Решение логических задач.

Лекция 5 (Часть 2).

Поиск на пространстве состояний.

Специальности : 230105, 010501

Состояния и операторы.

Определение 1. Под ***состоянием*** задачи будем понимать некоторая конфигурация исследуемой динамически меняющейся системы. При этом начальную и целевую конфигурации принято рассматривать в качестве, соответственно, ***начального*** и ***целевого состояний***.

Определение 2. *Оператор* преобразует одно состояние в другое. В общем случае мы будем предполагать, что операторы – это вычисления, преобразующие одни описания состояний в другие.

При *строковом описании* состояний удобным способом задания операторов являются *правила переписывания* или *продукции*.

Определение 3. *Правила переписывания* (*продукции*) определяют возможные способы преобразования одной строки в другую. Правила переписывания задаются в форме $S_i \rightarrow S_j$.

Пространство состояний.

Определение 4. ***Пространство состояний*****, достижимых из начального состояния, состоит из тех конфигураций системы, которые могут быть образованы из начальной с применением преобразований, допустимых некоторой совокупностью правил. Очень часто пространство состояний представляют в виде графа, вершины которого соответствуют состояниям, а дуги – операторам.**

При решении задач с использованием пространства состояний необходимо выбрать форму описания состояний задачи, эта форма должна быть единой для всех возможных состояний. Как правило, выбираемая форма описания имеет сходство с некоторым физическим свойством решаемой задачи и может принадлежать любому допустимому типу данных (массивы, строки, списки, деревья).

Решение задачи состоит в поиске пути из начального состояния в целевое посредством применения последовательности операторов.

Поиск на графе.

Все методы перебора в пространстве состояний могут быть смоделированы с помощью следующего теоретико-графового процесса:

***Начальная вершина* S соответствует описанию *начального состояния*. Вершины, непосредственно следующие за данной, получаются в результате использования операторов, которые применимы к описанию состояния, ассоциированного с этой вершиной.**

<u>Определение 1.</u> Пусть γ - некоторый специальный оператор, который строит все вершины, непосредственно следующие за данной. Будем называть процесс применения оператора γ к вершине *<u>раскрытием</u>* вершины. Для каждой раскрываемой вершины делается проверка, является ли вершина целевой.

<u>Определение 2.</u> От каждой дочерней вершины к родительской идут указатели, которые позволяют найти путь назад к начальной вершине после обнаружения целевой вершины. Вершины и указатели, построенные в процессе перебора, образуют поддерево всего неявно определенного дерева пространства состояний. Данное поддерево называют *деревом перебора*.

<u>Определение 3.</u> Решающую последовательность образуют операторы, которые связаны с дугами пути от целевой вершины к начальной.

Метод перебора в глубину.

Определение 5. Процесс перебора *в глубину* (depth-first process) характеризуется тем, что вначале раскрывается та вершина, которая была построена самой последней.

Определение 6. (глубина вершины в дереве).

1) Глубина корня дерева равна нулю.

2) Глубина любой последующей вершины дерева равна глубине родительской вершине, увеличенной на 1.

Определение 7. Вершиной с наибольшей глубиной в дереве перебора а некоторый момент служит раскрываемая в данный момент.

Определение 8. *Граничная глубина* при использовании метода перебора в глубину вводится в целях отсечения заведомо бесполезных путей в дереве и организации процедуры *возврата*. При построении вершины с глубиной, превышающей *граничную*, следующей будет раскрыта вершина наибольшей глубины, не превышающей граничного значения.

Базовая программа для решения задач поиска на графах состояний.

Задаваемые операторами переходы будем описывать бинарным предикатом *move(State, Move)*, где *Move* – правило перехода, применяемое к состоянию *State*. Предикат *update(State, Move, State1)* используется для поиска состояния *State1*, достижимого с помощью применения правила *Move* к состоянию *State*. В ряде случаев целесообразным является объединение процедур *move* и *update*. В реализованной нами программе мы их оставляем раздельными в целях простоты изложения и сохранения гибкости и модульности использующих их программ.

Допустимость возможных переходов оценивается предикатом *legal(State)*, который проверяет, удовлетворяет ли состояние *State* ограничениям задачи.

В целях предупреждения зацикливания программа сохраняет ранее пройденные состояния. Последовательность переходов из начального состояния в целевое строится путем наращивания списка *Moves* в третьем аргументе правила *solve_dfs* :

***solve_dfs(State, History, Moves)*, где *Moves* – последовательность переходов до достижения требуемого конечного состояния из текущего состояния *State*. *History* содержит ранее пройденные состояния.**

Листинг базовой программы организации поиска в глубину.

```
/* Построение последовательности переходов из начального состояния
   в конечное
   Первый аргумент - текущее состояние,
   Второй аргумент - список ранее пройденных состояний,
   Третий аргумент - последовательность переходов до достижения
                     требуемого конечного состояния из текущего состояния */

solve_dfs(State,History,[ ]):-
    final_state(State).

solve_dfs(State,History,[Move|Moves]):-
    move(State,Move),
    update(State,Move,State1),
    legal(State1),
    not(member(State1,History)),
    solve_dfs(State1,[State1|History],Moves).

/* Запуск базовой программы для решения задач путем поиска на графах
   пространства состояний с применением поиска в глубину */

test_dfs(Problem,Moves):-
    initial_state(Problem,State),
    solve_dfs(State,[State],Moves).
```

Примечание. Чтобы использовать базовую программу организации поиска при решении задачи, программист должен выбрать формальное представление для состояний задачи и аксиоматизировать процедуры *move, update* и *legal*.

Пример : задача о волке, козе и капусте.

Состояния представляются тройкой *wgc(B,L,R)*, где *B* – местонахождение лодки (и фермера) – левый или правый берег, *L* – список находящихся на левом берегу, *R* – список находящихся на правом берегу.

Начальное состояние :

wgc("Лодка на левом берегу.", ["Волк","Коза","Капуста"], [ ]).

Конечное состояние :

wgc("Лодка на правом берегу.", [ ], ["Волк","Коза","Капуста"]).

Для проверки зацикливания удобно сохранять списки обитателей в отсортированном виде. При нахождении *волка*, *козы* и *капусты* на одном берегу *волк* будет в списке перед *козой*, и оба они – перед *капустой*.

Переходы из состояния в состояние – это перевозка обитателей с одного берега на другой, каждый переход может быть специфицирован конкретным грузом (Cargo).

Допустимость переходов определяется условием задачи : в отсутствие фермера волк и коза, равно как и коза с капустой, не могут в отсутствие фермера находиться на одном берегу.

Программа для решения задачи о волке, козе и капусте (часть 1).

```
/* Начальное и конечное состояние */

  initial_state("Исходное состояние : все находятся на левом берегу ",
          wgc("Лодка на левом берегу.", ["Волк","Коза","Капуста"], [ ])).

  final_state(wgc("Лодка на правом берегу.",[ ], ["Волк","Коза","Капуста"])).

/* Определение возможности перехода из состояния в состояние */

  move(wgc("Лодка на левом берегу.",L,R),Cargo):-member(Cargo,L).
  move(wgc("Лодка на правом берегу.",L,R),Cargo):-member(Cargo,R).
  move(wgc(B,L,R),"Без груза").

/* Поиск состояния, достижимого из заданного */

  update(wgc(B,L,R),Cargo,wgc(B1,L1,R1)):-
     update_boat(B,B1),update_banks(Cargo,B,L,R,L1,R1).

/* Изменение местонахождения лодки */

  update_boat("Лодка на левом берегу.","Лодка на правом берегу.").
  update_boat("Лодка на правом берегу.","Лодка на левом берегу.").

/* Выбор груза для перевозки */

  select(X,[X|T],T).
  select(X,[H|T],[H|D]):-select(X,T,D).
/* Порядок сортировки */

  precedes("Волк",X).
  precedes(X,"Капуста").
```

Программа для решения задачи о волке, козе и капусте (часть 2).

```
/* Вставка элемента в список */

  insert(X,[Y|Ys],[X,Y|Ys]):-
     precedes(X,Y).

  insert(X,[Y|Ys],[Y|Zs]):-
     precedes(Y,X),insert(X,Ys,Zs).

  insert(X,[],[X]).

/* Изменение состава обитателей берегов
  Первый аргумент - перевозимый в лодке груз,
  Второй аргумент - текущее местонахождение лодки (левый или правый берег),
  Третий и четвертый аргументы - текущий состав обитателей правого и левого берега,
  Пятый и шестой аргументы - новый состав обитателей правого и левого берега. */

/* Ситуация, когда фермер переправляется через реку без груза */

  update_banks("Без груза",B,L,R,L,R).

/* Ситуация перевозки обитателя одного берега на другой */

  update_banks(Cargo,"Лодка на левом берегу.",L,R,L1,R1):-
     select(Cargo,L,L1),insert(Cargo,R,R1).

  update_banks(Cargo,"Лодка на правом берегу.",L,R,L1,R1):-
     select(Cargo,R,R1),insert(Cargo,L,L1).

/* Допустимость состояния */

  legal(wgc("Лодка на левом берегу.",L,R)):-not(illegal(R)).
  legal(wgc("Лодка на правом берегу.",L,R)):-not(illegal(L)).

  illegal(List):-member("Волк",List),member("Коза",List).
  illegal(List):-member("Коза",List),member("Капуста",List).
```

Выводы.

При формулировке задачи в пространстве состояний решение получается в результате применения операторов к описаниям состояний до тех пор, пока будет получено выражение, описывающее целевое состояние.

Исследование различных стратегий перебора может быть весьма эффективно реализовано с применением языка теории графов.

Эффективность механизма поиска может быть значительно повышена привлечением знаний проблемной области в виде разного рода эвристической информации.

Литература.

Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог : Пер. с англ. - М.: Мир, 1990. С. 224-238

Нильсон Н. Искусственный интеллект : Пер. с англ. - М.: Мир, 1973. С. 52-90

Ин Ц., Соломон Д. Использование Турбо-Пролога : пер. с англ. - М.: Мир, 1993. С. 511-521