МОДУЛНА СТРУКТУРА НА БАЗАТА ОТ ЗНАНИЯ В CLIPS

CLIPS поддържа средства за модулно изграждане и използване на бази от знания. За целта е предназначена конструкцията **`defmodule`**. Модулите в CLIPS позволяват конструкциите от дадено множество да бъдат групирани по такъв начин, че да бъде осъществяван явен контрол върху достъпа до тези конструкции от други модули.

Чрез ограничаване на достъпа до образците и класовете модулите могат да функционират като черни дъски (blackboards), които разрешават само определени факти или екземпляри да се виждат от определени модули. Модулите могат да бъдат използвани също за управление на работата на интерпретатора на правилата.

Дефиниране на модули

Синтаксис:

```
(defmodule <module-name> [<comment>]
   <port-specification>*)
<port-specification> ::=
  (export <port-item>) |
  (import <module-name> <port-item>)
```

```
<port-item> ::=
  ?ALL | ?NONE |
   <port-construct> ?ALL |
   <port-construct> ?NONE |
   <port-construct> <construct-name>+

<port-construct> ::=
  deftemplate | defclass | defglobal |
  deffunction | defgeneric
```

След като даден модул е дефиниран, той не може да бъде предефиниран (с изключение на модула MAIN, който може да бъде предефиниран еднократно). Модули се унищожават с помощта на командата **`clear`**. След стартиране на системата и след изпълнение на командата **`clear`** автоматично се конструира модулът **`(defmodule MAIN)`**. Всички вградени класове принадлежат на модула MAIN, но не е необходимо те да бъдат експортирани или импортирани.

Определяне на модула, в който е валидна дадена конструкция

Модулът, в който е валидна дадена конструкция, трябва да се определи при дефинирането на тази конструкция. При дефинирането на факти, образци, правила, потребителски функции, генерични функции, класове и екземпляри името на модула се включва като част от името на конструкцията. Модулът, в който е валидна дадена глобална променлива, се специфицира явно в конструкцията **`defglobal`**. Модулът, в който е валиден даден манипулатор на съобщение, се определя като част от спецификатора на класа. Модулът, в който е валиден даден метод, се определя като част от спецификатора на генеричната функция.

Пример. Следващите конструкции са валидни в модула DETECTION.

```
(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))
```

Цитиране на името на модул

Възможно е в различни модули да са дефинирани конструкции с еднакви имена, затова името на модула трябва да може да се посочва като част от името (“пълното” име) на съответната конструкция. Името на модула може да бъде посочено явно, като се запише пред името на конструкцията и се отдели от него с две двоеточия (::). Ако името на модула не е записано явно, по подразбиране се смята, че се цитира конструкция в текущия модул.

Във всеки момент от работата на CLIPS един модул е текущ. При стартиране на системата текущ става модулът MAIN. Текущият модул се променя при изпълнение на функцията **`set-current-module`** и при дефинирането на нов модул.

Импортиране и експортиране на конструкции

Конструкциите от един модул не могат да бъдат използвани от друг модул, освен ако не са експортирани и импортирани по специфичен начин.

Казва се, че една конструкция е видима за даден модул, когато тази конструкция може да бъде използвана в модула. За целта такава конструкция трябва да бъде експортирана от модула, в който е дефинирана, и да бъде импортирана в другия модул.

Спецификацията **`export`** в дефиницията на един модул се използва за посочване на конструкциите от модула, които могат да бъдат достъпни за други модули, ако бъдат импортирани в тях. Могат да се експортират само образци, класове, глобални променливи, потребителски функции и генерични функции. Един модул може да експортира само конструкции, които са видими за него (не непременно дефинирани в него).

Методи на генерични функции и манипулатори на съобщения не могат да се експортират явно. Експортирането на дадена генерична функция автоматично експортира всички нейни методи. Експортирането на даден клас автоматично експортира всички асоциирани с него манипулатори на съобщения. Факти, екземпляри и правила не могат да бъдат експортирани.

Спецификацията **`import`** в дефиницията на един модул се използва за посочване на конструкциите, експортирани от други модули, които ще бъдат видими за този модул. Могат да бъдат импортирани само образци, класове, глобални променливи, потребителски функции и генерични функции.

Методи на генерични функции и манипулатори на съобщения не могат да се импортират явно. Импортирането на дадена генерична функция автоматично импортира всички нейни методи. Импортирането на даден клас автоматично импортира всички асоциирани с него манипулатори на съобщения. Факти, екземпляри и правила не могат да бъдат импортирани.

Всеки модул трябва да бъде дефиниран преди името му да бъде използвано в някаква импортна спецификация. Освен това, всички конструкции, цитирани в импортните спецификации, трябва да бъдат предварително дефинирани.

Импортиране и експортиране на факти и екземпляри

Фактите и екземплярите са “собственост” на модула, в който са дефинирани съответните им образци и класове (а не на модула, в който са създадени). В този смисъл фактите и екземплярите са видими за тези модули, които импортират техните образци и класове. Това позволява базата от знания да бъде разделена на части по такъв начин, че правилата и останалите конструкции да могат да “виждат” онези факти и екземпляри, които са интересни за тях.

Образецът **`initial-fact`** и класът INITIAL-OBJECT трябва явно да бъдат импортирани от модула MAIN. Правилата, които съдържат в левите си страни образците **`initial-fact`** и **`initial-object`** (а също и правилата, в чиито леви страни автоматично се добавят посочените образци), не могат да бъдат активирани, докато не се импортират конструкциите, съпоставими с тези образци.

Екземплярите, създадени в един модул, трябва да имат уникални имена, но е възможно в даден момент да са видими различни екземпляри с едно и също име. В такива случаи е необходимо имената на екземплярите да бъдат разширени по начин, който позволява специфицирането на съответния модул.

Синтаксис:

```
<instance-name> ::=
  [<symbol>] |
  [::<symbol>] |
  [<module>::<symbol>]
```

[<symbol>]: търсенето на екземпляра с посоченото име се осъществява само в текущия модул;

[::<symbol>]: търсенето на екземпляра с посоченото име се осъществява най-напред в текущия модул и след това – рекурсивно в импортираните модули в реда, определен от дефиницията на текущия модул;

[<module>::<symbol>]: търсенето на екземпляра с посоченото име се осъществява в явно посочения модул.

Модули и изпълнение на правилата

Всеки модул има своя собствена RETE мрежа (индекс на условията) и свое собствено конфликтно множество (agenda). Когато потребителят зададе командата **`run`**, се изпълняват правилата, съответни на активиращите записи от конфликтното множество на модула, който е текущ фокус на интерпретатора.

След изпълнението на командите **`reset`** и **`clear`** текущ фокус става модулът MAIN. Изпълнението на правилата от конфликтното множество на текущия фокус продължава, докато друг модул стане текущ фокус, съответното конфликтно множество стане празно или бъде изпълнена команда **`return`**, включена в дясната страна на правило.

Всеки път, когато в процеса на работата си интерпретаторът излезе извън правилата, съдържащи се в модула, който е текущ фокус, текущият фокус се премахва от стека на фокусите и следващият модул от стека става текущ фокус на интерпретатора. Преди да бъде изпълнено избраното за изпълнение правило, модулът, в който е дефинирано то, става текущ модул.

Текущият фокус може да бъде променен с помощта на командата **`focus`**.

Фокусиране на интерпретатора на правилата върху определени модули

С помощта на командата **`focus`** може в стека на фокусите да бъдат добавени един или няколко модула. Посочените модули се добавят един след друг в стека на фокусите по такъв начин, че първият от тях става нов връх на стека на фокусите.

Синтаксис:

```
(focus <module-name>+)
```

Пример

```
> (clear)
> (defmodule MAIN (export ?ALL))
> (defrule MAIN::focus-example
     =>
     (printout t "Firing rule in module
             MAIN." crlf)
     (focus A B))
> (defmodule A
    (import MAIN deftemplate
        initial-fact))
```

```
> (defrule A::example-rule
    =>
    (printout t "Firing rule in module
            A." crlf))
> (defmodule B
   (import MAIN deftemplate
       initial-fact))
> (defrule B::example-rule
    =>
    (printout t "Firing rule in module
            B." crlf))
```

```
> (reset)
> (run)
Firing rule in module MAIN.
Firing rule in module A.
Firing rule in module B.
>
```

С помощта на функцията **`pop-focus`** текущият фокус се изважда от стека на фокусите. Като оценка на обръщението към **`pop-focus`** се получава името на текущия (досегашния) фокус.

При дефинирането на правило в конструкцията **`defrule`** могат да бъдат описани т. нар. ***свойства на правилото*** (rule properties). Това става с помощта на дескриптора **`declare`**:

```
(declare <rule-property>+)

<rule-property> ::=
  (salience <integer-expression>) |
  (auto-focus <boolean-symbol>)

<boolean-symbol> ::= TRUE | FALSE
```

Ако свойството **`auto-focus`** има стойност TRUE, тогава при активиране на правилото (т.е. при попадането му в съответната agenda) автоматично се изпълнява команда **`focus`** с аргумент името на модула, в който е дефинирано правилото. По подразбиране стойността на свойството **`auto-focus`** е FALSE. При такава стойност не се предприемат никакви специални действия в случай на активиране на правилото.