Курс лекции по дисциплине YaPZ7302 Языки представления знаний

Лекция 6. Модули

Цель: Изучение модулей.

План:

1. Понятие «модуль»
2. Создание модулей. Синтаксис конструктора defmodule
3. Предопределенный конструктор модуля MAIN
4. Определения модулей в конструкторах
5. Пример 1. Применение спецификации модуля
6. Пример 2. Использование модулей
7. Использование модулей в командах и функциях
8. Импорт и экспорт конструкций
9. Пример . Пример экспорта конструкций
10. Пример . Пример импорта конструкций

CLIPS предоставляет возможность разбиения базы данных и решения задачи на отдельные независимые модули. Для создания таких модулей служит конструктор defmodule. С помощью модулей можно группировать вместе отдельные элементы базы знаний и управлять процессом доступа к этим элементам во время решения некоторой задачи. Подобный процесс управления доступа к данным напоминает механизм пространства имен, используемый в C++, и глобальные и локальные области видимости в языках C и Ada. Однако, в отличие от механизмов в перечисленных выше языках, области видимости в CLIPS строго иерархичны и однонаправлены: если модуль A может видеть данные модуля в, это не означает, что модуль в может видеть данные модуля A. С помощью управления ограничением доступа к данным, содержащимся в различных модулях, при решении сложных задач модули могут реализовывать концепцию *доски объявлений (blackboard strategy* — стратегия решения задач с использованием разнородных источников знаний, взаимодействующих через общее

информационное поле). В этом случае отдельный модуль позволяет видеть строго определенный набор фактов и объектов правилам из других модулей. Кроме того, модули используются для управления потоком вычисления правил.

Создание модулей

Как уже упоминалось выше, для создания модулей служит конструктор defmodule.

Определение. Синтаксис конструктора defmodule

(defmodule <имя-модуля>

[<комментарии>]

<спецификации-импорта-экспорта>*)

<спецификация-импорта-экспорта> ::=

(export <элемент-слецификации>) |

(import <имя-модуля> <элемент-спецификации>

<элемент-спецификации>::= ?ALL |

?NONE |

<конструктор> ?ALL |

<конструктор> ?NONE |

<конструктор> <имя-конструктора>

<конструкция>:= deftemplate | defclass | defglobal | deffunction | defqeneric

После своего создания модуль не может быть переопределен или удален (за исключением системного модуля MAIN, который пользователь может один раз переопределить). Единственный способ удалить существующий модуль — выполнить команду clear. Во время запуска системы и при вызове команды clear CLIPS автоматически создает предопределенный системный модуль, указанный ниже.

Определение. Предопределенный конструктор модуля MAIN

(defmodule MAIN)

Все предопределенные системные классы принадлежат системному модулю MAIN, однако нет необходимости экспортировать или импортировать

системные классы в другие модули, они всегда находятся в области видимости определенных пользователем модулей. Предопределенный системный модуль MAIN не импортирует и не экспортирует никаких конструкций. Однако, в отличие от других модулей, пользователь может один раз переопределить модуль MAIN после запуска системы или выполнения команды clear.

Определения модулей в конструкторах

Для определения, в какой модуль будет помещена та или иная конструкция языка, созданная соответствующим конструктором, в конструкторе необходимо указать имя модуля. Конструкторы deffacts, deftemplate, defrule, deffunction, defgeneric, defclass и definstances для определения имени модуля позволяют включать его в имя соответствующей конструкции. Конструктор defglobal принимает имя модуля в специально отведенное для этого поле, которое следует сразу за ключевым словом defglobal. Конструктор defmessage-handier принимает имя модуля как часть определения класса, с которым связывается сообщение. Конструктор defmethod принимает имя модуля как часть определения родовой функции, которой принадлежит данный метод. Например, все приведенные ниже конструкторы будут помещены в модуль DETECTION.

Пример 1. Применение спецификации модуля

```
(defrule DETECTION::Find-Fault
(sensor (name ?name) (value bad))
 =>
(assert (fault (name ?name))))
(defglobal DETECTION ?*count* = 0)
(defmessage-handler DETECTION::COMPONENT get-charge ()
(* ?self:flux ?self:flow))
(defmethod DETECTION::+ ((?x STRING) (?y STRING))
(str-cat ?x ?y))
```

Выполните следующую последовательность действий:

Пример 2. Использование модулей

```
(clear)
(defmodule A)
(defmodule B)
(defrule  foo =>)
(defrule A::bar =>)
(list-defrules)
(set-current-module B)
(list-defrules)
```

Результат выполнения этих команд приведен на рис. 35.

Обратите внимание, что после определения нового модуля он становится текущим (имя текущего модуля можно получить с помощью функции get-current-module). Таким образом, правило foo было добавлено в текущий модуль B, т. к. при его создании модуль не был указан явно, а правило bar добавлено в модуль A, что явно указано в конструкторе. Сообщения, возникшие после определения правил, сообщают об определении в новых модулях фактов initial-fact, необходимых для безусловных правил. После этого, переключая текущий активный модуль с помощью команды set-current-module и используя команду list-defrules, можно убедиться, что правила находятся именно в тех модулях, в которых они должны находиться. Windows-версия CLIPS предоставляет еще один способ просмотра списка определенных пользователем модулей и изменения текущего модуля. Эта возможность реализована с помощью вложенного меню **Module,** содержащегося в меню **Browse.** Текущий модуль в этом меню отмечен флажком (рис. 36).

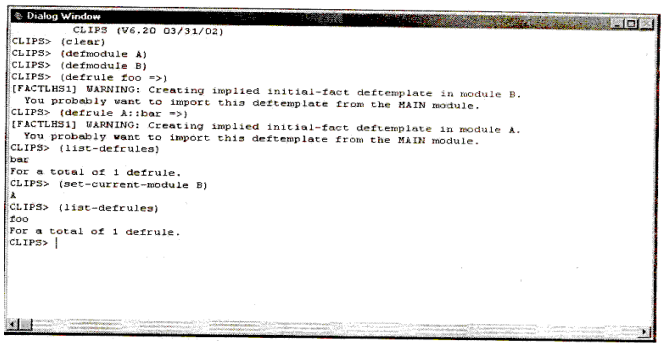

Рис. 35. Использование модулей

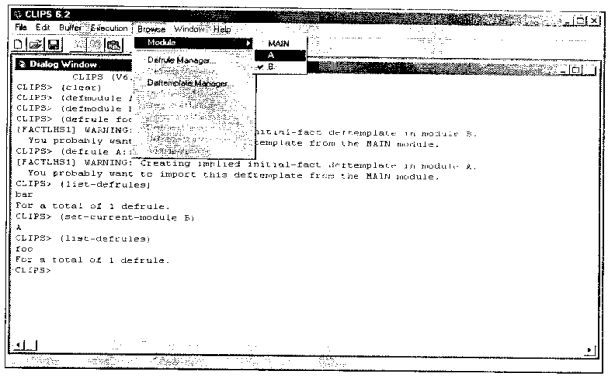

Рис. 36. Выбор активного модуля

Использование модулей в командах и функциях

Многие команды, например undefrule или ppdefrule, используют имя конструкции, которой оперируют. В предыдущих версиях CLIPS имени конструкции было вполне достаточно для однозначной идентификации. Однако после введения модулей стало возможным существование конструкций с одинаковыми именами в двух различных модулях. Модуль конструкции, используемой в команде, может быть задан явно или неявно.

Явное задание модуля выполняется с помощью имени модуля, разделенного с именем конструкции при помощи двойного двоеточия **::** . Имя модуля и символ **::** называются *спецификатором модуля* (module specifier). Например, запись MAIN**::** find-stuff ссылается на конструкцию find-stuff из модуля MAIN.

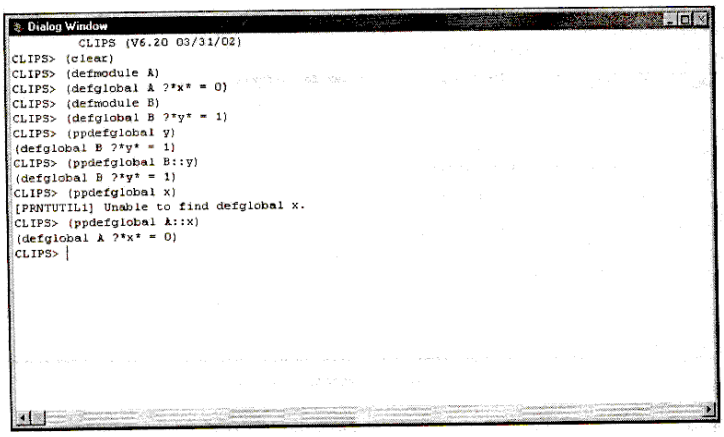

Рис. 37. Пример использования спецификатора модуля

Неявное задание модуля выполняется с помощью установки текущего активного модуля. Текущий модуль меняется при каждом определении нового

модуля или при вызове функции set-current-module. Так как модуль MAIN автоматически добавляется в систему при загрузке системы, а также при каждом вызове функции clear, то MAIN является текущим модулем по умолчанию. Таким образом, имя find-stuff ссылается на конструкцию find-stuff из модуля MAIN. Пример явного и неявного задания модуля в командах приведен на рис. 37.

Импорт и экспорт конструкций

За исключением специально экспортированных и импортированных, конструкции, определенные в одном модуле, не могут использоваться в другом модуле. Конструкция называется видимой или находящейся в пределах области видимости модуля, если она может использоваться в модуле. Например, если пользователь хочет указать в модуле в конструктор deftemplate с именем foo, определенный в модуле A, то модуль A должен экспортировать deftemplate foo, а модуль B должен импортировать deftemplate foo их модуля A. Подобная ситуация приведена на рис. 38.

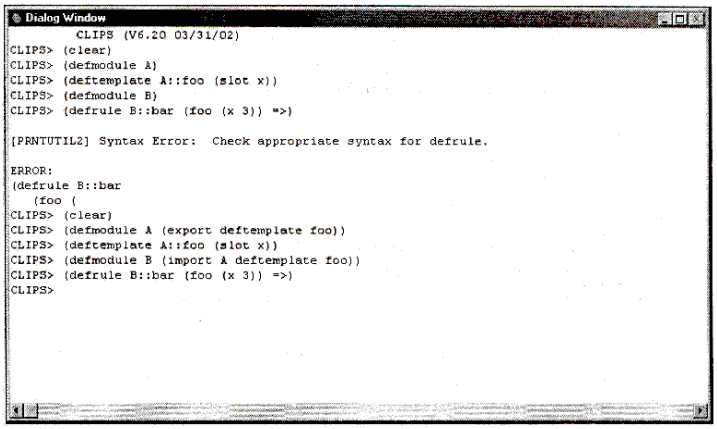

Рис. 38. Импорт/экспорт шаблонов

CLIPS не допускает существования двух конструкторов с одинаковыми именами, видимых в одном модуле.

Спецификация экспорта в определении модуля служит для определения, какие именно конструкции данного модуля могут импортироваться другими модулями. Экспортироваться способны только следующие конструкции: deftemplates, defclasses, defglobals, deffunctions, и defgenerics. Модуль может экспортировать любую видимую конструкцию

данных типов. При этом не обязательно, чтобы эта конструкция была непосредственно определена в данном модуле.

В CLIPS существует три типа спецификации экспорта.

- Во-первых, модуль может экспортировать все видимые в нем конструкции. Это осуществляется с помощью ключевого слова export и следующего за ним ключевого слова ?ALL.
- Во-вторых, модуль может экспортировать все видимые в нем конструкции заданного типа. Для этого используется ключевое слово export, тип конструкции и ключевое слово ?ALL.
- В-третьих, модуль может экспортировать некоторые отдельные конструкции заданного типа. Это осуществляется с помощью ключевого слова export, типа конструкции, списка из одного или более имен видимых конструкций заданного типа, которые необходимо экспортировать.

В приведенном ниже примере модуль A экспортирует все видимые в нем конструкции, модуль B — все конструкции deftemplate, а модуль C — три отдельных конструкции defglobal.

Пример . Пример экспорта конструкций

```
(defmodule A (export ?ALL))
(defmodule B (export deftempiate ?ALL))
(defmodule C (export defglobal foo bar yak))
```

Вместо ключевого слова ?ALL в спецификации экспорта может использоваться ключевое слово ?NONE. В этом случае модуль не будет экспортировать либо вообще никаких конструкций, либо не будет экспортировать никаких конструкций заданного типа.

Конструкции defmethod и defmessage-handler никогда не экспортируются явно. Экспорт конструктора defgeneric автоматически приводит к экспорту всех ассоциированных с ним конструкторов defmethod. Экспорт конструктора defclass — к автоматическому экспортированию всех связанных с классом обработчиков (конструкторов defmessage-handler).

Конструкции deffacts, definstances и defrules вообще не могут быть экспортированы.

Спецификация импорта в определении модуля служит для определения, какие конструкции из других модулей могут использоваться в данном модуле. Импортироваться могут только следующие конструкции: deftemplates, defclasses, defglobals, deffunctions и defgenerics.

В CLIPS существует три типа спецификации импорта.

- Во-первых, модуль может импортировать все конструкции, видимые в некотором заданном модуле. Это осуществляется с помощью ключевого слова import, имени модуля, из которого будет производиться импорт, и ключевого слова ?ALL.
- Во-вторых, модуль может импортировать все конструкции заданного типа, видимые в некотором заданном модуле. Для этого используется ключевое слово import, имя модуля, тип конструкции и ключевое слово ?ALL.
- В-третьих, модуль может импортировать некоторые отдельные конструкции заданного типа. Это осуществляется с помощью ключевого слова import, имени модуля, из которого будет производиться импорт, типа конструкции, списка из одного или более имен видимых конструкций заданного типа, которые необходимо импортировать.

В приведенном ниже примере модуль A импортирует все видимые конструкции модуля D, модуль B — все конструкции deftemplate из модуля D, а модуль C — три отдельных конструкции defglobal, также определенные в модуле D.

Пример. Пример импорта конструкций

```
(defmodule A (import D ?ALL))
(defmodule B (import D deftemplate ?ALL)}
(defmodule C (import D defglobal foo bar yak))
```

Вместо ключевого слова ?ALL в спецификации импорта может использоваться ключевое слово ?NONE. В этом случае модуль не будет

импортировать либо вообще никаких конструкций, либо не будет импортировать никаких конструкций заданного типа.

Конструкции defmethod и defmessage-handler никогда явно не импортируются. Импорт конструкторов defgeneric приводит к импортированию всех ассоциированных с ним конструкторов defmethod. Импорт конструкторов defclass приводит к автоматическому импортированию всех связанных с классом обработчиков (конструкторов defmessage-handler). Конструкции deffacts, definstances и defrules не могут быть импортированы.

Модуль должен быть определен до того, как он будет использован в спецификации импорта. Кроме того, указанные в спецификации импорта конструкции должны экспортироваться соответствующим модулем.

Контрольные вопросы:

1. Предназначение модулей.
2. Импорт и экспорт конструкций
3. Приведите пример импорта конструкций

Литература:

1. А. П. Частиков Т. А. Гаврилова Д. Л.Белов. РАЗРАБОТКА ЭКСПЕРТНЫХ СИСТЕМ. СРЕДА CLIPS. СПб: БХВ-Петербург, 2003

Ключевые слова:

Модуль, defmodule, предопределенный конструктор, экспорт конструкций, импорт конструкций