# Лабораторная работа №5: Исследование моделей планирования действий в системах искусственного интеллекта

## Цель работы

Изучение принципов описания и решения задачи планирования на основе продукционной модели с использованием механизма поиска в пространстве состояний.

## Основные теоретические положения

Задача построения башни из блоков (кубиков) заключается в последовательном выборе из неупорядоченной кучи блоков и постановки их друг на друга. План решения задачи представляет собой чередование двух фаз: выбора блока из кучи и установки его в башню. При этом больший по размерам блок не может ставиться на меньший по размерам и, следовательно, на каждом шаге решения задачи необходимо выбирать из кучи самый большой блок.

Каждый блок характеризуется *размером*, *цветом* и *положением*, поэтому для его описания можно использовать следующий шаблон:

```
(deftemplate block
  (slot size (type INTEGER))
  (slot place (type SYMBOL))
  (slot color (type SYMBOL))
)
```

Каждый блок может находиться либо в куче (heap), либо в руке (hand), когда он выбран из кучи, либо в башне (tower).

В результате решения задачи необходимо последовательно вывести информацию о действиях, составляющих план её решения и взаимное положение блоков в построенной по данному плану башне с указанием их цвета. С этой целью для блоков, находящихся в башне, необходимо определить отношение, указывающее, какой блок (upper) на каком (lower) стоит:

```
(deftemplate on
  (slot upper (type SYMBOL))
  (slot lower (type SYMBOL))
  (slot place (type SYMBOL)(default heap))
)
```

Если блок ставится в основание башни, слоту place присваивается значение tower, а слоты upper и lower остаются неопределенными. Если блок ставится на другой блок, слотам upper и lower должны присваиваться значения цветов соответствующих блоков.

Поскольку план решения задачи представляет собой чередование фаз поиска в компоненте пространства состояний («куче») самого большого блока (`find`) и его установки в башню (`build`), необходимо иметь факт, определяющий текущую задачу. Его можно определить с помощью следующего шаблона:

```
(deftemplate goal
  (slot current-task (type SYMBOL))
)
```

Для решения задачи необходимы правила для следующих действий:

- установки начального значения текущей задачи;
- выбора из кучи самого большого блока;
- установки первого блока в основание башни;
- установки в башню всех последующих блоков;
- определения окончания процесса, когда в куче нет больше блоков.

*Правило установки начального значения текущей задачи* должно активироваться исходным фактом (`initial-fact`) и устанавливать в качестве текущей задачи `find`.

**Правило выбора из кучи самого большого блока** должно активироваться следующими условиями:

- текущая задача – `find`;
- наличие в куче блока, для которого не существует большего по размерам.

Действия правила:

- взять найденный блок в руку (изменить его местоположение на hand);
- изменить текущую задачу на `build`.

**Правило установки блока в основание башни** (первого блока) имеет следующие условия активации:

- текущая задача – `build`;
- имеется блок в руке;
- отсутствуют блоки в башне – нет блока, у которого значение слота `place` равно `tower`.

Действия правила:

- изменить местоположение блока в руке на `tower`;
- изменить текущую задачу на `find`.

**Правило установки в башню последующих блоков** имеет условия активации:

- текущая задача – `build`;
- имеется блок в руке (для него надо запомнить цвет);
- имеется блок в башне, на котором не стоит другой блок (для него также надо запомнить цвет).

Действия правила:

- модифицировать местоположение блока в руке на `tower`;
- установить факт, что новый блок находится на блоке, который был до этого верхним;
- изменить текущую задачу на `find`.

**Правило определения окончания процесса** должно активироваться, когда в куче больше нет блоков и удалять из рабочей памяти факт текущей задачи.

# Постановка задачи

Разработать и отладить на языке CLIPS программу решения задачи планирования постройки башни из блоков с пошаговым отображением плана построения башни.

# Порядок выполнения работы

1. Разработать программу решения задачи планирования на языке CLIPS, реализующую описанные выше правила, в соответствии с вариантом задания, указанным в таблице. Исходное состояние должно описываться конструкцией `deffacts`, содержащей четыре факта, соответствующих блокам в куче.
2. Вывод результатов на экран должен обеспечивать пошаговое отображение плана построения башни.

# Варианты заданий

Для каждого блока в таблице через слеш указаны параметры: цвет/размер.

| № варианта | Блок-1 | Блок-2 | Блок-3 | Блок-4 |
|---|---|---|---|---|
| 1 | синий/10 | зелёный/8 | красный/18 | жёлтый/15 |
| 2 | красный/20 | коричневый/5 | жёлтый/12 | зелёный/8 |
| 3 | жёлтый/15 | красный/10 | белый/25 | коричневый/5 |
| 4 | зелёный/10 | жёлтый/14 | чёрный/18 | красный/11 |
| 5 | коричневый/5 | белый/25 | синий/14 | жёлтый/12 |
| 6 | красный/10 | чёрный/18 | жёлтый/15 | белый/25 |
| 7 | жёлтый/12 | синий/10 | зелёный/8 | чёрный/18 |
| 8 | белый/16 | красный/5 | коричневый/15 | синий/10 |
| 9 | чёрный/18 | жёлтый/15 | красный/20 | белый/10 |

# Содержание отчёта

- Цель работы.
- Краткое изложение основных теоретических понятий.
- Постановка задачи с кратким описанием порядка выполнения работы.
- Пошаговый план решения задачи.
- Результаты работы программы с краткими выводами.
- Общий вывод по проделанной работе.
- Код программы.

# Пример решения задачи

## Решение задачи в среде CLIPS

lab5.CLP

```
;;//Шаблон для блока, хранит размер , цвет, положение - в куче, в руке или помещен в башню
(deftemplate block
  (slot size (type INTEGER))
  (slot place (type SYMBOL)(default heap))
  (slot color (type SYMBOL))
)
;;//-------------------------------------------------------------------------------------
;;//Шаблон для отношения, задающего порядок расположения кубиков в башне (для основания значения полей upper,lower не определено
(deftemplate on
  (slot upper (type SYMBOL))
  (slot upper-size (type INTEGER))
  (slot lower (type SYMBOL))
  (slot lower-size (type INTEGER))
  (slot place (type SYMBOL)(default heap))
)
;;//-------------------------------------------------------------------------------------
;;//Действие, выполняемое в настоящий момент времени: поиск подходящего кубика или установка найденного на вершину башни
(deftemplate goal
  (slot current-task (type SYMBOL))
)
;;//-------------------------------------------------------------------------------------
;;//задаем начальное состояние кучи блоков с кубиками
(deffacts bloxx
  (block (size 10)(color blue))
  (block (size 20)(color red))
  (block (size 15)(color yellow))
  (block (size 10)(color green))
  (block (size 5)(color brown))
  (block (size 10)(color red))
  (block (size 12)(color yellow))
  (block (size 16)(color white))
  (block (size 18)(color black))
)
;;//-------------------------------------------------------------------------------------
;;//вывобит на экран информацию о блоке
```

```
(deffunction print-block(?color ?size)
  (printout t " <[block]  [color] = '" ?color "' [size] = '" ?size
"'/>"  crlf)
);;//конец deffunction print-block
;;//-------------------------------------------------------------------
-----------------------------------
;;//вывобит на экран информацию о наибольшем блоке
(deffunction print-biggest-block(?color ?size)
  (printout t crlf "Biggest block: ")
  (print-block ?color ?size)
  (printout t crlf )
);;//конец deffunction print-biggest-block
;;//-------------------------------------------------------------------
-----------------------------------
;;//Инициирует процесс поиска решения - устанавливает состояние системы "поиск
кубика"
(defrule init-system
    (initial-fact)
=>
    (assert (goal(current-task find)))
);;//конец defrule init-system
;;//-------------------------------------------------------------------
-----------------------------------
;;//Ищет наибольший блок в куче
(defrule find-biggest
  ?goal-ptr  <- (goal (current-task find))
  ?block-ptr <- (block (size ?s1)(color ?cl1)(place heap))
  (not
    (block
      (size ?s2&: (> ?s2 ?s1))
      (place heap)
    );;//end block
  );;//end (not): нет блока большего по размеру чем данный
=>
  (modify ?block-ptr (place hand))
  (modify ?goal-ptr (current-task build))
  (print-biggest-block ?cl1 ?s1);;//вывести информацию о блоке
);;//конец defrule find-biggest
;;//-------------------------------------------------------------------
-----------------------------------
;;//выводит блоки, находящиеся в куче
(defrule show-in-heap
  (declare (salience 2000))
  (goal (current-task find));;//текущ задача - "поиск"
  (block
    (place heap)
    (size  ?sz)
    (color ?cl)
  );;//end block
=>
```

```
  (printout t "Block in the heap: ")
  (print-block ?cl ?sz);;//вывести информацию о блоке
);;//конец defrule show-in-heap
;;//-------------------------------------------------------------------
-----------------------------------
;;//Устанавливает блок в основание башни
(defrule set-basis
  (declare (salience 1000))
  ?goal-ptr <- (goal (current-task build) );;//текущ задача - "строить"
  ?block-ptr <- (block (place hand)(color ?col)(size ?sz) );;// есть
блок в руке
  (not
    (block
      (place tower)
    );;//end block
  );;//end (not): нет блока, находящегося в башне
=>
  (modify ?goal-ptr (current-task find) );;// перейти к поиску следующего
  (modify ?block-ptr (place tower));;// установить блок в башню
  (assert
    (on
      (place tower)
      (upper ?col)
      (upper-size ?sz)
      (lower nil)
      (lower-size 0)
    );//end on
 );;//добавить новый факт, соотвествующий помещениб выбранного блока на верх
башни
);;//конец defrule set-basis
;;//-------------------------------------------------------------------
-----------------------------------
;;//Устанавливает все последующие (после заложенного в основание) блоки
(defrule set-next
  (declare (salience 1000))
  ?goal-ptr <- (goal (current-task build) );;//текущ задача - "строить"
  ?block-ptr <- (block (place hand) (color ?color-up) (size ?sz-
up));;// есть блок в руке
  (block (place tower) (color ?color-t) (size ?sz-t) );;//сохранили цвет
некоторого блока
  (not (on (place tower)(lower ?color-t)(lower-size ?sz-t)));;//нет
такого блока, который находится над данным
  (on (place tower)(upper ?color-t)(upper-size ?sz-t))
  ?on-ptr <- (on
               (place tower)
               (upper ?color-t)
               (upper-size ?sz-t)
       );;//выберем блок в башне, на котором нет блока сверху, т.е. верхний блок
=>
  (modify ?goal-ptr (current-task find) );;// перейти к поиску следующего
```

```
  (modify ?block-ptr (place tower));;// установить блок в башню
  (assert
    (on
      (place tower)
      (upper ?color-up)
      (upper-size ?sz-up)
      (lower ?color-t)
      (lower-size ?sz-t)
    );//end on
 );;//добавить новый факт, соотвествующий помещениб выбранного блока на верх
башни
);;//конец defrule set-next:
;;//-------------------------------------------------------------------
-----------------------------------
;;//останавливает процесс поиска, когда в куче ничего не осталось
(defrule finish-process
  (not (block (place heap)));;//нет ни одного блока в куче
  ?goal-ptr <- ( goal (current-task ?task))
=>
  (retract ?goal-ptr)
  (printout t "Solution found"  crlf "Order of the blocks from the
lowest to the highest" crlf)
);;//конец defrule finish-process
;;//-------------------------------------------------------------------
-----------------------------------
;;//выводит решение после окончания поиска
(defrule show-build-process
  (not (goal (current-task ?task) ));;//цель - показать решение
  (on (place tower)(upper ?color-up)(upper-size ?sz-up) );;//находим
блок на котором ничего не находится сверху
  (not (on (place tower)(lower ?color-up)(lower-size ?sz-up) ))
  ?on-ptr <- (on (place tower) (upper ?color-up)(upper-size ?sz-up)
);;//возьмем адрес отношения, соотв. данному блоку
=>
;;//(printout t "Block: color = " ?color-up " size = " ?sz-up  crlf)
  (print-block ?color-up ?sz-up)
  (retract ?on-ptr );;//удалим отношение, соотв. данному блоку, чтобы
продолжить процесс вывода решения
);;//конец defrule
```

## Пакетный файл

run_lab5.BAT

```
(load lab5.CLP)
(reset)
(run)
```