Herramientas

CLIPS

Contenido

- Introducción
- Elementos básicos de programación
- Variables
- Restricciones

CLIPS

- Historia
  - Inspirado en **OPS5** (Official Production System 5) y **ART** (Automated Reasoning Tool, Inference Corporation)
  - Implementado en **C** por eficiencia y portabilidad
  - Desarrollado por NASA (1986)
  - Licencia: *public domain*
- Herramienta para el desarrollo de Sistemas Expertos / SBC
  - Interprete de alto nivel
  - Interprete sistema de producción
  - Programación orientad a objetos COOL
  - Lenguaje operacional sintaxis tipo LISP

Interprete sistema de producción

- Memoria de trabajo: hechos (instancias)
- Base de reglas
- Motor de inferencias: adelante, RETE

- Programa CLIPS
  - Hechos y reglas (constructores de) (tb. objetos, funciones)
  - Los hechos que alcanzan la memoria de trabajo determinan que reglas se pueden disparar
  - El motor de inferencias determina que reglas y cuando se disparan
  - Actividad guiada por datos

Sobre CLIPS

- CLIPS user guide

  Introducción a la programación en CLIPS

- J. Giarrantano, G. Riley. EXPERT SYSTEMS, Principles and Programming, Third Edition. PWS Publishing Company, Boston 1998. ISBN 0-534-95053-1

- CLIPS reference manual
  - Vol I: programación básica
  - Vol II programación avanzada
  - Vol III: interfaces

Elementos básicos de programación

- Tipos de datos
- Funciones
- Constructores

Tipos de datos (I)

- Números: integer, float
  - -32, 15.09, -32.3e-7
- Símbolos: secuencia de letras, dígitos, $*=+/<>_?#.
  - CLIPS, Jess: Hola, sequence#1, _abc, nil
  - Sólo CLIPS: 127A
  - No válido: a&b

Tipos de datos (II)

- Strings
  - "Hola, mundo", "El dijo \"NO\"."
- Listas
  - (+ 3 2), (a b c), ("Hola, mundo"), ()
    (deftemplate foo (slot bar))

Hechos

- Ordenados: no requieren plantilla
  - (status walking)
  - (persona "Luis Prieto" hombre 53 1.83)

- No ordenados: requieren plantilla
  - (persona (nombre "Luis Prieto")
    (sexo hombre)
    (edad 53)
    (talla 1.83))

Operaciones con hechos (1)

- Añadir un hecho a la memoria de trabajo

  CLIPS> (assert (persona "Luis Prieto" hombre 53))

  <Fact-1>

- Examinar en la memoria de trabajo

  CLIPS> (facts)

  f-0   (persona "Luis Prieto" hombre 53)

  For a total of 1 fact.

- Eliminar un hecho de la memoria de trabajo

  CLIPS> (retract 0)

  CLIPS> (facts)

Plantillas para hechos ordenados

```
CLIPS> (deftemplate persona
"una persona concreta"
(slot nombre) (slot sexo)
(slot edad (type INTEGER))
(slot talla (default 1.70)))

CLIPS> (assert (persona
(nombre "Ana Rodriguez") (sexo mujer) (edad 52)))
<Fact-1>
```

Examinar hechos

```
CLIPS> (facts)
f-1     (persona (nombre "Ana Rodriguez") (sexo mujer) (edad
    52) (talla 1.7))
For a total of 1 fact.

CLIPS> (watch facts)
CLIPS> (modify 1 (edad 25))
<== f-1     (persona (nombre "Ana Rodriguez") (sexo mujer)
    (edad 52) (talla 1.7))
==> f-2     (persona (nombre "Ana Rodriguez") (sexo mujer)
    (edad 25) (talla 1.7))
<Fact-2>
```

Agrupar hechos

```
(deffacts varias-personas
    "normalmente en archivo"
    (persona
          (nombre "Juan Soto")
          (sexo hombre)
          (edad 32)
          (talla 1.83))
    (persona
          (nombre "Javier Ruiz")
          (sexo hombre)
          (edad 43)))
```

Para asertar: (reset)

Funciones

```
CLIPS> (printout t "El dijo \"NO\"." crlf)
El dijo "NO".

CLIPS> (deffunction hipotenusa (?a ?b)
(sqrt (+ (* ?a ?a) (* ?b ?b))))
CLIPS> (hipotenusa 4 3)
5

(deffunction inicializar ()
     (reset)
     (assert (comienzo)))
```

Crear reglas

(defrule <nombre-regla>
[<comentario>]
[<declaración>]
*<conditional-element>**
=>
<acciones>)*

Ejemplo de Regla

```
(defrule tareas-domingo "Cosas que hacer en
   domingo"
    (salience 0)
    (hoy es domingo)
    (es un dia soleado)
    =>
    (assert (tarea lavar coche))
    (assert (tarea cortar cesped)))
```

Otro ejemplo de regla

```
(deftemplate emergencia (slot tipo))
(deftemplate respuesta (slot accion))

(defrule emergencia-fuego "Un ejemplo de regla"
(emergencia (tipo fuego))
=>
(assert (respuesta
        (accion activar-dispersores))))
```

Disparo de regla

```
(load "E:/Docencia/IAI/Clips/EjemplosClase/Ejemplo2fuego.CLP")
Defining deftemplate: emergencia
Defining deftemplate: respuesta
Defining defrule: emergencia-fuego +j
TRUE
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (reset)
==> f-0     (initial-fact)
CLIPS> (assert (emergencia (tipo fuego)))
==> f-1     (emergencia (tipo fuego))
==> Activation 0      emergencia-fuego: f-1
<Fact-1>
CLIPS> (run)
==> f-2     (respuesta (accion activar-dispersores))
```

Variables en patrones I

```
(deftemplate   persona   (slot nombre)   (slot ojos)   (slot pelo))

(deffacts personas
   (persona (nombre Julia) (ojos azules) (pelo pelirroja))
   (persona (nombre Juan) (ojos verdes) (pelo rubio))
   (persona (nombre Javier) (ojos azules) (pelo moreno))
   (persona (nombre Jesus) (ojos verdes) (pelo rubio)))

(defrule encontrar-ojos-azules
   (persona (nombre ?nombre) (ojos azules))
 =>
   (printout t ?nombre " tiene los ojos azules." crlf))
```

Encontrar ojos azules I

```
CLIPS> (clear)
CLIPS> (load
    "E:/Docencia/IAI/Clips/EjemplosClase/EjemploVariables1.CLP"
    )
Defining deftemplate: persona
Defining deffacts: personas
Defining defrule: encontrar-ojos-azules +j
TRUE
CLIPS> (reset)
CLIPS> (run)
Javier tiene los ojos azules.
Julia tiene los ojos azules.
```

Variables en patrones II

```
(deftemplate   persona (slot nombre) (slot ojos)(slot pelo))
(deftemplate encontrar (slot ojos))
(deffacts personas
   (persona (nombre Julia) (ojos azules) (pelo pelirroja))
   (persona (nombre Juan) (ojos verdes) (pelo rubio))
   (persona (nombre Javier) (ojos azules) (pelo moreno))
   (persona (nombre Jesus) (ojos verdes) (pelo rubio)))

(defrule encontrar-ojos
   (encontrar (ojos ?ojos))
   (persona (nombre ?nombre) (ojos ?ojos))
   =>
   (printout t ?nombre " tiene ojos " ?ojos "." crlf))
```

Encontrar ojos azules II

```
CLIPS> (clear)
CLIPS> (load
    "E:/Docencia//IAI/Clips/EjemplosClase/EjemploVariables2.CLP")
Defining deftemplate: persona
Defining deftemplate: encontrar
Defining deffacts: personas
Defining defrule: encontrar-ojos +j+j
TRUE
CLIPS> (reset)
CLIPS> (assert (encontrar (ojos azules)))
<Fact-5>
CLIPS> (run)
Javier tiene ojos azules.
Julia tiene ojos azules.
```

Otras variables en las reglas

- Mudas (?), multicampo(?$)
- Hecho: (do washing on monday)
- Patrones

```
(do ?job on ?day)
(do ? ? monday)
(do ? on ?)
(do ? ? ?day)
(do $?)                ;multicampo muda
(do $? monday)
(do ?chore $?when) ; $?when ← (on monday)
```

Eliminar hechos desde las reglas

```
(defrule do-a-chore
    (today is ?day)
    ?chore <- (do ?job on ?day)  ;indice del hecho
    =>
    (printout t ?job " done")
    (retract ?chore)
)
```

Ampliando las Posibilidades de Confrontación: Restricciones

- Las restricciones, **field constrain**, permiten restringir los valores de un campo que satisfacen la confrontación de patrones.

Restricciones not **(~)**, or **(|)**

```
(defrule persona-sin-pelo-moreno
   (persona (nombre ?nombre) (pelo ~moreno))
   =>
   (printout t ?nombre " no tiene el pelo moreno." crlf))

(defrule persona-con-pelo-moreno-o-rubio
   (persona (nombre ?nombre) (pelo moreno | rubio))
   =>
   (printout t ?nombre " tiene el pelo moreno o rubio."
    crlf))
```

Restricciones and ***(&),***

- Habitualmente, junto a otras restricciones

```
(defrule pelo-moreno-o-rubio
   (persona (nombre ?nombre) (pelo ?color&moreno|rubio))
   =>
   (printout t ?nombre " tiene el pelo " ?color "." crlf))
```