Tema 2: Introducción a CLIPS

Carmen Graciani Díaz

José Luis Ruiz Reina

Dpto. Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

¿Qué es CLIPS?

- CLIPS $\equiv$ C Language Integrated Production Systems

  `http://www.ghg.net/clips/CLIPS.html`

- Desarrollado en el *Centro Espacial "Lyndon B. Johnson"* de la NASA, Houston, Texas (1984)

- Características:
  - Conocimiento: Reglas, objetos y procedimental
  - Portabilidad: implementado en C
  - Integración y Extensibilidad: Con programas en C, Java, FORTRAN, ADA...
  - Verificación y Validación
  - Documentación
  - Bajo coste: software libre

Una sesión con CLIPS

- Iniciar la sesión: `clips` (en EMACS `Alt-x run-clips`)

```
CLIPS> (+ 3 2)
5
CLIPS>
```

- Incluir hechos:

```
CLIPS> (assert (Tiene pelos))
<Fact-0>
CLIPS> (assert (Tiene pelos))
FALSE
CLIPS> (facts)
f-0     (Tiene pelos)
For a total of 1 fact.
```

- Identificador e índice de los hechos: `f-<índice>`

- Cerrar la sesión:

```
CLIPS> (exit)
```

Hechos ordenados

- `(<símbolo> <valor>*)`
- Tipos de valores: Números (enteros o como flotante), símbolos, cadenas.
- Distinción entre mayúsculas y minúsculas.
- Caracteres especiales: `"`, `(`, `)`, `&`, `|`, `<`, `~`, `;`, `?`, `$?`
- Símbolos reservados: `test`, `and`, `or`, `not`, `declare`, `logical`, `object`, `exists`, `forall`

Eliminar Hechos

```
CLIPS> (clear)
CLIPS> (facts)
CLIPS> (assert (Tiene pelos)
               (Tiene pezugnas)
               (Tiene cuello_largo)
               (Tiene rayas_negras))
<Fact-3>
CLIPS> (retract 1 2)
CLIPS> (facts)
f-0     (Tiene pelos)
f-3     (Tiene rayas_negras)
For a total of 2 facts.
CLIPS> (retract 2)
[PRNTUTIL1] Unable to find fact f-2.
CLIPS> (retract *)
CLIPS> (facts)
CLIPS>
```

Reglas

```
CLIPS> (clear)
CLIPS> (defrule Mamifero
          (Tiene pelos)
          =>
          (assert (Es mamifero)))
CLIPS> (defrule Ungulado
          (Es mamifero) (Tiene pezugnas)
          =>
          (assert (Es ungulado)))
CLIPS> (assert (Tiene pelos) (Tiene pezugnas)
               (Tiene cuello_largo)  (Tiene rayas_negras))
<Fact-3>
CLIPS> (agenda)
0      Mamifero: f-0
For a total of 1 activation.
CLIPS> (run)
CLIPS> (facts)
f-0     (Tiene pelos)
f-1     (Tiene pezugnas)
f-2     (Tiene cuello_largo)
f-3     (Tiene rayas_negras)
f-4     (Es mamifero)
f-5     (Es ungulado)
For a total of 6 facts.
CLIPS>
```

Reglas

```
CLIPS> (rules)
Mamifero
Ungulado
For a total of 2 defrules.
CLIPS> (ppdefrule Ungulado)
(defrule MAIN::Ungulado
   (Es mamifero)
   (Tiene pezugnas)
   =>
   (assert (Es ungulado)))
CLIPS> (agenda)
CLIPS>
```

Depurando errores

- Mostrar un mensaje ante determinadas acciones: `watch`
  - `facts <nombre de plantilla>*`
  - `activations <nombre de regla>*`
  - `rules <nombre de regla>*`
  - `focus`
- Grabando la sesión: `dribble-on`
- Ejecución parcial: `run`

Obteniendo ayuda

```
CLIPS> (help)
HELP_USAGE                    FUNCTION_SUMMARY
RELEASE_NOTES                 COMMAND_SUMMARY
CONSTRUCT_SUMMARY             INTEGRATED_EDITOR

MAIN Topic? command

COMMAND_SUMMARY

This section gives a general overview of the available CLIPS commands.

Subtopics:
ENVIRONMENT_COMMANDS          DEFGLOBAL_COMMANDS
DEBUGGING_COMMANDS            DEFFUNCTION_COMMANDS
DEFTEMPLATE_COMMANDS          GENERIC_FUNCTION_COMMANDS
FACT_COMMANDS                 COOL_COMMANDS
DEFFACTS_COMMANDS             DEFMODULE_COMMANDS
DEFRULE_COMMANDS              MEMORY_COMMANDS
AGENDA_COMMANDS               TEXT_PROCESSING_COMMANDS

COMMAND_SUMMARY Topic? fact
```

Obteniendo ayuda

```
COMMAND_SUMMARY
  FACT_COMMANDS

The following commands display information about facts.

FACTS: Display the facts in the fact-list.
(facts [<module-name>]
       [<start-integer-expression>
       [<end-integer-expression>
       [<max-integer-expression>]]])

LOAD-FACTS: Asserts facts loaded from a file.
(load-facts <file-name>)

SAVE-FACTS: Saves facts to a file.
(save-facts <file-name> [<save-scope> <deftemplate-names>*])
<save-scope> ::= visible | local

DEPENDENCIES: Lists the partial matches from which a fact or
              instance receives logical support.
(dependencies <fact-or-instance-specifier>)

PRESS <RETURN> FOR MORE. PRESS <A>,<RETURN> TO ABORT. a
COMMAND_SUMMARY Topic?
MAIN Topic?
CLIPS>
```

Utilizando el editor

animales.clp

```
;;; Reglas que permiten clasificar un animal en función
;;; de sus características.

;;; Reglas para animales concretos.

(defrule Jirafa "Características de una Jirafa"
  (Es ungulado)
  (Tiene cuello_largo)
  =>
  (assert (Es jirafa)))

(defrule Cebra "Características de una Cebra"
  (Es ungulado)
  (Tiene rayas_negras)
  =>
  (assert (Es cebra)))
```

Utilizando el editor

animales.clp

```
;;; Ungulados:

(defrule Ungulado_1 "Características de los ungulados"
  (Es mamifero)
  (Tiene pezugnas)
  =>
  (assert (Es ungulado)))

(defrule Ungulado_2 "Características de los ungulados"
  (Es mamifero)
  (Rumia)
  =>
  (assert (Es ungulado)))

;;; Mamíferos:

(defrule Mamifero_1 "Características de los mamíferos"
  (Tiene pelos)
  =>
  (assert (Es mamifero)))

(defrule Mamifero_2 "Características de los mamíferos"
  (Da leche)
  =>
  (assert (Es mamifero)))
```

Tabla de seguimiento

```
CLIPS> (load "animales.clp")
...
TRUE
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (watch rules)
CLIPS> (assert (Tiene pelos)
               (Tiene pezugnas)
               (Tiene rayas_negras))
==> f-0     (Tiene pelos)
==> Activation 0      Mamifero_1: f-0
==> f-1     (Tiene pezugnas)
==> f-2     (Tiene rayas_negras)
<Fact-2>
CLIPS> (run)
FIRE    1 Mamifero_1: f-0
==> f-3     (Es mamifero)
==> Activation 0      Ungulado_1: f-3,f-1
FIRE    2 Ungulado_1: f-3,f-1
==> f-4     (Es ungulado)
==> Activation 0      Cebra: f-4,f-2
FIRE    3 Cebra: f-4,f-2
==> f-5     (Es cebra)
CLIPS>
```

Tabla de seguimiento

I	Base de Hechos	E	Agenda	D
0	`(Tiene pelos)`	0	`Mamifero_1`: f-0	1
1	`(Tiene pezugnas)`	0		
2	`(Tiene rayas_negras)`	0		
3	`(Es mamifero)`	1	`Ungulado_1`: f-3,f-1	2
4	`(Es ungulado)`	2	`Cebra`: f-4,f-2	3
5	`(Es cebra)`	3		

Comodines: mudos, simples y múltiples

- Comodín simple: `?x, ?y, ?elemento, ?`
- Comodín múltiple: `$?x, $?y, $?elemento, $?`

notas.clp

```
(defrule Elimina "Elimina a los presentados de la lista"
  (Alumno ?nombre ? $?)
  ?l <- (Lista $?i ?nombre $?f)
  =>
  (retract ?l)
  (assert (Lista ?i ?f)))
```

Tabla de seguimiento

```
CLIPS> (clear)
CLIPS> (load "notas.clp")
...
TRUE
CLIPS> (assert (Lista Mar Ana Luis Pepe)
               (Alumno Mar) (Alumno Ana 2 3 9) (Alumno Luis) (Alumno Pepe 3))
==> f-0     (Lista Mar Ana Luis Pepe)
==> f-1     (Alumno Mar)
==> f-2     (Alumno Ana 2 3 9)
==> Activation 0      Elimina: f-2,f-0
==> f-3     (Alumno Luis)
==> f-4     (Alumno Pepe 3)
==> Activation 0      Elimina: f-4,f-0
<Fact-4>
CLIPS> (run)
FIRE    1 Elimina: f-4,f-0
<== f-0     (Lista Mar Ana Luis Pepe)
<== Activation 0      Elimina: f-2,f-0
==> f-5     (Lista Mar Ana Luis)
==> Activation 0      Elimina: f-2,f-5
FIRE    2 Elimina: f-2,f-5
<== f-5     (Lista Mar Ana Luis)
==> f-6     (Lista Mar Luis)
CLIPS> (facts)
f-1     (Alumno Mar)
f-2     (Alumno Ana 2 3 9)
f-3     (Alumno Luis)
f-4     (Alumno Pepe 3)
f-6     (Lista Mar Luis)
For a total of 5 facts.
```

Tabla de seguimiento

I	Base de Hechos	E	S	Agenda	D	S
0	`(Lista Mar Ana Luis Pepe)`	0	1			
1	`(Alumno Mar)`	0				
2	`(Alumno Ana 2 3 9)`	0		`Elimina`: f-2,f-0 (?n≡Ana ?l≡0 ?i≡Mar ?f≡Luis Pepe)		1
3	`(Alumno Luis)`	0				
4	`(Alumno Pepe 3)`	0		`Elimina`: f-4,f-0 (?n≡Pepe ?l≡0 ?i≡Mar Ana Luis ?f≡)	1	
5	`(Lista Mar Ana Luis)`	1	2	`Elimina`: f-2,f-5 (?n≡Ana ?l≡5 ?i≡Mar ?f≡Luis)	2	
6	`(Lista Mar Luis)`	2				

Restricciones en los patrones

- Valor literal

```
(Alumno Mar)
```

- Mismo valor para un mismo comodín

```
(Lista $?  ?n $?)  (Alumno ?n $?)
```

- Conectivas: `&` (y), `|` (ó), `~` (no)

```
(Alumno Luis|Pepe $?),(Alumno ~Mar&~Ana $?)
```

- Predicados: `:`

```
(Alumno ?  ?x&:(> ?x 5) $?)
```

- Valor de una función: `=`

```
(Alumno ?  ?  ?x =(* ?x ?x))
```

Patrones negativos

- No existencia de información.

lista.clp

```
(defrule Repasa-lista "Elimina a los que no están en la lista"
  ?a <- (Alumno ?n $?)
  (not (Lista $? ?n $?))
  =>
  (retract ?a))
```

I	Base de Hechos	E	S	Agenda	D	S
0	(Alumno Mar)	0		Repasa-lista: f-0 (?a≡0, ?n≡Mar)		0
1	(Alumno Ana 2 3 9)	0	2	Repasa-lista: f-1 (?a≡1, ?n≡Ana)	2	
2	(Alumno Luis)	0	1	Repasa-lista: f-2 (?a≡2, ?n≡Luis)	1	
3	(Alumno Pepe 3)	0		Repasa-lista: f-3 (?a≡3, ?n≡Pepe)		0
4	(Lista Mar Pepe)	0				

Tabla de seguimiento

```
CLIPS> (clear)
CLIPS> (load "lista.clp")
...
TRUE
CLIPS>  (assert (Alumno Mar) (Alumno Ana 2 3 9) (Alumno Luis) (Alumno Pepe 3)
                (Lista Mar Pepe))
==> f-0     (Alumno Mar)
==> Activation 0      Lista: f-0,
==> f-1     (Alumno Ana 2 3 9)
==> Activation 0      Lista: f-1,
==> f-2     (Alumno Luis)
==> Activation 0      Lista: f-2,
==> f-3     (Alumno Pepe 3)
==> Activation 0      Lista: f-3,
==> f-4     (Lista Mar Pepe)
<== Activation 0      Lista: f-3,
<== Activation 0      Lista: f-0,
<Fact-4>
CLIPS> (run)
FIRE    1 Lista: f-2,
<== f-2     (Alumno Luis)
FIRE    2 Lista: f-1,
<== f-1     (Alumno Ana 2 3 9)
CLIPS> (facts)
f-0     (Alumno Mar)
f-3     (Alumno Pepe 3)
f-4     (Lista Mar Pepe)
For a total of 3 facts.
CLIPS>
```

Negación de un patrón

- El patrón `(initial-fact)` y el comando `reset`.

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (defrule Crea-lista (not (Lista $?)) => (assert (Lista)))
CLIPS> (agenda)
CLIPS> (assert (Alumno Mar) (Alumno pepe))
<Fact-1>
CLIPS> (agenda)
CLIPS> (deffacts ejemplo (Alumno Mar) (alumno Luis))
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (Alumno Mar)
f-2     (alumno Luis)
For a total of 3 facts.
CLIPS> (agenda)
0      Crea-lista: f-0,
For a total of 1 activation.
CLIPS>
```

Plantillas

- Permiten abstraer la estructura de un hecho asignando un nombre a cada dato de un hecho.
- Constructor: `deftemplate`

```
(deftemplate <símbolo> [<cadena>]
  <definición de campo>*)

<definición de campo> ::=
        (slot|multislot <símbolo> <atributo>*)
```

- Cómo atributo se pueden indicar restricciones, valores por defecto, ...

Módulos

- Posibilidad de tener varias agendas simultáneamente
- Cada módulo tiene sus propias construcciones: hechos, reglas, ...
- El módulo actual

```
(defmodule <nombre>
  (export <elemento>)*
  (impot <módulo> <elemento>)*)
```

- La importación y exportación permite que distintas construcciones sean "visibles" por varios módulos

Módulos

- En cada momento se activan y desactivan las correspondientes reglas de cada agenda, aunque se dispara de la agenda *enfocada*
- El cambio de agenda se gestiona mediante una *pila de focos*
- Apilar:
  - La acción `focus`
  - Reglas con *autoenfoque*
  - Los comandos `run` y `reset`
- Desapilar:
  - Al vaciarse la agenda de la cima
  - Los comando `pop-focus` y `clear-focus-stack`
  - La acción `return`

Prioridades

- Reglas con prioridad explícita para controlar la ejecución
- Las activaciones se colocan en las agendas por orden de prioridad

```
(declare (salience <numero>))
```

- Valores:
  - Mínimo: -10000
  - Máximo 10000
  - Por defecto: 0

Ciclo básico de ejecución

- Si se alcanza el límite de disparos o no hay foco, se termina; sino se selecciona la primera regla de la agenda del módulo enfocado.
  - Si la agenda está vacía se elimina el módulo de la pila de focos.
  - Si la pila de focos está vacía, se interrumpe la ejecución.
  - Se reinicia el ciclo.
- Se ejecutan las acciones de la regla seleccionada.
  - La acción `return` elimina el módulo de la pila de focos. Se incrementa el número de disparos.
- Las reglas activadas por las acciones anteriores se incluyen en su agenda según su prioridad y la estrategia de resolución de conflictos.
  - Las reglas desactivadas se eliminan de su agenda.
- Se reinicia el ciclo.

Estrategias de resolución de conflictos

- Se colocan por orden de prioridad (de mayor a menor).
- Entre las de la misma prioridad :
  - **profundidad**: Sobre las existentes.
  - **anchura**: Tras las existentes.
  - **simplicidad**: Número de restricciones, de menos a más.
  - **complejidad**: Cómo la anterior pero de más a menos.
  - **LEX**: Antigüedad de uso (de menos a más).
  - **MEA**: Cómo la anterior pero de más a menos.
  - **aleatorio**.
- Si se activan simultáneamente y no se puede determinar su orden relativo, se colocan arbitrariamente.
  - Puede influir el orden de definición. **No** basar en esta dependencia el funcionamiento del sistema.

Bibliografía

- Giarratano, J.C. y Riley, G. *Sistemas expertos: Principios y programación (3 ed.)*. International Thomson Editores, 2001
  - Cap. 7: "Introducción a CLIPS"
  - Cap. 8: "Comparación de patrones"
  - Cap. 9: "Comparación avanzada de patrones"
  - Cap. 10: "Diseño modular y control de la ejecución"
  - Cap. 11: "Eficiencia de los lenguajes basados en reglas"
  - Cap. 12: "Ejemplos de diseño de sistemas expertos"
  - Apéndice E: "Resumen de comandos y funciones de CLIPS"

- Giarratano, Joseph C. y Riley, Gary D. *Expert Systems Principles and Programming (3 ed.)* (PWS Pub. Co., 1998).

Bibliografía

- CLIPS User's Guide
- CLIPS Reference Manual. Volume I. Basic Programming Guide
- CLIPS Reference Manual. Volume II. Advanced Programming Guide
- Kowalski, T.J. y Levy, L.S. *Rule–Based Programming* (Kluwer Academic Publisher, 1996)