Università di Bergamo
Facoltà di Ingegneria

Intelligenza Artificiale

Paolo Salvaneschi

B1_2 V1.3

Introduzione a JESS

Il contenuto del documento è liberamente utilizzabile dagli studenti, per studio personale e per supporto a lezioni universitarie.
Ogni altro uso è riservato, e deve essere preventivamente autorizzato dall' autore.
Sono graditi commenti o suggerimenti per il miglioramento del materiale

INDICE

- Caratteristiche generali
- Il linguaggio Jess: elementi di base
- Liste
- Funzioni
- Variabili
- Funzioni e liste
- Strutture di controllo
- Definire una funzione

INDICE

- Fatti
- Regole
- Patterns
- Conditional elements (CE)
- Strategia
- Gestione dell’agenda
- Moduli

Caratteristiche generali

- Sistema a regole implementato in Java
- Realizzato da

  Ernest J. Friedman-Hill
  Distributed Computing Systems
  Sandia National Laboratories
  Livermore, CA - USA
- Disponibile presso:

  http://herzberg.ca.sandia.gov/jess
- Licenza gratuita (incluso il sorgente) per usi non-commerciali

Caratteristiche generali

- Bibliografia:
  - Ernest J. Friedman-Hill
    JESS IN ACTION
    Manning, 2003
  - Documentazione scaricabile associata al prodotto

Caratteristiche generali

- Il sistema Jess si attiva da linea di comando

```
C:\> java jess.Main
Jess>
```

- O da file

```
C:\> java jess.Main [nome file]
```

Caratteristiche generali

- Puzzle
  - A foursome of golfers is standing at a tee, in a line from left to right. Each golfer wears different colored pants; one is wearing red paints;
  - The golfer to Fred's right is wearing blue pants;
  - Joe is in second in line;
  - Bob is wearing the plaid pants;
  - Tom isn't in position 1 or 4 and isn't wearing orange pants
- In what order will the four golfers tee off, and what color are each golfer’s pants?

tee -- supporto su cui appoggia la palla da golf
to tee off -- dare la mazzata iniziale

- Approccio:
  - Rappresento le possibili combinazioni di nomi, posizioni e colori
  - Scrivo la regola che descrive il problema
  - Attivo il motore di Jess per trovare la soluzione.

	1 2 3 4	red blue plaid orange
name	position	color
Fred	**?p1**	**?c1**
Joe	**?p2**	**?c2**
Bob	**?p3**	**?c3**
Tom	**?p4**	**?c4**

Caratteristiche generali

Es_1

Ordered facts

```
(deftemplate pants-color (slot of) (slot is))
(deftemplate position (slot of) (slot is))
```

Definisco la struttura dei fatti
che inserirò nella working memory

Fatti del tipo: `(pants-color (of Bob)(is plaid))`

Caratteristiche generali

```
(deftemplate pants-color (slot of) (slot is))
(deftemplate position (slot of) (slot is))

(defrule generate-possibilities
  =>
  (foreach ?name (create$ Fred Joe Bob Tom)
    (foreach ?color (create$ red blue plaid orange)
                        (assert (pants-color (of ?name)
                                             (is ?color))))
    (foreach ?position (create$ 1 2 3 4)
                        (assert (position (of ?name)
                                          (is ?position))))))
```

Carico i fatti nella working memory

16 fatti del tipo: **`(pants-color (of Bob)(is plaid))`**
e
16 fatti del tipo: **`(position (of Joe)(is 3))`**

Regola

```
(defrule find-solution
    ;; The golfer to Fred's right is wearing blue pants.
    (position (of Fred) (is ?p1))
    (pants-color (of Fred) (is ?c1))

    (position (of ?n&~Fred) (is ?p&:(eq ?p (+ ?p1 1))))
    (pants-color (of ?n&~Fred) (is blue&~?c1))

    ;; Joe is in position #2
    (position (of Joe) (is ?p2&2&~?p1))
    (pants-color (of Joe) (is ?c2&~?c1))

    ;; Bob is wearing the plaid pants
    (position (of Bob) (is ?p3&~?p1&~?p&~?p2))
    (pants-color (of Bob&~?n) (is plaid&?c3&~?c1&~?c2))
```

Each golfer wears different colored pants

Each golfer is in a different position

……

NB `(a ?X&:(> ?X 10)`

matches "a" facts with one field, a number greater than 10

……

```
  ;; Tom isn't in position 1 or 4 and isn't wearing
orange
  (position (of Tom&~?n)
  (is ~1&~4&?p4&~?p1&~?p2&~?p3))
  (pants-color (of Tom)
  (is ?c4&~orange&~blue&~?c1&~?c2&~?c3))
  =>
  (printout t Fred  " " ?p1 " " ?c1 crlf)
  (printout t Joe  "  " ?p2 " " ?c2 crlf)
  (printout t Bob  "  " ?p3 " " ?c3 crlf)
  (printout t Tom  "  " ?p4 " " ?c4 crlf)
  )


(reset)
(run)
```

commento

```
;; The golfer to Fred's right is wearing blue pants.
(position (of Fred) (is ?p1))
(pants-color (of Fred) (is ?c1))

(position (of ?n&~Fred) (is ?p&:(eq ?p (+ ?p1 1))))
(pants-color (of ?n&~Fred) (is blue&~?c1))
```

Each golfer wears different colored pants

The pant's color of the golfer to Fred's right is blue and is not the pant's color of Fred

commento

```
;; Joe is in position #2
(position (of Joe) (is ?p2&2&~?p1))
(pants-color (of Joe) (is ?c2&~?c1))
```

.....

Each golfer is in a different position

Joe is in position 2 and is not the position of Fred (?p1)

commento

```
;; Bob is wearing the plaid pants
(position (of Bob) (is ?p3&~?p1&~?p&~?p2))
(pants-color (of Bob&~?n) (is plaid&?c3&~?c1&~?c2))
```

Joe is in position ?p3
And is not in the position of Fred (?p1)
And is not in the position of Joe (?p2)
And is not in the position of “the golfer to Fred's right” (?p)
because “the golfer to Fred's right” is wearing blue pants, while Bob is wearing plaid pants

- Risultato:

```
C:\>java jess.Main C:\Programmi\Jess61p4\appoggio\es_1.clp

Jess, the Java Expert System Shell
Copyright (C) 2001 E.J. Friedman Hill and the Sandia Corporation
Jess Version 6.1p4 7/8/2003

Fred 1 orange
Joe  2 blue
Bob  4 plaid
Tom  3 red
```

Es_2

- Modifica: elimino parte della conoscenza

```
;; Joe is in position #2
  (position (of Joe) (is ?p2&2&~?p1))
  (pants-color (of Joe) (is ?c2&~?c1))
```

```
;; We don't know anything about Joe, really
  (position (of Joe) (is ?p2&~?p1))
  (pants-color (of Joe) (is ?c2&~?c1))
```

- Risultato:

```
C:\> java jess.Main C:\Programmi\Jess61p4\appoggio\es_2.clp
Jess, the Java Expert System Shell
Copyright (C) 2001 E.J. Friedman Hill and the Sandia Corporation
Jess Version 6.1p4 7/8/2003

Fred 2 orange
Joe  4 blue
Bob  1 plaid
Tom  3 red

Fred 2 orange
Joe  1 blue
Bob  4 plaid
Tom  3 red

Fred 1 orange
Joe  2 blue
Bob  4 plaid
Tom  3 red

Fred 3 orange
Joe  4 blue
Bob  1 plaid
Tom  2 red

Fred 1 orange
Joe  4 blue
Bob  3 plaid
Tom  2 red

Fred 1 orange
Joe  3 blue
Bob  4 plaid
Tom  2 red
Jess>
```

Caratteristiche generali

- Sono generate tutte le possibili soluzioni
- Il sistema degrada le proprie prestazioni in presenza di informazione incompleta; non fallisce

Caratteristiche generali

- Puzzles?
  - Regole di Business (costruzione di una proposta contrattuale, fatturazione,…)
  - Configurazione di prodotto
  - Interpretazione di dati e generazione di allarmi
  - ……..

Il linguaggio Jess: elementi di base

- Whitespace e Newline
  - non sono significativi se non tra " "
- Tokens: simboli, numeri, stringhe , commenti
- Simboli
  - Un simbolo contiene lettere, numeri e

    $ * . = + / < > _ ? #
  - Non può iniziare con un numero
  - Case sensitive
  - Nil FALSE TRUE

Il linguaggio Jess: elementi di base

- Numeri
  - Integer, Float, Long 2, -4, 3.1415, 6.0E4

    Float = double float Java

    Long = long integer Java
- Stringhe
  - **`"Hello, World"`**
  - `\"` Per inserire il simbolo ":

    `"\"Nonsense,\" he said firmly."`
  - no "escape sequences" (es: non si può inserire newline in una stringa usando "\n")
  - Newline inserito in una stringa è parte della stringa

    `"Hello,`

    `There"`

Il linguaggio Jess: elementi di base

- Commenti
  - Iniziano con **`";"`**
  - Fino alla fine della riga di testo
  - Possono apparire ovunque

```
; This is a list
(a b c)
```

Liste

- Liste (struttura di base di Jess)
  - Zero o più tokens (o liste) tra parentesi

Primo elemento:
list's head

```
(+ 3 2)
(a b c)
("Hello, World")
()
(deftemplate person (slot name))
```

Funzioni

- Funzioni
  - Tutto il codice in Jess (strutture di controllo, assegnamenti, chiamate a procedure) prende la forma di chiamata a funzione
  - Una chiamata a funzione è una lista
  - Se la testa della lista è il nome di una funzione esistente, la lista è una chiamata a funzione
  - Notazione prefissa
  - built-in functions (math, program control, string manipulations, access to Java APIs)

Funzioni

Nome funzione matematica predefinita

```
Jess> (+ 3  7)
10
Jess>

Jess> ( + ( * 25 4 ) (- 20 5 ) )
115
Jess>
```

Funzioni

Nome funzione i/o predefinita

```
Jess> (printout t "The answer is 42 !" crlf)
The answer is 42 !
Jess>
```

Console

Simbolo speciale: stampa newline

Valuta un file di codice Jess

```
Jess> (batch
  C:/Programmi/Jess61p4/examples/hello.clp)
Hello, world!
Jess>
```

Funzioni

Es_2a

```
(printout t Fred Joe Bob Tom crlf)
(printout t "Fred Joe Bob Tom" crlf)
(printout t Fred  " " Joe " " Bob " " Tom crlf)
(reset)
(run)
```

```
Jess> ( batch
appoggio/es_2a.clp )

FredJoeBobTom

Fred Joe Bob Tom

Fred Joe Bob Tom

Jess>
```

Simboli e stringhe

Funzioni

```
Jess> (bind ?x TRUE)
TRUE
Jess> (bind ?y FALSE)
FALSE
Jess> ?x
TRUE
Jess> ?y
FALSE
```

Il risultato della valutazione della funzione. Tutte le funzioni Jess restituiscono un risultato

Nome funzione logica predefinita

```
Jess> (and ?x ?y)
FALSE
Jess> (or ?x ?y)
TRUE
Jess> (and ?x not?y)
TRUE
Jess>
```

Variabili

- Variabili
  - Contenitori dotati di nome che contengono un valore
  - Untyped
  - Iniziano con "? " **?a** **?p1**
  - Non devono essere dichiarate. Sono create quando si assegna per la prima volta un valore

Variabili

– Assegnare un valore a una variabile:

```
Jess> (bind ?x "The value")
"The value"
Jess>
```

– Utilizzare:

```
Jess> (bind ?a1 7)
7
Jess> (+ ?a1 3)
10
Jess>
```

Variabili

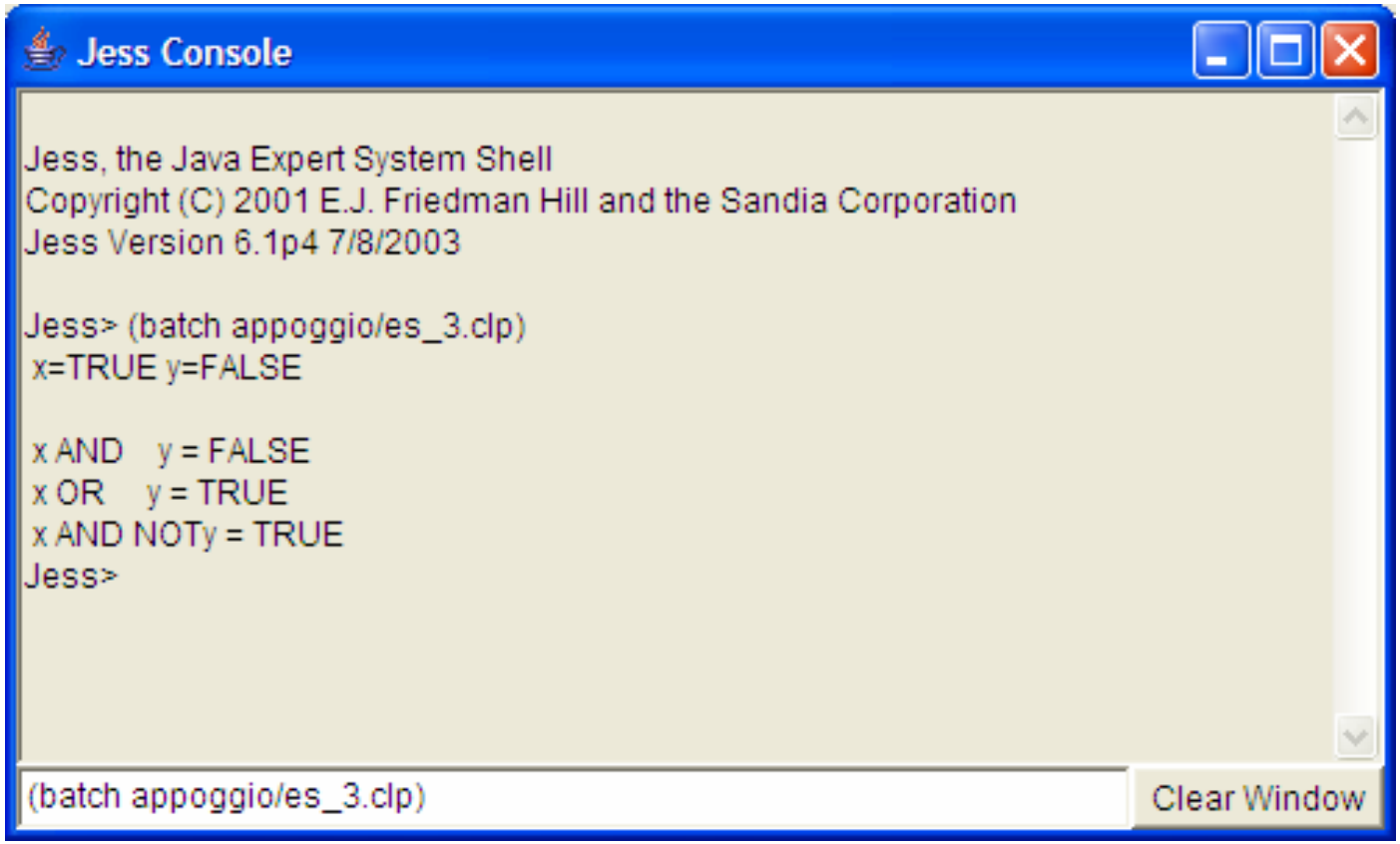
Es_3

```
;; Variabili, valori e funzioni predefinite

(bind ?x TRUE)
(bind ?y FALSE)
(printout t " x=" ?x " y=" ?y crlf crlf)

(bind ?res (and ?x ?y))
(printout t " x AND    y = " ?res crlf)

(bind ?res (or ?x ?y))
(printout t " x OR     y = " ?res crlf)

(bind ?res (and ?x not?y))
(printout t " x AND NOTy = " ?res crlf)
```

Variabili

Es_3

```
C:\Programmi\Jess61p4>java jess.Console
```



Variabili

- Variabili multifield

  `$?y`

  – Normali variabili con significato speciale solo nel contesto di

    - Deffunctions

      `(deffunction max ($?args)……`

      Numero variabile di argomenti della funzione

    - Parte LHS di regole

      (per eseguire il match di una variabile con più valori)

Variabili

- Variabili globali
- Ogni variabile è distrutta dal comando

  `(reset)`– inizializza la working memory
- Per creare variabili persistenti:

```
?*nome_var_globale*
?no-globale

(defglobal ?*x* = 3)
```

Variabili

```
(defglobal ?*x* = 3)
```

- Quando è creata è inizializzata al valore dato
- Dipendentemente dalla proprietà **`reset-globals`** è reinizializzata o no con il comando **`(reset)`**

Variabili

```
Jess> (get-reset-globals)
TRUE
Jess> (defglobal ?*x* = 3)
TRUE
Jess> ?*x*
3
Jess> (bind ?*x* 4)
4
Jess> (reset); reset ?*x* to its initial value of 3
TRUE
Jess> ?*x*
3
Jess> (bind ?*x* 4)
4
Jess> (set-reset-globals FALSE)
FALSE
Jess> (reset)
TRUE
Jess> ?*x*
4
Jess>
```

Funzioni di gestione di **reset-globals**

(default **TRUE**)

Funzioni e liste

- Jess assume che una lista è una chiamata a funzione
- Per costruire liste di dati *(plain list)* : **`create$`**

```
Jess> (bind ?grocery-list (create$ eggs bread milk))
(eggs bread milk)
Jess>
```

Funzioni e liste

```
Jess> (first$ ?grocery-list)
(eggs)
Jess>
Jess> (rest$ ?grocery-list)
(bread milk)
Jess>
Jess> (nth$  1 ?grocery-list)
eggs
Jess>(bind ?more-groceries (create$ ?grocery-
  list salt soap))
(eggs bread milk salt soap)
Jess>
```

La lista inizia da 1

Le liste non sono annidate (plain list)

- Strutture di controllo

Uno o più

(foreach <variable> <list> <expression>+)

valuta un'espressione per ogni elemento di una lista

```
Jess> (bind ?grocery-list (create$ eggs bread milk))
(eggs bread milk)
Jess> (foreach ?e ?grocery-list  (printout t ?e crlf))
eggs
bread
milk
Jess>
```

(foreach <variable> <list> <expression>+)

```
(foreach ?name (create$ Fred Joe Bob Tom)

   (foreach ?color (create$ red blue plaid orange)
      (assert (pants-color (of ?name)(is ?color))))
   (foreach ?position (create$ 1 2 3 4)
      (assert (position (of ?name)(is ?position))))
)
```

- Strutture di controllo

(while <Boolean expression> do <expression>+)

```
Jess> (batch appoggio/es_4.clp)
 somma=55
Jess>
```

```
;;
;; Somma i numeri da 1 a 10
;;
(bind ?i 1)
(bind ?sum 0)

(while (<= ?i 10) do
      (bind  ?sum  (+  ?sum  ?i))
      (bind  ?i  (+ ?i 1)))

(printout t " somma=" ?sum crlf)
```

Es_4

- Strutture di controllo

Es_5

(if <Boolean expression> then <expression>+ [else <expression>+])

opzionale

```
Jess> (batch appoggio/es_5.clp)
 I need to buy eggs
Jess>
```

E' un elemento della lista

```
;;
;; Esercizio IF
;;
(bind ?grocery-list (create$ eggs bread milk))

(if (member$ eggs ?grocery-list) then

      (printout t " I need to buy eggs"  crlf)
else
      (printout t " No eggs"  crlf))
```

Definire una funzione

- Definire una funzione

Zero o più

(deffunction <name> (<parameter>*) [<comment>]
<expression>*)

Es_6

```
Jess> ( batch appoggio/es_6.clp )
x? = 5
y? = 12
The greater of x and y is 12
Jess>
```

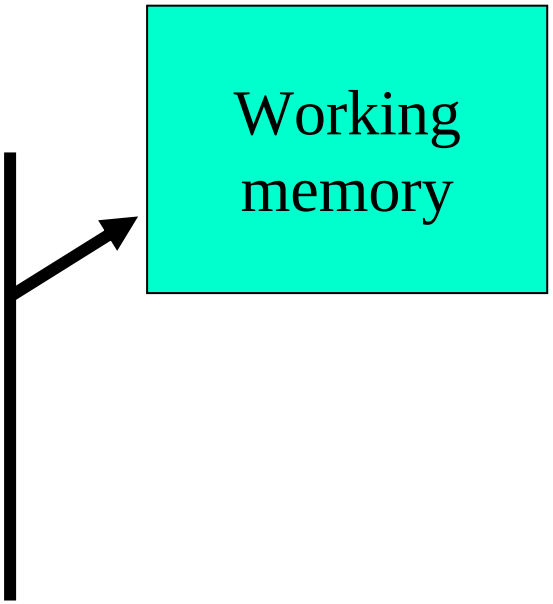
Definire una funzione

(deffunction <name> (<parameter>*) [<comment>] <expression>*)

```
;;
;; Esercizio deffunction
;;
(deffunction max (?a ?b)
    (if (> ?a ?b) then
        (return ?a)
    else
        (return ?b)))

(printout t "x? = " )
(bind ?x (read t))
(printout t "y? = " )
(bind ?y (read t))

(printout t "The greater of x and y is " (max ?x ?y) crlf)
```

- Fatti

```
(shopping-list  eggs  milk  bread)

(person  "Bob Smith"  Male  35)

(father-of  danielle  fred)

(temperature  floor1 18.5)
```



Fatti

- Definire un fatto (assert)

```
(assert (temperature  floor1 18.5))
```

- (facts) elenca tutti i fatti nella working memory

```
Jess> (assert (temperature  floor1 18.5))
<Fact-0>
Jess> (facts)
f-0   (MAIN::temperature floor1 18.5)
For a total of 1 facts.
Jess>
```

Fact Id

Ad ogni fatto è associato un numero identificativo univoco a partire da 0

Fatti

- (watch facts) visualizza i cambiamenti nella working memory

```
Jess> (assert (temperature  floor1 18.5))
   <Fact-0>
   Jess> (facts)
   f-0   (MAIN::temperature floor1 18.5)
   For a total of 1 facts.
   Jess>
 Jess> (watch facts)
 TRUE
 Jess> (assert (temperature  floor2  21))
  ==> f-1 (MAIN::temperature floor2 21)
 <Fact-1>
 Jess>
```

==>
Ho inserito un nuovo fatto

- Rimuovere un fatto (retract)

```
Jess> (facts)
f-0   (MAIN::temperature floor1 18.5)
f-1   (MAIN::temperature floor2 21)
For a total of 2 facts.
Jess> (retract 1)
 <== f-1 (MAIN::temperature floor2 21)
TRUE
Jess>
```

Fact id

<==
Ho eliminato un fatto

Fatti

- Rimuovere un fatto (retract)

```
Jess> (bind  ?id  (assert (temperature  floor3 30)) )
 ==> f-3 (MAIN::temperature floor3 30)
<Fact-3>
Jess> (facts)
f-0   (MAIN::temperature floor1 18.5)
f-2   (MAIN::temperature floor2 21)
f-3   (MAIN::temperature floor3 30)
For a total of 3 facts.
Jess> (retract ?id)
 <== f-3 (MAIN::temperature floor3 30)
TRUE
Jess> (facts)
f-0   (MAIN::temperature floor1 18.5)
f-2   (MAIN::temperature floor2 21)
For a total of 2 facts.

Jess>
```

La variabile contiene il Fact id

Fatti

- (clear)
  - Inizializza Jess
  - rimuove tutti i fatti dalla working memory
  - cancella variabili, regole, deffunctions
- (reset)
  - Elimina i fatti dalla working memory
  - Elimina le attivazioni dall'agenda
  - Asserisce il fatto iniziale **`(initial-fact)`**
  - Ricarica i **`deffacts`** (stato iniziale)
  - Inizializza le variabili globali se **`(set-reset-globals TRUE)`**

Fatti

- (run)
  - Avviamento del motore
- (exit)
  - fine

Fatti

- (deffacts) lista di fatti con nome

```
Jess> (deffacts catalog "Product catalog"
(product 344 mouse "$3")
(product 345 red-pen "$0.5")
(product 346 blue-pen "$0.6"))
TRUE
Jess> (reset)
 ==> f-0 (MAIN::initial-fact)
 ==> f-1 (MAIN::product 344 mouse "$3")
 ==> f-2 (MAIN::product 345 red-pen "$0.5")
 ==> f-3 (MAIN::product 346 blue-pen "$0.6")
TRUE
Jess> (facts)
f-0   (MAIN::initial-fact)
f-1   (MAIN::product 344 mouse "$3")
f-2   (MAIN::product 345 red-pen "$0.5")
f-3   (MAIN::product 346 blue-pen "$0.6")
For a total of 4 facts.
Jess>
```

I deffacts sono caricati nella working memory al comando (reset)

Fatti

- Unordered facts
  - Semplice lista `(person "Bob Smith" Male 35)`

    `(assert (person "Bob Smith" Male 35))`
- Ordered facts
  - Una riga di dati con colonne dotate di nome (slot)
  - **`deftemplate`** per definire la struttura del fatto
  - **`assert`** per caricare i singoli fatti della struttura definita

- Ordered facts

Es_7

es_7.clp

```
(deftemplate product "Product catalog"
(slot code)
(slot description)
(slot price))
(assert (product (code 345) (description red-pen) (price "$0.5")))
(assert (product (code 346) (description mouse) (price "$3")))
```

Testo descrittivo opzionale

```
Jess> ( batch appoggio/es_7.clp )
Jess> (facts)
f-0   (MAIN::product (code 345) (description red-pen)
       (price "$0.5"))
f-1   (MAIN::product (code 346) (description mouse) (price
       "$3"))
For a total of 2 facts.
Jess>
```

Fatti

- Default slots

es_8.clp

Es_8

```
(deftemplate product "Product catalog"
(slot code (default "to_be_defined"))
(slot description)
(slot price))

(assert (product (code 346) (description mouse) (price "$3")))
(assert (product (description red-pen) (price "$0.5")))
(assert (product (description blue-pen)))
```

```
Jess> ( batch appoggio/es_8.clp )
Jess> (facts)
f-0   (MAIN::product (code 346) (description mouse) (price "$3"))
f-1   (MAIN::product (code "to_be_defined") (description red-pen)
(price "$0.5"))
f-2   (MAIN::product (code "to_be_defined") (description blue-
pen) (price nil))
For a total of 3 facts.
Jess>
```

- Regole (defrule)

Rules KB

```
Jess>(defrule do-change-baby
     "If baby is wet, change baby's diaper."
     (baby-is-wet)
     =>
     (change-baby))
```

LHS *pattern*

RHS *action*

The LHS of a rule (the "if" part) consists of ***patterns*** *that match facts,* ***NOT*** *function calls.*
The actions of a rule (the "then" clause) are made up of function calls.

- Regole (defrule)

```
Jess> (defrule wrong-rule
      (eq 1 1)
      =>
      (printout t "Just as I thought, 1 == 1!" crlf))
```

LHS *pattern*

This rule ***will NOT fire*** *just because the function call (eq 1 1) would evaluate to true. Instead, Jess will try to find a fact on the knowledge base that looks like (eq 1 1). Unless you have previously asserted such a fact, this rule will* ***NOT*** *be activated and will not fire.*

Esempio completo

Es_9

```
;; Esercizio 1 regola

(deffunction change-baby ()
(printout t "Baby is now dry" crlf))

(defrule do-change-baby
      (baby-is-wet)
      =>
      (change-baby))

(assert (baby-is-wet))

(run)
```

```
Jess> (reset)
TRUE
Jess> (watch rules)
TRUE
Jess> (batch appoggio/es_9.clp)
FIRE 1 MAIN::do-change-baby f-0
Baby is now dry
1
Jess>
```

Accende la regola

Regole

Es_10

```
;; Esercizio parentela 1
;;
(assert (femmina Amelia))
(assert (femmina Alba))
(assert (madre Alba Paola))
(assert (madre Amelia Paola)(padre Alba Mario))
(assert (padre Amelia Mario))
(defrule sorella
   (padre ?x ?p)
   (padre ?y ?p)
   (madre ?x ?m)
   (madre ?y ?m)
   (femmina ?x)
  =>
       (assert (sorella ?x ?y))
      (printout t ?x " sorella di " ?y crlf)
  )
(run)
```

Regole

```
Jess> (reset)
TRUE
Jess> (facts)
f-0   (MAIN::initial-fact)
For a total of 1 facts.
Jess> (batch appoggio/es_10.clp)
Amelia sorella di Amelia
Alba sorella di Amelia
Amelia sorella di Alba
Alba sorella di Alba
4
……................
```

```
……..................
Jess> (facts)
f-0   (MAIN::initial-fact)
f-1   (MAIN::femmina Amelia)
f-2   (MAIN::femmina Alba)
f-3   (MAIN::madre Alba Paola)
f-4   (MAIN::madre Amelia Paola)
f-5   (MAIN::padre Alba Mario)
f-6   (MAIN::padre Amelia Mario)

f-7   (MAIN::sorella Amelia Amelia)
f-8   (MAIN::sorella Alba Amelia)
f-9   (MAIN::sorella Amelia Alba)
f-10  (MAIN::sorella Alba Alba)
For a total of 11 facts.
Jess>
```

Regole

Es_11

```
;; Esercizio parentela 2
;;
(assert (femmina Amelia))
(assert (femmina Alba))
(assert (madre Alba Paola))
(assert (madre Amelia Paola)(padre Alba Mario))
(assert (padre Amelia Mario))
(defrule sorella
   (padre ?x ?p)
   (padre ?y ?p)
   (madre ?x ?m)
   (madre ?y ?m)
   (femmina ?x)
   (not (test (eq ?x ?y)))
  =>
       (assert (sorella ?x ?y))
      (printout t ?x " sorella di " ?y crlf)
  )
(run)
```

Regole

```
Jess> (reset)
TRUE
Jess> (facts)
f-0   (MAIN::initial-fact)
For a total of 1 facts.
Jess> (batch appoggio/es_8.clp)
Alba sorella di Amelia
Amelia sorella di Alba
2
……
```

```
……
Jess> (facts)
f-0   (MAIN::initial-fact)
f-1   (MAIN::femmina Amelia)
f-2   (MAIN::femmina Alba)
f-3   (MAIN::madre Alba Paola)
f-4   (MAIN::madre Amelia Paola)
f-5   (MAIN::padre Alba Mario)
f-6   (MAIN::padre Amelia Mario)

f-7   (MAIN::sorella Alba Amelia)
f-8   (MAIN::sorella Amelia Alba)
For a total of 9 facts.
Jess>
```

Regole

- watch / unwatch
  - abilita / disabilita visualizzazione cambiamenti (fatti, attivazione regole,….)

  `(watch facts)` stampa messaggio ogni volta che un fatto è asserito o ritratto

  `(watch activations)` …regola attivata o deattivata

  `(watch rules)`… regola accesa

  ……

  `(watch all)`…

- Patterns
  - Si possono scrivere regole complesse di pattern matching

```
(deftemplate position (slot of) (slot is))

……................................


(position (of ?n&~Fred) (is ?p&:(eq ?p (+ ?p1
1))))
```

- Patterns

Es_12

  – *Literal constraints*. Specifica il valore di uno slot

```
(defrule literal-constraints
      (letters b c)
      =>
      )
```

```
Jess> (watch activations)
TRUE
Jess> (batch appoggio/es_12.clp)
TRUE
Jess> (assert (letters b d))
<Fact-0>
Jess> (assert (letters b c))
==>Activation: MAIN::literal-constraints:f-1
<Fact-1>
Jess> (run)
1
Jess>
```

- Patterns

Es_13

  – *Variable constraints.* Assegna valore matched a variabile

```
(defrule variables
     (a ?x ?y)
     =>
     (printout t "pattern matched : a " ?x " " ?y crlf )
      )
```

```
Jess> (batch appoggio/es_13.clp)
TRUE
Jess> (assert (b 2 4))
<Fact-1>
Jess> (run)
0
Jess> (assert (a 5 8))
==> Activation: MAIN::variables :  f-2
<Fact-2>
Jess> (run)
pattern matched : a 5 8
1
```

Patterns

- Note
  - La variabile matched in LHS è disponibile in RHS

```
(defrule variables
     (a ?x ?y)
     =>
     (printout t "pattern matched : a " ?x " " ?y crlf )
      )
```

  - Non si può utilizzare una variabile per eseguire il match dell'head di un fatto o del nome di uno slot

```
( a ?x ?y )
```

- Note

Es_14

  – La stessa variabile può apparire più volte nella stessa regola. Ogni volta che la variabile è utilizzata, deve eseguire il match con lo stesso valore

```
(defrule repeated-variables
      (a ?x)
      (b ?x)
      =>
      (printout t "x is : " ?x crlf )
       )
```

Patterns

```
Jess> (clear)
TRUE
Jess> (watch activations)
TRUE
Jess> (batch appoggio/es_14.clp)
TRUE
Jess> (deffacts repeated-variables (a 1) (a 2) (b 2) (b 3))
TRUE
Jess> (facts)
For a total of 0 facts.
Jess> (reset)
==> Activation: MAIN::repeated-variables :  f-2, f-3
TRUE
Jess> (facts)
f-0   (MAIN::initial-fact)
f-1   (MAIN::a 1)
f-2   (MAIN::a 2)
f-3   (MAIN::b 2)
f-4   (MAIN::b 3)
For a total of 5 facts.
```

I deffacts sono caricati nella working memory solo al comando (reset)

Solo la coppia di fatti (a 2) (b 2) attiva LHS (?x è lo stesso) anche se es. (a 1) attiva un pattern e (b 3) l'altro separatamente

```
Jess> (run)
x is : 2
1
Jess>
```

- Blank variables

Es_15

  - Esegue il match senza specificare il nome di variabile

```
(defrule blank-variables
      (letters b ? ? k)
      =>
      (printout t "OK" crlf ))
```

```
Jess> (reset)
TRUE
Jess> (batch appoggio/es_15.clp)
TRUE
Jess> (assert (letters b gg fff k))
<Fact-1>
Jess> (run)
OK
1
```

```
Jess> (reset)
TRUE
Jess> (batch appoggio/es_15.clp)
TRUE
Jess> (assert (letters b gg fff j))
<Fact-1>
Jess> (run)
0
Jess>
```

- Patterns

Es_16

  – *Connective constraints*. Condizioni and, or, not

```
(defrule example-3
    (code ?n1&~special-client)
    (items-purchased ?d1 ?d2&~?d1)
    (city Milan|Paris)
    =>
    (printout t "Order accepted" crlf))

(assert (code normal-client))
(assert (items-purchased dvd mouse))
(assert (city Milan))

(run)
```

```
Jess>  ( batch appoggio/es_16.clp )
Order accepted
1
Jess>
```

Patterns

- Note
  - Ordine di precedenza degli operatori: **~ & |**
  - **~** è applicato al vincolo immediatamente successivo
  - Non si possono usare parentesi per cambiare la precedenza

- Patterns

Es_17

  – *Predicate constraints*. Chiama una funzione per testare il match

```
(defrule example-3
    (code ?n1&~special-client)
    (items-purchased ?d1 ?d2&~?d1)
    (city Milan|Paris)
    (total-amount ?m&:(and (> ?m 100) (< ?m 1000)))
    =>
    (printout t "Order accepted" crlf))
(assert (code normal-client))
(assert (items-purchased dvd mouse))
(assert (city Milan))
(assert (total-amount 120))
(run)
```

```
Jess> ( batch appoggio/es_17.clp )
Order accepted
1
Jess>
```

Patterns

- Pattern bindings
  - Per utilizzare retract, modify, duplicate di fatti matched in LHS bisogna passare un puntatore agli stessi alla parte RHS
  - Si utilizza una “pattern binding variable”

Patterns

Es_18

```
(deffunction change-baby () (printout t "Baby is now dry" crlf))

(defrule change-baby-if-wet
      ?wet <- (baby-is-wet)
      =>
      (change-baby)
       (retract ?wet))
```

pattern binding variable

Quando la regola è accesa la variabile è caricata con un riferimento al fatto matched

```
Jess> (reset)
TRUE
Jess> (assert (baby-is-wet))
<Fact-1>
Jess> (facts)
f-0   (MAIN::initial-fact)
f-1   (MAIN::baby-is-wet)
For a total of 2 facts.
Jess> ( batch appoggio/es_18.clp )
TRUE
Jess> (run)
Baby is now dry
1
Jess> (facts)
f-0   (MAIN::initial-fact)
For a total of 1 facts.
Jess>
```

Conditional Elements (CE)

- Patterns
  - regole complesse di pattern matching relative a fatti individuali

- Conditional Elements (CE)
  - Regole che esprimono relazioni tra fatti

- Conditional Elements (CE)
  - Regole che esprimono relazioni tra fatti

```
(defrule example-3
    (code ?n1&~special-client)
    (city Milan|Paris)
    =>
    (printout t "Order accepted" crlf))
```

```
(defrule example-3
    (and (code ?n1&~special-client)
         (city Milan|Paris))
    =>
    (printout t "Order accepted" crlf))
```

Conditional Elements (CE)

- Le due regole sono equivalenti (and implicito)
- Ma si può usare anche or, not

```
(defrule example-3
    (code ?n1&~special-client)
    (city Milan|Paris)
    =>
    (printout t "Order accepted" crlf))
```

```
(defrule example-3
    (and (code ?n1&~special-client)
         (city Milan|Paris))
    =>
    (printout t "Order accepted" crlf))
```

Conditional Elements (CE)

Es_19

- OR

```
(deftemplate used-car (slot price) (slot mileage))
(deftemplate new-car (slot price) (slot warrantyPeriod))

(defrule might-buy-car
      ?candidate <- (or (used-car (mileage ?m&: (< ?m 50000)))
                        (new-car (price ?p&: (< ?p 20000))))
      =>
      (assert (candidate ?candidate)))

(assert (new-car (price 18000)))
(assert (used-car (mileage 30000)))

(run)
(facts)
```

- OR

```
Jess> (reset)
TRUE
Jess> ( batch appoggio/es_19.clp )
f-0   (MAIN::initial-fact)
f-1   (MAIN::new-car (price 18000) (warrantyPeriod nil))
f-2   (MAIN::used-car (price nil) (mileage 30000))
f-3   (MAIN::candidate <Fact-2>)
f-4   (MAIN::candidate <Fact-1>)
For a total of 5 facts.
Jess>
```

La regola è istanziata e accesa due volte, una per new-car e una per used-car che attivano ognuno un ramo dell'OR

Conditional Elements (CE)

- NOT

```
(defrule no-red-cars-1
      (auto (color ~red))
=>
)
```

La regola è accesa
quando un'auto non è di colore rosso
(si accende per ogni auto blu -non rossa-)
(non si accende quando non ci sono auto)

```
(defrule no-red-cars-2
      (not (auto (color red)))
=>
)
```

La regola è accesa
quando non ci sono fatti "auto di colore rosso"
(non si accende quando c'è un'auto di colore rosso)
(si accende quando non ci sono auto)
(si accende se ci sono solo auto blu)

- TEST
  - Un pattern con head **`test`** ha come body una funzione booleana il cui risultato determina se il pattern esegue il match

```
(deftemplate person (slot age))

(defrule test
     (person (age ?x))
     (test (> ?x 30))
     =>
     (printout t " Person age is over 30!" crlf))

(assert (person (age 45)))
(run)
```

```
Jess> (reset)
TRUE
Jess> ( batch
appoggio/es_20.clp )
 Person age is over 30!
1
Jess>
```

Es_20

Strategia

- Motore forward chaining
- Supporta anche l’utilizzo backward chaining

Gestione dell'agenda

- Conflict resolution
  - **`(set-strategy depth)`** default

    La regola attivata più di recente è attivata per prima

  - **`(set-strategy breadth)`**

    La regola attivata più di recente è attivata per ultima

  - Priorità

    Ogni regola ha la proprietà **`salience`**

Gestione dell'agenda

- **`salience`**
  - Le regole con salience maggiore si accendono per prime
  - Default salience: 0

```
(defrule example
(declare (salience +100))
……................
```

Moduli

- E' possibile separare la base di regole e fatti in moduli diversi

```
Jess> (defmodule WORK)
TRUE
Jess> (deftemplate WORK::job (slot salary))
TRUE
Jess> (list-deftemplates WORK)
WORK::job For a total of 1 deftemplates.
```

- Un modulo definisce un *namespace*
- Esiste un meccanismo per passare il flusso di controllo tra moduli: *focus*