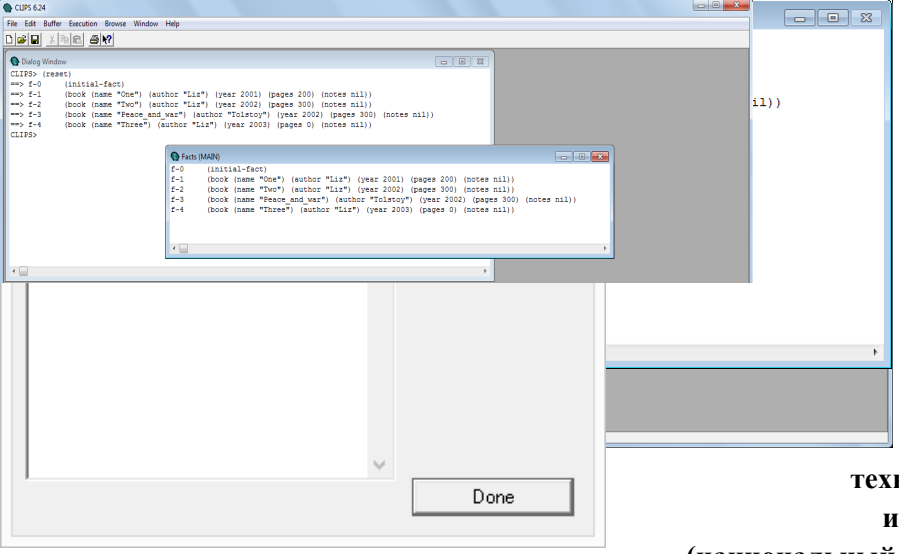

Министерство образования и науки
Российской Федерации

Калужский филиал
федерального государственного
бюджетного образовательного
учреждения высшего образования
**«Московский государственный
технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

ФАКТЫ В СРЕДЕ CLIPS

Методические указания по выполнению лабораторной работы
по курсу «Экспертные системы»

Калуга - 2017

УДК 004.891
ББК 32.813
Б435

Б435 Белов Ю.С., Гришунов С.С. Факты в среде CLIPS. Методические указания по выполнению лабораторной работы по курсу «Экспертные системы». — М.: Издательство МГТУ им. Н.Э. Баумана, 2017. — 33 с.

Методические указания по выполнению лабораторной работы по курсу «Экспертные системы» содержат описание синтаксиса базовой конструкции языка CLIPS — фактов, а также описание основных функций, необходимых для работы с ними.

Предназначены для студентов 4-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

УДК 004.891
ББК 32.813

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Экспертные системы» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания, ориентированные на студентов 4-го курса направления подготовки 09.03.04 «Программная инженерия», содержат краткое описание среды CLIPS и основных функций для работы с фактами в этой среде, учебные примеры с комментариями и пояснениями, а также задание на лабораторную работу.

Методические указания составлены для ознакомления студентов со средой NASA CLIPS v.6.2 и овладения навыками работы с фактами. Для выполнения лабораторной работы студенту необходимы минимальные знания по программированию на высокоуровневом языке программирования.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения лабораторной работы является формирование практических навыков по работе с фактами в среде CLIPS.

Основными задачами выполнения лабораторной работы являются:

1. понять что такое факт применительно к экспертным системам, его структуру и виды
2. изучить основные команды для работы с фактами в среде CLIPS
3. научиться создавать, удалять, изменять, сохранять и загружать факты
4. овладеть навыками очищения и повторной инициализации базы знаний.

Результатами работы являются:

- Созданные в среде CLIPS факты
- Сохраненные в файлах содержимые базы знаний и скриптов
- Подготовленный отчет

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

CLIPS представляет собой современный инструмент, предназначенный для создания экспертных систем *(expert system tool)*. Первоначально аббревиатура *CLIPS* была названием языка — C *Language Integrated Production System* (язык С, интегрированный с продукционными системами), удобного для разработки баз знаний и макетов экспертных систем. *CLIPS* состоит из интерактивной среды - экспертной оболочки со своим способом представления знаний, гибкого и мощного языка и нескольких вспомогательных инструментов. Сейчас, благодаря доброй воле своих создателей, *CLIPS* является абсолютно свободно распространяемым программным продуктом. Всем желающим доступен как сам *CLIPS* последней версии, так и его исходные коды. Официальный сайт *CLIPS* располагается по адресу: http://www.clipsrules.net/. Этот сайт поможет вам получить как сам *CLIPS,* так и всевозможный материал для его изучения и освоения (документацию, примеры, советы специалистов, исходные коды и многое другое).

Система, претендующая называться экспертной, должна обладать знаниями. Эти знания, естественно, должны быть ориентированы на конкретную предметную область, и из этих знаний должно непосредственно вытекать решение проблемы. Именно поэтому знания в экспертных системах предполагают определенную организацию и интеграцию (отдельные факты, сведения должны каким-либо образом соотноситься друг с другом и образовывать между собой определенные связи), т.е. знания должны быть соответствующе представлены.

Применение *CLIPS* для построения систем, основанных на знаниях, может быть обусловлено следующими причинами:

- этот язык является свободно распространяемым программным продуктом;
- его исполнительная система обладает вполне приемлемой производительностью;
- язык имеет четко сформулированный синтаксис;
- в него включено множество опробованных на практике конструкций из других инструментальных средств;
- язык допускает вызов внешних функций, написанных на других языках программирования; в свою очередь модули, написанные на *CLIPS,* могут быть вызваны программами, написанными на других языках;
- язык включает средства, позволяющие комбинировать порождающие правила и объектно-ориентированный подход.

CLIPS предлагает эвристические и процедурные подходы для представления знаний. Также средства *CLIPS* позволяют применять и объектно-ориентированный подход к организации знаний. Кроме того, язык предоставляет возможности комбинировать эти подходы.

В *CLIPS* используется оригинальный LISP-подобный язык программирования, ориентированный на разработку ЭС. Кроме того, *CLIPS* поддерживает еще две парадигмы программирования: объектно-ориентированную и процедурную.

Работа с *CLIPS*

Работа со средой *CLIPS* начинается с запуска файла *«CLIPSWin.exe».* Внешний вид главного окна *CLIPS* показан на рис. 1.

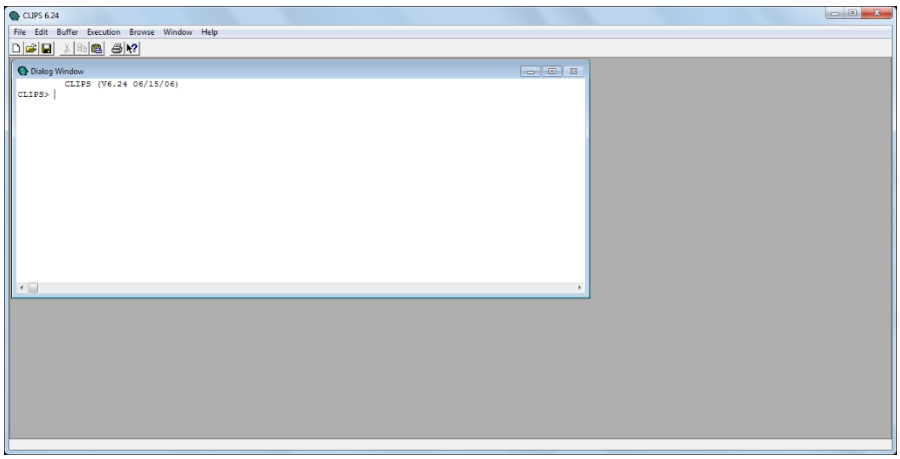

Рис.1 – Главное окно CLIPS

Windows-версия среды *CLIPS* полностью совместима с базовой спецификаций языка. Ввод команд осуществляется непосредственно в главное окно *CLIPS.* Однако, по сравнению с базовой, *Windows-версия* предоставляет множество дополнительных визуальных инструментов, значительно облегчающих жизнь разработчика экспертных систем.

Экспертные системы, созданные с помощью *CLIPS,* могут быть запущены тремя основными способами:

- вводом соответствующих команд и конструкторов языка непосредственно в среду *CLIPS;*
- использованием интерактивного оконного интерфейса *CLIPS* (например, для версий *Windows* или *Macintosh);*
- с помощью программ-оболочек, реализующих свой интерфейс общения с пользователем и использующих механизмы представления знаний и логического вывода *CLIPS.*

Основные элементы языка *CLIPS*

Синтаксис языка *CLIPS* можно разбить на три основных группы элементов, предназначенных для написания программ:

- примитивные типы данных;
- функции, использующиеся для обработки данных;
- конструкторы, предназначенные для создания таких структур языка, как факты, правила, классы и т.д.

CLIPS поддерживает 8 *примитивных типов данных*: `float, integer, symbol, string, external-address, fact-address, in- stance- name, instance-address.`

Для хранения численной информации предназначаются типы `float` и `integer`, для символьной — `symbol` и `string.`

Число в *CLIPS* может состоять только из цифр 0—9, десятичной точки «.», знака «+» или «—» и экспоненциального символа «e» с соответствующим знаком, в случае представления числа в экспоненциальной форме.

Количество значащих цифр зависит от аппаратной реализации. В этой же связи могут возникать ошибки округления.

Как в любом языке программирования, особенную осторожность необходимо проявлять при сравнении чисел с плавающей точкой, а также при сравнении с ними целых чисел.

Примеры целых чисел:

```
237 15 +12 -32
```

Примеры чисел с плавающей точкой:

```
237e3 15.09 +12.0 -32.3e-7
```

Последовательность символов, которая не удовлетворяет числовым типам, обрабатывается как тип данных `symbol`.

Тип данных symbol в *CLIPS* — последовательность символов, состоящая из одного или нескольких любых печатных символов кода *ASCII*. Как только в последовательности символов встречается символ-разделитель, `symbol` заканчивается. Следующие символы служат разделителями: любой непечатный ASCII-символ (включая пробел, символ табуляции, CR, LF), двойные кавычки «"», «(» «)», «&», «|», «<», «~», «;». Символы-разделители не могут включаться в `symbol` за исключением символа «<», который может быть первым символом в `symbol`. Кроме того, symbol не может начинаться с символа «?» или последовательности символов «$?», поскольку эти символы зарезервированы для переменных. Заметим, что *CLIPS* различает регистр символов. Ниже приведены примеры выражений символьного типа:

```
foo Hello B76-HI bad_value
127A 742-42-42 @+=-% Search
```

Тип данных `string` — это последовательность символов, состоящая из нуля и более печатных символов и заключенная в двойные кавычки. Если внутри строки встречаются двойные кавычки, то перед ними необходимо поместить символ «\». То же справедливо и для самого «\».

Несколько примеров:

```
"foo" "a and b" "I number" "a\'quote"
```

Отметим, что строка "abcd" не то же самое, что abcd. Они содержат одинаковые наборы символов, но являются экземплярами различного типа.

Основные команды среды *CLIPS*

Очистка памяти осуществляется при помощи одной из следующих команд: (clear) ИЛИ (reset).

При выполнении команды reset или clear текущий индекс фактов обнуляется.

Команда clear очищает текущий список фактов (а также все определенные конструкторы, которые уже были и еще будут рассмотрены ниже). В отличие от reset, команда clear не добавляет в список фактов факты, объявленные в *deffacts*. Эту команду также можно выполнить, выбрав пункт *«Clear CLIPS»* в меню *«Execution»*. При выборе данной команды на экране появляется диалоговое окно, представленное на рис. 2. Это окно запрашивает подтверждение пользователя на очистку текущей базы знаний.

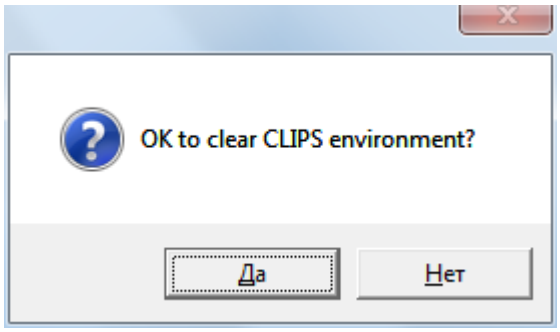

Рис.2 - Подтверждение очистки среды *CLIPS*

В случае если команда была набрана с клавиатуры, никакого подтверждения на выполнение этой операции система не запрашивает. Для очистки системы лучше использовать меню, так как потеря всех текущих данных из базы знаний может оказаться весьма болезненной.

Функция `clear` полностью очищает систему, т.е. удаляет все правила, факты и прочие объекты базы знаний *CLIPS,* добавленные конструкторами, приводит систему в начальное состояние, необходимое для каждой новой программы. Командой выхода при работе с CLIPS является команда `exit`:

(exit)

Действие этой команды аналогично нажатию на кнопку закрытия окна, после её ввода программа будет закрыта.

Для загрузки данных (кроме фактов) из файла в рабочую память можно использовать команду load с указанием пути к файлу:

(load имя_файла)

Для сохранения данных (кроме фактов) из рабочей памяти в файл можно использовать команду save:

(save имя_файла)

Команды сохранения и загрузки в файлы/из файлов дублированы в меню «File».

Для загрузки/сохранения данных из файла/в файл можно использовать как относительный, так и абсолютный путь. Аргументом команд загрузки/сохранения является строка, оканчивающаяся расширением загружаемого/сохраняемого файла.

Как можно заметить, наполнение списка фактов в *CLIPS* довольно кропотливое и длительное занятие. Если фактов достаточно много, этот процесс может растянуться на несколько часов или даже дней. Так как список фактов хранится в оперативной памяти компьютера, теоретически, из-за сбоя компьютера или, например, неожиданного отключения питания, список фактов можно безвозвратно потерять. Чтобы этого не произошло, а также для того чтобы сделать работу по наполнению базы знаний фактами более удобной, *CLIPS* предоставляет команды сохранения и загрузки списка фактов в файл — save-facts и load-facts соответственно.

Команда save-facts сохраняет факты из текущего списка фактов в текстовый файл. На каждый факт отводится одна строчка. Неупорядоченные факты сохраняются вместе с именами слотов. В функции существует возможность ограничить область видимости сохраняемых фактов. Для этого используется аргумент <границы-видимости>. Он может принимать значения local и visible. В случае если этот аргумент принимает значение visible, то сохраняются все факты, присутствующие в данный момент в системе. Если в качестве аргумента используется ключевое слово local, то сохраняются только факты из текущего модуля. По умолчанию аргумент <границы видимости> принимает значение local. После аргумента <границы видимости> может следовать список определенных в системе шаблонов. В этом случае будут сохранены только те факты, которые связаны с указанными шаблонами.

Для загрузки сохраненных ранее файлов используется функция load-facts.Функция имеет следующий формат:

(load-facts имя_файла)

Для сохранения фактов из рабочей памяти в файл можно использовать команду save-facts:

(save-facts имя_файла)

Несмотря на наличие графического пользовательского интерфейса, среду CLIPS трудно назвать дружественной в плане редактирования текстов программ. Поэтому при наборе и редактировании исходных кодов мы рекомендуем пользоваться сторонними продуктами, обеспечивающими более удобный интерфейс (в частности, табуляцию). Исходные коды могут помещаться в любые файлы (не обязательно *.clp),* редактироваться там, а затем загружаться в *CLIPS* с помощью соответствующих команд.

Зачастую (например, для подготовки отчета) необходимо осуществить запись всех операций (всего, что отображается в основном окне) в файл. Для этого подойдут следующие две команды.

Для начала записи информации из основного экрана *CLIPS* в выбранный файл можно использовать команду dribble-on:

(dribble-on имя_файла)

Для прекращения записи информации из основного экрана *CLIPS* в файлы следует использовать команду dribble-off:

```
(dribble-off)
```

ЗАДАЧИ И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Факты — одна из основных форм представления данных в *CLIPS*. Каждый факт представляет собой определенный набор данных, сохраняемый в текущем списке фактов — рабочей памяти системы. Список фактов представляет собой универсальное хранилище фактов и является частью базы знаний. Объем списка фактов ограничен только памятью вашего компьютера. Список фактов хранится в оперативной памяти компьютера, но *CLIPS* предоставляет возможность сохранять текущий список в файл и загружать список из ранее сохраненного файла.

В системе *CLIPS* фактом является список неделимых (или атомарных) значений примитивных типов данных. *CLIPS* поддерживает два типа фактов — упорядоченные факты `(ordered facts)` и неупорядоченные факты или *шаблоны* `(non-ordered facts` или `template facts)`. Ссылаться на данные, содержащиеся в факте, можно либо используя строго заданную позицию значения в списке данных для упорядоченных фактов, либо указывая имя значения для шаблонов.

Факты можно добавлять, удалять, изменять и дублировать, вводя соответствующие команды с клавиатуры либо из программы.

Упорядоченные факты

Упорядоченные (ориентированные) факты состоят из поля, обязательно являющегося данным типа symbol, и следующей за ним, возможно пустой, последовательности полей, разделенных пробелами. Ограничением факта служат круглые скобки.

Первое поле факта определяет так называемое *отношение,* или *связь* факта *(relation)*. Термин «связь» означает, что данный факт принадлежит некоторому определенному конструктором или неявно объявленному шаблону. Приведем несколько примеров фактов:

```
(name tanya olya lena)
(book one two)
```

Количество полей в факте не ограничено. Поля в факте могут хранить данные любого примитивного типа *CLIPS,* за исключением первого поля, которое обязательно должно быть типа `symbol`. Следующие слова зарезервированы и не могут быть использованы в качестве первого поля: `test, and, or, not, declare, logical, object, exist` и `forall`.

Добавление фактов в рабочую память осуществляется при помощи команды `assert`, которая имеет следующий синтаксис:

`(assert` факт)

Функция `assert` — одна из наиболее часто применимых команд в системе *CLIPS*. Без использования этой команды нельзя написать даже самую простую экспертную систему и запустить её на выполнение в среде *CLIPS*.

Функция `assert` позволяет добавлять факты в список фактов текущей базы знаний. Каждым вызовом этой функции можно добавить произвольное число фактов. При использовании команды `assert` необходимо помнить, что первое поле факта обязательно должно быть значением типа `symbol`. В случае удачного добавления фактов в базу знаний функция возвращает адрес последнего добавленного факта. Если во время добавления некоторого факта произошла ошибка, команда прекращает свою работу и возвращает значение `false`.

Приведем пример добавления факта в рабочую память.

Исходный код:

```
(assert (name tanya olya lena))
```

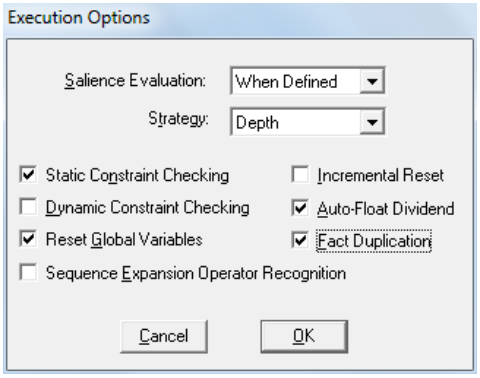

Рис.3 - Диалоговое окно «Execution Options»

Следует заметить, что, в соответствии с принятыми по умолчанию настройками *CLIPS,* нельзя вводить два одинаковых факта, т.е. факта со всеми совпадающими полями (последний введенный будет проигнорирован). Например, попытка добавить два факта `(name tanya olya lena)` приведет к ошибке и функция `assert` вернет значение `false`. Данную установку системы можно изменить с помощью функции `set-fact-duplication`. Кроме того, пользователям *Windows-версии* CLIPS доступен еще один способ настройки. Для этого необходимо открыть диалоговое окно «*Execution Options».* Выбрав пункт «*Options*» из меню *«Execution»,* нужно установить флажок *«Fact Duplication».* Внешний вид этого диалогового окна приведен на рис. 3.

Слотам неупорядоченного факта, значения которых не заданы, будут присвоены значения по умолчанию.

Для просмотра фактов (всех типов), содержащихся в рабочей памяти, можно использовать команду `facts`, которая имеет следующий синтаксис:

`(facts)`

После добавления факта в базу знаний рано или поздно встанет вопрос о том, как его оттуда удалить. Для удаления фактов из текущего списка фактов в системе *CLIPS* предусмотрена функция `retract`. Каждым вызовом этой функции можно удалить произвольное число фактов. Удаление некоторого факта может стать причиной удаления других фактов, которые логически связаны с удаляемым. Функция `retract` имеет следующий синтаксис:

`(retract` номер_факта [номер_факта ...])

Аргумент <номер_факта> может являться либо переменной, связанной с адресом факта с помощью правила, либо индексом факта без префикса (например, 3 для факта с индексом f-3), либо выражением, вычисляющим этот индекс (например, (+ 1 2) для факта с индексом f-3). Если в качестве аргумента функции retract использовался символ «*», то из текущей базы знаний системы будут удалены все факты. Функция `retract` не имеет возвращаемого значения.

Необходимо заметить, что функция retract не оказывает никакого воздействия на индекс следующих добавленных фактов, т.е. этот индекс не обнуляется. Если после удаления всех введенных фактов добавить в систему какой-нибудь факт, то он получит индекс f-n, несмотря на то что список фактов в данный момент пуст.

Неупорядоченные факты

Так как упорядоченный факт для представления информации использует строго заданные позиции данных, то для доступа к ней пользователь должен знать не только то, какие данные сохранены в факте, но и какое поле содержит эти данные. Неупорядоченные (неориентированные) факты (или шаблоны) предоставляют пользователю возможность задавать абстрактную структуру факта путем назначения имени каждому полю. Для создания шаблонов, которые впоследствии будут применяться для доступа к полям факта по имени, используется конструктор `deftemplate`. Конструктор

deftemplate аналогичен определениям записей или структур в таких языках программирования, как *Pascal* или *C*.

Конструктор deftemplate задает имя шаблона и определяет последовательность из нуля или более полей неупорядоченного факта, называемых также слотами. Слот состоит из имени, заданного значением типа symbol, и следующего за ним, возможно пустого, списка полей. Как и факт, слот с обеих сторон ограничивается круглыми скобками. В отличие от упорядоченных фактов слот неупорядоченного факта может жестко определять тип своих значений. Кроме того, слоту могут быть заданы значения по умолчанию.

Для описания шаблона используют команду deftemlate. Общий синтаксис:

```
(deftemlate идентификатор [«комментарий»]
   (slot идентификатор_поля_1)
   (slot идентификатор_поля_2)
   ...
   (slot идентификатор_поля_n)
   (multislot идентификатор_мультиполя_1)
   (multislot идентификатор_мультиполя_2)
   ...
   (multislot идентификатор мультиполя m)
)
```

При создании шаблона с помощью конструктора deftemplate каждому полю можно назначать определенные атрибуты, задающие значения по умолчанию или ограничения на значение слота. Рассмотрим эти атрибуты подробней.

Атрибут default/default-dynamic определяет значение, которое будет использовано в том случае, если при создании факта не задано конкретное значение слота. В *CLIPS* существует два способа определения значения по умолчанию, поэтому в конструкторе deftemplate предусмотрено два различных атрибута, задающих значения по умолчанию: default и default-dynamic.

Атрибут default определяет статическое значение по умолчанию. С его помощью задается выражение, которое вычисляется один раз при конструировании шаблона. Результат вычислений сохраняется вместе с шаблоном. Этот результат присваивается соответствующему слоту в момент объявления нового факта. Если в качестве значения по умолчанию используется ключевое слово ?DERIVE, то это значение будет извлекаться из ограничений, заданных для данного"; слота. По умолчанию для всех слотов установлен атрибут default ?DERIVE.

Если вместо выражения для значения по умолчанию используется ключевое слово ?NONE, то значение поля обязательно должно быть явно задано в момент выполнения операции добавления факта. Добавление факта без определения значений полей с атрибутом default ?NONE вызовет ошибку.

Атрибут default-dynamic предназначен для установки динамического значения по умолчанию. Этот атрибут определяет выражение, которое вычисляется всякий раз при добавлении факта по данному шаблону. Результат вычислений присваивается соответствующему слоту.

Простой слот может иметь только одно значение по умолчанию. У составного слота может быть определено любое количество значений по умолчанию (количество значений по умолчанию должно соответствовать количеству данных, сохраняемых в составном слоте).

В качестве дополнительных параметров поля можно указывать:

1. Тип поля (возможно использование сразу нескольких типов) (type INTEGER FLOAT STRING SYMBOL NUMBER LEXEME EXTERNAL-ADDRESS FACT-ADDRESS INSTANCE-NAME INSTANCE- ADDRESS)
2. Диапазон значений (range нач_значение кон_значение)

3. Список возможных значений
   (allowed-типы список_значений)
   В качестве таких типов могут выступать: allowed-symbols, allowed-strings, allowed-numbers, allowed-integers, allowed-floats, allowed-values
4. Список возможных значений
   (cardinality нач_значение кон_значение)

В конструкторе deftemplate поддерживается проверка статических и динамических ограничений.

Статическая проверка выполняется во время использования определения шаблона некоторой командой или конструктором. Например, для записи значений в слоты шаблона. Иначе говоря, статическая проверка выполняется до запуска программы. При несоответствии используемых значений с установленными ограничениями пользователю выводится соответствующее предупреждение об ошибке.

Ссылка на индекс факта в командах на изменение значения факта или его дублирование не связывает факт с соответствующим шаблоном явно. Это делает статическую проверку неоднозначной. Поэтому в командах, использующих индекс факта, статическая проверка не выполняется. Статическая проверка ограничений включена по умолчанию. Эту установку среды *CLIPS* можно изменить с помощью функции `set-static-constraint-checking.`

Помимо статической, *CLIPS* также поддерживает динамическую проверку ограничений. Если режим динамической проверки ограничений включен, то все новые факты, созданные с использованием некоторого шаблона и имеющие определенные значения, проверяются в момент их добавления в список фактов.

Если нарушение заданных ограничений произойдет в момент выполнения динамической проверки в процессе выполнения программы, то выполнение программы прекращается и пользователю будет выдано соответствующее сообщение.

По умолчанию в *CLIPS* отключен режим динамической проверки ограничений. Эту среду установки можно изменить с помощью функции `set- dinamic-constraint-checking.`

Помимо описанных выше функций для изменения состояний режимов статической и динамической проверки ограничений, пользователям *Windows*- версии среды *CLIPS* доступен визуальный способ настройки этих установок. Для этого необходимо открыть диалоговое окно *«Execution Options»,* выбрав пункт *«Options»* из меню *«Execution».* Внешний вид этого диалогового окна приведен на рис. 3.

Для включения или выключения необходимых режимов проверки ограничений атрибутов выставьте в соответствующее положение флажки *«Static Constraint* Checking» *и* «Dynamic Constraint Checking.» *и нажмите кнопку* «ОК».

Приведем пример шаблона.

Исходный код:

```
(deftemplate book
   (slot name(type STRING))
   (slot author (type STRING))
   (slot year(type INTEGER)
   (default 2006) (range 1400 2006) )
   (slot pages (type INTEGER))
   (multislot notes (cardinality 1 4))
)
```

Пример неориентированного факта (экземпляра шаблона `book`)

Исходный код:

```
(book
   (name "Sun")
   (author "Tom")
   (year 1990)
   (pages 120)
)
```

Приведем пример ввода неориентированного факта (экземпляра шаблона `book`).

Исходный код:

```
(assert (book (name Life) (author "Bob") (year 1890) (pages 300))
```

Для просмотра списка всех шаблонов в рабочей памяти можно использовать команду `list-deftempiates`, которая имеет следующий синтаксис:

`(list-deftemplates)`

Для просмотра конкретного шаблона можно использовать команду `ppdeftemplate`, которая имеет следующий синтаксис:

`(ppdeftemplate` идентификатор_шаблона)

Для удаления из памяти конкретного шаблона можно использовать команду `undeftemplate`, которая имеет следующий синтаксис:

`(undeftemplate` идентификатор_шаблона)

Для удаления всех шаблонов, находящихся в рабочей памяти, используется следующая команда:

`(undeftemplate` *)

Для просмотра всех определенных в текущей базе знаний шаблонов можно воспользоваться специальным инструментом *«Deftemplate Manager»* (Менеджер шаблонов), доступным в Windows-версии среды *CLIPS*. Для запуска менеджера шаблонов воспользуйтесь меню *«Browse»* и выберите пункт *«Deftemplate Manager»*.

Менеджер шаблонов позволяет в отдельном окне просматривать список всех шаблонов, доступных в текущей базе знаний, удалять выбранный шаблон и отображать все его свойства (например, такие как имена и типы слотов). Внешний вид менеджера шаблонов представлен на рис. 4.

После выполненной нами операции в текущей базе знаний находится два шаблона, о чем сообщается в заголовке окна менеджера `(Deftemplate Manager - 2 Items)`. Первый шаблон является предопределенным шаблоном `initial-fact`. Он не имеет слотов и всегда добавляется при запуске среды. Его нельзя удалить с помощью менеджера или просмотреть его определение. Назначение и примеры использования факта `initial-fact` будут рассмотрены ниже. Вторым шаблоном является только что добавленный шаблон `book`. Менеджер шаблонов позволяет вывести в главное окно среды его определение с помощью кнопки *«Pprint»* или удалить его из среды посредством кнопки *«Remove»*.

Флажок *«Watch»* позволяет включать/выключать режим отображения сообщений об использовании шаблонов для каждого присутствующего в системе шаблона в главном окне среды *CLIPS*. Если этот режим включен, пользовать будет получать сообщения при добавлении и удалении неупорядоченных фактов, использующих данный шаблон.

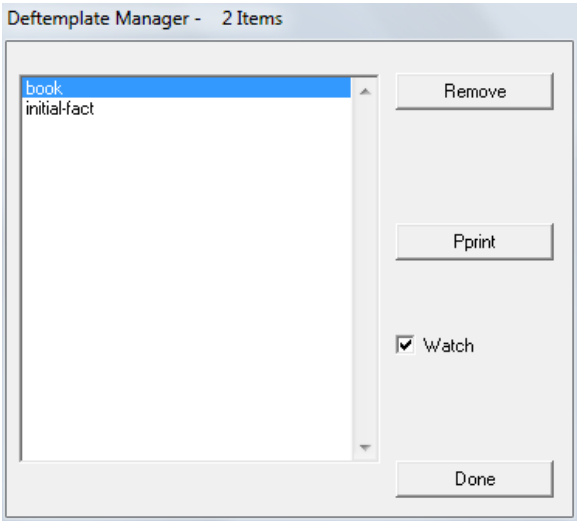

Рис.4 - Окно менеджера шаблонов

Для полноты картины следует также упомянуть о неявно создаваемых шаблонах. При использовании факта или ссылки на упорядоченный факт (например, в правиле) *CLIPS* неявно создает соответствующий шаблон с одним составным слотом. Имя неявно созданного составного слота не отображается при просмотре фактов. Неявно созданным шаблоном можно манипулировать и сравнивать его с любым тождественным, определенным пользователем шаблоном, несмотря на то что он не имеет отображаемой формы.

Предопределенный факт `initial-fact` шаблона `initial-fact` предоставляет удобный способ для запуска программ на языке *CLIPS:* правила, не имеющие условных элементов, автоматически преобразуются в правила с условием, проверяющим наличие факта `initial-fact`. Факт `initial-fact` можно обрабатывать так же, как и все остальные факты *CLIPS,* добавленные пользователем или программой с помощью команды `assert`.

Конструктор deffacts

Помимо конструктора `deftemplates`, *CLIPS* предоставляет конструктор `deffacts`, также предназначенный для работы с фактами. Данный конструктор позволяет определять список фактов, которые будут автоматически добавляться всякий раз после выполнения команды `reset`, очищающей текущий список фактов. Факты, добавленные с помощью конструктора `deffacts`, могут использоваться и удаляться так же, как и любые другие факты, добавленные в базу знаний пользователем или программой с помощью команды `assert`.

Общий синтаксис:

```
(deffacts идентификатор_объединения ["комментарий"]
   (факт_1)
   (факт_2)
   …
   (факт_n)
)
```

Приведем пример использования конструктора `deffacts`. Исходный код:

```
(deffacts books
   (book (name One) (author "Liz") (year 2001) (pages 200) )
   (book (name Two) (author "Liz") (year 2002) (pages 300) )
   (book (name Peace_and_war) (author "Tolstoy") (year 2002)
      (pages 300))
   (book (name Three) (author "Liz") (year 2003) (izd "Sun"))
```

)

Добавление конструктора `deffacts` с именем уже существующего конструктора приведет к удалению предыдущего конструктора, даже если новый конструктор содержит ошибки. В среде *CLIPS* возможно наличие нескольких конструкций `deffacts` одновременно и любое число фактов в них (как упорядоченных, так и неупорядоченных). Факты всех созданных пользователем конструкторов `deffacts` будут добавлены при инициализации системы.

В поля факта могут быть включены динамические выражения, значения которых будут вычисляться при добавлении этих фактов в текущую базу знаний *CLIPS*.

Проверить работу конструктора `deffacts` можно воспользовавшись диалогом *«Watch Options»*. Для этого выберите пункт *«Watch»* меню *«Execution»* или используйте комбинацию клавиш *<Ctrl>+<W>*. В диалоговом окне *«Watch Options»* включите режим просмотра изменения списка фактов, поставив галочку в поле *«Facts»,* как показано на рис. 5.

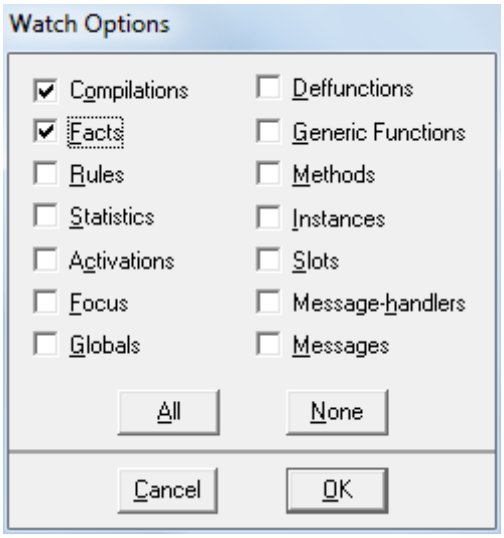

Рис. 5 - Диалоговое окно *«Watch Options»*

После этого нажмите кнопку «ОК» и введите в *CLIPS* приведенный выше конструктор `deffacts.` Затем в меню *«Execution»* выберите пункт «Reset» (комбинация клавиш *<Ctrl>+<E>)*. Если пример был набран правильно, то на экране должны появиться сообщения, аналогичные приведенным на рис. 6.

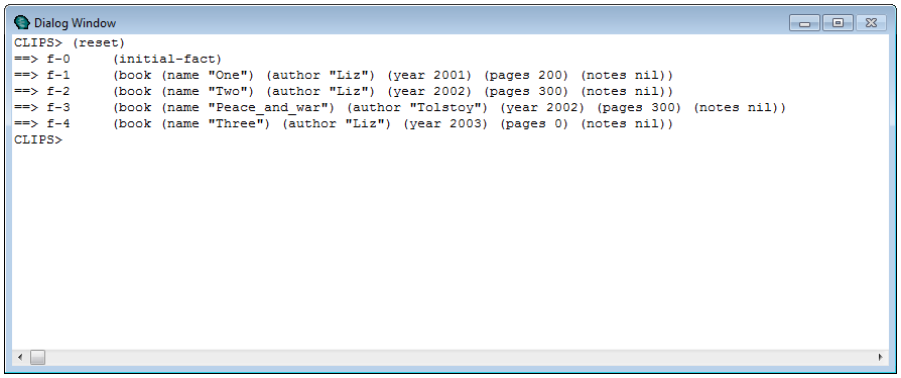

Рис.6 - Просмотр процесса добавления фактов

Как видно из этого примера, диалоговое окно *«Watch Options»* позволяет выводить дополнительную информацию о процессе работы *CLIPS*. Для выполнения этих действий можно воспользоваться командой `watch,` например:

```
(watch facts)
```

Используя функции assert и retract, можно выполнять большинство необходимых для функционирования правил действий. В том числе и изменения предшествующего факта.

Для изменения упорядоченных фактов доступен только этот способ. Для упрощения операции изменения неупорядоченных фактов CLIPS предоставляет функцию modify, которая позволяет изменять значения слотов таких фактов. Modify просто упрощает процесс изменения факта, но её внутренняя реализация эквивалентна вызовам пар функций retract и assert. Вызов modify позволяет изменять только один факт. В случае удачного выполнения функция возвращает новый индекс модифицированного факта.

Если в процессе выполнения произошла какая-либо ошибка, то пользователю выводится соответствующее предупреждение и функция возвращает значение false.

Синтаксис команды modify:

```
(modify идентификатор_факта (новое_значение)*)
```

Обратите внимание на движение фактов в базе знаний *CLIPS* при выполнении функции modify — сначала удаляется старый факт, а затем добавляется новый факт, идентичный предыдущему, но с новым значением заданного слота.

Если в шаблоне заданного факта отсутствует слот, значение которого требуется изменить, *CLIPS* выведет соответствующее сообщение об ошибке. Если заданный факт отсутствует в списке фактов, пользователь также получит соответствующее предупреждение.

Помимо функции modify, в *CLIPS* существует еще одна очень полезная функция, упрощающая работу с фактами, — функция duplicate. Эта функция создает новый неупорядоченный факт заданного шаблона и копирует в него определенную пользователем группу полей уже существующего факта того же шаблона. По выполняемым действиям функция duplicate аналогична modify, за исключением того, что она не удаляет старый факт из списка фактов. Одним вызовом функции duplicate можно создать одну копию некоторого заданного факта. Как и функция modify, duplicate в случае удачного выполнения возвращает индекс нового факта, а в случае неудачи — значение false.

Синтаксис команды duplicate:

```
(duplicate идентификатор_факта (новое_значение)*)
```

Если добавляемый с помощью duplicate факт уже присутствует в списке фактов, будет выдана соответствующая информация об ошибке и возвращено значение false. Факт при этом добавлен не будет. Это поведение можно изменить, разрешив существование одинаковых фактов в базе знаний.

Кроме функции assert, *CLIPS* предоставляет еще одну функцию, полезную при добавлении фактов, — assert-string. Эта функция принимает в качестве единственного аргумента символьную строку, являющуюся текстовым представлением факта (в том виде, в котором вы набираете его, например, в функции assert), и добавляет его в список фактов. Функция assert-string может работать как с упорядоченными, так и с неупорядоченными фактами. Одним вызовом функции assert-string можно добавить только один факт.

Синтаксис команды assert-string:

```
(assert-string Символьная <строка>)
```

Строковое выражение должно быть заключено в кавычки. Функция преобразует заданное строковое выражение в факт *CLIPS,* разделяя отдельные слова на поля, с учетом определенных в системе на текущий момент шаблонов. Если в строке необходимо записать внутреннее строковое выражение, представляющее, скажем, некоторое поле, то для включения в строковое выражение символа кавычек используется *обратная косая черта (backslash).*

Приведем пример использования команды assert-string.

Исходный код:

```
(assert-string "book-name \" Peace and_war \"")
```

Для добавления содержащегося в поле символа обратной косой черты используйте её дважды. Если обратная косая черта должна содержаться внутри подстроки, её необходимо использовать четыре раза.

Если добавление факта прошло удачно, функция возвращает индекс только что добавленного факта, в противном случае функция возвращает сообщение об ошибке и значение `false`. Функция `assert-string` не позволяет добавлять факт в случае, если такой факт уже присутствует в базе знаний (если вы еще не включили возможность присутствия одинаковых фактов).

Так же как и для конструкторов `deftemplate`, *CLIPS* предоставляет визуальный инструмент для манипуляции с определенными в данный момент в системе конструкторами `deffacts` – *«Deffacts Manager»* (Менеджер предопределенных фактов). Для запуска *«Deffacts Manager»* в меню *«Browse»* выберите пункт «Deffacts Manager». Внешний вид менеджера приведен на рис. 7.

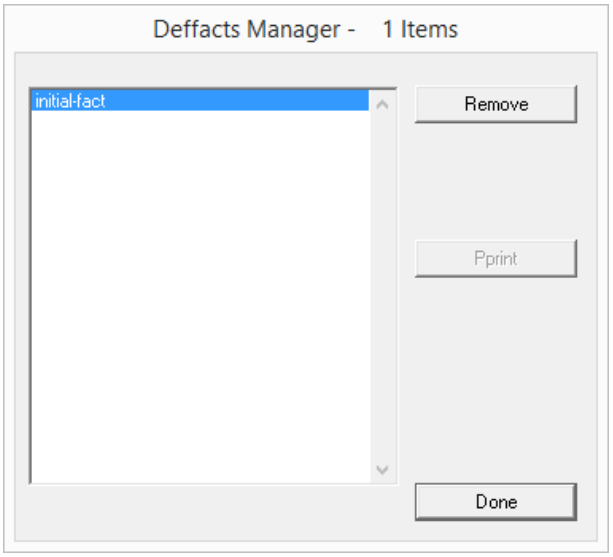

Рис.7 - Окно *«Deffacts Manager»*

Менеджер позволяет выводить в основное окно *CLIPS* информацию об определениях существующих в данный момент в системе конструкторов `deffacts` с помощью кнопки «Pprint», кроме `deffacts initial-fact`, и удалять любой существующий конструктор.

Также для просмотра всех находящихся в рабочей памяти фактов можно использовать окно «*Facts* `(MAIN)`» (рис. 8), которое появится при установке одноименного флага меню «*Window*».

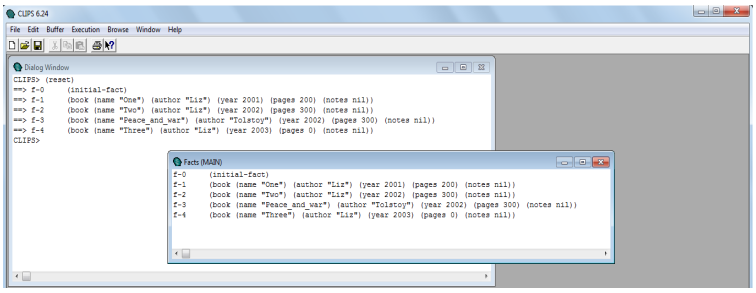

Рис.8 - Окно «Facts *(MAIN)»*

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

В среде CLIPS создать несколько упорядоченных фактов.

Определить неупорядоченные факты (шаблоны), описывающие предложенные объекты (сущности) согласно полученному варианту. Использовать не менее 6 полей с различными описателями.

Создать 5-8 ненаправленных фактов, объединенных в конструкции `deffacts` и демонстрирующих работу каждого описательного поля.

Сохранить факты в файл.

Добавить созданные факты в конструктор deffacts.

Продемонстрировать удаление, изменение фактов с последующей реинициализацией фактов.

ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ

Объекты (сущности) выбираются в соответствии с вариантом задания, который назначается преподавателем.

Все факты должны быть сохранены в файл посредством соответствующих команд *CLIPS*.

ВАРИАНТЫ ЗАДАНИЙ

1. Компьютеры.
2. Мониторы.
3. Принтеры.
4. Музыкальные инструменты.
5. Геометрические фигуры в пространстве.
6. Животные.
7. Автомобили.
8. Книги.
9. Компьютерные игры.
10. Музыкальные композиции.
11. Функции одной переменной.
12. Кошки.
13. Языки программирования.
14. Операционные системы.
15. Студенты.
16. Преподаватели.
17. Предметы, изучаемые в университете.
18. Небесные тела.
19. Оружие.
20. Цветы.
21. Деревья.
22. Птицы.
23. Дома.
24. Города.
25. Страны.
26. Мобильные телефоны.
27. Фирмы.
28. Фрукты.
29. Лодки.

30. Фотоаппараты.
31. Стулья.
32. Стереосистемы (музыкальные центры)
33. Одежда.
34. Водоемы.
35. Часы.
36. Носители информации.
37. Пассажирские поезда.
38. Сетевые карты.
39. Web-браузеры.
40. Кондитерские изделия.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Перечислите примитивные типы данных поддерживаемые CLIPS и их назначение.
2. Чем тип SYMBOL отличается от типа STRING? Приведите пример.
3. Какие команды необходимо использовать для загрузки и сохранения данных?
4. С помощью какой команды можно очистить рабочую память?
5. Как произвести запись из основного экрана CLIPS?
6. Перечислите типы фактов в CLIPS.
7. Какой командой можно добавить факт в рабочую память?
8. Какой командой можно изменить факт в рабочей памяти?
9. Какой командой можно удалить факт из рабочей памяти?
10. Чем команда DUPLICATE отличается MODIFY?
11. Приведите пример задания шаблона факта.
12. Для каких целей необходим слот и мультислот?
13. Какие существуют описательные поля и каково их назначение?
14. Как удалить шаблоны из памяти?
15. Чем действие команды CLEAR отличается от команды RESET?

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 3 занятия (6 академических часов: 5 часа на выполнение и сдачу лабораторной работы и 1 час на подготовку отчета).

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), этапы выполнения работы (со скриншотами), результаты выполнения работы (скриншоты и содержимое файлов), выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Малышева Е.Н. Экспертные системы [Электронный ресурс]: учебное пособие по специальности 080801 «Прикладная информатика (в информационной сфере)»/ Малышева Е.Н.— Электрон. текстовые данные.— Кемерово: Кемеровский государственный институт культуры, 2010.— 86 с.— Режим доступа: http://www.iprbookshop.ru/22126.— ЭБС «IPRbooks»
2. Павлов С.Н. Системы искусственного интеллекта. Часть 1 [Электронный ресурс]: учебное пособие/ Павлов С.Н.— Электрон. текстовые данные.— Томск: Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2011.— 176 с.— Режим доступа: http://www.iprbookshop.ru/13974.— ЭБС «IPRbooks»
3. Павлов С.Н. Системы искусственного интеллекта. Часть 2 [Электронный ресурс]: учебное пособие/ Павлов С.Н.— Электрон. текстовые данные.— Томск: Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2011.— 194 с.— Режим доступа: http://www.iprbookshop.ru/13975.— ЭБС «IPRbooks»
4. Чернышов, В.Н. Системный анализ и моделирование при разработке экспертных систем : учебное пособие / В.Н. Чернышов, А.В. Чернышов ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2012. - 128 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=277638 (22.02.2017).

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

5. Воронов, А.Е. Технология использования экспертных систем / А.Е. Воронов. - М. : Лаборатория книги, 2011. - 109 с. : ил. - ISBN 978-5-504-00525-6 ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=142527 (22.02.2017).
6. Трофимов, В.Б. Интеллектуальные автоматизированные системы управления технологическими объектами : учебно-практическое пособие / В.Б. Трофимов, С.М. Кулаков. - Москва-Вологда : Инфра-Инженерия, 2016. - 232 с. : ил., табл., схем. - Библиогр. в кн.. - ISBN 978-5-9729-0135-7 ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=444175 (22.02.2017).
7. Интеллектуальные и информационные системы в медицине: мониторинг и поддержка принятия решений : сборник статей / . - М. ; Берлин : Директ-Медиа, 2016. - 529 с. : ил., схем., табл. - Библиогр. в кн. - ISBN 978-5-4475-7150-4 ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=434736 (22.02.2017).
8. Джарратано Дж., Райли Г. Экспертные системы. Принципы разработки и программирование, 4-е издание.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2007. – 1152 с.: ил. – Парал. тит. англ.

Электронные ресурсы:

9. https://ru.wikipedia.org/wiki/CLIPS - CLIPS — Википедия
10. http://clipsrules.sourceforge.net/ - A Tool for Building Expert Systems (англ.)
11. http://clipsrules.sourceforge.net/WhatIsCLIPS.html - What is CLIPS? (англ.)