

Темы для курсовых работ по ИИ

Варианты курсовых работ разные по трудности. В скобках рядом с названием трудность в баллах. Указанное количество баллов вы получаете за выбор данной темы.

1.	Упрощение электрических цепей (8).....	3
2.	Программа для алгебраических вычислений (10).....	3
3.	Задача Баше: «Суммируйте до ...» (13).....	6
4.	Упрощение арифметических выражений (7).....	9
5.	Игра "Выдающийся ум" ("Быки и коровы") (3).....	9
6.	Нахождение геометрических аналогий (0)	11
7.	Логическая головоломка «зебра» (7).....	13
8.	Самообучающаяся программа для игры «шесть пешек» (12)	16
9.	Самообучающаяся программа для игры «классические крестики-нолики» (15).....	17
10.	Игра «классические крестики-нолики» с использованием метода минимакса (15)	18
11.	Самообучающаяся программа для игры «крестики-нолики с поддавками» (15)	18
12.	Игра «крестики-нолики с поддавками» с использованием метода минимакса (15)	19
13.	Моделирование управления предприятиями (12).....	19
14.	Топологическая игра "Ползунок" (8).....	22
15.	Алгоритм Ли для трассировки (0)	22
16.	Автоматическое доказательство теорем в исчислении высказываний (5)	22
17.	Метаинтерпретатор (3)	23
18.	Программа Eliza (10)	23
19.	Советник по транспорту (5)	23
20.	Задача Прима-Краскала («жадный алгоритм») (15).....	23
21.	Модуль для операций с мультимножествами (15)	24
22.	Модуль для последовательностей (14).....	25
23.	Миры Р. Смаллиана: «рыцари и лжецы» (Haskell)(15).....	26
24.	Миры Р. Смаллиана: обобщения задач с рыцарями и лжецами (Пролог) (14).....	26
25.	Миры Р. Смаллиана: тайна шкатулок Порции (Пролог) (14).....	27
26.	Миры Р. Смаллиана: Лев и Единорог (Пролог) (14)	29
27.	Миры Р. Смаллиана: Лев и Единорог (Haskell) (15).....	30
28.	Миры Р. Смаллиана: Траляля и Труляля (Пролог) (14)	30
29.	Миры Р. Смаллиана: Траляля и Труляля (Haskell) (15).....	31
30.	Пропозициональная логика (Haskell) (15)	31
31.	Детективные головоломки (15).....	33
32.	Построение самоссылочных утверждений о количестве цифр (15).....	35
33.	Муравей Лэнгтона (15)	36
34.	Моделирование эпидемий (14)	37
35.	Моделирование пожара (14)	37

36.	Двумерная диффузия (12).....	38
37.	Агрегирование ограниченное диффузией (13)	38
	Балльная раскладка рейтинга	39

1. Упрощение электрических цепей (8)

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 (10)

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

Приведем резюме требований и рекомендаций:

1. Все многочлены только от одной переменной, скажем, X.
2. Моном (одночлен) имеет вид <коэффициент>*x^<показатель степени>.
3. Показатель степени может быть только целой неотрицательной переменной.
4. Если коэффициент равен 1, то он отсутствует: x^<показатель степени>
5. Если коэффициент равен 0, то весь моном имеет вид: 0
6. Если показатель степени равен 0, то моном имеет вид: <коэффициент>
7. Если показатель степени равен 1, то моном имеет вид: <коэффициент>*x
8. Вид монома (указанный в пунктах 2–7) имеет место при вводе многочлена и при выводе, во внутреннем представлении многочлен есть список мономов, упорядоченных по степеням переменной.
9. Упорядоченность многочленов осуществляется при вводе мономов предикатом merge.
10. При сложении многочленов, используется упорядоченность многочленов (предикат merge).
11. При умножении используйте алгоритм:

$$\text{Многочлен } A = \sum_{i=1}^n a_i \text{ (} a_i \text{ – мономы) , многочлен } B = \sum b_i$$

$$A \times B = \sum_{i=1}^n (a_i \times B)$$

Т.е. удобно вначале организовать умножение монома на многочлен (почленное умножение; упорядоченность полученных мономов в произведение сохраняется), а потом в цикле использовать уже реализованное сложение многочленов.

12. При возведении в степень удобно использовать алгоритм:
Многочлен A надо возвести в степень n. Пусть E – многочлен = 1.

For I:=1 to n do

Умножаем А на Е, результат помещаем в Е.

13. При замене переменной на многочлен надо использовать уже реализованные операции: для замены переменной в мономе – возведение многочлена в степень и умножение результата на коэффициент; для замены во всем полиноме осталось сложить результаты для монома.
14. Все указания можно игнорировать – лишь бы ваша программа выполняла все поставленные задачи, была бы проста и понятна по структуре и элегантна.

3. Задача Баше: «Суммируйте до ...» (13)

Любитель математики и поэт Гаспар Клод Баше де Мезириак (1581–1638) известен, в частности, своей книгой «Игры и задачи, основанные на математике», послужившей источником многих тем более современной книги У. Болла и Г. Коксетера. Рассмотрим простую игру, предложенную Баше.

Участвуют в ней двое, A и B : сначала A называет какое-нибудь число, меньшее, скажем, шести; затем B прибавляет к нему любое число, меньшее шести, и называет полученную сумму; далее A делает то же самое и т. д. Выигрывает тот, кто первым назовет определенное (заранее заданное) число, скажем 50.

Оказывается, тот, кто начинает, обладает выигрышной стратегией и, следовательно, всегда выигрывает.

Обобщим задачу на случай произвольных значений параметров. Введем некоторые обозначения. Обозначим предельную сумму, по достижении которой игра заканчивается, как *limit*. Обозначим список чисел, которые играют роль потенциальных слагаемых, как *moves*, а *max* будет обозначать наибольшее из этих чисел.

Во-первых, очевидно, что существует такие *limit* и *moves*, для которых точное равенство *limit* и суммы чисел невозможно (например, *limit* = 10 и *moves* = [3]). Поэтому условие окончания игры мы определим следующим образом: выигрывает тот игрок, сумма чисел после хода которого станет не меньше *limit*.

Во-вторых, мы ограничим числа из *moves* натуральными (в этом случае мы можем гарантировать, что партия закончится за конечное число ходов). С учетом этого замечаний игра может быть определена следующим образом: пусть дано некое число *limit* и список натуральных чисел *moves*. Два игрока по очереди называют произвольное число из *moves*. Названные числа суммируются, выигрывает тот игрок, сумма чисел после хода которого станет не меньше *limit*.

Возможно, при каких-то значениях параметров *limit* и *moves* существуют выигрышные стратегии для первого или второго игрока. Пусть второй игрок будет компьютер. Требуется написать программу на SWI-Prolog'e, которая реализует (если это возможно) выигрышную стратегию для компьютера.

Проанализируем игру. Поскольку все числа в *moves* положительные и партия заканчивается, когда сумма чисел достигнет или превысит *limit*, то дерево ходов этой игры конечно. Это означает, что результат игры для любой позиции предопределен, т. е. до начала игры можно вычислить, какой из игроков выиграет партию (при правильной игре, конечно).

Единственная информация, которая однозначно определяет ход, – это сумма чисел, достигнутая к данному ходу. Это значит, что любая позиция в игре может быть описана единственным параметром – набранной суммой чисел. Поскольку сумма чисел в игре не может превышать *limit* + *max* – 1, то существует не более *limit* + *max* позиций: начальной позиции соответствует сумма 0, окончанию партии соответствует суммы от *limit* и выше. В зависимости от выбранного набора чисел *moves* позиции могут быть заполнены полностью или частично.

Назовем неотрицательное целое число $x \in [0, \text{limit} + \text{max} - 1]$ «хорошим» (с точки зрения компьютера!), если текущая сумма есть x и у человека, попавшего в эту позицию, нет ни одного хода, ведущего к выигрышу. В противном случае назовем число x «пло-

хим». Например, поскольку при достижении компьютером суммы не меньшей *limit* у человека вообще нет ходов (игра закончилась), то, следовательно, все числа из диапазона $[limit, limit+max-1]$ – «хорошие». С другой стороны, число *limit* – *max* есть «плохое», так как человеку достаточно в качестве слагаемого выбрать *max* и выиграть. Позицию, соответствующую текущей сумме *x* будет представлять в виде пары $[x, true]$, если *x* – «хорошее число», или $[x, false]$ – в противном случае. Если для всех чисел $x \in [0, limit+max-1]$ программа определит, какие из них являются «хорошими», а какие «плохими», то выигрышная стратегия для компьютера следующая – каждый раз он должен делать такой ход, чтобы перейти в «хорошую» позицию. Например, для случая *limit* = 10 и *moves* = [3] хорошими позициями являются 0, 4, 5, 6, 10, 11, 12 и, очевидно, компьютер, при своем втором ходе всегда выигрывает. Обратите внимание, что оценка выигрышности позиций проделана для всех чисел от 0 до 12, хотя с данным единственным возможным ходом «прибавлять 3», достижимыми позициями могут быть только 0, 3, 6, 9, 12.

Рассмотрим, каким образом можно оценить позиции. Это можно сделать рекурсивным алгоритмом, время работы которого линейно зависит от *limit*. Пусть позиции представляются списком пар вида:

```
[[Integer, Bool]]
```

Пары, в которых первые компоненты больше или равны *limit*, являются «хорошими», таким образом, для $n \geq limit$ имеем $[n, true]$. Остальные значения можно оценить рекурсивно, уменьшая *n* на каждом шаге. Будем использовать рекурсивную функцию

```
initPos1 :: [[Integer, Bool]] × Integer -> [[Integer, Bool]]
```

для которой список оцененных позиций есть

```
initPos = initPos1 ([], limit - 1 + max)
```

В исходной ситуации список оцененных («инициализированных») позиций – пуст. Он является накапливающим первым параметром функции *initPos1*. Значением второго параметра есть позиция, которую предстоит оценить, на каждом шаге этот параметр будет уменьшаться на 1.

Какими свойствами обладает функция *initPos1*(*ps*, *n*)?

- Если *n* меньше нуля, то *ps* содержит все инициализированные позиции, начиная с конца *limit* – 1 + *max* и до 0 – конец рекурсии.
- Если $n \geq limit$, то пару $[n, true]$ надо добавить в список *ps* и перейти к оценке позиции *n* – 1.
- Если существует какой-то ход *move*, при котором противник (человек) из позиции *n* переходит в «хорошую» позицию *n*+*move*, то позиция *n* оценивается как «плохая», добавляется в список *ps* пара $[n, false]$ и осуществляется рекурсивный вызов с *n*–1.
- Оставшаяся ситуация приходится на «хорошую» позицию $[n, true]$.

После инициализации списка позиций и начинается собственно игра. Ее можно реализовать в виде интерактивной программы на любом языке программирования.

Компьютер вычисляет в позиции *s* список ходов, которые приводят к «хорошим» позициям, и выбирает первый элемент в этом списке, если он не пуст. Если же у компьютера нет «хорошего» хода, то выбирается первый ход из списка *moves*.

Я написал программу на современном функциональном языке Haskell.

Смотрите примеры работы:

Приведем два сеанса игры. В первом выигрышная стратегия для компьютера имеется:

```

> limit
21
> moves
[4,3]
> initPos
[(0,True),(1,True),(2,True),(3,False),(4,False),(5,False),(6,False),(7,True),
(8,True),(9,True),(10,False),(11,False),(12,False),(13,False),(14,True),
(15,True),(16,True),(17,False),(18,False),(19,False),(20,False),(21,True),
(22,True),(23,True),(24,True)]

> main
Сумма равна 0
Ход человека:
3
Сумма равна 3
Ход компьютера:4
Сумма равна 7
Ход человека:
4
Сумма равна 11
Ход компьютера:4
Сумма равна 15
Ход человека:
4
Сумма равна 19
Ход компьютера:4
Сумма равна 23
компьютер победил

```

Выигрышной стратегии для компьютера нет:

```

> limit
21
> moves
[6,4,3]
> initPos
[(0,False),(1,False),(2,False),(3,True),(4,True),(5,True),(6,False),
(7,False),(8,False),(9,False),(10,False),(11,False),(12,True),(13,True),
(14,True),(15,False),(16,False),(17,False),(18,False),(19,False),(20,False),
(21,True),(22,True),(23,True),(24,True),(25,True),(26,True)]

> main
Сумма равна 0
Ход человека:
3
Сумма равна 3
Ход компьютера:6
Сумма равна 9
Ход человека:
4
Сумма равна 13
Ход компьютера:6
Сумма равна 19
Ход человека:
4
Сумма равна 23
человек победил

```

Литература: Болл У., Коксетер Г. Математические эссе и развлечения. – М.: Мир, 1986. – 476 с.

4. Упрощение арифметических выражений (7)

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

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") (3)

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" - количество 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),           % В -быки, С - 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 (0)

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 А, В и С и фигуры, представляющие собой варианты решений, следует дать ответ на вопрос: "Фигура А относится к фигуре В, как фигура С 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 А и В;
- применить это правило к фигуре С, чтобы перейти к фигуре Х;
- найти среди ответов фигуру Х. или ее ближайший эквивалент.

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

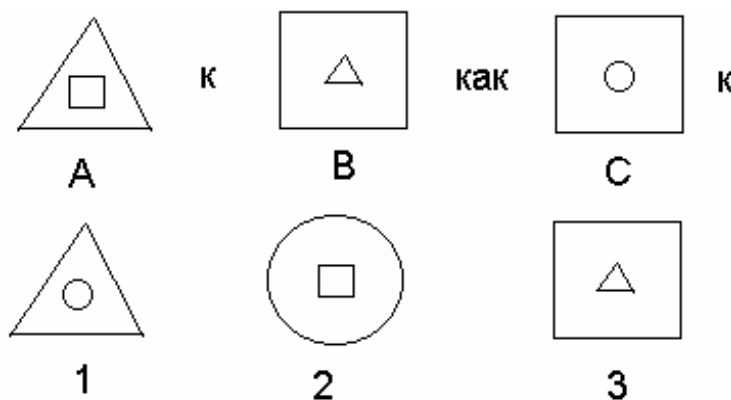


Рис. 1 Задача на геометрические аналогии

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.

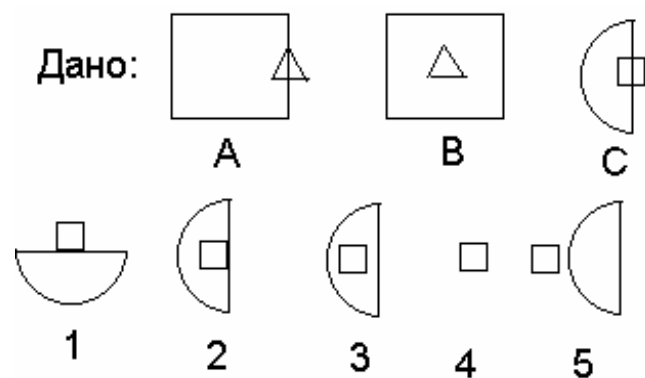


Рис. 2 Первый тест

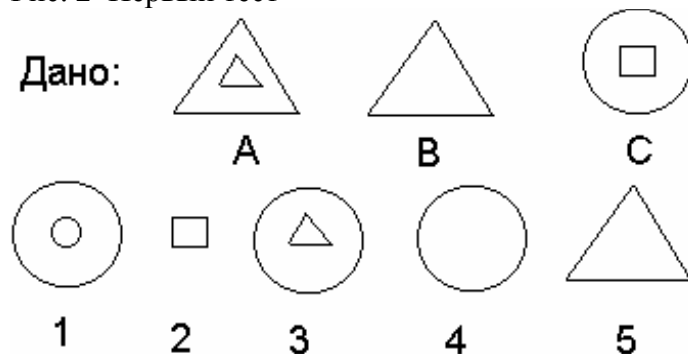


Рис. 3 Второй тест

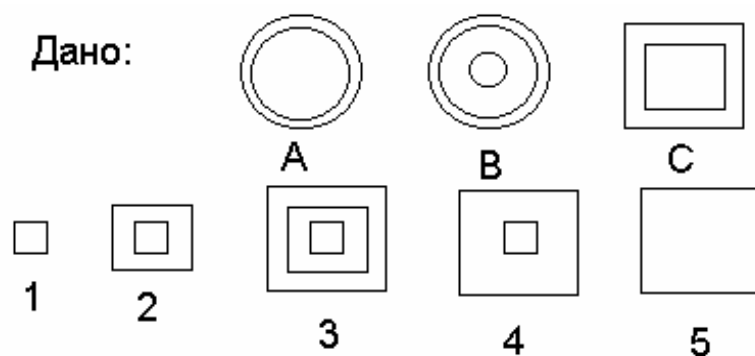


Рис. 4 Третий тест

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» (7)

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 про-

грамма 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. Проанализируем 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 `friend(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]].
```

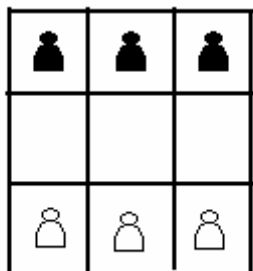
8. Самообучающаяся программа для игры «шесть пешек» (12)

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 удастся 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 проигрывает, 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 позициях 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 победы, увеличивая вероятности хороших ходов.

Для быстрого 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. Самообучающаяся программа для игры «классические крестики-нолики» (15)

Крестики-нолики (классические), или тик-тау-тоу, – это игра со следующими правилами:

- 1) играют на поле 3×3 ;

- 2) двое игроков ходят по очереди;
- 3) 1-й игрок ставит крестик на любом свободном поле, 2-й – нолик;
- 4) выигрывает тот, кто первым поставит в ряд три своих значка;
- 5) если поле заполнено, и никто не выиграл, то засчитывается ничья.

На поле 3×3 имеется 8 рядов из 3 клеток: три столбца, три строки и две диагонали.

Требуется написать программу на Прологе, чтобы компьютер мог играть против человека. Искусственный интеллект в программу может быть заложен с помощью самообучения (см. тему «Самообучающаяся программа для игры «шесть пешек»»).

Можно уменьшить число рассматриваемых позиций, если не различать позиции, полученные одна из другой с помощью осевой симметрии. Получается так потому, что квадратное поле 3×3 имеет 4 оси зеркальной симметрии (повороты рассматривать не будем):



10. Игра «классические крестики-нолики» с использованием метода минимакса (15)

Требуется написать программу на Прологе, чтобы компьютер мог играть против человека в классические крестики-нолики (см. тему «Самообучающаяся программа для игры «классические крестики-нолики»»). Искусственный интеллект в программу может быть заложен с помощью метода минимакса.

Простейшая оценочная функция для крестиков-ноликов следующая. Припишем каждой позиции число x , равное разности возможных линий из крестиков и возможных линий из ноликов. При наличии ряда из 3 крестиков (выигрыше крестиков) значение x пусть будет равно $+\infty$, ряда из трех ноликов (выигрыш ноликов) – $-\infty$. Примеры см. на рис.

$x = 0 =$ $= 8 - 8$	$x = 3 =$ $= 8 - 5$	$x = 1 =$ $= 6 - 5$	$x = 0 =$ $= 5 - 5$	$x = -1 =$ $= 4 - 5$	$x = -\infty$	$x = +\infty$

Рис. Оценка некоторых позиций

11. Самообучающаяся программа для игры «крестики-нолики с поддавками» (15)

Поддавки, или тоу-так-тик, или мизер, – это вариант крестиков-ноликов, в которых игрок проигрывает тогда, когда он выигрывает классических крестиках-ноликах.

Поддавки с одной стороны по сложности такие же, как и нормальные крестики-нолики, поскольку при правильной игре обоим игрокам гарантирована ничья. С другой стороны, поддавки сложнее: нужно перестраиваться на то, чтобы заставить противника ряд из его трех значков, а преимущество не у первого игрока, а у второго – первый игрок ставит на один крестик больше чем второй ноликов.

Требуется написать программу на Прологе, чтобы компьютер мог играть против человека. Искусственный интеллект в программу может быть заложен с помощью самообучения (см. тему «Самообучающаяся программа для игры «шесть пешек»»).

12. Игра «крестики-нолики с поддавками» с использованием метода минимакса (15)

Требуется написать программу на Прологе, чтобы компьютер мог играть против человека в крестики-нолики с поддавками. Искусственный интеллект в программу может быть заложен с помощью метода минимакса (см. тему «Игра «классические крестики-нолики» с использованием метода минимакса»).

13. Моделирование управления предприятиями (12)

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 $T \bmod N + 1$). 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 финансовых 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	1/4	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 продолжается 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-банка 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, находящемуся за терминалом. Такая программа должна уметь отвечать на запросы человека о состоянии игры. После того как будут 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.

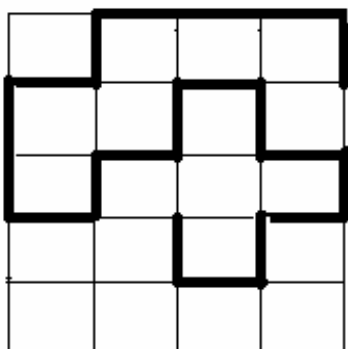
Литература для ссылок на постановку задачи:

Уэзерелл Ч. Этюды для программистов. М.: Мир, 1982, стр. 46-54.

В задаче необходимо будет строить случайную последовательность $v[0], v[1], \dots$, члены которой принадлежат конечному множеству, состоящему из элементов $x[1], x[2], \dots, x[n]$. Элементы $x[1], \dots, x[n]$ повторяются в последовательности $v[0], v[1], \dots$ с заданными частотами, соответственно равными $p[1], \dots, p[n]$ (т. е. относительно последовательности $v[0], v[1], \dots$ можно считать, что $x[i]$ встречается с ней с вероятностью $p[i]$ ($i=1, \dots, n$), $p[1]+p[2]+\dots+p[n]=1$). Для построения $v[0], v[1], \dots$ используется следующий прием. Интервал $(0,1)$ разбиваем на n интервалов, длины которых равны $p[1], \dots, p[n]$. Координатами точек разбиения будут $p[1], p[1]+p[2], p[1]+p[2]+p[3], \dots, p[1]+p[2]+\dots+p[n-1]$. Полученные интервалы обозначим через $I[1], \dots, I[n]$. Пусть $r[0], r[1], \dots$ - случайная числовая последовательность, члены которой равномерно распределены в интервале $(0,1)$. Перебираем числа $r[0], r[1], \dots$, и если очередное число $r[k]$ попадает в интервал $I[j]$ ($1 \leq j \leq n$), 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.

14. Топологическая игра "Ползунок" (8)

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.

15. Алгоритм Ли для трассировки (0)

Описание алгоритма в книге Стерлинг Л., Шапиро Э. Искусство программирования на языке ПРОЛОГ, М.: Мир, стр.217-219. Измените программу 17.6 так, чтобы можно было обрабатывать препятствия в виде прямоугольников, кругов и треугольников.

16. Автоматическое доказательство теорем в исчислении высказываний (5)

Напишите программу на SWI-прологе для автоматического доказательства теорем в исчислении высказываний. Постановка задачи и рекомендации в книге: Братко И. Программирование на языке ПРОЛОГ для искусственного интеллекта. -М.: Мир, 1990. На

стр. 524-532 приведен текст программы и на стр. 533 в разделе «Проект» предложено изменить программу с соответствии с некоторыми требованиями. Вы должны выполнить этот проект.

17. Метаинтерпретатор (3)

Напишите метаинтерпретатор для подсчета числа вызовов процедуры в некоторой программе на Прологе.

Надо определить предикат `count(+Цель, +P, -N)`, который при своем вызове вычисляет «Цель» (любая допустимая цель) и считает, сколько раз вызывался предикат с именем P.

Литература. Стерлинг Л., Шапиро Э. Искусство программирования на языке ПРОЛОГ, М.: Мир, стр. 238–245.

18. Программа Eliza (10)

Написать на Прологе упрощенный вариант классической программы Eliza. Постановка задачи и реализация приведены в книге

Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. - М.: Мир. - 1990, стр.184-185.

Задача для вас должна состоять из четырех этапов:

- 1) перепишите программу из книги, упростив ввод: входная фраза должна вводиться в виде списка атомов; для этого вместо `read_word_list` используйте просто `read`;
- 2) после того, как у вас программа заработает, ввод предложений сделайте в виде строки символов; теперь надо использовать предикат `read_word_list`;
- 3) теперь вам надо добиться, чтобы программа воспринимала русский язык и отвечала по-русски; для этого вам надо использовать версию SWI-prolog 3.2.8;
- 4) теперь расширьте программу путем добавления новых шаблонов для пар стимул/отклик.

19. Советник по транспорту (5)

Некоторая железнодорожная компания обслуживает n станций S_1, S_2, \dots, S_n . Она предлагает улучшить информационное обслуживание клиентов с помощью информационных терминалов, управляемых вычислительной машиной. Клиент набирает название своей станции направления S_a и станции назначения S_d , и ему должна выдаваться схема расписаний поездов с необходимыми пересадками и с минимальным общим временем поездки.

Разработайте программу для вычисления нужной информации. Пусть расписание (представляющее собой ваш банк данных) изображено соответствующей структурой данных, содержащей время отправления и прибытия всех имеющихся поездов. Разумеется, не все станции связаны непрерывными линиями.

20. Задача Прима-Краскала («жадный алгоритм») (15)

Дана плоская страна и в ней n городов. Нужно соединить все города телефонной связью так, чтобы общая длина телефонных линий была минимальной.

Уточнение задачи. В задаче речь идет о телефонной связи, т. е. подразумевается транзитивность связи: если i -й город связан с j -ым, а j -ый с k -ым, то i -й связан с k -ым. Подразумевается также, что телефонные линии могут разветвляться только на телефонной станции, а не в чистом поле. Наконец, требование минимальности (вместе с транзитивностью) означает, что в искомом решении не будет циклов. В терминах теории графов задача Прима-Краскала выглядит следующим образом:

Дан граф с n вершинами; заданы длины ребер. Найти остовное дерево минимальной длины.

Как известно, дерево с n вершинами имеет $n-1$ ребер. Оказывается, каждое ребро надо выбирать жадно (лишь бы ни возникали циклы).

Удобно представлять граф в виде двух списков: список вершин и список ребер. Каждое ребро, в свою очередь, есть тройка, содержащая две вершины и расстояние между ними.

Алгоритм Прима-Краскала
(краткое описание)

В цикле $n-1$ раз делай:

"выбрать самое короткое еще не выбранное ребро при условии, что оно не образует цикл с уже выбранными".

Выбранные таким образом ребра образуют искомое остовное дерево. Напишите программу для решения задачи Прима-Краскала на языке Haskell.

Исходные данные представляйте в виде списка троек (\langle вершина \rangle , \langle вершина \rangle , \langle расстояние \rangle). Совсем не обязательно граф должен быть полным. Результат будет под-списком исходного списка.

21. Модуль для операций с мультимножествами (15)

Программирование на языке Haskell.

«Мешок» или мультимножество есть собрание элементов, в котором каждый элемент может присутствовать один или более число раз.

Выберите эффективное представление для мультимножества.

Определите модуль для операций с мультимножествами, включающий следующие функции.

- Функция `listToBag` берет в качестве аргумента список элементов и возвращает мультимножество, представляющее данный список.
- Функция `bagToList` берет в качестве аргумента мультимножество и возвращает (один из возможных) списков, содержащий элементы мультимножества. В этом списке каждый элемент встречается столько раз, сколько раз он входит в мультимножество.
- Функция `bagToSet` превращает мультимножество в список, в котором каждый элемент встречается только один раз.
- Функция `bagEmpty` определяет, является ли данное мультимножество пустым множеством или нет.
- Функция `bagCard` берет в качестве аргумента мультимножество и определяет его мощность, т. е. суммарное количество вхождений всех элементов в мультимножество.
- Функция `bagElem` берет в качестве аргументов мультимножество и элемент и возвращает `True`, если элемент входит в мультимножество, и `False` в противном случае.
- Функция `bagOccur` берет в качестве аргументов мультимножество и элемент и возвращает количество вхождений данного элемента в мультимножество.
- Функция `bagEqual` берет в качестве аргументов два мультимножества и возвращает `True`, если мультимножества равны (т. е. имеют одни и те же элементы и одинаковое вхождение соответствующих элементов) и `False` в противном случае.
- Функция `bagSubbag` берет в качестве аргументов два мультимножества и возвращает `True`, если первое мультимножество есть подмультимножество второго мультимножества и `False` в противном случае. X есть подмультимножество Y , если каждый элемент X встречается в Y и имеет в Y не меньше вхождений, чем в X .
- Функция `bagUnion` берет в качестве аргументов два мультимножества и возвращает объединение мультимножеств. Объединение мультимножеств X и Y содержит все элементы, которые встречаются в X или в Y , причем количество вхождений элемента в мультимножество равно максимуму вхождений в X и в Y .
- Функция `bagIntersect` берет в качестве аргументов два мультимножества и возвращает пересечение мультимножеств. Пересечение мультимножеств X и Y содержит все эле-

менты, которые встречаются в X и в Y , причем количество вхождений элемента в мультимножество равно минимуму вхождений в X и в Y .

- Функция `BagSum` берет в качестве аргументов два мультимножества и возвращает сумму мультимножеств. Сумма мультимножеств X и Y содержит все элементы, которые встречаются в X или в Y , причем количество вхождений элемента в мультимножество равно сумме вхождений в X и в Y .
- Функция `bagDiff` берет в качестве аргументов два мультимножества и возвращает разность мультимножеств, первый аргумент минус второй. Разность мультимножеств X и Y содержит все элементы из X , что содержатся в Y с меньшим числом вхождений. Количество вхождений любого элемента в разность равно числу вхождений элемента в X минус число вхождений элемента в Y .
- Функция `bagInsert` берет в качестве аргументов мультимножество и элемент и возвращает мультимножество со вставленным элементом. Вставка элемента или добавляет единственное вхождение нового элемента в мультимножество или увеличивает на 1 число вхождений уже присутствующего элемента.
- Функция `bagDelete` берет в качестве аргументов мультимножество и элемент и возвращает мультимножество с удаленным элементом. Удаление элемента или уничтожает единственное вхождение элемента в мультимножество или уменьшает на 1 число вхождений уже присутствующего элемента.

22. Модуль для последовательностей (14)

Программирование на языке Haskell.

Рассмотрим следующий тип данных для представления последовательностей (т. е. списков)

```
data Seq a = Nil | Att (Seq a) a
```

Конструктор `Nil` представляет пустую последовательность. Конструктор `Att xz y` представляет последовательность, в которой «последний элемент» добавлен справа к начальной последовательности `xz`. Заметим, что `Att` подобно обыкновенному «:» для списков, за исключением того, что элемент добавляется в противоположный конец последовательности. `Att (Att (Att Nil 1) 2) 3` представляет ту же самую последовательность, что и обыкновенный список `1 : (2 : (3: []))`.

Напишите модуль, в котором реализуйте следующие операции для данного типа (операции есть аналоги подобных операций над встроенным типом функций):

- Функция `lastSeq` – для непустой последовательности `Seq` возвращает последний элемент;
- Функция `initialSeq` – для непустой последовательности `Seq` возвращает начальную последовательность (исходная последовательность без последнего элемента)⁴
- Функция `lenSeq` – возвращает длину последовательности;
- Функция `headSeq` – для непустой последовательности `Seq` возвращает ее голову (самый левый элемент);
- Функция `tailSeq` – для непустой последовательности `Seq` возвращает последовательность без первого элемента;
- Функция `conSeq x xs, conSeq :: a -> Seq a -> Seq a`, – возвращает последовательность, где первым элементом является `x` и хвостом `xs`.
- Функция `appSeq xs ys, appSeq :: Seq a -> Seq a -> Seq a`, – соединение двух последовательностей.
- Функция `revSeq :: Seq a -> Seq a` – обращение последовательности (аналог `reverse`);

- Функция $\text{filterSeq} :: (a \rightarrow \text{Bool}) \rightarrow \text{Seq } a \rightarrow \text{Seq } a$ – фильтрация последовательности (аналог `filter`);
- Функция $\text{mapSeq} :: (a \rightarrow b) \rightarrow \text{Seq } a \rightarrow \text{Seq } b$ – аналог функции `map`;
- Функция `listToSeq` – преобразования из списка последовательности, с теми же элементами и тем же порядком элементов; должно выполняться $\text{HeadSeq} (\text{listToSeq } xs) = \text{head } xs$, если `xs` – не пустой список.
- Функция `seqToList` – преобразования из последовательности списка, с теми же элементами и тем же порядком элементов; должно выполняться $\text{head} (\text{seqToList } xs) = \text{headSeq } xs$, если `xs` – не пустая последовательность.

23. *Миры Р. Смаллиана: «рыцари и лжецы» (Haskell)(15)*

В книге Р. Смаллиана «Как же называется эта книга?» приводится много логических задач, в которых персонажи высказывают некоторые автореферентные утверждения, прямо или косвенно говорящие об истинности этих утверждений. Персонажи этих задач населяют некоторый остров и относятся к одной из двух категорий: «рыцари», которые говорят только правду, и «лжецы», изрекающие только ложь. Предполагается, что каждый обитатель острова либо рыцарь, либо лжец. В статье [1] предлагаются метод решения некоторых задач с автореферентными рассуждениями с помощью программирования на языке Пролог.

Напишите программу на языке Haskell, использующую подход из статьи [1] и решающую задачи 1-8 из указанной статьи.

Литература:

1. Зюзьков В. М. Моделирование автореферентных рассуждений с помощью логического программирования // Доклады ТУСУР. Том 7. Автоматизированные системы обработки информации, управления и проектирования. Сборник научных трудов. Томск, Изд-во ТУСУР, 2002, с. 113-122.

24. *Миры Р. Смаллиана: обобщения задач с рыцарями и лжецами (Пролог) (14)*

В книге Р. Смаллиана «Как же называется эта книга?» приводится много логических задач, в которых персонажи высказывают некоторые автореферентные утверждения, прямо или косвенно говорящие об истинности этих утверждений. Персонажи этих задач населяют некоторый остров и относятся к одной из двух категорий: «рыцари», которые говорят только правду, и «лжецы», изрекающие только ложь. Предполагается, что каждый обитатель острова либо рыцарь, либо лжец. В статье [1] предлагаются метод решения некоторых задач с автореферентными рассуждениями с помощью программирования на языке Пролог.

В статье указываются возможные обобщения «мира»: на острове, кроме рыцарей и лжецов живут также «нормальные люди», которые могут говорить как правду, так и ложь.

Кроме того, возможны включение в постановку задачи высказывания о высказывании (назовем его *метавысказыванием*). Как это можно сделать также описывается в статье.

Напишите программу на языке Пролог, реализующую предложенные обобщения (остров населен рыцарями, лжецами и нормальными людьми и возможны метавысказывания) и решающую задачи 1-10 из указанной статьи.

Литература:

1. Зюзьков В. М. Моделирование автореферентных рассуждений с помощью логического программирования // Доклады ТУСУР. Том 7. Автоматизированные системы обработки

25. **Миры Р. Смаллиана: тайна шкатулок Порции (Пролог)** (14)

В книге Р. Смаллиана «Как же называется эта книга?» приводится много логических задач, в которых персонажи высказывают некоторые автореферентные утверждения, прямо или косвенно говорящие об истинности этих утверждений. Персонажи этих задач населяют некоторый остров и относятся к одной из двух категорий: «рыцари», которые говорят только правду, и «лжецы», изрекающие только ложь. Предполагается, что каждый обитатель острова либо рыцарь, либо лжец. В статье [1] предлагаются метод решения некоторых задач с автореферентными рассуждениями с помощью программирования на языке Пролог.

Рассмотрим задачи Р. Смаллиана с автореферентными утверждениями другого типа. В них речь идет о надписях на шкатулках, и эти надписи могут сообщать что-то о своей собственной истинности – вот в этом и заключается автореферентность.

Тайна шкатулок Порции

История первая

Задача 1

У Порции из комедии Шекспира «Венецианский купец» было три шкатулки: из золота, серебра и свинца. В одной из шкатулок хранился портрет Порции. Поклоннику предлагалось выбрать шкатулку, и если он был достаточно удачлив (или достаточно умен), чтобы выбрать шкатулку с портретом, то получал право назвать Порцию своей невестой. На крышке каждой шкатулки была сделана надпись, которая должна была помочь претенденту на руку и сердце Порции выбрать «правильную» шкатулку.

Предположим, что Порция вздумала выбирать мужа не по добродетелям, а по уму. На крышках шкатулок она приказала сделать следующие надписи:

На золотой	На серебряной	На свинцовой
Портрет в этой шкатулке	Портрет не в этой шкатулке	Портрет не в золотой шкатулке

Своему поклоннику Порция пояснила, что из трех высказываний, выгравированных на крышках шкатулок только одно истинно.

Какую шкатулку следует выбрать поклоннику Порции?

Задача 2

Поклонник Порции правильно выбрал шкатулку, они поженились и жили счастливо (по крайней мере, первое время). Но однажды Порции пришли в голову следующие мысли: «Хотя мой муж, выбрав шкатулку с моим портретом, проявил в известной мере ум, но в действительности задача была не такой уж трудной. Мне следовало бы придумать какую-нибудь задачку потруднее. Тогда у меня был бы действительно умный муж». Порция развелась со своим мужем и решила подыскать себе супруга поумнее.

На этот раз она приказала выгравировать на крышках шкатулок следующие надписи:

На золотой	На серебряной	На свинцовой
Портрет не в серебряной шкатулке	Портрет не в этой шкатулке	Портрет в этой шкатулке

Своему поклоннику Порция пояснила, что из трех высказываний, выгравированных на крышках шкатулок, по крайней мере, одно истинно и по крайней мере одно ложно. В какой шкатулке хранится портрет Порции?

Эпилог

Волею судеб удачливым претендентом на руку Порции оказался бывший муж. Будучи человеком умным, он сумел решить и вторую задачу. Они вновь поженились. Прямо из-под венца супруг привез Порцию в их дом, положил себе на колени, закатил ей изрядную порку, и Порция навсегда избавилась от глупостей.

История вторая

Вступив вторично в брак, Порция и её муж зажили счастливо. У них родилась дочь, Порция Вторая, которую мы в дальнейшем будем для краткости называть просто Порцией. Когда юная Порция подросла, она стала необычайно умной и красивой девушкой, совсем как её мама. Она также вздумала выбирать себе мужа «по методу шкатулок». Чтобы получить Порцию в жены, претендент на ее руку должен был пройти два испытания.

Задача 3

Во время этого испытания на крышке каждой шкатулки было выгравировано по две надписи. Порция пояснила, что на каждой крышке ложно не более чем одно высказывание.

На золотой	На серебряной	На свинцовой
1) Портрет не в этой шкатулке 2) Портрет написан художником из Венеции	1) Портрет не в золотой шкатулке 2) Портрет в действительности написан художником из Флоренции	1) Портрет не в этой шкатулке 2) В действительности портрет в серебряной шкатулке

В какой шкатулке находится портрет?

Задача 4

Если претендент на руку Порции проходил первое испытание, то его вели в другую комнату, посреди которой на столе были расставлены три другие шкатулки. На крышке каждой из них было выгравировано по две надписи. Порция пояснила, что на крышке одной шкатулки оба высказывания истинны, на крышке другой шкатулки оба высказывания ложны, а на крышке третьей шкатулки одно высказывание истинно и одно ложно.

На золотой	На серебряной	На свинцовой
1) Портрет не в этой шкатулке 2) Портрет в серебряной шкатулке	1) Портрет не в золотой шкатулке 2) Портрет в свинцовой шкатулке	1) Портрет не в этой шкатулке 2) Портрет в золотой шкатулке

В какой шкатулке находится портрет?

Напишите программу на языке Пролог, решающую подобные задачи, в частности, она должна уметь решать задачи 1–4.

Литература:

1. Зюзьков В. М. Моделирование автореферентных рассуждений с помощью логического программирования // Доклады ТУСУР. Том 7. Автоматизированные системы обработки

26. *Миры Р. Смаллиана: Лев и Единорог (Пролог) (14)*

В книге Р. Смаллиана «Как же называется эта книга?» приводится много логических задач, в которых персонажи высказывают некоторые автореферентные утверждения, прямо или косвенно говорящие об истинности этих утверждений. Персонажи этих задач населяют некоторый остров и относятся к одной из двух категорий: «рыцари», которые говорят только правду, и «лжецы», изрекающие только ложь. Предполагается, что каждый обитатель острова либо рыцарь, либо лжец. В статье [1] предлагаются метод решения некоторых задач с автореферентными рассуждениями с помощью программирования на языке Пролог.

Рассмотрим задачи Р. Смаллиана с автореферентными утверждениями другого типа.

Когда Алиса вошла в Лес Забывчивости, она забыла не все, а лишь кое-что. Она часто забывала, как ее зовут, но особенно ей легко удавалось забывать дни недели. Лев и Единорог частенько наведывались в Лес Забывчивости. Странные это были существа. Лев лгал по понедельникам, вторникам и средам и говорил правду во все остальные дни недели. Единорог же вел себя иначе: он лгал по четвергам, пятницам и субботам и говорил правду во все остальные дни недели.

Задача 1

Однажды Алиса повстречала Льва и Единорога, отдохавших под деревом. Те высказали следующие утверждения.

Лев. Вчера был один из дней, когда я лгу.

Единорог. Вчера был один из дней, когда я тоже лгу.

Из этих двух высказываний Алиса (девочка очень умная) сумела вывести, какой день недели был вчера. Что это был за день?

Задача 2

В другой раз Алиса повстречала одного Льва. Он высказал два утверждения:

1) Я лгал вчера.

2) После завтрашнего дня я буду лгать два дня подряд.

В какой день недели Алиса встретила Льва?

Задача 3

В какие дни недели Лев может высказать следующие утверждения:

1) Я лгал вчера.

2) Я буду лгать завтра.

Задача 4

В какие дни недели Лев может высказать следующее единое утверждение: «Я лгал вчера, и я буду лгать завтра». Предостережение! Ответ этой задачи не совпадает с ответом предыдущей задачи.

Напишите программу на языке Пролог, решающую подобные задачи, в частности, она должна уметь решать задачи 1–4.

Литература:

1. Зюльков В. М. Моделирование автореферентных рассуждений с помощью логического программирования // Доклады ТУСУР. Том 7. Автоматизированные системы обработки информации, управления и проектирования. Сборник научных трудов. Томск, Изд-во ТУСУР, 2002, с. 113-122.

27. *Миры Р. Смаллиана: Лев и Единорог (Haskell) (15)*

Задание такое же, как предыдущей теме, но язык реализации – Haskell.

28. *Миры Р. Смаллиана: Траляля и Труляля (Пролог) (14)*

В книге Р. Смаллиана «Как же называется эта книга?» приводится много логических задач, в которых персонажи высказывают некоторые автореферентные утверждения, прямо или косвенно говорящие об истинности этих утверждений. Персонажи этих задач населяют некоторый остров и относятся к одной из двух категорий: «рыцари», которые говорят только правду, и «лжецы», изрекающие только ложь. Предполагается, что каждый обитатель острова либо рыцарь, либо лжец. В статье [1] предлагаются метод решения некоторых задач с автореферентными рассуждениями с помощью программирования на языке Пролог.

Рассмотрим задачи Р. Смаллиана с автореферентными утверждениями другого типа.

Когда Алиса вошла в Лес Забывчивости, она забыла не все, а лишь кое-что. Она часто забывала, как ее зовут, но особенно ей легко удавалось забывать дни недели. Траляля и Труляля частенько навещали в Лес Забывчивости. Странные это были существа. Один из них лгал по понедельникам, вторникам и средам и говорил правду во все остальные дни недели. Другой лгал по четвергам, пятницам и субботам, но во все остальные дни недели говорил правду. Но Алиса не знала, кто из них ведет себя так или иначе. К тому же братья были так похожи друг на друга, что Алиса даже не могла различить их (воротнички, на которых были вышиты их имена, братья одевали очень редко). Бедняжке Алисе приходилось очень туго! Взять хотя бы следующие случаи.

Задача 1

Однажды Алиса встретила обоих братьев вместе, и они высказали следующие утверждения:

Первый. Я Траляля.

Второй. Я Труляля.

Кто из них в действительности был Траляля и кто – Труляля?

Задача 2

В другой день той же недели братцы высказали следующие утверждения:

Первый. Я Траляля.

Второй. Если это так, то я Труляля!

Кто из них Траляля и кто Труляля?

Задача 3

Как-то Алиса встретила обоих братцев и спросила у одного из них: «Вы лжете по воскресеньям?» Тот ответил: «Да!» Тогда она задала тот же вопрос другому братцу. Что он ответил?

Задача 4

В другой раз братья заявили следующее:

Первый. 1) Я лгу по субботам.

2) Я лгу по воскресеньям.

Второй. Я буду лгать завтра.

В какой из дней недели это было?

Задача 5

Однажды Алиса встретила одного из братцев. Он заявил следующее: «Я лгу сегодня или я Труляля». Можно было бы в этом случае определить, кто из братьев это был?

Задача 6

Предположим, что встреченный Алисой братец заявил: «Я лгу сегодня или я Труляля». Можно было бы в этом случае определить, кто из братьев это был?

Задача 7

Однажды Алиса встретила обоих братцев вместе. Они высказали следующие утверждения.

Первый. Если я Траляля, то он Труляля.

Второй. Если он Труляля, то я Траляля.

Можно ли определить, кто из братцев Траляля и кто Труляля? Можно ли определить, что это был за день недели?

Напишите программу на языке Пролог, решающую подобные задачи, в частности, она должна уметь решать задачи 1–7.

Литература:

1. Зюсков В. М. Моделирование автореферентных рассуждений с помощью логического программирования // Доклады ТУСУР. Том 7. Автоматизированные системы обработки информации, управления и проектирования. Сборник научных трудов. Томск, Изд-во ТУСУР, 2002, с. 113-122.

29. **Миры Р. Смаллиана: Траляля и Труляля (Haskell) (15)**

Задание такое же, как предыдущей теме, но язык реализации – Haskell.

30. **Пропозициональная логика (Haskell) (15)**

Определим тип данных PropStat, представляющий высказывания:

```
type Name = Char

data PropStat = Simple Name
              | And    PropStat PropStat
              | Or     PropStat PropStat
              | Not     PropStat
              | Implies PropStat PropStat
              | Equiv   PropStat PropStat
              deriving Show

-- p && q
-- p || q
-- not p
-- p -> q
-- p <-> q (or p == q)
```

Задача: создайте модуль PropLogic.hs, который должен содержать функции описанные ниже.

1. Определите тип окружение, с помощью которого задается конкретная интерпретация переменных.

```
type Environment a b = [(a, b)]
```

Определите функцию

```
allEnvs :: [Name] -> [Environment Name Bool]
```

Функция allEnvs xs возвращает список всех возможных интерпретаций переменных из списка.

Например,

```
allEnvs ['a', 'b'] =>
[(['a', True), ('b', True)], [(['a', True), ('b', False)], [(['a', False), ('b', True)],
[(['a', False), ('b', False)]]
```

2 . Определите функцию `stat :: Int -> PropStat`, которая по данному целому числу возвращает некоторое высказывание. Функция `stat` будет полезна для тестирования ваших функций. Определите следующее:

```
stat 1 = c ~ d
stat 2 = a ⊃ (a ∨ b)
stat 3 = (c ⊃ d) ~ (d ⊃ c)
stat 4 = ¬ (¬ a && ¬ c) ~ (¬ c ⊃ a)
```

Заметим, что правые стороны этих уравнений написаны в привычной логической форме: они должны быть заменены на соответствующие выражения типа `PropStat`.

3 . Определите функцию `eval :: PropStat -> Environment Name Bool -> Bool`, которая берет высказывание `p` и интерпретацию `e`, и оценивает `p` в интерпретации `e`.

Например, `eval (stat 3) [('c', True), ('d', False)] ⇒ False`, и
`eval (stat 3) [('c', False), ('d', False)] ⇒ True`.

Заметим, что $p \supset p' \equiv \neg p \vee p'$.

4 . Определите функцию `vars :: PropStat -> [Name]`, которая возвращает список всех переменных без дубликатов, содержащихся в высказывании. (Подсказка: Определите рекурсивную функцию `allvars`, возвращающую список переменных с дубликатами и воспользуйтесь функцией `nub` из библиотеки `List`.)

Например, `vars (stat 3) ⇒ "cd"`.

5 . Высказывание `p` называется тавтологией, если `p` есть истина при любой интерпретации переменных. Определите функцию, проверяющую является ли высказывание тавтологией: `tautology :: PropStat -> Bool`

Например, `tautology (stat 2) ⇒ True`, `tautology (stat 3) ⇒ False`, и
`tautology (stat 4) ⇒ True`.

6 . Пусть равенство высказываний понимается как логическая эквивалентность. Для определения равенства для высказываний введите декларацию экземпляра в виде

```
instance Eq PropStat where
  p == p' = ...
```

Например, `stat 1 == stat 3`, но `stat 3 /= stat 4`.

7 . Определите функцию `showPropStat :: PropStat -> String`, которая берет высказывание и возвращает строку, представляющую это высказывание. Заключайте все подвыражения в скобки.

Например, `showPropStat (stat 3) ⇒ "((c -> d) <-> (d -> c))"`

8 . Все высказывания могут быть переписаны к виду, содержащему только операции \supset и \neg . Справедливы следующие правила:

```
p & p' ≡ ¬ (¬p ∨ ¬p')
p ∨ p' ≡ ¬ p ⊃ p'
p ~ p' ≡ (p ⊃ p') & (p' ⊃ p)
```

Используя эти правила рекурсивно, определите функцию

```
transform :: PropStat -> PropStat,
```

которая переписывает высказывание в эквивалентную (\supset , \neg)-форму.

Например:

```
showPropStat (transform (stat 2)) ⇒ "(a -> ((not a) -> b))"
showPropStat (transform (stat 3)) ⇒
"(not ((not (not ((c -> d) -> (d -> c)))) -> (not ((d -> c) -> (c -> d)))))"
```


9. Напишите функцию `insertNeg`, которая приводит формулу к виду с тесными отрицаниями: `insertNeg w` выдает логическую формулу, получающуюся из логической формулы `w` внесением всех операторов отрицания внутрь конъюнкций и дизъюнкций.

Пример:

```
> insertNeg (Not (And (Var 'x') (Or (Var 'y') (Not (Var 'x')))))
```

```
Or (Not (Var 'x')) (And (Not (Var 'y')) (Var 'x')))
```

В логических обозначениях: $\neg(x \& (y \vee \neg x)) \Rightarrow \neg x \vee (\neg y \& x)$

31. Детективные головоломки (15)

Рассмотрим задачу:

Известны следующие факты:

1. Если A виновен и B не виновен, то C виновен.
2. C никогда не действует в одиночку.
3. A никогда не ходит на дело вместе с C .
4. Никто, кроме A , B и C , в преступлении не замешан, и, по крайней мере, один из этой тройки виновен.

Полностью доказать, кто виновен, а кто не виновен, из этих фактов не получится, но чтобы выдвинуть неопровержимое обвинение против одного из них, материала вполне достаточно. Условие этой задачи можно представить в виде формул логики высказываний:

$$A \& \neg B \supset C$$
$$C \supset (A \& C \vee B \& C)$$
$$\neg(A \& C)$$
$$A \vee B \vee C$$

Поскольку мы предполагаем, что известные факты являются истинами, то данные формулы должны быть истинными.

Полезно использовать следующую терминологию. Имеются различные миры, содержащие персонажей (в данном случае, A , B и C). Каждый персонаж может быть виновным (значение переменной – истина) или невиновным (значением переменных – ложь). Если в задаче участвуют n персонажей, то всего существуют 2^n различных гипотетических миров. Мы выбираем все миры, в которых данные формулы выполнимы. Если во всех этих мирах какая-то переменная истинна, то персонаж, именуемый данной переменной, является преступником. В терминах математической логики искомый мир должен быть моделью множества высказанных утверждений. Значит, алгоритм решения подобных задач состоит в поиске всех моделей, и в выборе тех переменных, которые всегда истинны.

Задание: написать программу на SWI-Prolog'e, решающую задачи данного вида.

Как представлять условие задачи? Формулы удобно записывать в виде списка термов, используя операторы для отрицания, конъюнкции, дизъюнкции и отрицания:

```
:- op( 100, fy, ~).
:- op( 110, xfy, &).
:- op( 120, xfy, v).
:- op( 130, xfy, =>).
```

Например, вышеприведенные формулы кодируются так:

```
[a & ~b => c, c => a & b v b & c, ~(a & c), a v b v c]
```

В программе понадобится генерировать список всех миров, уметь вычислять формулы при заданной интерпретации переменной и отбирать из миров модели. Задачу можно описать так:

```
задача(1, [a => ~ c, c => a v b, a & ~ b => c]).
```

а запрос может быть таким:

?- решение(1) .

Виновны: [b]

Yes

Приведем условия тестовых задач.

Задача 2

Мистер Макгрегор, владелец лавки из Лондона, сообщил по телефону в Скотланд-Ярд о том, что его ограбили. Трех преступников-рецидивистов A , B и C , подозреваемых в ограблении, вызвали на допрос. Установлено следующее:

1. Каждый из тройки подозреваемых A , B и C в день ограбления побывал в лавке, и никто больше в тот день в лавку не заходил.
2. Если A виновен, то у него был ровно один сообщник.
3. Если B не виновен, то C также не виновен.
4. Если виновны ровно двое подозреваемых, то A – один из них.
5. Если C не виновен, то B также не виновен.

Против кого надо выдвинуть обвинение?

Задача 3

На этот раз на допрос были вызваны четверо подозреваемых в ограблении: A , B , C и D . Неопровержимыми уликами доказано, что, по крайней мере, один из них виновен и что никто, кроме A , B , C и D , в ограблении не участвовал. Кроме того, удалось установить следующее:

1. A безусловно не виновен.
2. Если B виновен, то у него был ровно один соучастник.
3. Если C виновен, то у него было ровно два соучастника.

Особенно важно узнать, виновен или не виновен D , так как D был опасным преступником. К счастью, приведенных выше фактов достаточно, чтобы установить виновность или невиновность подозреваемого D . Итак, виновен или не виновен D ?

Задача 4

По обвинению в ограблении перед судом предстали A , B и C . Установлено следующее:

1. Если A не виновен или B виновен, то C виновен.
2. Если A не виновен, то C не виновен.

Можно ли на основании этих данных установить виновность каждого из трех подсудимых?

Задача 5

По обвинению в ограблении перед судом предстали A , B и C . Установлено следующее:

1. По крайней мере, один из трех подсудимых виновен.
2. Если A виновен и B не виновен, то C виновен.

Этих данных недостаточно, чтобы доказать виновность каждого из трех подсудимых в отдельности, но эти же данные позволяют отобрать двух подсудимых, о которых известно, что один из них заведомо виновен. О каких двух подсудимых идет речь?

Задача 6

Подсудимых четверо A , B , C и D . Установлено следующее:

1. Если A и B оба виновны, то C был соучастником.

2. Если A виновен, то по крайней мере один из обвиняемых B или C был соучастником.
3. Если C виновен, то D был соучастником.
4. Если A не виновен, то D виновен.

Кто из четырех подсудимых виновен вне всякого сомнения и чья вина остается под сомнением?

Задача 7

Подсудимых четверо A, B, C и D . Установлено следующее:

1. Если A виновен, то B был соучастником.
2. Если B виновен, то либо C был соучастником, либо A не виновен.
3. Если D не виновен, то A виновен и C не виновен.
4. Если D виновен, то A виновен.

Кто из подсудимых виновен и кто не виновен?

32. Построение самоссылочных утверждений о количестве цифр (15)

Рассмотрим утверждение A_0 :

«В предложении B цифра «ноль» содержится 1 раз(а); цифра «один» – 2 раз(а); цифра «два» – 3 раз(а); цифра «три» – 10 раз(а); цифра «четыре» – 0 раз(а); цифра «пять» – 12 раз(а); цифра «шесть» – 4 раз(а); цифра «семь» – 1 раз(а); цифра «восемь» – 1 раз(а); цифра «девять» – 2 раз(а)».

Мы можем создавать различные утверждения, подобные A_0 , если вместо указанных чисел (выделенных полужирным шрифтом) в A_0 подставлять другие натуральные числа. Все такие утверждения будем называть «цифровыми предложениями». Будем использовать также оборот речи «предложение A_0 говорит о предложении B ». Легко построить цифровое предложение B такое, что высказывание A_0 будет истинным.

Оказывается справедливо следующее:

- 1) существует такое цифровое предложение, что оно истинно и говорит о себе;
- 2) существуют такие цифровые различные предложения A и B , что они оба истинны, причем предложение A говорит о B , а предложение B говорит о A .

Цифровые предложения, удовлетворяющие свойству 1 называются (прямо) **самоссылочными**, а предложения со свойством 2 – **косвенно самоссылочными**.

Ваша цель – написать программу на SWI-Prolog’е, которая строит самоссылочные или косвенно самоссылочные предложения.

Можно использовать следующий алгоритм:

Подсчитаем количество различных цифр в A_0 и на основании этого строим истинное цифровое предложение A_1 , говорящее о A_0 :

«В предложении A_0 цифра «ноль» содержится 2 раз(а); цифра «один» – 5 раз(а); цифра «два» – 3 раз(а); цифра «три» – 1 раз(а); цифра «четыре» – 1 раз(а); цифра «пять» – 0 раз(а); цифра «шесть» – 0 раз(а); цифра «семь» – 0 раз(а); цифра «восемь» – 0 раз(а); цифра «девять» – 0 раз(а)».

Далее этот процесс повторяем, строя истинные цифровые предложения A_1, A_2, A_3, \dots , причем для любого k предложение A_k говорит о A_{k-1} .

Доказано, что независимо от того, с какого цифрового предложения A_0 начинаем, последовательность A_1, A_2, A_3, \dots становится периодической с некоторым периодом n (тогда предложения A_k и A_{k+n} косвенно самоссылочны), либо с какого-то момента все предложения совпадают (тогда предложение A_k самоссылочно).

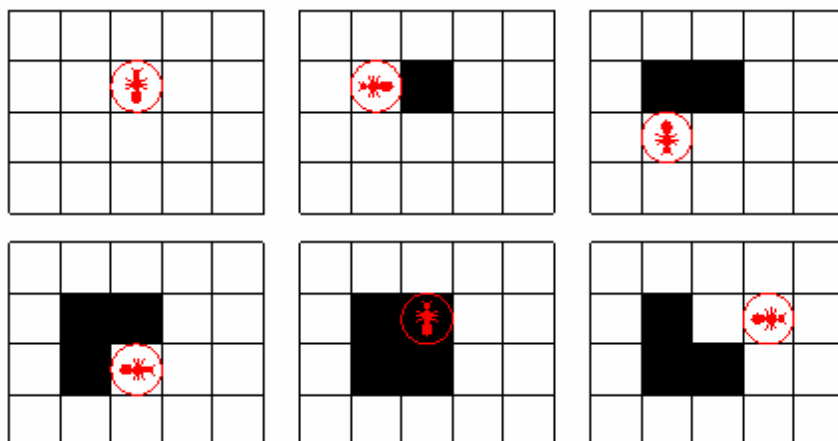
При программировании на Прологе цифровые предложения можно кодировать списками, например, для утверждения A_0 имеем список [1,2,3,10,0,12,4,1,1,2].

33. Муравей Лэнгтона (15)

Муравей Лэнгтона (Ant Christopher Langton's) – простая форма искусственной жизни – двумерная машина Тьюринга с 4-я состояниями изобретена в 1980-х годах.

Муравей начинает "жизнь" на бесконечной решетке, состоящей из черных и белых клеток, и его поведение следует трем правилам:

1. Если муравей находится на черной клетке, то он поворачивается вправо на 90 градусов и передвигается вперед на одну клетку.
2. Если муравей находится на белой клетке, то он поворачивается влево на 90 градусов и передвигается вперед на одну клетку.
3. Когда муравей покидает клетку, ее цвет меняется на противоположный.

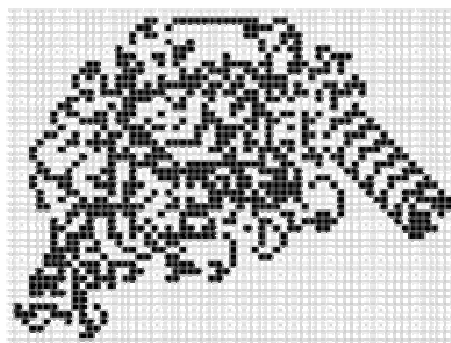


Когда муравей начинает на пустой решетке, то он с какого-то момента начинает строить диагональное «скоростное шоссе», выполняя бесконечно цикл из 104 шага, состоящий практически из «топтанья на месте».

На картинке слева состояние решетки после 386 шагов.

На картинке справа после 10647 шагов «шоссе» уже создано по направлению к нижнему правому углу.

Доказано, что «шоссе» будет строиться неограниченно. Существует гипотеза, что строительство «шоссе» будет происходить при любой начальной конфигурации клеток (но, возможно, придется ждать начало строительства очень долго). Гипотеза подтверждается многочисленными экспериментами.



Написать программу, моделирующую поведения муравья Лэнгтона. Язык программирования – любой.

34. Моделирование эпидемий (14)

Рассмотрим двухмерный клеточный автомат, в котором каждая клетка имеет 8 соседей. Клетка представляет одну особь в популяции и может быть "здоровой", "больной" или "здоровой с иммунитетом". Предполагаются следующие правила распространения эпидемий:

- здоровый организм может с некоторой вероятностью заболеть при больном соседе;
- больной организм по истечении некоторого времени выздоравливает и приобретает иммунитет;
- организм с иммунитетом заболеть не может;
- иммунитет с течением времени с некоторой вероятностью ослабляется вплоть до полного исчезновения.

В исходную полностью здоровую популяцию помещается один больной организм. Будем изучать распространение болезни при таких начальных условиях.

Состояние клетки задается целым числом s . Предлагается следующая интерпретация значения s :

- $s = 0$ – здоровый организм, не имеет иммунитета;
- $s < 0$ – организм здоровый и имеет иммунитет (степень иммунитета прямо пропорциональна абсолютному значению s);
- $s > 0$ – больной организм (величина s указывает продолжительность болезни).

Пусть болезнь определяется следующими параметрами: p – вероятность заболеть при больном соседе; q – вероятность ослабления иммунитета за единицу времени; ti – длительность болезни (у всех организмов болезнь продолжается одинаковое время); tv – длительность иммунитета в единицах времени.

Законы изменения состояния:

- 1) $s = 0 \rightarrow s := 0$ – если нет больных соседей (здоровый организм остается здоровым при здоровых соседях);
- 2) $s = 0 \rightarrow s := 1$ с вероятностью p – если есть больной сосед (здоровый организм без иммунитета может заболеть при больном соседе);
- 3) $0 < s < ti \rightarrow s := s+1$ – больной продолжает болеть определенное время;
- 4) $s = ti \rightarrow s := -tv$ – если больной достаточно долго проболел, то он выздоравливает и приобретает иммунитет;
- 5) $s < 0 \rightarrow s := s+1$ с вероятностью q – иммунитет со временем может ослабнуть.

Будем предполагать, что популяция является замкнутой, но неограниченной системой. Устраним границы поля: соединим любые два противоположных края поля, затем концы полученного цилиндра. Тем самым мы рассматриваем исходное поле как тор. Определим начальную конфигурацию – все клетки «здоровы», за исключением центральной «больной» клетки.

Напишите программу, моделирующую поведение данного клеточного автомата. Программа должна позволять пользователю менять параметры: p – вероятность заболеть при больном соседе; q – вероятность ослабления иммунитета за единицу времени; ti – длительность болезни; tv – длительность иммунитета в единицах времени.

Язык программирования любой.

Литература:

Зюжков В. М. Синергетика для программистов, Томск, ТУСУР, 2003 г.

35. Моделирование пожара (14)

Создайте с помощью клеточного автомата модель изменения растительности в лесу при пожарах.

Двумерная решетка представляет клетки с различными видами растительности. Состояние одной клетки описывается в виде пары (s, t) . Величина t – время, прошедшее

с тех пор, как на этой клетке был огонь, или, если огня никогда не было, то время с момента запуска клеточного автомата. Значение s есть один из следующих символов: e – «огонь», b – «выгоревшая земля», g – «трава», w – «редкий лес», f – «густой лес». Начальное состояние: $t = 0$ для всех клеток; все клетки, за исключением центральной, на которой огонь, получают случайно одно из значений g, w, f .

Правила перехода:

$(s, t) \rightarrow (e, 0)$, если горит соседняя клетка, и s – не выжженная земля (переход совершается с некоторой вероятностью, заданной как параметр);

иначе

$(e, 0) \rightarrow (b, 1)$,

$(b, 4) \rightarrow (g, 5)$,

$(g, 9) \rightarrow (w, 10)$,

$(w, 49) \rightarrow (f, 50)$;

иначе

$(s, t) \rightarrow (s, t + 1)$.

Язык программирования любой.

Литература:

Зюзьков В. М. Синергетика для программистов, Томск, ТУСУР, 2003 г.

36. Двумерная диффузия (12)

Напишите программу для двумерного клеточного автомата, моделирующего процесс диффузии.



a



б

Рис. Постепенная диффузия плотного кластера частиц: (*a*) исходное состояние; (*б*) диффузия после 30 шагов

Язык программирования любой.

Литература:

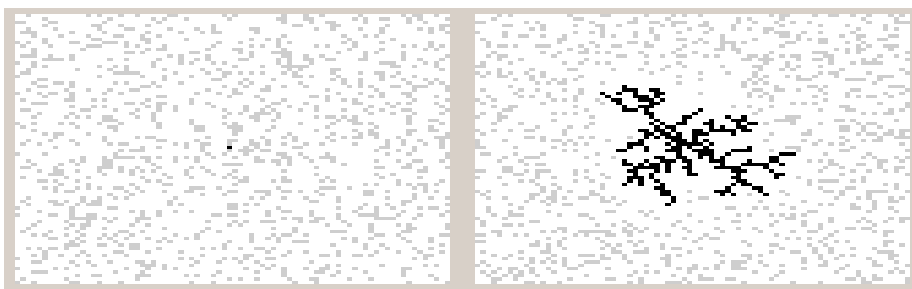
Зюзьков В. М. Синергетика для программистов, Томск, ТУСУР, 2003 г. (раздел 4.4)

37. Агрегирование ограниченное диффузией (13)

Напишите программу для двумерного клеточного автомата, моделирующего процесс ограниченного диффузией агрегирования.

Язык программирования любой.

На рис. изображен дендритный рост за счет ограниченного диффузией агрегирования: (*a*) исходное состояние; (*б*) дендрит после 20 шагов.



a

б

Литература:

Зюзьков В. М. Синергетика для программистов, Томск, ТУСУР, 2003 г. (раздел 4.4)

Балльная раскладка рейтинга

Элементы учебной деятельности	Максимальн. балл на 1-ую КТ с начала семестра	Максимальн. балл за период между 1КТ и 2КТ	Максимальн. балл за период между 2КТ и на конец семестра	Всего за семестр
Выбор темы курсовой работы (баллы за сложность)	15			15
Подбор и обзор литературы	2			2
Выбор алгоритма и структур данных	8			8
Написание программы		15		15
Тестирование программы и получение результатов			10	10
Полное оформление работы			10	10
Компонент своевременности	4	4	2	10
Итого максимум за период	29	19	22	70
Защита работы (максимум)				30
Нарастающим итогом	29	48	70	100