# Sistemas de producción. Control de ejecución

Virginia Francisco
Rubén Fuentes
Alberto Díaz
Carmen Fernández



## Control de ejecución en Flujo de ejecución en Clips

- A veces es necesario conseguir cierta secuencialidad en la ejecución
- Existen distintos métodos para controlar la ejecución:
  - Hechos de control
  - Prioridades
  - Módulos



## Hechos de control

- Los hechos de control empotran el control de ejecución en las reglas
- Por ejemplo, en un juego de dos jugadores para controlar el turno se crea un hecho (turno h) o (turno c) para establecer si el turno es del humano o del computador
- Desventajas:
  - Se mezcla el conocimiento del dominio con la estructura de control => Mantenimiento y desarrollo más costoso



## Hechos de control. Ejemplo

```
(defrule player-select
   (phase choose-player)
   =>
   (printout t "Who moves first (Computer:c Human:h)? ")
   (assert (player-move (read))))

(defrule human-move
   (player-move h)
   ...
   =>
   ...
   (assert (player-move c)))

(defrule computer-move
   (player-move c)
   ...
   =>
   ...
   (assert (player-move h)))
```

## Control de ejecución con prioridades

- Prioridad para reglas especiales
  - Regla inicial
    - Prioridad alta
  - Regla por defecto si no se aplica ninguna de las otras
    - Prioridad baja



## Control de ejecución con prioridades. Ejemplo

```
(defrule engine-sluggish ""
   (engine-sluggish yes)
   (not (repair ?))
   =>
   (assert (repair "Clean the fuel line.")))
(defrule engine-misfires ""
   (engine-misfires yes)
   (not (repair ?))
   =>
   (assert (repair "Point gap adjustment.")))
...
(defrule no-repairs ""
  (declare (salience -10))
  (not (repair ?))
  =>
  (assert (repair "Take your car to a mechanic.")))
(defrule system-banner ""
  (declare (salience 10))
  =>
  (printout t "The Engine Diagnosis Expert System"))
(defrule print-repair ""
  (declare (salience 10))
  (repair ?item)
  =>
  (printout t "Suggested Repair:")
  (format t " %s%n%n%n" ?item))
```



## Módulos (I)

- Cuando un problema es grande conviene dividirlo en subproblemas y poner cada subproblema en un módulo
- Con módulos la selección de reglas y el razonamiento son más eficientes
- Los módulos representan diferentes estados en la resolución del problema, y aíslan los hechos y las reglas
- El flujo de ejecución en Clips se hace por módulos, se comienza la ejecución en el modulo MAIN y para cambiar de módulo hay que indicarlo explícitamente
- Los módulos se definen con *defmodule* y se establece cuándo se usan con *focus*

```
(defmodule QUESTIONS)
```



## Módulos (II)

- Cada módulo tiene su propia agenda y su propia red RETE
  - `(agenda)` → muestra la agenda del módulo en curso
  - `(agenda *)` → muestra la agenda de cada módulo
- Las instrucciones del flujo de ejecución en Clips operan sobre las construcciones del módulo en curso
- Deberemos especificar a qué módulo pertenecen las construcciones con *<modulo>::*

```
(defrule RECOMMEND::form-1040EZ
    (user (income ?i&:(< ?i 50000))
    (dependents ?d&:(eq ?d 0)))
    (answer (ident interest) (text no))
     =>
    (assert (recommendation (form 1040EZ)
    (explanation "Income below threshold, no dependents"))))
```

## Módulos. Focus

- Clips mantiene un current-focus que determina la agenda que se usa
- El current-focus por defecto es el módulo MAIN
- Para cambiar el current-focus hay que usar la instrucción:

```
(focus <nombre-modulo>+)
```

- Cuando se cambia el current-focus Clips apila el focus anterior en la pila focus stack
- Cuando la agenda del current-focus se vacia se obtiene el siguiente focus del focus stack
- Funciones útiles relacionadas con el focus:
  - `(clear-focus-stack)`
  - `(pop-focus)`
  - `(get-current-module)`



## Módulos. Auto-focus

- Si una regla tiene declarado auto-focus con el valor true, el flujo de ejecución en Clips intenta activarla siempre, de tal forma que si se activa cambia el focus al módulo de esa regla
- Con *return* se devuelve el control al módulo en el que se estaba cuando se activó la regla con auto-focus a true
  - Esto permite definir reglas que se pueden activar desde cualquier módulo



## AutoFocus. Ejemplo

```
(defrule QUESTIONS::ask-question-by-id
  "Given the id of a question, ask it and assert the answer"
  (declare (auto-focus TRUE))
  (question (ident ?id) (text ?text) (type ?type))
  (not (answer (ident ?id)))
  ?ask <- (ask (ident ?id))
  =>
  (bind ?answer (ask-user ?text ?type))
  (assert (answer (ident ?id) (text ?answer)))
  (retract ?ask))
```



## Módulos. Exportación e importación de hechos

- Los hechos pueden ser compartidos entre módulos, para ello hay que exportarlos e importarlos
- Exportar:

```
(export ?ALL)
```

- Importar:

```
(import ?ALL)
```

## Módulos. Ejemplo (taxadvisor)

**Asesor de formularios de impuestos**

- Pregunta a un usuario sobre su situación financiera y le recomienda cual es el formulario de impuestos adecuado
- Etapas
  - Bienvenida
  - Preguntar al usuario
  - Decidir qué formularios son adecuados
  - Notificar al usuario la recomendación
- Módulos
  - INTERVIEW, RECOMMEND, REPORT
  - QUESTIONS
    - Auto-focus TRUE → Si tengo una pregunta sin respuesta, haz la pregunta



## Módulos. Ventajas

- Permiten dividir la base de conocimiento
- Se puede controlar qué hechos son visibles en cada módulo
- La ejecución es más eficiente al tener en la agenda y la red RETE únicamente los hechos y las reglas del módulo en curso



## Materiales de referencia


- Friedman-Hill, E. Jess in Action: Rule-Based Systems in Java. 2003 http://cisne.sim.ucm.es/record=b2548266~S6*spi
- Strauss, M. Jess. The Java Expert System Shell. Abril 2007 (tutorial disponible en el Campus Virtual que incluye una discusión en profundidad del ejemplo del taxadvisor)
- https://www.cs.us.es/cursos/ia2-2004/temas/tema-01.pdf