертикально-фрезерный	6Н11	250 мм
горизонтально-фрезерный	6М80Г	200 мм
горизонтально-фрезерный	6Н81Г	250 мм
вертикально-фрезерный	6Н11	320 мм
горизонтально-фрезерный	6М82Г	320 мм
вертикально-фрезерный	6М13	400 мм
горизонтально-фрезерный	6М83Г	400 мм
вертикально-фрезерный	6Н14	500 мм
горизонтально-фрезерный	6Н84Г	500 мм

Порядок выполнения работы

1. Поэкспериментировать с правилом «*принадлежит*», описанном в методических указаниях, и изучить с помощью отладчика как работает рекурсивный перебор элементов списка.
2. Создать во встроенном редакторе программу, в которой представлена информация из выше приведенной таблицы в виде унарных предикатов станок с аргументом – списком из трех параметров (из столбцов таблицы), правил логического вывода с рекурсией для поиска параметров станка по заданному одному параметру. Предикатов «станок» должно быть 10 (по одному на каждую строку таблицы). Программа должна содержать следующие правила:

   a) правило *найти*, с которого запускается программа

   найти(X) :- станок(L), параметры(X,L), phh(L).

   где: L – список параметров.

   b) правило *параметры* с рекурсией для поиска заданного параметра X (написать самостоятельно) и

   c) правило *phh* для вывода на экран списка параметров в виде строки (написать самостоятельно).
3. Оформить отчет по лабораторной работе.

*Содержание отчета***:**

1) титульный лист;
2) задание;
3) исходные тексты;
4) выводы по работе.

Методические указания.

Рекурсия в Прологе

Правило является рекурсивным, если содержит в качестве компоненты само себя. Рекурсия допустима в большинстве языков программирования (например, в Паскале), но там этот механизм не является таким важным, поскольку имеются другие, свойственные процедурным языкам, механизмы – циклы, процедуры и функции.

Рассмотрим преимущества использования рекурсии на примере. Пусть имеются следующие факты о том, какая валюта котируется выше:

```
doroje(dollar, rubl).
doroje(evro, rubl).
doroje(rubl, iena).
doroje(funt, euro).
```

Выполним запрос:

```
? doroje(evro, rubl).
Yes
```

Будет получен утвердительный ответ, поскольку такой факт явно описан в программе. Если же сделать запрос,

```
? doroje(evro, iena).
```

То ответ будет отрицательный, поскольку такой факт отсутствует. Аналогичным будет ответ на вопрос:

```
? doroje(funt, rubl).
```

Избежать таких неправильных ответов здесь можно введением правила, в котором допустимо сравнение между собой не только двух, но и трех объектов:

```
doroje1(X, Y):- doroje(X, Y). /* два объекта */
doroje1(X, Y):- doroje(X, Z), doroje(Z, Y). /* три объекта */
```

Второе правило описывает вариант, когда X>Z, а Z>Y, откуда делается вывод, что X>Y.

Однако цепочка взаимных сравнений может быть длинной. Например, при четырех сравнениях потребуется конструкция:

```
doroje2(X, Y):- doroje(X, M), doroje(M, K), doroje(K, Z), doroje(Z, Y).
```

Описывать такие длинные правила неудобно. Здесь выгоднее применить рекурсию, обратившись к правилу из самого этого правила:

```
doroje1(X, Y):- doroje(X, Y).
doroje1(X, Y):- doroje(X, Z), doroje1(Z, Y).
```

Первое предложение в этой конструкции определяет момент прекращения рекурсивных вызовов.

Второе правило описывает возможности рекурсивных вызовов, когда существуют непроверенные варианты решения. Вообще, любая рекурсивная процедура должна содержать:

1) *Нерекурсивное правило*, применяемое для завершения рекурсии (их может быть несколько, задающие разные условия для завершения рекурсии),

2) *Рекурсивное правило*, первая подцель которого вырабатывает новые значения аргументов, а вторая – рекурсивная подцель – использует эти значения.

Набор правил просматривается сверху вниз. Сначала делается попытка выполнения нерекурсивного правила. Если оно отсутствует, то рекурсивное правило может работать бесконечно.

Использование списков.

Списки – одна из наиболее часто употребляемых структур в ПРОЛОГе. *Список* – это набор объектов одного и того же типа. При записи список заключают в квадратные скобки, а элементы списка разделяют запятыми, например:

[1,2,3]
[1.1,1.2,3.6]
[“вчера”,”сегодня”,”завтра”]

Элементами списка могут быть любые термы ПРОЛОГа, т.е. атомы, числа, переменные и составные термы. Каждый непустой список может быть разделен на *голову* – первый элемент списка и *хвост* – остальные элементы списка. При этом список представляется в виде

[A|B] или [1| [2,3]] или [1| B] или [“вчера”| [“сегодня”,”завтра”]].

Это позволяет всякий список представить в виде бинарного дерева (рис.1). У списка, состоящего только из одного элемента, головой является этот элемент, а хвостом – пустой список. Для использования списка необходимо описать предикат списка.

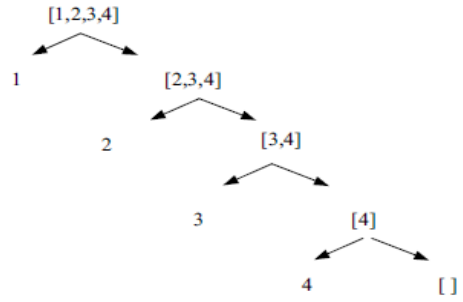

Рис. 1. Бинарное дерево списка.

В следующем примере 1 - голова списка, а [2, 3, 4, 5] - хвост. Пролог покажет это при помощи сопоставления списка чисел с образцом, состоящим из головы и хвоста.

```
?- [1, 2, 3, 4, 5] = [Head | Tail].
Head = 1
Tail = [2, 3, 4, 5]
Yes
```

Здесь *Head* и *Tail* - только имена переменных. Мы могли бы использовать X и Y или какие-нибудь другие имена переменных с тем же успехом. Заметим, что хвост списка всегда является списком. Голова, в свою очередь, есть элемент списка, что верно и для всех других элементов, расположенных до вертикальной черты. Это позволяет получить, скажем, второй элемент списка.

В Prolog’е используется специальный символ для разделения списка на голову и хвост – вертикальная черта |.

Например:

[1, 2, 3] или [1 | [2, 3]] или [1 | [2| [3]]] или [1 | [2 | [3 | [ ]]]]

Вертикальную черту можно использовать не только для отделения головы списка, но и для отделения произвольного числа начальных элементов списка:

[1, 2, 3] или [1, 2 | [3]] или [1, 2, 3 | [ ]]

Пример

Используем анонимные переменные для головы и списка, стоящего после черты, если нам нужен только второй элемент списка:

```
?- [слон, лошадь, осел, собака] = [_, X | _ ].
X = лошадь
Yes
```

Рассмотрим несколько процедур обработки списков. Обратите внимание, что все они используют рекурсию, в которой терминальное (базовое) правило определено для пустого списка.

Пример

Предикат «перестановка» выдает списки, полученные перестановкой элементов своего первого аргумента.

```
перестановка([],[]).
перестановка([H|L],Z):- перестановка(L,Y), место(H,Y,Z).
```

Пример использования:

```
?- перестановка([a,b,c],X).
X = [a, b, c] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [a, c, b] ;
X = [c, a, b] ;
X = [c, b, a] ;
No
```

Предположим, что имеется некоторый список, в котором **X** обозначает его голову, а **Y** — хвост списка. Такой список можно записать, как **[X|Y]**. Этот список может содержать, например, инструмент для обработки деталей:

[фреза, сверло, резец, хон]

Теперь предположим, что мы хотим определить содержится ли некоторый инструмент в указанном списке. В Прологе это можно сделать, определив, совпадает ли данный инструмент с головой списка. Если совпадает, то наш список завершается успехом. Если нет, то мы проверяем, есть ли нужный инструмент в хвосте исходного списка. Это значит, что снова проверяется голова, но уже очередного хвоста списка. Если мы доходим до конца списка, который будет пустым списком, то наш поиск завершается неудачей: указанного инструмента в исходном списке нет.

Для того, чтобы записать все это на Прологе, сначала надо

установить, что между объектом и списком, в который этот объект может входить, существует отношение. Для записи этого отношения будем использовать предикат **принадлежит**: целевое утверждение **принадлежит(X,Y)** является истинным («выполняется»), если терм, связанный с **X,** является элементом списка, связанного с **Y**. Имеются два условия, которые надо проверить для определения истинности предиката. Первое условие говорит, что **X** будет элементом списка **Y**, если **X** совпадает с головой списка **Y**. На Прологе этот факт записывается следующим образом:

```
принадлежит(X,[X|_]).
```

Эта запись констатирует, что **X** является элементом списка, который имеет **X** в качестве головы. В данном случае анонимная переменная «_» для обозначения хвоста списка.

Второе правило говорит о том, что **X** принадлежит списку при условии, что он входит в хвост этого списка, обозначаемый через **Y**. Для этого используем тот же самый предикат **принадлежит** для того, чтобы определить, принадлежит ли **X** хвосту списка. В этом и состоит суть рекурсии. На Прологе это выглядит так:

```
принадлежит(X,[_|Y]) :- принадлежит(X,Y).
```

Два этих правила в совокупности определяют предикат для отношения принадлежности и указывают Прологу, каким образом просматривать список от начала до конца при поиске некоторого элемента в списке. Наиболее важный момент, о котором следует помнить, встретившись с рекурсивно определенным предикатом, заключается в том, что прежде всего надо найти граничные условия и способ использования рекурсии.

Для предиката **принадлежит** в действительности имеются два типа граничных условий. Либо объект, который мы ищем, содержится в списке, либо нет. Первое граничное условие для предиката **принадлежит** распознается первым утверждением, которое приводит к прекращению поиска в списке, если первый аргумент предиката **принадлежит** совпадает с головой списка. Второе граничное условие встречается, когда второй аргумент предиката **принадлежит** является пустым списком.

Это все демонстрирует следующий пример на Прологе:

```
принадлежит(X,[X|_]).
принадлежит(X,[_|Y]) :- принадлежит(X,Y).
?- принадлежит(фреза,[фреза, сверло, резец, хон]).
true
?- принадлежит(протяжка,[фреза, сверло, резец, хон]).
false.
```

Контрольные вопросы

1. Как представляется список в Прологе? Примеры.
2. Что такое рекурсия?
3. Для чего используется представление списка в виде головы и хвоста?
4. Какое минимальное количество предложений Пролога необходимо для программирования рекурсии?

5. В чем преимущество использования рекурсии?

Лабораторная работа №3
Разработка экспертных систем

Цель работы

1) Ознакомиться с описанием ESWin и с языком представления знаний

2) Ознакомиться с пользовательским интерфейсом программы ESWin.

3) Разработать и отладить базу знаний о «выборе подачи фрезерного станка для чернового фрезерования».

Задание

Разработать базу знаний в виде фреймов и правил в среде ESWin для решения задачи выбора подачи фрезерного станка. Использовать исходные данные, приведенные в таблице 2. При этом вариант, задаваемый преподавателем, есть какой-либо подстолбец подач, указанных в таблице, например, выделенный в таблице. Для выделенного столбца должно быть сформировано как минимум 8 правил (количество ячеек в подстолбце). При этом условия правил должны обеспечивать выбор ячейки (рекомендуемой подачи), используя все факторы, указанные в таблице (названия столбцов и строк и тип фрезы).

Таблица 2. Выбор подачи.

Мощность станка или фрезерной головки в Квт	Жесткость системы «заготовка-приспособление»	Фрезы			
		Торцовые и дисковые		Цилиндрические	
		Подача на один зуб в мм при обработке			
		конструкционной стали	чугуна и медных сплавов	конструкционной стали	чугуна и медных сплавов
Фрезы с крупным зубом и фрезы со вставными ножами					
>10	Повышенная	0,20-0,30	0,40-0,60	0,40-0,60	0,60-0,80
	Средняя	0,15-0,25	0,30-0,50	0,30-0,40	0,40-0,60
	Пониженная	0,10-0,15	0,20-0,30	0,20-0,30	0,25-0,40
5-10	Повышенная	0,12-0,20	0,30-0,50	0,25-0,40	0,30-0,50
	Средняя	0,08-0,15	0,20-0,40	0,12-0,20	0,20-0,30
	Пониженная	0,06-0,10	0,15-0,25	0,10-0,15	0,12-0,20
<5	Средняя	0,06-0,07	0,15-0,30	0,08-0,12	0,10-0,18
	Пониженная	0,04-0,06	0,10-0,20	0,06-0,10	0,08-0,15
Фрезы с мелким зубом					
5-10	Повышенная	0,08-012	0,20-0,35	0,10-0,15	0,12-0,20
	Средняя	0,06-0,10	0,15-0,30	0,06-0,10	0,10-0,15
	Пониженная	0,04-0,08	0,10-0,20	0,06-0,08	0,08-0,12
<5	Средняя	0,04-0,06	0,12-0,20	0,05-0,08	0,06-0,12
	Пониженная	0,03-0,05	0,08-0,15	0,03-0,06	0,05-0,10

Порядок выполнения работы

1. Изучить пользовательский интерфейс и основные возможности программы ESWin на примере баз знаний, указанных преподавателем.
2. На основе выше приведенной таблицы создать и проверить работоспособность экспертной системы по выбору подачи для одного из вариантов параметров, заданных преподавателем.
3. Оформить отчет.

   Содержание отчета:

   - титульный лист;
   - задание;
   - краткое описание структуры базы знаний
   - исходный текст базы знаний;
   - выводы по работе

Методические указания.

ESWin — программная оболочка-интерпретатор для работы с продукционно-фреймовыми экспертными системами. Описываемая программная оболочка предназначена для решения задач обратного логического вывода на основе фреймов и правил-продукций.

Знания в базе знаний хранятся в виде фреймов и правил-продукций.

До начала работы с экспертной оболочкой база знаний находится в текстовом файле. В файле с расширением *.klb (KnowLedge Base) хранятся фреймы и правила-продукции (база знаний). При начале работы с программной оболочкой наличие данного файла обязательно. Этот файл создается пользователем с помощью специального редактора EdKB или вручную с помощью какого-либо стандартного текстового редактора (например, "Блокнот" или WordPad). Кроме файла *.klb в базу знаний может входить файл *.lvd, содержащий описания лингвистических переменных, если они используются в базе знаний. В файле с расширением *.dtb (DaTa Base) хранятся факты, полученные в процессе логического вывода и диалога с пользователем. При начале работы с программной оболочкой наличие данного файла необязательно. Имена файлов *.klb, *.dtb и *.lvd для одной базы знаний совпадают.

При работе с программной оболочкой (после загрузки в оперативную память базы знаний) фреймы и правила-продукции, находившиеся в файле с расширением *.klb, остаются неизменными. Факты, находившиеся в файле с расширением *.dtb, могут изменяться в процессе логического вывода (появляться, удаляться или менять свое значение в результате срабатывания правил-продукций или диалога с пользователем).

Фреймом называется поименованная структура данных для описания стереотипной ситуации (события, объекта, понятия) состоящая из характеристик этой ситуации (события, объекта, понятия), называемых слотами и их значений. Слот может не иметь значения или иметь одно значение, вид и интерпретация которого определяется типом слота. В пакете

ESWin 2.1 используется ограниченная концепция фрейма, при которой не поддерживается присоединение к фрейму процедур и правил-продукций.

В пакете ESWin используются фреймы трех типов: фрейм-класс, фрейм-экземпляр и фрейм-шаблон. В общем виде фрейм-класс выглядит следующим образом:

FRAME Имя фрейма
PARENT: Имя фрейма-родителя
OWNER: Имя фрейма (объекта), частью которого является данный фрейм
Имя слота 1 [Вопрос] {Имя файла}(Тип): (Знач. 1 слота 1; Знач. 2 слота 1 …)
Имя слота 2 [Вопрос] {Имя файла}: (Знач. 1 слота 2; Знач. 2 слота 2, …)
…
Имя слота n [Вопрос] {Имя файла}: (Знач. 1 слота n; Знач. 2 слота n, …)
ENDFR

Значением слота может быть число или строка, содержащая любые символы кроме точки с запятой. Вопрос - это строка, используемая в диалоге с пользователем при запросе значения слота от него. Тип по умолчанию символьный, но может быть и «численный» или «лп» (лингвистическая переменная). Имя файла – это имя файла (типа .txt, .gif или .htm), используемого в качестве подсказки (комментария) при вопросе пользователю. Вопрос и имя файла могут отсутствовать в описании слота. Если слот символьный, список значений слота используется для формирования меню при вопросе.

При конкретизации фрейма-класса происходит присваивание значений слотам и таким образом формируется фрейм-экземпляр. Фреймы-экземпляры составляют базу данных. База данных предназначена для временного хранения фактов или гипотез, являющихся промежуточными решениями или результатом общения экспертной системы с пользователем.

Фрейм с зарезервированным именем **Цель** служит для описания слотов, являющимися задачами, для решения которых создается экспертная система, или другими словами целевыми слотами, значения которых определяет система.

Правила-продукции имеют вид:

RULE Номер правила
  Отношение(Фрейм.Слот; Значение)
….
DO
  Отношение(Фрейм.Слот; Значение)
ENDR

Здесь до слова DO перечисляются условия (их конъюнкция), а после DO – заключения правила. В качестве отношений в условиях можно использовать арифметические отношения > (GT),< (LT),= (EQ),<> (NE) и отношение IN (часть-целое). В последнем случае значением слота является имя фрейма, частью которого является данный фрейм. В качестве отношений в заключении могут использоваться следующие обозначения:

= (EQ) - создание факта - слота во фрейме-экземпляре);

IN - включение во фрейм-владелец (создание связи - слота OWNER во фрейме-экземпляре);
DL - удаление слота во фрейме-экземпляре;
EX - запуск внешней программы;
FR - вывод фрейма-экземпляра;
GO - запуск правила;
MS - вывод на экран сообщения;
GR - вывод на экран графического файла (форматов *.gif, *.avi или *.htm).

Решение задачи с помощью оболочки ESWin и загруженной в нее базы знаний начинается с выбора цели логического вывода или задачи. В качестве цели логического вывода используется один из целевых слотов, содержащихся во фрейме-классе со специальным именем «Цель».

Для решения задачи в системе ESWin используется обратный логический вывод. Он начинается с поиска правила, в заключении которого присутствует выбранный целевой слот.

После нахождения правила начинается его интерпретация (перебор и проверка условий). При проверке условия ищется (определяется) значение указанного в условии слота.

Определение значения слота производится в следующем порядке:

1) ищется факт со значением этого слота в базе фактов;
2) ищется значение слота в одноименном фрейме-классе (если это значение в нем единственное);
3) ищется правило, задающее при срабатывании значение данного слота, и запускается его интерпретация;
4) ищется одноименная структура SOURCE для формирования SQL-запроса к внешней базе данных;
5) ищется слот во фрейме-классе с описанием всей необходимой информации для запроса значения слота у пользователя (в том числе во фреймах-родителях), и задается вопрос пользователю.

К каждому из этих шагов оболочка переходит в случае неудачи предыдущего шага.

Контрольные вопросы

1. Что такое экспертная система?
2. Что такое правило-продукция?
3. Что такое лингвистическая переменная?
4. Что такое фрейм? Что такое слот?
5. Какого назначение и в чем отличия фреймов-классов и фреймов-экземпляров?
6. Как обеспечивается наследование свойств в языке оболочки ESWin?
7. В чем заключается упрощение фреймов в оболочке ESWin?
8. Чем отличается обратный логический вывод от прямого?
9. Какого назначение Цели в оболочке ESWin?
10. Как влияет расположение правил в базе знаний в среде оболочки ESWin на решение задачи экспертной системой?
11. Что может быть в заключении правил в языке оболочки ESWin?

12.Какие два типа иерархических сетей фреймов поддерживает ESWin?

Лабораторная работа №4
Диалог на естественном языке на основе языка шаблонов AIML

Цель работы

Изучение общих принципов построения и функционирования подсистемы анализа запросов на естественном языке для системы накопления знаний.

Задание

Составить описание диалога на AIML, имитирующее диалог с промышленным роботом. При этом в диалоге система AIML должна воспринимать команды «взять деталь из входного бункера», «загрузить деталь в станок», «взять деталь из станка», «положить деталь в выходной бункер». Система должна также запоминать стадию обработки детали (где она находится) и отвечать на соответствующий вопрос пользователя.

Порядок выполнения работы

1. Изучить основные возможности и теги системы AIML.
2. Написать простую программу на AIML для ведения простого диалога с роботом (без переменных и проверки условий) со случайным выбором ответов.
3. Ввести в нее использование * для вставки в ответ слова из реплики пользователя.
4. Ввести в программу одну или несколько переменных в соответствии с поставленной задачей и их проверку при выборе ответа на реплику пользователя.

*Содержание отчета***:**

- титульный лист;
- задание;
- диалог на языке шаблонов AIML;
- выводы по работе.

Методические указания

Для работы AIML-бота необходимо 2 компонента: файл AIML и программа (интерпретатор), которая обрабатывает входящие реплики собеседника и формирует ответную реплику в соответствии с файлом AIML.

Файл AIML - содержит набор категорий (category). Категория открывается тегом <category> и закрывается тегом </category>. Каждому тегу открытия должен соответствовать тег закрытия. Иначе структура AIML будет нарушена и программа отвечать не сможет.

Категория включает паттерны (pattern) и шаблоны (template). Паттерн открывается тегом <pattern> и закрывается тегом </pattern>. Шаблон открывается тегом <template> и закрывается тегом </template>. Категория может включать только один паттерн и один шаблон. Паттерн и шаблон

должны находиться внутри категории.

Интерпретатор AIML работает очень просто. Он берет входящую фразу собеседника и сравнивает ее со всеми имеющимися паттернами в файле AIML. В случае совпадения входящей фразы с паттерном в какой-либо категории интерпретатор в качестве ответной реплики подставляет шаблон из этой категории.

Паттерн – это категория, заключающая в себя ключевые слова. Пишется большими буквами. Для определения произвольного слова или группы слов используется знак *. Примеры паттернов:

1) <pattern>ПРИВЕТ</pattern>

2) <pattern>ПРИВЕТ *</pattern>

3) <pattern>* ПРИВЕТ *</pattern>

4) <pattern>* ПРИВЕТ</pattern>

На первый взгляд может показаться, что данные паттерны будут работать одинаково и реагировать на любую реплику со словом "Привет". Но в AIML данные паттерны имеют существенные различия.

Паттерн 1 подойдет только для входящей реплики содержащей единственное слово "Привет" - и ни для какой другой.

Паттерн 2 подойдет только для входящей реплики, начинающейся со слова "Привет" и содержащей еще какие-нибудь слова после слова "Привет", например, "Привет, Робот!".

Паттерн 3 подойдет только для входящей реплики, начинающейся с одного или нескольких слов, за которыми следует слово "Привет" и содержащей еще какие-нибудь слова после слова "Привет", например, "Эй ты привет робот!".

Паттерн 4 подойдет только для входящей реплики, начинающейся с одного или нескольких слов, за которыми следует слово "Привет", например, "Робот, привет!".

Шаблон – это ответные реплики программы на ваши слова. Сюда может входить одна реплика, например:

```
<template>Добрый день!</template>
```

Или несколько реплик. В этом случае необходимо использовать дополнительный тег <random>. Пример:

```
<template>
    <random>
      <li>Добрый день! </li>
      <li>Здравствуйте </li>
      <li>Приветик ...  </li>
    </random>
</template>
```

В этом случае в качестве ответной реплики будет произвольно выбрана одна из реплик, находящаяся между тегами .

Случайно выбранный ответ можно комбинировать с обычной репликой. Например:

```
<template>
```

```
        <random>
          <li>Добрый день! </li>
          <li>Здравствуйте </li>
           <li>Приветик ...  </li>
         </random>
           Как дела?
    </template>
```

В этом случае программа ответит "Добрый день! Как дела?" или "Здравствуйте Как дела?" или "Приветик ... Как дела?"

В шаблоне также возможно создать ссылку на другой паттерн. Делается это с помощью тега <srai>. Рассмотрим категорию:

```
    <category>
      <pattern>ПРИВЕТ</pattern>
      <template>
        <random>
          <li>добрый день</li>
          <li>Здравствуйте!</li>
        </random>
      </template>
    </category>
```

Эта категория позволяет боту отвечать либо "добрый день" либо "Здравствуйте!" в ответ на произнесенное собеседником "Привет". Для того, чтобы бот отвечал аналогично на более сложное приветствие, например: "Привет, Робот!" можно добавить еще одну категорию:

```
    <category>
      <pattern>ПРИВЕТ *</pattern>
      <template>
        <random>
          <li>добрый день</li>
          <li>Здравствуйте!</li>
        </random>
      </template>
    </category>
```

А можно поступить проще и просто добавить ссылку на существующий паттерн "ПРИВЕТ" с помощью тега <srai/>:

```
    <category>
      <pattern>ПРИВЕТ *</pattern>
      <template><srai>ПРИВЕТ</srai></template>
    </category>
```

Think – позволяет задавать значения переменным, сохраняя их в памяти.

Как пример можно рассмотреть:

```
<category>
  <pattern>ПОДНИМИ * СО СТОЛА</pattern>
  <template>Поднимаю, <star/>.
    <think>
              <set name="object"><star/></set>
    </think>
  </template>
</category>
```

Здесь тег <star/> заменяется на слово, соответствующее символу * в образце <pattern>, а object – имя переменной для запоминания этого слова с целью использования в дальнейшем. Для сохранения переменной используется тег <set name=”object”>.

В итоге получаем следующее (рисунок 2):

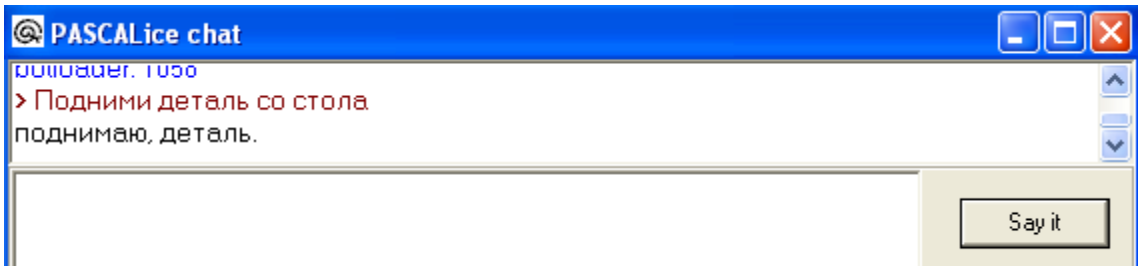

Рис. 2. Пример выполнения тега think

Проверять значения переменных можно с помощью условий. Условие (Condition) имеет три формы: "multi condition" (несколько условий) ,"list-condition"(список условий) , и "single name list-condition" (единое имя списка условий). Разница между этими формами в том, что в тэге "multi condition" будут оцениваться все заданные условия, в остальных же тегах — оценка условий может завершиться после первого использования <li>, при котором условие является выполненным.

Рассмотрим пример::

```
<category>
  <pattern>ЕСТЬ ЛИ ДЕТАЛИ В ЗАХВАТЕ</pattern>
    <template>
      <condition>
        <li name="object" value="">Нет. Я пуст</li>
        <li>Я поднял <get name="object"/></li>
      </condition>
    </template>
</category>
```

Здесь первая строка после <condition> кодирует условие «переменная *object* имеет значение *пусто*», вторая строка выполняется если переменная *object* имеет какое-то значение. В ответ на реплику пользователя робот сообщит о наличии в его захвате детали или об отсутствии детали в захвате. Название переменной *object* вставляется в ответ пользователю с помощью тега <get name=”object”>. Имя переменной, проверяемой в условии, можно

задавать и непосредственно в теге <condition>, например, <condition name=”object”>.

Теперь посмотрим, как это будет выполнено (рисунок 3):

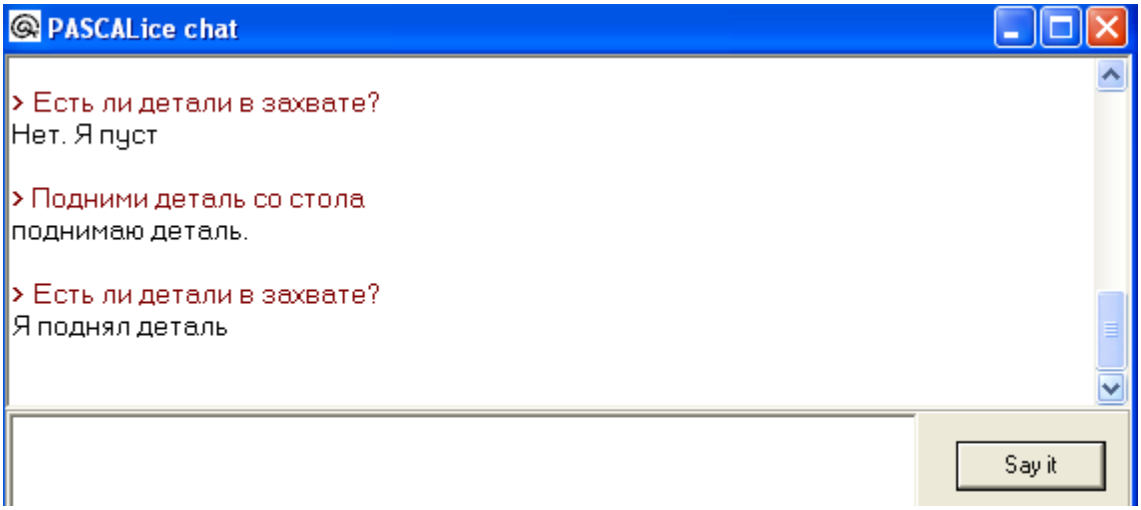

Рис. 3. Пример выполнения тега Condition.

Контрольные вопросы

1. В чем трудности моделирования понимания естественного языка?
2. Особенности естественного языка как метода представления знаний.
3. Перечислите основные методы анализа естественного языка
4. В чем суть метода шаблонов?
5. Основные теги языка AIML?
6. Как можно запоминать информацию в ходе диалога, используя язык AIML?
7. Назначение тега <think>?
8. Назначение тега <condition>?

Литература

К лабораторным работам № 1 и № 2.

1. И. Братко. Программирование на языке ПРОЛОГ для искусственного интеллекта. – М.:Мир, 1990.
2. Д.Марселлус. Программирование экспертных систем на Турбо-Прологе. - М.: Финансы и статистика, 1994.
3. Логический подход к искусственному интеллекту. - М.: Мир, 1990.
4. Ин Ц., Соломон Д. Использование Турбо-Пролога. – М.: Мир, 1993. – 608 С.
5. Доорс Дж., Рейблейн А.Р., Вадера С. Пролог - язык программирования будущего. – М.: ФиС, 1990. – 144 С.
6. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. – М.: Мир, 1990. – 235 С.
7. Клоксин У., Меллиш Д. Программирование на языке Пролог. – М.: Мир, 1987. –336 С.
8. Стобо Дж. Язык программирования Пролог. – М.: Мир, 1993. – 368 С.

9. Ю.В.Новицкая. Основы логического и функционального программирования. - Уч. пособие, Новосибирск: Изд-во НГТУ, 2004.
10. А.В.Гаврилов, Ю.В.Новицкая. Основы программирования на Турбо-Прологе. – Уч. пособие, Новосибирск: Изд-во НГТУ, 1993.
11. Сырецкий Г.А. Информатика. Часть III. Основы логического программирования на PDC Prolog. —Новосибирск: Изд-во НГТУ, 1994.

К лабораторной работе № 3.

1. Экспертная оболочка ESWin. http://www.insycom.ru
2. А.В.Гаврилов. Системы искусственного интеллекта. Ч. 1, Уч.-метод. указания, НГТУ, 2001.
3. А.В.Гаврилов. Системы искусственного интеллекта. Уч.-метод. указания для заочников, НГТУ, 2004.
4. А.В. Гаврилов, Ю.В. Новицкая. Разработка экспертных систем. Уч.-метод. указания к лабораторным работам, НГТУ, 2000.
5. Джексон П. Введение в экспертные системы. – М., СпБ., Киев: "Вильямс", 2001.
6. А.В.Гаврилов. Гибридные интеллектуальные системы. – Новосибирск: НГТУ, 2003.
7. Р.Левин, Д.Дранг, Б.Эдельсон. Практическое введение в технологию искусственного интеллекта и экспертных систем с иллюстрациями на Бейсике. - М.: Финансы и статистика, 1990.
8. Э.В. Попов, И.Б. Фоминых, Е.Б. Кисель, М.Д.Шапот. Статические и динамические экспертные системы. – М.: Финансы и статистика, 1996.
9. Т.А. Гаврилова, В.Ф. Хорошевский. Базы знаний интеллектуальных систем. – СпБ, Питер,2000.
10. Обработка знаний. – М: Мир, 1990.
11. Э.В. Попов. Экспертные системы. – М.: Наука, 1987.
12. Построение экспертных систем. Под ред. Ф. Хейес-Рота, Д. Уотермена, Д. Лената. – М.: Мир, 1987.
13. Представление и использование знаний. – М.: Мир, 1989.
14. Д. Уотерман. Руководство по экспертным системам. – М.: Мир, 1989.
15. К. Таунсенд, Д. Фохт. Проектирование и программная реализация экспертных систем на персональных ЭВМ. – М.: Финансы и статистика, 1990.

К лабораторной работе № 4.

1. Э.В.Попов. Общение с ЭВМ на естественном языке. – М.: УРСС, 2004.
2. А.В.Гаврилов. Системы искусственного интеллекта. Уч.-метод. указания для заочников, НГТУ, 2004.
3. И.С.Евдокимова. Естественно-языковые системы: курс лекций. – Улан-Удэ: Изд-во ВСГТУ, 2006.
4. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика : учеб. пособие / Большакова Е.И., Клышинский Э.С., Ландэ Д.В., Носков А.А., Пескова О.В., Ягунова Е.В. - М.: МИЭМ, 2011. - 272 с.