

# Intelligenza Artificiale I

## Esercitazione 1

Marco Piastra

# Jess?

Acronimo di *Java Expert System Shell*

- Sistema scritto in Java

Autore: Ernest Friedman-Hill, Sandia National Laboratories in Livermore, Canada

Un sistema a regole: il costrutto fondamentale è costituito da *regole*

*if <cond> then <action>*

Deriva da un sistema pre-esistente, detto CLIPS

Utilizza la sintassi del linguaggio LISP

Un programma in Jess consiste in un insieme di regole da applicarsi iterativamente ad una collezione di fatti

Caratteristiche salienti:

- Linguaggio (quasi) dichiarativo: i programmi non descrivono uno schema di flusso
- Regole, per rappresentare le conoscenze in modo esplicito
- Funzioni, per esprimere conoscenze procedurali
- Linguaggio (quasi) ad oggetti:  
si usano fatti compositi, organizzati come una collezione di slot

# Java: uso della variabile CLASSPATH

Non sapete cos'è?

E' normale: vuol dire che non avete mai usato Java

## Impostazione della variabile CLASSPATH

```
$ CLASSPATH=<value>      $ è il prompt, non scrivetelo
$ export CLASSPATH
per verificare
$ echo $CLASSPATH
```

Nel nostro caso:

```
$ CLASSPATH=/home/opt/Jess61p8
$ export CLASSPATH
```

Rispettare SEMPRE la differenza tra maiuscolo e minuscolo! (è Linux, non Windows)

Attivazione di Jess

```
$ java jess.Main
Jess>
```

Uscita da Jess

```
Jess> (exit)
$
```

(Coraggio, ce l'avete fatta.)

# Liste

(file finitestatemachine.jess)

- Si usa la sintassi del LISP

Semplicissima, talvolta diabolica: l'unica struttura dati è la lista.

*Lots of Impossible Stupid Parenthesis*

*Esempio:*

```
(deftemplate fsm
  (slot current-state)
  (multislot input-stream)
  (multislot output-stream)
)
```

Una lista che contiene altre liste (obbligatorio il bilanciamento delle parentesi)

*Esempio:*

```
(exit)
```

Una lista semplice. E' anche un comando Jess.

- *Qualsiasi espressione corretta in Jess è una lista*

# deftemplate

(file finitestatemachine.jess)

- Definisce uno schema (o *template*) di fatti (strutture dati di Jess)

*Esempio:*

```
(deftemplate event
  (slot current-state)
  (slot input-symbol)
  (slot output-symbol)
  (slot new-state)
)
```

Serve a definire fatti del tipo:

```
(event
  (current-state even)
  (input-symbol 1)
  (output-symbol 1)
  (new-state odd)
)
```

Un particolare fatto in Jess, tutti gli slot hanno un valore

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Significato: un mondo intero, praticamente tutto Jess.

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

Una lista, come ci si poteva aspettare

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
             (input-stream ?rest)
             (output-stream ?os ?output))
)
```

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

Nome della regola, serve solo come identificativo

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                 (input-stream ?is $?rest)
                 (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
             (input-stream ?rest)
             (output-stream ?os ?output))
)
```



# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
             " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
            (input-stream ?rest)
            (output-stream ?os ?output))

)
```

Separatore:  
cercatelo  
come prima  
cosa

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
             " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
             (input-stream ?rest)
             (output-stream ?os ?output))
)
```

Parte sinistra  
(LHS - Left Hand Side):  
sono le condizioni  
che stabiliscono se  
la regola è applicabile

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
             (input-stream ?rest)
             (output-stream ?os ?output))
)
```

La parte sinistra descrive una sorta di *pattern* che si applica ai fatti noti

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                 (input-stream ?is $?rest)
                 (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
             (input-stream ?rest)
             (output-stream ?os ?output))
)
```

Parte destra  
(*RHS - Right Hand Side*):  
sono le azioni  
da effettuare quando  
la regola viene  
applicata (*FIRE*)

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                 (input-stream ?is $?rest)
                 (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Si possono asserire,  
ritrattare o  
modificare fatti  
Si possono eseguire  
altre azioni (p.es.  
stampa di un  
messaggio)

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Una variabile

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                 (input-stream ?is $?rest)
                 (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
            (input-stream ?rest)
            (output-stream ?os ?output))
)
```

Si distingue (da un simbolo) per il punto interrogativo

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                 (input-stream ?is $?rest)
                 (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Si lega (*binding*) ad un valore singolo



# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

Oppure ad un fatto (dipende dalla posizione)

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
            (input-stream ?rest)
            (output-stream ?os ?output))
)
```

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Una multivariabile  
(\$ + ?)

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Si lega ad una sequenza di valori (una lista)

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Diverse occorrenze  
della stessa variabile:  
si legano allo stesso  
valore (*binding*)

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Una condizione, nella LHS della regola

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                 (input-stream ?is $?rest)
                 (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
            (input-stream ?rest)
            (output-stream ?os ?output))
)
```

Se la regola è applicabile, ciascuna condizione si lega ad un fatto (non necessariamente diverso)

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
            (input-stream ?rest)
            (output-stream ?os ?output))
)
```

Ciascuna condizione descrive una sorta di pattern, che si applica ad un fatto singolo

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                 (input-stream ?is $?rest)
                 (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
             (input-stream ?rest)
             (output-stream ?os ?output))
)
```

Anche questa condizione descrive un *pattern*, che si applica ad un fatto avente un *template* diverso



# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))
  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))
  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Le variabili nella LHS definiscono vincoli: in questo caso i due fatti che rendono la regola applicabile devono avere valori identici

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
              (input-stream ?rest)
              (output-stream ?os ?output))
)
```

Anche questi due valori  
devono essere identici

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
            (input-stream ?rest)
            (output-stream ?os ?output))
)
```

Questa variabile invece  
si lega al fatto che si lega  
al *pattern*

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                 (input-stream ?is $?rest)
                 (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
            (input-stream ?rest)
            (output-stream ?os ?output))
)
```

Il valore viene usato  
nella RHS (azioni):  
si modifica il fatto stesso

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
            (input-stream ?rest)
            (output-stream ?os ?output))
)
```

Usando il valore di altre variabili

# defrule

(file finitestatemachine.jess)

- Definisce una regola

*Esempio:*

```
(defrule state-transition
  ?current <- (fsm (current-state ?cs)
                  (input-stream ?is $?rest)
                  (output-stream $?output))

  (event (current-state ?cs)
         (input-symbol ?is)
         (output-symbol ?os)
         (new-state ?ns))

  =>
  (printout t "From state " ?cs " input " ?is
            " to state " ?ns " output " ?os crlf)
  (modify ?current (current-state ?ns)
             (input-stream ?rest)
             (output-stream ?os ?output))
)
```

In generale, le variabili  
si legano nella LHS  
e vengono usate nella RHS

# deffacts

(file paritychecker-fsa.jess)

- Definisce i fatti iniziali

Facendo un (**reset**) si cancellano i fatti noti e si torna ai fatti definiti con **deffacts**

*Esempio:*

```
(deffacts test-string
  (fsm (current-state even)
    (input-stream 0 1 1 1 0 0 1 1)
    (output-stream))
)
```

# deffacts

(file paritychecker-fsa.jess)

- Definisce i fatti iniziali

Facendo un **(reset)** si cancellano i fatti noti e si torna ai fatti definiti con **deffacts**

*Esempio:*

```
(deffacts test-string  
  (fsm (current-state even)  
       (input-stream 0 1 1 1 0 0 1 1)  
       (output-stream))  
)
```

Si definisce un fatto, singolo in questo caso, come iniziale



# deffacts

(file paritychecker-fsa.jess)

- Definisce i fatti iniziali

Facendo un (**reset**) si cancellano i fatti noti e si torna ai fatti definiti con **deffacts**

*Esempio:*

```
(deffacts test-string  
  (fsm (current-state even)  
       (input-stream 0 1 1 1 0 0 1 1)  
       (output-stream))  
)
```

Si definisce un fatto, singolo in questo caso, come iniziale. Possono essere definiti più fatti con un singolo **deffacts**

# deffacts

(file paritychecker-fsa.jess)

- Definisce i fatti iniziali

Facendo un (**reset**) si cancellano i fatti noti e si torna ai fatti definiti con **deffacts**

*Esempio:*

```
(deffacts test-string  
  (fsm (current-state even)  
        (input-stream 0 1 1 1 0 0 1 1)  
        (output-stream))  
)
```

Si definisce un fatto, singolo in questo caso, come iniziale. Possono essere definiti più fatti con un singolo **deffacts**.

Possono esserci più **deffacts** nello stesso programma.

# deffacts

(file paritychecker-fsa.jess)

- Definisce i fatti iniziali

Facendo un **(reset)** si cancellano i fatti noti e si torna ai fatti definiti con **deffacts**

*Esempio:*

```
(deffacts test-string  
  (fsm (current-state even)  
       (input-stream 0 1 1 1 0 0 1 1)  
       (output-stream))  
)
```

Si definisce un fatto, singolo in questo caso, come iniziale. Possono essere definiti più fatti con un singolo **deffacts**.

Possono esserci più **deffacts** nello stesso programma.

Tutti i fatti iniziali vengono trattati nello stesso modo: vengono asseriti eseguendo un **(reset)**.

# Caricare un programma

Prima cosa, far partire il Jess (è meglio)

```
$ java jess.Main  
Jess>
```

- Caricamento di un file

```
(batch paritychecker-fsa.jess)
```

Oppure

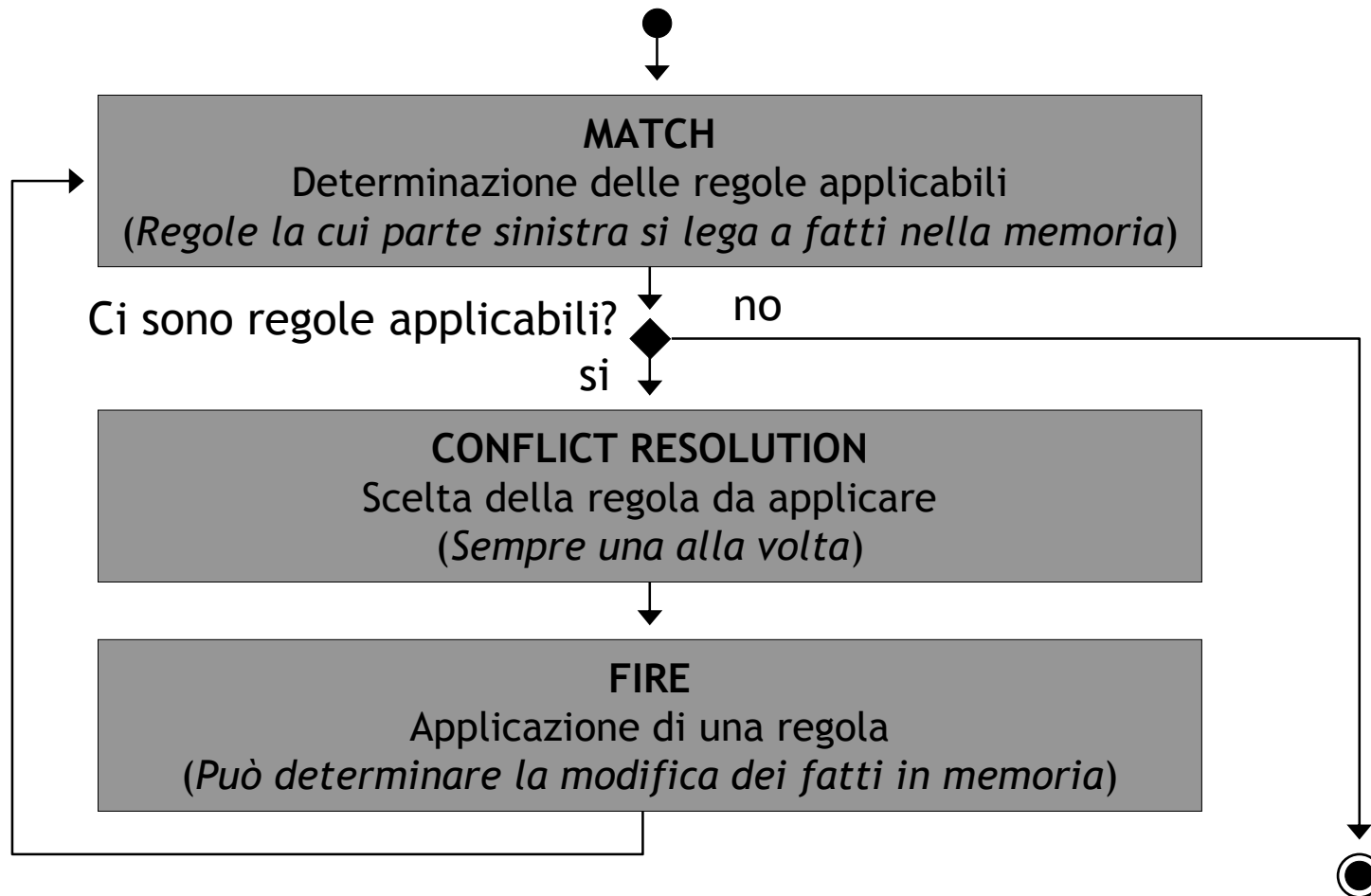
```
(batch "paritychecker-fsa.jess")
```

Se vi è andata bene, Jess risponde:

```
TRUE
```

# Come funziona Jess (prima approssimazione)

- Una volta attivato, Jess esegue un ciclo



# Comandi di attivazione

- **Reset:** cancella i fatti in memoria e asserisce nuovamente i fatti iniziali

I fatti iniziali sono definiti con **deffacts**

Eseguire sempre un reset prima di una nuova attivazione, non si sa mai

**(reset)**

Jess risponde:

**TRUE**

- **Run:** attiva il ciclo principale

**(run)**

Jess risponde:

<dipende dal programma>

**TRUE**

# Comandi utili per capire cosa accade in un programma

- Esegui un ciclo alla volta  
(**run 1**)  
Oppure (**run n**) per eseguire  $n$  cicli
- Elenca i fatti in memoria  
(**facts**)  
La risposta è un po' criptica, ma si capisce
- Elenca le regole applicabili (*MATCHed*)  
(**agenda**)  
La risposta è criptica, si capirà meglio più avanti