СПбГЭТУ «ЛЭТИ»
Кафедра Вычислительной техники
Дисциплина «Искусственный интеллект»

Лекция 9

Реализация поиска для головоломки «8-ка» в среде CLIPS

Общий алгоритм *эвристического* поиска

Обозначения: *S* – вершина, соответствующая начальному состоянию;
G – вершина, соответствующая целевому состоянию;
OPEN – список порожденных, но нераскрытых вершин;
CLOSED – список раскрытых вершин.

1. *OPEN* := {*S* };
2. Если *OPEN* = ∅, завершить алгоритм. Путь в целевое состояние не существует.
3. Выбрать из списка *OPEN* вершину *n* ***с минимальным значением ЦФ***: для всех вершин *m* в *OPEN* *f*(*n*) <= *f*(*m*). Удалить *n* из *OPEN* и добавить в *CLOSED*
4. Сгенерировать список вершин-потомков *n*, удалить из него вершины образующие петли; для оставшихся установить указатель на вершину *n* (родителя).
5. Если в списке вершин-потомков присутствует *G*, завершить поиск. Сформировать путь от вершины *S* до вершины *G* (обратным проходом по указателям)
6. *Иначе* (если *G* ***не достигнута***) ***для каждой новой вершины*** *n*’ (потомков *n*) выполнить следующие операции:

   6.1 Вычислить ЦФ *f*(*n*‘)

   6.2 Если *n*' не присутствует в списке *OPEN* или в *CLOSED*, добавить ее в список, назначив оценку *f*(*n*') и установив указатель на вершину-родитель *n*.

   6.3 Если *n*' присутствует в списке *OPEN* или в списке *CLOSED*, сравнить новое значение $f(n')_{new}$ с прежним $f(n')_{old}$

   6.4 Если $f(n')_{new} >= f(n')_{old}$, прекратить обработку новой вершины.

   6.5 Если $f(n')_{new} < f(n')_{old}$, заменить в списке существующую вершину новой, причем, если существующая вершина была в списке *CLOSED*, перенести ее в список *OPEN*.

Головоломка 8-ка: Представление состояний

LeftTop	MiddleTop	RightTop
LeftMiddle	MiddleMiddle	RightMiddle
LeftBottom	MiddleBottom	RightBottom

Шаблон представления вершины в дереве поиска

```
(deftemplate Node))
  (slot LeftTop(type NUMBER))            ;;  9 слотов для  положения фишек
  (slot MiddleTop(type NUMBER))
  (slot RightTop(type NUMBER))
  (slot LeftMiddle(type NUMBER))
  (slot MiddleMiddle(type NUMBER))
  (slot RightMiddle(type NUMBER))
  (slot LeftBottom (type NUMBER))
  (slot MiddleBottom(type NUMBER))
  (slot RightBottom(type NUMBER))

  (slot Depth(type NUMBER))              ;;  глубина вершины в дереве
  (slot Id(type NUMBER) (default 0))     ;; уникальный идентификатор вершины
  (slot Status(type NUMBER) (default 0))     ;; статус вершины:
                                                0 – не раскрыта,
                                                1 – раскрыта
                                                2 – соответствует решению
  (slot From (type NUMBER))      ;; ссылка на родителя
  (slot Exp (type NUMBER))       ;; значение целевой функции для данной вершины
)
```

Как присваивать Id вершинам?

- Каждая вершина в дереве поиска должно иметь уникальный Id
- Для этого каждой новой вершине надо присваивать значение Id на единицу большее, чем у предыдущей
- Реализация в CLIPS через глобальную переменную:

```
(defglobal
   ?*Id* = 0    ;;  объявляем и инициализируем глобальную переменную
)

(deffunction Get_Id()          ;; функция получения следующего Id
  (bind ?*Id* (+ ?*Id* 1))  ;;  инкрементируем Id
   ?*Id*
)
```

Как вычислять значение целевой функции?

- *В поиске A* вычисление значения целевой функции* требует знания:
  - *числа* шагов от начальной ситуации к текущей – глубины вершины;
  - номеров фишек на каждой позиции.

Целевое состояние:

1	2	3
8		4
7	6	5

Для функции «*Число фишек не на своем месте*»:

```
(deffunction W(?Depth ?LeftTop ?MiddleTop ?RightTop
?RightMiddle ?RightBottom ?MiddleBottom
?LeftBottom ?LeftMiddle)
 (bind ?a ?Depth)        ;; f(n) := g(n)
 (if (not (= ?LeftTop 1)) then (bind ?a (+ 1 ?a)))     ;; подсчет f(n)
 (if (not (= ?MiddleTop 2)) then (bind ?a (+ 1 ?a)))   ;; в части h(n)
 (if (not (= ?RightTop 3)) then (bind ?a (+ 1 ?a)))    ;; по всем (8-ми!)
 (if (not (= ?RightMiddle 4)) then (bind ?a (+ 1 ?a))) ;; ячейкам
 (if (not (= ?RightBottom 5)) then (bind ?a (+ 1 ?a)))
 (if (not (= ?MiddleBottom 6)) then (bind ?a (+ 1 ?a)))
 (if (not (= ?Left Bottom 7)) then (bind ?a (+ 1 ?a)))
 (if (not (= ?LeftMiddle 8)) then (bind ?a (+ 1 ?a)))
 ?a    ;; возвращаемое значение
)
```

Инициализация корневой вершины дерева поиска

- Объявление факта, соответствующего начальной вершине и инициализация минимума целевой функции:

2	8	3
1	6	4
7		5

```
(deffacts start        ;; факты соответствующие исходному состоянию
 (Node(LeftTop 2) (MiddleTop 8) (RightTop 3)
       (LeftMiddle 1) (MiddleMiddle 6)(RightMiddle 4)
       (LeftBottom 7) (MiddleBottom 0) (RightBottom 5)
 (Depth 0)
 (From 0)
 (Exp (W 0 2 8 3 4 5 0 7 1))
 (Id (Get_Id))
 )
(min (W 0 2 8 3 4 5 0 7 1)) ;; фиксируется текущее значение min f(n)
)
```

Приоритетность действий

Поскольку алгоритм реализуется правилами, **порядок действий должен обеспечиваться приоритетами !**

1. Если целевое состояние достигнуто, зафиксировать вершину (установить статус 2)
   - Восстановить путь (обратным проходом)
   - Удалить остальные вершины, завершить алгоритм
2. Если целевое состояние недостижимо, завершить алгоритм
3. Удалить вершины с повторяющимися состояниями (**максимальный приоритет**)
4. Выбрать в множестве *OPEN* вершину с минимальным значением целевой функции
5. Породить вершины-потомки выбранной вершины, добавить в список

Удаление повторных состояний

```
(defrule move_circle
   (declare (salience 1000)) ;; максимальный приоритет!!!
   (Node(Exp ?X))                ;; первый факт
    ?f<-(Node (Exp ?Y&~?X)(Status 0));; второй факт
                                     ;;с неравным значением ЦФ
   (test(< ?X ?У))   ;;бoльшим, чем у первого состояния
= >
    (retract ?f) ;; удаление вершины с бoльшей ЦФ
    (printout t "delete rule" crlf)
 )
```

Проверка достижения цели

- Решение найдено, если порождена вершина, соответствующая целевому состоянию. Ей необходимо установить статус 2:

```
(defrule goal_test                    ;;проверка целевого состояния
  (declare (salience 500))
  ?f<-(Node (LeftTop 1) (MiddleTop 2) (RightTop 3);; состояние
  (LeftMiddle 8) (MiddleMiddle 0) (RightMiddle 4) ;; целевое
  (LeftBottom 7) (MiddleBottom 6) (RightBottom 5)
  (Status ~2) ;; текущий статус вершины «не целевое»
  (From ?Id))
=>
  (modify ?f(Status 2));;текущая вершина помечается как «целевая»
)
```

Бэктрекинг для восстановления пути - решения

- После того как достигнута вершина с целевым состоянием, *всем вершинам*, лежащим на пути к ней надо установить статус 2
- Обратный проход до начальной вершины выполняется с помощью значения в слоте **`From:`**

```
(defrule select_answer        ;; изменение статуса промежуточных вершин
                              ;; - бэктрекинг
   (declare (salience 500))
   (;; вершина со статусом 2, ссылается на
        Node(Status 2) (From ?Id))    ;; родителя
   ?f<-(Node(Id ?Id) (Status ~2))
=>
   (modify ?f(Status 2))
)
```

Удаление ненужных вершин

- Если *решение найдено* (и другие правила невыполнимы или отработали), то необходимо удалить ненужные вершины:

```
(defrule delete_not_answer
   (declare (salience 400))  ;; более низкий приоритет !!!
   (Node(Status 2))          ;;  если появилась вершина со статусом 2
   ?f<-(Node(Status ~2))     ;;  все вершина со статусом НЕ 2, надо удалить
=>
   (retract ?f)
)
```

- Правило выполняется , если:
  - решение найдено и
  - есть вершины, не соответствующие решению.

  Такие вершины удаляются

Правила останова программы

Программа должна завершаться либо **когда решение найдено**, либо когда **оно не существует**. Этим случаям соответствуют правила:

```
(defrule Stop_1                    ;; выполняется, если решение не существует
   (declare (salience 200))        ;; приоритет низкий
   (not (Node(Status 0|2)))        ;; в базе фактов нет вершин со статусом 0 или 2,
=>                                 ;; т.е. нераскрытых и с целевым состоянием
   (halt)
   (printout t "no solutions" crlf)
)

(defrule Stop_2                    ;; выполняется, если решение найдено
   (declare (salience 200))
   (Node(Status 2))                ;; есть вершина с целевым состоянием
=>
   (halt)
   (printout t "fined solution" crlf)
)
```

Обновление текущего минимума целевой функции

- Для раскрытия должна выбираться вершина из *OPEN* с минимальным значением целевой функции, меньшим текущего *min*

```
(defrule find_min ;; определение текущего минимума ЦФ
   (declare (salience 150)) ;; приоритет низкий!
  ?fmin<-(min ?min)  ;; адрес факта, хранящего значение текущего
                     ;; min целевой функции
   (Node(Exp ?E&:(< ?E ?min))(Status 0)) ;;существование
                ;; вершины, у которой значение целевой функции
                ;; меньше текущего min
=>
  (retract ?fmin)     ;; удалить факт с текущим min
  (assert (min ?E))   ;; создать  факт с новым текущим min
)
```

Правило перехода между состояниями

```
(defrule make_new_path_LeftTop ;; пустое место в левом верхнем углу
    (declare (salience 100))    ;; приоритет самый низкий!!
    ?fmin <- (min ?min)         ;; получение ссылки на факт с текущим минимумом
    ?f<-(Node(Status 0) (Depth ?L) (Id ?Id) ;; определение состояния с пустым полем
             (LeftTop 0) (MiddleTop ?MT) (RightTop ?RT) ;; в верхнем левом углу
             (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
             (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
    (Exp ?E& :(= ?E ?min))) ;; проверка ЦФ вершины на min
=>
    (printout t ?min " " (fact-slot-value ?f Exp) crlf)
    (modify ?f(Status 1))      ;; изменение статуса вершины на «раскрыта»
    (bind ?a (W (+ 1 ?L) ?MT 0 ?RT ?RM ?RB ?MB ?LB ?LM))
    (retract ?fmin)
    (assert (min ?a))
    (assert (Node(LeftTop ?MT) (MiddleTop 0) (RightTop ?RT)
                 (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
                 (LeftBottom ?LB) (MiddleBottom ?MB)(RightBottom ?RB)
    (Depth(+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
    (assert (Node(LeftTop ?LM) (MiddleTop ?MT) (RightTop ?RT)
                 (LeftMiddle 0)(MiddleMiddle ?MM) (RightMiddle ?RM)
                 (LeftBottom ?LB) (MiddleBottom ?MB)(RightBottom ?RB)
    (Depth(+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LM ?MT ?RT ?RM
    ?RB ?MB ?LB 0)) (Id (Getld))
)
```