Intelligenza Artificiale
Complementi ed Esercizi

Rule-Based Systems

A.A. 2008-2009

Rule-Based Systems

- Applicazione ciclica di regole che operano su una memoria di lavoro contenente i fatti al fine di derivare fatti nuovi da fatti noti
- Un sistema di regole contiene un insieme di regole che specificano delle condizioni di attivazione e degli effetti
- In particolare, ciascina regola ha la forma
  - **<condizioni> => <azioni>**

Regole (1/2)

- **<LHS - Left Hand Side> => <RHS - Right Hand Side>**
  - Esempi di regole:

madre(x)=madre(y)
padre(x)=padre(y)
maschio(x)
=>
fratello(x,y)

madre(x)=madre(y)
padre(x)=padre(y)
femmina(x)
=>
sorella(x,y)

Regole (2/2)

- Nel codice si manifestano come flusso condizionale:

```
if(condizioneA) then flussoA;
else if (condizioneB) then flussoB;
else flussoC;
```

- Oppure come delega polimorfica

```
Class clazz=Class.forName("it.jugmi.Clazz");
((MyObject)clazz.newInstance()).doSomething();
```

Polimorfismo e regole

- Il paradigma OO ci permette di rendere regole modificabili per estensione

```
// Command
String command=request.getParamter("action");
Class commandClass=Class.forName(command);
Command command=(Command)commandClass.newIstance();
command.execute();

// Strategy
String strategyClass=System.getProperty("default.strategy");
Class strategyClass=Class.forName(strategyClass);
Strategy strategy=(Strategy)strategyClass.newInstance();
strategy.print(System.out, xml);
```

- Comportamento = regola

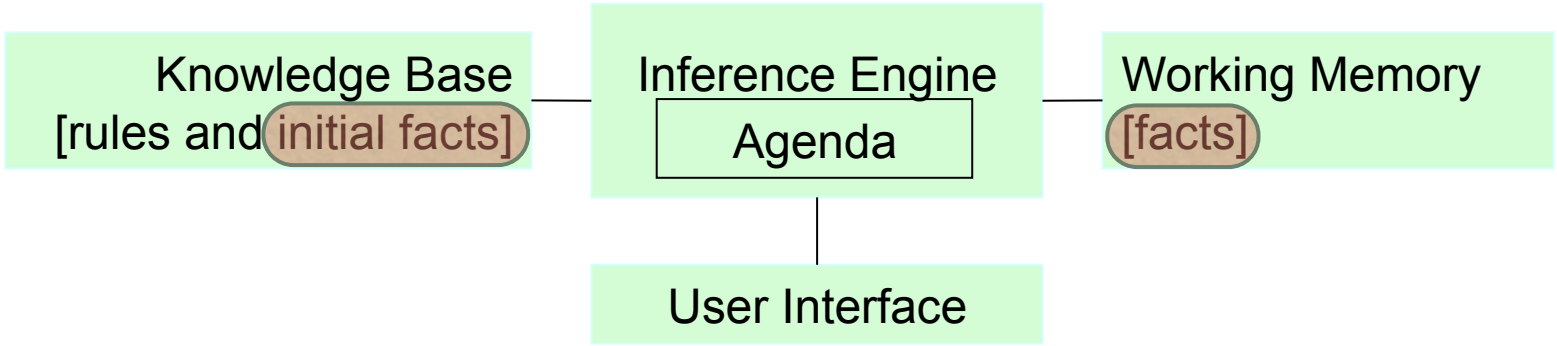
Implementare le regole

- **Staticamente**:
  - implementate nel codice, parametriche o meno, polimorfiche o meno
- **Dinamicamente**:
  - vengono lette a runtime e possono essere sostituite a caldo cancellandone o creandone di nuove:
    - Rules Engine
    - Sistemi Esperti

Apprendimento

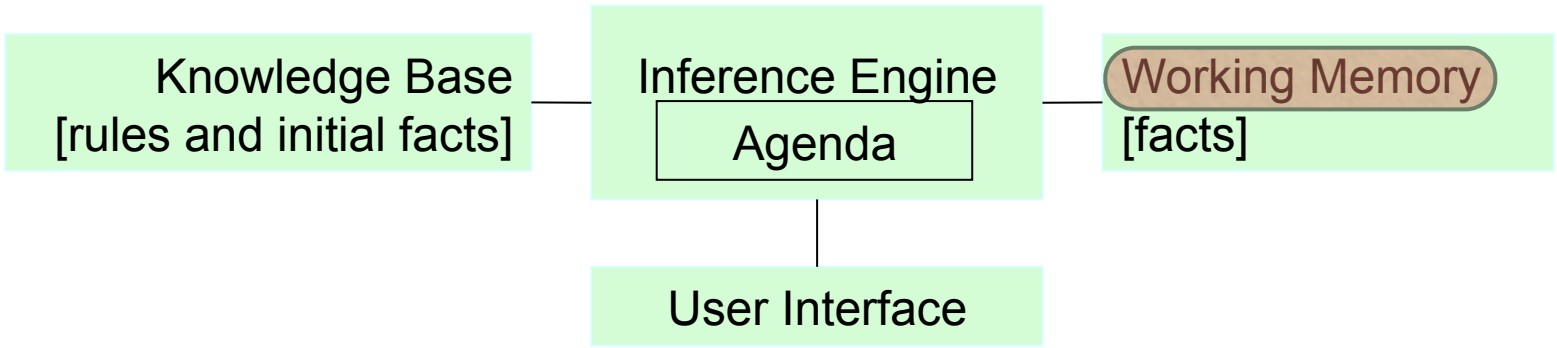
- Regole modificano automaticamente la Knowledge Base a partire da:
  - Gli attuali risultati delle regole
  - I risultati attuali confrontati con quelli precedenti
- Nuove regole possono essere create o eliminate

Architettura generale (1/4)



- I **fatti** sono asserzioni su proprietà, relazioni, proposizioni etc. nella forma di stringhe.
- Servono a descrivere gli stati della computazione, compreso lo **stato iniziale**
- Sono statici e inattivi rispetto al valore pragmatico e all'utilizzo dinamico della conoscenza che contengono

Architettura generale (2/4)

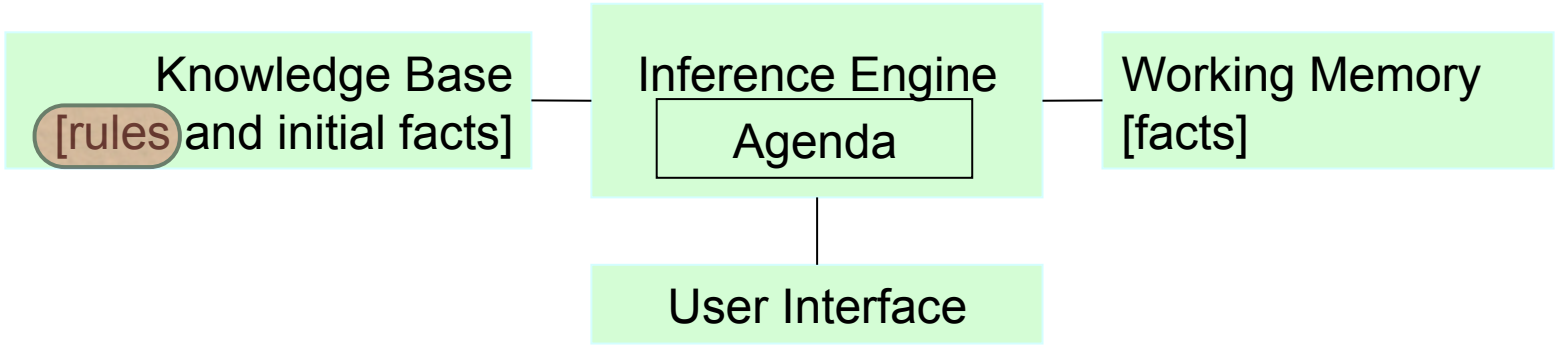


- La **Working Memory** (Memoria di Lavoro) è un dispositivo utilizzato per trattenere asserzioni temporanee (informazioni sullo stato del problema da risolvere)

Memoria di lavoro

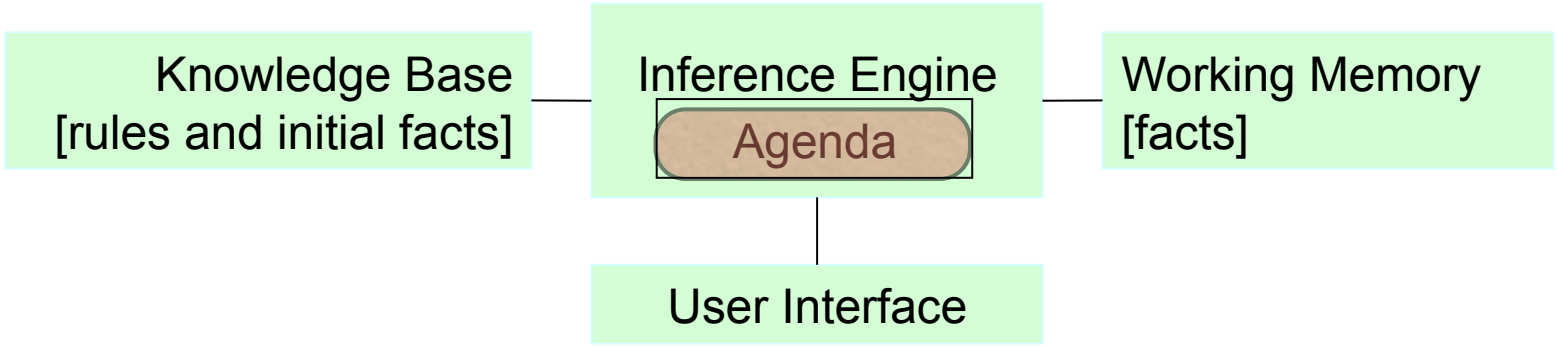
Femmina(Amelia)
Femmina(Alba)
(madre(Amelia) = Paola)
(madre(Alba) = Paola)
(padre(Amelia) = Mario)
(padre(Alba) = Mario)

Architettura generale (3/4)



- Le **regole** corrispondono a dati strutturati in forma IF-THEN impiegati dal motore inferenziale per inferire una soluzione al problema proposto.
- Si possono vedere come le unità base dell'inferenza che porta dalla descrizione dello stato iniziale alla soluzione del problema.

Architettura generale (4/4)



- L'**Agenda** è un dispositivo utilizzato per trattenere l'elenco delle regole attivate dai fatti presenti attualmente nella memoria di lavoro. Il suo contenuto varia dinamicamente durante l'esecuzione (Firing) delle regole stesse.

Regole, definizioni

- Una regola si dice **Attivata** se la condizione espressa è verificata.
- Una regola si dice **Fired** se le operazioni in essa indicate sono state eseguite.
- Un sistema **termina la computazione** quando non si hanno più regole attive in agenda.

Memoria di lavoro

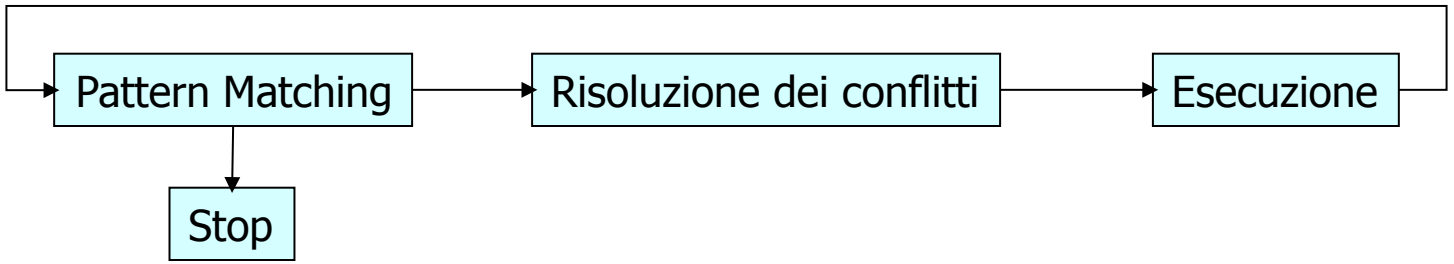
Femmina(Amelia)
Femmina(Alba)
(madre(Amelia) = Paola)
(madre(Alba) = Paola)
(padre(Amelia) = Mario)
(padre(Alba) = Mario)

Regola

madre(*x*) = madre(*y*)
padre(*x*) = padre(*y*)
Femmina(*x*)
=>
Sorella(*x*, *y*)

Interprete delle regole (1/4)

- L'interprete delle regole segue iterativamente i seguenti passi:
  - Pattern Matching
  - Risoluzione dei conflitti
  - Esecuzione



Interprete delle regole (2/4)

- **Pattern Matching** [Costruzione dell'agenda]:
  1. L'inteprete considera ciascuna regola rispetto alla situazione corrente
  2. Compara i valori che compaiono nella LHS con gli elementi di memoria corrispondenti per identificare quelle per le quali sono soddisfatte le LHS

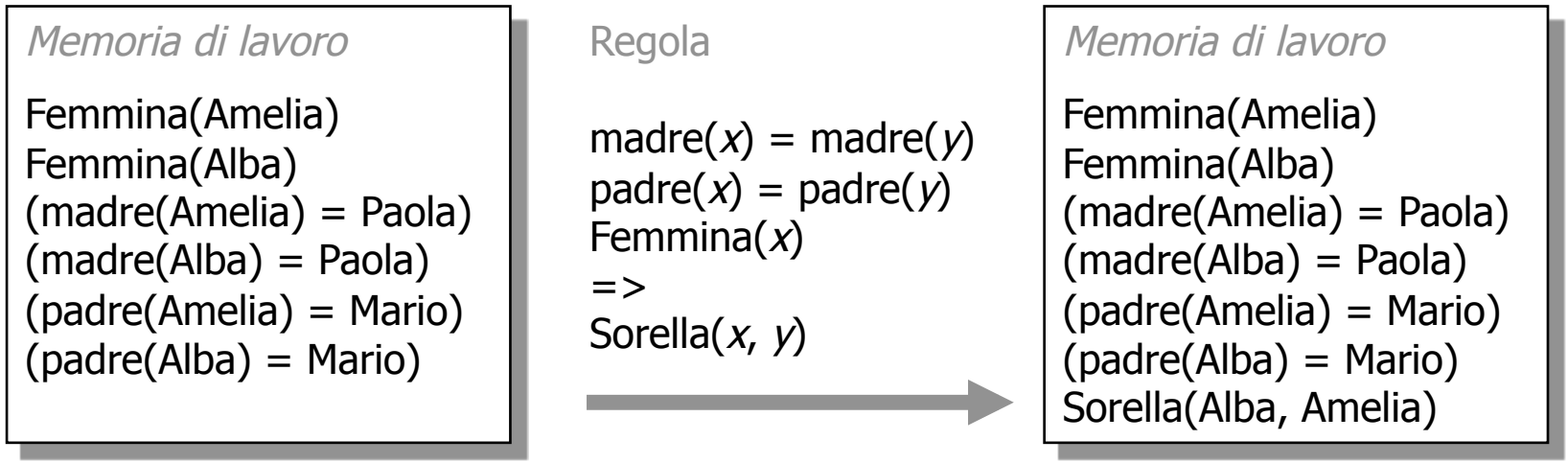
Algoritmo RETE*"Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem", Charles L. Forgy, Artificial Intelligence 19 (1982), 17-37."*

Interprete delle regole (3/4)

- **Risoluzione dei conflitti**: se vengono individuate più corrispondenze, l'interprete le immette nel CONFLICT SET e le tiene in considerazione come possibili candidati per l'esecuzione.
- Il processo di RISOLUZIONE DEI CONFLITTI selezione l'ESEMPLARE DOMINANTE di regola basandosi su una strategia di risoluzione dei conflitti, ad esempio dando precendenza a:
  - Regole più specifiche (piu pattern in LHS)
  - Regole più generiche (meno fatti nella LHS)

Interprete delle regole (4/4)

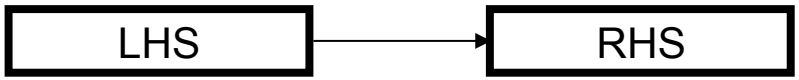
- Esecuzione [Firing delle regole]: esegue le azioni descritte dal membro di destra dell'ESEMPLARE DOMINANTE selezionato dal CONFRONTO (o eventualmente dal RISOLUTORE DEI CONFLITTI).



Rete

- Il motore di regole esegue la scansione della KB
- Se una parte dei fatti è stabile, creare una rete di nodi per cui non sia più necessario valutarli
- Ricalcolare la rete solo per certi tipi di eventi
- Charles L. Forgy Presentò il suo lavoro nel 1982 e da qui una serie di expert shell: OPS5, Arts, CLIPS, Jess

Esempi di regole IF-THEN



Situazione	Azione	if temp > 300° then stop boiler
Premessa	Conclusione	if stain is grapos then organism is strep
Antecedente	Conseguente	if x is a dog then x is an animal

Left-Hand Side (LHS)

- Ogni LHS contiene Elementi condizione: condizioni che devono essere verificate affinché la regola sia applicata
- Ogni LHS è una Clausola logica e condizione sufficiente per eseguire l'azione indicata nella RHS
- Queste condizioni vengono assegnate descrivendo delle configurazioni (di solito parti) della memoria globale: vengono specificati gli identificatori degli elementi di memoria insieme agli attributi ad essi associati e ai valori richiesti nella condizione

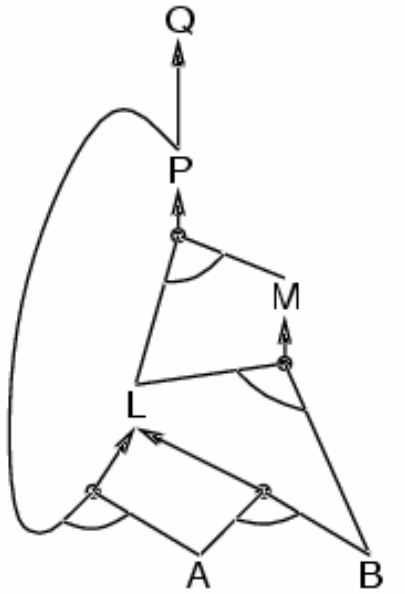
Right-Hand Side (RHS)

- Ogni Right-Hand Side (RHS) contiene la descrizione di Azioni: azioni che bisogna condurre ad effetto qualdo la regola viene eseguita.
- Le azioni possibili includono attività come:
  - inserire nuove descrizioni di stato nella memoria globale
  - modificare descrizioni di stato già esistenti
  - eseguire azioni definite dall'utente per una produzione particolare

Risoluzione Forward Chaining

- Applica tutte le regole le cui premesse sono soddisfatte nella KB
  - Vengono aggiunte le conclusioni alla KB, fino a trovare la query

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Risoluzione Backward Chaining

- Idea: lavora backwards dalla query (goal) q:
  - Per dimostrare q
    - controlla se q è già noto, oppure,
    - dimostra attravero BC tutte le premesse di una qualche regola che conclude q
  - Per evitare loops: controlla se i nuovi sottogoal sono già nello stack dei goal
  - Evita di ripetere lavoro controllando si i nuovi sottogoal sono già stati dimostrati veri o falsi.

Esercizio

- Dimostrare tramite forward e backward chaining: A(H1,H2,H3)

$$D(x) \wedge D(y) \wedge C(x,y) \Rightarrow B(x,y)$$
$$B(x,y) \wedge B(y,z) \Rightarrow A(x,y,z)$$
$$F(x,y) \wedge F(y,z) \Rightarrow E(x,z)$$
$$D(x) \wedge E(x,y) \Rightarrow B(x,y)$$
$$D(H1)$$
$$D(H2)$$
$$C(H1,H2)$$
$$C(H2,H3)$$
$$F(H4,H3)$$
$$F(H2,H4)$$

JESS
Java Expert System Shell

- Shell per sistemi di regole implementato in Java
- Disponibile presso: www.jessrules.com
- Closed source, esiste licenza di ricerca
- Forward Reasoning
- Java, jess 100% o ibrido
- Linguaggio proprietario
- Aderente alle specifiche JSR-94

Esempio approccio Java

```
Rete r = new Rete();

Fact f = new Fact("letters", r);
ValueVector vv = new ValueVector();
vv.add(new Value("a", RU.ATOM));
vv.add(new Value("b", RU.ATOM));
vv.add(new Value("c", RU.ATOM));
f.setSlotValue("__data", new Value(vv, RU.LIST));
r.assertFact(f);
r.executeCommand("(facts)");
```

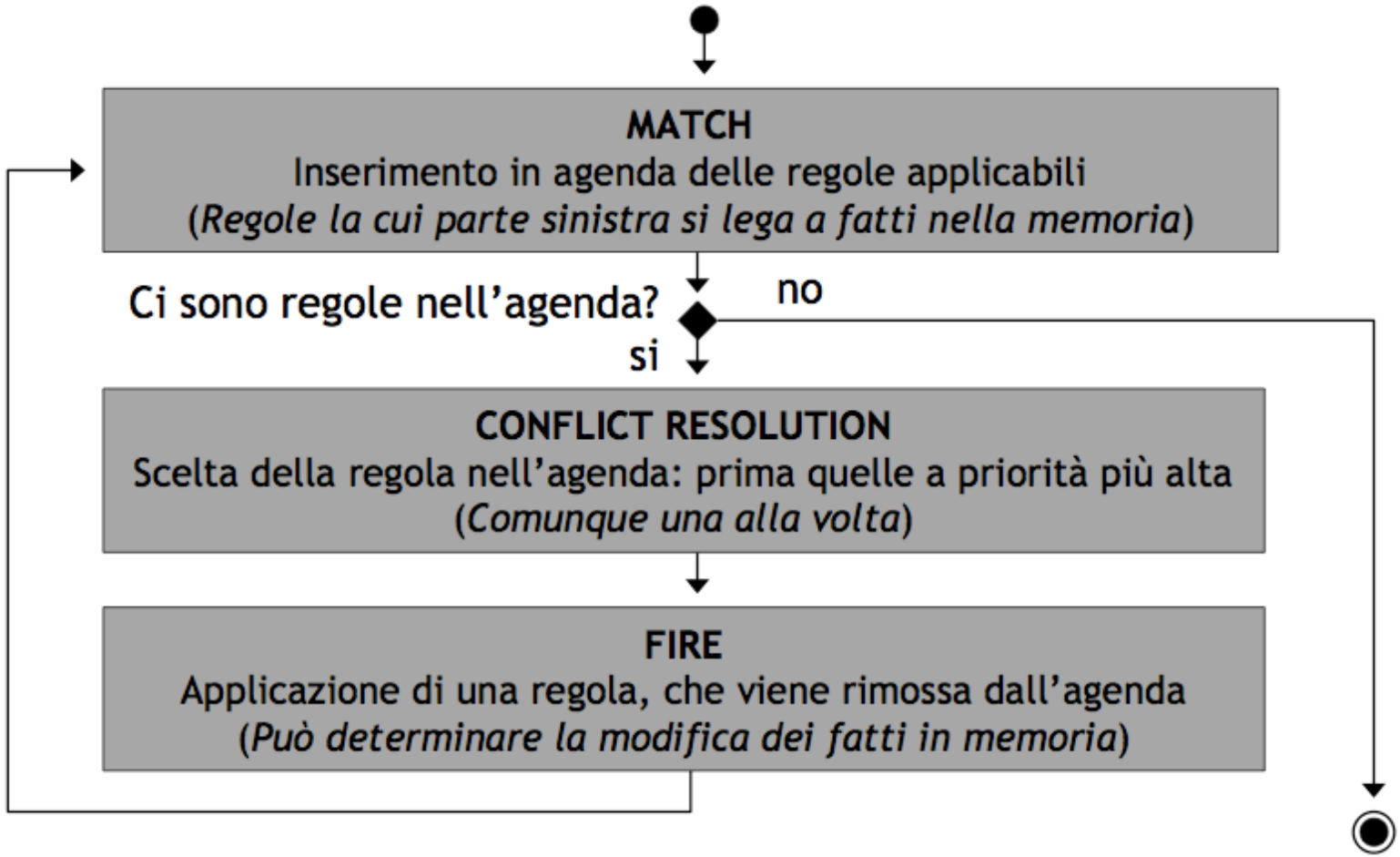
Esempio approccio jess 100%

```
(import jess.examples.pumps.*)
(import javax.swing.*)
(import java.awt.*)

;;(defclass machine jess.examples.pumps.Machine )
(deftemplate machine (slot name) (slot class) (slot OBJECT))
(defclass tank Tank extends machine)
(defclass pump Pump extends machine)

(deffacts idle-fact
  (idle 0)
  (adjust-time 0))

(defrule warn-if-low
```

Quando usare un rule engine?

- Quanta business logic contiene l'applicazione?
- Quanto è complessa la business logic?
- La business logic cambia frequentemente?
- L'applicazione dovrà essere mantenuta nel tempo?
- Ci sono requisiti speciali nelle performance?

Come funziona un sistema di regole generale



Rete
Elementi della struttura a rete

- **Memorie Alpha**
  - Ciascuna di esse rappresenta una condizione (un letterale) nella LHS di una regola
    - Sono ammesse fattorizzazioni: condizioni identiche, anche in regole diverse, sono rappresentate dalla stessa memoria Alpha

Rete
Memorie Alpha

```
(defrule "Sorella"
   (Madre ?m ?x)
   (Madre ?m ?y)
   (Padre ?p ?x)
   (Padre ?p ?y)
   (Femmina ?x)
   ⇒
   (Sorella ?x ?y))
```

Memorie *Alpha*

(*Madre* ?v1 ?v2)

(*Padre* ?v1 ?v2)

(*Femmina* ?v1)

Le due condizioni simili si traducono con una sola memoria *Alpha*

Non si preserva il nome delle variabili

Rete
Elementi della struttura a rete

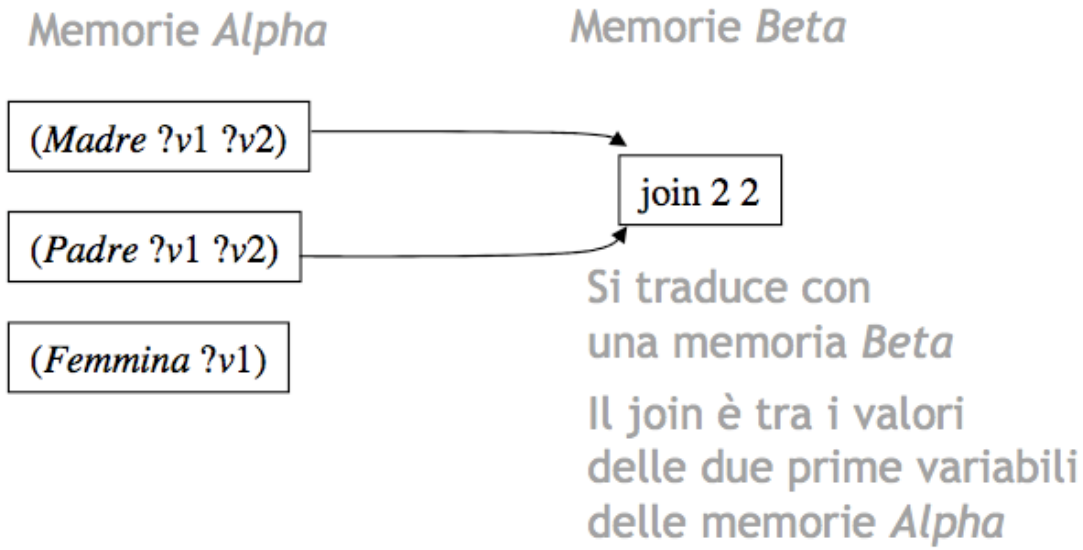
- **Memorie Beta**
  - Ciascuna di esse rappresenta una combinazione binaria di condizioni, nella LHS di una regola
    - Di fatto è un join tra i valori (esattamente come in una base di dati relazionale)

Rete
Memorie Beta (1/2)

Esempio:

```
(defrule "Sorella"
  (Madre ?m ?x)
  (Madre ?m ?y)
  (Padre ?p ?x)
  (Padre ?p ?y)
  (Femmina ?x)
  ⇒
  (Sorella ?x ?y))
```

Questa combinazione



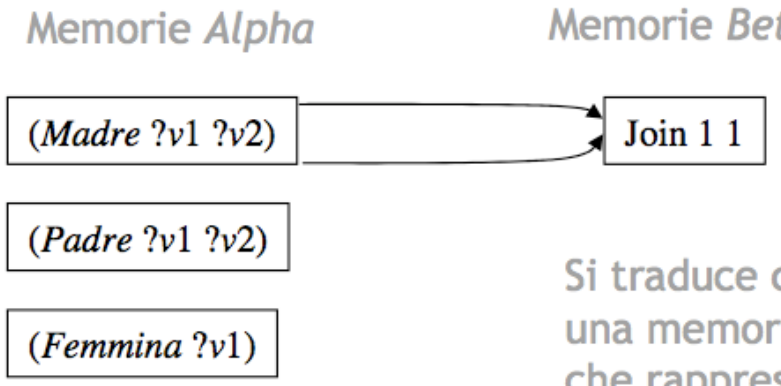
Si traduce con una memoria *Beta*

Il join è tra i valori delle due prime variabili delle memorie *Alpha*

Rete
Memorie Beta (2/2)

Esempio:

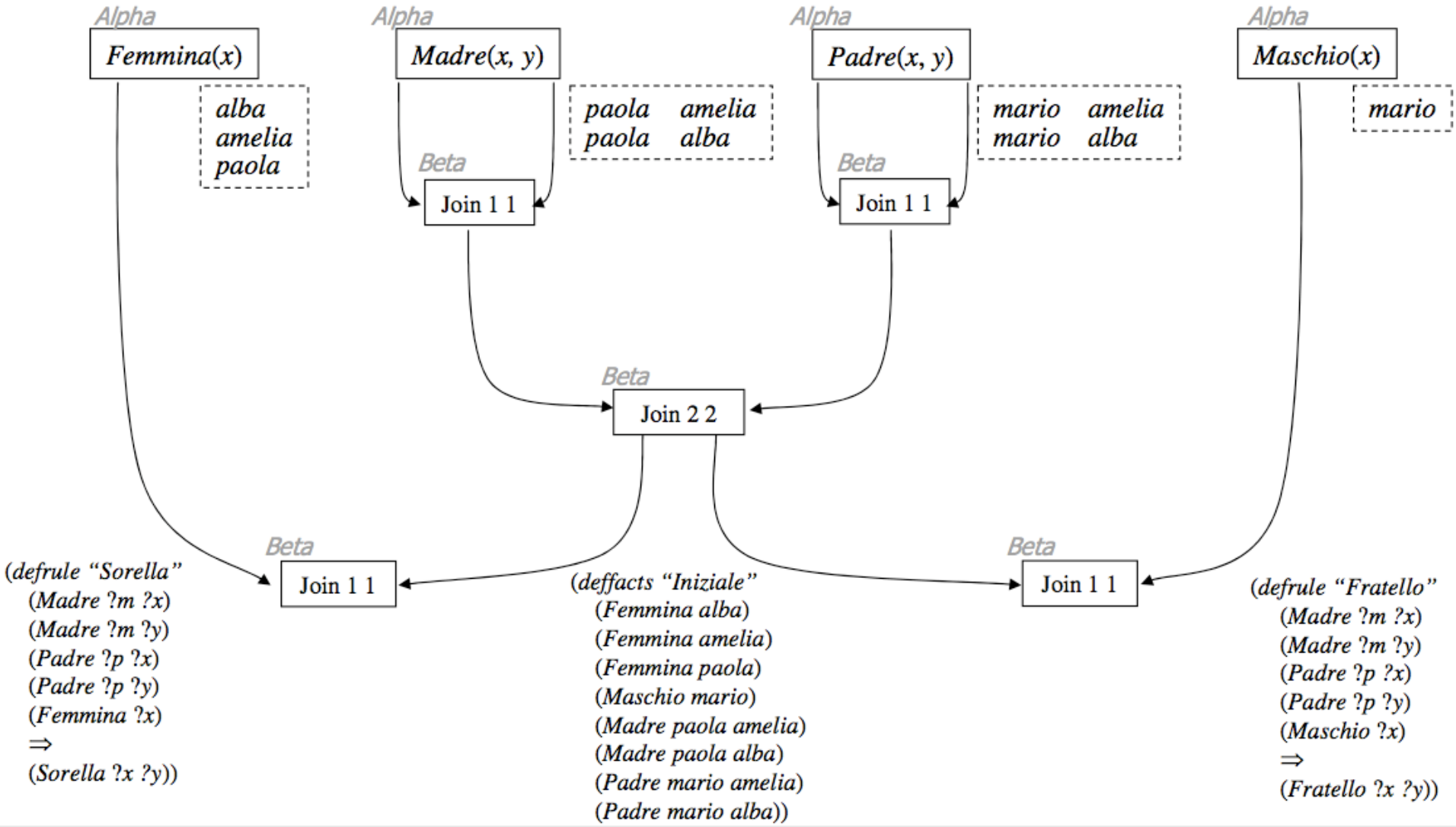
(*defrule "Sorella"*
Questa combinazione
(*Madre ?m ?x*)
(*Madre ?m ?y*)
(*Padre ?p ?x*)
(*Padre ?p ?y*)
(*Femmina ?x*)
⇒
(*Sorella ?x ?y*))



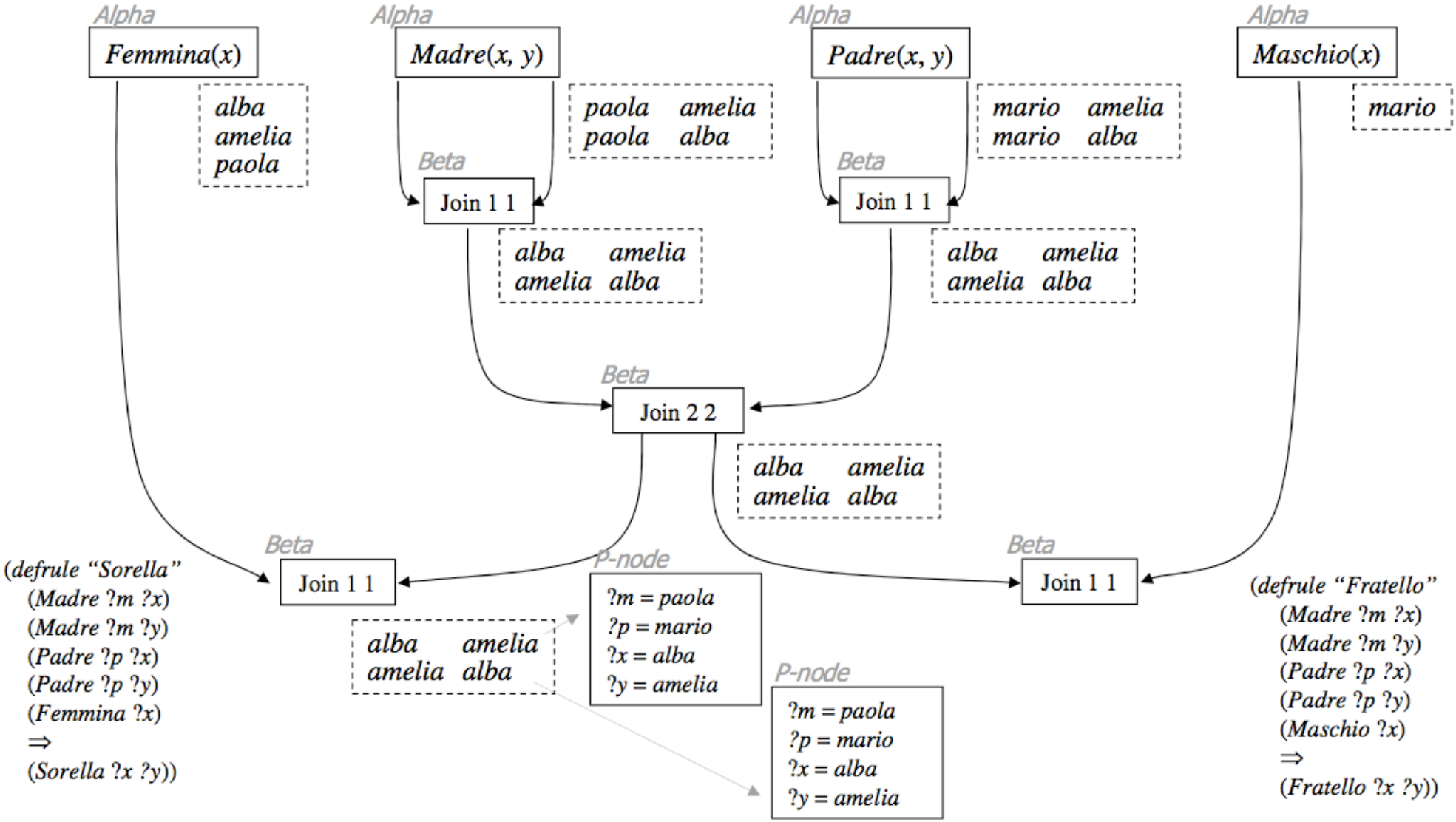
Si traduce con
una memoria *Beta*
che rappresenta un join
sulla stessa memoria *Alpha*

Anche i join possono essere
fattorizzati: lo stesso join può
occorrere in più regole

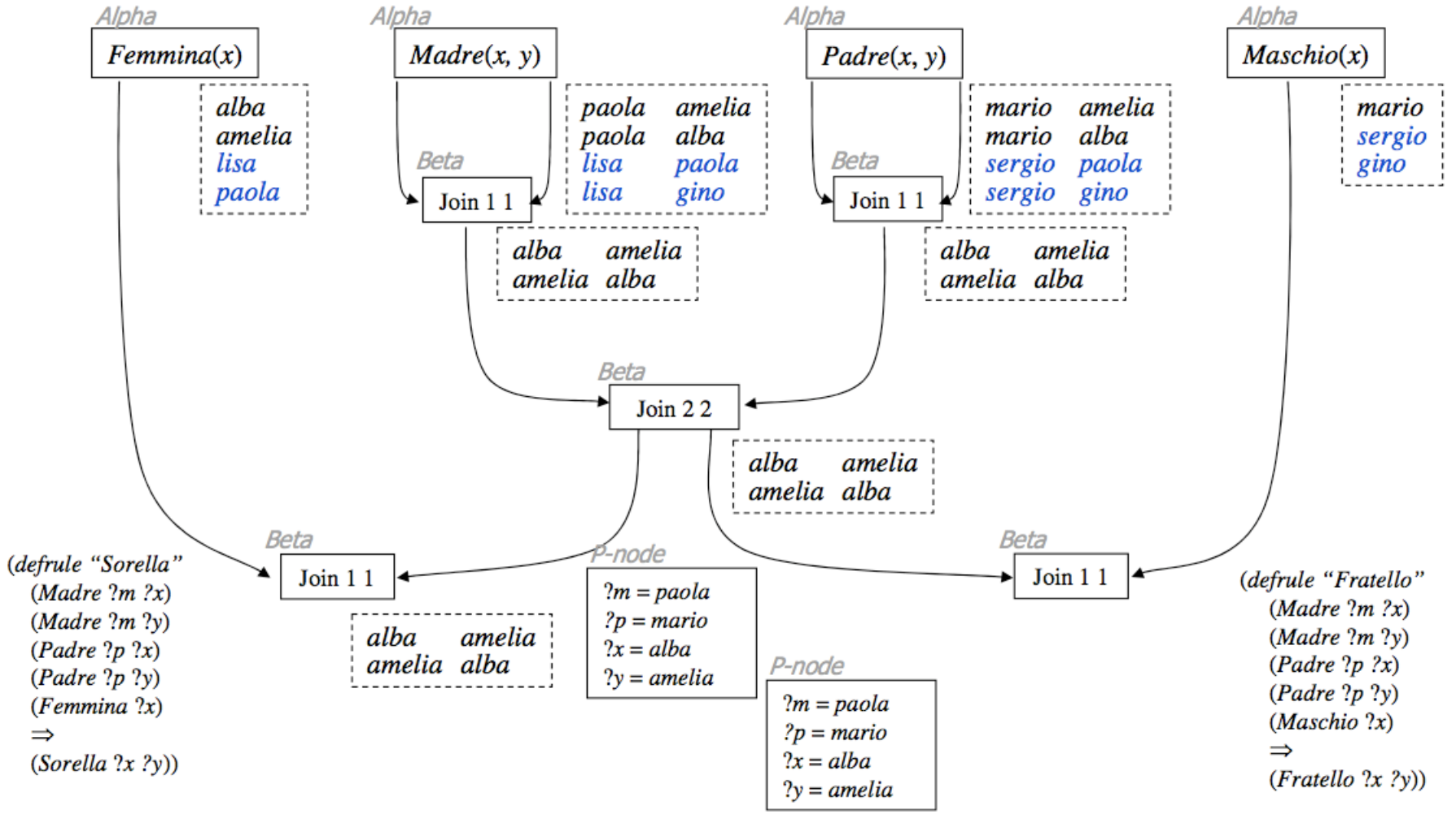
Esempio


Alpha
Femmina(x)
alba
amelia
paola
Alpha
Madre(x, y)
paola amelia
paola alba
Beta
Join 1 1
Alpha
Padre(x, y)
mario amelia
mario alba
Beta
Join 1 1
Alpha
Maschio(x)
mario
Beta
Join 2 2
Beta
Join 1 1
Beta
Join 1 1
(defrule "Sorella"
(Madre ?m ?x)
(Madre ?m ?y)
(Padre ?p ?x)
(Padre ?p ?y)
(Femmina ?x)
⇒
(Sorella ?x ?y))
(deffacts "Iniziale"
(Femmina alba)
(Femmina amelia)
(Femmina paola)
(Maschio mario)
(Madre paola amelia)
(Madre paola alba)
(Padre mario amelia)
(Padre mario alba))
(defrule "Fratello"
(Madre ?m ?x)
(Madre ?m ?y)
(Padre ?p ?x)
(Padre ?p ?y)
(Maschio ?x)
⇒
(Fratello ?x ?y))

Esempio: P-node



Esempio: nuove regole



Sintassi JESS

I fatti vengono espressi come tuple

```
(maschio Mario)
(femmina Paola)
(padre Alba Mario)
(padre Amelia Mario)
(madre Alba Paola)
(madre Amelia Paola)
```

Le regole vengono espresse con una sintassi particolare

```
(defrule sorella
   (padre ?x ?p)
   (padre ?y ?p)
   (madre ?x ?m)
   (madre ?y ?m)
   (femmina ?x)
  =>
       (assert (sorella ?x ?y)))
```

Sintassi JESS

Due funzioni molto comuni (compaiono solo nelle *azioni*)

(assert <fact>**)**

inserimento di <fact> nella memoria di lavoro
(e aggiornamento della struttura Rete)

(retract <fact>**)**

rimozione di <fact> dalla memoria di lavoro
(e aggiornamento della struttura Rete)

Sintassi JESS

Il sistema Jess si attiva da linea di comando

$ java jess.Main
Jess>

La prima linea attiva il sistema,
nella seconda compare il prompt del Lisp Listener

Tipicamente le regole sono scritte su un file,
ma possono anche essere inserite direttamente

Jess> (batch regole.clp)

Caricamento del file **regole.clp**

L'attivazione prevede

l'inizializzazione della memoria di lavoro
l'attivazione del ciclo principale

Jess> (reset)
Jess> (run)

il sistema rimane attivo finchè vi sono regole da eseguire

Esercizio

- Sia data la seguente base di conoscenza proposizionale KB:

$$\text{TemperaturaMite} \wedge \text{AcqueStagnanti} \Rightarrow \text{PresenzaZanzare}$$

$$\text{Pianura} \Rightarrow \text{NoDeflusso}$$

$$\text{TerrenoArgilloso} \Rightarrow \text{NoAssorbimento}$$

$$\text{Piogge} \wedge \text{NoDeflusso} \wedge \text{NoAssorbimento} \Rightarrow \text{AcqueStagnanti}$$

$$\text{Primavera} \Rightarrow \text{TemperaturaMite}$$

$$\text{Estate} \Rightarrow \text{TemperaturaMite}$$

$$\text{Autunno} \Rightarrow \text{TemperaturaMite}$$

$$\text{Autunno} \Rightarrow \text{Piogge}$$

Autunno
Pianura
TerrenoArgilloso

Dire se vale KB $\models$ PresenzaZanzare.

Giustificare la risposta tramite forward chaining

Soluzione

- Con una possibile esecuzione di forward chaining si inseriscono nella base di conoscenza le seguenti formule atomiche
  - TemeraturaMite
  - Piogge
  - NoDeflusso
  - NoAssorbimento
  - AcquesStagnanti
  - PresenzaZanzare

Esempio: ZOOKEEPER (1/3)

Regole

```
(defrule bird-test
      (has-feathers ?)
=>
      (bird ?)
)

(defrule mammal-test
      (gives-milk ?)
=>
      (mammal ?)
)
```

```
(defrule ungulate-test1
      (mammal ?)
      (chews-cud ?)
=>
      (is-ungulate ?)
)

(defrule ungulate-test2
      (mammal ?)
      (has-hoofs ?)
=>
      (is-ungulate ?)
)
```

Working Memory

```
(gives-milk animal)

(has-hoofs animal)
```

Esempio: ZOOKEEPER (2/3)

Regole

```
(defrule bird-test
      (has-feathers ?)
=>
      (bird ?)
)

(defrule mammal-test
      (gives-milk ?)
=>
      (mammal ?)
)
```

```
(defrule ungulate-test1
      (mammal ?)
      (chews-cud ?)
=>
      (is-ungulate ?)
)

(defrule ungulate-test2
      (mammal ?)
      (has-hoofs ?)
=>
      (is-ungulate ?)
)
```

Working Memory

```
(gives-milk animal)

(has-hoofs animal)

(mammal animal)
```

Esempio: ZOOKEEPER (3/3)

Regole

```
(defrule bird-test
      (has-feathers ?)
=>
      (bird ?)
)

(defrule mammal-test
      (gives-milk ?)
=>
      (mammal ?)
)

(defrule ungulate-test1
      (mammal ?)
      (chews-cud ?)
=>
      (is-ungulate ?)
)

(defrule ungulate-test2
      (mammal ?)
      (has-hoofs ?)
=>
      (is-ungulate ?)
)
```

Working Memory

```
(gives-milk animal)

(has-hoofs animal)

(mammal animal)

(is-ungulate animal)
```

Esercizio x casa

un agricoltore (farmer)
una volpe (fox)
una capra (goat)
un cavolo (cabbage)

una barca
due rive (shore-1, shore-2)

Obiettivo
tutti i partecipanti sono su una riva (shore-1)
l'agricoltore deve traghettare tutti sulla riva opposta (shore-2)

Vincoli:
se lasciata sola con la capra, la volpe mangia la capra
se lasciata sola con il cavolo, la capra mangia il cavolo

Esercizio

- Mostrare l'esecuzione di un semplice sistema di produzioni per ordinare una stringa fatta di "a","b" e "c".
- Insieme di regole:
  - **ba => ab**
  - **ca => ac**
  - **cb => bc**
- Memoria di lavoro: la stringa in una fase intermedia del processo di ordinamento (all'inizio la stringa da ordinare).
- Strategia: la prima regola applicabile.

Soluzione

Ciclo	Memoria di lavoro	Insieme dei conflitti	Regola selezionata
0	cbaca	1,2,3	1
...	...	...	...

Esercizio A (1/2)

- Si consideri la seguente base di conoscenza:

$$
\begin{array}{ll}
KB1. & \forall x\big(sciopero(x) \Rightarrow \neg lavaggiostrade(x)\big) \\
KB2. & \forall x\big(beltempo(x) \Rightarrow \neg piove(x)\big) \\
KB3. & \forall x\big(piove(x) \Rightarrow stradabagnata(x)\big) \\
KB4. & \forall x\big(\neg piove(x) \wedge \neg lavaggiostrade(x) \Rightarrow \neg stradabagnata(x)\big) \\
KB5. & \forall x\big(lavaggiostrade(x) \Rightarrow stradabagnata(x)\big) \\
KB6. & \forall x\big(pari(x) \wedge \neg sciopero(x) \Rightarrow lavaggiostrade(x)\big) \\
KB7. & \forall x\big(dispari(x) \Rightarrow \neg lavaggiostrade(x)\big) \\
KB8. & \forall xy\big(stradabagnata(x) \wedge auto(y) \Rightarrow velocità50(y,x)\big) \\
KB9. & \forall xy\big(\neg stradabagnata(x) \wedge auto(y) \Rightarrow velocità70(y,x)\big) \\
KB10. & auto(500) \\
KB11. & sciopero(OGGI) \\
KB12. & beltempo(OGGI)
\end{array}
$$

- scrivere le regole in OPS5 (o JESS) e si dimostri con forward chaining la seguente formula:

Velocita70(500,OGGI)

Esercizio A (2/2)

- Utilizzare la seguente tabella:

Ciclo	Memoria di lavoro	Insieme dei conflitti	Regola selezionata

Esercizio B

- Si consideri la seguente base di conoscenza:

1	$\forall xyz[pred2(x,y) \wedge pred7(y,z) \Rightarrow pred2(x,z)]$
2	$\forall xy[pred3(x,y) \wedge pred4(y) \Rightarrow pred3(y,x)]$
3	$\forall xy[pred3(x,y) \wedge pred4(x) \Rightarrow pred2(x,y)]$
4	$\forall xy[pred2(x,y) \wedge pred3(x,y) \wedge pred4(y) \Rightarrow pred1(x,y)]$
5	$\forall xy[pred5(x,y) \wedge pred4(y) \Rightarrow pred4(x)]$
6	$\forall xy[pred1(x,y) \wedge pred5(x) \Rightarrow pred6(y)]$
7	$pred2(A,C)$
8	$pred2(D,B)$
9	$pred3(B,A)$
10	$pred4(C)$
11	$pred4(A)$
12	$pred5(B,A)$

- scrivere le regole in OPS5 (o JESS) e si dimostri con backward chaining la seguente formula:

Pred1(A,B)

Esercizio C (1/2)

- Si consideri la seguente base di conoscenza:

1 $(H(y) \wedge G(x)) \Rightarrow C(x)$

2 $(F(x) \wedge H(x)) \Rightarrow D(x)$

3 $F(x) \Rightarrow G(x)$

4 $G(x) \Rightarrow D(x)$

5 $(D(x) \wedge G(y)) \Rightarrow E(y)$

6 $(C(x) \wedge D(y)) \Rightarrow A(x)$

7 $(C(x) \wedge E(y)) \Rightarrow A(y)$

8 $F(Term1)$

9 $F(Term2)$

10 $H(Term1)$

- scrivere le regole in OPS5 (o JESS) e si dimostri con forward chaining la seguente formula:

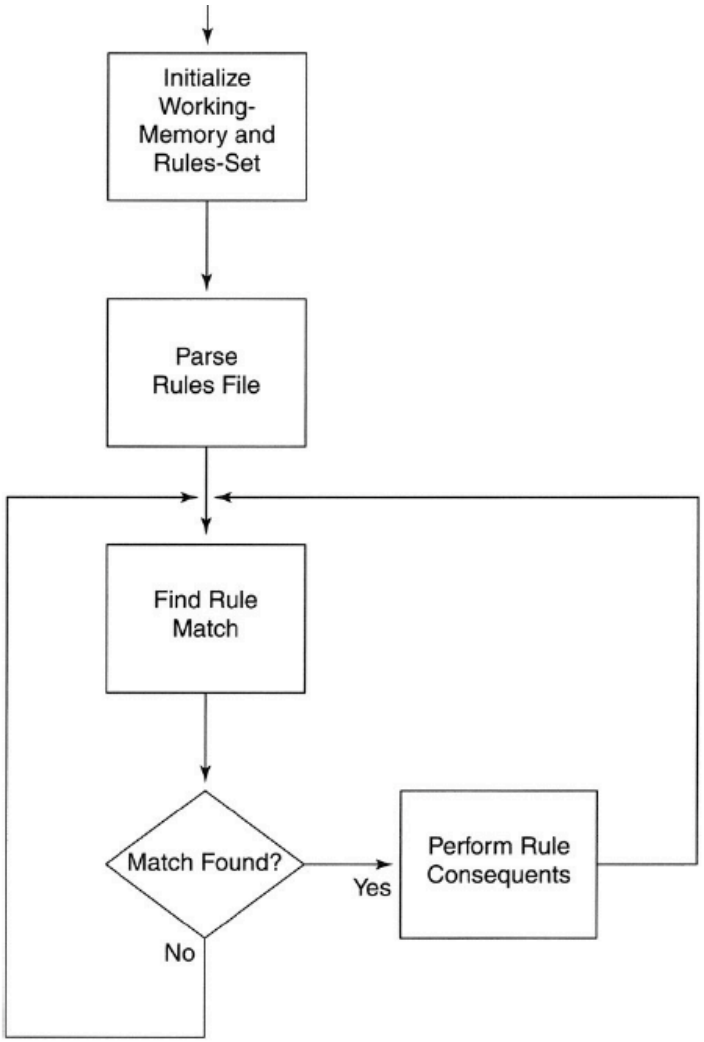
A(Term2)

Esercizio C (2/2)

- Utilizzare la seguente tabella:

Ciclo	Memoria di lavoro	Insieme dei conflitti	Regola selezionata

Diagramma di flusso RBS



Strategie di risoluzione dei conflitti

- Basate su regole:
  - la prima regola applicabile
  - la più specifica o con condizioni più stringenti
    - ES: c1 and c2 and c3 < c1 and c2
  - non di nuovo la stessa sotto le stesse condizioni (rifrazione)
  - la più trascurata: usata meno di recente
  - non di nuovo lo stesso effetto
  - la più recentemente attivata (focus)
  - pattern matching apporssimato

Strategie di risoluzione dei conflitti

- Basate sugli oggetti:
  - su una graduatoria di importanza degli oggetti che compaiono nei pattern
  - (Stanza in-fiamme)(Bambino in-pericolo)=> (Salva-bambino) (Esci)
  - (Stanza in-fiamme)(Luce accesa) => (Spegni-luce)(Esci)

Strategie di risoluzione dei conflitti

- Basate sull'effetto di regole:
  - si applica una funzione di valutazione agli stati risultanti e si sceglie il migliore

Esempio di file di regole

```
(defrule init
 (true null)
 =>
 (add (is-milk-giver animal))
 (add (has-hooves animal))
 (add (has-black-stripes animal))
)

 (defrule mammal-1
  (has-hair ?)
  =>
  (add(is-animal ?))
 )

(defrule mammal-2
  (is-milk-giver ?)
  =>
  (add(is-mammal ?))
 )

 (defrule bird-1
  (has-feathers ?)
  =>
  (add(is-bird ?))
 )

 (defrule bird-2
  (is-flier ?)
  (is-egg-layer ?)
  =>
  (add(is-bird ?))
 )
```

```
(defrule carnivore-1
 (is-meat-eater ?)
 =>
 (add(is-carnivore ?))
)


(defrule carnivore-2
 (has-pointed-teeth ?)
 (has-claws ?)
 (has-forward-eyes ?)
 =>
 (add(is-carnivore ?))
)

(defrule ungulate-1
 (is-mammal ?)
 (has-hooves ?)
 =>
 (add(is-ungulate ?))
)

(defrule ungulate-2
 (is-mammal ?)
 (is-cud-chewer ?)
 =>
 (add(is-ungulate ?))
)

(defrule even-toed
  (is-mammal ?)
  (is-cud-chewer ?)
  =>
  (add(is-even-toed ?))
 )
```

```
(defrule cheetah
 (is-mammal ?)
 (is-carnivore ?)
 (is-tanwny-colored ?)
 (has-dark-spots ?)
 =>
 (add(is-cheetah ?))
 (print("Animal is a cheetah"))
 (quit null)
)

(defrule tiger
 (is-mammal ?)
 (is-carnivore ?)
 (is-tawny-colored ?)
 (has-black-strips ?)
 =>
 (add(is-tiger ?))
 (print("Animal is a tiger"))
 (quit null)
)

(defrule giraffe
 (is-ungulate ?)
 (has-long-neck ?)
 (has-long-legs ?)
 (has-dark-spots ?)
 =>
 (add(is-giraffe ?))
 (print("Animal is a giraffe"))
 (quit null)
)
```

funzioni di supporto

parseElement

```
/*
 *  parseElement()
 *
 *  Parse a single consequent or antecedent from the file.
 *
 */

char *parseElement( char *block, memoryElementType **met )
{
  memoryElementType *element;
  int i=0; int balance = 1;
  element = (memoryElementType *)malloc(sizeof(memoryElementType));
  element->element[i++] = *block++;
  while (1) {

    if (*block == 0) break;
    if (*block == ')') balance--;
    if (*block == '(') balance++;
    element->element[i++] = *block++;

    if (balance == 0) break;

  }
  element->element[i] = 0;
  element->next = 0;

  if (*met == 0) *met = element;
  else {
    memoryElementType *chain = *met;
    while (chain->next != 0) chain = chain->next;
    chain->next = element;
  }

  return block;
}
```

Strutture dati
RBS (1/2)

```
#ifndef _COMMON_H
#define _COMMON_H

#define MEMORY_ELEMENT_SIZE 80

typedef struct memoryElementStruct *memPtr;

typedef struct memoryElementStruct {
  int active;
  char element[MEMORY_ELEMENT_SIZE+1];
  struct memoryElementStruct *next;
} memoryElementType;

#define MAX_MEMORY_ELEMENTS 40

#define MAX_RULES   40
```

Strutture dati
RBS (2/2)

```
typedef struct {
  int active;
  char ruleName[MEMORY_ELEMENT_SIZE+1];
  memoryElementType *antecedent;
  memoryElementType *consequent;
} ruleType;


#define MAX_TIMERS     10

typedef struct {
  int active;
  int expiration;
} timerType;

#endif /* _COMMON_H */
```

Strutture dati
RBS

- **`memoryElementType`** è utilizzato per rappresentare i fatti nella working memory e le regole nel rules-set.
- **`element`** modella il fatto della working memory
- **`ruleType`** rappresenta una singola regola
- la parola **`active`** definisce se la regola è attiva nel sistema
- **`ruleName`** è il nome della regola
- **`antecedent`** e **`consequent`** rappresentano le liste concatenate di termini
- **`timerType`** è utilizzato per rappresentare un singolo timer.

main function

```
int main( int argc, char *argv[] )
{
    int opt, ret;
    char inpfile[80]={0};

    extern void processTimers( void );
    extern int parseFile( char * );
    extern void interpret( void );

    while ((opt = getopt(argc, argv, "hdr:")) != -1) {

           switch( opt ) {

                case 'h':
                     emitHelp();
                     break;

                case 'd':
                     debug = 1;
                     printf("Debugging enabled\n");
                     break;

                case 'r':
                     strcpy(inpfile, optarg);
                     break;

           }

    }

    if (inpfile[0] == 0) emitHelp();

    bzero( (void *)workingMemory, sizeof(workingMemory) );
    bzero( (void *)ruleSet, sizeof(ruleSet) );
    bzero( (void *)timers, sizeof(timers) );

    ret = parseFile( inpfile );

    if (ret < 0) {
           printf("\nCould not open file, or parse error\n\n");
           exit(0);
    }

    while (1) {

           interpret();

           if (debug) {
                printWorkingMemory();
           }
           processTimers();

           if (endRun) break;

           sleep(1);

    }

    return 0;
}
```

main function

- il main provvede all'inizializzazione del sistema:
  - vengono catturati i commandi passati a linea di comando con la funzione **`getopt`**
    - 'h' per ottenere le informazioni di help
    - 'd' per abilitare il debug
    - 'r' per specificare il file di regole da utilizzare.

parseFile function (1/2)

- La funzione di parsing ha lo scopo di ‘parsare’ una stringa composta nel seguente modo:

```
(defrule <rule-name>
        (antecedent-terms)
 =>
        (consequent-terms)
)
```

parseFile function (2/2)

```
int parseFile( char *filename )
{
    FILE *fp;
    char *file, *cur;
    int fail = 0;

    extern int debug;

    file = (char *)malloc(MAX_FILE_SIZE);

    if (file == NULL) return -1;

    fp = fopen(filename, "r");
    if (fp == NULL) {
            free(file);
            return -1;
    }
    fread( file, MAX_FILE_SIZE, 1, fp);
    cur = &file[0];
    while (1) {
            cur = strstr( cur, "(defrule" );
            if (cur == NULL) {
                  fail = 1;
                  break;
            }
            if (!strncmp(cur, "(defrule", 8)) {
                  int i=0;
                  cur+=9;
                  while (*cur != 0x0a) {
                        ruleSet[ruleIndex].ruleName[i++] = *cur++;
                  }
                  ruleSet[ruleIndex].ruleName[i++] = 0;
                  cur = skipWhiteSpace( cur );
                  /* Parse the antecedents */
                  cur = parseAntecedent( cur, &ruleSet[ruleIndex] );
                  if (cur == NULL) {
                        fail = 1;
                        break;
                  }
                  /* Should be sitting on the '=>' */
                  if (!strncmp(cur, "=>", 2)) {
                        cur = skipWhiteSpace( cur+2 );
                        /* Parse the consequents */
                        cur = parseConsequent( cur, &ruleSet[ruleIndex] );
                        if (cur == NULL) {
                              fail = 1;
                              break;
                        }
                        /* Ensure we're closing out the current rule */
                        if (*cur == ')') {
                              cur = skipWhiteSpace( cur+1 );
                        } else {
                              fail = 1;
                              break;
                        }
                  } else {
                        fail = 1;
                        break;
                  }
                  ruleSet[ruleIndex].active = 1;
                  ruleIndex++;
            } else {
                  break;
            }
    }
    if (debug) {
            printf("Found %d rules\n", ruleIndex);
    }
    free( (void *)file );
    fclose(fp);
    return 0;
}
```

funzioni di supporto

skipWhiteSpace

```
char *skipWhiteSpace( char *block )
{
  char ch;
  while (1) {
    ch = *block;

    while ((ch != '(') && (ch != ')') && (ch != '=') &&
           (ch != 0) && (ch != ';')) {
      block++;
      ch = *block;
    }
    if (ch == ';') {
      while (*block++ != 0x0a);
    } else break;
  }
  return block;
}
```

funzioni di supporto

parseAntecedent - parseConsequent

```
/*
 *  parseAntecedent()
 *
 *  Parse an antecedent from the file.
 *
 */
char *parseAntecedent( char *block, ruleType *rule )
{
  while (1) {
    block = skipWhiteSpace( block );
    if (*block == '(') {

      block = parseElement( block, &rule->antecedent );

    } else break;
  }
  return block;
}
```

```
/*
 *  parseConsequent()
 *
 *  Parse a consequent from the file.
 *
 */

char *parseConsequent( char *block, ruleType *rule )
{
  while (1) {

    block = skipWhiteSpace( block );

    if (*block == '(') {

      block = parseElement(block, &rule->consequent);

    } else break;

  }

  return block;
}
```

RuleCheck

searchWorkingMemory

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "common.h"

extern memoryElementType workingMemory[MAX_MEMORY_ELEMENTS];
extern ruleType ruleSet[MAX_RULES];

memoryElementType *chain = NULL;

/*
 *  searchWorkingMemory()
 *
 *  Search the working memory for the two passed terms.
 *
 */

int searchWorkingMemory( char *term1, char *term2 )
{
  int ret = 0;
  int curMem = 0;
  char wm_term1[MEMORY_ELEMENT_SIZE+1];
  char wm_term2[MEMORY_ELEMENT_SIZE+1];

  while (1) {

    if (workingMemory[curMem].active) {

      /* extract the memory element */
      sscanf(workingMemory[curMem].element, "(%s %s)", wm_term1, wm_term2);
      if (wm_term2[strlen(wm_term2)-1] == ')') wm_term2[strlen(wm_term2)-1] = 0;

      if ((!strncmp(term1, wm_term1, strlen(term1))) &&
          (!strncmp(term2, wm_term2, strlen(term2)))) {

        ret = 1;
        break;

      }

    }

    curMem++;

    if (curMem == MAX_MEMORY_ELEMENTS) {
      break;
    }

  }

  return ret;
}
```

RuleCheck

addToChain

```
/*
 *  addToChain()
 *
 *  Create a chain second terms from working memory where the first terms
 *  match, and the second term in the antecedent is the '?' symbol.
 *
 */

void addToChain( char *element )
{
  memoryElementType *walker, *newElement;;

  newElement = (memoryElementType *)malloc(sizeof(memoryElementType));

  strcpy( newElement->element, element );

  if (chain == NULL) {
    chain = newElement;
  } else {
    walker = chain;
    while (walker->next) walker = walker->next;
    walker->next = newElement;
  }

  newElement->next = NULL;
}
```

RuleCheck

checkPattern (1/2)

```
/*
 *  checkPattern()
 *
 *  Try to match all of the antecedents of a rule to facts in working
 *  memory.  This includes matching antecedent rules with '?' terms
 *  with facts from working memory.
 *
 */

int checkPattern( int rule, char *arg )
{
  int ret=0;
  char term1[MEMORY_ELEMENT_SIZE+1];
  char term2[MEMORY_ELEMENT_SIZE+1];
  memoryElementType *antecedent = ruleSet[rule].antecedent;

  /* Build a replacement string (based upon the '?' element) */
  while (antecedent) {

    sscanf( antecedent->element, "(%s %s)", term1, term2);
    if (term2[strlen(term2)-1] == ')') term2[strlen(term2)-1] = 0;

    /* If the antecedent element is variable, find the matches
     * in the working memory and store the matched terms.
     */
    if (term2[0] == '?') {
      int i;
      char wm_term1[MEMORY_ELEMENT_SIZE+1];
      char wm_term2[MEMORY_ELEMENT_SIZE+1];

      for (i = 0 ; i < MAX_MEMORY_ELEMENTS ; i++) {

        if (workingMemory[i].active) {

          sscanf( workingMemory[i].element, "(%s %s)", wm_term1, wm_term2 );
          if (wm_term2[strlen(wm_term2)-1] == ')')
            wm_term2[strlen(wm_term2)-1] = 0;

          if (!strncmp(term1, wm_term1, strlen(term1))) addToChain(wm_term2);

        }

      }

    }

    antecedent = antecedent->next;

  }

}
```

RuleCheck

checkPattern (2/2)

```
/* Now that we have the replacement strings, walk through the rules trying
  * the replacement string when necessary.
  */

 do {

   memoryElementType *curRulePtr, *temp;

   curRulePtr = ruleSet[rule].antecedent;

   while (curRulePtr) {

     sscanf( curRulePtr->element, "(%s %s)", term1, term2 );
     if (term2[strlen(term2)-1] == ')') term2[strlen(term2)-1] = 0;

     if (!strncmp( term1, "true", strlen(term1))) {
       ret = 1;
       break;
     } else {
       if ((term2[0] == '?') && (chain)) strcpy(term2, chain->element);
     }

     ret = searchWorkingMemory( term1, term2 );

     if (!ret) break;

     curRulePtr = curRulePtr->next;

   }

   if (ret) {

     /* Cleanup the replacement string chain */
     while (chain) {
       temp = chain;
       chain = chain->next;
       free(temp);
     }

     strcpy(arg, term2);

   } else {

     if (chain) {
       temp = chain;
       chain = chain->next;
       free(temp);
     }

   }

 } while (chain);

 return ret;
```

RuleCheck

checkRule

```
/*
 *  checkRule()
 *
 *  Check to see if the rule will fire, based upon the facts in
 *  working memory.
 *
 */
int checkRule( int rule )
{
  int fire = 0;
  char arg[MEMORY_ELEMENT_SIZE]={0};

  extern int fireRule( int, char * );

  fire = checkPattern(rule, arg);

  if (fire == 1) {
    fire = fireRule( rule, arg );
  }
  return fire;
}
```