Введение в CLIPS

Оболочка для разработки экспертных систем – C Language Integrated Production System

Язык программирования CLIPS

- CLIPS – среда (система) разработки ЭС.
- Определяет язык программирования, описанный в Clips Basic Programming Guide.
- Поддерживает программирование на правилах (с использованием встроенного алгоритма вывода – RETE), процедурное, ООП.
- Преемник OPS, начало разработки – 1984.
- Актуальная версия CLIPS 6.3, выпущена 17 марта 2015.
- Не поддерживает обратный вывод (в оригинале).

Основные элементы CLIPS

- База знаний – представлена в виде **фактов**, имеет атрибуты слотового представления.
- Начальные знания представляются в виде некоторого набора исходных фактов.
- Факты образуют список, по умолчанию присутствует нулевой.
- Список правил, шаблоны правил.
- Машина логического вывода, осуществляющая прямой вывод (forward chaining) алгоритомом RETE.

Язык CLIPS

- Имеет общее название COOL.
- Тесная интеграция с C, возможность компиляции и использования функций, написанных на C.
- Синтаксис похож на LISP, заимствования из Ada, Pascal и др.
- Разрабатывался в Космическом центре NASA/Джонсон, там же большая часть упоминаемых систем на его основе была реализована, пока в 1995 не свернули финансирование.
- До пятой версии ориентирован на правила, дальше добавили поддержку процедурного программирования, ООП и проч.
- Имеет простейшую IDE для работы, а также интерпретаторы для некоторых ОС.

Основы

- Case-sensitive, нет поддержки кириллицы.
- Восемь типов полей – вещественное, целое, символ, строка, внешний адрес, адрес факта, имя экземпляра, адрес экземпляра.
- Символы – некоторые последовательности, из печатных символов (0 или более), Lazy-students_are-!?#$^B012389, символы ? и $ в начало символа не ставить.
- Команды – в круглых скобках, иначе это интерпретируется как символ.
- Строки – в двойных кавычках.

Базовые команды

`(load <file-name>)` – загрузка из файла. Функция, по умолчанию отображает процесс, TRUE/FALSE.

`(save <file-name>)` – сохранение в файл текущих конструкций (environment).

`bsave/bload` – аналогично, двоичный файл.

`(clear)` – очистка окружения. Некоторые элементы не удаляются/не изменяются.

`(exit [<integer-expression>])` – выход.

`(reset)` – Очищает список фактов, целей, объекты (instances), реинициализация, смена модуля на MAIN.

`(agenda)` – текущие правила, готовые к активации

Определение фактов

```
(deftemplate car_problem
   (slot name)
   (slot status)
)
(deffacts trouble_shooting
   (car_problem (name ignition_key) (status on))
   (car_problem (name engine) (status wont_start))
   (car_problem (name headlights) (status work))
)
(defrule rule1
   (car_problem (name ignition_key) (status on))
   (car_problem (name engine) (status wont_start))
=>
   (assert (
       car_problem (name starter) (status faulty)))
)
```

Определение фактов

Определяем шаблон факта:

```
(deftemplate <relation-name> [<optional-comment>]
   <slot-definition>*)
<slot-definition> = (slot <slot-name>) |
(multislot <slot-name>)
```

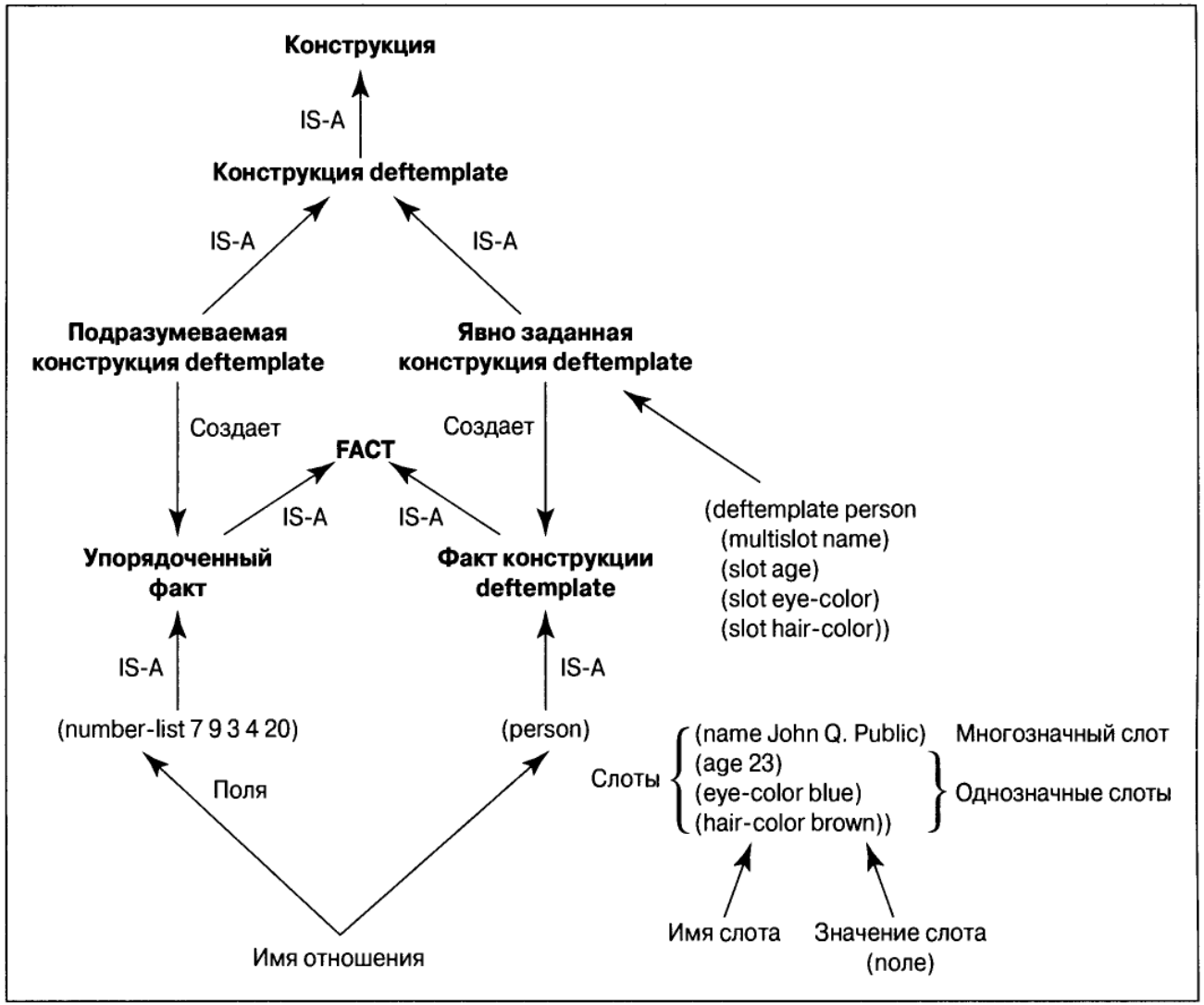
Факты делятся на факты с конструкцией deftemplate (non-ordered facts) и упорядоченные факты (ordered facts). Порядок слотов не важен.

Многозначные слоты могут хранить несколько (0 и более) значений.

(Person (name Albus Dumbledore) (age 112)) – в каком случае такое описание факта допустимо?

Просмотреть текущую коллекцию фактов можно командой (facts), сбросить – (reset)

Виды фактов


Конструкция
IS-A
Конструкция deftemplate
IS-A
IS-A
Подразумеваемая конструкция deftemplate
Явно заданная конструкция deftemplate
Создает
Создает
FACT
IS-A
IS-A
Упорядоченный факт
Факт конструкции deftemplate
(deftemplate person
(multislot name)
(slot age)
(slot eye-color)
(slot hair-color))
IS-A
IS-A
(number-list 7 9 3 4 20)
(person)
Поля
Имя отношения
Слоты
(name John Q. Public)
(age 23)
(eye-color blue)
(hair-color brown))
Многозначный слот
Однозначные слоты
Имя слота
Значение слота (поле)

Список фактов

Добавление фактов происходит командой assert:

`(`**`assert`** `<fact>+)`

Просмотреть текущую коллекцию фактов можно командой (facts), сбросить – `(reset)`

Каждому факту система присваивает индекс, не обязательно последовательный.

По умолчанию ввод дублирующих фактов не производится – выполнение такой команды результата не имеет.

Извлечение фактов – команда `(`**`retract`** `<fact-index>+)`**`.`**

Правила

(**defrule** <rule-name> [<comment>]
    [<declaration>] ; Rule Properties
    <conditional-element>* ; Left-Hand Side (LHS)
    =>
    <action>*) ; Right-Hand Side (RHS)

Right-hand side и **left-had side** – блоки. По умолчанию **and**.

Salience – приоритет правил.

Могут изменять текущую повестку, перемещать фокус по модулям.

Порядок работы

Agenda – список правил, для которых были выполнены все условия (готовых к активации), и которые ещё не были выполнены. У каждого модуля своя, аналог стека.

1. Если предел активаций не достигнут, то выбирается первое правило из Agenda. Если пуст – переход по фокусу.
2. Для выполненного правила активируется RHS, изменяющая как повестку, так и фокус.
3. Agenda перестраивается с учётом приоритета (salience) и стратегии разрешения конфликтов.
4. По завершению – выход.

Стратегии выбора

Если **Agenda** содержит несколько правил, то необходимо выбрать одно из них для выполнения. Существуют различные стратегии выбора, которые, помимо прочих, содержат:

1. BFS – аналог очереди.
2. DFS – аналог стека.
3. RANDOM – случайный выбор.

Манипулировать стратегиями можно из оболочки (выбирать нужную), либо с помощью команды (set-strategy random).

Модули

Для структурирования программы можно использовать механизм модулей – аналог пространства имён.

```
(defmodule DETECTION)
(defmodule ISOLATION)
(defmodule RECOVERY)

(defrule example1 =>) ; in current module
(defrule ISOLATION::example2 =>) ; in module ISOLATION

(get-current-module)
(set-current-module DETECTION)
```

При этом факты принадлежат определённым модулям, равно как и правила. По умолчанию вводится модуль MAIN.

Разграничение

Можно структурировать работу в рамках одного модуля, например следующим образом:

```
(deffacts control-information
      (phase detection)
      (phase-seq detection isolation recovery))

(defrule change-phase
      (declare (salience -10))
      ?phase <- (phase ?current-phase)
      ?list <- (phase-seq ?next-phase $?other-phases)
      =>
      (retract ?phase ?list)
      (assert (phase ?next-phase))
      (assert (phase-seq ?other-phases ?next-phase))
)
```

Импорт и экспорт фактов

«Перемещать» факты между модулями можно следующим образом:

```
(export ?ALL)
(export ?NONE)
(export deftemplate ?ALL)
(export deftemplate ?NONE)
(export deftemplate <deftemplate-name>+)

(import <module-name> ?ALL)
(import <module-name> ?NONE)
(import <module-name> deftemplate ?ALL)
(import <module-name> deftemplate ?NONE)
(import <module-name> deftemplate <deftemplate-name>+)
```

Для возврата в предыдущий модуль можно использовать команду (return)

Переменные и имена фактов

Переменные действуют в рамках одного правила, могут появляться неоднократно, сопоставляются. Для фактов можно использовать имена фактов.

```
(defrule eye-color
        (find (eye-color ?eye))
        (person (name ?name) (eye-color ?eye))
=>
        (printout t crlf ?name " has " ?eye " eye." crlf)
)

(defrule process-moved-information
        ?f1 <- (moved (name ?name) (address ?address))
        ?f2 <- (person (name ?name))
=>
        (retract ?f1) ;a MUST or infinite loop
        (modify ?f2 (address ?address))
)
```

Переменные и имена фактов

Переменные действуют в рамках одного правила, могут появляться неоднократно, сопоставляются. Для фактов можно использовать имена фактов.

```
(defrule eye-color
        (find (eye-color ?eye))
        (person (name ?name) (eye-color ?eye))
=>
        (printout t crlf ?name " has " ?eye " eye." crlf)
)

(defrule process-moved-information
        ?f1 <- (moved (name ?name) (address ?address))
        ?f2 <- (person (name ?name))
=>
        (retract ?f1) ;a MUST or infinite loop
        (modify ?f2 (address ?address))
)
```

Wildcards

Могут представлять некоторые значения (символы) в шаблонах сопоставления. Однозначные и многозначные.

```
(deftemplate person
        (multislot name)
        (slot phone-number))

(defrule print-telephone-number
        (person (name ? ? ?last-name)
        (phone-number ?phone))
        =>
        (printout t ?phone crlf))

(defrule print-telephone-number
         (person (name $? ?last-name)
         (phone-number ?phone))
        =>
        (printout t ?phone crlf))
```

Ограничения на значения полей

Могут использоваться для фильтрации фактов, логические операции not(~), and(&), or(|). Это связанные ограничения.

```
(defrule complex-eye-hair-match
       (person (name ?name1)
               (eyes ?eyes1&blue|green)
               (hair ?hair1&~black))
       (person (name ?name2&~?name1)
               (eyes ?eyes2&~eyes1)
               (hair ?hair2&red|?hair1))
       =>
       (printout t ?name1 " has " ?eyes1 "eyes and " ?hair1 " hair."
               crlf)
       (printout t ?name2 " has " ?eyes2 "eyes and " ?hair2 " hair."
               crlf))
```

Предикативные ограничения

При ограничении на значение переменной в виде предиката можно указывать его после двоеточия, в виде вызова функции.

```
(defrule complex-eye-hair-match
       (person (name ?name1)
               (eyes ?eyes1&blue|green)
               (age ?age1&:(< ?age1 30)))
       (person (name ?name2&~?name1)
               (eyes ?eyes2&~eyes1)
               (hair ?hair2&red|?hair1))
       =>
       (printout t ?name1 " has " ?eyes1 "eyes and " ?hair1 " hair."
               crlf)
       (printout t ?name2 " has " ?eyes2 "eyes and " ?hair2 " hair."
               crlf))
```

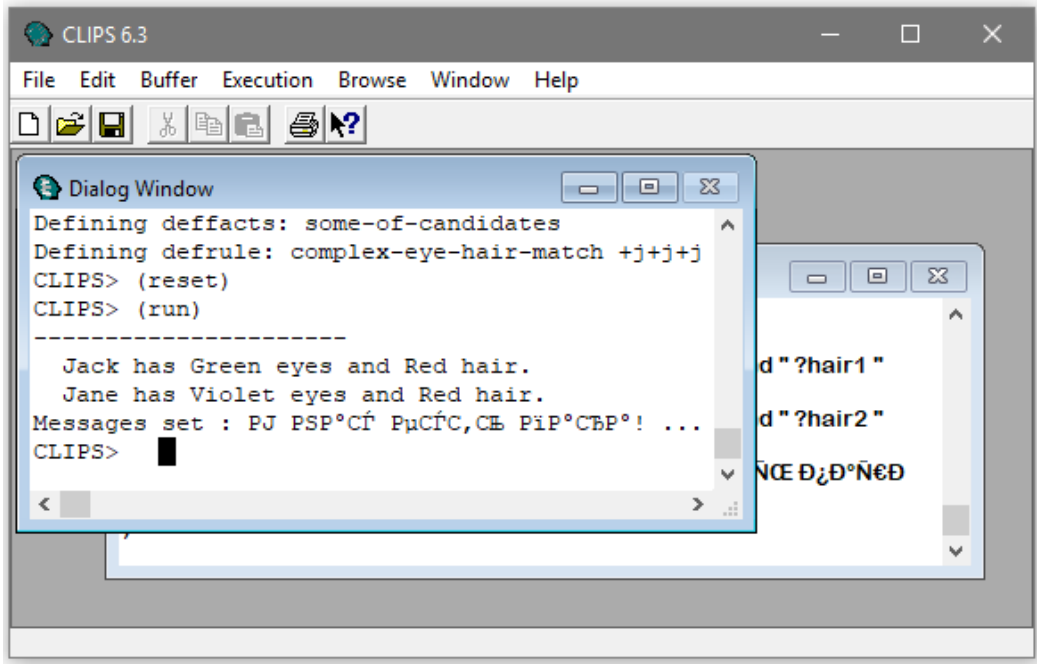
Условный элемент test

Является аналогом предиката, однако при этом является самостоятельным условным элементом (conditional element, CE).

```
(defrule complex-eye-hair-match
        (person (name ?name1)
                (eyes ?eyes1&blue|green)
                (age ?age1&:(< ?age1 30)))
        (person (name ?name2&~?name1)
                (eyes ?eyes2&~eyes1)
                (age ?age2&:(< ?age2 30)))
                (hair ?hair2&red|?hair1))
        (test (< (abs (- ?age1 ?age2)) 10)
        =>
        (printout t ?name1 " has " ?eyes1 "eyes and " ?hair1 " hair."
                crlf)
        (printout t ?name2 " has " ?eyes2 "eyes and " ?hair2 " hair."
                crlf))
```

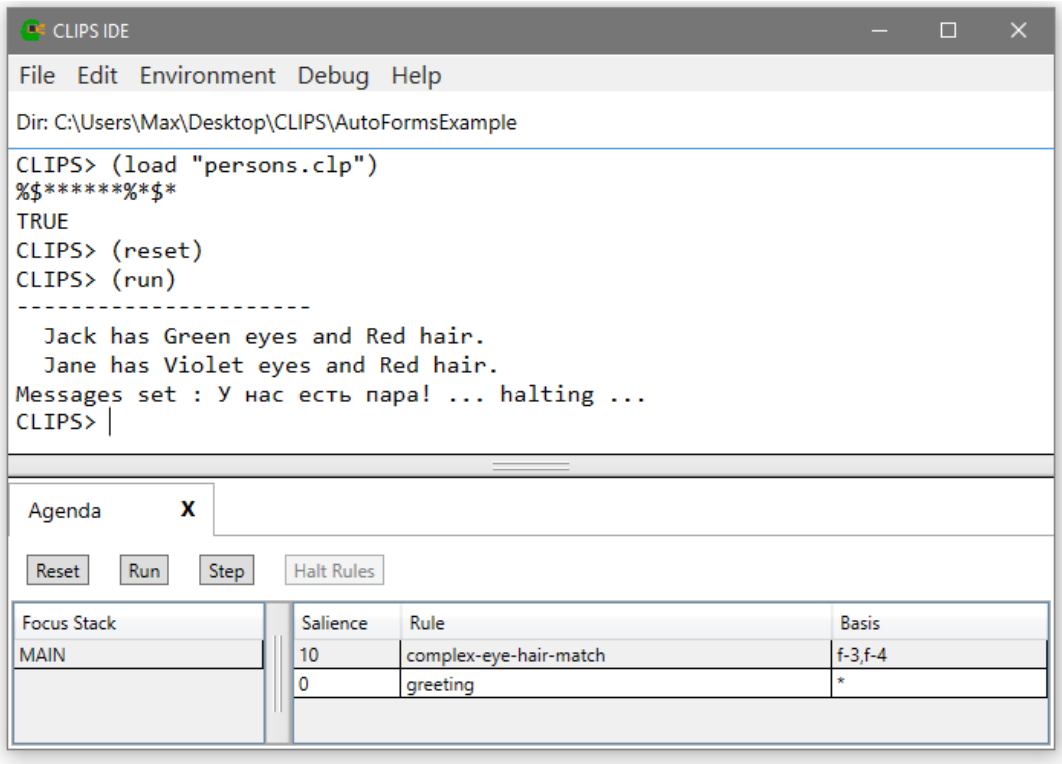
Русификация и GUI

Технически CLIPS Engine (движок) поддерживает Unicode, однако найти удобные IDE с возможностью использовать UTF-8 сложно. В дистрибутиве «из коробки» – MDI-интерфейс, не поддерживается Unicode (её мы использовали на лекциях):



Русификация и GUI (2)

IDE от Gary Riley – Unicode поддерживает, однако редактирование файлов только сторонними средствами (нет встроенного редактора). При этом неплохой набор средств отладки и трассировки кода.



Разработка примера ЭС

Для ЭС с графическим интерфейсом можно воспользоваться примером с оф. сайта (проект RouterFormsExample для проекта C# Windows Forms), однако там довольно сложные механизмы асинхронного взаимодействия, при этом пример иллюстрирует перенаправление вывода CLIPS, чего не всегда достаточно (иногда необходимо анализировать вывод ЭС).

Воспользуемся другим механизмом – в составе стандартных примеров есть проект CLIPSCLRWrapper, представляющий собой проект DLL-библиотеки (впрочем, эта же библиотека используется и в проекте RouterFormsExample). Её можно использовать для работы с машиной вывода CLIPS, добавления и извлечения фактов, а также управления (комадны Reset, Clear, Run и прочие).

Алгоритм работы

… обсуждался на лекциях.

Литература и ссылки

1. Код примеров приведён с сайта https://www.csie.ntu.edu.tw/~sylee/courses/es/index.html#outline
2. Рассел С., Норвиг П. Искусственный интеллект. Современный подход, 2-е изд., – М.: «Вильямс», 2006.
3. Люгер Дж. Ф. Искусственный интеллект. Стратегии и методы решений сложных проблем, 4-е изд., – М.: «Вильямс», 2003.