ЛАБОРАТОРНАЯ РАБОТА №6

ПОСТРОЕНИЕ МАШИНЫ ВЫВОДА В ПРОДУКЦИОННОЙ ЭКСПЕРТНОЙ СИСТЕМЕ

Цель работы.

Изучение механизма обратного логического вывода в продукционных ЭС и его реализация средствами языка Пролог.

Краткие теоретические сведения

Механизм вывода предназначен для построения заключений на основе знаний, содержащихся в базе знаний. Действия механизма вывода аналогичны рассуждениям человека-эксперта, который оценивает проблему и предлагает возможные решения. По запросу пользователя механизм вывода выполняет поиск решений в базе знаний, а также оценивает достоверность предлагаемых решений.

В данной работе рассматривается построение механизма обратного логического вывода (обратной цепочки рассуждений). Предлагаемая реализация обратного логического вывода близка к механизму вывода, используемого в оболочке для построения продукционных ЭС GURU.

В начале процесса консультации (т.е. рассмотрения набора правил) для механизма вывода указывается целевая переменная (цель). Под целью понимается некоторая переменная, значение которой механизм вывода должен определить в результате консультации. Механизм вывода анализирует правила базы знаний до тех пор, пока не установит значение целевой переменной или пока не выяснит, что найти такое значение невозможно. В первом случае ЭС сообщает о найденном решении, во втором - о невозможности нахождения решения.

При использовании обратного вывода ЭС для нахождения значения целевой переменной просматривает заключения правил, имеющихся в базе знаний. ЭС находит правило, в заключении которого может быть определено значение цели (т.е. правило, в заключении которого выполняется присваивание целевой переменной некоторого значения). Если таких правил несколько, то выбор конкретного правила зависит от реализации конкретной ЭС; обычно для рассмотрения выбирается первое правило из тех, в заключениях которых может быть определена целевая переменная. Анализируется посылка выбранного правила. Если она верна (т.е. выполняются указанные в ней условия), то правило включается (срабатывает): выполняются действия, указанные в его заключении. В результате целевая переменная получает некоторое значение. Если посылка неверна, правило не включается, и происходит обращение к следующему правилу, в заключении которого может быть определена целевая переменная. Если такого правила нет, то консультация завершается с выдачей сообщения о невозможности нахождения решения.

Если посылка выбранного правила содержит одну или несколько переменных, значения которых неизвестны системе (еще не найдены), то такие переменные поочередно (обычно - в порядке их расположения в посылке, слева направо) рассматриваются в качестве временных целей. ЭС пытается определить значение каждой из временных целей так же, как это делается для основной целевой переменной: отыскиваются правила, в которых временная цель может получить значение, проверяются посылки этих правил и т.д. Этот процесс может повторяться многократно, так как для определения временной цели в

одном из правил может потребоваться найти значения других переменных, содержащихся в посылке этого правила, и т.д.
После определения значений неизвестных переменных ЭС выполняет проверку посылки правила, для которого эти переменные потребовались (т.е. правила, в заключении которого определяется целевая переменная). В зависимости от реализации механизма вывода, проверка посылки правила может производиться один раз (после определения всех неизвестных переменных) или многократно (после определения каждой из неизвестных переменных). В последнем случае поиск значений всех неизвестных переменных может и не потребоваться. Например, если условия в посылке связаны логической операцией “и”, и одно из условий не выполняется, то поиск неизвестных переменных для проверки других условий не требуется: рассмотрение данного правила прекращается (так как уже определено, что оно не должно включаться), и для рассмотрения выбирается следующее правило.

Реализация обратного логического вывода

Примечание. При составлении программы на языке Пролог, реализующей логический вывод, необходимо учитывать, что все переменные, указанные в правилах базы знаний – это не то же самое, что переменные программы на Прологе. Для программы все переменные, используемые в правилах базы знаний, являются *не переменными, а данными для обработки*.
Не следует также путать целевую переменную продукционной ЭС (которая является для программы просто элементом данных) и целевой предикат программы на Прологе, указываемый в разделе goal.

Для указания целевой переменной и выполнения ее поиска можно воспользоваться следующим предикатом.

```
vyvod:-
retractall (znach(_,_)),
goal_var (G),
obr_vyvod (G, Result),
nl, write ("Решение: ", Result),
readchar(_), !.
vyvod:-
nl, write ("Цель не найдена"), readchar (_).
```

Здесь стандартный предикат retractall удаляет из памяти все предикаты базы данных znach, созданные в ходе предыдущей консультации (назначение предикатов znach будет рассмотрено ниже). Если таких предикатов нет, то предикат retractall завершается успешно, не выполняя никаких действий (см. лабораторную работу 5). Затем определяется имя целевой переменной ЭС из предиката базы данных goal_var. Это имя присваивается переменной программы G. Предикат obr_vyvod непосредственно реализует процесс обратного логического вывода. В результате его доказательства определяется значение целевой переменной (переменная Result), и оно выводится на экран. Если вывести значение целевой переменной не удается, то доказательство предиката obr_vyvod заканчивается неудачей. В этом случае заканчивается неудачей доказательство первого клоза предиката vyvod, и доказывается его второй клоз: на экран выводится сообщения о невозможности определения цели.
Предикат obr_vyvod можно реализовать следующим образом.

```
obr_vyvod (G, Result):- znach (G, Result), !.
obr_vyvod (G, Result):-
var (G, Zapros), Zapros<>"",
write (Zapros), readln (Result),
assert (znach(G, Result)), !.
obr_vyvod(G, Result):-
rule (N, Usl, assign(G, Result)),
proverka (Usl),
assert (znach(G, Result)), !.
```

Таким образом, для реализации предиката obr_vyvod использованы три клоза. В первом из них проверяется, нет ли в базе данных факта (предиката znach), который указывал бы значение целевой переменной.

Во втором клозе предпринимается попытка найти в базе данных описание целевой переменной, т.е. предикат var, первым аргументом которого является переменная с именем, хранящимся в программной переменной G. Затем проверяется, предусмотрен ли запрос этой переменной (имеется ли в описании
переменной текст запроса). Если в базе данных удается найти соответствующий предикат var с текстом запроса, то этот текст выводится на экран (предикатом write), и у пользователя ЭС запрашивается значение переменной, которая в
данный момент является целевой. Это значение сохраняется в базе данных (т.е. в памяти): для этого стандартный предикат assert создает предикат базы данных znach, аргументами которого является имя переменной и ее значение,
введенное пользователем. Это требуется, чтобы не повторять запрос переменной, если затем в процессе консультации снова потребуется ее значение.

Примечание. Так как znach - предикат базы данных, его необходимо объявить в разделе global facts.

Важно отметить, что для основной целевой переменной (указанной в предикате базы знаний goal_var) обращение к первым двум клозам *всегда* заканчивается неудачей, так как в начале консультации значение целевой
переменной неизвестно (еще не определено), а в базе знаний (конечно, если она правильно составлена) *не может быть* предусмотрен запрос основной целевой переменной у пользователя.

В третьем клозе отыскивается правило (предикат базы знаний rule), в котором определяется значение цели. Для такого правила проверяется условная часть с помощью предиката proverka. Если условная часть оказывается верной (предикат proverka доказывается успешно), то выполняется действие, указанное в заключении правила: переменной Result присваивается значение, заданное в функторе assign. Имя найденной целевой переменной и ее значение заносятся в базу данных в форме предиката znach.

Таким образом, найденное значение переменной становится фактом базы данных и может использоваться в последующих выводах (если эта переменная не была основной целевой переменной). Предикат отсечения (!) требуется для того, чтобы при выполнении одного из правил остальные правила, в которых определяется та же цель, не сохранялись в памяти как неиспользованные альтернативы.

Если условная часть оказывается неверной (доказательство предиката proverka завершается неудачей), то происходит возврат к следующему правилу (т.е. предикату rule), в котором определяется та же целевая переменная. Предикат proverka, выполняющий проверку условной части правила, можно реализовать следующим образом.

```
proverka([]):- !.
proverka(Usl):- Usl=[H|T],
prov1 (H),
proverka (T).
```

Условная часть (посылка) правила представляет собой *список* условий, каждое из которых реализовано в виде функтора. Поэтому предикат proverka выполняет поочередную проверку этих условий, начиная с первого. Принцип
работы этого предиката такой же, как и у рассмотренных ранее предикатов для обработки списков (см. лабораторную работу 2). Непосредственно проверка каждого условия реализуется предикатом prov1, который должен завершаться успешно при выполнении условия, а при невыполнении - завершаться
неудачей. Если какое-либо из условий в посылке оказывается ложным (т.е. доказательство предиката prov1 для него завершается неудачей), то проверка остальных условий, т.е. доказательство предиката proverka(T), не выполняется. Предикат proverka в этом случае завершается неудачей.
Предикат prov1 предназначен для проверки каждого из условий, входящих в условную часть правила. Как отмечено выше, условная часть правила реализована в виде списка и состоит из функторов, которыми обозначены различные операции сравнения (см. лабораторную работу 6). В качестве примера рассмотрим реализацию предиката prov1 для сравнения на равенство.

```
prov1(Uslovie):- Uslovie=eq(Perem, Vel), !,
obr_vyvod(Perem, X),
X=Vel.
```

Здесь переменной Uslovie присвоен один из функторов, составляющих посылку правила (этот функтор передается из предиката proverka). Сначала проверяется, обозначает ли проверяемый функтор операцию проверки на равенство (т.е. является ли он функтором eq). Если оказывается, что проверяется другой функтор (например, ge или lt), то сравнение Uslovie=eq(Perem, Vel) заканчивается неудачей, и происходит возврат к другим клозам предиката prov1, реализующим другие операции сравнения.
Если оказывается, что выполняется сравнение на равенство, то переменная Perem связывается с именем переменной, указанной в функторе eq, а переменная Vel - с величиной, с которой требуется выполнить сравнение.
Переменная, указанная в функторе, становится временной целью; для нее выполняется процесс поиска значения (обратного вывода) с помощью предиката obr_vyvod. Затем значение, найденное в результате обратного вывода (X), сравнивается с указанным в функторе (Vel); если они равны, это означает,
что проверяемое условие (из посылки правила) верно. Если эти значения не равны (т.е. проверяемое условие ложно), то предикат X=Vel заканчивается неудачей; вместе с ним заканчивается неудачей и предикат prov1 (т.е. проверка данного условия). Другие клозы предиката prov1 (реализующие другие операции сравнения) при этом не рассматриваются, так как они исключены из рассмотрения предикатом отсечения (!). Предикат prov1 для других операций сравнения реализуется аналогично. Следует обратить внимание, что операции сравнения "больше", "меньше", "больше или равно" и "меньше или равно" выполняются для числовых данных, а в предлагаемой реализации ЭС все данные хранятся в строковой форме. Поэтому при проверке соответствующих операций сравнения требуется преобразование типов. Ниже приводится пример клоза предиката
prov1

для проверки условия "больше".

```
prov1(Uslovie):-Uslovie=gt(Perem,Vel),!,
obr_vyvod(Perem, X),
str_real(Vel, Nvel),
str_real(X, Nx),
Nx > Nvel.
```

Для преобразования типов здесь использован стандартный предикат str_real, преобразующий свой первый аргумент (строковую переменную) в вещественное число (второй аргумент).

Пример работы механизма вывода

Рассмотрим процесс консультации с ЭС, используемой для диагностики неисправностей некоторого устройства. База знаний для этой ЭС приведена в лабораторной работе 5. В предикате vyvod определяется имя целевой переменной ЭС. Для этого в базе знаний, загруженной с диска, отыскивается предикат goal_var("D"). Переменная программы G принимает значение "D". Затем начинается доказательство предиката obr_vyvod("D", Result); переменная Result пока не имеет значения (является неизвестной).
Попытка доказательства первых двух клозов предиката obr_vyvod заканчивается неудачей, так как в базе данных еще нет предикатов znach, и нет предиката var с запросом переменной "D". При доказательстве третьего клоза предиката obr_vyvod находится правило:

```
rule(4,[eq("Neispr", "тип 1")],assign("D", "прекратить работу"))
```

Предикат proverka выполняет проверку условной части этого правила:
[eq("Neispr","тип 1")].

В предикате prov1 переменная Neispr становится временной целью: выполняется попытка доказать предикат obr_vyvod("Neispr", X), т.е. определить значение переменной Neispr. Попытка доказать первых два клоза предиката obr_vyvod заканчивается неудачей (по тем же причинам, что и для основной целевой переменной). При доказательстве третьего клоза находится правило:

```
rule(1,[gt("T","200")],assign("Neispr","тип 1"))
```

Для этого правила предикат proverka выполняет проверку условной части: [gt("T","200")]. В предикате prov1 временной целью становится переменная T. Ее значение вводится во втором клозе предиката obr_vyvod, так как эта переменная запрашивается у пользователя ЭС. Пусть введено значение "120"; при этом создается предикат базы данных znach("T","120"). Сравнение 120>200 (в предикате prov1) заканчивается неудачей. Таким образом, попытка доказать условие gt("T","200") заканчивается неудачей. Поэтому завершается неудачей предикат proverka, выполнявший проверку условной части правила 1.
Происходит возврат к следующему правилу, в заключении которого имеется переменная Neispr. Это правило 2:

```
rule(2,[gt("T","100"),le("T","200"),eq("Vibr","да")], assign("Neispr","тип 2"))
```

При проверке его условной части (предикатами proverka и prov1) временной целью дважды становится переменная T и один раз - Vibr. Значение переменной T определяется в первом клозе предиката obr_vyvod, так как оно

уже сохранено в базе данных. Значение переменной Vibr вводится во втором клозе предиката obr_vyvod. Пусть для переменной Vibr введено значение "да"; при этом создается предикат базы данных znach(“Vibr”, ”да”). В результате условная часть правила 2 оказывается верной, и в базе данных создается предикат znach("Neispr", "тип 2").
На этом заканчивается доказательство предиката obr_vyvod("Neispr", X), вызванного из предиката prov1 при проверке условия eq("Neispr"," тип 1").
Переменная программы X (а для ЭС – переменная Neispr) при этом получила значение "тип 2". Сравнение "тип 2"="тип 1" завершается неудачей. Поэтому заканчиваются неудачей предикаты prov1(eq("Neispr", "тип 1")) и proverka ([eq("Neispr","тип 1")]). Таким образом, заканчивается неудачей проверка условной части правила 4. Происходит переход к следующему правилу (т.е. предикату rule), в котором определяется переменная "D". Это правило 5:

rule(5,[eq("Neispr", "тип 2")], assign("D", "прекратить работу")).

Предикаты proverka и prov1 выполняют проверку посылки этого правила. Переменная Neispr снова становится временной целью (в предикате prov1). Ее значение определяется уже в первом клозе предиката obr_vyvod, так как оно было найдено ранее: в базе данных имеется предикат znach("Neispr", "тип 2").
Сравнение "тип 2"="тип 2" выполняется успешно. Поэтому успешно доказывается предикат prov1(eq("Neispr", "тип 2")). Предикат proverka([eq("Neispr", "тип 2")]), проверявший условную часть правила 5, также доказывается успешно. Таким образом, правило 5 включается. В базе данных создается предикат znach("D", "прекратить работу").
На этом успешно завершается доказательство предиката obr_vyvod("D", Result). Переменная Result получила значение "прекратить работу". Это значение выводится на экран в предикате vyvod.

Порядок выполнения работы

В программе, разработанной в лабораторной работе 5, реализовать механизм обратного логического вывода. Отладить его на примере работы с базой знаний, построенной в лабораторной работе 5.

Контрольные вопросы

1. Механизм вывода в ЭС. Обратный логический вывод.
2. Реализация обратного логического вывода на языке Пролог.
3. Реализация проверки условной части продукционного правила на языке Пролог.
4. Реализация ввода данных от пользователя в процессе логического вывода.
5. Пример последовательности обратного логического вывода в экспертной системе, реализованной на языке Пролог.