

# Intelligenza Artificiale I

## Esercitazione 5 Sistemi a regole

Marco Piastra

## Sistemi a regole (Jess)

- Un sistema a regole contiene regole e fatti
  - Nelle regole, tutte le variabili sono universalmente quantificate
  - I fatti non contengono variabili

- Ciascuna regola è un'implicazione

<LHS - *Left Hand Side*>  $\Rightarrow$  <RHS - *Right Hand Side*>

LHS e RHS

Sono congiunzioni di fbf atomiche in forma positiva

Morale: le regole Jess sono traducibili in regole di Horn (quasi vero ...)

(defrule "Sorella"

(Madre ?m ?x)

(Madre ?m ?y)

(Padre ?p ?x)

(Padre ?p ?y)

(Femmina ?x)

$\Rightarrow$

(Sorella ?x ?y))

(defrule "Fratello"

(Madre ?m ?x)

(Madre ?m ?y)

(Padre ?p ?x)

(Padre ?p ?y)

(Maschio ?x)

$\Rightarrow$

(Fratello ?x ?y))

## Sistemi a regole (Jess)

- Un sistema a regole contiene regole e fatti
  - Nelle regole, tutte le variabili sono universalmente quantificate
  - I fatti non contengono variabili

- Ciascuna regola è un'implicazione

<LHS - *Left Hand Side*>  $\Rightarrow$  <RHS - *Right Hand Side*>

LHS e RHS

Sono congiunzioni di fbf atomiche in forma positiva

Morale: le regole Jess sono traducibili in regole di Horn (quasi vero ...)

Regola “Sorella”

$$\forall m \forall p \forall x \forall y$$

$$(( \text{Madre}(m, x)$$

$$\wedge \text{Madre}(m, y)$$

$$\wedge \text{Padre}(p, x)$$

$$\wedge \text{Padre}(p, y)$$

$$\wedge \text{Femmina}(x) )$$

$$\rightarrow$$

$$\text{Sorella}(x, y))$$

Regola “Fratello”

$$\forall m \forall p \forall x \forall y$$

$$(( \text{Madre}(m, x)$$

$$\wedge \text{Madre}(m, y)$$

$$\wedge \text{Padre}(p, x)$$

$$\wedge \text{Padre}(p, y)$$

$$\wedge \text{Maschio}(x) )$$

$$\rightarrow$$

$$\text{Fratello}(x, y))$$

## Sistemi a regole (Jess)

- Un sistema a regole contiene regole e fatti
  - Nelle regole, tutte le variabili sono universalmente quantificate
  - I fatti non contengono variabili

- Ciascuna regola è un'implicazione

$\langle \text{LHS} - \text{Left Hand Side} \rangle \Rightarrow \langle \text{RHS} - \text{Right Hand Side} \rangle$

LHS e RHS

Sono congiunzioni di fbf atomiche in forma positiva

Morale: le regole Jess sono traducibili in regole di Horn (quasi vero ...)

Regola “Sorella”

$$\{ \neg \text{Madre}(m, x) \\ \neg \text{Madre}(m, y) \\ \neg \text{Padre}(p, x) \\ \neg \text{Padre}(p, y) \\ \neg \text{Femmina}(x), \\ \text{Sorella}(x, y) \}$$

Regola “Fratello”

$$\{ \neg \text{Madre}(m, x) \\ \neg \text{Madre}(m, y) \\ \neg \text{Padre}(p, x) \\ \neg \text{Padre}(p, y) \\ \neg \text{Maschio}(x) \\ \text{Fratello}(x, y) \}$$

## Sistemi a regole (Jess)

- Un sistema a regole contiene regole e fatti
  - Nelle regole, tutte le variabili sono universalmente quantificate
  - I fatti non contengono variabili
- Ciascun fatto è una fbf atomica
  - Una fbf base (*ground*): non contiene variabili

Fatto 1  
(*Femmina paola*)

Fatto 4  
(*Maschio mario*)

Fatto 2  
(*Femmina amelia*)

Fatto 5  
(*Padre mario amelia*)

Fatto 3  
(*Madre paola amelia*)

## Sistemi a regole (Jess)

- L'uso dei simboli funzionali è limitato

Solo in forma di `deftemplate`

*Esempio:*

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

Si può vedere come la definizione di possibili termini:

```
currentState(event)
inputSymbol(event)
outputSymbol(event)
newState(event)
```

## Sistemi a regole (Jess)

- Un sistema a regole contiene regole e fatti
  - Nelle regole, tutte le variabili sono universalmente quantificate
  - I fatti non contengono variabili
- Ciascun fatto è una fbf atomica
  - Una fbf base (*ground*): non contiene variabili

Fatto 1

*Femmina(paola)*

Fatto 4

*Maschio(mario)*

Fatto 2

*Femmina(amelia)*

Fatto 5

*Padre(mario, amelia)*

Fatto 3

*Madre(paola, amelia)*

## Forward chaining in $L_{PO}$

- Descrizione:

Per stabilire se  $\Gamma \models \varphi$

( $\Gamma$  regole e fatti come clausole definite universalmente quantificate,  $\varphi$  fatto)

Si applicano a  $\Gamma$  le regole di inferenza *INST* e *GMP* in modo esaustivo

*INST*:  $\forall x_1 \forall x_2 \dots \forall x_n \varphi \vdash \varphi [x_1/c_1, x_2/c_2 \dots x_n/c_n]$

*GMP*:  $\alpha_1, \alpha_2, \dots, \alpha_n, \alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n \rightarrow \beta \vdash \beta$

( $\alpha_1, \alpha_2, \dots, \alpha_n, \beta$  sono fbf *ground*, vale a dire dove non occorrono variabili)

Si ottiene un insieme  $\Sigma$  di fatti,  $\Sigma = \{\psi : \Gamma \vdash \psi, \text{ per } INST \text{ e } GMP\}$

L'algoritmo termina con successo se  $\varphi \in \Sigma$

Il metodo è completo per le clausole di Horn, in assenza di simboli funzionali

Un solo simbolo funzionale è sufficiente per causare divergenza

Tutti i fatti devono essere *base (ground)*, cioè non contenere variabili

Il metodo prevede la generazione di tutte le *istanziamenti*

Esistono soluzioni migliorative: p.es. algoritmo Rete



# *Forward Chaining*

Aspetti dell'implementazione effettiva

- **Ipotesi**

Derivata dal teorema di Herbrand

Si calcola il sistema di Herbrand di tutte le regole

Applicazione esaustiva della regola *INST*

Si applicano tutte le regole istanziate a tutti i fatti

Applicazione esaustiva della regola *GMP*

Dalla teoria sappiamo che il metodo è corretto e completo

(Per la classe di fbf cui si applica)

# Forward Chaining

Aspetti dell'implementazione effettiva

## ▪ Ipotesi

Derivata dal teorema di Herbrand

Si calcola il sistema di Herbrand di tutte le regole

Applicazione esaustiva della regola *INST*

Si applicano tutte le regole istanziate a tutti i fatti

Applicazione esaustiva della regola *GMP*

Dalla teoria sappiamo che il metodo è corretto e completo

(Per la classe di fbf cui si applica)

Ha delle prestazioni terribili:

La regola “Sorella” ha 4 variabili distinte:

Occorre generare  $n^4$  istanziazioni  
(dove  $n$  è il numero di costanti del linguaggio)

Per una sola regola

*Va fatto per tutte le regole di un programma*

Regola “Sorella”

$$\{ \neg \text{Madre}(m, x) \\ \neg \text{Madre}(m, y) \\ \neg \text{Padre}(p, x) \\ \neg \text{Padre}(p, y) \\ \neg \text{Femmina}(x), \\ \text{Sorella}(x, y) \}$$

# Forward Chaining

Aspetti dell'implementazione effettiva

## ▪ Ipotesi

Derivata dal teorema di Herbrand

Si calcola il sistema di Herbrand di tutte le regole

Applicazione esaustiva della regola *INST*

Si applicano tutte le regole istanziate a tutti i fatti

Applicazione esaustiva della regola *GMP*

Dalla teoria sappiamo che il metodo è corretto e completo

(Per la classe di fbf cui si applica)

Ha delle prestazioni terribili:

La regola “Sorella” ha 4 variabili distinte:

Occorre generare  $n^4$  istanziazioni

Anche limitando  $n$  alle sole costanti citati nei fatti, si ha un numero elevato

*Poche di queste sono effettivamente usate*

Regola “Sorella”

$$\{ \neg \text{Madre}(m, x) \\ \neg \text{Madre}(m, y) \\ \neg \text{Padre}(p, x) \\ \neg \text{Padre}(p, y) \\ \neg \text{Femmina}(x), \\ \text{Sorella}(x, y) \}$$

# Algoritmo *Rete* (C. Forgy, 1979)

*Forse il più noto per i sistemi a regole*

- Idea di base

Pre-compilare le regole

Generando una struttura a grafo che facilita l'identificazione dei *match*  
(istanziamenti di regole in base ai fatti)

Pro

Si velocizza enormemente l'esecuzione

Contro

Si consuma molta memoria

Si assume che le regole sono stabili e che i fatti cambino spesso

Ricompilare la struttura a grafo di continuo sarebbe controproducente

# Algoritmo *Rete* (C. Forgy, 1980)

- 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*

Esempio:

(defrule "Sorella"

(Madre ?m ?x)

(Madre ?m ?y)

(Padre ?p ?x)

(Padre ?p ?y)

(Femmina ?x)

⇒

(Sorella ?x ?y))

Memorie *Alpha*

<Madre ?1 ?2>

<Padre ?1 ?2>

<Femmina ?1>

# Algoritmo *Rete* (C. Forgy, 1980)

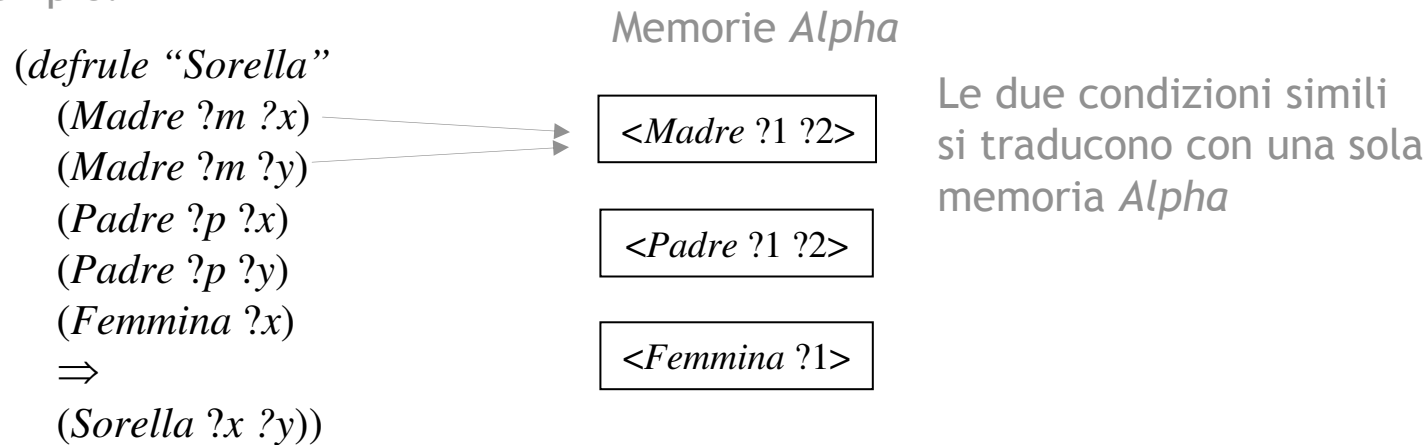
- Elementi della struttura a rete

## 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*

Esempio:



# Algoritmo *Rete* (C. Forgy, 1980)

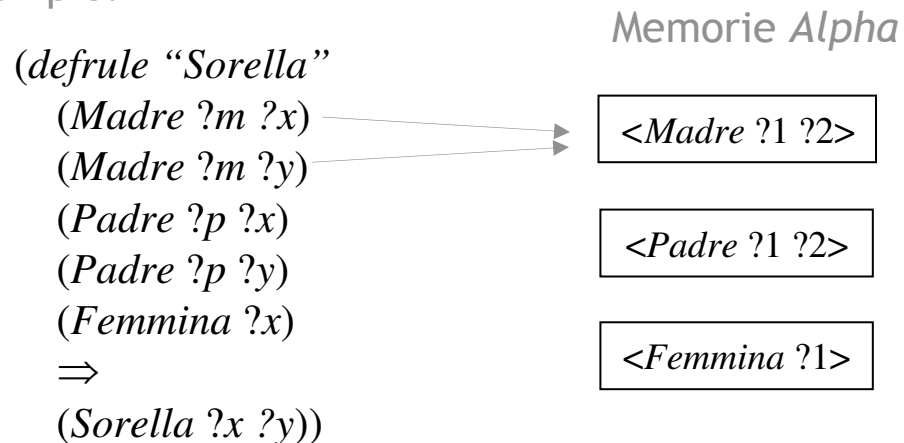
- Elementi della struttura a rete

## 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*

Esempio:



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

Non si preserva il nome delle variabili

# Algoritmo *Rete* (C. Forgy, 1980)

- 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*

Esempio:

(defrule "Sorella"

(Madre ?m ?x)

(Madre ?m ?y)

(Padre ?p ?x)

(Padre ?p ?y)

(Femmina ?x)

⇒

(Sorella ?x ?y))

## Memorie *Alpha*

<Madre ?1 ?2>

<Padre ?1 ?2>

<Femmina ?1>

Si traducono solo le condizioni  
vale a dire le LHS delle regole

Le RHS non vengono tradotte



# Algoritmo *Rete* (C. Forgy, 1980)

- 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 un DB relazionale)

Esempio:

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

## Memorie *Alpha*

<Madre ?1 ?2>

<Padre ?1 ?2>

<Femmina ?1>

# Algoritmo *Rete* (C. Forgy, 1980)

- 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 un DB relazionale)

Esempio:

Questa  
combinazione

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

Memorie *Alpha*

<Madre ?1 ?2>

<Padre ?1 ?2>

<Femmina ?1>

# Algoritmo *Rete* (C. Forgy, 1980)

- 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 un DB relazionale)

Esempio:

Questa  
combinazione

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

Memorie *Alpha*

<Madre ?1 ?2>

<Padre ?1 ?2>

<Femmina ?1>

a

b

Memorie *Beta*

<Join a.2 b.2 where a.1 = b.1>

Si traduce con  
una memoria *Beta*

# Algoritmo *Rete* (C. Forgy, 1980)

- 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 un DB relazionale)

Esempio:

Questa  
combinazione

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

Memorie *Alpha*

<Madre ?1 ?2>

<Padre ?1 ?2>

<Femmina ?1>

a

b

Memorie *Beta*

<Join a.2 b.2 where a.1 = b.1>

Si traduce con  
una memoria *Beta*  
Il join è tra i valori  
delle due prime variabili  
delle memorie *Alpha*

# Algoritmo *Rete* (C. Forgy, 1980)

- Elementi della struttura a rete

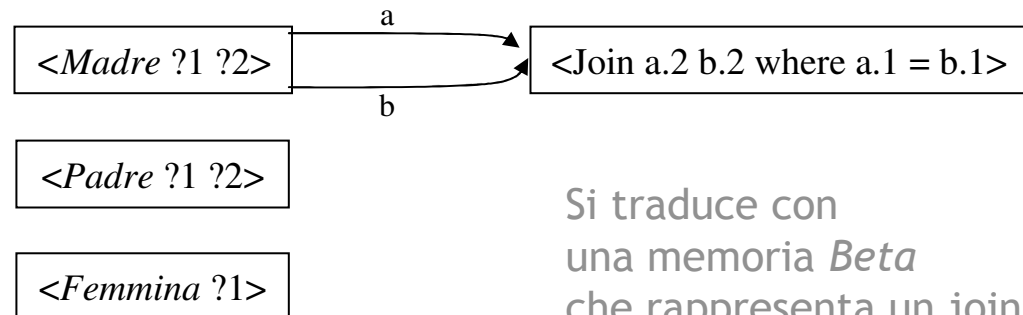
## Memorie *Beta*

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 un DB relazionale)

Esempio:

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

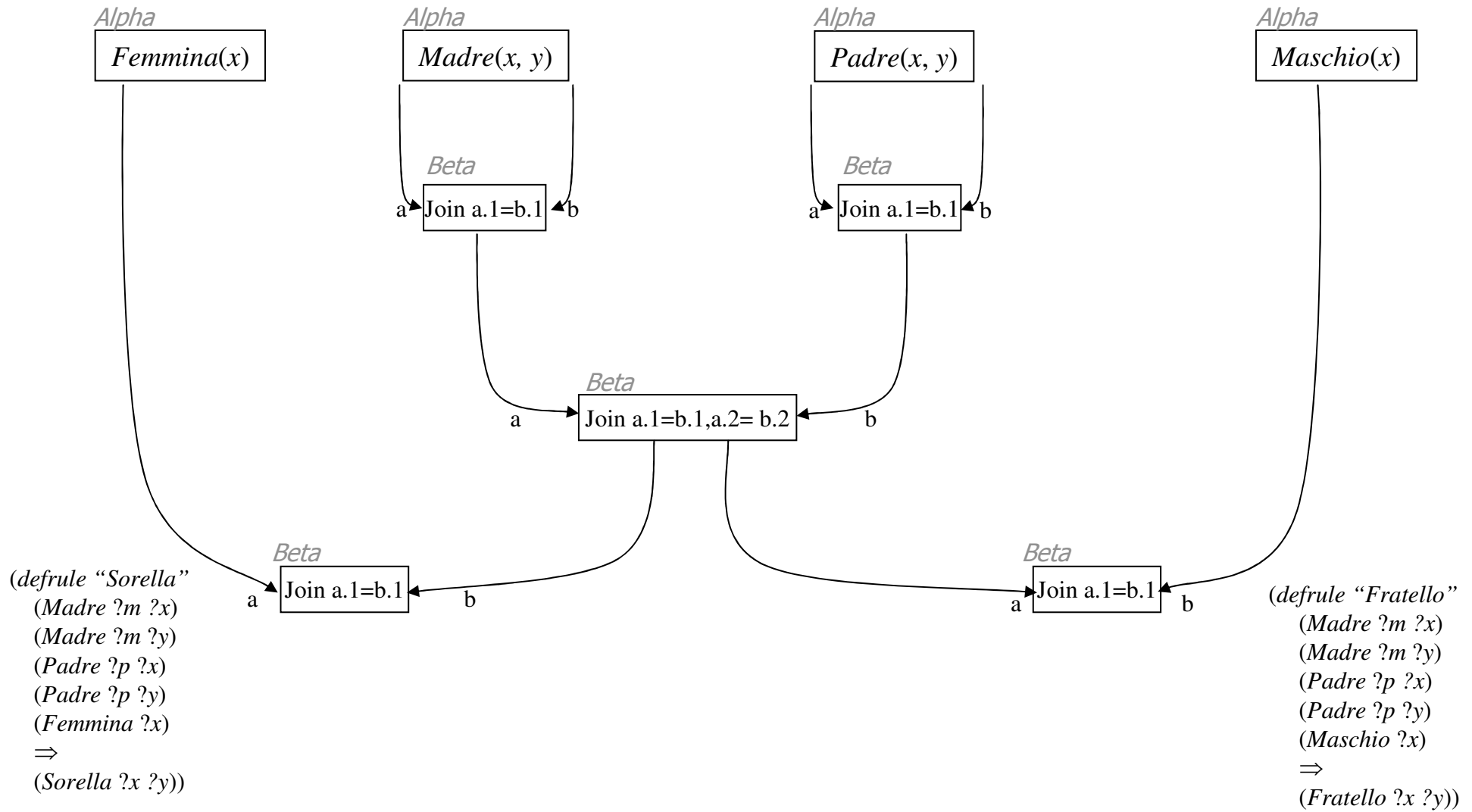
Memorie *Alpha*



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

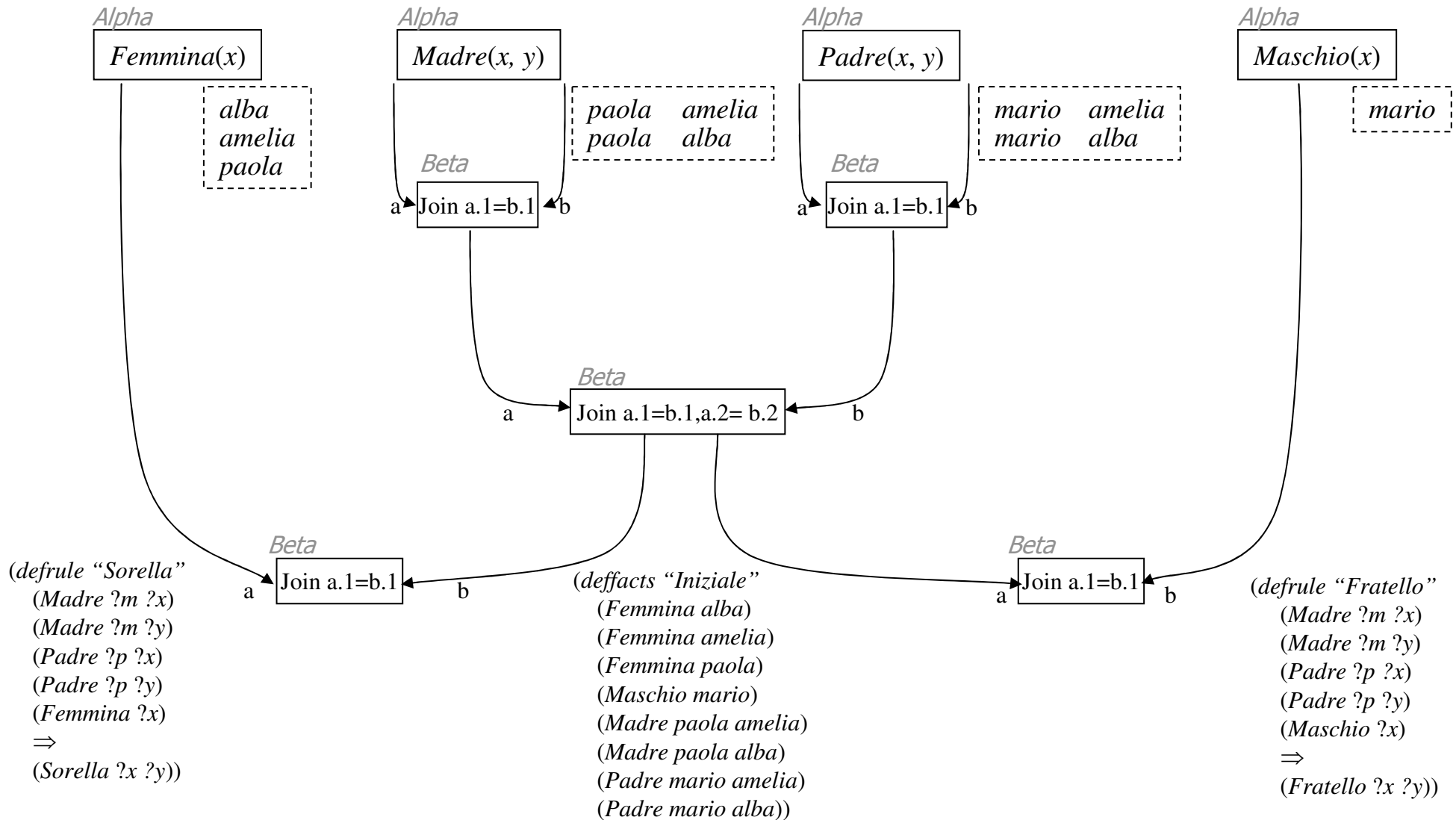
Anche i join possono essere  
fattorizzati: lo stesso join può  
occorrere in più regole

# Algoritmo Rete (C. Forgy, 1980)



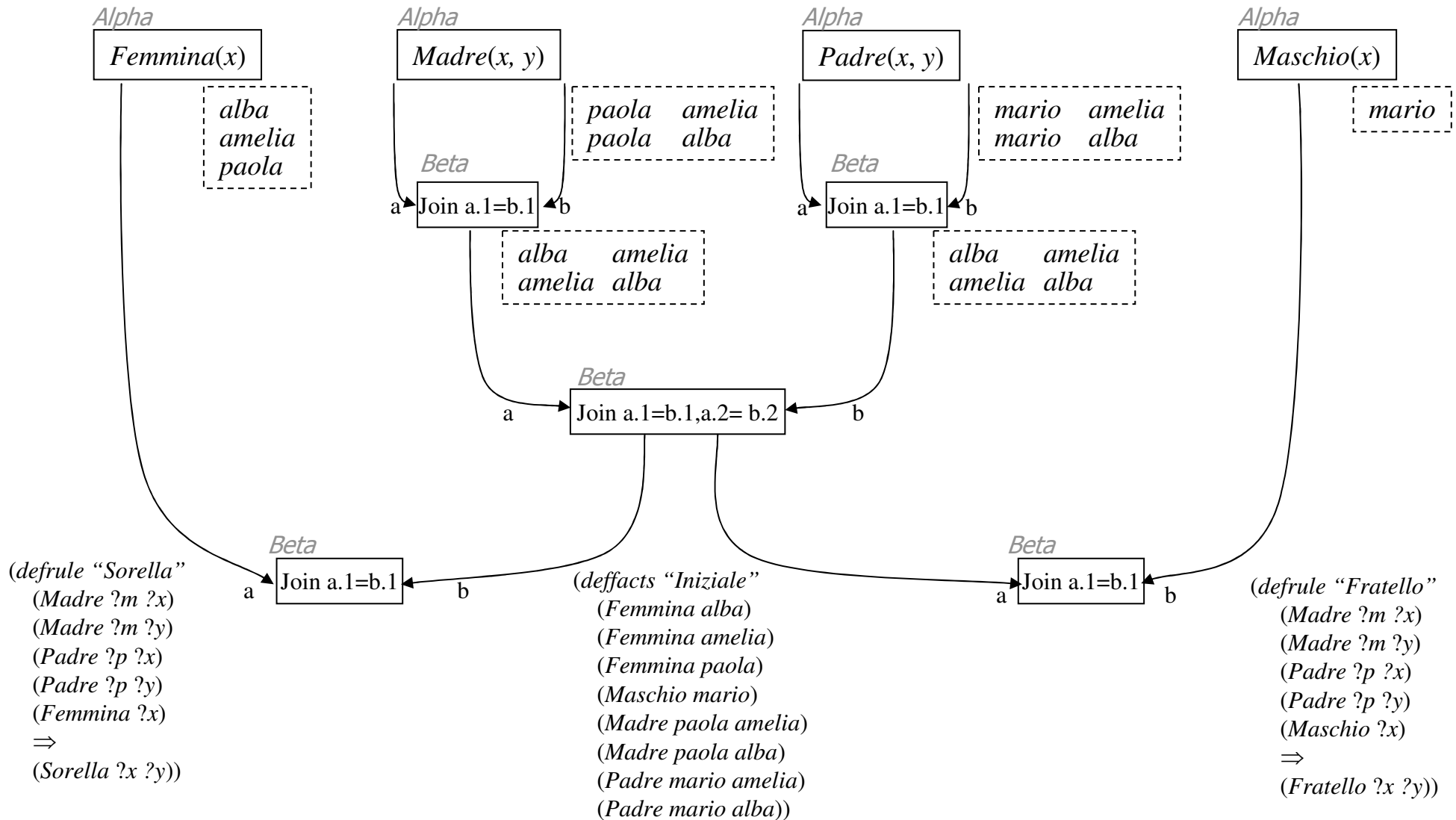
I fatti vengono “agganciati”  
alle memorie *Alpha*  
della struttura *Rete* ...

# Algoritmo *Rete* (C. Forgy, 1980)



... e quindi "fluiscono" attraverso la struttura delle memorie Beta ...

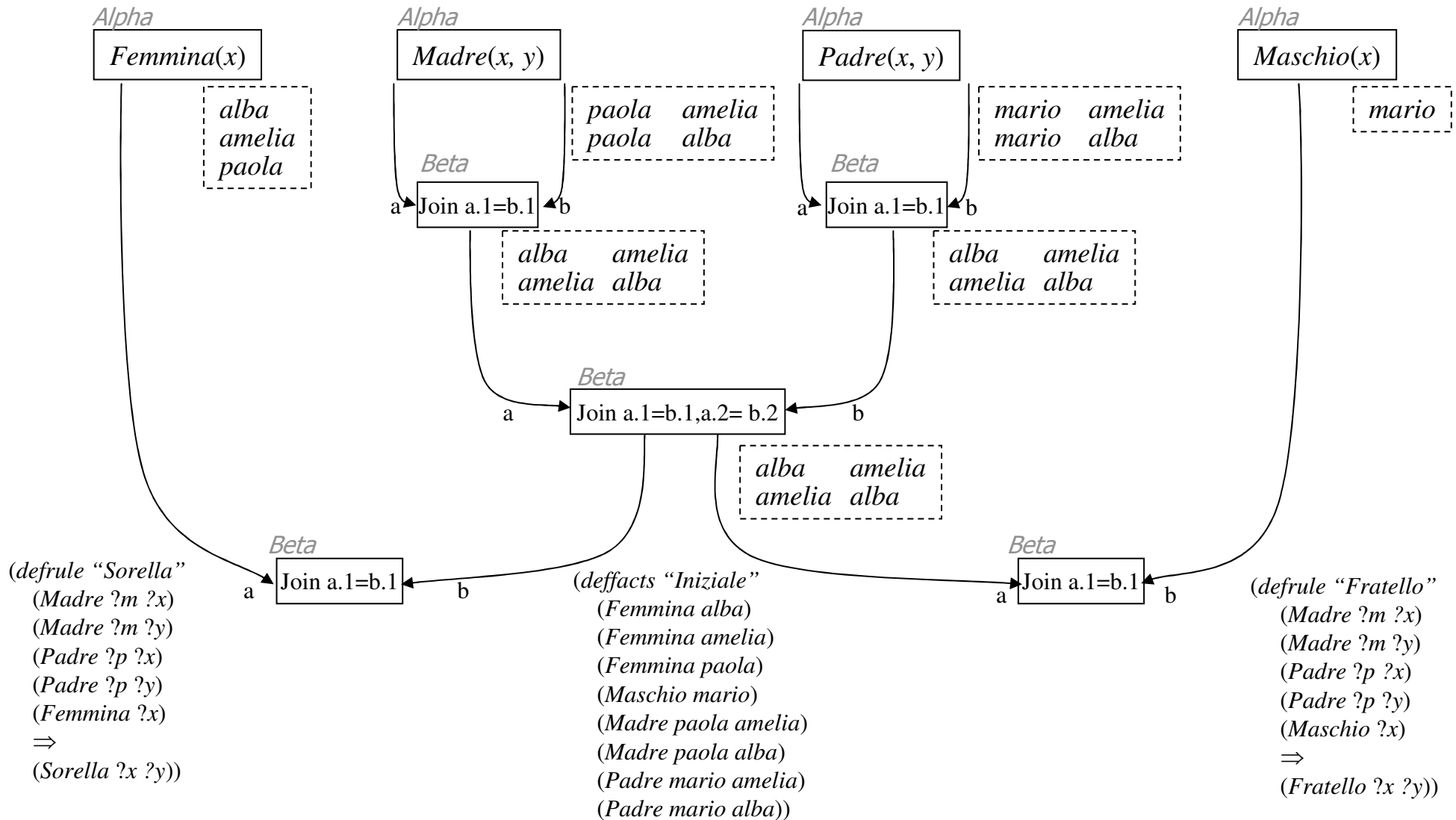
# Algoritmo Rete (C. Forgy, 1980)





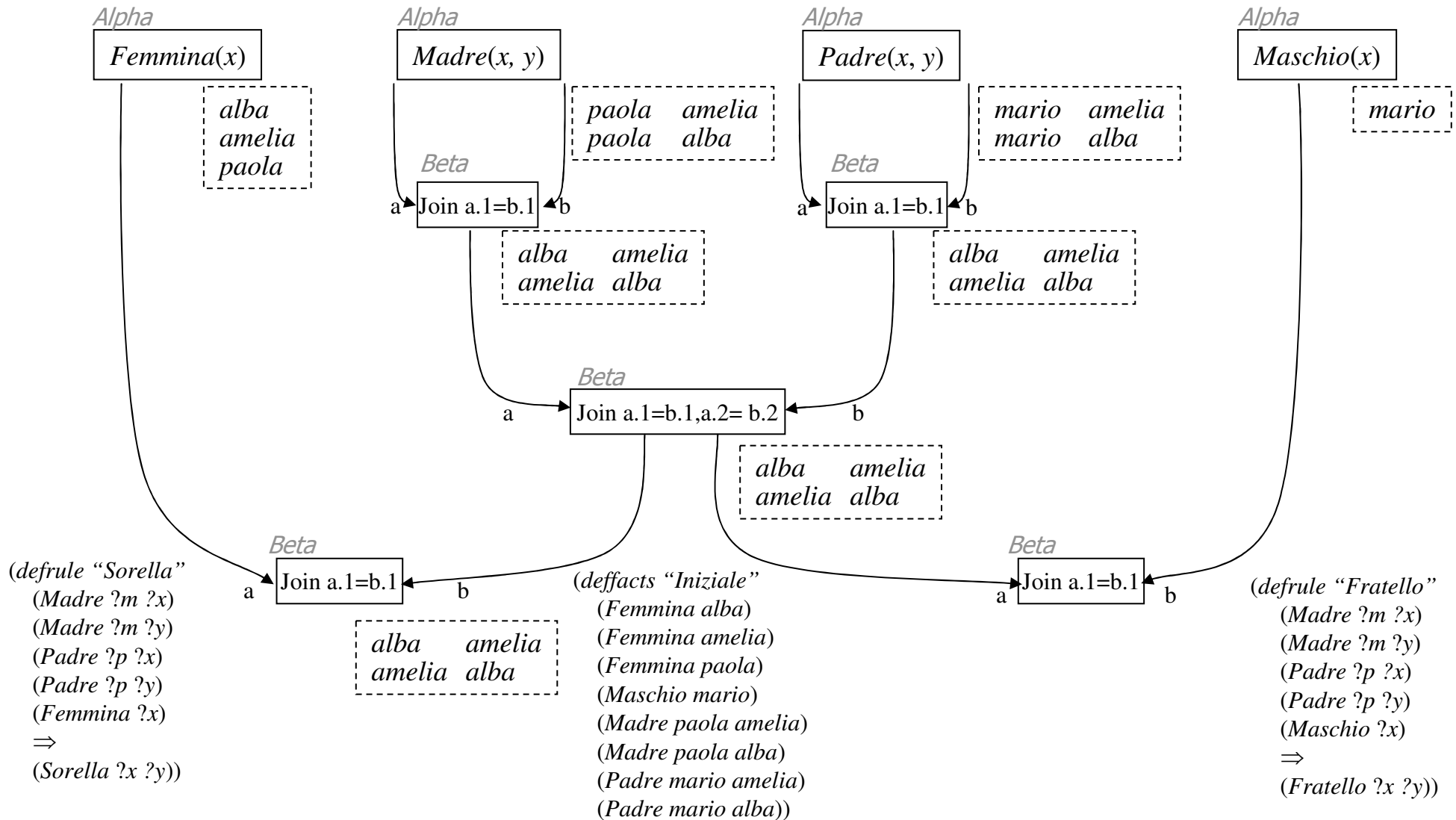
... e quindi “fluiscono”  
attraverso la struttura  
delle memorie *Beta* ...

# Algoritmo *Rete* (C. Forgy, 1980)



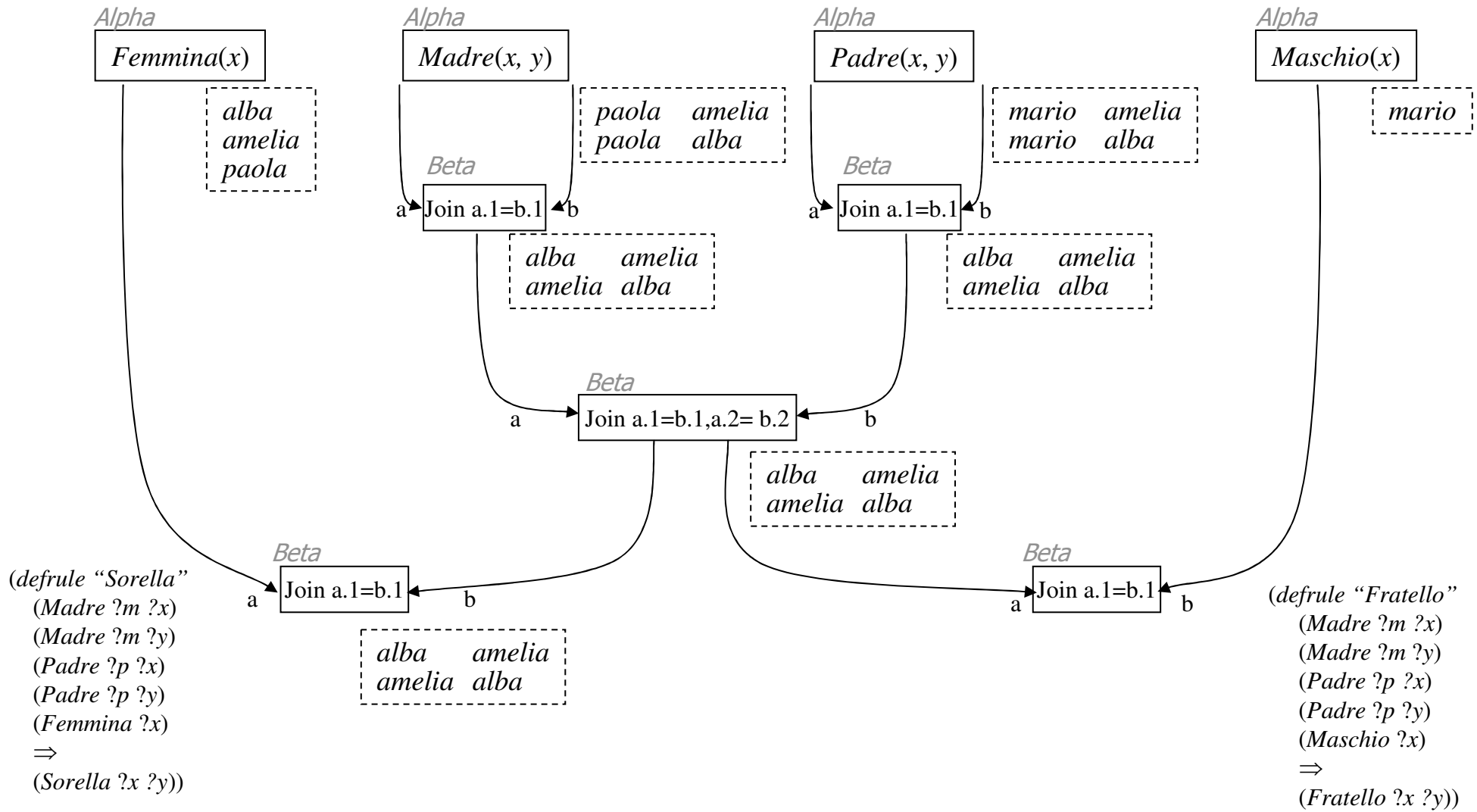
... e quindi "fluiscono" attraverso la struttura delle memorie Beta ...

# Algoritmo Rete (C. Forgy, 1980)



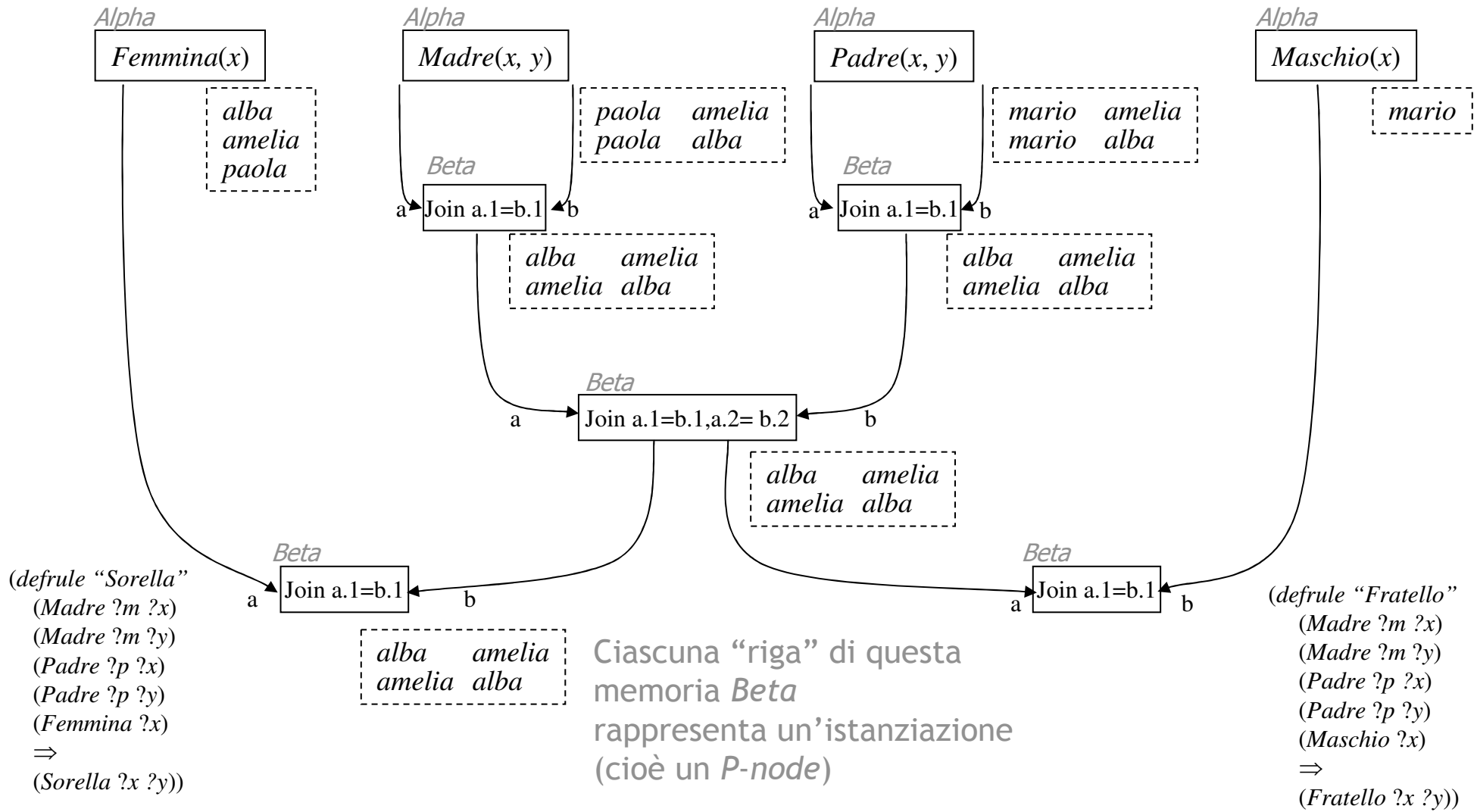
... fino a creare le istanziazioni delle regole

# Algoritmo Rete (C. Forgy, 1980)



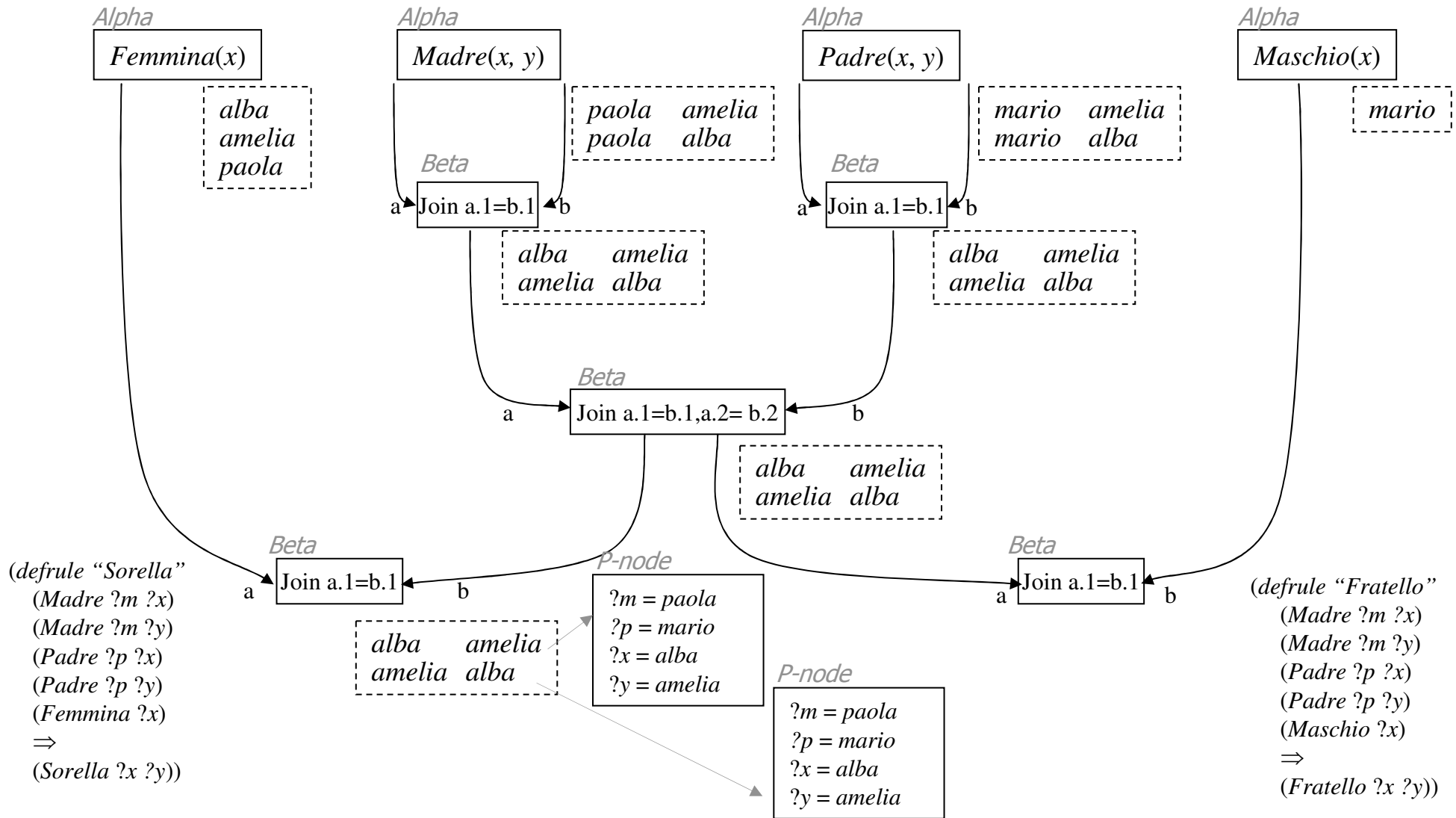
... fino a creare le istanziazioni delle regole

# Algoritmo Rete (C. Forgy, 1980)



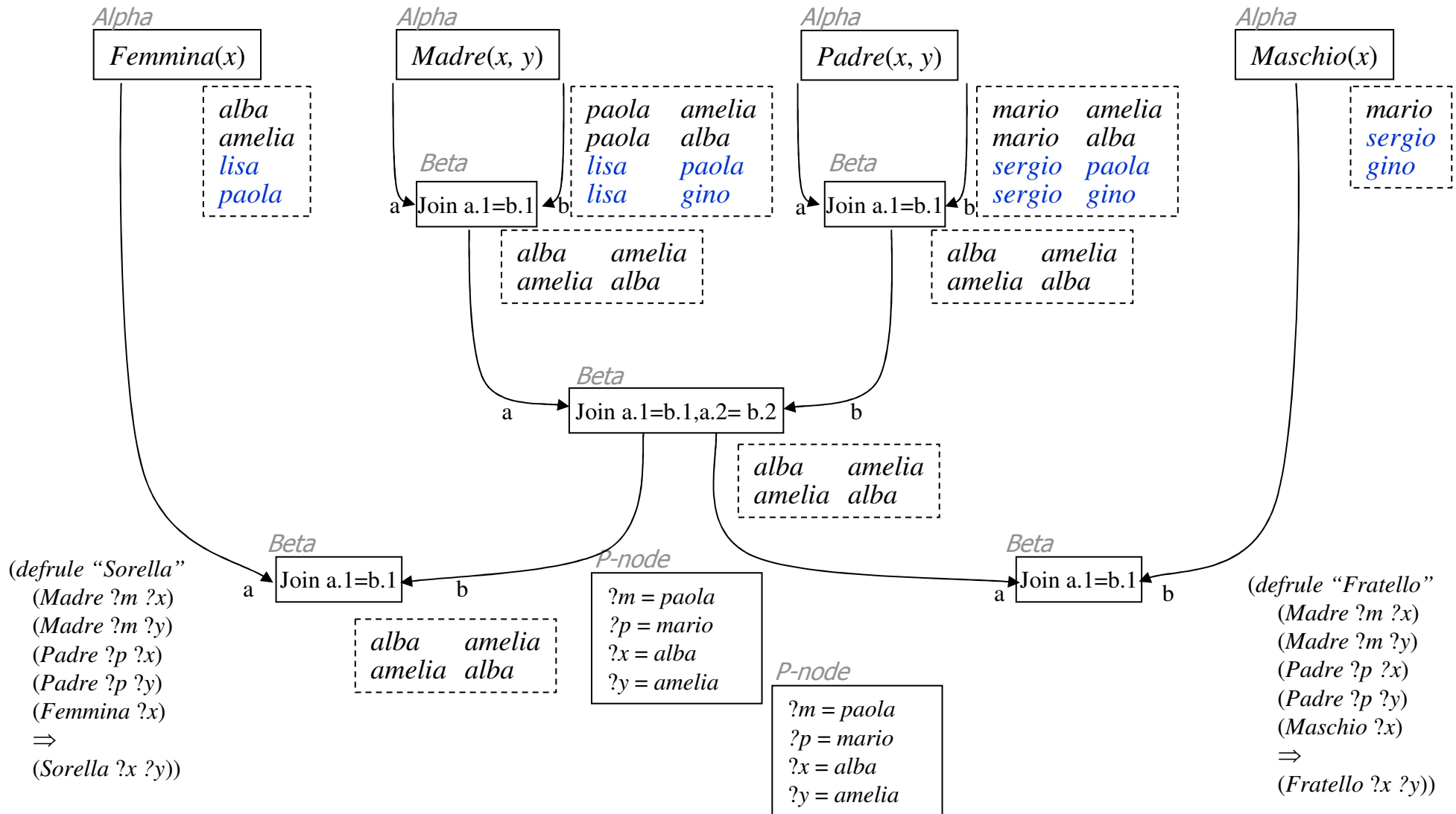
... fino a creare le istanziazioni delle regole

# Algoritmo Rete (C. Forgy, 1980)



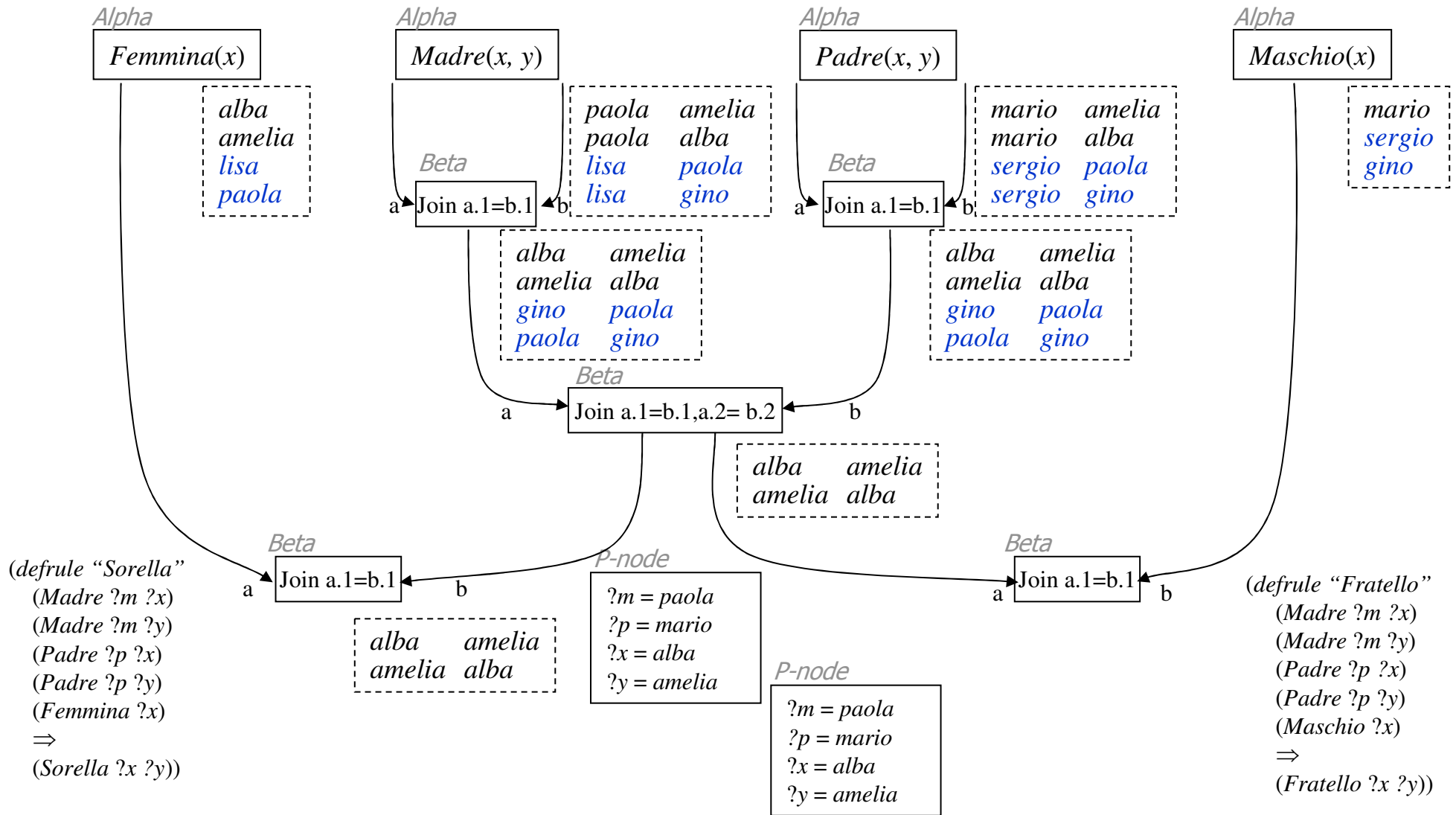
Nuovi fatti “fluiscono”  
attraverso la struttura Rete ...

# Algoritmo Rete (C. Forgy, 1980)



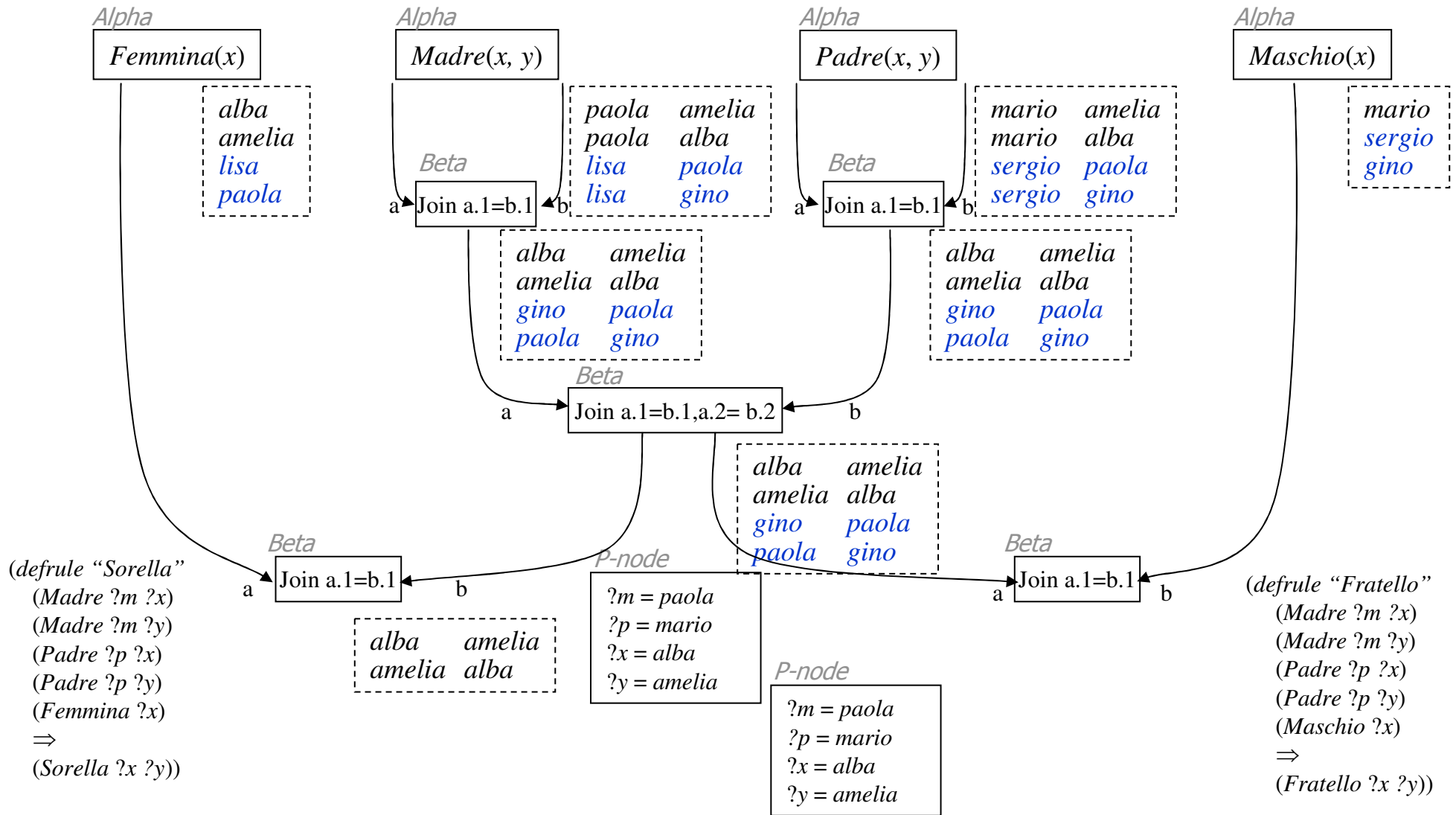
Nuovi fatti “fluiscono”  
attraverso la struttura Rete ...

# Algoritmo Rete (C. Forgy, 1980)



Nuovi fatti “fluiscono”  
attraverso la struttura Rete ...

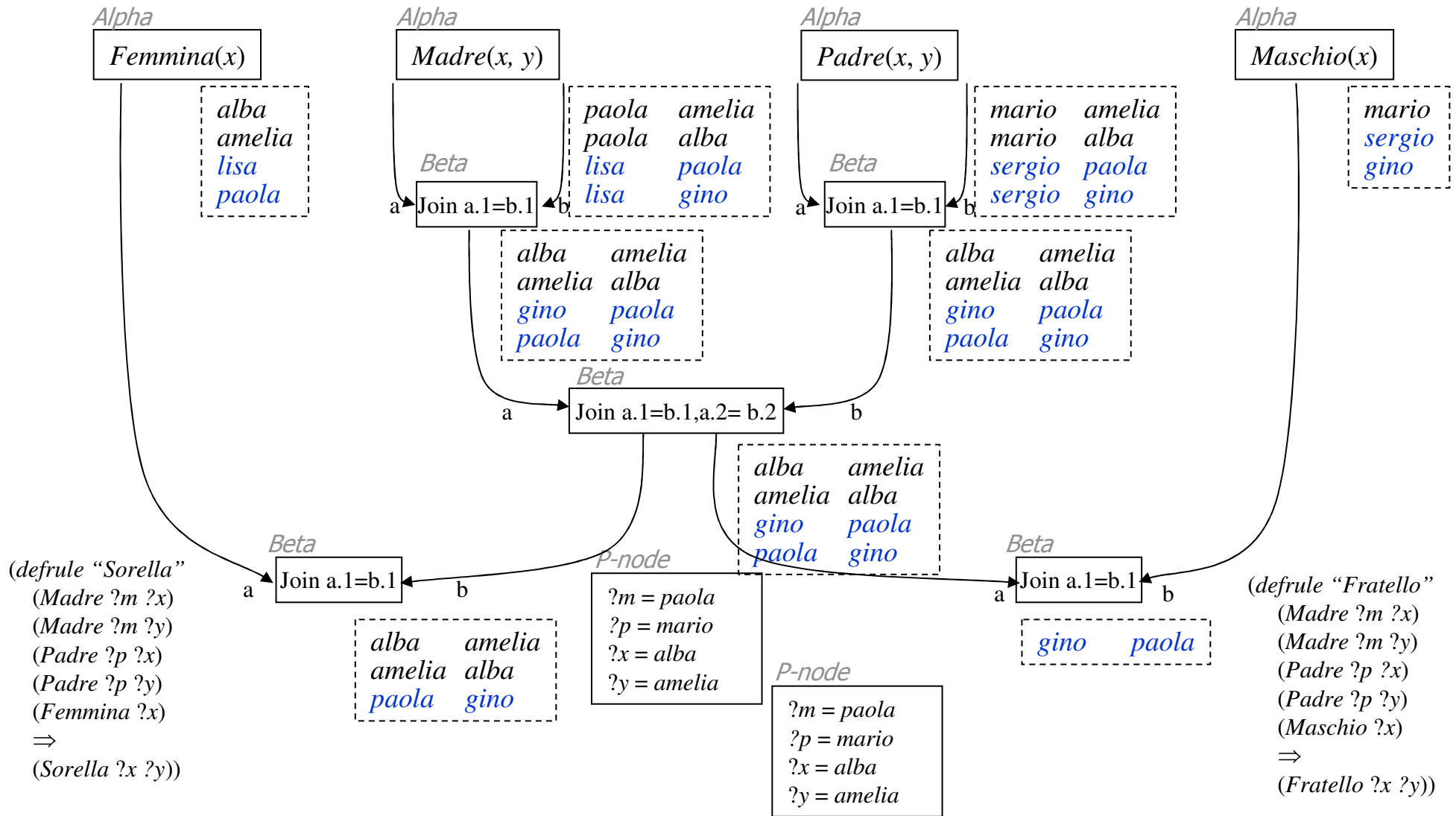
# Algoritmo Rete (C. Forgy, 1980)





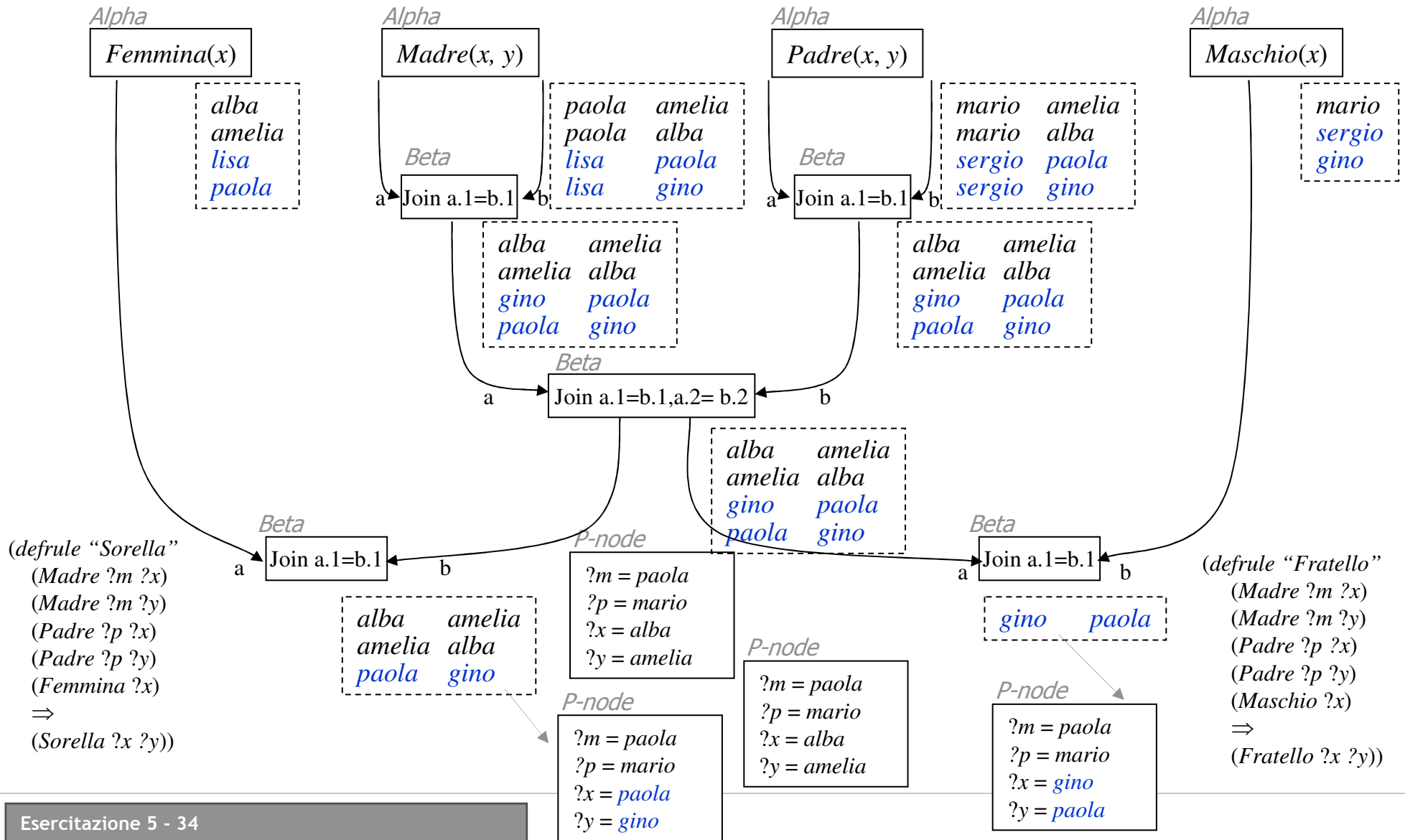
Nuovi fatti “fluiscono”  
attraverso la struttura Rete ...

# Algoritmo Rete (C. Forgy, 1980)



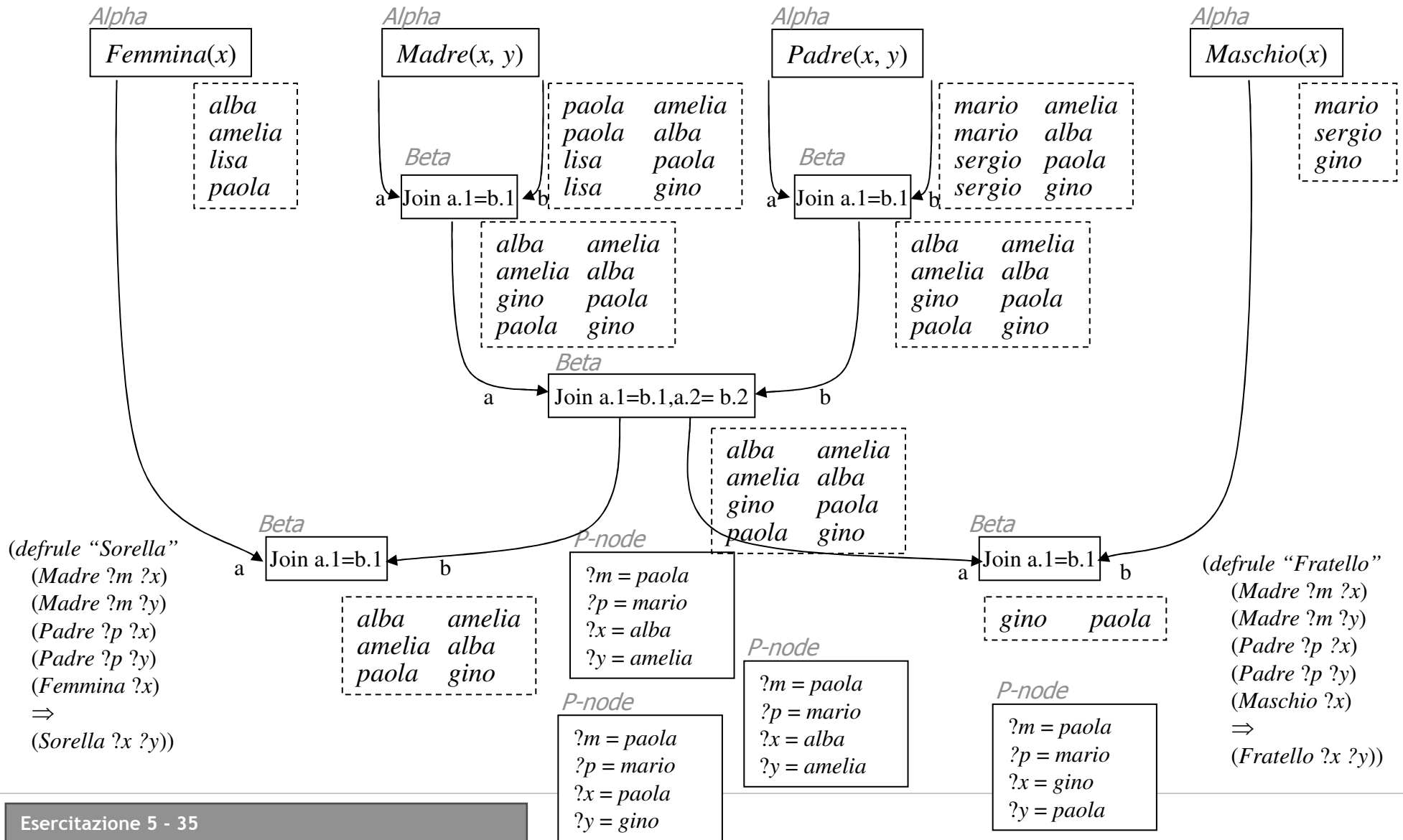
... e creano nuovi P-node

# Algoritmo Rete (C. Forgy, 1980)



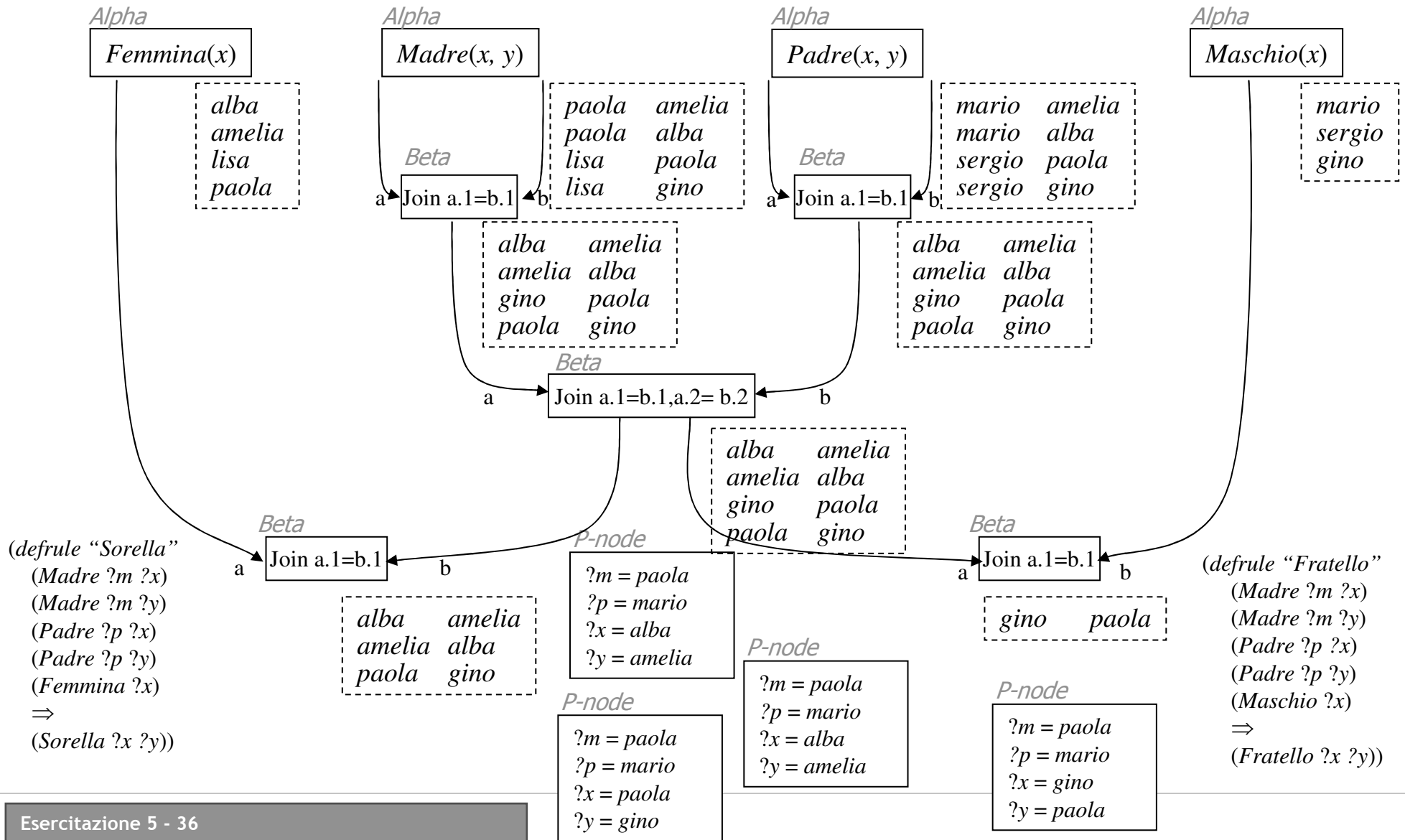
I *P-node* vengono creati una volta sola: quando si aggiorna il contenuto delle memorie *Alpha* e *Beta*

# Algoritmo Rete (C. Forgy, 1980)



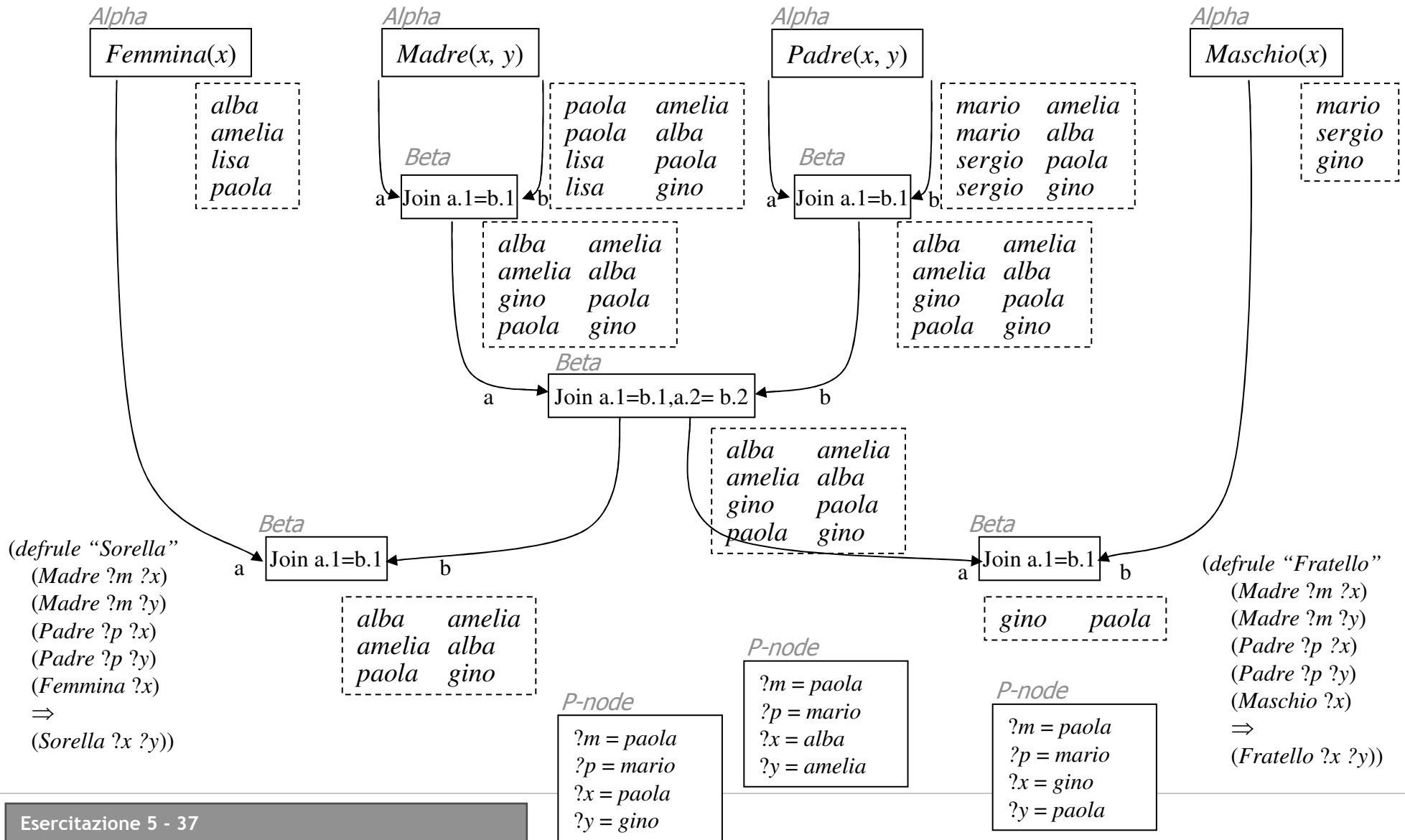
# Algoritmo Rete (C. Forgy, 1980)

I *P-node* mantengono il legame con le regole associate e vengono rimossi quando la regola è eseguita (*FIRE*)



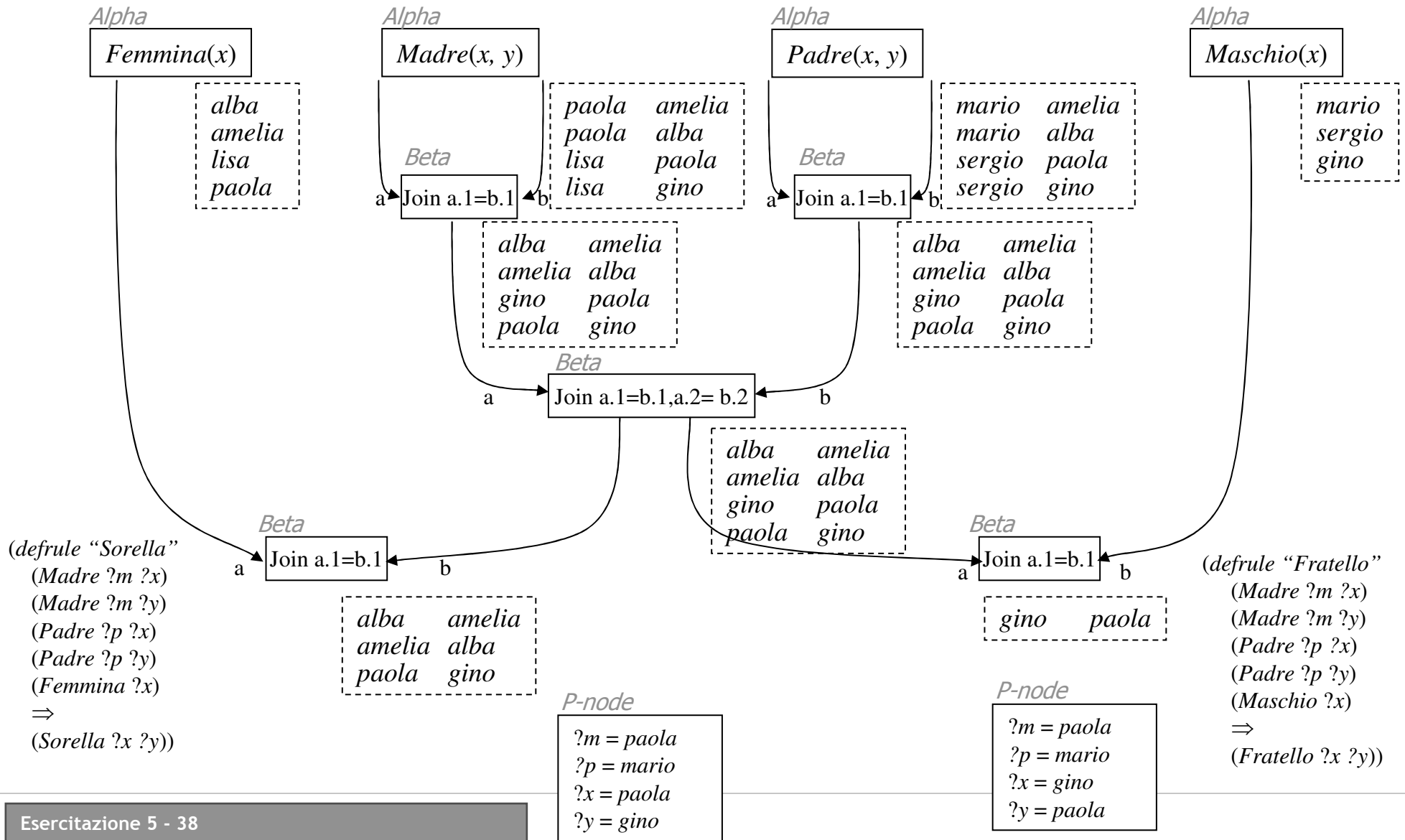
# Algoritmo Rete (C. Forgy, 1980)

I *P-node* mantengono il legame con le regole associate e vengono rimossi quando la regola è eseguita (*FIRE*)



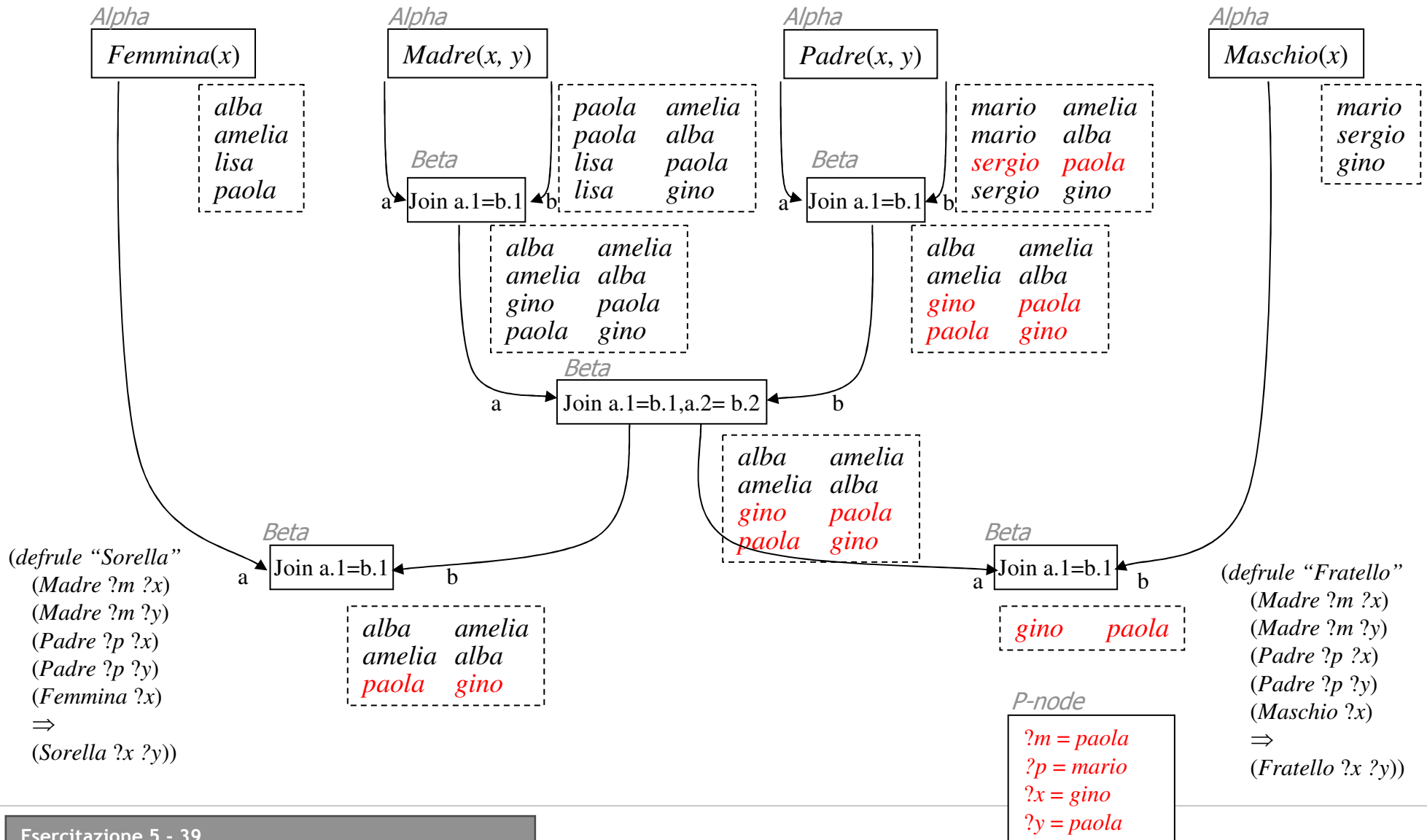
# Algoritmo Rete (C. Forgy, 1980)

I *P-node* mantengono il legame con le regole associate e vengono rimossi quando la regola è eseguita (*FIRE*)



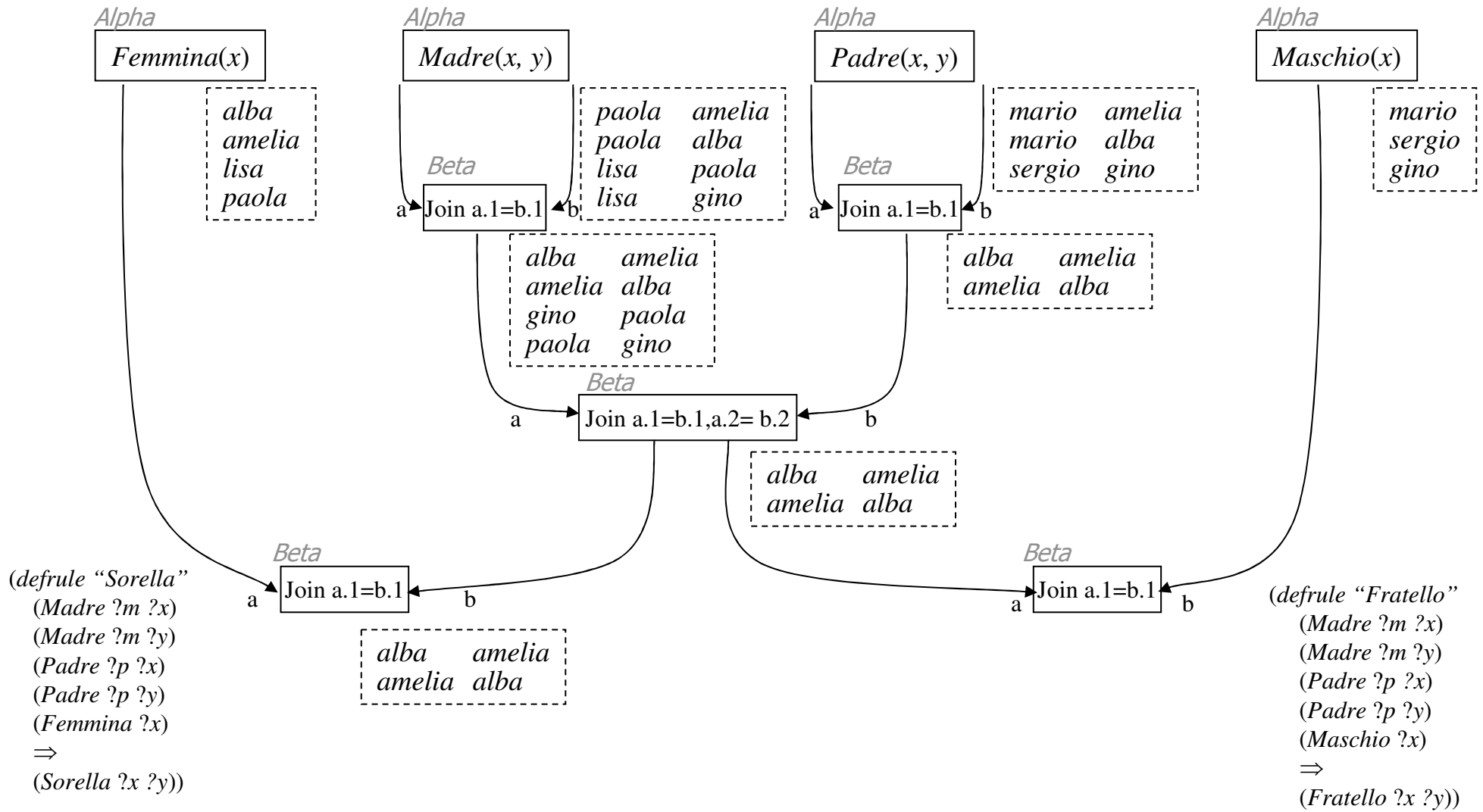
I P-node vengono rimossi anche quando viene ritrattato un fatto che li giustifica

# Algoritmo Rete (C. Forgy, 1980)



I P-node vengono rimossi anche quando viene ritrattato un fatto che li giustifica

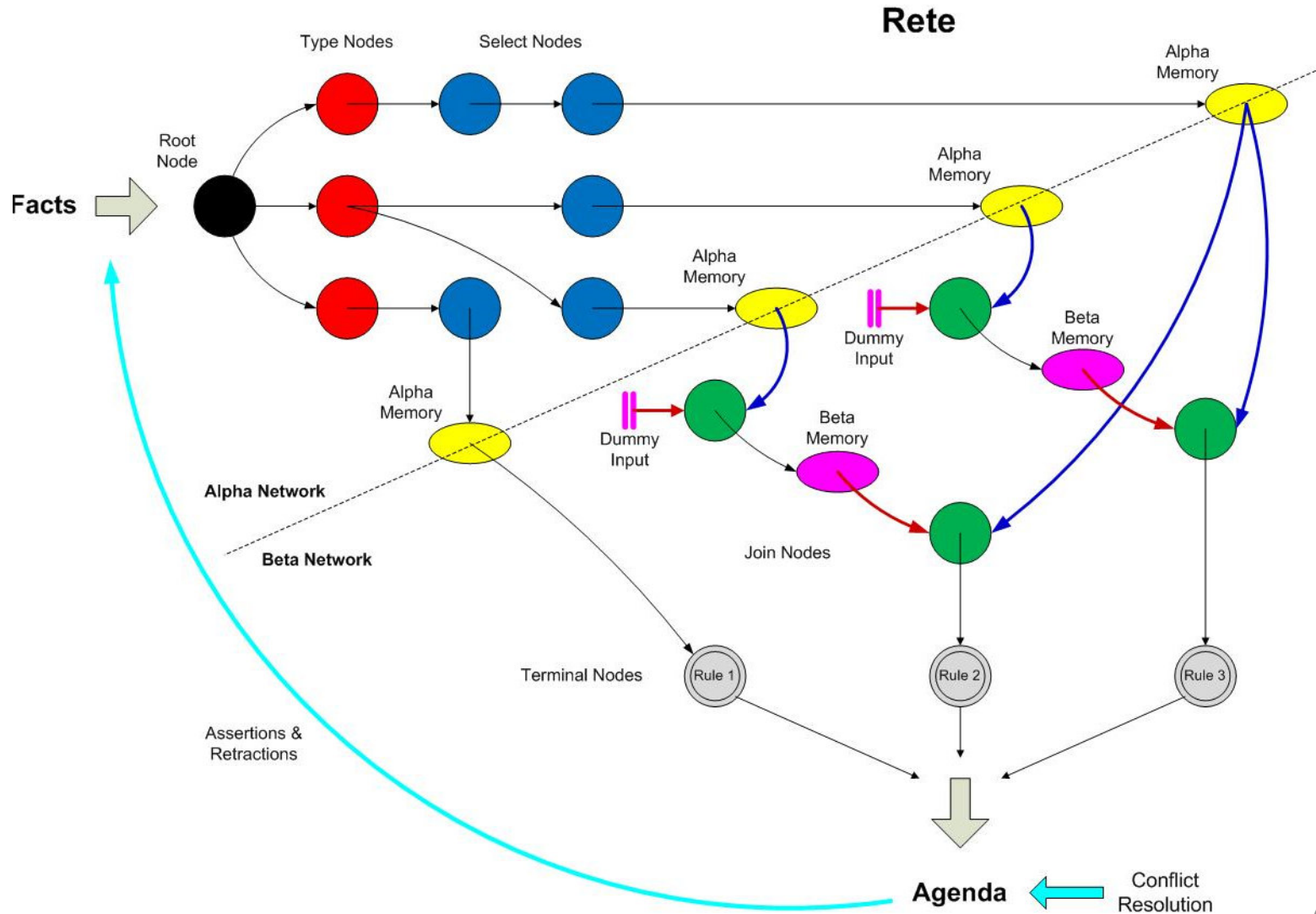
# Algoritmo Rete (C. Forgy, 1980)





I *P-node* vengono rimossi anche quando viene ritrattato un fatto che li giustifica

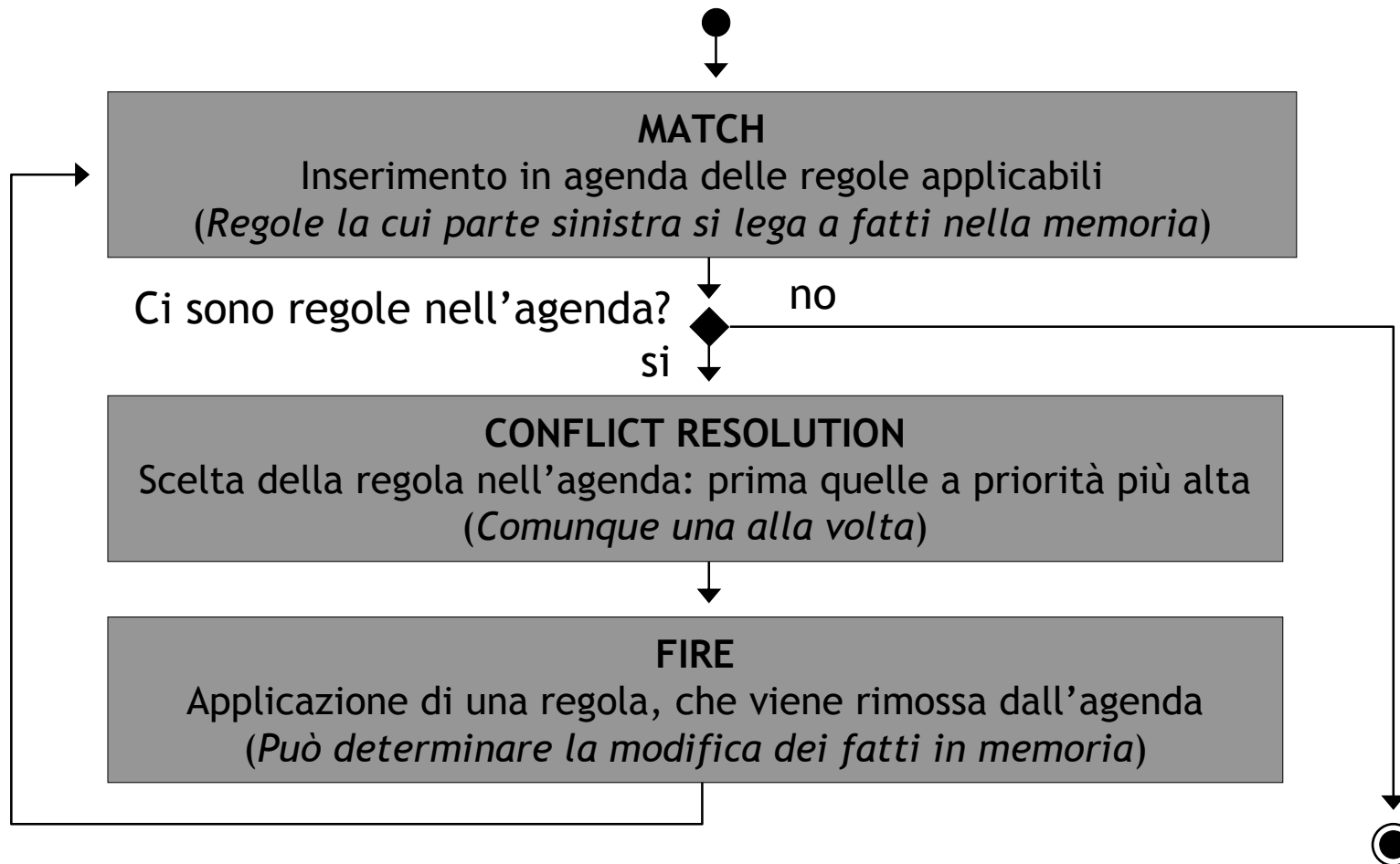
# Algoritmo Rete (C. Forgy, 1980)



(Da wikipedia)

## Come funziona *Jess* (approssimazione)

- L'agenda contiene le regole applicabili



# Come funziona *Jess*

BATCH: caricamento delle regole e costruzione della struttura *Rete*  
RESET: azzeramento delle memorie *Alpha* e *Beta*

