**Программная среда разработки ЭС CLIPS: Назначение и основные возможности. Базовые типы данных и представление фактов.**

**Общая характеристика среды CLIPS**

Среда CLIPS (C Language Integrated Production System) предназначена для построения ЭС и поддерживает три основных способа представления знаний:

* продукционные правила для представления эвристических знаний;
* функции для представления процедурных знаний;
* объектно-ориентированное программирование.

Поддерживаются 6 основных черт ООП: классы, обработчики сообщений, абстракции, инкапсуляция, наследование и полиморфизм. Правила могут сопоставляться с объектами и фактами.Приложения могут разрабатываться с использованием только правил, только объектов или их комбинации. Предусмотрена интеграция с другими средствами: CLIPS может вызываться из процедурных языков, выполнять свои функции и затем возвращать управление вызвавшей программе. С другой стороны, процедурный код может быть определен как внешняя функция и вызван из CLIPS.

Первоначально CLIPS поддерживал возможность для представления только правил и фактов. Начиная с версии 6.0, правила могут сопоставляться с объектами, так же как с фактами. Более того, объекты могут использоваться безправил путем посылки сообщений и в этом случае отпадает необходимость в машине вывода, если используются только объекты.

**Базовые типы данных и представление фактов в системеCLIPS**

В CLIPS поддерживаются восемь базовых типов данных: *целые* (integer) и *вещественные* (float) числа, *символьные* (symbol) и *строковые* (string) данные, *внешний адрес* (external-address), *адрес факта* (fact-address), *имя экземпляра* (instance-name) и *адрес экземпляра* (instance-address).

*Целые числа* состоят из знака (необязательного для положительных чисел) и последовательности десятичных цифр. Например: 27; +125; -38.

*Вещественные* числа содержат мантиссу (в которой дробная часть отделяется точкой) и необязательный порядок, состоящий из символа "e" и целого числа. Например: 12.0; -1.59; 237e3; -32.3e-7.

*Символьное* значение задается продолжающейся до ограничителя последовательностью отображаемых ASCII-символов. Ограничителями являются: любой неотображаемый ASCII-символ (пробел, табуляция, возврат каретки, перевод строки), кавычка, открывающая и закрывающая скобки, амперсанд (&), вертикальная черта (⏐), знак «меньше» (<) и тильда (~). Точка с запятой (;) также является ограничителем и используется для указания на начало комментария. Ограничители не включаются в символьное значение, за исключением символа "<", стоящего в начале значения. Символьное значение не может начинаться с символа "?" или пары символов "$?", но может содержать их внутри себя. CLIPS различает символы нижнего и верхнего регистров, поэтому abc и ABC воспринимаются как два разных значения. Примеры символьных значений: bad\_value; 456-93-039; @+=%.

*Строковое* значение – это заключенная в кавычки последовательность (в том числе пустая) отображаемых символов. Чтобы включить кавычки в строковое значение, перед ними необходимо поставить символ "\". Чтобы вставить в строковое значение символ "\", перед ним необходимо поместить еще один символ "\", т.е. записать два символа "\" подряд. Примеры строковых значений: "abc"; "a & b"; "a\"quote"; "fgs\\85".

*Внешний адрес* – адрес внешней структуры данных, возвращаемый интегрированной в CLIPS функцией (написанной на языках C или Ada). Этот тип данных создается только как результат вызова функции, его невозможно специфицировать вводом значения. Отображаемое представление внешнего адреса:

<Pointer-XXXXXX>,

где ХХХХХХ – внешний адрес.

*Адрес факта* используется для ссылки на факты. Факт представляет собой список атомарных значений, на которые можно ссылаться либо позиционно (в упорядоченных фактах), либо по имени (в неупорядоченных фактах). Отображаемый формат адреса факта:

<Fact-XXX>,

где ХХХ – индекс факта.

*Имя экземпляра* используется для ссылки на экземпляры классов. *Экземпляр* представляет собой *объект*, являющийся представителем некоторого *класса*. Объектами в CLIPS по определению являются целые и вещественные числа, символьные и строковые значения, многоместные значения, внешние адреса, адреса фактов или экземпляры определенных пользователем классов (создаваемых с помощью конструкции defclass). *Имя экземпляра* представляется заключенным в квадратные скобки символьным типом. Например: [pump-1]; [foo]; [+++]; [123-890]. Скобки не являются частью имени, а только указывают тип значения.

*Адрес экземпляра* может быть получен путем связывания значения, возвращаемого функцией instance-address, или связывания переменной с экземпляром, сопоставляемым с объектным образцом в левой части правила Невозможно специфицировать адрес экземпляра вводом значения. Отображаемое представление адресаэкземпляра в CLIPS:

<Instance-XXX>,

где ХХХ – имя экземпляра.

На экземпляры определяемых пользователем классов можно ссылаться либо по имени либо по адресу. Адреса экземпляров должны использоваться, когда критично время решения.

В CLIPS место, занимаемое одним значением базового типа данных, называется *полем* (field). Все значения базовых типов являются *одноместными* (single-field value). Последовательность из нуля или более одноместных значений рассматривается как *многоместное значение* (multifield value). Они отображаются в скобках, причем одноместные значения разделяются пробелами. Примеры многоместных значений: (a 123); (); (x 3.0 "red" 567).

Одной из основных форм представления информации в CLIPS-системах являются *факты*. Они используются правилами для вывода новых фактов из имеющихся. Все текущие факты в CLIPS помещаются в список фактов (fact-list). По формату представления в CLIPS выделяют два типа фактов: *упорядоченные* и *неупорядоченные*.

*Упорядоченный факт* состоит из заключенной в скобки последовательности одного или более разделенных пробелами полей, причем первое поле должно быть символьного типа, а остальные могут быть любыми базовыми типами данных. Первое поле специфицирует отношение, которое применяется к остальным полям факта. Примеры упорядоченных фактов: (высота 100); (студент Сидоров); (отец Иван Петр); (однокурсники Иванов Петров Сидоров).

Для работы с фактами используются следующие команды: assert – добавляет факт в факт-список; retract – удаляет факт из списка; modify – модифицирует список; duplicate – дублирует факт. Например, команда

(assert (length 150) (width 15) (weight “big”))

добавляет в список фактов три факта, каждый из которых состоит из двух полей.

Эти команды могут исполняться как в режиме командной строки, так и включаться в CLIPS-программы. Команды retract, modify и duplicate требуют, чтобы факты были идентифицированы с помощью индекса факта (fact-index) либо адреса факта (fact-address).

*Индекс факта* представляет собой уникальный целочисленный индекс, приписываемый факту всякий раз, когда факт добавляется (или модифицируется). Индексация фактов начинается с нуля и инкрементируется при каждом новом или измененном факте.

*Идентификатор факта* (fact-identifier) представляет собой краткую нотацию для отображения факта. Он состоит из символа “f”, за которым через тире следует индекс факта. Например, f-10 ссылается на факт с индексом 10.

Для задания исходного множества фактов используется конструкция deffacts, имеющая следующий синтаксис:

(deffacts <имя\_группы\_фактов> ["<комментарий>"] <факт>\*),

где <имя\_группы\_фактов> – идентификатор символьного типа;

<комментарий> – необязательное поле комментария;

<факт>\* – произвольная последовательность фактов, записанных через разделитель.

Пример использования конструкции deffacts:

(deffacts stud "Студент"

(student name John)

(student spec “COMPUTER”)).

Факты, определенные конструкцией deffacts, добавляются в список фактов всякий раз при выполнении команды reset.

*Неупорядоченные факты* представляют собой список взаимосвязанных именованных полей, называемых *слотами*. Наличие имен полей позволяет выполнять доступ к полям по именам, в отличие от упорядоченных фактов, где поля специфицируются своим местоположением в факте. Существует два типа слотов: одиночные и мультислоты. Одиночный слот (или просто слот) содержит единственное поле, тогда как мультислот может содержать любое число полей.

Для спецификации состава неупорядоченных фактов (т.е. содержащихся в них слотов) используются *шаблоны*, которые задаются конструкцией deftemplate. Синтаксис конструкции deftemplate определен ниже:

(deftemplate <имя шаблона> [“<комментарий>”]

<определение слота-1>

. . .

<определение слота-N>)

Пример шаблона, содержащего три одиночных слота, представлен ниже:

(deftemplate object “Шаблон объекта”

(slot name)

(slot location)

(slot weight))

Пример конкретного неупорядоченного факта на основе данного шаблона представлен ниже:

(object (name car) (location 100) (weight 600)).

**Представление и обработка продукций в CLIPS.**

**Представление правил в базе знаний. Типы условных элементов.**

Правила являются основным способом представления знаний в CLIPS. Для задания правил используется конструкция defrule со следующим синтаксисом:

(defrule <имя\_правила> ["<комментарий>"]

[<объявление>]

<условный элемент>\*;Левая часть правила (антецедент)

=>

<действие>\*) ;Правая часть правила (консеквент)

где <имя\_правила> - идентификатор символьного типа, уникальный для данной группы правил; <комментарий> - необязательное поле комментария; <объявление> - необязательный элемент, позволяющий задавать дополнительные свойства правила (например, значимость) с помощью оператора declare; <условный элемент>\* - произвольная последовательность условных элементов; <действие>\* - произвольная последовательность действий.

Пример задания правила:

(defrule R1

(days 2)

(works 100)

=>

(printout t crlf "Свободного времени нет" crlf)

(assert (freetime "no"))).

В левой части правило содержит два условных элемента, сопоставляемых с упорядоченными фактами, а в правой – две команды. В команде вывода сообщений printout параметр t задает стандартный режим вывода, а crlf – символ возврата и перевода курсора на новую строку. Команда assert добавляет в факт-список новый упорядоченный факт.

Антецедент правила состоит из последовательности условных элементов (УЭ). Если все УЭ правила удовлетворяются при текущем состоянии базы данных, то правило помещается в список готовых к выполнению правил –агенду. В CLIPS используется **шесть** типов условных элементов:

* УЭ-образцы (Pattern Conditional Elements);
* УЭ-проверки (Test Conditional Elements);
* УЭ “ИЛИ” (Or Conditional Elements);
* УЭ “И” (And Conditional Elements);
* УЭ “НЕ” (Not Conditional Elements);
* УЭ “Существует” (Exists Conditional Elements);
* УЭ “Для всех” (Forall Conditional Elements);
* логические УЭ (Logical Conditional Elements).

*УЭ-образец*состоит из совокупности ограничений на поля, масок (wildcards) полей и переменных, используемых при сопоставлении УЭ с образцом – фактом или экземпляром объекта. УЭ-образец удовлетворяется каждой сущностью, которая удовлетворяет его ограничениям. *Ограничения на поля* используются для проверки одного поля или слота факта либо экземпляра объекта. Ограничение на поле может состоять из единственного литерала или из нескольких связанных ограничений. В УЭ-образцах используются следующие конструкции:

* литеральные ограничения (Literal Constraints);
* одно и многоместные маски (Single- and Multifield Wildcards);
* одно и многоместные переменные (Single- and Multifield Variables);
* ограничения со связками (Connective Constraints);
* предикатные ограничения (Predicate Constraints);
* ограничения возвращаемым значением (Return Value Constraints).

*Литеральное ограничение* не содержит переменных и масок, а задает точное значение (константу целого, вещественного, символьного или строкового типа, либо имя экземпляра), которое должно сопоставляться с полем образца. Все литеральные ограничения должны совпадать с соответствующими полями сопоставляемой сущности. Упорядоченный УЭ-образец содержит только литералы и имеет следующий синтаксис: (<constant-1> ... <constant-n>). Например, (data 1 one “two”). Пример УЭ-образца для неупорядоченных фактов: (person (name Bob) (age 20)).

*Одно- и многоместные маски*позволяют игнорировать некоторые поля в процессе сопоставления. *Одноместная маска* обозначается символом “?” и сопоставляется с любым значением, занимающим точно одно поле в соответствующем месте сопоставляемой сущности. *Многоместная маска*, обозначается парой символов “$?” и сопоставляется с любыми значениями, занимающими произвольное число полей в сопоставляемой сущности. Маски могут использоваться в одном образце в любых комбинациях. Не допускается лишь использование многоместной маски в одноместном слоте (содержащем единственное поле) неупорядоченных фактов или объектов. Например,УЭ (data ? blue red $?) будет сопоставляться со следующими упорядоченными фактами:

(data 1 blue red),

(data 5 blue red 6.9 "avto"),

но не будет сопоставлен со следующими фактами:

(data 1.0 blue "red"),

(data 1 blue).

*Одно- и многоместные переменные*используются для запоминания значений полей, с целью их дальнейшего использования в других условных элементах антецедента или в операторах консеквента правила. Одноместные переменные начинаются с символа “?”, за которым следует символьное значение, начинающееся с буквы. Например: ?x, ?var, ?age. Многоместные переменные начинаются с префикса “$?”, за которым также следует символьное значение, начинающееся с буквы. Например: $?y, $?zum. В именах переменных не допускается использование кавычек. При первом появлении переменная работает так же, как в маске, т.е. связывается с любым значением в данном поле(ях). Последующие появления переменной требуют, чтобы поле сопоставлялось со связанным значением переменной. Имена переменных являются локальными в пределах каждого правила.

Пусть имеется три факта:

(data 2 blue green),

(data 1 blue),

(data 1 blue red)

и правило:

(defrule find-data-1

(data ?x ?y ?z)

=>

(printout t ?x " : " ?y " : " ?z crlf))

УЭ данного правила будет сопоставляться с первым и третьим фактом, поэтому в результате срабатывания правила будет выведено:

1 : blue : red

2 : blue : green

*Ограничения со связками* используются для связывания индивидуальных ограничений и переменных друг с другом с помощью связок & (“и”), | (“или”) и ~ (“не”), используемых в традиционном смысле. Старшинство операций обычное, за исключением случая, когда первым ограничением является переменная, за которой следует связка &. В этом случае первая переменная трактуется как отдельное ограничение, которое также должно удовлетворяться. Например, ограничение ?x&red⏐blue трактуется как ?x&(red⏐blue), а не как (?x&red)⏐blue.

Пример правила с УЭ, содержащим ограничения со связками:

(defrule r1

(data (value ?x&~red&~green))

=>

(printout t "slot value = " ?x crlf)).

Например, для факта (data (value blue)) это правило выведет сообщение:

slot value = blue.

*Предикатное ограничение* позволяет ограничить значение поля, основываясь на истинности булевого выражения. Для этого используется предикатная функция, которая вызывается в процессе сопоставления с образцом и возвращает в случае неудачи символьное значение FALSE. Если возвращается значение FALSE, то ограничение не удовлетворяется, в противном случае оно удовлетворяется. Предикатное ограничение задается с помощью символа “:”, за которым следует вызов предикатной функции. Данное ограничение может использоваться в комбинации с ограничением со связками, а также связанной переменной. В последнем случае переменная сначала связывается некоторым значением, а затем к ней применяется предикатное ограничение. В таком варианте предикатные ограничения часто применяются для проверки типов данных. При этом в качестве предикатных функций используются встроенные функции CLIPS, в частности:

(numberp <выражение>) – функция возвращает значение TRUE, если <выражение> имеет числовой тип (integer или float), в противном случае возвращается символ FALSE;

(floatp <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип float, иначе возвращается символ FALSE;

(integerp <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип integer, иначе – символ FALSE;

(symbolp <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип symbol, иначе – символ FALSE;

(stringp <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип string, иначе – символ FALSE.

Пусть заданы факты: ((data 1) (data 2) (data red)). Тогда для определения значений числового типа можно использовать следующий УЭ

(data ?x&:(numberp ?x)),

который сопоставится с первыми двумя фактами. Тот же результат может быть получен использованием УЭ (data ?x&~:(symbolp ?x)).

*Ограничение возвращаемым значением* использует в качестве ограничения значение, возвращаемое внешней функцией. Эта функция вызывается непосредственно из УЭ-образца с использованием следующего синтаксиса:

=<вызов-функции>

Возвращаемое функцией значение одного из базовых типов подставляется непосредственно в УЭ-образец на позицию, из которой была вызвана функция, и используется далее как литеральное ограничение. Например, следующее правило, содержащее УЭ-образец с ограничением возвращаемым значением**:**

(defrule twice

(data (x ?x) (y =(\* 2 ?x)))

=>...)

будет сопоставляться со всеми неупорядоченными фактами, у которых значение в слоте y равно удвоенному значению слота x.

*Условный элемент-проверка* имеет следующий синтаксис:

(test <function-call>).

УЭ-проверка удовлетворяется, если функция, вызываемая из него, возвращает значение, отличное от FALSE. Как и в предикатном ограничении, можно сравнивать уже связанную некоторым значением переменную, используя любые функции (алгебраическое и логическое сравнение, вызов внешних функций). В УЭ-проверку могут быть встроены внешние функции любого вида. В следующем правиле проверяется, что модуль разности двух чисел не меньше трех:

(defrule example-1

(data ?x)

(value ?y)

(test (>= (abs (- ?y ?x)) 3))

=>...)

*Условный элемент “ИЛИ”* задается следующей конструкцией:

(or <УЭ-1> ... < УЭ-N>)

и удовлетворяется, если удовлетворяется хотя бы один УЭ внутри этой конструкции. Наличие такого УЭ позволяет сократить число правил, т.к. тоже самое можно было бы записать множеством правил с одинаковой правой частью. При этом правило будет активизироваться несколько раз,по числу удовлетворяемых комбинаций. Например, правило

(defrule r1

(man stud)

(or (spec computeer) (age 20))

=>...)

эквивалентно двум следующим:

(defrule r2

(man stud)

(spec computeer)

=>...)

(defrule r3

(man stud)

(age 20)

=>...)

*Условный элемент “И”* задается следующей конструкцией:

(and <УЭ-1> ... < УЭ-N>)

и удовлетворяется, если удовлетворяются все УЭ внутри этой конструкции. В CLIPS все УЭ в антецедентах правил неявно объединены по “И”, однако использование УЭ “И” для явного задания конъюнктивной связи позволяет комбинировать УЭ “И” и УЭ “ИЛИ” в любых сочетаниях. Пример такой комбинации приведен в следующем правиле:

(defrule r1

(sys-mode search)

(or (and (distance high) (resol little))

(and (distance low) (resol big)))

=>...)

*Условный элемент “НЕ”* задается следующей конструкцией:

(not <УЭ>)

и удовлетворяется, если содержащийся внутри него УЭ не удовлетворяется. Предварительно связанные переменные могут использоваться внутри УЭ “НЕ” как свободные. Однако, переменные, которые связываются внутри УЭ-“НЕ”, могут использоваться только в этом образце. Следующее правило ищет факты, у которых второе поле – red, а третье и четвертое поля не совпадают:

(defrule not-double

(not (data red ?x ?x))

=>...)

*Условный элемент “Существует”* имеет следующий синтаксис:

(exists <УЭ-1> ... <УЭ-N>)

и используется для определения, удовлетворяется ли группа УЭ, специфицированных внутри условного элемента “Существует”, хотя бы одним набором образцов-сущностей в базе данных. Например, правило:

(defrule example

(exists (a ?x) (b ?x))

=>...)

будет активизировано, если в базе данных имеется хотя бы одна пара фактов, содержащих в первых полях значения a и b, а вторые поля которых совпадают.

*Условный элемент “Для всех”*имеет следующий синтаксис:

(forall <УЭ-1> ... <УЭ-N>)

и используется для определения, удовлетворяется ли группа УЭ, специфицированных внутри условного элемента “Для всех”, для каждого появления УЭ-1. Например, следующее правило активизируется, если каждый студент научился чтению, письму и арифметике:

(defrule all-students-passed

(forall (student ?name)

(reading ?name)

(writing ?name)

(arithmetic ?name))

=>

(printout t "All students passed." crlf))

*Логические условные элементы* обеспечивают возможность *поддержания истинности* различных сущностей (фактов и экземпляров), создаваемых правилами, использующими логические УЭ. Сущность-образец, создаваемая оператором правой части правила, может быть сделана логически зависимой от сущностей-образцов, сопоставляемых с *логическим* УЭ в антецеденте правил. Сущности-образцы, сопоставляемые с логическими УЭ в антецеденте правил обеспечивают *логическую поддержку* фактам и экземплярам, создаваемым в консеквенте правила. Сущность-образец может логически поддерживаться несколькими группами сущностей-образцов из одного или различных правил. Если любая поддерживающая сущность удаляется из группы поддерживающих сущностей и не существует никаких других поддерживающих групп, то поддерживаемая сущность удаляется из рабочей памяти.

Сущность-образец имеет безусловную поддержку, если она создается без логической поддержки, т.е. с помощью конструкций deffacts, definstaces, с помощью высокоуровневых команд или правил без логической поддержки образцов. Безусловная поддержка сущности удаляет всю логическую поддержку (без удаления самой сущности), при этом дальнейшая логическая поддержка безусловно поддерживаемой сущности игнорируется.

Удаление правила, генерировавшего для сущности логическую поддержку, удаляет логическую поддержку, генерируемую этим правилом, но не влечет удаления сущности, даже если для нее не осталось логической поддержки. Логический УЭ имеет следующий синтаксис:

(logical <УЭ>+)

Логический УЭ группирует образцы точно так же, как УЭ “И” и может использоваться в сочетании с УЭ “И”, УЭ “ИЛИ” и УЭ “НЕ”. Однако логические УЭ можно применять только в первых образцах правила. Например, следующее правило допустимо:

(defrule ok

(logical (a))

(logical (b))

(c)

=>

(assert (d)))

Вместе с тем, следующее правило является недопустимым:

(defrule not-ok-1

(logical (a))

(b)

(logical (c))

=>

(assert (d)))

**Логический вывод в системеCLIPS.**

**Стратегии разрешения конфликтов.**

Базовый цикл работы МЛВ в системеCLIPS:

1. Работа МЛВ останавливается, если достигнут предел активации правил или нет текущего фокуса. В противном случае, для выполнения выбирается верхнее правило агенды того модуля, которому принадлежит текущий фокус. Если в этой агенде нет правил, текущий фокус извлекается из стека фокусов и управление переходит к следующему модулю. Если стек фокусов пуст, то выполнение останавливается, в противном случае вновь выполняется шаг 1.

2. Выполняются операторы, содержащиеся в консеквенте выбранного правила. Использование в консеквенте правила функции return не может переместить текущий фокус из стека фокусов. Счетчик числа правил инкрементируется для контроля предела активации правил.

3. Результатом шага 2 может быть активация или деактивация правил. Активированные правила (у которых удовлетворяется условия в антецеденте) помещаются в агенду того модуля, в котором они определены. Размещение в агенде определяется значимостью (salience) правила и текущей стратегией разрешения конфликтов. Деактивированные правила удаляются из агенды.

4. При использовании динамических значимостей, значения значимостей всех правил, содержащихся в агенде переоцениваются. Цикл повторяется с шага 1.

*Агенда* представляет собой список правил, условия которых удовлетворяются, но которые еще не были выполнены. Каждый модуль имеет собственную агенду. При извлечении правил агенда работает аналогично стеку – первым выполняется правило, находящееся на вершине агенды. Когда правило становится активным (условия в его левой части удовлетворяются), оно помещается в агенду в соответствии со следующими правилами:

1. Вновь активируемые правила помещаются над всеми правилами с более низкой *значимостью* (salience) и ниже всех правил с более высокой значимостью.

2. Для определения места среди правил равной значимости используется текущая стратегия разрешения конфликта.

3. Если в результате добавления или удаления факта одновременно активизируются несколько правил и шаги 1 и 2 не позволяют выполнить упорядочение, то эти правила упорядочиваются между собой произвольно.

*Значимость* позволяет пользователю назначать правилу приоритет, который учитывается при его помещении в агенду. Первым выбирается и срабатывает правило с максимальной значимостью. Значимость может принимать целое значение в диапазоне от -10000 до +10000. По умолчанию ее значение равно 0. Для явного назначения правилу значимости используется оператор:

(declare <rule- salience>),

который добавляется в левую часть правила и размещается перед первым условным элементом, например:

(defrule r1

(declare (salience 500))

(fire test-1)

=>

(printout t "Rule r1 firing." crlf))

Значение значимости может назначаться в один из трех моментов: при определении правила, при активизации правила и в каждом цикле выполнения (последние два случая соответствуют *динамической значимости*). По умолчанию значение значимости назначается только при определении правила. Для изменения такого поведения может использоваться команда set-salience-evaluation.

*Стратегии разрешения конфликтов.* В CLIPS поддерживается семь стратегий разрешения конфликтов: *“вглубь”* (depth), *“вширь”* (breadth), *“простоты”* (simplicity), *“сложности”* (complexity), *LEX*, *MEA* и *случайного выбора* (random).

Текущая стратегия может быть установлена, используя пункт меню “Execution/Options” оконного интерфейса, или командой set-strategy. При этом агенда переупорядочивается на основе новой стратегии. Синтаксис команды:

(set-strategy <strategy>),

где <strategy>::= depth⏐breadth⏐simplicity⏐complexity⏐ lex⏐mea⏐random.

По умолчанию используется стратегия depth.

*Стратегия “вглубь”.* Вновь активируемые правила помещаются в агенду над всеми правилами такой же значимости. Например, пусть факт f-1 активирует правила rule-1 и rule-2, а факт f-2 активирует правила rule-3 и rule-4. Тогда если f-1 устанавливается раньше, чем f-2, то rule-3 и rule-4 окажутся в агенде выше правил rule-1 и rule-2. Однако положение правила rule-1 относительно правила rule-2 и правила rule-3 относительно правила rule-4 будет произвольным.

*Стратегия “вширь”.* Вновь активируемые правила помещаются ниже всех правил с такой же значимостью. Например, пусть факт f-1 активирует правила rule-1 и rule-2, а факт f-2 активирует правила rule-3 и rule-4. Тогда, если f-1 устанавливается раньше, чем f-2, то rule-1 и rule-2 окажутся в агенде выше правил rule-3 и rule-4. Однако, положение правила rule-1 относительно правила rule-2 и правила rule-3 относительно правила rule-4 будет произвольным.

*Стратегия “простоты”.* Среди правил одинаковой значимости, вновь активируемые правила помещаются над всеми правилами с равной или большей специфичностью (specificity). *Специфичность* правила определяется числом сравнений, которые должны быть выполнены в левой части правила.

Специфичность инкрементируется при:

* каждом сравнении с константой или предварительно связанной переменной;
* каждом вызове функции, сделанном из левой части правила в условном элементе с предикатным ограничением (:), ограничением возвращаемым значением (=) или УЭ-проверкой (test).

Булевы функции “и”, “или”, “не” не увеличивают специфичность правила, но их аргументы увеличивают. Вызовы функций, выполняемые из функций, не увеличивают специфичность. Например, следующее правило:

(defrule example

(item ?x ?y ?x)

(test (and(numberp ?x) (> ?x (+ 10 ?y)) (< ?x 100)))

=>…)

имеет специфичность 5 (считаются операторы (item ?x ?y ?x), ?x, numberp, >, <).

*Стратегия “сложности”.* Среди правил одинаковой значимости, вновь активируемые правила помещаются над всеми правилами с равной или меньшей специфичностью.

*Стратегия LEX.* Для определения места правила в агенде среди правил одинаковой значимости в первую очередь используется *новизна* образцов, активирующих данное правило. Каждый факт и экземпляр в БД помечаются “временным тегом”, указывающим его новизну по отношению ко всем другим фактам и экземплярам в системе. Для определения местоположения правила в агенде образцы (факты или экземпляры), связанные с активацией каждого правила сортируются по убыванию новизны. Правило с более поздним образцом помещается выше правил с более ранними образцами. Чтобы определить относительный порядок размещения двух правил, отсортированные временные теги образцов, активирующих правила, сравниваются попарно, начиная с самых больших значений. Сравнение продолжается до тех пор, пока не будет обнаружено, что временной тег одной активации больше соответствующего временного тега другой активации. Правило с большим значением временного тега помещается в агенду выше другого правила. Данная стратегия поясняется примером на рис. 16.1. В данном случае раньше сработает правило Rule1, т.к. временной тег образца, связанного с его третьим условным элементом (8) больше, чем временной тег соответствующего образца у правила Rule2 (6).

Если одно правило имеет больше образцов, чем другое, а все сравниваемые временные теги идентичны, то правило с большим числом временных тегов помещается выше. Если два правила имеют равную новизну, правило с более высокой специфичностью помещается выше правила с более низкой специфичностью.



*Стратегия MEA.* Для определения места правила в агенде среди правил равной значимости в первую очередь используется временной тег образца, связанного с первым условным элементом в правиле. Правило, у которого этот временной тег больше временных тегов, связанных с первыми условными элементами других правил, помещается в агенду выше них. Если временные теги первых образцов равны, то для определения места правила используется стратегия LEX. В примере на рис. 16.2 раньше сработает правило Rule1.

*Стратегия случайного выбора*. Каждому правилу сопоставляется случайное число, которое используется для определения его местоположения в агенде среди правил равной значимости. Это случайное число сохраняется при изменении стратегии, а в случае возврата к случайной стратегии разрешения конфликта восстанавливается тот же порядок среди правил, которые находились в агенде, когда стратегия была изменена.



**Пример построения экспертной системы в среде CLIPS**

Ниже в качестве примера приведен фрагмент простой экспертной системы в среде CLIPS, определяющей наличие у студента свободного времени в зависимости от количества несделанных лабораторных работ и оставшегося до зачета времени:

(defrule data-input

(initial-fact)

=>

(printout t crlf "Введите число дней до зачета (целое значение): ")

(bind ?days (read))

(if (numberp ?days)

then (assert (days ?days))

else (printout t "Введите число" crlf))

(printout t crlf "Введите число несделанных лабораторных работ (в %)")

(bind ?works (read))

(assert (works ?works)))

(defrule R1

(days ?days)

(works ?works)

(test (and (= ?days 1) (<> ?works 0)))

=>

(printout t crlf "Свободного времени нет" crlf)

(assert (freetime "no")))

(defrule R2

(days ?days)

(works ?works)

(test (and (= ?days 2) (>= ?works 10)))

=>

(printout t crlf crlf "Свободного времени нет" crlf)

(assert (freetime "no")))

Правило data-input обеспечивает ввод исходных значений числа оставшихся до зачета дней и количества несделанных работ. Оно инициируется исходным фактом (initial-fact) в рабочей памяти и выдает приглашение на ввод значения. Команда bind осуществляет занесение введенных значений в факты рабочей памяти. Кроме того, для переменной выполняется контроль типа введенного значения.