Министерство образования Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

В.М. Зюзьков

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ЭКСПЕРТНЫЕ СИСТЕМЫ

Учебное методическое пособие

1999

Зюзьков В.М.
Искусственный интеллект и экспертные системы: Учебное методическое пособие. — Томск: Томский межвузовский центр дистанционного образования, 1999. — 76 с.

Методическое пособие содержат учебную программу, требования к выполнению контрольных заданий и курсовой работы, варианты заданий, темы курсовых работ, рекомендуемую литературу. В качестве источника для получения теоретических знаний студенту необходимо и достаточно использовать учебные пособия по логическому программированию и искусственному интеллекту, автор В. М. Зюзьков.

СОДЕРЖАНИЕ

1. УЧЕБНАЯ ПРОГРАММА ПО ДИСЦИПЛИНЕ "ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ЭКСПЕРТНЫЕ СИСТЕМЫ"

Цель и задачи дисциплины, ее место в учебном процессе

Целью преподавания дисциплины является обучение методам искусственного интеллекта и логическому программированию как инструменту для создания программ искусственного интеллекта.

В результате изучения дисциплины студенты должны:

- знать основные методы искусственного интеллекта для решения задач;
- уметь программировать на Прологе;
- уметь создавать экспертные системы.

Для данного курса необходимо изучение следующих предметов: "Алгоритмические языки и программирование" и "Дискретная математика".

Содержание лекций

1. Введение в искусственный интеллект.

Предмет "Искусственный интеллект". Структура исследований в области искусственного интеллекта. Этапы в разработке искусственного интеллекта (1637-1992). Психологическая теория интеллекта. Представление знаний и вывод знаний.

2. Логическое программирование

Введение в логическое программирование. История. Логический вывод. Применение метода резолюций для ответов на вопросы.

Введение в язык Пролог. Особенности языка Пролог. Пример Пролог-программы: родственные отношения. Фразы Хорна как средство представления знания. Алгоритм работы интерпретатора Пролога.

Семантика Пролога. Порядок предложений и целей. Декларативная и процедурная семантики.

Структуры данных. Арифметика в Прологе. Структуры. Списки. Примеры использования структур.

Внелогические предикаты управления поиском. Ограничение перебора. Примеры, использующие отсечение. Отрицание как неудача. Трудности с отсечением и отрицанием. Программирование повторяющихся операций.

Метапрограммирование. Эквивалентность программ и данных. Предположение об открытости мира.

Внелогические предикаты базы знаний и ввода-вывода. Доступ к программам и обработка программ. Ввод и вывод. Программа, которая учится у пользователя. Программирование второго порядка. Запоминающие функции.

Модификация синтаксиса (операторная запись).

Примеры программ.

3. Решение задач.

Задача о кубиках. Общий метод решения задач. Основные методы поиска. Сведение задач к подзадачам. И/ИЛИ-графы. Игры и минимаксный принцип.

4. Экспертные системы.

Функции и структура экспертной системы. Продукции и неопределенность. Требования к современным экспертным системам.

2. КОНТРОЛЬНЫЕ ЗАДАНИЯ

2.1. Контроль обучения

Дисциплина "Искусственный интеллект и экспертные системы" изучается на протяжении двух семестров (9, 10). В процессе дистантного обучения дисциплине студент должен выполнить два контрольных задания, выполнить компьютерную экзаменационную работу и курсовую работу. Каждое контрольное задание требует создания нескольких программ.

В 9 семестре студент выполняет два контрольных задания. Контрольное задание состоит из нескольких задач, требующих составить программы на Прологе. Составленные и отлаженные программы (*обязательно с комментариями в тексте*) студент по мере выполнения периодически пересылает *электронной почтой или на дискете* диспетчеру центра дистантного обучения, который в свою очередь пересылает их лектору. Лектор проверяет программы и при правильном выполнении программы студент получает подтверждение о том, что они зачтены. Если программа составлена неправильно, студент получает от лектора текстовый файл, в котором содержится описание ошибок программы. При выполнении двух контрольных заданий студент получает зачет. В конце семестра студент выполняет компьютерную экзаменационную работу. Ему потребуется ответить на 10 случайно выбранных вопросов из 100 возможных. По результатам ответа студенту ставится экзаменационная оценка.

В 10 семестре студент делает курсовую работу и получает дифференцируемый зачет.

2.2. Инсталляция SWI-Prolog'a

Интерпретатор с языка Пролог представлен в виде упакованного файла pl.arj. SWI-Prolog работает в среде Windows (3.11 или 95); для его установки на компьютере достаточно распаковать файл pl.arj вместе со всеми хранящимися в этом файле каталогами. Запустите файл pl.exe из каталога pl\bin и вы войдете в интерпретатор. Пролог выдает приглашение "?-" и ждет вашего ввода.

Чтобы набирать программы, надо воспользоваться каким-либо внешним текстовым редактором (SWI-Prolog не имеет собственного редактора). Например, вы можете пользоваться стандартной программой Notepad ("блокнот"). Набранные программы сохраняйте в виде текстовых файлов с расширением pl (но можно и расширение txt) в рабочем каталоге. По умолчанию рабочим каталогом будет pl\bin, но вы можете это изменить: для этого соз-

дайте ярлык для pl.exe и измените рабочий каталог в свойствах этого ярлыка. Чтобы загрузить программу из файла c:\pl\work\name.pl, находясь в среде интерпретатора, вы должны в ответ на приглашение "?-" набрать имя файла в квадратных скобках

?- [name].

если установлен рабочий каталог - pl\work, или набрать имя файла с полным путем к нему,

?-['c:\pl\work\name.pl'].

если установлен другой рабочий каталог. Отметим следующую особенность работы в SWI-Prolog с блокнотом. После набора текста программы обязательно перейдите курсором на новую строчку, иначе SWI-Prolog при загрузке файла с программой выдаст ошибку "неожиданный конец файла".

2.3. Первое контрольное задание

Задание состоит из 5 задач, в которых требуется составить программы на Прологе. Первые две задачи требуют запрограммировать простые предикаты. Следующие две - требуют написать простые программы. В последней задаче требуется составить более сложную программу на Прологе (как правило, требуется определить несколько предикатов). При составлении программ (если не оговорено противное) можно использовать все встроенные предикаты Пролога. Тексты всех программ, если вы мыслите в духе логического программирования, получаются небольшие.

SWI-Prolog не имеет стандартного help'a для Windows, для этого используется предикат help. Вызов help(<имя предиката>) выдает на экран информацию об этом предикате. Вызов help(7) выдает на экран список всех встроенных предикатов с комментариями. Текстовый файл руководства по SWI-Prolog - pl\library\manual. Отладку предикатов можно осуществлять с помощью предиката трассировки trace(<имя предиката>), трассировка предиката отключается - trace(<имя предиката>, -all).

Пример задания с решением

Задача 1

Постройте предикат position_max(+L, -M, -N), который в списке L находит максимальное значение M и порядковый номер N этого значения.

Задача 2

Определите умножение целых чисел через сложение и вычитание.

Задача 3

Сортировка списка простым обменом (по возрастанию).

Задача 4

Пусть бинарное дерево задается рекурсивной структурой tree(<левое поддерево>,<корень>,<правое поддерево>) и пустое дерево задано термом nil. Составьте программу subtree(+S, +T), определяющую, является ли S поддеревом T.

Задача 5a

Определим операторы:

```
:- op( 100, fy, ~).
:- op( 110, xfy, &).
:- op( 120, xfy, v).
```

Булева формула есть терм, определяемый следующим образом: константы true и false - булевы формулы; если X и Y - булевы формулы, то и X v Y, X & Y, ~X - булевы формулы, здесь v и & - бинарные инфиксные операторы дизъюнкции и конъюнкции, а ~ - унарный оператор отрицания.

Напишите программу, распознающую логические формулы в конъюнктивной нормальной форме, т.е. формулы, являющиеся конъюнкцией дизъюнкций литералов, где литерал - атомарная формула или ее отрицание.

Задача 5b

Напишите предикат p(+N, -L) - истинный тогда и только тогда, когда список L содержит все последовательности (списки) из N нулей, единиц и двоек, в которых никакая цифра не повторяется два раза подряд (нет куска вида XX).

Решение

Задача 1

Будем использовать метод накапливающего параметра. Для этого введем вспомогательный предикат position_max(+List, +I, +M0, +N0, -M, -N), где I равен номеру рассматриваемого элемента в исходном списке, M0 - текущее значение максимума, N0 - позиция текущего максимума.

```
position_max([X|T],M,N):-
    position_max(T,1,X,1,M,N).
position_max([],_,M,N,M,N).

position_max([A|T],I,X,_,M,N):-
    A>X,
    K is I+1,
    position_max(T,K,A,K,M,N).

position_max([A|T],I,X,Y,M,N):-
    A=<X,
    K is I+1,
    position_max(T,K,X,Y,M,N).
```

```
?- position_max([3,2,5,1,4,3],M,N).
M = 5
N = 3
Yes
```

Задача 2

Определим предикат p(+X,+Y,?R), где X и Y сомножители, R - произведение.

Делаем рекурсию по второму аргументу предиката, используя рекурсивное определение X*Y=X*(Y-1)+X.

```
p(X,0,0).

p(X,Y,R):- Y>0,
           Y1 is Y-1,
           p(X,Y1,R1),
           R is R1+X.

p(X,Y,R):-  Y<0,
            Y1 is -Y,
            p(X,Y1,R).
```

Задача 3

Заданный список L сортируется по следующему алгоритму:

(1) если L содержит два соседних элемента, нарушающих требуемое упорядочение, то эти элементы меняются местами, после чего к полученному списку применяется этот же алгоритм;

(2) если в L не встречается ни одной неупорядоченной пары соседних элементов, то список L отсортирован и тем самым является искомым списком.

Предикат ord(+L) проверяет является ли список L отсортированным.

```
ord([]).
ord([_]).
ord([X,Y|T]):-
    X =< Y,
    ord([Y|T]).

change(L,L):-
    ord(L),!.
change(L,S):-
    append(L1,[X,Y|L2],L),
```

```
    X>Y,!,
    append(L1,[Y,X|L2],Z),
    change(Z,S).
```

Отсечения в данном случае ликвидируют многочисленные правильные ответы.

Задача 4

Для решения используется очевидная рекурсия.

```
subtree(X, X).
subtree(X, tree(L,_, _)):-
        subtree(X,L).
subtree(X,tree(_,_,R)):-
        subtree(X,R).
```

Задача 5а

Из определения операций видно, что конъюнкция и дизъюнкция являются право ассоциативными, т. е. запрос

?- X & Y & Z = A & B

приводит к унификации X = A, Y & Z = B.

Определим два вспомогательных предиката literal(X) и dis(X), которые проверяют является ли X литералом и элементарной дизъюнкцией соответственно.

```
literal(true).
literal(false).
literal(~ X):-
        literal(X).

dis(X):-
        literal(X).
dis(X v Y):-
        literal(X),
        dis(Y).

con(X):- dis(X).
con(X & Y):-
        dis(X),
        con(Y).
```

```
?- con( true & (~ false v true v false) & ~ true).
Yes
```

Задача 5b

```
p(1,[[0],[1],[2]]).
p(N,R):- N>1,
        N1 is N-1,
        p(N1,R1),
        t(R1,R).

t([],[]).
t([X|T],Z):-
        t(T,R1),
        s(X,R),
        append(R,R1,Z).

s([0|T],[[1,0|T],[2,0|T]]).
s([1|T],[[0,1|T],[2,1|T]]).
s([2|T],[[0,2|T],[1,2|T]]).
```

Варианты заданий

Вариант 1

1. Определите возведение в целую степень через умножение и деление.
2. Напишите предикат p(+L, -N) - истинный тогда и только тогда, когда N - предпоследний элемент списка L, имеющего не менее двух элементов.
3. Напишите предикат, аналогичный предикату subst (см. вариант 8, задача2), но производящий взаимную замену X на Y, т.е. X->Y, Y->X.
4. Напишите предикат, который определяет, является ли данное натуральное число простым.
   Воспользуйтесь более общей задачей:
   ispr(N, M) - "Число N не делится ни на одно число большее или равное M и меньшее N".
   Имеем ispr(N, M) -истинно, во-первых, если N = M, и, во-вторых, если истинно ispr(N,M+1) и N не делится на M.
5. Напишите предикат p(+N, +K, -L) - истинный тогда и только тогда, когда L - список всех последовательностей (списков) длины K из чисел 1,2,...,N.

Вариант 2

1. Напишите предикат p(+X, +N, -L) - истинный тогда и только тогда, когда L - список из N раз повторенных элементов X.
2. Напишите предикат p(+L, -S) - истинный тогда и только тогда, когда S - список списков элементов списка L, например, p([a, b, c],[[a], [b], [c]]) - истина.
3. Сортировка списка простой вставкой (по возрастанию).
4. Сортировка списка простым выбором (по возрастанию).

5. Напишите предикат p(+N, -L) - истинный тогда и только тогда, когда список L содержит все последовательности (списки) из N нулей и единиц, в которых никакая цифра не повторяется три раза подряд (нет куска вида XXX).

Вариант 3

1. Напишите предикат p(+L, -S) - истинный тогда и только тогда, когда L - список списков, а S - список, объединяющий все эти списки в один.
2. Напишите предикат p(+L, -S) - истинный тогда и только тогда, когда список S есть циклическая перестановка элементов списка L, например, p([f, g, h, j], [g, h, j, f]) -истина.
3. Напишите предикат p(+L, -N) - истинный тогда и только тогда, когда N - количество различных элементов списка L.
4. Напишите предикат p(+X, +Y, -Z) - истинный тогда и только тогда, когда Z есть "пересечение" списков X и Y, т.е. список, содержащий их общие элементы, причем кратность каждого элемента в списке Z равняется минимуму из его кратностей в списках X и Y.
5. Определите отношение ordered(+Tree), выполненное, если дерево Tree является упорядоченным деревом целых чисел, т. е. число, стоящее в любой вершине дерева, больше любого элемента в левом поддереве и меньше любого элемента в правом поддереве. Указание. Можно использовать вспомогательные предикаты ordered_left(+X, +Tree) и ordered_right(+X, +Tree), которые проверяют, что X меньше (больше) всех чисел в вершинах левого (правого) поддерева дерева Tree и дерево Tree - упорядочено.

Вариант 4

1. Напишите предикат p(+X, +N, +V, -L) - истинный тогда и только тогда, когда список L получается после добавления X на N-е место в список V.
2. Напишите предикат p(+N, +V, -L) - истинный тогда и только тогда, когда список L получается после удаления N-го элемента из списка V.
3. Запрограммируйте предикат p(+A,+B), распознающий, можно ли получить список элементов A из списка элементов B посредством вычеркивания некоторых элементов.

   Алгоритм: Если A - пустой список, то ответом будет "да". В противном случае нужно посмотреть, не пуст ли список B. Если это так, то ответом будет "нет". Иначе нужно сравнить первый элемент списка A с первым элементом списка B. Если они совпадают, то надо снова применить тот же алгоритм к остатку списка A и остатку списка B. В противном случае нужно снова применить тот же алгоритм к исходному списку A и остатку списка B.
4. Напишите предикат p(+X, +Y, +L) - истинный тогда и только тогда, когда X и Y являются соседними элементами списка L.
5. Определим операторы:

:- op(100, fy, ~).
:- op(110, xfy, &).
:- op(120, xfy, v).

Булева формула есть терм, определяемый следующим образом: константы true и false - булевы формулы; если X и Y - булевы формулы, то и X v Y, X & Y, ~X - булевы формулы, здесь v и & - бинарные инфиксные операторы дизъюнкции и конъюнкции, а ~ - унарный оператор отрицания.

Напишите программу, распознающую логические формулы в дизъюнктивной нормальной форме, т.е. формулы, являющиеся дизъюнкцией конъюнкций литералов, где литерал - атомарная формула или ее отрицание.

Вариант 5

1. Напишите предикат p(+V, -L) - истинный тогда и только тогда, когда список L получается после удаления всех повторных вхождений элементов в список V, например, p([a, b, c, d, d, a], [a, b, c, d]) - истина.
2. Напишите предикат p(+V, -L) - истинный тогда и только тогда, когда список L получается после удаления из списка V всех элементов, стоящих на четных местах, например, p([1, 2, 3, 4, 5, 6], [1, 3, 5]) - истина.
3. Определите отношение sum_tree(+TreeOfInteger, -Sum), выполненное, если число Sum равно сумме целых чисел, являющихся вершинами дерева TreeOfInteger.
4. Определим операторы:
   :- op(100, fy, ~).
   :- op(110, xfy, &).
   :- op(120, xfy, v).

   Булева формула есть терм, определяемый следующим образом: константы true и false - булевы формулы; если X и Y - булевы формулы, то и X v Y, X & Y, ~X - булевы формулы, здесь v и & - бинарные инфиксные операторы дизъюнкции и конъюнкции, а ~ - унарный оператор отрицания.

   Напишите предикат p(+T), определяющий, является ли данный терм T булевой формулой.
5. Определите предикат occurances(+Sub,+Term,-N), истинный, если число N равно числу вхождений подтерма Sub в терм Term. Предполагается, что терм Term не содержит переменных.

Вариант 6

1. Напишите предикат p(+V, +X, -L) - истинный тогда и только тогда, когда список L получается из списка V после удаления всех вхождений X на всех уровнях, например, p([1, [2, 3, [1]], [3, 1]], 1, [[2, 3, []], [3]]) - истина.
2. Напишите обобщение предиката member, когда ищется элемент на всех уровнях в списке.

3. Встроенный предикат functor(+Term, ?Functor, ?Arity) определяет для заданного составного терма Term его функтор Functor и местность Arity.
   Встроенный предикат arg(+N, +Term, ?Value) определяет для целого числа N и заданного составного терма Term его N-ый аргумент Value.
   Определите предикаты functor1 и arg1 - аналоги предикатов functor и arg через предикат univ (=..)
4. Напишите предикат range(?M, ?N, ?L), истинный тогда и только тогда, когда L - список целых чисел, расположенных между M и N включительно (предикат должен допускать различное использование, когда не менее двух из трех аргументов конкретизованы).
   (Указание. Используйте предикаты var(+X) и nonvar(+X)).
5. Определим операторы:
   :- op(100, fy, ~).
   :- op(110, xfy, &).
   :- op(120, xfy, v).

Булева формула есть терм, определяемый следующим образом: константы true и false - булевы формулы; если X и Y - булевы формулы, то и X v Y, X & Y, ~X - булевы формулы, здесь v и & - бинарные инфиксные операторы дизъюнкции и конъюнкции, а ~ - унарный оператор отрицания.

Напишите программу, задающую отношение negation_inward(+F1,-F2), которое выполнено, если логическая формула F2 получается из логической формулы F1 внесением всех операторов отрицания внутрь конъюнкций и дизъюнкций.

Вариант 7

1. Определите предикат p(+U, +V, -L) - истинный тогда и только тогда, когда список L есть список всех элементов списка U, не содержащихся в списке V.
2. Определите предикат p(+U, +V, -L) - истинный тогда и только тогда, когда L - список всех элементов, содержащихся либо в списке U, либо в списке V, но не одновременно в U и V.

1. Напишите новую версию процедуры "предок", которая вырабатывает список представителей всех промежуточных поколений, располагающихся между предком и потомком. Предположим, например, что Генри является отцом Джека, Джек - отцом Ричарда, Ричард - отцом Чарльза, а Чарльз - отцом Джейн. При запросе о том, является ли Генри предком Джейн, должен выдаваться список, характеризующий родственную связь этих людей, конкретно: [джек, ричард, чарльз].
2. Напишите предикат gcd(+A,+B,-D) - истинный тогда и только тогда, когда D -наибольший общий делитель двух целых положительных чисел A и B.
3. Разработайте программу "Советник по транспорту". Выберите либо сеть, состоящую из городов, либо транспортную сеть маршрутов поездов или

автобусов в пределах одного города. Вы должны информировать систему о том, откуда и куда Вы собираетесь добраться, а система должна выдавать рекомендации о том, какими поездами, автобусами, самолетами и т. д. Вам следует воспользоваться, чтобы добраться до пункта назначения. Указание: см. "Поиск в глубину" (Учебное пособие, часть 1, 2.2).

Вариант 8

1. Определите предикат p(+V, -L) - истинный тогда и только тогда, когда L - список всех элементов списка V, встречающихся в нем более одного раза.
2. Напишите предикат subst(+V, +X, +Y, -L) - истинный тогда и только тогда, когда список L получается после замены всех вхождений элемента X в списке V на элемент Y.
3. Напишите вариант программы plus(?X, ?Y, ?Z), пригодный для сложения, вычитания и разбиения чисел на слагаемые.
   (Указание. Используйте для порождения чисел встроенный предикат between(+Low, +High, ?Value), который порождает все целые числа от нижней границы Low до верхней границы High.)
4. Опишите процедуру для предиката расщепить/4, которая берет список целых чисел L1 и целое число N и выдает списки L2 и L3 такие, что числа из исходного списка, меньшие, чем N, помещаются в список L2, а остальные - в список L3.
5. Множественное число большинства английских существительных получается путем добавления буквы "s" к форме единственного числа. Но если существительное заканчивается буквой "y", следующей за согласной, множественное число образуется путем замены буквы "y" на сочетание "ies"; если же существительное заканчивается буквой "o", следующей за согласной, множественное число образуется путем добавления сочетания "es". Напишите утверждения для предиката множественное_число/2, которые задают все эти правила. Указание. Воспользуйтесь предикатом name/2.

Вариант 9

1. Напишите предикат p(+V, -L) - истинный тогда и только тогда, когда список L получается из списка V после удаления всех повторяющихся элементов, т. е. из списка получается множество.
2. Напишите предикат exists(+P, +L), который проверяет "Существует ли элемент списка L, удовлетворяющий предикату P?"
3. Напишите программу вычисления целочисленного квадратного корня из натурального числа N, определяемого как число I, такое, что $I*I \le N$, но $(I+1)*(I+1) > N$. Используйте определение предиката between/3 для генерирования последовательности натуральных чисел с помощью механизма возвратов.

4. Напишите предикат для вычисления чисел Фибоначчи, используя метод накапливающего параметра.
5. Одним из примеров использования предиката name/2 может служить генерация новых атомов для представления вновь вводимых объектов, например, abc1, abc2, abc3 и т.д. Эти имена характеризуются тем, что все они состоят из корня, определяющего тип именуемого объекта, и целочисленного суффикса для различения объектов одного типа. Напишите программу новое_имя(+X, -Y). Последовательность имен создается с помощью возвратов. Указание. Воспользуйтесь предикатом int_to_atom(+N,-X), который конвертирует натуральное число N в атом X.

Вариант 10

1. Напишите предикат all(+P, +L), который проверяет "Для всех ли элементов списка L выполняется предикат P? "
2. Напишите предикат filter(+V, +P, -L) - истинный тогда и только тогда, когда список L есть список всех элементов из списка V, удовлетворяющих предикату P ("фильтрация" списка).
3. Определите предикат p(+V, +N, -L) - истинный тогда и только тогда, когда L - список элементов списка V, встречающихся в нем не менее N раз. Проверьте работу этого предиката на примере [a, a, b, a, c, b, c, a, b, b, d, a, b] для N=1,2,5,0.
4. Напишите предикат summa_digits(+N, -S) - истинный тогда и только тогда, когда S - сумма цифр натурального числа N.
5. Построить программу "сжать", назначение которой - преобразование английских слов в их "звуковой" код. Этот процесс предусматривает "сжатие" примерно одинаково звучащих слов в одинаковый их код - своего рода, аббревиатуру этих слов. Слова "сжимаются" в соответствии со следующими правилами:
   - первая буква слова сохраняется;
   - все последующие за ней гласные, а также буквы "h", "w" и "y" удаляются;
   - сдвоенные буквы заменяются одиночными;
   - закодированное слово состоит не более чем из четырех букв, остальные буквы удаляются.

   Примеры: сжать(barrington, brng) и сжать(llewellyn, ln) - выполнено.
   Указание. Воспользуйтесь предикатом name/2.

2.4. Второе контрольное задание

Второе контрольное задание состоит из двух задач. Вам надо запрограммировать на Прологе решение логической задачи и создать экспертную систему.

Каким образом можно создавать экспертные системы на Прологе вы прочитаете в 3 разделе. Там же формулируется точно задача, что вам необходимо сделать.

Логическую задачу вы решаете в соответствии с данным вам вариантом. Для решения логической задачи (“волк, коза и капуста”, “отец и два сына”, “ 8 ферзей”, “ рыцари и дамы”, “миссионеры и людоеды”, “обезьяна и банан” и др.) вам необходимо представить задачу в виде пространства состояний и найти решение с помощью поиска в глубину или ширину.

Логические задачи

Отец и два сына

Отец, два его сына и лодка находятся по одну сторону реки. Отец весит 80 кг, сыновья - по 40 кг. Как переправить эту семью на другую сторону, если лодка выдерживает только 80 кг?

Указания к решению. Различные состояния этой задачи задаются информацией, на каком берегу находятся лодка, отец, первый сын и второй сын. Поэтому структура state(Father,Son1,Son2,Boat) полностью описывает состояние. Возможные значение каждого аргумента: атомы west (западный берег) и east (восточный берег). Начальное состояние: state(east,east,east,east). Конечное состояние: state(west,west,west,west).

Миссионеры и людоеды

Три миссионера и три людоеда находятся по одну сторону реки, через которую они хотят переправиться. В их распоряжении имеется лодка, которая может выдержать вес только двух человек. Кроме того, если в какой-то момент число людоедов станет больше числа миссионеров, миссионеры будут съедены независимо от того, на каком берегу реки это случится.

Указания к решению. Различные состояния этой задачи однозначно задаются информацией, на каком берегу находятся лодка и сколько миссионеров и людоедов на этом же берегу.

Поэтому структура

state(ЛокализацияЛодки,

ЧислоМиссионеровНаТомБерегуГдеЛодка,

ЧислоЛюдоедовНаТомБерегуГдеЛодка)

полностью описывает состояние. Допустимые состояния для решения задачи - это те, когда людоеды не могут съесть миссионеров ни на том берегу, где лодка, ни на противоположном,

Возможные значение первого аргумента: атомы west (западный берег) и east (восточный берег). Возможные значения остальных аргументов: 0, 1, 2 или 3.

Начальное состояние: state(east,3, 3). Конечное состояние: state(west,3,3).

Задача об обезьяне и банане

Возле двери комнаты стоит обезьяна. В середине этой комнаты к потолку подвешен банан. Обезьяна голодна и хочет съесть банан, однако она не может дотянуться до него, находясь на полу. Около окна этой же комнаты на полу лежит ящик, которым обезьяна может воспользоваться. Обезьяна может предпринимать следующие действия: ходить по полу, залезать на ящик, двигать ящик (если она уже находится около него) и схватить банан, если она уже стоит на ящике прямо под бананом. Может ли обезьяна добраться до банана?

Указания к решению задачи. Различные состояния задачи можно описать структурой

```
state(ПоложениеОбезьяныВКомнате,  %  значения: дверь, окно, середина
      ОбезьянаНаЯщикеИлиНет,        %  значения: ящик, пол
      ПоложениеЯщикаВКомнате,       %  значения: дверь, окно, середина
      ИмеетИлиНеИмеетБанан)         % значения: да, нет
```

Существует 4 типа ходов:

1) схватить банан - если обезьяна на ящике в середине комнаты и банана не имеет;
2) залезть на ящик, если обезьяна находится на полу, рядом с ящиком;
3) подвинуть ящик с одного разрешенного места на другое, если обезьяна на полу рядом с ящиком;
4) перейти по полу с одного разрешенного места на другое.

Начальное состояние: state('дверь','пол','окно','нет'). Конечное состояние: state(_,_,_,'да').

Задача о 8 ферзях

На шахматной доске 8×8 надо расставить 8 ферзей, чтобы ни один из ферзей не находился под боем другого.

Указания к решению. Положение одного ферзя на доске задается парой целых чисел от 1 до 8: номера вертикалей и горизонталей. Легко видеть, что каждая вертикаль (как впрочем, и горизонталь) должна содержать по одному ферзю. Поэтому, мы можем экономно задавать положение на доске восьми ферзей списком из 8 чисел, задающих значения только горизонтальной координаты: так, например, список [2,5,7,4,3,6,8] говорит, что первый ферзь стоит на клетке (2, 1), второй - (5, 2), третий - (7, 3) и т. д. Различные состояния решения задачи мы можем представить в виде списка горизонтальных координат уже стоящих на доске ферзей, так, например, список [2] говорит о том, что на доске стоит только один ферзь на клетке (2,1), список [2,5] говорит о том, что на доске два ферзя на клетках (2, 1) и (5,2). Список длиной 8, говорит о том, что расставлены все ферзи.

Теперь нужно определить разрешенные переходы из одного состояния задачи в другое. Для этого надо определить предикат

'не бьет'(НовыйФерзь, ФерзиУжеСтоящиеНаДоске)

Этот предикат легко определить рекурсивно:

"НовыйФерзь не бьет список ферзей [F|T], если он не бьет ферзь F и не бьет список ферзей T".

Начальное состояние: []. Конечное состояние: [_,_,_,_,_,_,_,_].

Поскольку решение этой задачи при поиске выдает последовательность состояний, т.е. список списков, а нас интересует только конечная расстановка, то во втором аргументе предикат solve (см. программу поиска в глубину в лекциях) достаточно конкретизовать только голову списка: solve([],[H|_]).

Задача о волке, козе и капусте

На одном берегу реки находится фермер, волк, коза и капуста. Рядом лодка. Как фермер может переправиться со всем этим "хозяйством" на другой берег? С собой на лодку он может взять только один объект: волка, козу или капусту. Когда он в лодке, на любом из берегов не должна быть "криминальная" ситуация: волк не должен находиться с козой, а коза не должна быть вместе с капустой.

Указания к решению. Различные состояния этой задачи задаются информацией, на каком берегу находятся фермер, волк, коза и капуста, лодка всегда находится там, где фермер. Поэтому структура state(Фермер,Волк,Коза,Капуста) полностью описывает состояние. Возможные значение каждого аргумента: атомы west (западный берег) и east (восточный берег). Начальное состояние: state(east,east,east,east). Конечное состояние: state(west,west,west,west).

Рыцари и дамы

На берег реки приезжают 3 рыцаря, каждый со своей дамой. В их распоряжении имеется лодка, способная вместить не более 2-х человек. Как смогут перебраться на другой берег рыцари со своими дамами, если требуется выполнить условие: ни одна дама не может остаться без своего рыцаря в обществе других рыцарей - она тут же подвергается насилию (нравы "рыцарей" не слишком-то изменились с тех пор). Лошади переплывают реку сами, дамы гребут веслами, как и рыцари, лодка может пересекать реку сколько угодно раз.

Указания к решению. Удобно рыцарей обозначать атомами a, b и c и их соответствующих дам тоже теми же атомами. Различные состояния этой задачи задаются информацией, на каком берегу находятся лодка и списком рыцарей и дам, находящихся же на этом берегу. Поэтому структура state(НаКакомБерегуЛодка,

СписокРыцарейНаТомБерегуГдеЛодка,

СписокДамНаТомБерегуГдеЛодка)

полностью описывает состояние. Возможные значение первого аргумента: атомы west (западный берег) и east (восточный берег). Второй и третий аргу-

менты - списки из элементов a, b и c. В этой задаче списки рыцарей и дам нужно рассматривать как множества, т. е. не учитывать порядок перечисления и поэтому воспользоваться специальными предикатами для операций с множествами. Удобно, для отладки, так написать программу, чтобы список рыцарей (a, следовательно, дам) задавался фактом knight([a,b,c]); в этом случае легко будет менять число рыцарей для тестирования.
Начальное состояние: state(east,X,Y), где выполнено условие knight(M) и X и Y равны как множества списку M.
Конечное состояние: state(west, M,M), где выполнено условие knight(M).

Варианты логическиой задачи

Вариант 1
"Отец и два сына". Поиск в глубину.

Вариант 2
"Отец и два сына". Поиск в ширину.

Вариант 3
"Миссионеры и людоеды". Поиск в глубину.

Вариант 4
"Миссионеры и людоеды". Поиск в ширину.

Вариант 5
"Задача об обезьяне и банане". Поиск в глубину.

Вариант 6
"Задача об обезьяне и банане". Поиск в ширину.

Вариант 7
Задача о 8 ферзях. Поиск в глубину.

Вариант 8
Задача о волке, козе и капусте. Поиск в глубину.

Вариант 9
Задача о волке, козе и капусте. Поиск в ширину.

Вариант 10
Рыцари и дамы. Поиск в глубину с ограничением глубины до 16

"Вопросы образуют так называемую пирамидальную иерархию... и в этой иерархии имеется так называемый уровень Тютиквоцитока, именуемый также верхним пределом, поскольку выше этого уровня никто уже не в состоянии понять ни вопроса, ни ответа... Зато из ответов на вопросы, задаваемые ниже барьера Тютиквоцитока, можно извлекать практическую пользу, и тут нет ничего удивительного и ничего нового, ибо... не обязательно знать ни историю возникновения ржи, ни способы ее выращивания, ни теорию и практику хлебопечения, а нужно только вонзить зубы в лепешку, и баста."

Станислав Лем "Осмотр на месте"

3. СОЗДАНИЕ ЭКСПЕРТНЫХ СИСТЕМ НА ПРОЛОГЕ

Экспертные системы - наиболее быстро развивающаяся и плодотворная область применения Пролога. В данном случае под экспертной системой понимается программа, поддерживающая принятие решения пользователем для классификации объекта из предметной области. Описываются принципы построения и организации экспертных систем в терминах их компонент: базы знаний и механизма вывода. Показано как проектировать и реализовать экспертную систему, основанную на продукциях.

В педагогических целях программа создается с помощью нисходящего программирования и изложение построения программы прекращается за шаг, до того как полноценная экспертная система будет построена. Предлагается этот шаг сделать самим студентам.

3.1. Метаинтерпретатор

Простой метаинтерпретатор

Пролог является естественным языком для построения экспертных систем. Существуют различные подходы к программированию экспертных систем. Опишем простой и элегантный метод создания оболочки экспертных систем на базе метаинтерпретатора Пролога.

Метапрограммы обращаются с другими программами как с данными; они выполняют их анализ, преобразование и моделирование. Легкость разработки метапрограмм, или метапрограммирования, на Прологе обусловлена эквивалентностью программ и данных - и те и другие являются термами Пролога. Мы рассматриваем метапрограммы особого класса - метаинтерпретаторы.

Метаинтерпретатор для некоторого языка - это интерпретатор для языка, написанный на том же самом языке. Некоторые языки программирования, такие как Лисп и Пролог, позволяют легко разрабатывать метаинтерпретаторы, что является важным свойством таких языков. Оно делает возможным построение интегрированной среды программирования и обеспечивает доступ к вычислительным средствам языка.

Опишем простой интерпретатор для чистого Пролога. Отношение *solve(Goal)* истинно, если цель *Goal* истинна в отношение программы, подлежащей интерпретации.

```
% программа 1
 solve(true):-!.    % поскольку  clause(true,true) - также истина
 solve((A,B)):-
     solve(A),
     solve(B).
 solve(A):-
     clause(A,B),
     solve(B).
```

Интерпретатору соответствует следующее декларативное толкование. Константа *true* является истинной. Конъюнкция *(A,B)* истинна, если истинна цель *A* и истинна цель *B*. Цель *A* истинна, если в интерпретируемой программе существует предложение *A :- B* , такое, что цель *B* истинна.

Теперь дадим процедурное толкование этим трем предложениям программы 1. Факт *solve* устанавливает, что пустая цель, представленная в Прологе атомом *true*, достижима. Следующее предложение относится к конъюнктивным целям. Оно читается так: “Для достижения конъюнкции целей *(A,B)* необходимо достичь цели *A* и цели *B*”. Общий случай редукции цели покрывается последним предложением программы. Чтобы доказать некоторую цель, из программы выбирается предложение, заголовок которого унифицируется с целью, а затем рекурсивно применяется тело предложения.

Процедурное толкование предложений Пролога необходимо, чтобы показать, что метаинтерпретатор, представленный программой 1, действительно отражает возможности Пролога при реализации абстрактной вычислительной модели логического программирования. Такими возможностями, например являются выбор для редукции крайней левой цели и использование последовательного поиска с возвратом при выборе предложения для редукции цели. Порядок целей в теле предложения *solve*, содержащего конъюнктивные цели, гарантирует, что самая левая цель в конъюнкции решается первой. Последовательный поиск и возврат осуществляются при доказательстве цели *clause* согласно принципам выполнения Пролог-программ.

Постоянная работа интерпретатора обеспечивается третьим предложением программы 1. При вызове предложения *clause* выполняется унификация с заголовками предложений, имеющихся в программе. Это предложение от-

ветственно также за получение различных решений при возвратах. Кроме того, возвраты происходят и в конъюнктивном правиле (возвраты от цели *B* к цели *A*).

Поучителен разбор протокола работы метаинтерпретатора, представленного программой 1, при доказательстве некоторой цели.

Определение предиката *member:*

```
?- listing(member).

member(A, [A|B]).
member(A, [B|C]) :-
        member(A, C).
```

Интерпретация предиката *member:*

```
?- solve(member(X,[a,b,c])).

X = a ;
X = b ;
X = c ;
No
```

Включение трассировки для *solve:*

```
?- trace(solve,[+call,+exit]).
      solve/1: call exit
Yes
```

Трассировка (частичная) *solve:*

```
?- solve(member(X,[a,b,c])).
T Call:  (  6) solve(member(G988, [a,b,c]))
T Call:  (  7) solve(true)
T Exit:  (  7) solve(true)
T Exit:  (  6) solve(member(a, [a,b,c]))

X = a ;
T Call:  (  7) solve(member(G988, [b,c]))
T Call:  (  8) solve(true)
T Exit:  (  8) solve(true)
T Exit:  (  7) solve(member(b, [b,c]))
T Exit:  (  6) solve(member(b, [a,b,c]))
```

X = b ;
T Call: (8) solve(member(G988, [c]))
T Call: (9) solve(true)
T Exit: (9) solve(true)
T Exit: (8) solve(member(c, [c]))
T Exit: (7) solve(member(c, [b,c]))
T Exit: (6) solve(member(c, [a,b,c]))

X = c ;
T Call: (9) solve(member(G988, []))
No

Построение дерева доказательства

Простым примером использования метаинтерпретатора является построение дерева доказательства в процессе решения определенной цели. Дерево доказательства полезно для средств объяснения в экспертных системах.

```
% программа 2
 solve(true,true):-!.
 solve((A,B),(ProofA,ProofB)):-
     solve(A,ProofA),
     solve(B,ProofB).
 solve(A,(A:-Proof)):-
     clause(A,B),
     solve(B,Proof).
```

Основным отношением метаинтерпретатора является отношение *solve(Goal,Tree),* где *Tree* - дерево доказательства для решения цели *Goal* .

Дерево доказательства представляется структурой *Goal:-Proof,* где *Proof* - конъюнкция ветвей (подцелей) доказываемой цели *Goal.* Программа 2, реализующая предикат *solve/2*, является простым расширением программы 1. Три её предложения точно соответствуют трем предложениям метаинтерпретатора для чистого Пролога.

Факт *solve* утверждает, что пустая цель истинна и имеет тривиальное дерево доказательства, представляемое атомом *true.* Во втором предложении утверждается, что дерево доказательства конъюнктивной цели *(A,B)* представляет собой конъюнкцию деревьев доказательства целей *A* и *B.* Последнее предложение *solve* строит дерево доказательства *A:-Proof* для цели *A,* в котором *Proof* строится рекурсивно при решении тела предложения, используемого для редукции цели *A.*

Рассмотрим пример использования программы 2 для интерпретации предиката member:

?- solve(member(X,[a,b,c]),P).

P = member(a, [a,b,c]) :- true
X = a ;

P = member(b, [a,b,c]) :- (member(b, [b,c]) :- true)
X = b ;

P = member(c, [a,b,c]) :- (member(c, [b,c]) :- (member(c, [c]) :- true))
X = c ;
No

Метаинтерпретатор для полного Пролога

Для того чтобы обрабатывать программы, в которых используются средства, выходящие за рамки чистого Пролога, метаинтерпретатор, представленный программой 2, должен быть расширен.

Различные системные предикаты не определяются предложениями программы и поэтому требуют отдельной обработки. Простейший способ обращения к этим системным предикатам состоит в непосредственном их вызове с использованием метапеременных. Необходима таблица, устанавливающая, какие предикаты являются системными. Будем считать, что она состоит из фактов вида *system(Predicate)* для каждого системного предиката. Предложения метаинтерпретатора, оперирующие с системными предикатами, имеют вид

solve(A):- system(A),A.

```
% программа 3
 solve(true):-!.
 solve((A,B)):-!,
     solve(A),
     solve(B).
 solve(not(A)):-!,
     not(solve(A)).
 solve(A):-
     not system(A),
     clause(A,B),
     solve(B).
 solve(A):-
     system(A),
     A.

% некоторые системные предикаты
```

```
system(is(_,_)).
system(_=_).
system(_<_).
system(_>_).
system(write(_)).
system(nl).
```

Дополнительное предложение *solve* делает действие системных предикатов невидимым для интерпретатора. Существуют некоторые системные предикаты, которые должны быть видимыми, например, отрицание. Проблемой в этом метаинтерпретаторе является корректное моделирование отсечения.

Продемонстрируем работу программы 3 на предикате p.

```
p(X,Y):-
    not(X=15),
    Y is X*X.
p(15,0).
p(X,X):-
    X<0.
```

Интерпретация предиката p:

?- solve(p(20,X)).

X = 400 ;
No

?- solve(p(-5,X)).
X = 25 ;
X = -5 ;
No

?- solve(p(15,X)).

X = 0 ;
No

Построение дерева доказательства для полного Пролога

Метаинтерпретатор для построения дерева доказательства, представленный в программе 2, также может быть расширен добавлением специальных предложений для системных целей и отрицания.

```
% программа 4
solve(true,true):-!.
```

```
solve((A,B),(ProofA,ProofB)):-!,
    solve(A,ProofA),
    solve(B,ProofB).
solve(not(A),'не доказуемо'(A)):-!,
    not(solve(A,_)).
solve(A,(A:-Proof)):-
    not system(A),
    clause(A,B),
    solve(B,Proof).
solve(A,(A:-true)):-
    system(A),
    A.
system(is(_,_)).
system(_=_).
system(_<_).
system(_>_).
system(write(_)).
system(nl).
```

Интерпретируем сложную цель:
?- solve((member(X,[a,b,c]),not member(X,[c,d])),Y).

X = a
Y = (member(a, [a,b,c]) :- true), 'не доказуемо'(member(a, [c,d])) ;

X = b
Y = (member(b, [a,b,c]) :- (member(b, [b,c]) :- true)), 'не доказуемо'(member(b, [c,d])) ;
No

3.2. Экспертная система

Экспертная система спрашивает у пользователя недостающую информацию.

Общепринято представлять экспертную систему в виде базы знаний и механизма вывода. Механизм вывода будет построен путем усовершенствования метаинтерпретатора (программа 4). Базы знаний, образованные средствами Пролога, являются выполняемыми программами. Рассмотрим упрощенный пример такой базы знаний - база знаний "Болезни". Для более наглядного инфиксного изображения предикатов специально введены операторы: ‘-рекомендовано', 'имеет', 'имеет симптом', 'имеет признак'.

```
:- op(100,xfx,['- рекомендовано','имеет','имеет симптом',
            'имеет признак']).

X '- рекомендовано' 'лечь в постель и принять аспирин':-
    X 'имеет' 'простуда',
    not( X 'имеет' 'уязвимый возраст').
X '- рекомендовано' 'вызвать врача':-
    X 'имеет' 'простуда',
    X 'имеет' 'уязвимый возраст'.
X '- рекомендовано' 'вызвать врача':-
    X 'имеет' 'острый фарингит'.
X '- рекомендовано' 'лечь в постель и принять аспирин':-
      X 'имеет' 'грипп'.

X 'имеет' 'простуда':-
    X 'имеет симптом' 'мышечные боли',
    X 'имеет симптом' 'лихорадка'.
X 'имеет' 'уязвимый возраст':-
    X 'имеет признак' 'моложе 8 лет'.
X 'имеет' 'уязвимый возраст':-
    X 'имеет признак' 'старше 60 лет'.
X 'имеет' 'острый фарингит':-
    X 'имеет симптом' 'лихорадка',
    X 'имеет симптом' 'нарывы в горле'.
X 'имеет' 'грипп':-
    X 'имеет симптом' 'насморк',
    X 'имеет симптом' 'мышечные боли',
    not(X 'имеет симптом' 'лихорадка').
hy(X,Y):-
  member(Y,['лечь в постель и принять аспирин','вызвать врача']),
  X '- рекомендовано' Y.
```

Предикат *hy* является целью для экспертной системы с данной базой знаний, с его помощью пользователь пытается выяснить справедливость гипотез *'лечь в постель и принять аспирин'* и *'вызвать врача'*. Попытаемся получить рекомендацию для Боба:

```
?- solve(hy(bob,Y)).
No
```

В базе знаний не заложены конкретные симптомы, предполагается, что экспертная система может задавать пользователю вопросы в случае нехватки информации.

Мы вводим специальную процедуру *askable/1*, которая в случае безуспешного решения цели интерпретатором может направить её на рассмотрение пользователю.

```
% программа 5
:- dynamic untrue/1.
solve(true):-!.
 solve((A,B)):-!,
      solve(A),
      solve(B).
 solve(not(A)):-!,
      not(solve(A)).
 solve(A):-
      not system(A),
      clause(A,B),
      solve(B).
 solve(A):-
      system(A),
      A.
 solve(A):-
     askable(A),
     not known(A),
     ask(A,Answer),
     respond(Answer,A).
 system(is(_,_)).
 system(_=_).
 system(_<_).
 system(_>_).
 system(write(_)).
 system(nl).
 system(member(_,_)).

ask(A,Answer):-
    display_query(A),
    read(Answer).

respond(yes,A):-
    assert(A).
respond(no,A):-
    assert(untrue(A)),fail.

known(A):-A,!.
known(A):-
    untrue(A).

display_query(A):-
```

```
    write(A),
    write('?').
```

Предикат *askable* определяет какие вопросы будут заданы пользователю. Чтобы избежать повторения одних и тех же вопросов, программа записывает ответы на вопросы, что обеспечивается предикатом *respond* . Если на вопрос *A* последовал ответ *yes*, то в программу вводится факт *A*. Если же получен ответ *no*, то в программу вводится факт *untrue(A)*. Эта информация используется предикатом *known*, чтобы избежать задания вопросов, ответы на которые уже известны программе. Директива *dynamic* для *untrue* добавлено для того, чтобы избежать предупреждение Пролога об отсутствии определения *untrue*.

Дополняем базу знаний предикатом *askable* и директивой dynamic для предикатов *‘имеет симптом’* и *‘имеет признак’*.

```
dynamic 'имеет симптом'/2, 'имеет признак'/2.

askable(_ 'имеет симптом' _).
askable(_ 'имеет признак' _).
```

Теперь получим рекомендации для Боба:

?- solve(hy(bob,Y)).
bob имеет симптом мышечные боли? yes.
bob имеет симптом лихорадка? yes.
bob имеет признак моложе 8 лет? no.
bob имеет признак старше 60 лет? no.
Y = 'лечь в постель и принять аспирин' ;
bob имеет симптом насморк? yes.
bob имеет симптом нарывы в горле? yes.

Y = 'вызвать врача' ;
No

Повторный запрос для Боба приводит сразу к тому же ответу:

?- solve(hy(bob,Y)).

Y = 'лечь в постель и принять аспирин' ;
Y = 'вызвать врача' ;
No

Оболочка независима от базы знаний

В экспертной системе оболочка должна быть независима от базы знаний, поэтому механизм вывода и база знаний помещаются в два разных файла. Такие предикаты как *system* и *askable* присутствуют в обоих файлах, следовательно необходима специальная директива для интерпретатора Пролога *multifile.*

```
% программа 6
:- multifile system/1, askable/1.
% остальной текст программы  как в программе 5, за исключением
% того, что убираем правило  system(member(_,_))
```

База знаний "Болезни" находится в отдельном файле и дополняется правилом system(member(_,_)).

Пользователь может просить объяснения "почему?"

Усовершенствованная версия оболочки позволяет также успешно вести диалоги и с другой стороны. Когда программа задает вопрос, пользователь может ответить своим собственным вопросом "почему?". Естественным ответом оболочки должно быть правило, на основании которого программа пытается сделать вывод. Такую возможность легко ввести в оболочку, расширяя все отношения дополнительным аргументом - используемым текущим правилом. Поскольку в Пролог-программах доступ к глобальному состоянию процесса вычислений невозможен, это правило должно быть явно представлено в дополнительном аргументе.

Интерфейс для обеспечения ответа на вопрос "почему?" реализуется дополнительными правилами предиката *respond*. Предикат *respond* переписывает текущее родительское правило и приглашает пользователя ответить еще раз. Формат, в котором будет представлено это правило, определяется предикатом *display_rule,* позволяющим пользователю представлять правила в удобной для него форме.

Повторные ответы на вопрос "почему?", использующие предложение *respond,* приводят к повторной переформулировке родительского правила. Решение состоит в выдаче прародительского правила в ответ на второй вопрос "почему?", прапрародительского правила в ответ на третье "почему?" и так далее вверх по дереву поиска. С этой целью дополнительный аргумент содержит список правил, учитывающих предысторию.

Для учета случая, когда все правила для ответа исчерпаны, необходимо дополнительное предложение.

Трудности возникают, когда приходится объяснять использования правила *A* в процессе доказательства *not(A).* В этом случае предыстория вывода

теряется, и аргумент *Rules* в третьем предложении *respond* становится неконкретизованной переменной.

```
% программа 7
:- multifile system/1, askable/1.
:- dynamic untrue/1.

% дополнительный аргумент - список содержащий текущее
% родительское  правило и правила из предыстории
 solve(true,_):-!.
 solve((A,B),Rules):-!,
      solve(A,Rules),
      solve(B,Rules).
 solve(not(A),_):-!,
      not(solve(A,_)).
 solve(A,Rules):-
      not system(A),
      clause(A,B),
      solve(B,[rule(A,B)|Rules]).
 solve(A,_):-
      system(A),
      A.
solve(A,Rules):-
     askable(A),
     not known(A),
     ask(A,Answer),
     respond(Answer,A,Rules).

system(is(_,_)).
system(_=_).
system(_<_).
system(_>_).
system(write(_)).
system(nl).

ask(A,Answer):-
    display_query(A),
    read(Answer).

% дополнительный аргумент - список содержащий текущее
% родительское  правило и правила из предыстории

respond(yes,A,_):-
    assert(A).
respond(no,A,_):-
    assert(untrue(A)),fail.
```

```
% три дополнительных правила для respond

respond(why,A,Rules):-
   var(Rules),!,
   write('  хочу использовать ложность '),
   write(A),nl,
   ask(A,Answer),
   respond(Answer,A,[]).
respond(why,A,[Rule|Rules]):-
   write('  хочу воспользоваться правилом:'),
   display_rule(Rule),
   ask(A,Answer),
   respond(Answer,A,Rules).
respond(why,A,[]):-
   write(' <== возможности объяснения исчерпаны '),nl,
   ask(A,Answer),
   respond(Answer,A,[]).
known(A):-A,!.
known(A):-
   untrue(A).

display_query(A):-
   write(A),
   write('?').

display_rule(rule(A,B)):-
   nl,write('Если '),
   write_conjunction(B),
   write('    то '),
   write(A),nl.

write_conjunction((A,B)):-
   !,write(A),write(' и '),
   write_conjunction(B).
write_conjunction(A):-
   write(A),nl.
```

Вторым аргументом предиката *solve(Goal,Rules)* в программе 7 является список правил, используемых для редукции предковых вершин цели *Goal* в текущем дереве доказательства. Список правил обновляется с помощью предиката *solve* в процессе редукции цели. Для представления правила выбрана структура *rule(A,B)*. Единственным предикатом, на который влияет выбор представления правила, является предикат *display_rule*.

Выясним рекомендации для Боба:

?- solve(hy(bob,Y),[]).

bob имеет симптом мышечные боли? why.

хочу воспользоваться правилом:
Если bob имеет симптом мышечные боли и bob имеет симптом лихорадка
то bob имеет простуда

bob имеет симптом мышечные боли? why.

хочу воспользоваться правилом:
Если bob имеет простуда и not bob имеет уязвимый возраст
то bob - рекомендовано лечь в постель и принять аспирин

bob имеет симптом мышечные боли? why.
хочу воспользоваться правилом:
Если member(лечь в постель и принять аспирин,
[лечь в постель и принять аспирин,вызвать врача])
и bob - рекомендовано лечь в постель и принять аспирин
то hy(bob, лечь в постель и принять аспирин)

bob имеет симптом мышечные боли? why.

<== возможности объяснения исчерпаны

bob имеет симптом мышечные боли? yes.
bob имеет симптом лихорадка? yes.
bob имеет признак моложе 8 лет? no.
bob имеет признак старше 60 лет? no.

Y = 'лечь в постель и принять аспирин' ;

bob имеет симптом насморк? why.

хочу воспользоваться правилом:
Если bob имеет симптом насморк и bob имеет симптом мышечные боли
и not bob имеет симптом лихорадка
то bob имеет грипп

bob имеет симптом насморк? yes.
bob имеет симптом нарывы в горле? why.

хочу воспользоваться правилом:

Если bob имеет симптом лихорадка и bob имеет симптом нарывы в горле
то bob имеет острый фарингит

bob имеет симптом нарывы в горле? yes.
Y = 'вызвать врача' ;
No

Объяснение решения

Ответы на вопросы “почему?” - это простое средство объяснения, описывающее одну локальную цепочку рассуждений. Следующая программа дополняет программу 4 возможностью объяснения, поясняющего полное доказательство решенного вопроса.

```
% программа 8
% how(Goal) - объясняет как была доказана цель Goal

how(Goal):-
      solve(Goal,Proof),
      interpret(Proof).

solve(true,true):-!.
solve((A,B),(ProofA,ProofB)):-!,
     solve(A,ProofA),
     solve(B,ProofB).
solve(not(A),'не доказуемо'(A)):-!,
     not(solve(A,_)).
solve(A,(A:-Proof)):-
     not system(A),
     clause(A,B),
     solve(B,Proof).
solve(A,(A:-true)):-
     system(A),
     A.

system(is(_,_)).
system(_=_).
system(_<_).
system(_>_).
system(write(_)).
system(nl).

interpret((ProofA,ProofB)):-!,
    interpret(ProofA),
    interpret(ProofB).
```

```
interpret((A:-'как было сказано')):-!,
    nl,write(A),write(' <= как было сказано'),nl.
interpret('не доказуемо'(A)):-!,
    nl,write(A),write(' <= не доказуемо'),nl.
interpret(Proof):-
    fact(Proof,Fact),
    nl,write(Fact),write(' - это факт'),nl.
interpret(Proof):-
    rule(Proof,Head,Body,Proof1),
    display_rule(rule(Head,Body)),
    interpret(Proof1).

fact((Fact:-true),Fact).

rule((Goal:-Proof),Goal,Body,Proof):-
    not(Proof=true),
    extract_body(Proof,Body).

extract_body((ProofA,ProofB),(BodyA,BodyB)):-
    !,
    extract_body(ProofA,BodyA),
    extract_body(ProofB,BodyB).
extract_body((Goal:-Proof),Goal).
extract_body('не доказуемо'(B),not(B)).

display_rule(rule(A,B)):-
   nl,write('Если '),
   write_conjunction(B),
   write('    то '),
   write(A),nl.

write_conjunction((A,B)):-
   !,write(A),write(' и '),
   write_conjunction(B).
write_conjunction(A):-
   write(A),nl.
```

Основная идея состоит в интерпретации доказательства цели, где доказательство представлено в метаинтерпретаторе программы 4.

Формирование объяснения:

?- how(member(X,[a,b,c])).

member(a, [a,b,c]) - это факт

X = a ;

Если member(b, [b,c])

то member(b, [a,b,c])

member(b, [b,c]) - это факт

X = b ;

Если member(c, [b,c])
то member(c, [a,b,c])

Если member(c, [c])
то member(c, [b,c])

member(c, [c]) - это факт

X = c ;
No

Определим предикат для нахождения факториала.

```
f(0,1).
f(X,Y):-
  X>0,
  X1 is X-1,
  f(X1,Y1),
  Y is X*Y1.
```

Посмотрим как метаинтерпретатор объяснит, что 2! = 2 :
?- how(f(2,X)).

*Если 2 > 0 и 1 is 2 - 1 и f(1, 1) и 2 is 2 * 1*
то f(2, 2)
2 > 0 - это факт
1 is 2 - 1 - это факт

*Если 1 > 0 и 0 is 1 - 1 и f(0, 1) и 1 is 1 * 1*
то f(1, 1)
1 > 0 - это факт
0 is 1 - 1 - это факт
f(0, 1) - это факт
*1 is 1 * 1 - это факт*

*2 is 2 * 1 - это факт*

X = 2
Yes

Ограничение объяснения

Хотя предыдущее объяснение вычисления факториала 2! исчерпывающе понятно, оно имеет недостаток. Это чрезмерная полнота объяснения, что приводит, даже для очень маленькой базы знаний, к выводу слишком большого объема информации. Экранный текст, произведенный экспертной системой, имеющей сотни правил, становится невразумительным.

Поэтому дополняем программу 8 возможностью указать, что достаточно объяснять при получении решения. Для этого делаем небольшую модификацию программы и используем предикат *explainThis(f(_,_))*, который указывает вывод каких именно предикатов нужно объяснять.

```
% программа 9
:- dynamic explainThis/1.

how(Goal):-
       solve(Goal,Proof),
       interpret(Proof).

solve(true,true):-!.
solve((A,B),(ProofA,ProofB)):-!,
      solve(A,ProofA),
      solve(B,ProofB).
solve(not(A),'не доказуемо'(A)):-!,
      not(solve(A,_)).
solve(A,(A:-Proof)):-
      not system(A),
      clause(A,B),
      solve(B,Proof).
solve(A,(A:-true)):-
      system(A),
      A.

system(is(_,_)).
system(_=_).
system(_<_).
system(_>_).
system(write(_)).
system(nl).

interpret((ProofA,ProofB)):-!,
     interpret(ProofA),
```

```
    interpret(ProofB).
interpret((A:-'как было сказано')):-!,
    nl,write(A),write(' <= как было сказано'),nl.
interpret('не доказуемо'(A)):-!,
    nl,write(A),write(' <= не доказуемо'),nl.
interpret(Proof):-
    fact(Proof,Fact),
    explainFact(Fact).
interpret(Proof):-
    rule(Proof,Head,Body,Proof1),
    explainRule(Head,Body),
    interpret(Proof1).

fact((Fact:-true),Fact).

rule((Goal:-Proof),Goal,Body,Proof):-
    not(Proof=true),
    extract_body(Proof,Body).

extract_body((ProofA,ProofB),(BodyA,BodyB)):-
    !,
    extract_body(ProofA,BodyA),
    extract_body(ProofB,BodyB).
extract_body((Goal:-Proof),Goal).
extract_body('не доказуемо'(B),not(B)).

display_rule(rule(A,B)):-
   nl,write('Если '),
   write_conjunction(B),
   write('    то '),
   write(A),nl.

write_conjunction((A,B)):-
   !,write(A),write(' и '),
   write_conjunction(B).
write_conjunction(A):-
   write(A),nl.

explainRule(Head,Body):-
    explainThis(Head),!,      % об'яснять ли вывод предиката
    nl,write(Head),
    write('  доказано с использованием правила '),
    display_rule(rule(Head,Body)).
explainRule(_,_).

explainFact(Fact):-    % об'яснять ли вывод факта
```

```
    explainThis(Fact),!,
    nl,write(Fact),write(' - это факт'),nl.
explainFact(_).
```

Теперь укажем в программе, что объяснять нужно только вывод предиката факториал f/2.

```
 explainThis(f(_,_)).
```

Получаем объяснение того, что 4! = 24

?- how(f(4,X)).

f(4, 24) доказано с использованием правила
*Если 4 > 0 и 3 is 4 - 1 и f(3, 6) и 24 is 4 * 6*
то f(4, 24)

f(3, 6) доказано с использованием правила
*Если 3 > 0 и 2 is 3 - 1 и f(2, 2) и 6 is 3 * 2*
то f(3, 6)

f(2, 2) доказано с использованием правила
*Если 2 > 0 и 1 is 2 - 1 и f(1, 1) и 2 is 2 * 1*
то f(2, 2)

f(1, 1) доказано с использованием правила
*Если 1 > 0 и 0 is 1 - 1 и f(0, 1) и 1 is 1 * 1*
то f(1, 1)

f(0, 1) - это факт

X = 24
Yes

Правдоподобные рассуждения

Последний пример использования метаинтерпретаторов для экспертных систем связан с применением механизма рассуждений в условиях неопределенности. Причиной для введения такого механизма является наличие неопределенной информации - правил и фактов. Дедуктивный вывод при неопределенных предположениях должен приводить к неопределенным заключениям. Существует несколько способов представления неопределенности в правилах и способов вычисления неопределенности заключений. Основное

требование состоит в том, что в предельном случае, когда все правила определенные, поведение системы повторяло стандартный механизм дедукции.

Остановимся на следующем подходе. С каждым правилом или фактом свяжем коэффициент определенности *c, 0< c <= 1*. Логическая программа с неопределенностями - это множество пар *<Clause,Factor>*, где *Clause* - предложение, а *Factor* - коэффициент определенности. Оболочка экспертной системы, работающая в условиях неопределенности, получена путем непосредственного усовершенствования программы 7. Отношение *solve(Goal,Proof,Certainty)* истинно, когда цель *Goal* удовлетворена с определенностью *Certainty*, а *Proof* - дерево доказательства *Goal*. В программе предполагается, что предложения с коэффициентами определенности представлены с использованием предиката *clause_cf(A,B,CF).*

```
% программа 10
:- multifile system/1, askable/1, clause_cf/3.
:- dynamic untrue/1, askable/1, clause_cf/3.

% введем стартовый предикат
expert(X):-
     solve(X,[],C),nl,
     write('доказано с уверенностью '),
     write(C).

solve(true,_,1):-!.
solve((A,B),Rules,C):-!,
     solve(A,Rules,C1),
     solve(B,Rules,C2),
     C is min(C1,C2).
solve(not(A),_,1):-!,
     not(solve(A,_,_)).
solve(A,Rules,C):-
     not system(A),
     clause(A,B),
     solve(B,[rule(A,B)|Rules],C).
solve(A,Rules,C):-
   not system(A),
   clause_cf(A,B,C1),
   solve(B,[rule(A,B)|Rules],C2),
   C is C1*C2.
solve(A,_,1):-
     system(A),
     A.
solve(A,Rules,C):-
    askable(A),
    not known(A),
```

```
     ask(A,Answer),
     respond(Answer,A,Rules,C).

system(is(_,_)).
system(_=_).
system(_<_).
system(_>_).
system(write(_)).
system(nl).

ask(A,Answer):-
    display_query(A),
    read(Answer).
respond(yes,A,_,1):-
    assert(A).
respond(no,A,_,0):-
    assert(untrue(A)),fail.
respond(why,A,Rules,C):-
    var(Rules),!,
    write('  хочу использовать ложность '),
    write(A),nl,
    ask(A,Answer),
    respond(Answer,A,[],C).
respond(why,A,[Rule|Rules],C):-
    write('  хочу воспользоваться правилом:'),
    display_rule(Rule),
    ask(A,Answer),
    respond(Answer,A,Rules,C).
respond(why,A,[],C):-
    write(' <== возможности объяснения исчерпаны '),nl,
    ask(A,Answer),
    respond(Answer,A,[],C).

% добавляем правила для respond
respond(C,A,_,C):-
    number(C),C>0,
    assert(clause_cf(A,true,C)).
respond(0,A,_,0):-
    assert(untrue(A)),fail.

% добавляем правило для known
known(A):-
    clause_cf(A,true,_),!.
known(A):-A,!.
known(A):-
    untrue(A).
```

```
display_query(A):-
    write(A),
    write('?').

display_rule(rule(A,B)):-
    nl,write('Если '),
    write_conjunction(B),
    write('    то '),
    write(A),nl.

write_conjunction((A,B)):-
    !,write(A),write(' и '),
    write_conjunction(B).
write_conjunction(A):-
    write(A),nl.
```

База знаний “Болезни” в отдельном файле:

```
:- op(100,xfx,['- рекомендовано','имеет','имеет симптом',
            'имеет признак']).
:-dynamic 'имеет симптом'/2, 'имеет признак'/2.

clause_cf(
X '- рекомендовано' 'лечь в постель и принять аспирин',
     (X 'имеет' 'простуда',
     not( X 'имеет' 'уязвимый возраст')),
0.9).

X '- рекомендовано' 'вызвать врача':-
     X 'имеет' 'простуда',
     X 'имеет' 'уязвимый возраст'.
X '- рекомендовано' 'вызвать врача':-
     X 'имеет' 'острый фарингит'.

clause_cf(
X '- рекомендовано' 'лечь в постель и принять аспирин',
       X 'имеет' 'грипп',
0.7).

clause_cf(
X 'имеет' 'простуда',
     (X 'имеет симптом' 'мышечные боли',
     X 'имеет симптом' 'лихорадка'),
0.8).
```

```
X 'имеет' 'уязвимый возраст':-
    X 'имеет признак' 'моложе 8 лет'.

X 'имеет' 'уязвимый возраст':-
    X 'имеет признак' 'старше 60 лет'.

clause_cf(
X 'имеет' 'острый фарингит',
    (X 'имеет симптом' 'лихорадка',
    X 'имеет симптом' 'нарывы в горле'),
0.9).

clause_cf(
X 'имеет' 'грипп',
    (X 'имеет симптом' 'насморк',
    X 'имеет симптом' 'мышечные боли',
    not(X 'имеет симптом' 'лихорадка')),
0.8).

askable(_ 'имеет симптом' _).
askable(_ 'имеет признак' _).
hy(X,Y):-    member(Y,['лечь в постель и принять аспирин',
                       'вызвать врача']),  X '- рекомендовано' Y.

system(member(_,_)).
```

Протокол работы:

```
?- expert(hy(bob,Y)).
bob имеет симптом мышечные боли? 0.9.
bob имеет симптом лихорадка? 0.8.
bob имеет признак моложе 8 лет? no.
bob имеет признак старше 60 лет? 0.

доказано с уверенностью 0.576000
Y = 'лечь в постель и принять аспирин' ;
bob имеет симптом насморк? yes.
bob имеет симптом нарывы в горле? 1.

доказано с уверенностью 0.720000
Y = 'вызвать врача' ;
No

?- expert(hy(cat,Y)).
cat имеет симптом мышечные боли? no.
```

cat имеет симптом насморк? 0.7.
cat имеет симптом лихорадка? 0.1.
cat имеет симптом нарывы в горле? 0.5.

доказано с уверенностью 0.090000
Y = 'вызвать врача' ;
No

То, что по силам читателю, предоставь ему самому.

Людвиг Витгенштейн

3.3. Задание на экспертную систему

Несложное комбинирование программ 9 и 10 позволит создать полноценную оболочку экспертной системы. Эта оболочка должна удовлетворять следующим требованиям:

1) если база знаний является просто программой на “почти” полном Прологе, то оболочка ведет себя как метаинтерпретатор этой программы с объяснением как было получено доказательство;

2) если предполагается, что часть информации будет запрашиваться у пользователя, то оболочка ведет активный диалог и может объяснить почему она спрашивает пользователя;

3) применяется механизм рассуждения в условиях неопределенности.

Придумайте свою базу знаний с неопределенностями. Эта база знаний должна включать также правила для предикатов *askable* и *explainThis*. В этой базе знаний необходим целевой предикат, с помощью которого проверяется достоверность различных гипотез (в базе знаний “Болезни” таким предикатом является *hy(_,_)*).

Обеспечьте, чтобы оболочка получала все возможные решения, а не останавливалась на одном.

В качестве образца приведем возможную базу знаний для идентификации животных:

Если X имеет шерсть, то X - млекопитающее.

Если X кормит детенышей молоком, то X - млекопитающее.

Если X имеет перья, то X - птица.
Если X летает и откладывает яйца, то X - птица.

Если X млекопитающее и ест мясо, то X - хищник.

Если X млекопитающее и имеет острые зубы, когти и глаза, направленные вперед, то X - хищник.

Если X хищник и имеет рыжевато-коричневый цвет и темные пятна, то X - гепард.

Если X хищник и имеет рыжевато-коричневый цвет и черные полосы, то X - тигр.

Если X птица и не летает и плавает, то X - пингвин.

Если X птица и летает очень хорошо, то X - альбатрос.

Трудности классификации хорошо иллюстрирует отрывок из рассказа “Аналитический язык Джона Уилкинса” Хорхе Луиса Борхеса.

“Это ... напоминает классификацию, которую доктор Франц Кун приписывает одной китайской энциклопедии под названием “Небесная империя благодетельных знаний”.

На ее древних страницах написано, что животные делятся на
а) принадлежащих Императору, б) набальзамированных, в)прирученных, г) сосунков, д) сирен, е) сказочных, ж) отдельных собак, з) включенных в эту классификацию, и) бегающих как сумасшедшие, к) бесчисленных, л) нарисованных тончайшей кистью из верблюжьей шерсти, м) прочих, н) разбивших цветочную вазу, о) похожих издали на мух.”

Вот так может выглядеть протокол работы экспертной системы в окончательном варианте:

?- expert.
Введите цель | : hy(cat,X).

cat имеет симптом мышечные боли? no.
cat имеет симптом насморк? why.
хочу воспользоваться правилом:
Если cat имеет симптом насморк и cat имеет симптом мышечные боли и not cat имеет симптом лихорадка
то cat имеет грипп
cat имеет симптом насморк? 0.7.
cat имеет симптом лихорадка? why.
хочу воспользоваться правилом:
Если cat имеет симптом лихорадка и cat имеет симптом нарывы в горле
то cat имеет острый фарингит
cat имеет симптом лихорадка? 0.1.

cat имеет симптом нарывы в горле? 0.5.

доказано hy(cat, вызвать врача)
с уверенностью 0.090000
Объясняем вывод:

cat - рекомендовано вызвать врача доказано с использованием правила
Если cat имеет острый фарингит
то cat - рекомендовано вызвать врача

cat имеет острый фарингит доказано с использованием правила
Если cat имеет симптом лихорадка и cat имеет симптом нарывы в горле
то cat имеет острый фарингит

cat имеет симптом лихорадка <= как было сказано

cat имеет симптом нарывы в горле <= как было сказано
No

4. КУРСОВЫЕ РАБОТЫ

Выполнение курсовой работы по искусственному интеллекту требует решения одной из нижеперечисленных задач и, как результат, создания программы на Прологе и написания пояснительной записки к работе. Созданную программу и пояснительную записку (*в электронном виде*) студент пересылает по электронной почте диспетчеру центра дистантного обучения, который в свою очередь пересылает их лектору. Лектор проверяет программу и пояснительную записку и при правильном выполнении работы студент получает подтверждение о том, что они зачтены. Если программа или записка составлена неправильно, студент получает от лектора текстовый файл, в котором содержится описание ошибок программы и недостатков пояснительной записки. За выполненную курсовую работу студент получает дифференцированный зачет.

Варианты тем для курсовых работ

Варианты курсовых работ разные по трудности. Это будет учитываться при выставлении оценки за курсовую работу.

1. Упрощение электрических цепей

Постановка задачи.

Цепь состоит из компонентов только трех видов: резисторов, емкостей и индуктивностей. Преобразовать цепь в более простую, используя упрощающие правила при параллельном и последовательном соединении компонент одного вида. Треугольники преобразовывать в звезды. Используйте следующее представление предметной области. Структуры r(Номинал), l(Номинал) и c(Номинал) изображают соответственно резистор, индуктивность и емкость с номиналами. Вся цепь представляется в базе данных фактами вида

comp(<метка элемента>,
<элемент: резистор, индуктивность или емкость>,
<список узлов элемента>).

Упрощение цепи сводится к изменению базы данных.

2. Программа для алгебраических вычислений

Объекты, с которыми вы будете работать, - это многочлены от одной переменной, представленных в символьном виде с вещественными коэффициентами. Многочлены должны изображаться как арифметические выраже-

ния, так умножение изображается знаком ‘*’, а возведение в степень - знаком ‘^’.

Для манипуляций с многочленами нужны некоторые предикаты, чтобы пользователь мог получать ответы на вопросы, на которые не удается ответить с помощью традиционных языков программирования. Для этого вам понадобится обозначать многочлены идентификаторами, и хранить в базе данных пару (имя многочлена, сам многочлен). Предикаты выполняют некоторые операции над своими операндами и помещают результат в качестве значения некоторого имени многочлена.

Список предикатов.

1. Ввести многочлен и записать его под некоторым именем.

2. Образовать алгебраическую сумму (разность, произведение) двух многочленов и записать полученный многочлен под некоторым именем.

3. Возвести данный многочлен в целую степень и результат записать под некоторым именем.

4. Заменить каждое вхождение переменной в многочлене на данный многочлен и результат записать под некоторым именем.

5. Вычислить производную многочлена по переменной и результат записать под некоторым именем.

6. Напечатать данный многочлен.

Многочлен представлять в виде суммы членов, включающих только операции умножения и возведения в степень. Каждый такой одночлен (моном) состоит из числового коэффициента (первый сомножитель), и переменной в соответствующей степени. Следует упрощать запись одночлена, когда он является только числовым или коэффициент при переменной равен плюс-минус единице или степень равна единице. Следует также приводить подобные члены, т. е. объединять одночлены, имеющие одинаковые степени у переменной, с соответствующим изменением коэффициентов.

Литература (необязательная)

1. Уэзерелл Ч. Этюды для программистов: Пер. с англ. -М.: Мир,1982, стр. 114-120.

2. Стерлинг Л., Шапиро Э. Искусство программирования на языке ПРОЛОГ, М.: Мир, стр. 57-61.

Указания. Некоторые фрагменты программы:

```
% Полиномы хранятся в виде фактов
% pol(+Name, -<список одночленов>).

% Ввод полинома: enter(+Name)

enter(X):-
   write('Введите полином с именем '),write_ln(X),
   enter(X,[],_).
```

```
enter(X,S,P):-
    write_ln('Введите очередной одночлен или end'),
    read(T),
    ((T=end,place(X,S,P));
     (T \= end,
      enter(X,[T|S],P))).

% привести подобные во введенном полиноме, упорядочить члены
% загрузить в базу данных

place(X,S,P):-
    merge(S,[],P),
    retractall(pol(X,_)),
    assert(pol(X,P)).

% сложение полиномов, представленных списками одночленов
merge([],L,L).
merge([X|L1],L2,L):-
    insert(X,L2,L3),
    merge(L1,L3,L).

insert(X,[],[X]).
insert(X,[Y|T],[Z|T]):-
    equal(X,Y,Z),Z \= 0,!.
insert(X,[Y|T],T):-
    equal(X,Y,0),!.
insert(X,[Y|T],[X,Y|T]):-
    low(X,Y).
insert(X,[Y|T],[Y|T1]):-
    not(low(X,Y)),
    insert(X,T,T1).

/* equal(X,Y,Z)  - из двух мономов X, Y при приведении подобных полу-
чаем Z; low(X,Y) -  моном X предшествует моному Y. */

% приведение полинома к более "читабельному" виду
canon([],0).
canon([X],X).
canon([X,Y],Y+X).
canon([X,Y|T],Z+Y+X):-
    length(T,M),M>0,
    canon(T,Z).

% показать полином
show(X):-
   pol(X,P),
```

```
   canon(P,P1),
   write('Полином с именем '),write_ln(X),write_ln(P1).

% сложение полиномов
plus_pol(X,Y,Z):-
   pol(X,P1),
   pol(Y,P2),
   merge(P1,P2,P),
   retractall(pol(Z,_)),
   assert(pol(Z,P)),
   show(Z).
```

Протокол:

```
?- enter(a).
Введите полином с именем a
Введите очередной одночлен или end
-8*x^5.
Введите очередной одночлен или end
9.
Введите очередной одночлен или end
10*x^5.
Введите очередной одночлен или end
x^3.
Введите очередной одночлен или end
-7*x^3.
Введите очередной одночлен или end
end.
Yes

?- show(a).
Полином с именем a
2 * x ^ 5 + -6 * x ^ 3 + 9
Yes

?-plus_pol(a,a,b).
Полином с именем b
4 * x ^ 5 + -12 * x ^ 3 + 18
Yes
```

3. Игра "Суммируйте до 20"

Два игрока по очереди называют какое-либо число в интервале от 1 до 3 включительно. Названные числа суммируются, выигрывает тот игрок, сумма

чисел после хода которого равна 20. Напишите программу на SWI-Prologe, выигрывающую данную игру на стороне компьютера.
Указания. Используйте запоминающие функции (см. "Курс лекций по логическому программированию"). Напишите программу в таком виде, чтобы ее можно было легко модернизировать для решения более общей задачи. Более общая задача: перед началом игры компьютер спрашивает "Какой список допустимых ходов (какие числа можно называть) и какая предельная (выигрышная) сумма?"

4. Упрощение арифметических выражений

Назовем арифметическим выражением терм, при конструировании которого используются только атомы, числа, скобки и знаки арифметических операций. Напишите программу для упрощения арифметических выражений на SWI-Prologe.

В целом, задача упрощения выражений является достаточно сложной и в каком-то смысле неконкретизованной, т. к. единого верного решения для этой задачи нет. Если арифметическое выражение имеет несколько вариантов более простого представления, то какой из них выбрать в качестве решения? Это зависит от того, для каких целей нам необходимо упрощение.

Задачу упрощения выражения поставим следующим образом. Необходимо найти эквивалентное выражение, форма записи которого является более короткой, чем форма записи исходного выражения. Для упрощения выражений используйте различные рекурсивные "правила переписывания", каждое из которых "упрощает" какое-нибудь подвыражение в исходном выражении. Правила переписывания должны соответствовать обычным математическим преобразованиям, как-то: приведению подобных и т. п., и представляются правилами Пролога (см. программу "дифференцирование выражений" из курса лекций).

Ваша программа должна быть некоторым компромиссом между желательной простотой написания и той сложностью, которой от нее требует поставленная задача.

Литература (не обязательная): Лорьер Ж.-Л. Системы искусственного интеллекта. - М., Мир, 1991.- С. 129-131, 471.

5. Игра "Выдающийся ум" ("Быки и коровы")

Вы должны написать программу, которая разгадывает секретный код в игре "Выдающийся ум" (игра имеет второе название - "Быки и коровы"). В эту игру играют два игрока. Игрок А выбирает секретный код, представляющий собой последовательность из N десятичных цифр (обычно начинающие игроки выбирают N равным 4, опытные - 5). Игрок В пытается угадать задуманный код и спрашивает игрока А о числе "быков" (число "быков" - количе-

ство совпадающих цифр в одинаковых позициях предполагаемого и задуманного кодов) и числе "коров" (число "коров" - количество совпадающих цифр, входящих в предполагаемый и задуманный код, но находящийся в разных позициях). Код угадан, если число быков равно N.

Существует очень простой алгоритм этой игры: вводится некоторый порядок на множестве допустимых правильных предположений; выдвижение очередных предположений учитывает накопленную к этому моменту информацию, и так до тех пор, пока секретный код не будет раскрыт.

Вместо формального определения алгоритма игры обратимся к интуиции читателя: предположения считаются удачными, если ответы на вопросы угадывающего совпадают с ответами, которые были бы даны при разгадке кода.

Если предлагаемый алгоритм запрограммировать, предполагая, что все цифры в коде должны быть различны, то алгоритм будет "играть" в силу опытных игроков: для раскрытия кода из четырех различных цифр ему требуется в среднем 4-6 попыток, наблюдавшийся максимум - 8 попыток. Для вашего задания коды могут быть произвольны, даже с повторяющимися цифрами. Поэтому число попыток приблизительно удваивается.

Человеку стратегию, используемую в алгоритме, применить нелегко, поскольку она требует значительной счетной работы. С другой стороны, управляющая структура Пролога - недетерминированный выбор, моделируемый поиск с возвратами, - представляется идеальной для реализации этого алгоритма.

Приведем основную часть программы.

```
'выдающийся ум'(Cod):-
        'чистка',
        'предположение'(Cod),
        'проверка'(Cod),
        сообщение.

'предположение'([X1,X2,X3,X4]):-
       'выбор'([X1,X2,X3,X4], [1,2,3,4,5,6,7,8,9,0]).

% проверка предложенной гипотезы

'проверка'(Cod):-
       not 'противоречивое'(Cod),
       'вопрос'(Cod).

'противоречивое'(Cod):-
       'запрос'(OldCod,B,C),                    % B -быки, C - коровы
       not 'соответствуют быки и коровы'(OldCod,Cod,B,C).
```

```
'соответствуют быки и коровы'(OldCod,Cod,B,C):-
        'точное совпадение'(OldCod,Cod,N1),
        B=:=N1,                                 % правильное число быков
        'общие члены'(OldCod,Cod,N2),
        C=:=N2-B.                               % правильное число коров

% оценка гипотезы

вопрос(Cod):-
        repeat,write('Гипотеза '),write(Cod),nl,
        write('Сколько быков?'),nl,
        read(B),nl,
        write('Сколько коров?'),nl,
        read(C),nl,
        'допустимо'(B,C),!,
        assert('запрос'(Cod,B,C)),
        B=:=4.
```

Спрашивающая процедура 'предположение'(Cod), которая действует как генератор, использует процедуру выбора 'выбор'([X1,X2,X3,X4], [1,2,3,4,5,6,7,8,9,0]) для выбора списка из четырех десятичных цифр.

Процедура 'проверка'(Cod) испытывает предложенную гипотезу Cod. Сначала проверяется, что Cod не противоречит всем ранее полученным ответам (т. е. непротиворечив с каждым из них), затем задается вопрос о числе быков и коров в предположении Cod. Кроме того, процедура вопрос(Cod) управляет циклом "образовать и проверить", который завершается только тогда, когда число быков равно 4, что является признаком отыскания правильного хода.

Процедура 'вопрос' запоминает предыдущие ответы на вопросы в фактах 'запрос'(Cod,B,C). Предикат 'точное совпадение'(OldCod,Cod,N1) определяет число N1 - количество одинаковых цифр на одних и тех же позициях в Cod и OldCod. Предикат 'общие члены' определяет количество одинаковых цифр в двух предположениях, без учета их позиций. Предикат 'допустимо' проверяет, что количество "быков" и "коров" имеют допустимые значения. Предикат 'чистка' убирает из памяти перед каждой игрой старые факты 'запрос'. И, наконец, предикат 'сообщение' говорит о победе компьютера и сообщает, за сколько ходов, для этого достаточно подсчитать число запомненных фактов 'запрос'.

Вам остается только запрограммировать все недостающие предикаты. Для предиката 'запрос' необходимо еще добавить директиву компилятору

```
:- dynamic 'запрос'/3.
```

6. Нахождение геометрических аналогий

Программа поиска геометрических аналогий была составной частью докторской диссертации Т. Эванса в MIT в середине 1960-х гг. Вы должны применить вариант этой программы на Прологе.

Рассмотрим, в чем заключаются задачи на геометрические аналогии, которые обычно используют для испытания умственных способностей. В каждой задаче такого рода предлагается несколько фигур. На рис. 1 представлен вариант простой задачи на отыскание геометрических аналогий. Часть фигур (на рисунке - нижний ряд) - возможные ответы. Используя фигуры A, B и C и фигуры, представляющие собой варианты решений, следует дать ответ на вопрос: "Фигура A относится к фигуре B, как фигура C относится к …". К какой фигуре? Ответ надо выбрать из трех вариантов решений в нижнем ряду.

Интуитивные представления позволяют записать следующий алгоритм для решения этой задачи (такие понятия, как "найти", "применить" и "правило", здесь не описаны):

- найти правило, определяющее связь фигур A и B;
- применить это правило к фигуре C, чтобы перейти к фигуре X;
- найти среди ответов фигуру X. или ее ближайший эквивалент.

В рассматриваемой задаче (см. рис. 1) связь фигуры A с фигурой B определяется обменом местами (с соответствующим изменением масштабов) изображений квадрата и треугольника. "Очевидный" ответ для фигуры C - обмен местами изображений квадрата и круга. Соответствующая фигура имеет в нижнем ряду номер 2.

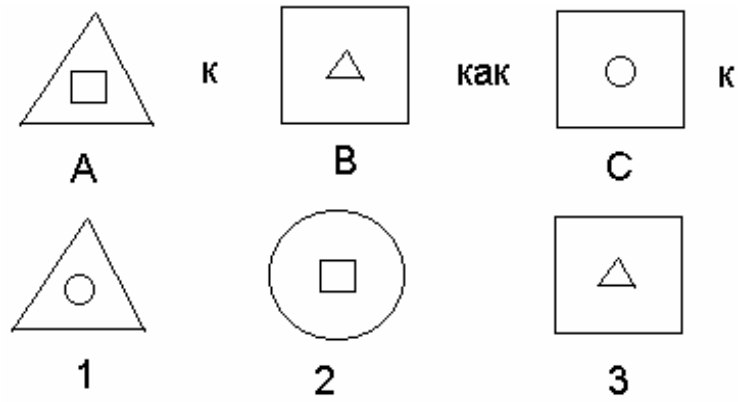


Рис. 1. Задача на геометрические аналогии

Следующая программа предназначена для решения простых задач на геометрические аналогии.

```
analogy(A is_to B,C is_to X,Answers):-
        match(A,B,Rule),
        match(C,X,Rule),
        member(X,Answers).

match(inside(Figure1,Figure2),
      inside(Figure2,Figure1),invert).
match(above(Figure1,Figure2),
      above(Figure2,Figure1),invert).

% предложения и данные для тестирования

test_analogy(Name,X):-
     figures(Name,A,B,C),
     answers(Name,Answers),
     analogy(A is_to B,C is_to X,Answers).

figures(test,inside(square,triangle),
             inside(triangle,square),
             inside(circle,square)).

answers(test,[inside(circle,triangle),
              inside(square,circle),
              inside(triangle,square)]).
```

Ее основным отношением является предикат analogy(Pair1,Pair2,Answers), в котором каждая пара Pair представляется термом X is_to Y. В программе, конечно, должно быть дано определение инфиксному оператору is_to. Элементы пары Pair1 находятся в таком же отношении, как и элементы пары Pair2, а во второй элемент пары Pair2 принадлежит множеству ответов Answers.

Большое значение имеет выбор способа представления фигур в рассматриваемой задаче. Этот выбор оказывает существенное влияние на "интеллектуальность" программы. В программе фигуры представлены термами Пролога. Например, фигура А на рис. 1 - квадрат в треугольнике - представляется термом inside(square,triangle).

Связь между фигурами отыскивается с помощью предиката match(A,B,Rule). Это отношение истинно, если операция Rule сопоставляет А и В. Для решения рассматриваемой задачи применяется операция invert, которая обеспечивает обмен местами ее аргументов.

Предикат match в программе используется двояко. В первом случае с его помощью производится сопоставление двух данных фигур. Во втором случае для заданных операции и фигуры подбирается вторая фигура. Однако эти детали несущественны для недетерминированного поведения программы.

Наконец, с помощью предиката member проверяется, принадлежит ли данная фигура множеству ответов.

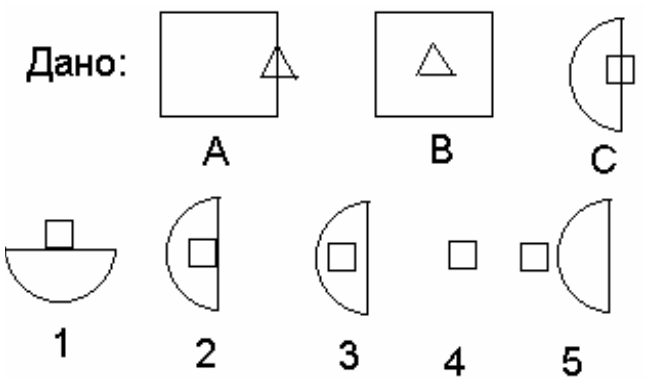


Рис. 2. Первый тест

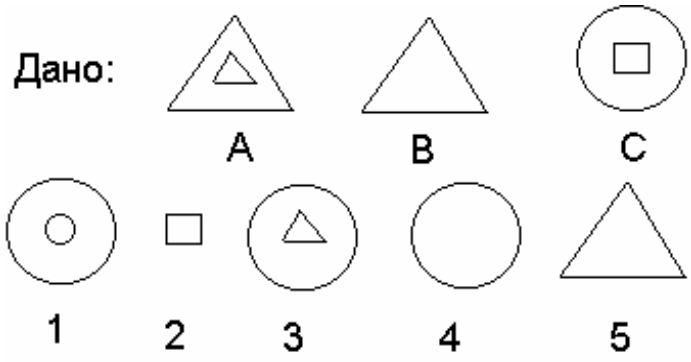


Рис. 3. Второй тест

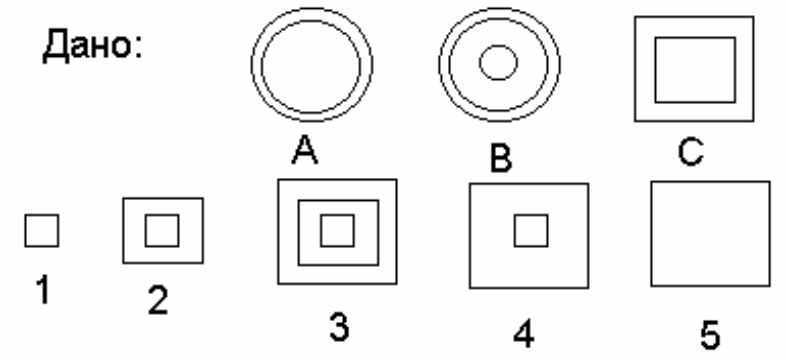


Рис. 4. Третий тест

В курсовой работе вы должны изменить описанную выше программу так, чтобы с ее помощью можно было решить три задачи на геометрические аналогии, представленные на рис. 2-4.

7. Логическая головоломка "зебра"

Напишите программу для решения следующей логической головоломки. В пяти домах, окрашенных в разные цвета, обитают мужчины разных национальностей. Они держат разных животных, предпочитают разные напитки и курят сигареты разных марок. Известно, что:

1. Англичанин живет в красном доме.
2. У испанца есть собака.
3. Кофе пьют в зеленом доме.
4. Украинец пьет чай.
5. Зеленый дом - первый по правую руку от дома цвета слоновой кости.
6. Курильщик "Уинстона" держит улиток.
7. Сигареты "Кул" курят в желтом доме.
8. Молоко пьют в среднем доме.
9. Норвежец живет в крайнем слева доме.
10. Мужчина, курящий "Честерфилд", живет в доме, соседнем с домом мужчины, у которого есть лиса.
11. Сигареты "Кул" курят в доме, соседнем с домом, где имеется лошадь.
12. Мужчина, предпочитающий "Лаки страйк", пьет апельсиновый сок.
13. Японец курит сигареты "Парламент".
14. Норвежец живет в доме рядом с голубым домом.

Вопросы: "У кого есть зебра?", "Кто пьет воду?".

Для решения этой задачи полезно использовать метод "образовать и проверить". Опишем его.

Метод "образовать и проверить" - общий прием, используемый при проектировании алгоритмов и программ. Суть его состоит в том, что один процесс или программа генерируют множество предполагаемых решений задачи, а другой процесс или программа проверяет эти предполагаемые решения, пытаясь найти те из них, которые действительно являются решениями задачи.

Обычно программы, реализующие метод "образовать и проверить", конструировать проще, чем программы, в которых решение находится непосредственно, однако они менее эффективны. Стандартный прием оптимизации программ типа "образовать и проверить" заключается в стремлении погрузить программу проверки в программу генерации предполагаемых решений настолько "глубоко", насколько это возможно. В пределе программа проверки полностью переплетается с программой генерации предполагаемых решений, которая начинает порождать только корректные решения.

Используя вычислительную модель Пролога, легко создавать логические программы, реализующие метод "образовать и проверить". Обычно такие программы содержат конъюнкцию двух целей, одна из которых действует

как генератор предполагаемых решений, а вторая проверяет, являются ли эти решения приемлемыми:

find(X):-generate(X), test(X).

Эта Пролог-программа действует подобно обычной процедурной программе, выполняющей генерацию вариантов и их проверку. Если при решении вопроса *find(X)?* успешно выполняется цель *generate(X)* с выдачей некоторого *X*, то затем выполняется проверка *test(X).* Если проверка завершается отказом, то производится возвращение к цели *generate(X)*, с помощью которой генерируется следующий элемент. Процесс продолжается итерационно до тех пор, пока при успешной проверке не будет найдено решение с характерными свойствами или генератор не исчерпает все альтернативные решения.

Однако программисту нет необходимости интересоваться циклом "образовать и проверить". Он может рассматривать этот метод более абстрактно, как пример недетерминированного программирования. В этой недетерминированной программе генератор вносит предположение о некотором элементе из области возможных решений, а затем просто проверяется, корректно ли данное предположение генератора.

В качестве генератора обычно используется предикат *member*, порождающий множество решений. На вопрос *member(X,[a,b,c])?* будут даны в требуемой последовательности решения *X=a, X=b* и *X=c.* Таким образом, предикат *member* можно использовать в программах, реализующих метод "образовать и проверить" для недетерминированного выбора корректного элемента из некоторого списка.

Метод "образовать и проверить" можно применить для решения логических головоломок. Логическая головоломка состоит из нескольких фактов относительно небольшого числа объектов, которые имеют различные атрибуты. Минимальное число фактов относительно объектов и атрибутов связано с желанием выдать единственный вариант назначения атрибутов объектам.

Метод решения логических головоломок опишем на следующем примере.

Три друга заняли первое, второе и третье места в соревнованиях универсиады. Друзья разной национальности, зовут их по-разному и любят они разные виды спорта.

Майкл предпочитает баскетбол и играет лучше, чем американец. Израильтянин Саймон играет лучше теннисиста. Игрок в крикет занял первое место.

Кто является австралийцем? Каким спортом занимается Ричард?

Подобные логические головоломки изящно решаются посредством конкретизации значений подходящей структуры данных и выделения значения, приводящего к решению. Каждый ключ к разгадке преобразуется в факт относительно структуры данных. Это может быть сделано с использованием абстракции данных до определения точной формы структуры данных. Про-

анализируем первый ключ к разгадке: "Майкл предпочитает баскетбол и играет лучше, чем американец". Очевидно, речь идет о двух разных людях. Одного зовут Майклом, и он занимается баскетболом, в то время как второй - американец. Кроме того, Майкл лучше играет в баскетбол, чем американец. Предположим, что Friends - структура данных, подлежащая конкретизации, тогда наш ключ может быть выражен следующей конъюнкцией целей:

'играет лучше'(Man1,Man2,Friends), 'имя'(Man1,'майкл'),
'спорт'(Man1,'баскетбол'), 'национальность'(Man2,'американец').

Аналогично второй ключ можно представить конъюнкцией целей:

'играет лучше'(Man1,Man2,Friends), 'имя'(Man1,'саймон'),
'национальность'(Man1,'израильтянин'), 'спорт'(Man2,'теннис').

Наконец, третий ключ к разгадке выразится следующим образом:

'первый'(Friends,Man),'спорт'(Man,'крикет').

Базовая программа для решения головоломок следующая.

```
solve_puzle(puzle(Qeys,Questions,Solve),Solve):-
        solve(Qeys),
        solve(Questions).
solve([Qey|Qeys]):-
    Qey,solve(Qeys).
solve([]).
```

Вычислению подлежит отношение solve_puzle(Puzle,Solve), где Solve является решением головоломки Puzle. Головоломка представляется структурой puzle(Qeys,Questions,Solve), где структура данных, подлежащая конкретизации, представляется ключами и вопросами, а получаемые значения определяются аргументом Solve.

Программа solve_puzle тривиальна. Все, что она делает, состоит в последовательном решении каждого ключа и вопроса, которые представляются как цели Пролога и выполняются с использованием метапеременных.

Ключи и вопросы для нашего примера даны в оставшейся части программы:

```
test_puzle(Name,puzle(Qeys,Questions,Solve)):-
    structure(Name,Structure),
    qeys(Name,Structure,Qeys),
    questions(Name,Structure,Questions,Solve).

structure(test,[friend(N1,C1,S1),friend(N2,C2,S2),friend(N3,C3,S3)]).

qeys(test,Friends,
     [('играет лучше'(X1,Y1,Friends), 'имя'(X1,'майкл'), 'спорт'(X1,'баскетбол'),
        'национальность'(Y1,'американец')),
      ('играет лучше'(X2,Y2,Friends), 'имя'(X2,'саймон'),
```

```
        'национальность'(X2,'израильтянин'), 'спорт'(Y2,'теннис')),
      ('первый'(Friend,X), 'спорт'(X,'крикет'))
     ]).

questions(test, Friends,
     [member(Q1,Friends), 'имя'(Q1,Name),
      'национальность'(Q1,'австралиец'), member(Q2,Friends),
      'имя'(Q2,'ричард'), 'спорт'(Q2,Sport)],
     [['Имя австралийца -',Name],['Ричард играет в ',Sport]]).

'играет лучше'(A,B,[A,B,C]).
'играет лучше'(A,C,[A,B,C]).
'играет лучше'(B,C,[A,B,C]).
'имя'(friend(A,B,C),A).
'национальность'(friend(A,B,C),B).
'спорт'(friend(A,B,C),C).
'первый'([X|Xs],X).
find(Y):-
   test_puzle(test,X),
   solve_puzle(X,Y).
```

Каждый человек имеет три атрибута и может быть представлен структурой fiend(Name,Country,Sport). Есть три друга распределение мест, которых в итоге соревнования имеет существенное значение. Поэтому в качестве структуры данных для решения задачи упорядоченную последовательность из трех элементов, т. е. список:

[friend(N1,C1,S1),friend(N2,C2,S2),friend(N3,C3,S3)].

Запуск предиката

?- find(X).

выдает решение

X = [['Имя австралийца -', майкл], ['Ричард играет в ', теннис]].

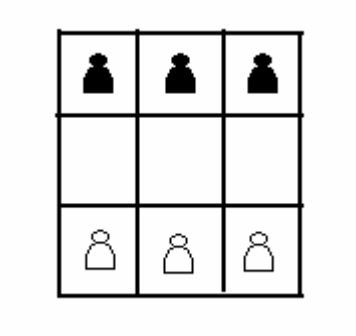
8. Самообучающаяся программа

Постановка задачи.

Курсовая работа посвящена созданию программы для игры с компьютером в качестве противника. Программу нужно сделать самообучающейся: первоначально, компьютер будет вам проигрывать, но постепенно он должен накапливать опыт и, в конце концов, станет вам достойным противником.

1. Описание игры

В игру "шесть пешек" играют на доске размером 3 на 3 клетки. Каждый из игроков имеет по три пешки. Начальная позиция показана на следующем рисунке.



Ходы разрешается делать лишь двух типов:

1) пешка может передвинуться на одну клетку вперед, если эта клетка пуста;

2) пешка может взять пешку другого цвета, стоящую справа или слева на соседней клетке по диагонали, и остаться на освободившейся клетке.

Взятая пешка снимается с доски. Ходы пешек, как видно из этих правил, в основном совпадают с ходами пешек в обычных шахматах. Однако в отличие от шахмат нашим пешкам не разрешается делать двойной ход в начале партии, брать пешку противника на проходе и превращаться в какие-либо другие фигуры того же цвета.

Партия считается выигранной в следующих трех случаях:

1) когда одну из пешек удается провести в третий ряд;

2) когда взяты все пешки противника;

3) когда противник не может сделать очередного хода.

Играющие делают ходы по очереди, передвигая каждый раз по одной пешке. Очевидно, что закончится вничью игра не может; далеко не так очевидно, какой из игроков имеет преимущество: делающий второй ход или тот, кто начинает игру.

Теория игры в шесть пешек совершенно тривиальна, тем не менее, я убедительно прошу читателя не проводить никакого анализа. Построив программу и постигнув все тонкости "шестипешия" в процессе обучения её игре, вы получите гораздо больше удовольствия.

2. Алгоритм самообучающейся программы

Первоначально, все возможные ходы в любой позиции равновероятны. Начиная с исходной позиции, вы делаете первый ход, и после этого программа может сделать любой разрешенный ход. Ход программа делает случайным образом. Никакой определенной стратегии нет и она еще ничего не умеет.

Обучение происходит следующим образом. Любая партия закончится после третьего хода программы. Если партию программа выиграла, то стратегия не меняется. Если программа проиграет, то необходимо понизить оценку (“вероятность”) тех ходов, которые сделала программа. Более подробно, пусть S1, S2, S3, S4, S5, S6, S7 - список последовательных позиций в партии, которую проиграла программа (для упрощения изложения, возможно, пришлось изменить нумерацию позиций). Программа делала ходы в пози-

циях S2, S4 и S6. Таким образом, для обучения программы необходимо уменьшить оценку S6 и, возможно, вероятности S2, S4.

Можно придумать и другую систему обучения. Например, можно не только наказывать программу после проигрыша, уменьшая вероятности плохих ходов, но и поощрять после обед, увеличивая вероятности хороших ходов.

Для быстрейшего самообучения в программу следует заложить и второго партнера, играющего по той же или другой системе, так чтобы машина играла сама с собой.

Базовая программа на Прологе игры с самообучением :

```
game:-
  'инициализировать'(Position,Gamer),
  'отобразить игру'(Position,Gamer),
  game(Position,Gamer,[],Rezalt).

game(Position,Gamer,History,Rezalt):-
        /* History - список пар (Позиция/Кто ходил перед этим) */
    'игра окончена'(Position,Gamer,Rezalt),!,
    'объявить'(Rezalt),
    'самообучение'(Position,Gamer,History,Rezalt).

game(Position,Gamer,History,Rezalt):-
    'выбрать ход'(Position,Gamer,Go),!,
    'ходить'(Go,Position,PositionNext),
    'другой игрок'(Gamer,Gamer1),
    'отобразить игру'(PositionNext,Gamer1),
    game(PositionNext,Gamer1,[PositionNext/Gamer|History],Rezalt).
```

Возможно, следующие предикаты будут полезны.

Позиция представляется в виде пары
(Список белых пешек - Список черных пешек),
координаты пешек - целые числа от 1 до 9.
Исходная позиция - [1,2,3]-[7,8,9].
value(Позиция,С_точки_зрения,Оценка) - предикат базы знаний

```
'самообучение'(_,Gamer,_,Gamer).

 'самообучение'(Position,Gamer,History,Rezalt):-
    not(Gamer=Rezalt),
    'другой игрок'(Gamer,Gamer1),
    'уменьшить оценки'([Position/Gamer1|History],Gamer).
```

```
'возможный ход белыми'(W-B,From-Where)
'возможный ход черными'(W-B,From-Where)
'выбрать ход'(Position,white,From-Where)

'выбрать ход'(Position,black,Go):-
    /* находим список всех позиций-преемников данной Position
       и выбираем среди них позицию с большей оценкой */
        'позиции-преемники'(Position,black,Sons),
        'позиция с лучшей оценкой'(Sons,P),
        'ходить'(Go,Position,P).
```

9. Моделирование управления предприятиями

Язык программирования - любой.

Решением совета директоров крупного промышленного концерна вы назначены президентом компании. Компания владеет несколькими фабриками. Каждый месяц компания покупает сырье, обрабатывает его и продает изготовленную продукцию публике, ждущей ее с нетерпением. Вам теперь придется решать, сколько и каких товаров выпускать, стоит ли и когда именно расширять производственные мощности, как финансировать их расширение и как принять скромно-застенчивый вид, отчитываясь о незаконных прибылях. Перед тем как приступить к работе, вы строите модель промышленности в целом, чтобы в частном порядке отработать свою линию поведения. И вот какую игру вы в результате изобрели.

Начальная ситуация

Моделирование ведется с шагом по времени в один месяц. В начале игры каждый игрок (президент компании) получает две обычные фабрики, четыре единицы сырья и материалов (сокращенно ЕСМ), две единицы готовой продукции (сокращенно ЕГП) и 10000 долл. наличными. Игроки занумерованы от 1 до N, и в первом круге игрок 1 - старший. С каждым кругом (т.е. ежемесячно) роль старшего переходит к следующему по порядку номеров игроку, после N-го старшим становится опять первый (так что номер старшего в круге T вычисляется по формуле T mod N +1). На торгах при прочих равных условиях выигрывает самый старший игрок (тот, кто будет старшим в следующем круге).

Ежемесячные операции

Описанные ниже сделки происходят каждый месяц и именно в таком порядке. Если в какой-то момент компания не может выполнить своих фи-

нансовых обязательств, она немедленно объявляется банкротом. Ее имущество пропадает, и она выходит из игры (так что лучше иметь наготове запас наличных). Все денежные расчеты происходят между отдельными игроками и одним общим банком. Невозможна передача денег прямо от игрока к игроку, что исключает сговор, направленный против какой-либо компании. Банк, кроме того, контролирует источники сырья и скупает всю готовую продукцию.

1. Постоянные издержки. Каждый игрок (в порядке убывания старшинства, начиная со старшего) платит 300 долл. за каждую имеющуюся у него ЕСМ, 500 долл. за каждую наличную ЕГП, 1000 долл. за владение каждой обычной фабрикой, 1500 долл. - за владение автоматизированной. Это постоянные ежемесячные издержки каждого игрока, даже если он в этом круге не предпринимает никаких других действий.

2. Определение обстановки на рынке. Банк решает и сообщает игрокам, сколько ЕСМ продаст в этот раз и какова их минимальная цена. Объявляется также, сколько ЕГП в общей сложности будет закуплено и какова максимальная цена.

Таблица 1. Уровни цен на ЕСМ и ЕГП

Уровень	ЕСМ	Миним. цена	ЕГП	Максим. цена
1	1.0P	$800	3.0P	$6500
2	1.5P	650	2.5P	6000
3	2.0P	500	2.0P	5500
4	2.5P	400	1.5P	5000
5	3.0P	300	1.0P	4500

В таблице 1 приведены пять уровней предложения ЕСМ и спроса на ЕГП (обратите внимание, что с ростом одной из этих величин другая убывает), а также верхние и нижние границы цен для каждого случая. В число игроков P не включены те, кто обанкротился, и P может, таким образом, быть меньше N. Произведения 1.5P и 2.5P округляются до ближайшего целого с недостатком. В таблице 2 приведена матрица вероятностей перехода, в соответствии с которой банк определяет новый месячный уровень спроса и предложения, исходя из прежнего. Предполагается, что в нулевом месяце уровень равен 3.

Таблица 2

Старый уровень	Новый уровень				
	1	2	3	4	5
1	1/3	1/3	1/6	1/12	1/12
2	¼	1/3	1/4	1/12	1/12
3	1/12	1/4	1/3	1/4	1/12
4	1/12	1/12	1/4	1/3	1/4
5	1/12	1/12	1/6	1/3	1/3

3. Заявки на сырье и материалы. Каждый игрок тайно от других готовит заявку на ЕСМ и предлагается цена не ниже банковской минимальной (запрос нуля ЕСМ или предложение цены ниже минимальной автоматически исключает игрока из торгов в этом месяце). Все заявки раскрываются одновременно, и имеющиеся ЕСМ распределяются между игроками в порядке убывания предложенной цены. Если сырья не хватает на всех, заявки с предложением более низкой цены не удовлетворяются. При прочих равных условиях сырье достается самому старшему игроку. Игроки платят за сырье при его получении. Банк не сохраняет оставшееся после удовлетворения заявок сырье на следующий месяц.

4. Производство продукции. Все игроки по очереди (по убыванию старшинства, начиная со старшего) объявляют, сколько ЕСМ они собираются переработать в ЕГП в текущем месяце и на каких фабриках. Каждый игрок обязан тут же покрыть расходы на производство. Обычная фабрика может за месяц переработать одну ЕСМ при затратах в 2000 долл. Автоматизированная фабрика может либо сделать то же, либо переработать 2 ЕСМ при затратах в 3000 долл. Конечно, чтобы переработать ЕСМ, их надо иметь.

5. Продажа продукции. При покупке банком у игроков ЕГП организуются примерно такие же торги, как и при продаже ЕСМ. Заявленные цены не должны превышать максимальную цену, установленную банком, причем банк покупает ЕГП в первую очередь у тех, кто заявил более низкую цену. При прочих равных условиях предпочтение отдается старшему игроку. Если предложение превышает спрос, наиболее дорогие ЕГП остаются непроданными. Игроки получают деньги за продукцию при ее продаже.

6. Выплата ссудного процента. Каждый игрок платит один процент от общей суммы непогашенных ссуд, в том числе и тех, которые будут погашены в текущем месяце.

7. Погашение ссуд. Каждый игрок, получивший ссуду сроком до текущего месяца, должен ее погасить. Поскольку возвращение ссуд предшествует получению новых, платить надо наличными.

8. Получение ссуд. Теперь каждый игрок может получить ссуду. Ссуды обеспечиваются имеющимися у игрока фабриками; под обычную фабрику дается ссуда 5000 долл., под автоматизированную - 10000 долл. Общая сумма непогашенных ссуд не может превышать половины гарантированного капитала, но в этих пределах можно свободно занимать. Банк немедленно выплачивает ссуду игроку. Срок погашения ссуды истекает через 12 месяцев - например, ссуду, взятую в 3-м месяце, возвращать надо в 15-м. Нельзя погашать ссуды раньше срока.

9. Заявки на строительство. Игроки могут строить новые фабрики. Обычная фабрика стоит 5000 долл. и начинает давать продукцию на 5-ый месяц после начала строительства; автоматизированная фабрика стоит 10000 долл. и дает продукцию на 7-ой месяц после начала строительства. Обычную фабрику можно автоматизировать за 7 000 долл., реконструкция продолжает-

ся 9 месяцев, все это время фабрика может работать как обычная. Половину стоимости фабрики надо платить в начале строительства, вторую половину - за месяц до начала выпуска продукции в этой же фазе цикла. Общее число имеющихся и строящихся фабрик у каждого игрока не должно превышать шести.

Окончание игры и подсчет результатов

Игра заканчивается после некоторого фиксированного числа кругов (13 или более) или когда обанкротятся все игроки, кроме одного. Чтобы посчитать общий капитал компании, нужно сложить стоимость всех фабрик (по цене, по которой их можно было построить заново), стоимость имеющихся у нее ЕСМ (по минимальной банковской цене текущего месяца), стоимость имеющихся ЕГП (по максимальной банковской цене текущего месяца) и имеющиеся у компании наличные. Если к концу игры приходят несколько игроков, результаты считаются по их капиталам. Любой игрок в любой момент может узнать состояние дел любого другого игрока - его капитал, наличные, взятые ссуды, все, что касается готовой продукции, имеющихся и строящихся фабрик. Во время торгов игроки ничего не знают о заявках, сделанных другими, но, как только банк собрал все заявки, они обнародуются и количество купленных или проданных каждым игроком единиц становится известно всем. Игроки могут сами вести любые записи, но банк не предоставляет им никакой информации, кроме той которая предусмотрена правилами игры.

Задание

Задача состоит из двух частей. Первая заключается в том, чтобы написать программу, которая управляет ходом моделирования - программу-банкир. Эта программа должна полностью контролировать игру: устанавливать цены, закупать продукцию и продавать сырье, проводить торги, вести учет и т.д. Эта программа должна в соответствующие моменты опрашивать игроков и добиваться соблюдения ими всех правил. Программа-банкир периодически (например, ежемесячно) выдает сводный финансовый отчет. Поскольку отчеты эти предстоит читать людям, они должны быть понятны и приемлемы с эстетической точки зрения.

Вторая часть задачи - написать программы поведения игроков. Всего играет три игрока: один - игрок-человек, находящийся у терминала, два игрока - "вложенные" в машину подпрограммы. Каждая программа-игрок должна быть в состоянии отвечать на любые запросы программы-банкира по ходу игры: она должна уметь предлагать цену на сырье, принимать решения, продавать готовую продукцию и т. п. Реализуйте одну из программ-игроков таким образом, чтобы она просто передавала свои запросы игроку-человеку, нахо-

дящемуся за терминалом. Такая программа должна уметь отвечать на запросы человека о состоянии игры. После того как будет написаны программы-игроки, их надо объединить с программой-банкиром, чтобы получилась полная игровая система. Проведите с этой системой несколько игр и изучите результаты. Для того, чтобы результаты были достаточно реальны надо написать нетривиальные программы-игроки.

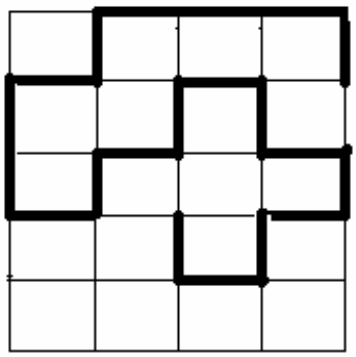
Эта игра - пример последовательного, или пошагового, моделирования, при котором все события (кроме банкротств), происходят в строго определенном заранее известном порядке. Цикл по месяцам - удобная структура для ведущей программы.

Литература для ссылок на постановку задачи:
Уэзерелл Ч. Этюды для программистов. М.: Мир, 1982, стр. 46-54.

В задаче необходимо будет строить случайную последовательность v[0],v[1],...,члены которой принадлежат конечному множеству, состоящему из элементов x[1],x[2],...,x[n]. Элементы x[1],...,x[n] повторяются в последовательности v[0],v[1],... с заданными частотами, соответственно равными p[1],...,p[n] (т. е. относительно последовательности v[0],v[1],... можно считать, что x[i] встречается с ней с вероятностью p[i] (i=1,...,n),p[1]+p[2]+...+p[n]=1). Для построения v[0],v[1],... используется следующий прием. Интервал (0,1) разбиваем на n интервалов, длины которых равны p[1],...,p[n]. Координатами точек разбиения будут p[1],p[1]+p[2],p[1]+p[2]+p[3],..., p[1]+p[2]+...+p[n-1]. Полученные интервалы обозначим через I[1],...,I[n]. Пусть r[0],r[1],... - случайная числовая последовательность, члены которой равномерно распределены в интервале (0,1). Перебираем числа r[0],r[1],..., и если очередное число r[k] попадает в интервал I[j] (1<=j<=n), то в качестве v[k] берем x[j] (вероятность попадания r[k] в некоторый интервал равна длине этого интервала). Если вдруг окажется, что r[k] совпадает с точкой разбиения (концом интервала), то можно условно считать, что число r[k] попало в интервал, лежащий справа от него.

10. Топологическая игра "Ползунок"

Написать программу для игры с компьютером в игру "Ползунок". Компьютер должен использовать минимаксный принцип для выигрышной стратегии. Язык программирования – любой.



Описание игры.

Решетка для этой игры размером 5 на 6 точек изображена на рисунке. Правила этой игры просты: каждый играющий по очереди проводит ортогональные отрезки прямых длиной в одну единицу. Получающаяся в результате траектория игры должна быть непрерывной, причем каждый последующий ход можно делать с его любого ее конца. Игрок, вынужденный замкнуть траекторию проигрывает. На рисунке приведена типичная позиция, при которой следующий ход является последним – тот кто его делает, проигрывает.

Об истории игры см. книгу: М. Гарднер, Крестики-нолики. М., Мир, 1988, с. 269.

5. КАК ВЫПОЛНЯТЬ КУРСОВУЮ РАБОТУ И ОФОРМЛЯТЬ ПОЯСНИТЕЛЬНУЮ ЗАПИСКУ

Общие положения

Основные задачи и цели курсового проектирования:

1) приобретение навыков и методов программирования достаточно сложных задач искусственного интеллекта;
2) подготовка к выполнению дипломного проекта.

В курсовой работе должна быть разработана тема в соответствии с заданием, одобренным кафедрой.

Общие требования к построению пояснительной записки (ПЗ)

Структура построения ПЗ

ПЗ к работе должна содержать следующие разделы:

1) титульный лист;
2) реферат;
3) задание на проектирование;
4) содержание;
5) введение;
6) основная часть работы;
7) заключение;
8) список литературы;
9) приложения.

Титульный лист

Титульный лист оформляется согласно ГОСТ 2.105-79, форма титульного листа приведена в приложении 1.

Реферат

Реферат - краткая характеристика работы с точки зрения содержания, назначения, формы и других особенностей. Перечисляются ключевые слова работы, указывается количество страниц и приложений. Реферат размещают на отдельной странице. Заголовком служит слово "Реферат", написанное прописными буквами.

Задание на проектирование

Форма задания заполняется студентом в соответствии с полученным заданием. Форма задания приведена в приложении 2.

Содержание

Содержание включает наименования всех разделов, подразделов и пунктов, если они имеют наименование, а также список литературы

и приложения с указанием номера страниц, на которых они начинаются. Слово "Содержание" записывается в виде заголовка, симметрично тексту, прописными буквами. Пример оформления содержания приведен в приложении 3.

Введение

Введение содержит основную цель курсовой работы, область применения разрабатываемой темы.

Заключение

Заключение должно содержать краткие выводы по выполненной работе. Также следует указать, чему программист научился на примере этой задачи (на этот вопрос легко ответить, если сформулировать его в виде: "Что я в следующий раз сделаю иначе?").

Список литературы

В список литературы входят все те и только те источники литературы, на которые имеются ссылки в ПЗ. Примеры библиографических описаний источников, помещаемых в список литературы, приведены в приложении 4.

Приложения

Приложения содержат вспомогательный материал: листинг программы и листинг тестов.

Программа должна быть самодокументированная, т.е.

- программа должна иметь простую и понятную структуру,
- в программе должны быть прокомментированы используемые структуры данных,
- для каждой функции должно быть указано, что она делает, что является входными данными и результатом,
- должен быть прокомментирован используемый алгоритм.

Основная часть курсовой работы

В основной части должно быть решение поставленной задачи, в частности:

- анализ задачи;
- обоснование выбора алгоритма;
- обоснование выбора структур данных;
- описание алгоритма;
- обоснование набора тестов.

Об анализе задачи

Разработка алгоритма представляет собой задачу на построение. Поэтому, как обычно в таких случаях (можно, например, вспомнить о методе

решения геометрических задач на построение), необходим этап анализа задачи. Он позволяет установить, что является входом и выходом будущего алгоритма, выделить основные необходимые отношения между входными и выходными объектами и их компонентами, выделить подцели, которые нужно достичь для решения задачи, и как следствие этого, выработать подход к построению алгоритма. Результатом этапа анализа задачи должна быть спецификация алгоритма, т. е. формулировка в самом общем виде того, что (в рамках выбранного подхода) должен делать алгоритм, чтобы переработать входные данные в выходные.

Об описании алгоритма

Прежде всего, нужно иметь в виду, что такое описание предназначено не для машины, а для человека. Другими словами, речь идет не о программе, а о некотором тексте (т. е. о словесном описании), по которому можно получить представление об общей структуре разрабатываемого алгоритма, о смысле его отдельных шагов и их логической взаимосвязи. Сохранение достаточно высокого уровня описания алгоритма также облегчает его обоснование. Поэтому шаги алгоритма должны описываться в терминах тех объектов и отношений между ними, о которых идет речь в формулировке задачи. Например, для "геометрической" задачи шаги алгоритма следует описывать как действия над точками, прямыми и т. п., как проверки свойств типа принадлежности трех точек одной прямой и т. п. Но не должно быть работы с кодами этих объектов, например с матрицей координат точек некоторого множества.

При программировании на Прологе описание предикатов должно заключаться в указании, для каких отношений между сущностями (объектами предметной области) они введены. Какие аргументы предиката являются входными, а какие выходными?

Нужно подобрать набор тестов, достаточный для демонстрации работы программы и ее реакции на экстремальные ситуации и неправильное обращение.

Правила оформления ПЗ к курсовой работе

ПЗ пишется в редакторе MS Word шрифтом Times New Roman, размером 12, на формате А4. Нумерация страниц должна быть сквозной, первой страницей является титульный лист. Номер страницы проставляется вверху посередине. Заголовки разделов пишутся прописными буквами по середине текста. Заголовки подразделов пишутся с абзаца строчными буквами, кроме первой прописной. В заголовке не допускаются переносы слов. Точку в конце заголовка не ставят. Если заголовок состоит из двух предложений, то их разделяют точкой.

6. РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Основная литература

1. Братко И. Программирование на языке Пролог для искусственного интеллекта.- М.:Мир,1990.- 560 с.
2. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог.- М.: Мир, 1990. - 235 с.
3. Логический подход к искусственному интеллекту: от классической логики к логическому программированию: Пер.с франц./Тейз А., Грибомон П., Луи Ж. и др. - М.:Мир,1990.- 432 с.
4. Лорьер Ж.Л. Системы искусственного интеллекта. М.: Мир,1991.- 568 с.

Дополнительная литература

1. Малпас Дж. Реляционный язык Пролог и его применение. - М.: Наука, 1990. - 464 с.
2. Годфруа Ж. Что такое психология.Т.1 - М.:Мир,1992.
3. Лем С. Сумма технологии -М.:Мир,1968. - 608 с.

ПРИЛОЖЕНИЕ 1

ФОРМА ТИТУЛЬНОГО ЛИСТА К КУРСОВОЙ РАБОТЕ

Министерство образования Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении
и проектировании (КСУП)

Самообучающаяся программа

Пояснительная записка к курсовой работе по дисциплине
"Искусственный интеллект и экспертные системы"

Студент гр. 589-1
С. С. Лавров
20.12.99

1999

ПРИЛОЖЕНИЕ 2

ФОРМА ЗАДАНИЯ ДЛЯ КУРСОВОЙ РАБОТЫ

Министерство образования Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении
и проектировании (КСУП)

УТВЕРЖДАЮ
Заведующий кафедрой КСУП
_____Ю. А. Шурыгин

ЗАДАНИЕ

по курсовому проектированию по дисциплине
"Искусственный интеллект и экспертные системы"

студенту_____
группа_____ факультет ФВС

1. Тема проекта: "Самообучающаяся программа"
2. Срок сдачи студентом законченной работы 25.12.99.
3. Исходные данные к проекту (здесь должен быть текст задания)
4. Дата выдачи задания: 01.6.99 г.

Задание принял к исполнению
(ФИО)

ПРИЛОЖЕНИЕ 3

ПРИМЕР ОФОРМЛЕНИЯ СОДЕРЖАНИЯ

СОДЕРЖАНИЕ

ПРИЛОЖЕНИЕ 4

ПРИМЕРЫ БИБЛИОГРАФИЧЕСКИХ ОПИСАНИЙ ИСТОЧНИКОВ, ПОМЕЩАЕМЫХ В СПИСОК ЛИТЕРАТУРЫ

1. Хендерсон П. Функциональное программирование. Применение и реализация. - М.: Мир, 1983. - 349 с.
2. Филд А., Харрисон П. Функциональное программирование. - М.: Мир, 1993. - 637 с.
3. Шеховцов А. С. Квазисинхронное регулирование гистерезисных электродвигателей // Тез. докл. на науч.-техн. конф. 21-23 дек. 1998 г. - Т. 4 - Томск: Издательство Томского государственного педагогического университета, 1999. - 133 с.
4. Калянов Г. Н. CASE-технологии проектирования программного обеспечения // Кибернетика и системный анализ. - 1993. - №5. - С. 152-164.