ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Вахтин А.А., Гаршина В.В.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ CLIPS ДЛЯ КУРСА «ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ИНФОРМАЦИОННЫХ СИСТЕМАХ»

Учебно-методическое пособие для вузов

Воронеж 2010

Утверждено научно-методическим советом факультета компьютерных наук, 19 марта 2010 г., протокол №20

Рецензент: д.ф.м.н., проф. зав каф. Программного обеспечения и администрирования информационных систем ВГУ , Артемов М.А.

Учебно-методическое пособие подготовлено на кафедре программирования и информационных технологий факультета компьютерных наук Воронежского государственного университета.

Рекомендовано для проведения лабораторных занятий по предметам ”Представление знаний информационных систем”, “Разработка экспертных систем”, ”Интеллектуальные информационные системы” со студентами 4 и 5 курсов дневного отделения факультета компьютерных наук.

Для специальностей 230201 (071900) – информационные системы и технологии и 230200 (654700)- информационные системы (бакалавр)

Введение

Название языка CLIPS — аббревиатура от **C Language Integrated Production System**. Язык был разработан в Центре космических исследований NASA (NASA's Johnson Space Center) в середине 1980-х годов и во многом сходен с языками, созданными на базе LISP, в частности OPS5 и ART. Использование C в качестве языка реализации объясняется тем, что компилятор LISP не поддерживается частью распространенных платформ, а также сложностью интеграции LISP-кода в приложения, которые используют отличный от LISP язык программирования. Хотя в то время на рынке уже появились программные средства для задач искусственного интеллекта, разработанные на языке C, специалисты из NASA решили создать такой продукт самостоятельно. Разработанная ими система в настоящее время доступна во всем мире, и нужно сказать, что по своим возможностям она не уступает множеству гораздо более дорогих коммерческих продуктов (http://www.ghgcorp.com/clips/CLIPS.html).

1. Основные теоретические сведения

CLIPS использует продукционную модель представления знаний, которая реализуется следующими основными компонентами языка описания правил:

1) база фактов (fact base);
2) база правил (rule base):
3) блок вывода (машина логического вывода).

На них возлагаются следующие функции: база фактов представляет исходное состояние проблемы, база правил содержит операторы, которые преобразуют состояние проблемы, приводя его к решению. Машина логического вывода CLIPS сопоставляет эти факты и правила и выясняет, какие из правил можно активизировать. Это выполняется циклически, причем к цикл состоит из трех шагов:

(1) сопоставление фактов и правил;

(2) выбор правила, подлежащего активизации;

(3) выполнение действий, предписанных правилом.

Такой трехшаговый циклический процесс иногда называют "циклом распознавание-действие".

Принципиальным отличием данной системы от аналогов является то, что она полностью реализована на языке С. Причем исходные тексты ее программ опубликованы в Internet.

В CLIPS используется оригинальный LISP – подобный язык программирования, ориентированный на разработку экспертных систем (ЭС). CLIPS поддерживает две парадигмы программирования: объектно-ориентированную и процедурную.

1.1. Основные элементы программирования в CLIPS

CLIPS предоставляет три основных элемента для написания программ:

Простые типы данных;

Конструкции для пополнения базы знаний;

Функции для манипулирования данными.

1.1.1. Простые типы данных

Для представления информации в CLIPS предусмотрено восемь простых типов данных: Float, integer, symbol, string, external-address, fact-address, instance-name, instance-address.

При записи числа используются цифры (0-9), десятичная точка (.), знак (-,+), (e).

Любое число, состоящее только из цифр, перед которыми стоит знак, сохраняется как целое (В CLIPS – **integer,** в С – long integer). Все остальные числа сохраняются как **Float** (в С – double Float). **Symbol** – последовательность символов кода ASCII, причем как только в этой последовательности

встречается символ-разделитель, symbol заканчивается. Символы-разделители: пробел, табуляция, двойные кавычки, (,), &, |, <, >, ~, ;.

String – последовательность символов, заключенных в двойные кавычки (“a and b”). Причем, если внутри строки встречаются двойные кавычки, то перед ними надо поместить символ (\) (“a and \”b”).

1.1.2. Работа с базой знаний в CLIPS. Факты

Факт представляет собой основную единицу данных, используемую правилами. Факты помещаются в текущий список фактов fact-list. Количество фактов с списке и объем информации, содержащейся в факте ограничивается только размером памяти компьютера.

Факт может описываться *индексом* или *адресом.* Всякий раз, когда факт добавляется (изменяется) ему присваивается уникальный целочисленный индекс. Индексы в fact-list начинаются с нуля.

Идентификатор факта – это короткая запись факта, которая состоит из символа факта – f и индекса факта (f-10). Например:

f-0 (today is Sunday)

f-1 (weather is warm)

Факты представимы в двух форматах: *позиционные* и *непозиционные.*

Позиционные факты – состоят из выражения символьного типа, за которым следует последовательность (возможно, пустая) из полей, разделенных пробелами. Вся запись заключается в скобки. Для того чтобы обратиться к информации, содержащейся в позиционном факте, пользователь должен знать какие данные содержаться в факте и в каком поле они хранятся.

Пример:

(altitude is 10000 feet)

(grocery_list bread milk eggs)

(today is Sunday)

(weather is warm)

Поля в позиционных фактах могут быть любого простого типа, за исключением первого поля, которое всегда должно быть типа symbol.

В тексте программы факты можно включать в базу не по одиночке, а целым массивом. Для этого в CLIPS имеется команда **deffacts.**

(deffacts today

(today is Sunday)

(weather is warm)

)

Выражение начинается с команды **deffacts**, затем приводится имя списка фактов, который программист собирается определить (в нашем примере – today), а за ним следуют элементы списка, причем их количество не ограничивается.

В CLIPS существуют следующие зарезервированные слова, которые не могут использоваться как первое поле любого факта: **test, and, or, not, declare, logical, object, exists, forall**.

Непозиционные факты (шаблонные факты) – реализуются через конструкцию, подобную структуре или записи в языках С и Паскаль. Шаблонные факты позволяют задавать имена каждому из полей факта.

Для задания шаблона, который затем может использоваться при доступе к полям по именам, используется конструкция

(deftemplate <name>)

(slot-1)

(slot-2)

. . . . . .

(slot-N)

где <name> – имя шаблона, (slot-N) – именованное поле (или слот). Слоты

могут быть ограничены по типу, значению, числовому диапазону, могут содержать значение по умолчанию. Порядок следования слотов значения не имеет.

Пример:

```
(deftemplate student
“a student record”
(slot name (type STRING))
(slot age (type NUMBER) (default 18))
)
```

Каждое определение шаблона состоит из произвольного имени шаблона, необязательного комментария и некоторого количества определений слотов (начинаются с ключевого слова slot или field). Слот включает поле данных, например name, и тип данных, например STRING. Можно указать и значение по умолчанию, как в приведенном выше примере, где возраст студента по умолчанию равен 18.

Если в программу включено приведенное выше определение шаблона, то выражение

```
(deffacts students
(student (name “fred”))
(student (name “jack”) (age 19))
)
```

приведет к тому, что в базу фактов после выполнения команды reset будет добавлено

```
(student (name “fred”) (age 18))
(student (name “jack”) (age 19))
```

1.1.3. Операции над фактами

Факты можно добавлять к списку фактов (**assert),** удалять из списка

фактов (**retract**), изменять (**modify**), дублировать (**duplicate**)

Рассмотрим работу с базой фактов в CLIPS на практике. Сразу после запуска CLIPS-приложения (**clipswin.exe**) на выполнение на экране появится сообщение, извещающее пользователя, что он работает с интерпретатором.

CLIPS>

В режиме интерпретатора пользователь может использовать множество команд, так можно включить в базу фактов прямо из командной строки с помощью **assert**, например:

CLIPS> (assert (today is Sunday))

<Fact-0>

CLIPS> (assert (weather is warm))

<Fact-l>

Для вывода списка фактов, имеющихся в базе, используется команда **facts**:

CLIPS> (facts)

f-0 (today is Sunday)

f-1 (weather is warm)

Для удаления фактов из базы используется команда **retract**.

CLIPS> (retract 1)

CLIPS> (facts)

f-0 (today is Sunday)

Эти же команды, **assert** и **retract**, используются в выполняемой части правила (заключении правила) и с их помощью выполняется программное изменение базы фактов.

Часто приходится пользоваться и другой командой интерпретатора, **clear**, которая очищает базу фактов (как правило, эта команда доступна в одном из выпадающих меню).

CLIPS> (clear)

CLIPS> (facts)

Массив фактов можно удалить из базы командой **undeffacts**. Например, если в программе был введен массив фактов с именем today:

(deffacts today
(today is Sunday)
(weather is warm)
)

который необходимо удалить, то в командной строке необходимо ввести следующую команду:

CLIPS> (undeffacts today)

Выражение **deffacts** можно вводить и в командную строку интерпретатора, но *лучше записать его в текстовый файл с помощью редактора CLIPS или любого другого текстового редактора* (файлы имеют расширение .clp). Загрузить этот файл в дальнейшем можно с помощью команды в меню File либо из командной строки.

CLIPS> (load “my file”)

Однако после загрузки файла факты не передаются сразу же в базу фактов CLIPS. Команда deffacts просто указывает интерпретатору, что существует массив today, который содержит множество фактов. Собственно загрузка выполняется командой reset.

CLIPS> (reset)

Команда **reset** сначала очищает базу фактов, а затем включает в нее факты из всех ранее загруженных массивов. Она также добавляет в базу единственный определенный системой факт:

f-0 (initial-fact)

Это делается по умолчанию, поскольку иногда имеет смысл включить в программу правило **start rule**, которое может быть сопоставлено с этим фактом и позволит выполнить какие-либо инициализирующие операции.

Можно проследить, как выполняется команда reset, если перед выполнением приведенных выше команд установить режим слежения среды разработки. Для этого нужно вызвать команду **Watch** из меню **Execution** и установить в ней флажок **Facts.**

1.1.4. Работа с базой правил. Правила

В языке CLIPS правила имеют следующий формат:

(defrule <имя правила>

< необязательный комментарий >

< необязательное объявление >

< предпосылка_1 >

< предпосылка_m >

=>

< действие_1 >

< действие_n >

)

Например:

(defrule chores

“Things to do on Sunday”

(declare (salience 10))

(today is Sunday)

(weather is warm)

=>

(assert (wash car))

(assert (chop wood))

)

В этом примере **Chores** – произвольно выбранное имя правила. Предпосылки условной части правила – это:

(today is Sunday)

(weather is warm)

сопоставляются затем интерпретатором с базой фактов, а действия, перечисленные в выполняемой части правила (она начинается после пары символов =>), вставят в базу два факта:

(wash car)

(chop wood)

в случае, если правило будет активизировано. Приведенный в тексте правила комментарий “Things to do on Sunday” (Что сделать в воскресенье) поможет в дальнейшем вспомнить, чего ради это правило включено в программу. Выражение

(declare (salience 10))

указывает на степень важности правила. Пусть, например, в программе имеется другое правило

```
(defrule fun
“Better things to do on Sunday”
(salience 100)
(today is Sunday)
(weather is warm)
=>
(assert (drink beer))
(assert (play guitar))
)
```

Поскольку предпосылки обоих правил одинаковы, то при выполнении оговоренных условий они будут «конкурировать» за внимание интерпретатора,

Предпочтение будет отдано правилу, у которого параметр **salience** имеет более высокое значение, в данном случае – правилу **fun**. Параметру **salience** может быть присвоено любое целочисленное значение в диапазоне **[-10 000, 10 000].** Если параметр **salience** в определении правила опущен, ему по умолчанию присваивается значение 0.

Обычно в определении правила присутствуют и переменные (они начинаются с символа **?**). Если, например, правило

```
(defrule pick-a-chore
"Allocating chores to days"
(today is ?day)
(chore is ?job)
=>
(assert (do ?job on ?day))
)
```

будет сопоставлено с фактами

```
(today is Sunday)
(chore is carwash)
```

то в случае активизации оно включит в базу новый факт

```
(do carwash on Sunday)
```

Аналогично, правило

```
(defrule drop-a-chore
"Allocating chores to days"
(today is ?day)
?chore <- (do ?job on ?day)
=>
(retract ?chore)
)
```

отменит выполнение работ по дому **(?chore**). Обратите внимание на то, что оба экземпляра переменной **?day** должны получить одно и то же значение.

Переменная **?chore** в результате сопоставления должна получить ссылку на факт (это делает оператор **<-**), который мы собираемся исключить из базы. Таким образом, если это правило будет сопоставлено с базой фактов, в которой содержатся

(today is Sunday)

(do carwash on Sunday)

то при активизации правила из базы будет удален факт

(do carwash on Sunday)

Отметим, что факт:

(do carwash on Sunday)

будет сопоставлен с любым из представленных ниже образцов

(do ? ? Sunday)

(do ? on ?)

(do ? on ?when)

Если за префиксом **?** не следует имя переменной, он рассматриваются как универсальный символ подстановки, которому может быть сопоставлен любой элемент.

При написании правил в части посылок иногда требуются некоторые логические операции, например, необходимо указать факты, что «сегодня суббота или воскресенье», «цветок не синий», «шар большой и зеленый». Это реализуется специальными логическими операторами: **«ИЛИ», «НЕ», «И»**, которые обозначаются как |, ~, & соответственно. Таким образом указанные выше факты запишутся следующим образом:

(today is Saturday|Sunday)

(flower is ~blue)

(ball is big&green)

1.1.5. Функции для манипулирования данными. Определение функций

Существует несколько типов функций: *пользовательские* и *системные*. Системные определены внутри среды CLIPS изначально, пользовательские – фрагменты кода, написанные пользователями на CLIPS или C.

Хотя CLIPS не ориентирован на численные вычисления, в нем предусмотрены стандартные математические и арифметические функции: **+, -, *, /, ** ** (возведение в степень), **Abs, Sqrt, Mod, Min, Max**.

Пример 1:

```
(+  2  5  8)
```

Конструкция **deffunction** позволяет пользователю определять новые функции непосредственно в среде CLIPS.

```
(deffunction <имя функции> (<аргумент> ... < аргумент >)
<выражение>
<выражение>
)
```

Пример 2:

```
(deffunction hypotenuse (?a ?b)
(sqrt (+ (* ?a ?a) (* ?b ?b)))
)
```

Аргументы-переменные должны иметь префикс **?**, как это показано в приведенном примере.

Вызовы функций в CLIPS имеют префиксную форму: аргументы стоят после ее названия. Вызов функции производится в скобках:

```
(hypotenuse  7  4 )
```

После открывающейся скобки следует имя функции, затем идут аргументы, каждый из которых отделен одним или несколькими пробелами.

Аргументами функции могут быть данные простых типов, переменные или вызовы других функций.

Функция возвращает результат последнего выражения в списке. Иногда выполнение функции имеет побочные эффекты, как в приведенном ниже примере.

```
(deffunction init (?day)
(reset)
(assert (today is ?day))
)
```

В результате после запуска функции на выполнение командой

```
CLIPS> (init Sunday)
```

будет выполнена команда **reset** и, следовательно, очищена база фактов, а затем в нее будет включен новый факт **(today is Sunday)**.

А в результате запуска функции **hypotenuse** на выполнение командой

```
CLIPS> (hypotenuse 3 4)
```

будет выдан известный ответ

```
CLIPS> 5.0
```

Пример 3 .

```
(deffunction between(?lb ?value ?ub)
                (or (> ?lb ?value) (> ?value ?ub)))
```

Эта функция определяет, попало ли заданное целочисленное значение в диапазон между нижним и верхним пределами.

В некоторых задачах бывает полезным оператор присвоения bind. Например, переменной ?a присваивается значение 4:

```
(bind ?a 4)
```

Для подробного изучения функциональных возможностей языка CLIPS рекомендуем воспользоваться литературным источником [5].

1.2. Наблюдение за процессом интерпретации программы

Теперь на простом примере познакомимся с возможностями, которые предоставляет среда разработки CLIPS в части отладки программы, состоящей из правил и фактов. Введите в текстовый файл правило, а затем загрузите этот файл в среду CLIPS.

```
(defrule start
(initial-fact)
=>
(printout t “hello, world” crlf)
)
```

Здесь стандартная команда **printout t** организует вывод на экран, символ **crlf** – это символ перевода строки.

Выполните команду **reset**. Для этого либо введите эту команду в командной строке интерпретатора

```
CLIPS> (reset)
```

либо выберите в меню команду **Execution->Reset**, либо нажмите **<CTRL+U>.**

Затем запустите интерпретатор. Для этого либо введите эту команду **run** в командную строку интерпретатора

```
CLIPS> (run)
```

либо выберите в меню команду **Execution->Reset**, либо нажмите **<CTRL+R>.**

В ответ программа должна вывести сообщение **hello, world**, знакомое всем программистам мира. Для повторного запуска программы повторите команды

reset и **run**.

Если в меню **Execution->Watch** ранее был установлен флажок **Rules** или перед запуском программы на выполнение вы ввели в командную строку команду **watch rules**, то на экране появится результат трассировки процесса выполнения

CLIPS> (run) FIRE 1

start: f-0

hello, world

В этом сообщении в строке, начинающейся с **FIRE**, выведена информация об активизированном правиле: **start** – это имя правила, а **f-0** – имя факта, который «удовлетворил» условие в этом правиле. Команда **watch** позволяет организовать несколько разных режимов трассировки. Если перед запуском программы вы ввели

CLIPS> (dribble-on “dribble.clp”)

TRUE

то выведенный протокол трассировки будет сохранен в файле **dribble.clp**. Сохранение протокола прекратится после ввода команды

CLIPS> (dribble-off)

TRUE

Это очень удобная опция, особенно на этапе освоения языка.

1.3. GUI-интерфейс CLIPS

Для того чтобы иметь возможность наблюдать за всеми изменениями, происходящими в состоянии CLIPS, используется команда **Window->All Above** Данная команда открывает окна **Facts** (содержит факты из списка фактов) и **Agenda** (содержит все правила из списка активных правил).

Ввести программу в CLIPS можно непосредственно из диалогового окна, появившегося после запуска. Но в этом случае все написанные правила после

закрытия CLIPS будут потеряны. Поэтому текст программы необходимо сохранить в файле. В CLIPS имеется встроенный редактор.

Для загрузки содержимого файла в базу знаний CLIPS, нужно воспользоваться пунктом **Load Constructs** меню **File.**

Для того чтобы CLIPS активизировал начальный факт (initial-fact с идентификатором F-0) необходимо выбрать **Executtion->Reset** . Данная команда удаляет существующие факты из списка фактов, включает в список фактов исходный факт (initial-fact), включает в список фактов все факты, описанные в конструкциях (deffacts).

Затем, по команде **Executtion->Run** CLIPS начнет выполнение всех правил программы. Для сохранения протокола работы программы, а также получения ответа в текстовом файле необходимо сразу после запуска CLIPS выполнить команду **File -> TurnDribbleOn** и в диалоговом окне ввести имя файла, в который будет сохраняться содержимого главного окна CLIPS. После окончания работы программы выполнить **TurnDribbleOff**, по этой команде файл для вывода будет закрыт.

2. Лабораторная работа №1. Решение задач на планирование

Цель работы: ознакомление с основными принципами программирования на языке CLIPS, получение навыков работы в оболочке CLIPS, разработка программы на составление планов действий технической системой в заданной предметной области.

Технические средства: персональный компьютер с операционной системой Windows 9x/Me/NT/2000/XP.

Программные средства: текстовый редактор (Блокнот, WordPad и т.п.), оболочка языка CLIPS версии 6.2 и выше.

2.1. Задачи на планирование действий

Задачи планировщика – определить последовательность действий модуля решения, например системы управления. Традиционное планирование основано на знаниях, поскольку создание плана требует организации частей знаний и частичных планов в процедуру решения. Планирование используется в экспертных системах при рассуждении о событиях, происходящих во времени. Планирование находит применение в производстве, управлении, робототехнике, в задачах понимания естественного языка.

Планы создаются путем поиска в пространстве возможных действий до тех пор, пока не будет найдена последовательность, необходимая для решения задачи. Это пространство представляет состояния мира, котрые изменяются при выполнении каждого действия. Поиск заканчивается, когда достигается целевое состояние (описание мира).

2.2. Пример программы по планированию действий робота – "Робот и ящик"

Имеются 2 комнаты – A и B, в комнате A находится робот, в комнате B – ящик; задача – вытолкнуть ящик в комнату A.

Решим эту задачу с помощью шаблонных фактов. Введем шаблон in, определяющий местоположение предмета:

```
(deftemplate in
(slot object (type SYMBOL))
(slot location (type SYMBOL))
)
```

Слот Object будет задавать название предмета или робота, location – название места, где этот предмет или робот находится.

Чтобы задать роботу конкретную цель действий зададим шаблон goal:

```
(deftemplate goal
(slot object (type SYMBOL))
(slot from (type SYMBOL))
(slot to (type SYMBOL))
)
```

слот object – определяет название объекта, который необходимо переместить, слоты from и to определяют откуда и куда.

На основе шаблонов in и goal запишем начальные факты:

```
(deffacts world
(in (object robot) (location RoomA))
(in (object box) (location RoomB))
(goal (action push) (object box) (from RoomB) (to RoomA))
)
```

Первый факт соответствует тому, что робот находится в комнате A, второй, что ящик в комнате B, третий – перетащить ящик из комнаты B в A.

Заключительным этапом создания данной программы является создание

правил. В данной задаче необходимо реализовать три правила, которые осуществляли бы следующие действия робота:

1) перемещение робота в комнату, где находится объект;
2) перемещение робота с объектом в комнату, указанную в цели;
3) останов программы если цель достигнута.

Реализуем первое действие:

```
(defrule move
(goal (object ?X) (from ?Y))
(in (object ?X) (location ?Y))
?robot-position <- (in (object robot) (location ~?Y))
=>
(modify ?robot-position (location ?Y))
)
```

В данном правиле имеются три предпосылки. В первой предпосылке, использующей шаблон goal задаются значения переменных ?X и ?Y. Во второй определяется наличие объекта ?X в комнате ?Y. В третьей предпосылке проверяется, что местоположение робота не соответствует ?Y и запоминается ссылка на данный факт в переменной ?robot-position. Если все предпосылки данного правила истинны, то с помощью оператора modify меняется значение слота location на значение переменной ?Y факта ?robot-position, т. е. робот перемещается в комнату, в которой находится объект, который необходимо переместить.

Аналогично реализуется правило перемещения робота с ящиком, в комнату, указанную в цели:

```
(defrule push
(goal (object ?X) (from ?Y) (to ?Z))
?object-position <- (in (object ?X) (location ?Y))
```

```
?robot-position <- (in (object robot) (location ?Y))
=>
(modify ?robot-position (location ?Z))
(modify ?object-position (location ?Z))
)
```

В данном случае изменяются два факта, ссылки на которые задаются в переменных ?object-position и ?robot-position: значение слота location меняется на значение переменной ?Z, соответствующей значению куда необходимо переместить предмет роботом.

Останов выполнения программы в CLIPS осуществляется с помощью команды (halt). Условием останова является наличие факта, что предмет, указанный в цели (слот object) находится в комнате, указанной в слоте to:

```
(defrule stop
(goal (object ?X) (to ?Y))
(in (object ?X) (location ?Y))
=>
(halt)
)
```

Полный текст программы представлен в листинге 1.

Листинг 1. Программа «робот и ящик»

```
;; ШАБЛОНЫ
;; Шаблон «цель»
(deftemplate goal
(slot action (type SYMBOL))
(slot object (type SYMBOL))
(slot from (type SYMBOL))
```

```
(slot to (type SYMBOL))
)
;; Шаблон “в”
(deftemplate in
(slot object (type SYMBOL))
(slot location (type SYMBOL))
)
;; ФАКТЫ
;; Робот в комнате A,
;; ящик в комнате B,
;; цель - вытолкнуть ящик в комнату A.
(deffacts world
(in (object robot) (location RoomA))
(in (object box) (location RoomB))
(goal (action push) (object box) (from RoomB) (to RoomA))
)
;;ПРАВИЛА
;; Прекратить процесс, когда цель будет достигнута.
(defrule stop
(goal (object ?X) (to ?Y))
(in (object ?X) (location ?Y))
=>
(halt)
)
;; Если робот отсутствует в том месте, где находится объект,
;; который нужно передвинуть,
;; переместить туда робот.
(defrule move
(goal (object ?X) (from ?Y))
(in (object ?X) (location ?Y))
```

```
?robot-position <- (in (object robot) (location ~?Y))
=>
(modify ?robot-position (location ?Y))
)
;; Если робот и объект не в том помещении,
;; которое указано в цели,
;; переместить туда робот и объект.
(defrule push
(goal (object ?X) (from ?Y) (to ?Z))
?object-position <- (in (object ?X) (location ?Y))
?robot-position <- (in (object robot) (location ?Y))
=>
(modify ?robot-position (location ?Z))
(modify ?object-position (location ?Z))
)
```

Чтобы запустить программу в CLIPS необходимо:

1) скопировать текст программы в любой текстовый редактор (редактор CLIPS, Notepad) и сохранить его с расширением clp, например, **robot.clp**;

2) загрузить этот файл в среду CLIPS с помощью меню File\Load… (Ctrl+L);

3) выполнить команду **reset**. Для этого либо введите эту команду в командной строке интерпретатора:

CLIPS> (reset)

либо выберите в меню команду **Execution->Reset** (**CTRL+U);**

4) в меню **Execution->Watch** установите флажок **Rules,** чтобы наблюдать результат трассировки процесса выполнения;

5) затем запустите интерпретатор. Для этого либо введите эту команду **run** в командную строку интерпретатора:

CLIPS> (run)

либо выберите в меню команду **Execution->Reset** (**CTRL+R).**

2.3. Порядок выполнения работы. Задания

1. Изучение интерфейса оболочки языка CLIPS.
2. Знакомство с основами программирования на языке CLIPS и выполнение примеров из части 1 данного руководства .
3. Выполнение в режиме отладки (включить просмотр правил и фактов) программы "Робот и ящик" (предварительно набрать исходный текст программы из листинга 1 в текстовом редакторе и сохранить под именем robot.clp, затем загрузить в CLIPS).
4. Модифицировать программу "Робот и ящик" таким образом, чтобы ящик необходимо было передвинуть в комнату С. Выполнить в режиме отладки.
5. Разработать программу на планирование действий технического объекта согласно варианту задания, выданного преподавателем.

Варианты заданий.

В каждой задаче необходимо вводить в базу фактов несколько заданий для робота или процесса, которые должны выполняться последовательно.

1. Авиаперевозки. Организовать перевозку с помощью воздушного грузового транспорта. Следует предусмотреть ситуации загрузки и разгрузки грузов в самолет, взлет, перелет из одного аэропорта в другой в другой, посадку, учет веса груза и грузоподъемность самолета.

2. Шиномонтажная мастерская. Робот должен проводить диагностику и смену колеса с пробитой покрышкой. Для каждой марки автомашины определен свой класс колес. Колес ограниченное количество, если какие-либо колеса отсутствуют, программа должна известить об этом.

3. Планирование действий по построению пирамиды из блоков(A,B,C,....) разного размера и разной формы по заранее определенным правилам установки, которые задаются в базе фактов.

4. Робот и ящик. Существуют 4 комнаты, выходящие дверями в коридор. Робот, который может перемещаться из комнат в коридор и обратно, двигать ящики.
5. Разработать предметную область, связанную с управлением грузовым лифтом. Лифт может перемещаться между этажами, согласно вызовам, и перевозить грузы, учитывая их вес.
6. Автомат по продаже воды. Автомат имеет конечное количество газированной воды и несколько видов сиропа. Выдача воды с сиропом и без сиропа выдается в соответствии с номиналом монеты и запросом (без сиропа, с вишневым сиропом, лимонным и т. п.).
7. Музыкальный автомат. Имеется определенное количество пластинок, автомат должен по требованию, заданному в базе фактов устанавливать нужную пластинку, если такой не имеется в базе, сообщить об этом. Предусмотреть счетчик времени (каждый шаг выполнения программы – окончание определенного количества времени), по окончании времени проигрывания пластинки она должна автоматически убираться.
8. Автомат разлива воды в бутылки. Имеется определенное количество воды и бутылки заданной емкости. Необходимо разлить воду в бутылки. Автомат должен сообщать, что вода закончилась или не хватает емкости. Не наполнять емкость, если воды меньше, чем объем заданной емкости.
9. Закачка файлов. Имеется список файлов в очереди закачки и их размер. За каждый шаг программы закачивается определенное количество байт (квота). Если количество байт оставшееся для полного скачивания файла меньше, чем квота, то остаток должен передаваться другому файлу.
10. Гирлянда. Имеются лампочки различных цветов, формы и т. п. Необходимо в базе правил задать правила включения и выключения определенных лампочек в момент времени. Под моментом времени считать один шаг программы.
11. Кофе-машина. В кофе-машину загружаются зерна кофе. Машина берет определенную порцию зерен, перемалывает, варит кофе и подает его.

Предусмотреть возможность ввода нескольких сортов кофе, добавление сливок и сахара по запросу. Количество ресурсов (кофе, сахара, сливок, воды) должно быть ограничено, если какое-либо кофе получить не возможно, автомат должен сообщить об этом.

12. Процесс сборки изделия. Имеются детали, которые участвуют в сборке деталей. Каждая деталь должна использоваться в определенной последовательности. Когда изделии собрано, автомат должен переходить к сборке нового изделия. Процесс прекращается только тогда, когда сборка не возможна (закончились или не хватает какой-либо детали).
13. Продажа билетов. Имеется определенное количество билетов на различные мероприятия. Необходимо по запросу и заданному количеству монет выдавать билеты. Автомат должен выдавать сдачу и оповещать, если билеты на запрашиваемое мероприятие закончилось.
14. Библиотека. Необходимо выдавать книги из заданной базы абонентам по запросу. Книга может отсутствовать в библиотеке или быть на руках у другого абонента.
15. Комплексный обед. Имеется меню с ценами. По предъявленным запросам (перечень блюд, закусок и напитков и их количество) составить комплексные обеды и предъявить цену.

3. Лабораторная работа №2. Решение задач из логики высказываний на CLIPS

3.1. Элементы математической логики. Логика высказываний

При решении логических задач с помощью экспертных систем бывает полезным применение аппарата алгебры высказываний, позволяющего представлять факты и правила в виде логических выражений.

Под ***высказыванием*** p мы будем понимать всякое утвердительное предложение, относительно которого можно сделать заключение, истинно оно или нет. Содержением высказывания не интересуются, интерес представляет лишь истинность или ложность высказывания. Будем считать высказывание истинным, если оно равно 1, ложным – если оно равно 0. Над высказываниями можно производить логические операции.

3.1.1. Основные операции над высказываниями

Для высказываний X и Y можно определить следующие основные логические операции.

1. *Отрицание* $(\neg X)$ – высказывание, которое истинно тогда и только тогда, когда X ложно. В разговорной речи высказыванию $\neg X$ соответствуют фразы: “не X”, “неверно, что X”.

2. *Конъюнкция* $(X \wedge Y)$ – логическим умножение. Высказывание, которое истинно тогда и только тогда, когда истинны оба высказывания. В разговорной речи ей соответствует союз “и” ($X \wedge Y$ – “X и Y”).

3. *Дизъюнкция* $(X \vee Y)$ – логическим сложение. Высказывание, которое ложно тогда и только тогда, когда ложны оба высказывания. В разговорной речи ей соответствует союз “или” ($X \vee Y$ – “X или Y”).

4. *Импликация* $(X \to Y)$ – логическое следование. Высказывание, которое ложно тогда и только тогда, когда X – истинно, а Y – ложно. В разговорной речи импликации соответствуют следующие высказывания: “X

только тогда, когда *Y*", "из *X* следует *Y*", "если *X* то *Y*". При этом *X* – посылка, а *Y* – заключение.

5. *Эквиваленция* ($X \leftrightarrow Y$) – высказывание, которое истинно тогда и только тогда, когда истинностные значения высказываний *X* и *Y* совпадают. В разговорной речи эквиваленция соответствует высказываниям вида: "*X* эквивалентно *Y*", "X тогда и только тогда, когда *Y*", "*X* необходимо и достаточно для *Y*".

Таблицей истинности логических операций называется таблица, в которой отражены результаты операций на всех возможных наборах значений высказываний.

Таблица 1. Таблица истинности для логических операций

A	B	$\neg A$	$\neg B$	$A \wedge B$	$A \vee B$	$A \to B$	$A \leftrightarrow B$
И	И	Л	Л	И	И	И	И
И	Л	Л	И	Л	И	Л	Л
Л	И	И	Л	Л	И	И	Л
Л	Л	И	И	Л	Л	И	И

При помощи операций возможно создавать комбинации из высказываний.

Для сложных высказываний, можно создавать комбинации, построенные из нескольких исходных высказываний посредством применения логических операций $\neg, \wedge, \vee, \to, \leftrightarrow$. Их называют *формулами алгебры высказываний.*

При вычислении по формуле учитывается приоритет логической операции. Перечисленные выше логические операции расположены в порядке убывания приоритета. Изменить порядок логических вычислений можно с помощью расстановки скобок.

Исходные высказывания могут быть постоянными, то есть иметь определенные значения "истина" или "ложь". Если элементарное высказывание не имеет определенного значения, то это переменное высказывание. *Например:*

1) A:="Джек лает", B:="Джек любит кости"

C:= $A \wedge B$ (Джек лает и любит кости) – это постоянное высказывание

2) A(X):="Собака (X) лает", B(X):="Собака (X) любит кости"

C(X):= $A(X) \wedge B(X)$ *(Собака с именем X лает и любит кости) – это переменное высказывание.*

Пропозициональной формулой *(ПФ)* называется логическое выражение, содержащее переменные соответствующие логическим высказываниям, 0, 1 – константы и логические операции $\neg$, $\vee$, $\vee$**,** $\rightarrow$, $\leftrightarrow$, называемые пропозициональными связками, скобки (,) используемые для определения приоритета операций. ПФ определяется индуктивно следующим образом:

1.Отдельно взятая переменная (высказывание) и константа (0, 1) – это ПФ.

2. Если *A* и *B*, составленные из допустимых символов – ПФ, то и $\neg A$, $\neg B$, $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \leftrightarrow B$ – тоже ПФ.

Никаких других ПФ, кроме образованных по правилу 2, нет.

Пример

$p \vee q \wedge r \rightarrow p$ – пропозициональная формула

$p \vee q \wedge r \rightarrow \vee p$ – не пропозициональная формула

Таблицей истинности для ПФ является перечень значений данной ПФ при всех возможных значениях входящих в нее переменных.

Пропозициональная формула называется ***тавтологией***, если на всех значениях входящих в нее переменных она равна 1. Обозначение: $\models A$ - пропозициональная формула *A* есть тавтология.

Приведем ряд тавтологий, могущих оказаться полезными при преобразовании высказываний:

Закон двойного отрицания**:** $\models \neg\neg p \leftrightarrow p$ (1)

Закон исключенного третьего: $\models p \vee \neg p$ (2)

Идемпотентность операций дизъюнкции и конъюнкции:

$\models p \wedge p \leftrightarrow p$ (3)

$\models p \vee p \leftrightarrow p$ (4)

р из конъюнкции**:** $\models p \wedge q \rightarrow p$ (5)

р из дизъюнкции: $\models p \rightarrow p \vee q$ (6)

Коммутативность операций дизъюнкции и конъюнкции:

$|= p \wedge q \leftrightarrow q \wedge p$ (7)

$|= p \vee q \leftrightarrow q \vee p$ (8)

Ассоциативность операций дизъюнкции и конъюнкции:

$|= (p \wedge q) \wedge r \leftrightarrow p \wedge (q \wedge r)$ (9)

$|= (p \vee q) \vee r \leftrightarrow p \vee (q \vee r)$ (10)

Разложение операций дизъюнкции и конъюнкции:

$|= p \vee (q \wedge r) \leftrightarrow (p \vee q) \wedge (p \vee r)$ (11)

$|= p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$ (12)

Правила де Моргана: $|= \neg (\wedge\ p) \leftrightarrow \neg p \vee \neg q$ (13)

$|= \neg (\vee\ p) \leftrightarrow \neg p \wedge \neg q$ (14)

Закон контрапозиции: $|= \rightarrow \quad p \leftrightarrow \neg p \rightarrow \neg q$ (15)

Транзитивность импликации: $|= (p \rightarrow q) \wedge (q \rightarrow r) \leftrightarrow p \rightarrow r$ (16)

Закон косвенного доказательства: $|= (\neg p \rightarrow q) \wedge (\neg p \rightarrow \neg q) \rightarrow p$ (17)

Закон разбора случаев: $|= (p \vee q) \wedge (p \rightarrow r) \wedge (q \rightarrow r) \rightarrow r$ (18)

Транзитивность эквиваленции: $|= (p \leftrightarrow q) \wedge (q \leftrightarrow r) \rightarrow (p \leftrightarrow r)$ (19)

Закон противоположности: $|= (p \leftrightarrow q) \rightarrow (\neg p \leftrightarrow \neg q)$ (20)

Представление единицы:

$|= 1 \leftrightarrow p \vee \neg p$ (21)

$|= 1 \leftrightarrow p \rightarrow p$ (22)

Представление нуля*:*

$|= 0 \leftrightarrow p \wedge \neg p$ (23)

$|= 0 \leftrightarrow \neg (p \rightarrow p)$ (24)

Представление импликации через дизъюнкцию и отрицание**:**

$|= (p \rightarrow q) \leftrightarrow (\neg p \vee q)$ (25)

Представление эквиваленции**:**

$|= (p \leftrightarrow q) \leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$ (26)

$|= (p \leftrightarrow q) \leftrightarrow (\neg p \vee q) \wedge (p \vee \neg q)$ (27)

$|= (p \leftrightarrow q) \leftrightarrow (p \wedge q) \vee (\neg p \wedge \neg q)$ (28)

Представление конъюнкции: $|= p \wedge q \leftrightarrow \neg(p \rightarrow \neg q)$ (29)

Представление дизъюнкции**:** $|= p \vee q \leftrightarrow (\neg p \rightarrow q)$ (30)

3.2. Примеры программ логического доказательства

Рассмотрим **задачу 1**. Кто из учеников A, B, C и D играет, а кто не играет в шахматы, если известно, что:

1) Если A или B играет, то C не играет.

2) Если B не играет, то играют C и D.

3)C играет.

Пусть высказывание P истинно, если ученик P играет в шахматы. Условия 1-3 истинны, т.е. значения этих выражений равны единице. В этом случае мы приходим к системе логических уравнений:

$A \vee B \rightarrow \neg C = 1$ (1)

$\neg B \rightarrow C \wedge D = 1$ (2)

$C = 1$ (3)

Составим таблицу истинности для этих двух выражений:

Таблица 2. Неполная таблица истинности для задачи 1

A	B	C	D	$A \vee B \rightarrow \neg C$	$\neg B \rightarrow C \wedge D$
1	1	1	1	0	1
1	1	1	0	0	1
1	0	1	1	0	1
1	0	1	0	0	0
0	1	1	1	0	1
0	1	1	0	0	1
0	0	1	1	1	1
0	0	1	0	1	0

Так как из условия известно, что ученик C играет, то в данном случае достаточно рассмотреть неполную таблицу истинности, а именно только те случаи, когда $C = 1$. Из таблицы мы можем увидеть единственное решение нашей задачи, когда условия (1) и (2) одновременно истинны – $A = 0$, $B = 0$, $C = 1$, $D = 1$.

Также пользуясь результатами таблицы истинности можно, можно написать небольшую программу на CLIPS, которая представляет наши вычисления в виде манипуляций фактами по определенным правилам.

В начале программы у нас всего один факт - ученик С играет в шахматы:

```
(deffacts situation
(C is playing chess)
)
```

Из табилцы истинности видно, что если $C = 1$, то уравнение (1) истинно тогда и только тогда, когда A и B одновременно ложны. Можно написать следующее правило:

```
(defrule rule_1
(C is playing chess)
=>
(assert (A isn't playing chess))
(assert (B isn't playing chess))
)
```

Из таблицы истинности и уравнения (2) видно, что если B ложь, то истинны C и D, если C или D ложно, то B истинно. При других вариантах значений A, B, C выражение либо ложно, либо вызывает неопределенность, например, из фактов, что C и D истинны не возможно определить истинно B или ложно. Выпишем два соответствующих правила:

```
(defrule rule_2_B_is_false
(B isn’t playing chess)
=>
 (assert (D is playing chess))
```

)

(defrule rule_2_D_is_false
(D isn't playing chess)
=>
(assert (B is playing chess))
)

Конечно, правило rule_2_D_is_false никогда не выполнится, так как в программе нет правил, которые порождали бы факт, соответствующий продукции данного правила, но оно приведено в данном примере для полноты решения поставленной задачи в CLIPS.

Программа останавливается, когда про всех учеников что-нибудь выяснено:

(defrule stop
(A ? playing chess)
(B ? playing chess)
(C ? playing chess)
(D ? playing chess)
=>
(halt)
)

Полный текст программы представлен в листинге 2.

Листинг 2. Программа задачи 1. «кто играет в шахматы?»

```
;;из условия известно, что C - играет
(deffacts situation
(C is playing chess)
)

;;из первого высказывания и факта, что C – играет выясняем,
```

```
;;что A и B не играют
(defrule rule_1
(C is playing chess)
=>
(assert (A isn't playing chess))
(assert (B isn't playing chess))
)

;;из второго высказывания и факта, что B не играет выяснаем,
;;что D играет (C – играет, известно из условия)
(defrule rule_2_B_is_false
(B isn’t playing chess)
=>
 (assert (D is playing chess))
)

;;из второго высказывания и факта, что D не играет выяснаем,
;;что B играет (C – играет, известно из условия)
;;данное правило выполняться не будет никогда!!!!
 (defrule rule_2_D_is_false
(D isn't playing chess)
=>
(assert (B is playing chess))
)

;;останавливаемся после того, когда все про всех выясним
(defrule stop
(A ? playing chess)
(B ? playing chess)
(C ? playing chess)
```

(D ? playing chess)

=>

(halt)

)

Рассмотрим **задачу 2.** Угон машины. Следователь допрашивает 4-х гангстеров по делу о похищении автомобиля.

Джек: Если Том не угонял автомобиль, то его угнал Боб

Боб: Если Джек не угонял, то его угнал Том

Фред: Если Том не угонял, то его угнал Джек

Том: Если Боб не угонял, то его угнал я

Выяснилось, что Боб солгал, а Том сказал правду. Правдивы ли показания Джека и Фреда? Кто угнал машину?

Представим данные высказывания в виде логических выражений, где D, B, F, T – Джек, Боб, Фред и Том соответственно:

Джек: $\neg T \to B$

Боб: $\neg D \to T$

Фред: $\neg T \to D$

Том: $\neg B \to T$

Используя представление импликации через дизъюнкцию и отрицание (25) (см. п. 3.1.1) можно убедиться, что показания Джека и Тома, Боба и Фреда эквивалентны:

Джек: $\neg T \to B = \neg(\neg T) \vee B = T \vee B$

Том: $\neg B \to T = \neg(\neg B) \vee T = B \vee T$

Боб: $\neg D \to T = \neg(\neg D) \vee T = D \vee T$

Фред: $\neg T \to D = \neg(\neg T) \vee D = T \vee D$

Воспользуемся этим при написании программы на CLIPS.

Для решения данной задачи создадим шаблон:

(deftemplate gangster

```
  (slot name (type SYMBOL))
  (slot guilty (type SYMBOL) (default Unk))
  (slot truthful (type SYMBOL) (default Unk))
)
```

В слоте name будет хранится имя подозреваемого, guilty будет определять виновен или нет (соответствие истинным или ложным значениям простого высказывания), truthful – определяет правдиво или ложно показание. Слоты guilty и truthful по умолчанию определяется как Unk, что означает неопределенность. В дальнейшем это будет использовано, чтобы исключить повторное выполнение правил, если значение соответствующего слота уже было определено (Yes или No).

Используя полученный шаблон составим базу фактов в соответствии с условием задачи:

```
(deffacts content
  (gangster (name Jack))
  (gangster (name Bob)(truthful No))
  (gangster (name Fred)(guilty No))
  (gangster (name Tom)(truthful Yes))
)
```

Во всех четырех высказывания фигурируют только имена Тома, Джека и Боба. Следовательно, без нарушения условия задачи можно установить, что Фред не виновен – (gangster (name Fred)(guilty No)).

Так как показания Джека и Тома эквивалентны, то рассмотрим их вместе. Если Джек или Том говорит правду и Том невиновен, то виновен Боб, если не виновен Боб, то виновен – Том. Соответствующие правила будут следующие:

```
(defrule Jack_Tom_truthful_Tom_not_guilty
  (gangster (name Jack|Tom) (truthful Yes))
  (gangster (name Tom) (guilty No))
  ?Bob<-(gangster(name Bob) (guilty Unk))
=>
```

```
  (modify ?Bob(guilty Yes))
)

(defrule Jack_Tom_truthful_Bob_not_guilty
  (gangster (name Jack|Tom) (truthful Yes))
  (gangster (name Bob) (guilty No))
  ?Tom<-(gangster(name Tom) (guilty Unk))
=>
  (modify ?Tom(guilty Yes))
)
```

Обратите внимание, что в первой предпосылке данных правил в слоте name используется связка |, что означает истинность предпосылки если в базе будет факт у которого значение слота name будет иметь значение Jack или Tom и truthful – Yes.

Если Том или Джек сказали не правду, то Том и Боб не виновны:

```
(defrule Jack_Tom_not_truthful
  (gangster (name Jack|Tom) (truthful No))
  ?X<-(gangster (name Bob|Tom) (guilty Unk))
=>
  (modify ?X(guilty No))
)
```

В данном правиле также используется связка | в обоих предпосылках. Данное правило применимо как для Боба, так и Тома, если значение слота guilty равно Unk у соответствующих фактов.

Других правил на основании высказываний Джека и Тома и известных фактов о виновности или невиновности Тома и Боба построить не возможно, так как они несут неопределенность. Например, Если известно, что Джек или Том сказали правду и Том виновен, Боб может быть как виновен, так и невиновен, и наоборот: если Боб виновен, то виновность Тома из показаний Джека и Тома определить не возможно.

Аналогично выводятся правила относительно показаний Боба и Фреда:

```
(defrule Bob_Fred_truthful_Jack_not_guilty
  (gangster (name Bob|Fred) (truthful Yes))
  (gangster (name Jack) (guilty No))
  ?Tom<-(gangster(name Tom) (guilty Unk))
=>
  (modify ?Tom(guilty Yes))
)

(defrule Bob_Fred_truthful_Tom_not_guilty
  (gangster (name Bob|Fred) (truthful Yes))
  (gangster (name Tom) (guilty No))
  ?Jack<-(gangster(name Jack) (guilty Unk))
=>
  (modify ?Jack(guilty Yes))
)

(defrule Bob_Fred_not_truthful
  (gangster (name Bob|Fred) (truthful No))
  ?X<-(gangster (name Jack|Tom) (guilty Unk))
=>
  (modify ?X(guilty No))
)
```

Все предыдущие правила основывались на том, что истинность или ложность высказываний известна, но в задаче есть высказывания, истинность или ложность которых необходимо установить. Для этого составим соответствующие правила:

```
(defrule Jack_Tom_is_truthful
  ?X<-(gangster(name Jack|Tom) (truthful Unk))
  (gangster(name Tom|Bob) (guilty Yes))
```

```
=>
  (modify ?X(truthful Yes))
)

(defrule Jack_Tom_is_not_truthful
  ?X<-(gangster(name Jack|Tom) (truthful Unk))
  (gangster(name Tom) (guilty No))
  (gangster(name Bob) (guilty No))
=>
  (modify ?X(truthful No))
)

(defrule Bob_Fred_is_truthful
  ?X<-(gangster(name Bob|Fred) (truthful Unk))
  (gangster(name Tom|Jack) (guilty Yes))
=>
  (modify ?X(truthful Yes))
)

(defrule Bob_Fred_is_not_truthful
  ?X<-(gangster(name Bob|Fred) (truthful Unk))
  (gangster(name Tom) (guilty No))
  (gangster(name Jack) (guilty No))
=>
  (modify ?X(truthful No))
)
```

Данные правила построены в соответствии с логическими выражениями, построенными на высказываниях из условия задачи. Например, показания Тома и Джека истины, если виновен Том или Боб, а если Тим и Боб не виновны, то Том и Джек сказали неправду.

Если про всех все известно, то программа прекращает работу:

```
(defrule stop
  (gangster (name Jack) (guilty ~Unk) (truthful ~Unk))
  (gangster (name Bob) (guilty ~Unk) (truthful ~Unk))
  (gangster (name Fred) (guilty ~Unk) (truthful ~Unk))
  (gangster (name Tom) (guilty ~Unk) (truthful ~Unk))
=>
  (halt)
)
```

Полный текст программы приведен в листинге 3.

Листинг 3. Программа решения задачи 2. «Кто угнал автомобиль?»

```
;;шаблон
(deftemplate gangster
  (slot name (type SYMBOL));;имя
  (slot guilty (type SYMBOL) (default Unk));;виновность
  (slot truthful (type SYMBOL) (default Unk));;истинно ли высказывание
)

;;вводятся факты в соответствии с условием задачи
(deffacts content
  (gangster (name Jack))
  (gangster (name Bob)(truthful No))
  (gangster (name Fred)(guilty No));;Фред не участвовал
  (gangster (name Tom)(truthful Yes))
)

;;останавливаемся когда про всех все известно
(defrule stop
  (gangster (name Jack) (guilty ~Unk) (truthful ~Unk))
```

```
  (gangster (name Bob) (guilty ~Unk) (truthful ~Unk))
  (gangster (name Fred) (guilty ~Unk) (truthful ~Unk))
  (gangster (name Tom) (guilty ~Unk) (truthful ~Unk))
=>
  (halt)
)

;;в соответствии с показаниями Джека и Тома

(defrule Jack_Tom_truthful_1
  (gangster (name Jack|Tom) (truthful Yes))
  (gangster (name Tom) (guilty No))
  ?Bob<-(gangster(name Bob) (guilty Unk))
=>
  (modify ?Bob(guilty Yes))
)

(defrule Jack_Tom_truthful_2
  (gangster (name Jack|Tom) (truthful Yes))
  (gangster (name Bob) (guilty No))
  ?Tom<-(gangster(name Tom) (guilty Unk))
=>
  (modify ?Tom(guilty Yes))
)

(defrule Jack_Tom_not_truthful
  (gangster (name Jack|Tom) (truthful No))
  ?X<-(gangster (name Bob|Tom) (guilty Unk))
=>
  (modify ?X(guilty No))
```

```
)

;;в соответствии с показаниями Боба и Фреда

(defrule Bob_Fred_truthful_1
  (gangster (name Bob|Fred) (truthful Yes))
  (gangster (name Jack) (guilty No))
  ?Tom<-(gangster(name Tom) (guilty Unk))
=>
  (modify ?Tom(guilty Yes))
)

(defrule Bob_Fred_truthful_2
  (gangster (name Bob|Fred) (truthful Yes))
  (gangster (name Tom) (guilty No))
  ?Jack<-(gangster(name Jack) (guilty Unk))
=>
  (modify ?Jack(guilty Yes))
)

(defrule Bob_Fred_not_truthful
  (gangster (name Bob|Fred) (truthful No))
  ?X<-(gangster (name Jack|Tom) (guilty Unk))
=>
  (modify ?X(guilty No))
)

;;Выясняем кто говорит правду, а кто нет

(defrule Jack_Tom_is_truthful
```

```
  ?X<-(gangster(name Jack|Tom) (truthful Unk))
  (gangster(name Tom|Bob) (guilty Yes))
=>
  (modify ?X(truthful Yes))
)

(defrule Jack_Tom_is_not_truthful
  ?X<-(gangster(name Jack|Tom) (truthful Unk))
  (gangster(name Tom) (guilty No))
  (gangster(name Bob) (guilty No))
=>
  (modify ?X(truthful No))
)

(defrule Bob_Fred_is_truthful
  ?X<-(gangster(name Bob|Fred) (truthful Unk))
  (gangster(name Tom|Jack) (guilty Yes))
=>
  (modify ?X(truthful Yes))
)

(defrule Bob_Fred_is_not_truthful
  ?X<-(gangster(name Bob|Fred) (truthful Unk))
  (gangster(name Tom) (guilty No))
  (gangster(name Jack) (guilty No))
=>
  (modify ?X(truthful No))
)
```

3.3. Варианты заданий

1. В школе было разбито стекло. В содеянном были заподозрены четыре ученика: Миша, Леня, Толя и Дима. Вот что они сказали в кабинете директора:

   Леня: Я не виноват. Я даже не подходил к окну. Миша знает, кто это сделал.

   Дима: Стекло разбил не я. С Мишей я не был знаком до поступления в школу. Это сделал Толя.

   Толя: Я не виновен. Это сделал Миша. Дима говорит неправду, утверждая, что я разбил стекло.

   Миша: Я не виноват. Стекло разбил Леня. Дима может поручиться за меня, т.к. знает меня очень давно.

   При дальнейших расспросах ребята сознались, что 2 фразы из сказанных ими – истинны, а одна, соответственно, ложна. Требуется выяснить, кто разбил стекло.

2. Определите, кто из подозреваемых участвовал в преступлении, если известно:

   a) если Иванов не участвовал или Петров участвовал, то Сидоров участвовал;

   b) если Иванов не участвовал, то Сидоров не участвовал.

3. Аня, Вика и Сергей решили пойти в кино. Учитель, хорошо знавший ребят, высказал предположения:

   a) Аня пойдет в кино только тогда, когда пойдут Вика и Сергей;

   b) Аня и Сергей пойдут в кино вместе или же оба останутся дома;

   c) чтобы Сергей пошел в кино необходимо чтобы пошла Вика.

   Когда ребята пошли в кино, оказалось, что учитель немного ошибся: из трех его утверждений истинными оказались только два. Кто из ребят пошел в кино?

4. Виктор, Роман, Леонид и Сергей заняли на олимпиаде по физике четыре первых места. Когда их спросили о распределении мест, они дали три

ответа:

a) Сергей – первый, Роман – второй;

b) Сергей – второй, Виктор – третий;

c) Леонид – второй, Виктор – четвертый.

Известно, что в каждом ответе только одно утверждение истинно. Как распределились места?

5. Алеша, Боря и Гриша нашли в земле старинный сосуд. Рассматривая удивительную находку, каждый высказал по два предположения:

   Алеша: Это сосуд греческий и изготовлен в V веке.

   Боря: Это сосуд финикийский и изготовлен в III веке.

   Гриша: Это сосуд не греческий и изготовлен в IV веке.

   Учитель истории сказал ребятам, что каждый из них прав только в одном из двух предположений. Где и в каком веке изготовлен сосуд?

6. В нарушении правил обмена валюты подозреваются четыре работника банка – A, B, C, D. Известно, что:

   a) если A нарушил, то и B нарушил правила обмена валюты;

   b) если B нарушил, то и C нарушил или A не нарушал;

   c) если D не нарушил, то A нарушил, а C не нарушал;

   d) если D нарушил, то и A нарушил.

   Кто из подозреваемых нарушил правила обмена валюты?

7. Богданову, Демидову и Смирнову предъявлено обвинение в ограблении ювелирного магазина. С места преступления они скрылись на автомобиле. На следствии Богданов сказал, что они уехали на синем «Москвиче», Демидов сказал, что это была черная «Волга», Смирнов утверждал, что это были «Жигули», но не синие. Известно, что каждый из них правильно назвал либо марку автомобиля, либо цвет. Определите марку автомобиля и ее цвет.

8. Четыре подруги Аня, Маша, Настя, Вика пришли в магазин. Продавец сказал, что осталось только четыре платья: Красное, Розовое, Оранжевое, Синее.

a) Красное платье купила Аня, а розовое – Маша;

b) Аня взяла розовое платье, а Вика купила оранжевое;

c) Настя забрала розовое, а Вика – синее платье.

Кто купил синее платье и какое платье выбрала Вика, если известно, что половина каждого утверждения истинна, а половина – ложь.

9. В олимпиаде по биологии участвовало пять девушек: Алла, Нина, Вика, Рита, Соня. Об итогах олимпиады имеется пять высказываний:

a) первое место заняла Алла, а Рита оказалась третьей;

b) пятой была Вика, а вот Нина поднялась на первое место;

c) первое место заняла Соня, а вот Вика была второй;

d) Рита на последнем, пятом, месте, а Нина была предпоследней;

e) Нина была четвертой, а первой – Алла.

Кто занял первое место и на каком месте была Алла, если известно, что в каждом высказывании одно утверждение правильное, а другое – нет.

10. Перед началом Турнира Четырех болельщики высказали следующие предположения по поводу своих кумиров:

a) Макс победит, Билл – второй;

b) Билл – третий, Ник – первый;

c) Макс – последний, а первый – Джон.

Когда соревнования закончились, оказалось, что каждый из болельщиков был прав только в одном из своих прогнозов.

11. Классный руководитель пожаловался директору, что у него в классе появилась компания из 3х учеников, один из которых всегда говорит правду, другой всегда лжет, а третий говорит через раз то ложь, то правду. Директор знает, что их зовут Коля, Саша и Миша, но не знает, кто из них правдив, а кто – нет. Однажды все трое прогуляли урок астрономии. Директор знает, что никогда раньше никто из них не прогуливал астрономию. Он вызвал всех троих в кабинет и поговорил с мальчиками.

Коля: Я всегда прогуливаю астрономию. Не верьте тому, что скажет Саша.

Саша: Это был мой первый прогул этого предмета.

Миша: Все, что говорит Коля – правда.

Определите, кто из них кто.

12. Однажды Алиса повстречала Льва и Единорога, отдыхавших под деревом. 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. Они 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 сумела вывести, какой день недели был вчера. Что это был за день?

13. Покупатель в каждом из магазинов A, B, C и D сделал по одной покупке и приобрел джойстик, дискеты, бумагу и картридж. Известно, что:

a) джойстик и картридж купил не в A;
b) в C зашел когда уже купил дискеты и бумагу;
c) в D не было ни картриджа, ни дискет;
d) в B приехал, когда джойстик уже был куплен;
e) из D уходил без джойстика.

Необходимо определить в каком магазине были куплены дискеты.

14. Три подразделения A, B, C компании хотят получить по итогам квартала максимальную прибыль. Были сделаны следующие прогнозы:

a) если A получит максимальную прибыль, то максимальную прибыль получат также B и C;
b) либо A и C получат максимальную прибыль одновременно, либо одновременно не получат;
c) для того, чтобы C получило максимальную прибыль, необходимо, чтобы и B получило максимальную прибыль.

По завершении квартала оказалось, что один из трех прогнозов ложен. Какие из названных подразделений получили максимальную прибыль?

15. Есть 2 комнаты. В каждой из комнат будет находиться либо принцесса, либо тигр, хотя вполне может статься, что сразу в обеих комнатах обнаружится по тигру или там окажутся одни лишь принцессы. Уздник должен угадать, кто находится в каждой комнате. Если он угадает, то женится на принцессе, если нет, то его растерзает тигр. На табличках, прикрепленных к дверям каждой из комнат, написано:

    I. В этой комнате находится принцесса, а в другой комнате сидит тигр

    II. В одной из этих комнат находится принцесса; кроме того, в одной из этих комнат сидит тигр

    Причем, известно, что на одной табличке написана правда, а на другой – ложь.

    А вы на месте узника, какую бы дверь открыли?

16. В школе-новостройке в каждой из двух аудиторий может находиться либо кабинет информатики, либо кабинет физики. На дверях аудиторий повесили шутливые таблички. На первой повесили табличку: «По крайней мере, в одной из этих аудиторий размещается кабинет информатики», а на второй аудитории – табличку с надписью «Кабинет физики находится в другой аудитории». Проверяющему, который пришел в школу известно только, что надписи либо истинны, либо обе ложны. Помогите проверяющему найти кабинет информатики.

17. Фамилии командира взвода, заместителя командира взвода и командиров трех отделений – Антипов, Борисов, Васильев, Григорьев, Дмитриев. Фамилии командиров первого и второго отделений начинаются на F и L и они решили подружиться. Дмитриев недавно узнал, что командир взвода и командир второго отделения – двоюродный братья. Григорьев дружит с командиром и его заместителем. У Васильева нет двоюродных братьев. Определите фамилии командира, заместителя командира и командиров отделений.

4. Лабораторная работа №3. Реализация эвристических поисковых алгоритмов на примере алгоритма A*

4.1. Эвристический алгоритм поиска в пространстве состояний

Большинство поисковых задач можно сформулировать как задачи поиска в пространстве состояний пути от исходного состояния заданной задачи до целевого состояния путем повторения возможных преобразований. При этом для организации поиска в пространстве состояний удобно использовать дерево поиска (или его более общую форму – граф).

Одним из подобных алгоритмов поиска является т.н. **алгоритм A***, где используются априорные оценки стоимости пути до целевого состояния, что обеспечивает высокую эффективность поиска.

Основная идея алгоритма состоит в использовании для каждого узла n на графе пространства состояний оценочной функции вида $f(n) = g(n) + h(n)$. Здесь $g(n)$ соответствует расстоянию на графе от узла n до начального состояния, а $h(n)$ – оценка расстояния от узла n до узла, представляющего конечное (целевое) состояние. Чем меньше значение оценочной функции $f(n)$, тем "лучше", т.е. узел n лежит на более коротком пути от исходного состояния к целевому. Идея алгоритма состоит в том, чтобы с помощью $f(n)$ отыскать кратчайший путь на графе от исходного состояния к целевому.

Алгоритм.

Введем следующие обозначения:

s – узел начального состояния;

g– узел конечного (целевого) состояния;

OPEN – список выбранных, но необработанных узлов;

CLOSED – список обработанных узлов.

Шаги:

1. $OPEN := \{s\}$.

2. Если $OPEN := \{\}$, то прекратить выполнение. Пути к целевому состоянию на графе не существует.
3. Удалить из списка *OPEN* узел *n*, для которого $f(n) \le f(m)$ для любого узла *m*, уже присутствующего в списке *OPEN*, и перенести его в список *CLOSED*.
4. Сформировать список очередных узлов, в который возможен переход из узла *n*, и удалить из него все узлы, образующие петли; с каждым из оставшихся связать указатель на узел *n*.
5. Если в сформированном списке очередных узлов присутствует *g*, то завершить выполнение. Сформировать результат – путь, порожденный прослеживанием указателей от узла *g* до узла *s*.
6. В противном случае для каждого очередного узла n', включенного в список выполнить следующую последовательность операций:
   6.1 Вычислить $f(n')$.
   6.2 Если n' не присутствует ни в списке *OPEN*, ни в списке *CLOSED*, добавить его в список, присоединить к нему оценку $f(n')$ и установить обратный указатель на узел *n*.
   6.3 Если n' уже присутствует в списке *OPEN* или в списке *CLOSED*, сравнить новое значение $f(n') = new$ с прежним $f(n') = old$.
   6.4 Если $old \le new$, прекратить обработку нового узла.
   6.5 Если $old > new$, заменить новым узлом прежний в списке, причем, если прежний узел был в списке *CLOSED*, перенести его в список *OPEN*.

4.2. Пример решения задачи поиска в пространстве состояний

Возьмем классическую головоломку – игру в "Восемь". В ней принимает участие 8 пронумерованных фишек, которые могут перемещаться по игровому полю 3x3. Цель состоит в том, чтобы из некоторого случайного расположения фишек перейти к упорядоченному.

Если представить пространство поиска в игре в восемь в виде дерева, то $g(n)$ – это глубина от первой вершины до n-й вершины. В качестве $h(n)$, можно, например, выбрать число шашек, стоящих не на своих местах. Стратегия прохождения вершин – по минимуму оценочной функции.

На рис. 1. показан результат поиска. Выбираем вершину с наименьшим из значений оценочной функции, указанных цифрами в кружочках, применяем оператор и раскрываем вершину, затем создаем дочерние вершины (при этом не возвращаемся к уже появившимся вершинам). Повторяем эту процедуру до целевого состояния.

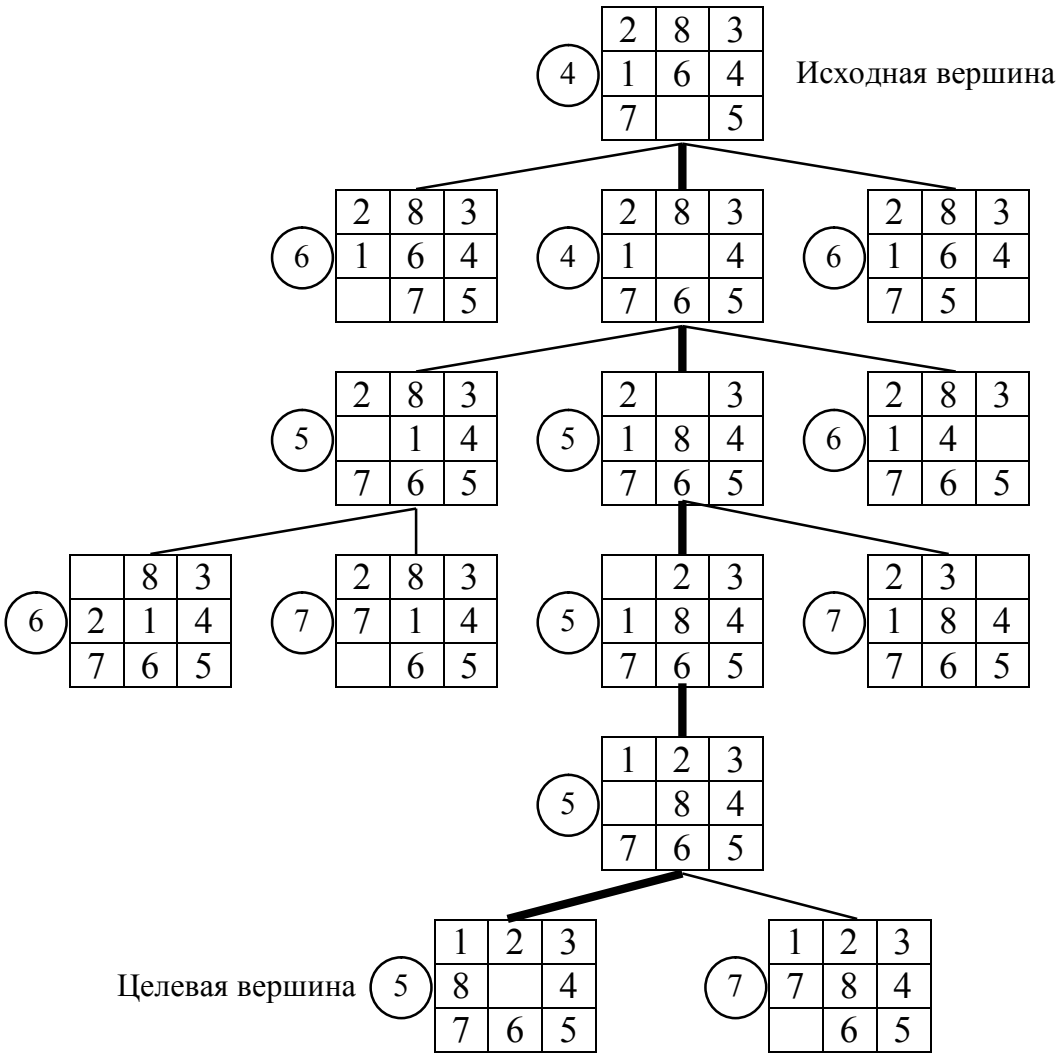

Рис 1. Поиск решения в игре «восемь» по алгоритму A*

Результаты применения оценочной функции приведены на рис 4.1. Значение f для каждой вершины заключены в кружок. Видно, что найден решающий

путь, но использование оценочной функции позволило раскрыть значительно меньше вершин чем при полном поиске на графе.

При реализации решения данной задачи на CLIPS необходимо будет создать шаблон ситуации игры:

```
(deftemplate Field
  (slot LeftTop(type NUMBER))
  (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 Level(type NUMBER))
  (slot Id(type NUMBER) (default 0))
  (slot State(type NUMBER) (default 0))
  (slot From (type NUMBER))
  (slot Exp (type NUMBER))
)
```

Слоты LeftTop, MiddleTop, RightTop, LeftMiddle, MiddleMiddle, RightMiddle, LeftBottom, MiddleBottom и RightBottom являются ячейками игрового поля, в которых задаются номера соответствующих фишек. Если поле пустое, то значение соответствующего ему слота будет равно 0. Level – определяет число шагов от начальной ситуации к текущей. Id – уникальный номер состояния. State – определяет принадлежность данной ситуации множеству *OPEN* если значение 0, *CLOSED* – значение 1, если ситуация является промежуточной на пути к решению, то значение данному слоту будет присваиваться 2 (определяется, когда цель будет достигнута). From – Id ситуации, из которой возможен переход в данное состояние. Exp – значение

целевой функции для данной ситуации.

Так как Id каждой ситуации должно быть уникальным, то целесообразнее присваивать новой ситуации на единицу больший Id, чем у предыдущего. Чтобы хранить номер предыдущей ситуации используем глобальную переменную и соответствующую функцию:

```
(defglobal
  ?*Id* = 0
)
(deffunction Get_Id()
  (bind ?*Id* (+ ?*Id* 1))
  ?*Id*
)
```

Для вычисления значения целевой функции ситуации необходимо число шагов от начальной ситуации к текущей и номера фишек на каждой позиции:

```
(deffunction W(?Level ?LeftTop ?MiddleTop ?RightTop ?RightMiddle ?RightBottom ?MiddleBottom ?LeftBottom ?LeftMiddle)
  (bind ?a ?Level)
  (if (not (= ?LeftTop 1)) then
    (bind ?a (+ 1 ?a))
  )
  (if (not (= ?MiddleTop 2)) then
    (bind ?a (+ 1 ?a))
  )
  (if (not (= ?RightTop 3)) then
    (bind ?a (+ 1 ?a))
  )
  (if (not (= ?RightMiddle 4)) then
    (bind ?a (+ 1 ?a))
  )
  (if (not (= ?RightBottom 5)) then
```

```
  (bind ?a (+ 1 ?a))
 )
 (if (not (= ?MiddleBottom 6)) then
  (bind ?a (+ 1 ?a))
 )
 (if (not (= ?LeftBottom 7)) then
  (bind ?a (+ 1 ?a))
 )
 (if (not (= ?LeftMiddle 8)) then
  (bind ?a (+ 1 ?a))
 )
 ?a
)
```

В базу фактов помещаем начальную ситуацию и инициализируем минимум:

```
(deffacts start
 (min (W 0 2 8 3 4 5 0 7 1))
 (Field (LeftTop 2)      (MiddleTop 8)    (RightTop 3)
        (LeftMiddle 1)    (MiddleMiddle 6) (RightMiddle 4)
        (LeftBottom 7)    (MiddleBottom 0) (RightBottom 5)
        (Level 0) (From 0) (Exp (W 0 2 8 3 4 5 0 7 1)) (Id (Get_Id))
 )
)
```

Далее необходимо создать правила для следующих действий:

1) Если решений найдено, то выделить его.
2) Удалить ситуации, которые не являются промежуточными между начальной и конечной ситуацией.
3) Останов программы если найдено решение или решений нет.
4) Удаление повторяющихся ситуаций.
5) Поиск *min* – минимального значения из значений целевой функции

ситуаций множества *OPEN*.

6) Порождение новых всевозможных ситуаций из текущего, значение целевой функции которого равна *min*.

Обратите внимание, что список действий составлен в порядке приоритета, т. е. прежде чем приступить к следующему шагу необходимо выполнить предыдущий. Аналогично в программе будут расставлены приоритеты у соответствующих правил.

Считается, что решение найдено если на каком-то этапе в программе была порождена ситуация соответствующая конечной. В этом случае необходимо выделить все промежуточные ситуации от начальной до конечной. Данные действия выполняются в следующих правилах:

```
(defrule start_select_answer
  (declare (salience 500))
  ?f<-(Field (LeftTop 1)    (MiddleTop 2) (RightTop 3)
  (LeftMiddle 8) (MiddleMiddle 0) (RightMiddle 4)
  (LeftBottom 7) (MiddleBottom 6) (RightBottom 5) (State ~2) (From ?Id))
=>
  (modify ?f(State 2))
)

(defrule select_answer
  (declare (salience 500))
  (Field (State 2) (From ?Id))
  ?f<-(Field (Id ?Id) (State ~2))
=>
  (modify ?f(State 2))
)
```

В первом правиле помечается ситуация, соответствующая конечной, а во втором с помощью значения в слоте From помеченной ситуации ищется непомеченная ситуация.

Если предыдущие правила не выполнимы и решение найдено, то необходимо удалить лишнее:

```
(defrule delete_not_answer
  (declare (salience 400))
  (Field (State 2))
  ?f<-(Field (State ~2))
=>
  (retract ?f)
)
```

Первая предпосылка гарантирует, что действие данного правила будет выполнено только в том случае, если решение найдено, вторая позволяет выбрать для удаления правила не соответствующие решению.

Возможность останова программы реализуем в следующих правилах:

```
(defrule Stop_1
  (declare (salience 200))
  (not (Field(State 0|2)))
=>
  (halt)
  (printout t "no solutions" crlf)
)

(defrule Stop_2
  (declare (salience 200))
  (Field(State 2))
=>
  (halt)
  (printout t "fined solution" crlf)
)
```

Действия первого правила выполняются, если решение не найдено: в базе фактов нет ситуаций помеченных 0 или 2. Второе правило выполняется, если

решение найдено.

Для удаления повторяющихся ситуаций создадим правило:

```
(defrule move_circle
  (declare (salience 1000))
  (Field (Exp ?X))
  ?Field2<-(Field (Exp ?Y&~?X))
  (test (< ?X ?Y))
=>
  (retract ?Field2)
)
```

Обратите внимание на предпосылку (test (< ?X ?Y)), которая позволяет выбрать для удаления факт, целевая функция которого больше.

Для поиска минимального значения создадим правило, которое будет выполняться, когда будет существовать ситуация из множества *OPEN* у которой значение целевой функции меньше заданной в *min*:

```
(defrule find_min
  (declare (salience 150))
  ?fmin<-(min ?min)
  (Field (Exp ?E& :(< ?E ?min)) (State 0))
=>
 (retract ?fmin)
 (assert (min ?E))
)
```

Для порождения новых ситуаций создадим 9 правил, каждое из которых будет соответствовать местонахождению пустого поля. Так как принцип их реализации идентичный, рассмотрим одно из них (остальные правила см. в листинге 3):

```
(defrule make_new_path_RightBottom
  (declare (salience 100))
  ?fmin<-(min ?min)
```

```
?f<-(Field (State 0) (Level ?L) (Id ?Id)
      (LeftTop ?LT)    (MiddleTop ?MT) (RightTop ?RT)
    (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
    (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom 0) (Exp ?E& :(= ?E ?min)))
   =>
   (modify ?f(State 1))
    (bind ?a (W (+ ?L 1) ?LT ?MT ?RT 0 ?RM ?MB ?LB ?LM))
    (retract ?fmin)
    (assert (min ?a))
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
                   (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle 0)
                   (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RM)
                   (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
            )
    )
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
                   (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
                   (LeftBottom ?LB) (MiddleBottom 0) (RightBottom ?MB)
                   (Level (+ ?L 1)) (From ?Id) (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM ?MB 0 ?LB ?LM)) (Id (Get_Id))
            )
     )
   )
```

В каждом правиле выполняются следующие действия:

- помечается выбранная ситуация как принадлежащая множеству *CLOSED*.
- создаются новые всевозможные ситуации.
- запоминается значение целевой функции одной из новых ситуаций (инициализация поиска *min*).

Полный текст программы приведен в листинге 4.

Листинг 4. Программа поиска решения для игры в «восемь»

```
;;задается шаблон в соответствии с квадратом 3x3
(deftemplate Field
  (slot LeftTop(type NUMBER))
  (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 Level(type NUMBER));;задает уровень в дереве
  (slot Id(type NUMBER) (default 0))
  (slot State(type NUMBER) (default 0));;0 - не рассматривалось, 1 - рассмотрено 2 - соответствует решению
  (slot From (type NUMBER))
  (slot Exp (type NUMBER));;значение целевой функции
)

;глобальная переменная
(defglobal
  ?*Id* = 0
)

;целевая функция
(deffunction W(?Level ?LeftTop ?MiddleTop ?RightTop ?RightMiddle ?RightBottom ?MiddleBottom ?LeftBottom ?LeftMiddle)
  (bind ?a ?Level)
  (if (not (= ?LeftTop 1)) then
```

```
    (bind ?a (+ 1 ?a))
   )
   (if (not (= ?MiddleTop 2)) then
    (bind ?a (+ 1 ?a))
   )
   (if (not (= ?RightTop 3)) then
    (bind ?a (+ 1 ?a))
   )
   (if (not (= ?RightMiddle 4)) then
    (bind ?a (+ 1 ?a))
   )
   (if (not (= ?RightBottom 5)) then
    (bind ?a (+ 1 ?a))
   )
   (if (not (= ?MiddleBottom 6)) then
    (bind ?a (+ 1 ?a))
   )
   (if (not (= ?LeftBottom 7)) then
    (bind ?a (+ 1 ?a))
   )
   (if (not (= ?LeftMiddle 8)) then
    (bind ?a (+ 1 ?a))
   )
   ?a
  )

  ;;определяет идентификатор (чтобы можно найти элементы в последовательности)
  (deffunction Get_Id()
   (bind ?*Id* (+ ?*Id* 1))
```

```
  ?*Id*
)

;;задаем начальное положение
(deffacts start
  (min (W 0 2 8 3 4 5 0 7 1))
  (Field (LeftTop 2)       (MiddleTop 8)    (RightTop 3)
        (LeftMiddle 1)     (MiddleMiddle 6) (RightMiddle 4)
        (LeftBottom 7)     (MiddleBottom 0) (RightBottom 5)
        (Level 0);;это корень дерева
        (From 0) (Exp (W 0 2 8 3 4 5 0 7 1)) (Id (Get_Id))
  )
)

;;удаляем правила, которые встретились повторно
(defrule move_circle
  (declare (salience 1000))
  (Field (Exp ?X));;первый факт
  ?Field2<-(Field (Exp ?Y&~?X) (State 0));;второй факт
  (test (< ?X ?Y));;удаляется ситуация, у которой целевая функция больше
=>
  (retract ?Field2)
  (printout t "delete rule" crlf)
)

;;выбираем узлы из множества Open, и создаем соответствующие пути из
него
;;для этого создается 9 правил с одинаковым приоритетом, что дает
случайность
(defrule make_new_path_LeftTop
```

```
    (declare (salience 100))
    ?fmin <- (min ?min)
    ?f<-(Field (State 0) (Level ?L) (Id ?Id)
         (LeftTop 0)    (MiddleTop ?MT) (RightTop ?RT)
     (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
     (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB) (Exp ?E& :(= ?E
?min)))
   =>
    (printout t ?min " " (fact-slot-value ?f Exp) crlf)
    (modify ?f(State 1))
    (bind ?a (W (+ 1 ?L) ?MT 0 ?RT ?RM ?RB ?MB ?LB ?LM))
    (retract ?fmin)
    (assert (min ?a))
    (assert (Field (LeftTop ?MT) (MiddleTop 0) (RightTop ?RT)
              (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
              (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
              (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
         )
    )
    (assert (Field (LeftTop ?LM) (MiddleTop ?MT) (RightTop ?RT)
              (LeftMiddle 0) (MiddleMiddle ?MM) (RightMiddle ?RM)
              (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
              (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LM ?MT ?RT ?RM
?RB ?MB ?LB 0)) (Id (Get_Id))
         )
     )
     (printout t "LeftTop" crlf)
   )

   (defrule make_new_path_MiddleTop
```

```
(declare (salience 100))
?fmin<-(min ?min)
?f<-(Field (State 0) (Level ?L) (Id ?Id)
      (LeftTop ?LT)    (MiddleTop 0) (RightTop ?RT)
 (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
 (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB) (Exp ?E& :(= ?E
?min)))
=>
(printout t ?min " " (fact-slot-value ?f Exp) crlf)
(modify ?f(State 1))
(bind ?a (W (+ 1 ?L) 0 ?LT ?RT ?RM ?RB ?MB ?LB ?LM))
(retract ?fmin)
(assert (min ?a))
(assert (Field (LeftTop 0) (MiddleTop ?LT) (RightTop ?RT)
          (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
          (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
          (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
      )
)
(assert (Field (LeftTop ?LT) (MiddleTop ?MM) (RightTop ?RT)
          (LeftMiddle ?LM) (MiddleMiddle 0) (RightMiddle ?RM)
          (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
          (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LT ?MM ?RT ?RM
?RB ?MB ?LB ?LM)) (Id (Get_Id))
      )
)
(assert (Field (LeftTop ?LT) (MiddleTop ?RT) (RightTop 0)
          (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
          (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
          (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LT ?RT 0 ?RM ?RB
```

```
?MB ?LB ?LM)) (Id (Get_Id))
            )
        )
        (printout t "MiddleTop" crlf)
      )

      (defrule make_new_path_RightTop
        (declare (salience 100))
        ?fmin<-(min ?min)
        ?f<-(Field (State 0) (Level ?L) (Id ?Id)
             (LeftTop ?LT)    (MiddleTop ?MT) (RightTop 0)
         (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
         (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB) (Exp ?E& :(= ?E
?min)))
      =>
       (printout t ?min " " (fact-slot-value ?f Exp) crlf)
       (modify ?f(State 1))
       (bind ?a (W (+ 1 ?L) ?LT 0 ?MT ?RM ?RB ?MB ?LB ?LM))
       (retract ?fmin)
       (assert (min ?a))
       (assert (Field (LeftTop ?LT) (MiddleTop 0) (RightTop ?MT)
                   (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
                   (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
                   (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
             )
       )
       (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RM)
                   (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle 0)
                   (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
                   (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LT ?MT ?RM 0 ?RB
```

```
?MB ?LB ?LM)) (Id (Get_Id))
            )
        )
        (printout t "RightTop" crlf)
    )

    (defrule make_new_path_LeftMiddle
        (declare (salience 100))
        ?fmin<-(min ?min)
        ?f<-(Field (State 0) (Level ?L) (Id ?Id)
            (LeftTop ?LT)    (MiddleTop ?MT) (RightTop ?RT)
        (LeftMiddle 0) (MiddleMiddle ?MM) (RightMiddle ?RM)
        (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB) (Exp ?E& :(= ?E
?min)))
    =>
        (printout t ?min " " (fact-slot-value ?f Exp) crlf)
        (modify ?f(State 1))
        (bind ?a (W (+ 1 ?L) 0 ?MT ?RT ?RM ?RB ?MB ?LB ?LT))
        (retract ?fmin)
        (assert (min ?a))
        (assert (Field (LeftTop 0) (MiddleTop ?MT) (RightTop ?RT)
                (LeftMiddle ?LT) (MiddleMiddle ?MM) (RightMiddle ?RM)
                (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
                (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
            )
        )
        (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
                (LeftMiddle ?MM) (MiddleMiddle 0) (RightMiddle ?RM)
                (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
                (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LT ?MT ?RT ?RM
```

```
?RB ?MB ?LB ?MM)) (Id (Get_Id))
            )
        )
       (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
                   (LeftMiddle ?LB) (MiddleMiddle ?MM) (RightMiddle ?RM)
                   (LeftBottom 0) (MiddleBottom ?MB) (RightBottom ?RB)
                   (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LT ?MT ?RT ?RM
?RB ?MB 0 ?LB)) (Id (Get_Id))
            )
        )
        (printout t "LeftMiddle" crlf)
      )

      (defrule make_new_path_MiddleMiddle
        (declare (salience 100))
        ?fmin<-(min ?min)
        ?f<-(Field (State 0) (Level ?L) (Id ?Id)
             (LeftTop ?LT)    (MiddleTop ?MT) (RightTop ?RT)
         (LeftMiddle ?LM) (MiddleMiddle 0) (RightMiddle ?RM)
         (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB) (Exp ?E& :(= ?E
?min)))
      =>
       (printout t ?min " " (fact-slot-value ?f Exp) crlf)
       (modify ?f(State 1))
       (bind ?a (W (+ 1 ?L) ?LT 0 ?RT ?RM ?RB ?MB ?LB ?LM))
       (retract ?fmin)
       (assert (min ?a))
       (assert (Field (LeftTop ?LT) (MiddleTop 0) (RightTop ?RT)
                   (LeftMiddle ?LM) (MiddleMiddle ?MT) (RightMiddle ?RM)
                   (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
```

```
            (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
        )
    )
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle ?LM) (MiddleMiddle ?RM) (RightMiddle 0)
            (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LT ?MT ?RT 0 ?RB
?MB ?LB ?LM)) (Id (Get_Id))
        )
    )
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle ?LM) (MiddleMiddle ?MB) (RightMiddle ?RM)
            (LeftBottom ?LB) (MiddleBottom 0) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LT ?MT ?RT ?RM
?RB 0 ?LB ?LM)) (Id (Get_Id))
        )
    )
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle 0) (MiddleMiddle ?LM) (RightMiddle ?RM)
            (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp (W (+ 1 ?L) ?LT ?MT ?RT ?RM
?RB ?MB ?LB 0)) (Id (Get_Id))
        )
    )
    (printout t "MiddleMiddle" crlf)
)

(defrule make_new_path_RightMiddle
 (declare (salience 100))
 ?fmin<-(min ?min)
```

```
?f<-(Field (State 0) (Level ?L) (Id ?Id)
        (LeftTop ?LT)    (MiddleTop ?MT) (RightTop ?RT)
    (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle 0)
    (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB) (Exp ?E& :(= ?E
?min)))
   =>
   (printout t ?min " " (fact-slot-value ?f Exp) crlf)
   (modify ?f(State 1))
   (bind ?a (W (+ ?L 1) ?LT ?MT 0 ?RT ?RB ?MB ?LB ?LM))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop 0)
            (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RT)
            (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
        )
   )
   (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle ?LM) (MiddleMiddle 0) (RightMiddle ?MM)
            (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp (W (+ ?L 1) ?LT ?MT ?RT ?MM
?RB ?MB ?LB ?LM)) (Id (Get_Id))
        )
    )
   (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RB)
            (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom 0)
            (Level (+ ?L 1)) (From ?Id) (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RB 0
?MB ?LB ?LM)) (Id (Get_Id))
        )
```

```
    )
    (printout t "RightMiddle" crlf)
  )

  (defrule make_new_path_LeftBottom
    (declare (salience 100))
    ?fmin<-(min ?min)
    ?f<-(Field (State 0) (Level ?L) (Id ?Id)
        (LeftTop ?LT)   (MiddleTop ?MT) (RightTop ?RT)
     (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
     (LeftBottom 0) (MiddleBottom ?MB) (RightBottom ?RB) (Exp ?E& :(= ?E ?min)))
  =>
    (printout t ?min " " (fact-slot-value ?f Exp) crlf)
    (modify ?f(State 1))
    (bind ?a (W (+ ?L 1) ?LT ?MT ?RT ?RM ?RB ?MB ?LM 0))
    (retract ?fmin)
    (assert (min ?a))
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle 0) (MiddleMiddle ?MM) (RightMiddle ?RM)
            (LeftBottom ?LM) (MiddleBottom ?MB) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
        )
    )
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
            (LeftBottom ?MB) (MiddleBottom 0) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM ?RB 0 ?MB ?LM)) (Id (Get_Id))
        )
```

```
    )
    (printout t "LeftBottom" crlf)
  )

  (defrule make_new_path_MiddleBottom
    (declare (salience 100))
    ?fmin<-(min ?min)
    ?f<-(Field (State 0) (Level ?L) (Id ?Id)
        (LeftTop ?LT)   (MiddleTop ?MT) (RightTop ?RT)
     (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
     (LeftBottom ?LB) (MiddleBottom 0) (RightBottom ?RB) (Exp ?E& :(= ?E ?min)))
  =>
    (printout t ?min " " (fact-slot-value ?f Exp) crlf)
    (modify ?f(State 1))
    (bind ?a (W (+ ?L 1) ?LT ?MT ?RT ?RM ?RB ?LB 0 ?LM))
    (retract ?fmin)
    (assert (min ?a))
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
            (LeftBottom 0) (MiddleBottom ?LB) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
        )
    )
    (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
            (LeftMiddle ?LM) (MiddleMiddle 0) (RightMiddle ?RM)
            (LeftBottom ?LB) (MiddleBottom ?MM) (RightBottom ?RB)
            (Level (+ ?L 1)) (From ?Id) (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM ?RB ?MM ?LB ?LM)) (Id (Get_Id))
        )
```

```
        )
        (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
                    (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
                    (LeftBottom ?LB) (MiddleBottom ?RB) (RightBottom 0)
                    (Level (+ ?L 1)) (From ?Id) (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM 0
?RB ?LB ?LM)) (Id (Get_Id))
                )
        )
        (printout t "MiddleBottom" crlf)
      )

      (defrule make_new_path_RightBottom
        (declare (salience 100))
        ?fmin<-(min ?min)
        ?f<-(Field (State 0) (Level ?L) (Id ?Id)
             (LeftTop ?LT)    (MiddleTop ?MT) (RightTop ?RT)
         (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
         (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom 0) (Exp ?E& :(= ?E
?min)))
      =>
        (printout t ?min " " (fact-slot-value ?f Exp) crlf)
        (modify ?f(State 1))
        (bind ?a (W (+ ?L 1) ?LT ?MT ?RT 0 ?RM ?MB ?LB ?LM))
        (retract ?fmin)
        (assert (min ?a))
        (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
                    (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle 0)
                    (LeftBottom ?LB) (MiddleBottom ?MB) (RightBottom ?RM)
                    (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
                )
```

```
 )
 (assert (Field (LeftTop ?LT) (MiddleTop ?MT) (RightTop ?RT)
          (LeftMiddle ?LM) (MiddleMiddle ?MM) (RightMiddle ?RM)
          (LeftBottom ?LB) (MiddleBottom 0) (RightBottom ?MB)
          (Level (+ ?L 1)) (From ?Id) (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM
?MB 0 ?LB ?LM)) (Id (Get_Id))
        )
 )
 (printout t "RightBottom" crlf)
)

(defrule find_min
 (declare (salience 150))
;;приоритет ниже чем у правила исключающего циклы и выше чем у
правил порождения новых ходов
 ?fmin<-(min ?min)
 (Field (Exp ?E& :(< ?E ?min)) (State 0))
=>
 (retract ?fmin)
 (assert (min ?E))
)

;;если нашли решение, то выделяем его
(defrule start_select_answer
 (declare (salience 500))
 ?f<-(Field (LeftTop 1)   (MiddleTop 2) (RightTop 3)
  (LeftMiddle 8) (MiddleMiddle 0) (RightMiddle 4)
  (LeftBottom 7) (MiddleBottom 6) (RightBottom 5) (State ~2) (From ?Id))
=>
 (printout t "start select answer Id=" (fact-slot-value ?f Id) crlf)
```

```
  (modify ?f(State 2))
)

(defrule select_answer
  (declare (salience 500))
  (Field (State 2) (From ?Id))
  ?f<-(Field (Id ?Id) (State ~2))
=>
  (modify ?f(State 2))
  (printout t "select answer Id=" ?Id crlf)
)

;;удаляем остальные
(defrule delete_not_answer
  (declare (salience 400))
  (Field (State 2))
  ?f<-(Field (State ~2))
=>
  (retract ?f)
  (printout t "delete not answer" crlf)
)

;;делаем останов если решений нет
(defrule Stop_1
  (declare (salience 200))
  (Field(State ?x))
  (not (Field(State 0|2)))
=>
  (halt)
  (printout t "no solutions" crlf)
```

```
)

;;делаем останов если решение есть
(defrule Stop_2
  (declare (salience 200))
  (Field(State 2))
=>
  (halt)
  (printout t "fined solution" crlf)
)
```

4.3. Варианты заданий

Задание. Для задачи из варианта придумать адекватную задаче оценочную функцию и составить программу, реализующую алгоритм A*. Причем программа должна показывать (в любом виде) дерево поиска с указанием значений узлов и их оценок.

1. Трем миссионерам и трем каннибалам необходимо переправиться на другой берег реки. На берегу реки находится лодка (без гребца), которая может вместить не более 2-х человек. Необходимо организовать переправу так, чтобы ни на одном берегу количество каннибалов не превышало количество миссионеров.
2. Три рыцаря, каждый в сопровождении оруженосца, съехались на берегу реки, намереваясь переправиться на другую сторону. Им удалось найти двухместную лодку и переправа произошла бы легко, если бы не одно затруднение: все оруженосцы, словно сговорившись, наотрез отказались оставаться в обществе незнакомых рыцарей без своих хозяев. Необходимо организовать переправу, соблюдая условия оруженосцев.
3. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (5, 2). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x, y) в одну из

трех точек: или точку с координатами (x+3, y), или в точку с координатами (x, y+3), или в точку с координатами (x, y+4). Выигрывает игрок, после хода которого расстояния от фишки до точки с координатами (0, 0) не меньше 13 единиц. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.

4. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (3, 2). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x, y) в одну из трех точек: или в точку с координатами (x+3, y), или в точку с координатами (x, y+2), или в точку с координатами (x, y+4). Выигрывает игрок, после хода которого расстояние от фишки до точки с координатами (0, 0) больше 12 единиц. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
5. На координатной плоскости стоит фишка. Игроки ходят по очереди. В начале игры фишка находится в точке с координатами (0, -5). Ход состоит в том, что игрок перемещает фишку из точки с координатами (x, y) в одну из трех точек: или в точку с координатами (x+4, y), или в точку с координатами (x, y+4), или в точку с координатами (x+4, y+4). Выигрывает игрок, после хода которого расстояние от фишки до точки с координатами (0, 0) не меньше 13 единиц. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
6. Перед игроками лежат две кучи камней, в первой из которых 1, а во второй – 2 камня. У каждого игрока неограниченное количество камней. Игроки ходят по очереди. Ход состоит в том, что игрок или увеличивает в 3 раза число камней в какой-то куче, или добавляет 2 камня в какую-то кучу. Выигрывает игрок, после хода которого общее число камней становится не менее 17 камней. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
7. Перед игроками лежат две кучи камней, в первой из которых 6, а во второй – 5 камней. У каждого игрока неограниченно много камней. Игроки ходят по

очереди. Ход состоит в том, что игрок увеличивает или в 2 раза или в 3 раза число камней в какой-то куче. Выигрывает игрок, после хода которого общее число камней в двух кучах становится менее 48 камней. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.

8. Перед двумя игроками лежат две кучи камней, в первой из которых 3, а во второй – 6 камней. У каждого игрока неограниченно много камней. Игроки ходят по очереди. Ход состоит в том, что игрок или удваивает число камней в куче, или добавляет 2 камня в какую-то кучу. Выигрывает игрок, после хода которого число камней в двух кучах становится не менее 24 камней. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
9. Даны три кучи камней, содержащих соответственно 2, 3 и 4 камня. За один ход разрешается или удвоить количество камней в меньшей куче (если их две – то в каждой из них), или добавить по 1 камню в каждую из всех трех куч. Выигрывает тот игрок, после хода которого во всех трех кучах суммарно становится не менее 23 камней. Игроки ходят по очереди. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
10. Перед игроками лежат две кучки камней, в первой из которых 3, а во второй – 4 камня. У каждого игрока неограниченно много камней. Ходят игроки по очереди. Делая очередной ход, игрок или увеличивает в какой-то кучке число камней в 2 раза, или добавляет в какую-то кучку 3 камня. Выигрывает тот игрок, после хода которого общее число камней в двух кучках становится не менее 23. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
11. Перед игроками лежат две кучки камней, в первой из которых 2, во второй – 3 камня. У каждого игрока неограниченное количество камней. Игроки ходят по очереди. Ход состоит в том, что игрок или увеличивает число камней в какой-то куче в 3 раза, или добавляет 3 камня в любую из куч. Выигрывает игрок, после хода которого общее число камней в двух кучах становится не менее 33. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.

12. Даны две горки фишек, содержащих соответственно 2 и 4 фишки. За один ход разрешается или удвоить количество фишек в какой-нибудь горке, или добавить по две фишки в каждую из двух горок. Выигрывает тот игрок, после чьего хода в двух горках суммарно становится не менее 24 фишек. Игроки ходят но очереди. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
13. Два игрока играют в следующую игру. Перед ними лежат две кучки фишек, в первой из которых 3, а во второй – 5 фишек. У каждого игрока неограниченно много фишек. Ходят игроки по очереди. Делая очередной ход, игрок или увеличивает в какой-то кучке число фишек в 2 раза, или добавляет в какую-то кучку 2 фишки. Выигрывает тот игрок, после хода которого общее число фишек в двух кучках становится не менее 23. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
14. Даны три кучи камней, содержащих соответственно 3, 4, и 5 камней. За один ход разрешается или удвоить количество камней в меньшей куче (если таких две – то лишь в одной из них), или добавить 2 камня в большую из куч (если таких две – то лишь в одну из них). Выигрывает тот игрок, после хода которого во всех трех кучах суммарно становится не менее 23 камней. Игроки ходят по очереди. Выяснить, кто выигрывает при правильной игре – первый или второй игрок.
15. Имеются три колышка A, B, C, и n дисков разного размера, перенумерованных от 1 до n в порядке возрастания их размеров. Сначала все диски надеты на колышек A так, что диски меньшего размера находятся на дисках большего размера. Требуется перенести все диски с колышка A на колышек C соблюдая следующие условия: диски можно переносить только по одному, больший диск можно ставить на меньший.
16. Имеется n населенных пунктов, перенумерованных от 1 до n. Некоторые пары пунктов соединены дорогами. Определить, кратчайший путь из пункта x в пункт y ($1 \le x \le n, 1 \le y \le n$).

5. Лабораторная работа №4. Объектное программирование в CLIPS

5.1. Использование объектно-ориентированных средств в CLIPS

Использование объектно-ориентированных средств в CLIPS позволяет значительно упростить программирование правил, поскольку для обновления данных можно применять механизм передачи и обработки сообщений методами классов.

Общий синтаксис определения класса выглядит следующим образом:

(defclass <name> [<comment>]
(is-a <superclass-name>+)
[<role>]
[<pattern-match-role>]
<slot>*
<handler-documentation>*)
<role> ::= (role concrete | abstract)
<pattern-match-role>
::= (pattern-match reactive | non-reactive)
<slot> ::= (slot <name> <facet>*) |
(single-slot <name> <facet>*) |
(multislot <name> <facet>*)
<facet> ::= <default-facet> | <storage-facet> |
<access-facet> | <propagation-facet> |
<source-facet> | <pattern-match-facet> |
<visibility-facet> | <create-accessor-facet>
<override-message-facet> | <constraint-attributes>
<default-facet> ::=
(default *?DERIVE* | ?NONE | <expression>*) |

(default-dynamic <expression>*)

<storage-facet> ::= (storage *local* | shared)

<access-facet>

::= (access *read-write* | read-only | initialize-only)

<propagation-facet> ::= (propagation *inherit* | no-inherit)

<source-facet> ::= (source *exclusive* | composite)

<pattern-match-facet>

::= (pattern-match *reactive* | non-reactive)

<visibility-facet> ::= (visibility *private* | public)

<create-accessor-facet>

::= (create-accessor *?NONE* | read | write | read-write)

<override-message-facet>

::= (override-message *?DEFAULT* | <message-name>)

<handler-documentation>

::= (message-handler <name> [<handler-type>])

<handler-type> ::= *primary* | around | before | after

5.1.1. Наследование

Если класс Б наследуется от А, то Б является потомком А, а А – родителем Б. Каждый создаваемый пользователем класс должен быть напрямую унаследован хотя бы от одного класса (обычно от OBJECT). Случай прямого наследования от более чем одного класса называется **множественным наследованием**. Новый класс наследует слоты и сообщения от своих родителей. При множественном наследовании необходимо учитывать следующие правила:

1. Класс имеет более высокое старшинство, чем его родители
2. В списке классов-родителей классы должны располагаться по старшинству

3. При конфликте имен слотов или сообщений выбирается слот или сообщение более старшего класса.

Пример 1

(defclass A (is-a USER))

Класс A является прямым наследником класса USER. Список старшинства классов для A: A USER OBJECT.

Пример 2

(defclass B (is-a USER))

Класс B является прямым наследником класса USER. Список старшинства классов для B: B USER OBJECT.

Пример 3

(defclass C (is-a A B))

Класс C является прямым наследником классов A и B. Список старшинства классов для C: C A B USER OBJECT.

Пример 4

(defclass D (is-a B A))

Класс D является прямым наследником классов A и B. Список старшинства классов для D: D B A USER OBJECT.

Пример 5

(defclass E (is-a A C))

В соответствии с правилом 2, A долже быть старше C. В нашем случае, C – это потомок A и является более сташим в соответствии с правилом 1. Ошибка.

Пример 6

defclass E (is-a C A))

Правильное определение класса из примера 5. Список старшинства для E: E C A B USER OBJECT.

5.1.2. Описатели классов

Абстрактные и конкретные классы

Абстрактный класс предназначен только для наследования, на его основе не могут создаваться экземпляры. На основе конкретного класса могут создаваться его экземпляры.

Слоты

Слот – это место для хранения значений поля класса. Каждый экземпляр класса содержит копию всех слотов своего родителя. Количество слотов класса ограничено только размером свободной памяти, имя слота – любой набор символов, за исключением зарезервированных слов. Потомок класса содержит слоты родителя. В случае конфликта имен слотов, он разрешается в соответствии с правилом старшинства.

Пример.

```
(defclass A (is-a USER)
(slot fooA)
(slot barA))

(defclass B (is-a A)
(slot fooB)
(slot barB))
```

Список старшинства для A: A USER OBJECT. Экземпляр класса A будет иметь 2 слота: fooA и barA. Список страшинства для B: B A USER OBJECT. Экземпляр класса B будет иметь 4 слота: fooB, barB, fooA, barA.

Для каждого слота может быть определен набор **фасетов**. Фасеты описывают различные свойства слотов: значения по умолчанию, вид хранения, видимость и т.п. Более подробно фасеты будут рассмотрены ниже.

Создание экземпляра класса производится командой (make-instance a of A)

– создается экземпляр с именем a класса A. Другой вариант (создание массива экземпляра классов):

```
(definstances my_inst
  (a of A)
  (b of A)
  (c of A)
)
```

Тип поля слота

Слот может содержать как одно, так и несколько значений. По умолчанию слот содержит только одно значение. Ключевое слово **multislot** устанавливает тип слота, позволяющий хранить несколько значений, а **slot** или **singleslot** устанавливает тип слота, который может содержать только одно значение. Многозначные слоты кранятся как значения с несколькими полями. Манипуляции с ними производятся посредством стандартных функций **nth$** и **length$.** Для установки значения слота используется функция **slotinsert$**. Слоты с одним значением хранятся в CLIPS как обычные переменные стандартных типов.

Пример

```
CLIPS> (clear)
CLIPS>
(defclass A (is-a USER)
(role concrete)
(multislot foo (create-accessor read)
(default abc def ghi)))
CLIPS> (make-instance a of A)
[a]
CLIPS> (nth$ 2 (send [a] get-foo))
def
```

CLIPS>

Если при создании слота указывается модификатор для создания методов для записи или чтения по умолчанию ((create-accessor read-write)), то экземпляр класса будет реагировать на сообщения get-имя_слота и put-имя_слота соответственно чтением и записью значения слота. Создание обработчиков сообщений будет рассмотрено позже.

5.1.3. Фасеты

Фасет для задания значений по умолчанию

Фасеты используются для задания значений слота по умолчанию при создании экземпляра класса. Фасет default используется для задания статических значений слота. Фасет default-dynamic используется для заданий значения слота, которое задается всякий раз при создании нового экземпляра класса.

Пример

```
CLIPS> (clear)
CLIPS> (setgen 1)
1
CLIPS>
(defclass A (is-a USER)
(role concrete)
(slot foo (default-dynamic (gensym))
(create-accessor read)))
CLIPS> (make-instance a1 of A)
[a1]
CLIPS> (make-instance a2 of A)
[a2]
CLIPS> (send [a1] get-foo)
gen1
```

```
CLIPS> (send [a2] get-foo)
gen2
CLIPS>
```

Фасет Storage

Фасет определяет, будет ли значение слота храниться локально в экземпляре класса (local), либо это значение будет одно для всех экземпляров класса (shared).

Пример

```
CLIPS> (clear)
CLIPS>
(defclass A (is-a USER)
(role concrete)
(slot foo (create-accessor write)
(storage shared)
(default 1))
(slot bar (create-accessor write)
(storage shared)
(default-dynamic 2))
(slot woz (create-accessor write)
(storage local)))
CLIPS> (make-instance a of A)
[a]
CLIPS> (send [a] print)
[a] of A
(foo 1)
(bar 2)
(woz nil)
CLIPS> (send [a] put-foo 56)
```

```
56
CLIPS> (send [a] put-bar 104)
104
CLIPS> (make-instance b of A)
[b]
CLIPS> (send [b] print)
[b] of A
(foo 56)
(bar 2)
(woz nil)
CLIPS> (send [b] put-foo 34)
34
CLIPS> (send [b] put-woz 68)
68
CLIPS> (send [a] print)
[a] of A
(foo 34)
(bar 2)
(woz nil)
CLIPS> (send [b] print)
[b] of A
(foo 34)
(bar 2)
(woz 68)
CLIPS>
```

Фасет типа доступа к слоту

Для слота может быть задано три типа фасетов:

Read-write

Read-only

Initialize-only

Пример работы с разными типами фасетов:

```
CLIPS> (clear)
CLIPS>
(defclass A (is-a USER)
(role concrete)
(slot foo (create-accessor write)
(access read-write))
(slot bar (access read-only)
(default abc))
(slot woz (create-accessor write)
(access initialize-only)))
CLIPS>
(defmessage-handler A put-bar (?value)
(dynamic-put (sym-cat bar) ?value))
CLIPS> (make-instance a of A (bar 34))
[MSGFUN3] bar slot in [a] of A: write access denied.
[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A
FALSE
CLIPS> (make-instance a of A (foo 34) (woz 65))
[a]
CLIPS> (send [a] put-bar 1)
[MSGFUN3] bar slot in [a] of A: write access denied.
[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A
FALSE
```

CLIPS> (send [a] put-woz 1)

[MSGFUN3] woz slot in [a] of A: write access denied.

[PRCCODE4] Execution halted during the actions of message-handler put-bar primary in class A

FALSE

CLIPS> (send [a] print)

[a] of A

(foo 34)

(bar abc)

(woz 65)

CLIPS>

5.1.4. Обработчики сообщений

Изменение значений свойств объектов по правилам ООП производится самими объектами, поэтому в языке CLIPS это реализовано посредством ***обработчиков сообщений.***

Общий синтаксис команды создания обработчика сообщений:

(defmessage-handler <class-name> <message-name>

[<handler-type>] [<comment>]

(<parameter>* [<wildcard-parameter>])

<action>*)

Вызов обработчика сообщений экземпляра класса:

(send [имя_экземпляра] имя_метода параметры)

Обработчик сообщений уникально идентифицируется наименованием, наименованием класса и типом. Для класса обработчик сообщений может задаваться как при создании определения класса, так и после. Заметим, что при создании определения класса создается только заголовок обработчика

сообщений. Собственно программный код обработчика создается позже при посмощи команды **defmessage-handler**.

Пример:

;;создаем класс «прямоугольник» и объявляем у него обработчик сообщений, позволяющий находить его площадь

```
(defclass rectangle (is-a USER)
(slot side-a (default 1))
(slot side-b (default 1))
(message-handler find-area))

;;создаем тело обработчика сообщений
(defmessage-handler rectangle find-area ()
(* ?self:side-a ?self:side-b))

;;создаем ещё один обработчик сообщений, позволяющий напечатать полученную площадь прямоугольника
(defmessage-handler rectangle print-area ()
(printout t (send ?self find-area) crlf))
```

Ссылка на активный (т.е. принимающий сообщение в данный момент) экземпляр сущности может быть получена при помощи переменной ?self. Имя этого параметра зарезервировано.

Пример

```
defclass A (is-a USER)
(role concrete)
(slot foo (default 1))
(slot bar (default 2)))

CLIPS>
(defmessage-handler A print-all-slots ()
```

```
(printout t ?self:foo " " ?self:bar crlf))
CLIPS> (make-instance a of A)
[a]
CLIPS> (send [a] print-all-slots)
1 2
CLIPS>
```

5.2. Пример объектно-ориентированного программирования в CLIPS.

Рассмотрим следующую задачу. Необходимо разработать и реализовать объектную модель автомата по продаже газированной воды. Вода может быть выдана как с сиропом (стоимостью 3 монеты) или без сиропа (стоимостью 1 монета). Количество сиропа и воды ограничено. В листинге 5 представлена реализация данной задачи в CLIPS.

Автомат по продаже воды реализован в виде класса gas-water-automate, родительский класс которого является USER, класс не абстрактный (role concrete). Чтобы класс мог быть использован при сопоставлении образцов во время выполнения правил pattern-match выставлен как reactive. У класса имеются два слота gas-water и syrup, доступные как для чтения, так и для записи (create-accessor read-write). Данные слоты будут использоваться для хранения количества порций воды и сиропа, их тип задается как INTEGER.

Объект данного класса в нашей программе будет называться our-automate. Зададим воды 2, а сиропа 3 порции.

Выдачу воды реализуем в обработчике сообщений getwater, в который в качестве параметра будем передавать номинал монеты. Вызов данного сообщения реализуем в правиле, которое будет выполняться, когда в базе фактов появится факт со словом money и номиналом монеты. Например, чтобы получить воду с сиропом необходимо задать следующий факт:

```
(deffacts world
  (money 3)
```

)

Листинг 5. Программа моделирующая автомат по продаже воды

```
(defclass gas-water-automate
  (is-a USER);;родительский класс
  (role concrete);;класс неабстрактный
  (pattern-match reactive);;класс активный
  (slot gas-water (type INTEGER) (create-accessor read-write));;вода
  (slot syrup (type INTEGER) (create-accessor read-write));;сироп
;;предварительно объявляем обработчик сообщений:
  (message-handler getwater)
)

(definstances automates;;объявляем экземпляр класса
  (our-automate of gas-water-automate
    (gas-water 2)
    (syrup 3)
  )
)

;;обработчик сообщений
(defmessage-handler gas-water-automate getwater (?money)
  (if (= ?money 1) then;;если 1 монета, то выдаем воду без сиропа
    (if (> (dynamic-get gas-water) 0) then;;если воды нет – оповещаем об этом
      (dynamic-put gas-water (- (dynamic-get gas-water) 1))
      (printout t "Your gas-water, please" crlf)
    else
      (printout t "Sorry, no more gas-water" crlf)
    )
  else
```

```
    (if (= ?money 3) then;;если  3 монеты, то выдаем воду с сиропом
      ;;если воды или сиропа нет, то нужно оповестить об этом
      (if (and (> (dynamic-get gas-water) 0) (> (dynamic-get syrup) 0)) then
        (dynamic-put gas-water (- (dynamic-get gas-water) 1))
        (dynamic-put syrup (- (dynamic-get syrup) 1))
        (printout t "Your gas-water with syrup, please" crlf)
      else
        (printout t "Sorry, no more gas-water or syrup" crlf)
      )
    else;;если ввели не 1 и не 3 монеты, то сообщаем об этом
      (printout t "Wrong money" crlf)
    )
  )
)

;;правило, которое выполняется когда в базе фактов есть монеты
(defrule drinkwater
  ?f<-(money ?money)
  =>
;;вызываем обработчик сообщений
  (send [our-automate] getwater ?money)
;;удаляем факт
(retract ?f)
)
```

5.3. Задания

1. Выполните вариант задания, выданного преподавателем из лабораторной работы № 1 с помощью объектно-ориентированных средств в CLIPS.
2. Выполните полученную программу с различными начальными данными.

Литература

1. Питер Джексон. Введение в экспертные системы. М.: Изд. дом "Вильямс", 2001,– 622с.
2. Рассел С., Норвиг.П. Искусственный интеллект: современный подход, 2-е изд..: Пер. С англ. – М.: Изд. дом "Вильямс", 2006, -1408 с.
3. Люгер Дж. Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е изд. ..: Пер. с англ. – М.: Изд. дом "Вильямс", 2003,– 864 с.
4. Адрес языка CLIPS в Интернете: http://www.ghg.net/clips/CLIPS.html
5. А.П. Частиков, Т.А. Гаврилов, Д.Л. Белов. Разработка экспертных систем. Среда CLIPS. – СПб.:БХВ-Петербург, 2003. – 608 с.
6. В.В. Корнеев, А.Ф. Гаев, С.В. Васютин, В.В. Райх . Базы данных. Интеллектуальная обработка информации. – М.: “Нолидж”, 2000

СОДЕРЖАНИЕ

Введение ... 3
1. Основные теоретические сведения ... 3
1.1. Основные элементы программирования в CLIPS ... 4
1.1.1. Простые типы данных ... 4
1.1.2. Работа с базой знаний в CLIPS. Факты ... 5
1.1.3. Операции над фактами ... 7
1.1.4. Работа с базой правил. Правила ... 10
1.1.5. Функции для манипулирования данными. Определение функций 14
1.2. Наблюдение за процессом интерпретации программы ... 16
1.3. GUI-интерфейс CLIPS ... 17
2. Лабораторная работа №1. Решение задач на планирование ... 19
2.1. Задачи на планирование действий ... 19
2.2. Пример программы по планированию действий робота – "Робот и ящик" ... 19
2.3. Порядок выполнения работы. Задания ... 25
3. Лабораторная работа №2. Решение задач из логики высказываний на CLIPS ... 28
3.1. Элементы математической логики. Логика высказываний ... 28
3.1.1. Основные операции над высказываниями ... 28
3.2. Примеры программ логического доказательства ... 32
3.3. Варианты заданий ... 45
4. Лабораторная работа №3. Реализация эвристических поисковых алгоритмов на примере алгоритма A* ... 50
4.1. Эвристический алгоритм поиска в пространстве состояний ... 50
4.2. Пример решения задачи поиска в пространстве состояний ... 51
4.3. Варианты заданий ... 75
5. Лабораторная работа №4. Объектное программирование в CLIPS ... 79
5.1. Использование объектно-ориентированных средств в CLIPS ... 79

5.1.1. Наследование 80
5.1.2. Описатели классов 82
5.1.3. Фасеты 84
5.2. Пример объектно-ориентированного программирования в CLIPS. 90
5.3. Задания 92
Литература 93