Решение логических задач.

Лекция 5 (Часть 1).

Метод “Образуй и проверь”.

Специальности : 230105, 010501

Недетерминированное программирование.

<u>Определение.</u> Недетерминизм – техническое понятие, используемое для сжатого определения абстрактных моделей вычислений.

Недетерминированная машина, перед которой возникло несколько альтернативных путей решения, осуществляет корректный выбор очередного действия, в частности, с применением механизма последовательного поиска и возвратов.

Следует отметить тот факт, что недетерминизм только моделируется, не присутствуя реально.

Примером недетерминированного программирования может послужить метод “образуй и проверь”. В этом методе подпрограмма-генератор строит альтернативные пути решения задачи, а подпрограмма-контроллер осуществляет конкретный выбор из ряда альтернатив путем проверки корректности сделанного генератором предположения.

Суть метода “Образуй и проверь”.

<u>Определение.</u> “Образуй и проверь” – общий прием проектирования алгоритмов и программ. Суть его состоит в том, что один процесс или программа генерирует множество предполагаемых решений задачи, а другой процесс или программа проверяет эти предполагаемые решения и пытается найти те из них, которые действительно являются решениями задачи.

Обычно программы, реализующие метод “образуй и проверь”, содержат конъюнкцию двух целей, одна из которых действует как генератор предполагаемых решений, а вторая проверяет допустимость полученных решений :

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

Если проверка *test(X)* завершается отказом, то производится возвращение к цели *generate*, с помощью которой генерируется следующий элемент. Процесс продолжается итерационно до тех пор, пока при успешной проверке не будет найдено решение с требуемыми свойствами или генератор не исчерпает все альтернативные решения.

Стандартный прием оптимизации программ типа “образуй и проверь” заключается в стремлении погрузить программу проверки как можно более “глубоко” в программу генерации. В пределе проверка и генерация реализуется одним правилом, которое порождает только корректные решения.

Пример1 – задача об офицерах.

На одном вечере среди гостей оказалось пять офицеров: пехотинец, артиллерист, летчик, связист и сапер. Один из них был капитаном, трое майорами и один- в звании подполковника. Из разговоров удалось выяснить следующее :

1) У Якова такое же звание, как у его друга сапера;
2) офицер-связист и Филипп - большие друзья;
3) офицер-летчик вместе с Борисом и Леонидом недавно побывали в гостях у Филиппа;
4) незадолго до званного вечера у артиллериста и сапера почти одновременно вышли из строя радиоприемники. Оба в один день обратились к Леониду с просьбой зайти к ним и помочь связисту устранить неисправность.
5)Филипп чуть не стал летчиком, но потом по совету своего друга сапера избрал другой род войск;
6)Яков по званию старше Леонида, а Борис старше Филиппа;
7)Андрей, пятый офицер, накануне вечера был в гостях у Леонида.

Нужно :

- определить звание каждого офицера;
- род войск в котором каждый служит.

Решение.

Условие задачи позволяет определить в виде фактов следующие отношения :

“принадлежит рассматриваемому множеству офицеров” :

```
oficer("Jakov").
oficer("Philip").
oficer("Leonid").
oficer("Boris").
oficer("Andrey").
```

Рода войск, в которых служат рассматриваемые офицеры :

```
can_be("Pehotinets").
can_be("Artillerist").
can_be("Letchik").
can_be("Svyazist").
can_be("Saper").
```

Имеет одинаковое звание с кем-либо из рассматриваемых офицеров :

```
same("Jakov").
```

Является старшим по званию :

```
senior("Jakov","Leonid").
senior("Boris","Philip").
```

Рода войск, в которых не служит конкретный офицер :

```
not_was("Philip","Letchik").
not_was("Philip","Svyazist").
not_was("Philip","Saper").
not_was("Jakov","Saper").
not_was("Boris","Letchik").
not_was("Leonid","Letchik").
not_was("Leonid","Artillerist").
not_was("Leonid","Saper").
not_was("Leonid","Svyazist").
```

Порождение и контроль допустимости решений.

Род войск Y, в которых служит офицер X, находится с помощью предиката can_be и затем проверяется, не относится ли данный род войск к тем родам, в которых офицер X не служит :

```
rod_voisk(X,Y):-oficer(X),can_be(Y),not(not_was(X,Y)).
```

При определении рода войск, в котором служит каждый из офицеров, род войск для каждого офицера определяется с помощью предиката rod_voisk, а допустимость полученного решения для каждого офицера определяется с помощью предиката condition :

```
who_is_who(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5):-
        rod_voisk(X1,Y1),
        rod_voisk(X2,Y2),
        rod_voisk(X3,Y3),
        rod_voisk(X4,Y4),
        rod_voisk(X5,Y5),
        condition(X1,X2,X3,X4,X5),
        condition(Y1,Y2,Y3,Y4,Y5),!.
```

Определение воинского звания каждого офицера может послужить примером симбиоза проверки и генерации : с помощью предиката oficer генерируются имя офицера для заданного в заголовке правила zvanie звания, последующая конъюнкция ЦУ проверяет допустимость решения.

Программа для решения задачи “офицеры”.

```
/* Задача об офицерах */
predicates
   oficer(string)
   zvanie(string,string)
   senior(string,string)
   same(string)
   rod_voisk(string,string)
   can_be(string)
   not_was(string,string)
   condition(string,string,string,string,string)
   who_is_who(string,string,string,string,string,
                  string,string,string,string,string)

clauses
   oficer("Jakov").
   oficer("Philip").
   oficer("Leonid").
   oficer("Boris").
   oficer("Andrey").

   same("Jakov").

   senior("Jakov","Leonid").
   senior("Boris","Philip").

   can_be("Pehotinets").
   can_be("Artillerist").
   can_be("Letchik").
   can_be("Svyazist").
   can_be("Saper").

   not_was("Philip","Letchik").
   not_was("Philip","Svyazist").
   not_was("Philip","Saper").
   not_was("Jakov","Saper").
   not_was("Boris","Letchik").
   not_was("Leonid","Letchik").
   not_was("Leonid","Artillerist").
   not_was("Leonid","Saper").
   not_was("Leonid","Svyazist").

   rod_voisk(X,Y):-oficer(X),can_be(Y),not(not_was(X,Y)).

   zvanie(X,"Major"):-oficer(X),same(X).

   zvanie(X,"Capitan"):-oficer(X),not(same(X)),
                        oficer(Y),same(Y),senior(Y,X).
zvanie(X,"Podpolkovnik") : - oficer(X),not(same(X)),
                        oficer(X1),same(X1),
                        oficer(X2),senior(X1,X2),
                        oficer(X3),senior(X,X3).
zvanie(X,"Major") : - oficer(X),not(same(X)),
                not(zvanie(X,"Capitan")),
                not(zvanie(X,"Podpolkovnik")).

condition(X1,X2,X3,X4,X5) : - X1<>X2,X1<>X3,X1<>X4,X1<>X5,
                        X2<>X3,X2<>X4,X2<>X5,
                        X3<>X4,X3<>X5,X4<>X5.

who_is_who(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5) : -
        rod_voisk(X1,Y1),
        rod_voisk(X2,Y2),
        rod_voisk(X3,Y3),
        rod_voisk(X4,Y4),
        rod_voisk(X5,Y5),
        condition(X1,X2,X3,X4,X5),
        condition(Y1,Y2,Y3,Y4,Y5),!.
```

Пример 2 – задача об N ферзях.

Суть задачи : требуется разместить N ферзей на шахматной доске размера N×N так, чтобы на каждой горизонтальной, вертикальной или диагональной линии было не больше одной фигуры. В первоначальной формулировке этой задачи речь шла о размещении 8 ферзей на шахматной доске таким образом, чтобы они, согласно правилам игры в шахматы, не угрожали друг другу.

Для N=2 и N=3 решения не существует, для N=4 решение единственное (без учета симметричного ему решения).

Для реальной шахматной доски 8×8 существует 88 (а с учетом симметричных - 92) решений этой задачи.

В приведенной далее программе отношение *queen(N,Qs)* истинно, если *Qs* – решение задачи об *N* ферзях. Решение представляется некоторой перестановкой списка от *1* до *N*. Порядковый номер элемента этого списка определяет номер вертикали, а сам элемент – номер горизонтали, на пересечении которых стоит ферзь.

Реализация “Образуй и проверь” : генерация решений.

В состав генератора возможных решений в предлагаемом здесь решении входят предикаты range и select.

```
/* Нахождение списка целых чисел между M и N включительно */

    range(N,N,[N]).
    range(M,N,[M|Ns]):-M<N,M1=M+1,range(M1,N,Ns).

/* Выделение элемента из списка */

    select(X,[X|Xs],Xs).
    select(X,[Y|Ys],[Y|Zs]):-select(X,Ys,Zs).
```

Проверка допустимости полученных решений.

Реализуется отрицанием предиката attack. В определении предиката attack использована инкапсуляция взаимосвязи диагоналей. Два ферзя находятся на одной и той же диагонали на расстоянии *N* вертикалей друг от друга, если номер горизонтали одного ферзя на *N* больше или на *N* меньше, чем номер горизонтали другого ферзя.

```
/* Проверка наличия атаки некоторого ферзя */

    attack(X,Xs):-attack(X,1,Xs).

    attack(X,N,[Y|Ys]):-X=Y+N;X=Y-N.
    attack(X,N,[Y|Ys]):-N1=N+1,attack(X,N1,Ys).
```

Смысл предиката *attack(X,Xs)* можно выразить словами : “Ферзь *X* атакует некоторого другого ферзя из списка *Xs*”. Диагонали проверяются итерационно до тех пор, пока не будет достигнут конец доски.

Последовательное размещение ферзей.

Чтобы проверить, в безопасном ли положении находится новый ферзь, необходимо знать позиции ранее размещенных ферзей. Следовательно, искомое решение должно строиться снизу вверх с применением накапливающего параметра. Использование накапливающего параметра приводит к размещению ферзей, начиная с правой границы доски.

Правило queens осуществляет проверку корректности размещения каждого ферзя непосредственно после генерации его потенциальной позиции на доске :

```
queens(N,Qs):-range(1,N,Ns),queens(Ns,[ ],Qs).

queens([ ],Qs,Qs).

queens(UnplacedQs,SafeQs,Qs):-
        select(Q,UnplacedQs,UnplacedQs1),
        not(attack(Q,SafeQs)),
        queens(UnplacedQs1,[Q|SafeQs],Qs).
```

Программа для решения задачи об N ферзях.

```
/* Задача об N ферзях */
domains
      ilist=integer*
predicates
      m(integer,integer)
      d(ilist,integer)
      t(integer)
      r
      range(integer,integer,ilist)
      select(integer,ilist,ilist)
      attack(integer,ilist)
      attack(integer,integer,ilist)
      queens(integer,ilist)
      queens(ilist,ilist,ilist)
      run_fritz(integer)
      run(integer)
clauses
/* Прорисовка шахматной доски */
      m(X,N) : - write("  "),write("▒▒"),X1=X+1,
                 X1<>78,!,m(X1,N).
      m(X,N) : - cursor(1,0),write("  "),!.

      t(0) : - !.
      t(X) : - X1=X-1,t(X1),cursor(X,0),write("  ").

      r:-attribute(7),
        gotowindow(1),cursor(3,8),
        gotowindow(2).
/* Прорисовка ферзей */
      d([ ],P).
      d([X|T],P) : - S=P*2,Q=X-1,cursor(Q,S),
                     attribute(6),writef("@@"),
```

```
/* Решение задачи об N ферзях */
/* Нахождение списка целых чисел
   между M и N включительно */
      range(N,N,[N]).
      range(M,N,[M|Ns]) : - M<N,M1=M+1,range(M1,N,Ns).
/* Выделение элемента из списка */
      select(X,[X|Xs],Xs).
      select(X,[Y|Ys],[Y|Zs]) : - select(X,Ys,Zs).
/* Проверка наличия атаки некоторого ферзя */
      attack(X,Xs) : - attack(X,1,Xs).
      attack(X,N,[Y|Ys]) : - X=Y+N;X=Y-N.
      attack(X,N,[Y|Ys]) : - N1=N+1,attack(X,N1,Ys).
/* Расстановка ферзей на доске */
      queens(N,Qs) : - range(1,N,Ns),queens(Ns,[],Qs).
      queens([ ],Qs,Qs).
      queens(UnplacedQs,SafeQs,Qs) : -
            select(Q,UnplacedQs,UnplacedQs1),
            not(attack(Q,SafeQs)),
            queens(UnplacedQs1,[Q|SafeQs],Qs).
/* Запуск программы */
      run_fritz(N):-
            gotowindow(1), gotowindow(2), queens(N,Qs),
            m(0,N),t(13),d(Qs,1), N3=N+5,N4=N+N+7,
            makewindow(3,6,0,"",N3,5,8,30),
            makewindow(4,6,0,"",5,N4,19,20),
            r,readchar(_), clearwindow,
            gotowindow(4),removewindow,
            gotowindow(3),removewindow, fail.
  run(N) : - makewindow(1,6,2,"Задача об N ферзях",0,0,25,60),
            makewindow(2,6,0,"",5,5,17,26),
            run_fritz(N).
```

Решение логических головоломок.

Решение логических головоломок опишем на примере уже рассмотренной нами задаче об офицерах. Подобные логические головоломки, наряду с моделированием естественных рассуждений средствами логики предикатов 1-го порядка, могут быть решены посредством конкретизации подходящей структуры данных. Проанализируем первый ключ к разгадке : “У Якова такое же звание, как у его друга сапера”. Очевидно, речь идет о двух разных людях. Одного зовут Яков, другой является сапером, но в то же время оба имеют одно и то же воинское звание. В то же время о воинской специальности Якова ничего не известно, хотя из приведенной формулировки понятно, что он – не сапер, поскольку воинские специальности у всех различны. Предположим, что список : [oficer("Яков",Z1,R1),oficer("Филипп",Z2,R2), oficer("Борис",Z3,R3),oficer("Леонид",Z4,R4), oficer("Андрей",Z5,R5)] есть структура данных, подлежащая конкретизации. Тогда наш ключ может быть выражен следующей конъюнкцией целей :

can_carry_degree("Яков",Z1), can_be("Яков",R1), R1<>"Сапер",

find_eq_zv([oficer("Филипп",Z2,R2), oficer("Борис",Z3,R3),

oficer("Леонид",Z4,R4), oficer("Андрей",Z5,R5)],Z1).

Посредством применения правила find_eq_zv идет поиск сапера с требуемым воинским званием.

Задача об офицерах : анализ ключей.

Тот факт, что “офицер-связист и Филипп – большие друзья”, говорит о том, что Филипп – не связист. Кроме того, из утверждений :

“офицер-летчик вместе с Борисом и Леонидом недавно побывали в гостях у Филиппа” и “Филипп чуть не стал летчиком, но потом по совету своего друга-сапера избрал другой род войск” следует то, что :

1). Филипп, Борис и Леонид – не летчики;

2). Филипп – не сапер.

Относительно воинской специальности Леонида на основании утверждения : “незадолго да званного вечера у артиллериста и сапера почти одновременно вышли из строя радиоприемники и оба в один день обратились к Леониду с просьбой зайти к ним и помочь связисту устранить неисправность” можно заключить, что Леонид не является ни артиллеристом, ни сапером, ни связистом. Сказанное о Филиппе, Борисе и Леониде с учетом утверждения “Яков по званию старше Леонида, а Борис старше Филиппа” выражается приведенной на следующем слайде конъюнкцией целей.

Конъюнкция целей.

can_carry_degree("Яков",Z1),

can_carry_degree("Филипп",Z2),can_be("Филипп",R2),

R2<>"Летчик", R2<>"Сапер", R2<>"Связист",

can_carry_degree("Борис",Z3),can_be("Борис",R3), R3<>"Летчик",

can_carry_degree("Леонид",Z4),can_be("Леонид",R4),

R4<>"Летчик", R4<>"Артиллерист", R4<>"Связист", R4<>"Сапер",

senior(Z1,Z4),senior(Z3,Z2).

Здесь с помощью предиката senior формально определяется отношение “старший по званию”.

Таким образом, решение задачи об офицерах представляется как последовательное решение ключей с конкретизацией переменных в структурах - элементов списка : [oficer("Яков",Z1,R1),oficer("Филипп",Z2,R2), oficer("Борис",Z3,R3),oficer("Леонид",Z4,R4), oficer("Андрей",Z5,R5)]

Программа 2 для решения задачи об офицерах : генератор решений.

```
/* Задача об офицерах. Вариант 2. */
domains

    slist=string*
    oficer=oficer(string,string,string)
    oficers=oficer*

predicates

    member(string,slist)
    is_a_set(slist)
    counter(string,integer,slist)
    name(string)
    zvanie(string)
    rod_voisk(string)
    can_be(string,string)
    can_carry_degree(string,string)
    senior(string,string)
    find_eq_zv(oficers,string)
    who_is_who(oficers)
    run

clauses
/* Определение принадлежности списку */
    member(H,[H|_]) : - !.
    member(Elem, [ _ |T]) : - member(Elem,T).

/* Является ли список множеством ? */
    is_a_set([ ]).
    is_a_set([Head|Tail]) : - not(member(Head,Tail)),
                    is_a_set(Tail).

/* Подсчет числа офицеров искомого звания */
    counter( _,0, [ ]).
    counter( Zvanie,N, [Zvanie | Set]) : - !,
                    counter(Zvanie,N1,Set), N=N1+1.
    counter(Zvanie, N, [ _ | Set ]) : -
                    counter(Zvanie,N,Set).

/* Исходные данные для генератора решений */
    name("Яков").
    name("Филипп").
    name("Леонид").
    name("Борис").
    name("Андрей").

    zvanie("Майор").
    zvanie("Капитан").
    zvanie("Подполковник").

    rod_voisk("Пехотинец").
    rod_voisk("Артиллерист").
    rod_voisk("Летчик").
    rod_voisk("Связист").
    rod_voisk("Сапер").

    can_be(X,Y):-name(X),rod_voisk(Y).

    can_carry_degree(X,Y):-name(X),zvanie(Y).

/* Определение старшего по званию */
    senior("Майор","Капитан").
    senior("Подполковник","Майор").
```

Контроль допустимости решений.

```
/* Определение наличия среди офицеров сапера
   с заданным воинским званием */
    find_eq_zv([oficer(_,Z,"Сапер")|_],Z):-!.
    find_eq_zv([_|Tail],Z):-find_eq_zv(Tail,Z).

/* Решатель задачи об офицерах */
    who_is_who([oficer("Яков",Z1,R1),oficer("Филипп",Z2,R2),
                oficer("Борис",Z3,R3),oficer("Леонид",Z4,R4),
                oficer("Андрей",Z5,R5)]):-
          can_carry_degree("Яков",Z1),can_be("Яков",R1),
           R1<>"Сапер",
          can_carry_degree("Филипп",Z2),can_be("Филипп",R2),
           R2<>"Летчик", R2<>"Сапер", R2<>"Связист",
          can_carry_degree("Борис",Z3),can_be("Борис",R3),
           R3<>"Летчик",
          can_carry_degree("Леонид",Z4),can_be("Леонид",R4),
           R4<>"Летчик", R4<>"Артиллерист", R4<>"Связист",
           R4<>"Сапер",
          can_carry_degree("Андрей",Z5),can_be("Андрей",R5),
          is_a_set([R1,R2,R3,R4,R5]),
          counter("Майор",3,[Z1,Z2,Z3,Z4,Z5]),
          counter("Подполковник",1,[Z1,Z2,Z3,Z4,Z5]),
          counter("Капитан",1,[Z1,Z2,Z3,Z4,Z5]),
          find_eq_zv([oficer("Филипп",Z2,R2),
              oficer("Борис",Z3,R3),oficer("Леонид",Z4,R4),
              oficer("Андрей",Z5,R5)],Z1),
          senior(Z1,Z4),senior(Z3,Z2).
```

Литература.

Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог : Пер. с англ. - М.: Мир, 1990. С. 164-174.