Introducción a Jess Interpretado: Indice

- Instalación de Jess Interpretado
- Tratamiento de Hechos
  - A) Hechos ordenados
  - B) Hechos no ordenados: deftemplate
  - C) Crear hechos
    - Iniciales: deffacts
    - Añadir / quitar hechos en ejecución: assert / retract
- Tratamiento de Reglas

Instalación de Jess Interpretado

- Desempaqueta el Jess71p2.zip en el directorio que quieras
  - mejor donde tienes instalados el resto de tus programas
- Utiliza un editor de textos que compruebe paréntesis
  - El más simple: *notepad*++
  - Si conoces *emacs* es cómodo
- Ejecución en línea de comandos
  - **cd "c:\Program Files\Jess71p2\bin"**
  - **jess.bat**
  - Aparece el prompt: **Jess>**

A) - Hechos ordenados

- Secuencia de literales separados por espacios
  - Codifican la información según la posición
  - El primer literal representa una relación entre los restantes
  - Los restantes son como atributos o slots sin nombre

```
(convenio)
(alumnos Juan Luis Pedro)
(lista-de-la-compra pan leche arroz)
```

- Se incluyen en la B. Hechos (no se declaran)

```
(assert (alumnos Juan Luis Pedro))
(assert (temp 25) )
```

- El encaje o *matching* con el LHS de una regla
  - Los literales deben estar en el mismo orden que en la regla
- Se usan para conceptos con poca información
  - Es dificil manejar sus atributos por separado
  - Para ese manejo se usan los hechos NO ordenados

B) - Hechos no ordenados: deftemplate

- Define un tipo de hecho, tiene varios slots (atributos) con nombre

```
(deftemplate persona
       (slot nombre (type SYMBOL) (default "sin nombre"))
       (multislot apellidos (type SYMBOL))
       (slot edad (type NUMBER)(default (+ 10 15)))
       (slot estado (type SYMBOL)(allowed-values soltero
                    libre casado viudo)(default soltero)))
```

- Para cada slot se puede definir:
  - el tipo: type (valores posibles siguiente transparencia)
  - Valor por defecto: default (admite cualquier operación)
  - Valores permitidos: allowed-values
  - Slot multivaluados: multislot
- Se necesita definir cada hecho con **assert** o **deffacts** (se ve después)

Sintaxis completa deftemplate

((deftemplate *template-name*

[extends *template-name*]

["Documentation comment"]

[(declare (slot-specific TRUE | **FALSE**)

(backchain-reactive TRUE | **FALSE**)

(from-class *class name*)

(include-variables TRUE | **FALSE**)

(ordered TRUE | **FALSE**))]

(slot | multislot slot-name /*-- multislot es multivaluado --*/

([(type ANY | INTEGER | FLOAT | NUMBER | SYMBOL | STRING |

LEXEME | OBJECT | LONG)]

[(default default value)]

[(default-dynamic *expression*)]

[(allowed-values *expression*+)])*)

C) – Crear hechos: iniciales con deffacts

- Sintaxis

(deffacts *deffacts-name* ["Documentation comment"] *fact)**

- Ejemplo

```
(deffacts alumnos "mi clase"    /* -- hechos iniciales --*/
   (curso IAIC 2010-2011)
   (persona (nombre Pepe)(apellidos Gomez Garcia))
   (persona (nombre Juan)(edad 25)))
```

→ **No olvidar hacer:**

```
(reset) /*- borra todos los hechos de MT, añade los deffact */
(facts) /*---- lista los hechos actuales en la M.T. --*/
f-0   (MAIN::initial-fact)
f-1   (MAIN::personaZ (nombre Pepe) (edad 25) (estado soltero) (apellidos Gomez
Garcia))
f-2   (MAIN::personaZ (nombre Juan) (edad 25) (estado soltero) (apellidos ))
f-3   (MAIN::curso IAIC 2010-2011)
```

- Si vuelvo a ejecutar deffacts, debo ejecutar reset de nuevo

Añadir / quitar hechos en ejecución: assert / retract

- Para incluir / eliminar en la MT un hecho (ordenado o no)

  (assert <hecho>) (retract <hecho>)

- Pueden colocarse en el RHS de las reglas o independientes
- Ejemplos

```
(assert (tipo-hecho bien ordenado))
(assert (persona (edad 50))
```


Reglas

- Sintaxis:

```
(defrule <nombre-regla>       ;; para eliminar : undefrule
[<documentación opcional>]    ;; Se pone entre “ ”
[(declare (salience <num>))]  ;; prioridad de ejecución
(patrón 1)
(patrón 2)
...
(patrón N)
=>
(acción 1)
(acción 2)
...
(acción N)
)
```

- Ver el contenido de una regla **(ppdefrule marca-del-600)**
- Una regla sin LHS se ejecuta solo cada vez que se ejecute el **reset**.

Reglas: Ejemplo

- Ejemplo:

```
(deffacts ini
(letra 1 c)
(letra 2 a)
(letra 3 c)
(letra 4 b)
(letra 5 a)
(letra 6 b) )
((defrule r2  "para ordenar letras H5Ej3.clp"
   ?h1<-(letra ?i c) ;;  asigna el hecho 1 a la variable ?h1, ?i = 1
   ?h2<-(letra ?j a)
   (test (eq ?i (- ?j 1)))    ;; se ejecuta regla si ?i = ?j - 1
=>
 (retract ?h1)  ;; quita el hecho 1 de la MT
 (retract ?h2) ;; incluye em la MT el hecho (letra 1 a)
 (assert (letra ?i a))
 (assert (letra ?j c)))
```


Modificar desde fuera la ejecución de la regla

- Permite usar una regla con diferentes fines :
  - busco: solteros, casados, libres
- Al repetir **?est** fuerza que coincida su valor en los hechos que equiparen

```
(deftemplate busca (slot estado))  ;; un hecho para indicar qué busco
(defrule busca-candidato
    (busca (estado ?est)) ;; persona y busca han de tener el mismo ?est
    ?candidato <- (persona (nombre ?nom) (edad ?ed) (estado ?est))
                      ;;  asigno un hecho a la variable ?candidato
    (test (< ?ed 30))  ;; solo ejecuto regla si cumple test
  =>
    (modify ?candidato (estado libre)) ;; cambia el estado de candidato
    (assert (tengo candidato))
    (printout t "mi candidato es: " ?nom " estado anterior: " ?est
                                                        crlf) )
 (assert (busca (estado soltero))) ;; decido buscar solteros
(reset)
(run)
```

Operaciones de Entrada y Salida interactivas - I

- Permite introducir un hecho entre comillas

```
(defrule inserta-hecho   ; no tiene LHS: se dispara si (reset) + (run)
        =>                ; para escribir un texto al disparar regla
      (printout t "Escribe un hecho como cadena" crlf)
      (assert-string (read)))  ; para leer un hecho
(reset)
(run)
Escribe un hecho como cadena        ; el ordenador escribe esto
"(persona (nombre NEO) ) "          ; el usuario escribe eso
                                    ; el ssitema añade el hecho:
;;   (persona (nombre NEO) (apellidos ) (edad 25) (estado soltero))
```


Operaciones de Entrada y Salida interactivas – I I

- Permite introducir una línea y construir un hecho concatenando paréntesis

```
(defrule lee-linea
        =>
      (printout t "Introduce datos." crlf)
      (bind ?cadena (readline))
      (assert-string (str-cat "(" ?cadena ")")))
Introduce datos                     ; el ordenador escribe esto
persona                             ; el usuario escribe eso
                                    ; el sistema añade el hecho:
; (persona (nombre "sin nombre") (apellidos ) (edad 25) (estado
   soltero))
```

- Ejecución condicional : para poner apellidos a quien no tenga

```
(defrule poner-apellidos ;; solo equiparan personas sin apellidos
   ?persona <-(persona (nombre ?nombre) (apellidos))
=>
   (printout t crlf "Introduce apellidos para " ?nombre ": " )
   (modify ?persona (apellidos (read)))))
```

Utilidades auxiliares

- Listar las templates definidas (nativas y por el usuario)

```
(list-deftemplates)
```

- Trazar lo que va pasando (watch , unwatch)

```
(watch facts)
(watch rules)
(watch activations)
(clear)   ;; limpia MT incluidos hechos iniciales: deffacts
(batch "C:/ejemploletrasversion1.clp") ; carga ese fichero
            ;; recuerda poner dentro del fichero: (reset) (run)
```

- Otras utilidades

```
(matches <nombre-de-regla>) ;; Muestra hechos que equipara LHS y RHS
(list-< def <templates|
```


Sobre sintaxis general

- Comentarios con ; resto de línea ó /* entre estos signos */