



Università di Bergamo
Facoltà di Ingegneria

Intelligenza Artificiale

Paolo Salvaneschi

B1_1 V1.3

Sistemi a regole

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

- Tecnologie per l'inferenza logica
- Programmazione logica
- Sistemi di produzioni (sistemi a regole)
- Architettura di un sistema a regole
- Forward e backward chaining
- Sviluppo di sistemi a regole

- La logica:
 - sistema per la rappresentazione formale del ragionamento
 - formalismo di rappresentazione e regole di derivazione sintattica (regole di inferenza)

- Utilizzando una logica scriviamo:
 - Una base di conoscenza
 - Un insieme di fatti
- Ci si può chiedere se, dato un nuovo fatto, questo sia una conseguenza logica rispetto alla base di conoscenza ed ai fatti preesistenti
- Ci si può chiedere quali nuovi fatti possono essere conseguenze logiche rispetto alla base di conoscenza ed ai fatti preesistenti

- Tecnologia dei sistemi basati sulla rappresentazione della conoscenza e sull'inferenza logica
 - La logica del primo ordine e l'inferenza in FOL sono la base per la realizzazione delle relative tecnologie
 - Sono aggiunti altri aspetti (es. funzioni di calcolo o input/output)
 - Tecnologie:
 - Programmazione logica (Prolog)
 - Sistemi di produzioni (sistemi a regole)

- Programmazione logica (Prolog)
 - Backward chaining (dept first,backtracking)
 - Funzioni di calcolo e input/output
 - Modifica della KB (assert, retract)
 - Negation as failure ($\neg P$ è provato se il sistema fallisce nel provare P)

Sistemi di produzioni (sistemi a regole)

- Sistemi di produzioni (sistemi a regole)
(XCON configuratore computer DEC; sistemi esperti; linguaggio OPS-5)
 - Forward chaining
 - Forward chaining efficiente (algoritmo RETE)
 - Funzioni di calcolo e input/output
 - Gestione della propagazione di fatti irrilevanti
 - Utilizzo di algoritmi backward chaining
 - Selezione delle regole da attivare (conflict set)

Sistemi di produzioni (sistemi a regole)

KB Base di conoscenza

Motore inferenziale

Deriva ciò che è implicato logicamente
dalla base di conoscenza KB

Insieme di regole del tipo:

if <condizione> *then* <conclusione/azione>

Antecedente

Consequente

Sistemi di produzioni (sistemi a regole)

- Esempio di regole

if <condizione> then <conclusione/azione>

madre(x) = madre(y)

padre(x) = padre(y)

Maschio(x)

\Rightarrow

Fratello(x, y)

madre(x) = madre(y)

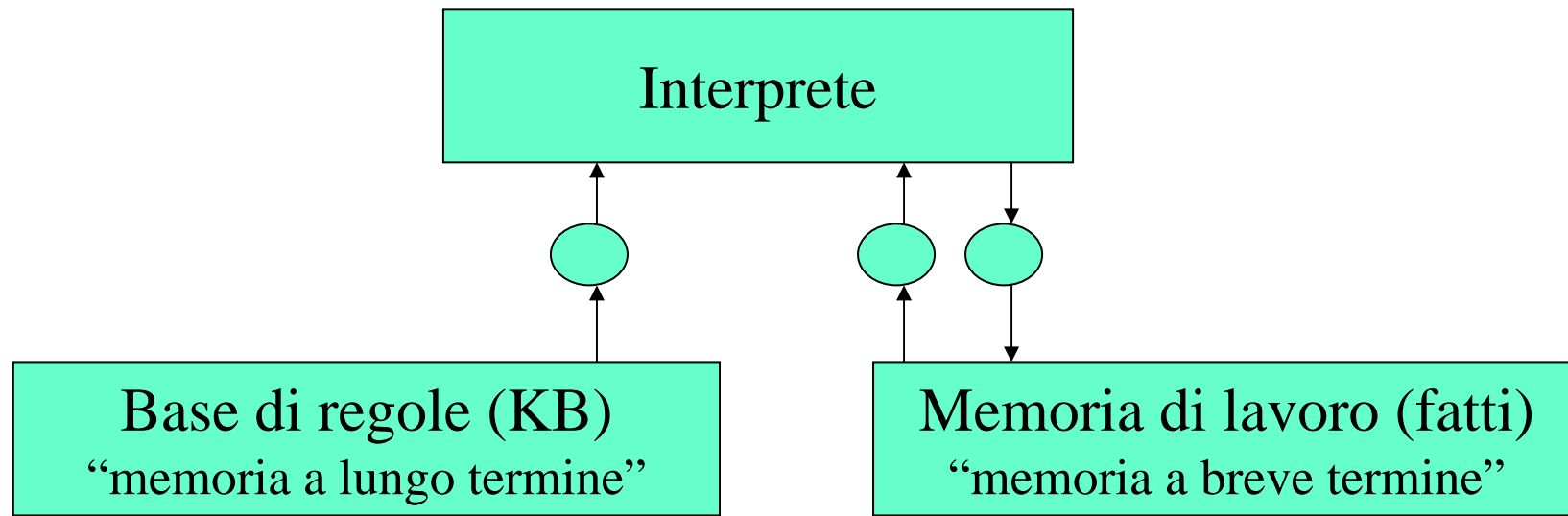
padre(x) = padre(y)

Femmina(x)

\Rightarrow

Sorella(x, y)

Architettura di un sistema a regole

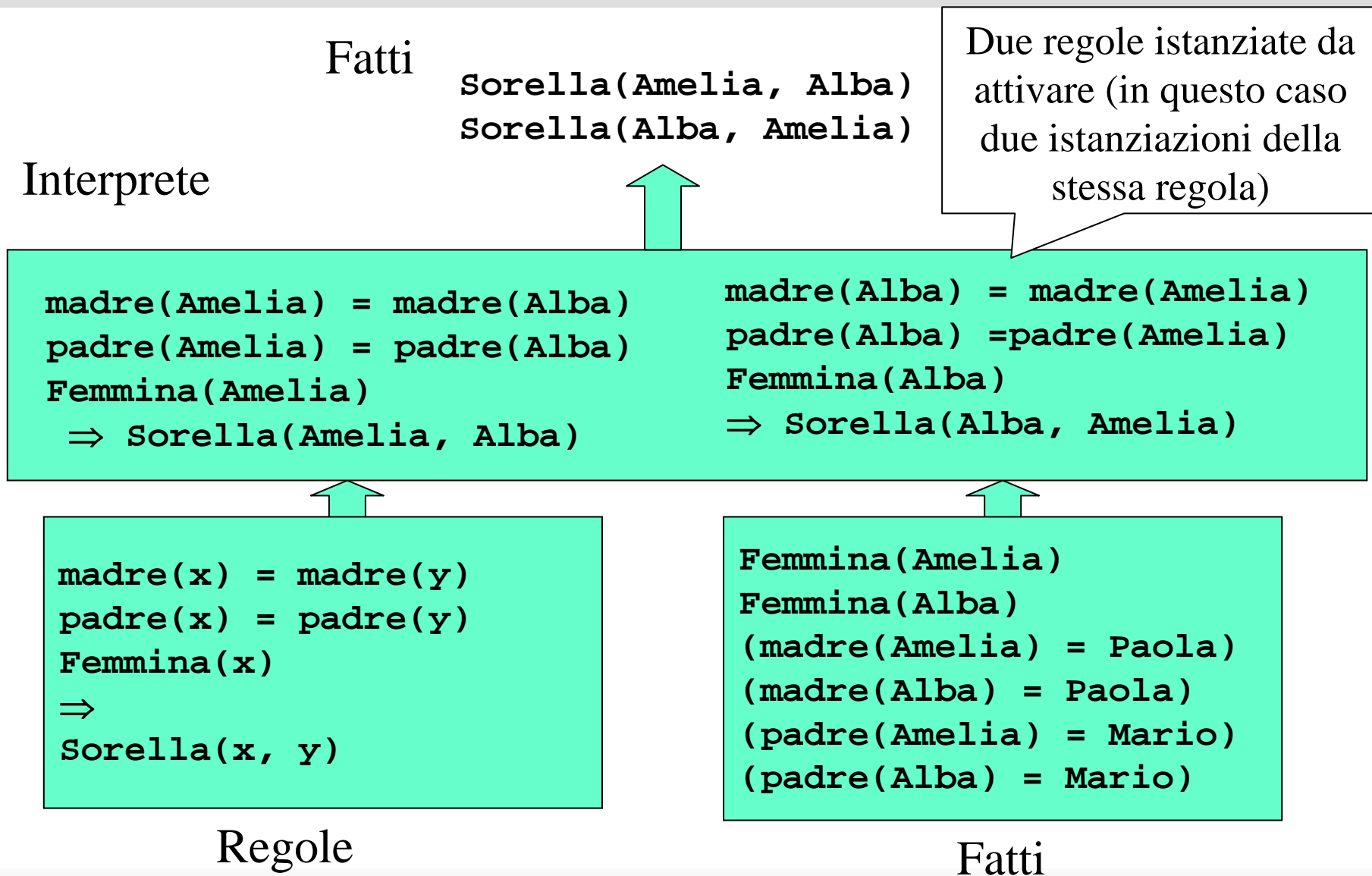


- **Base di regole** Contiene tutte le regole di produzione
- **Memoria di lavoro** Memorizza lo stato raggiunto dal processo di risoluzione di un problema – i fatti conosciuti
- **Interprete** Applica un meccanismo di inferenza alle regole ed ai fatti per risolvere il problema

Architettura di un sistema a regole

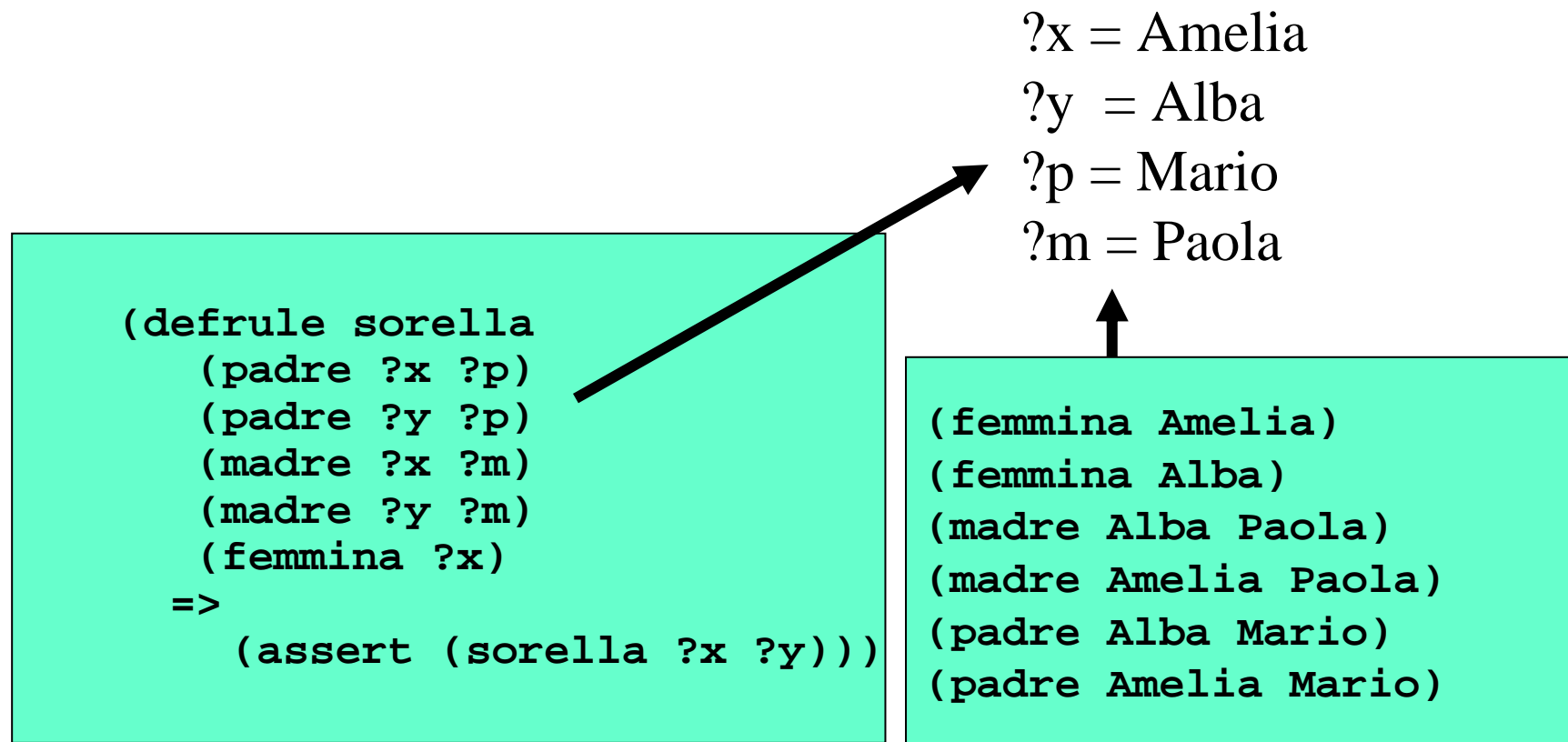
- Memoria di lavoro
 - La memoria di lavoro contiene la rappresentazione dei fatti specifici
 - Le regole operano sulla memoria di lavoro
 - Le condizioni sono istanziate sulla base dei fatti specifici
 - Le azioni comportano tipicamente l'asserzione o la ritrattazione di fatti specifici nella memoria di lavoro (oltre ad altro, es. output)

Sistemi di produzioni (sistemi a regole)



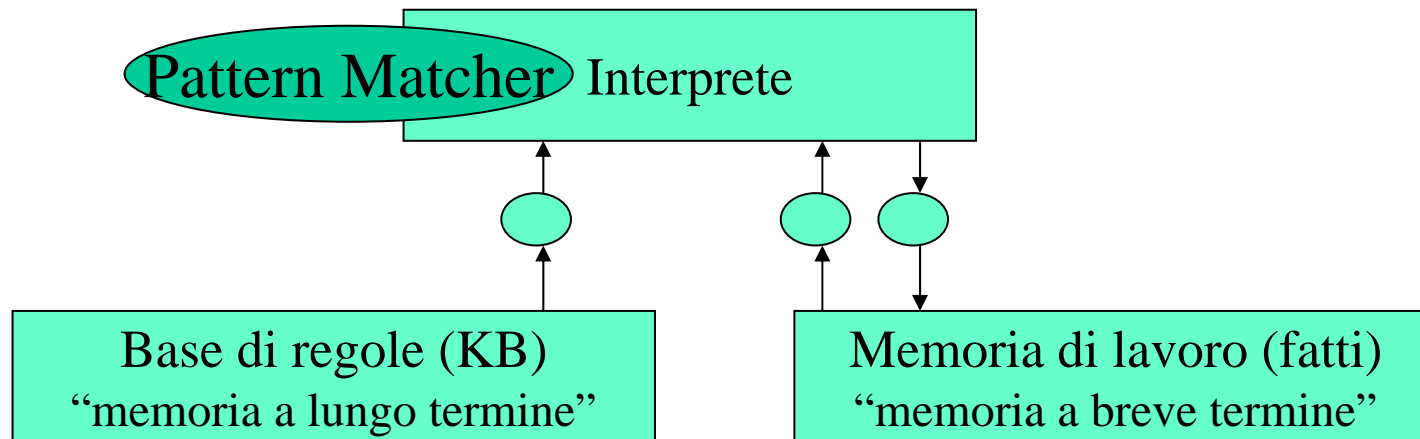
Sistemi di produzioni (sistemi a regole)

- Regole e fatti nella sintassi di JESS



Architettura di un sistema a regole

- Pattern matcher
 - determina l'insieme delle regole potenzialmente eseguibili (nel caso di forward chaining: le regole il cui antecedente è soddisfatto dai fatti contenuti nella memoria di lavoro)



Architettura di un sistema a regole

- Pattern matcher

```
(defrule NomeDellaRegola "commento"  
(pattern_1)  
(pattern_2)  
...  
(pattern_N)  
=>  
(action_1)  
...  
(action_M)  
)
```

Fatti

```
(padre mario luca)  
(padre luca alberto)
```

Regole

```
(defrule R_nonno "regola che  
trova i nonni"  
(padre ?x ?y)  
(padre ?y ?z)  
=>  
(assert (nonno ?x ?z))  
)
```

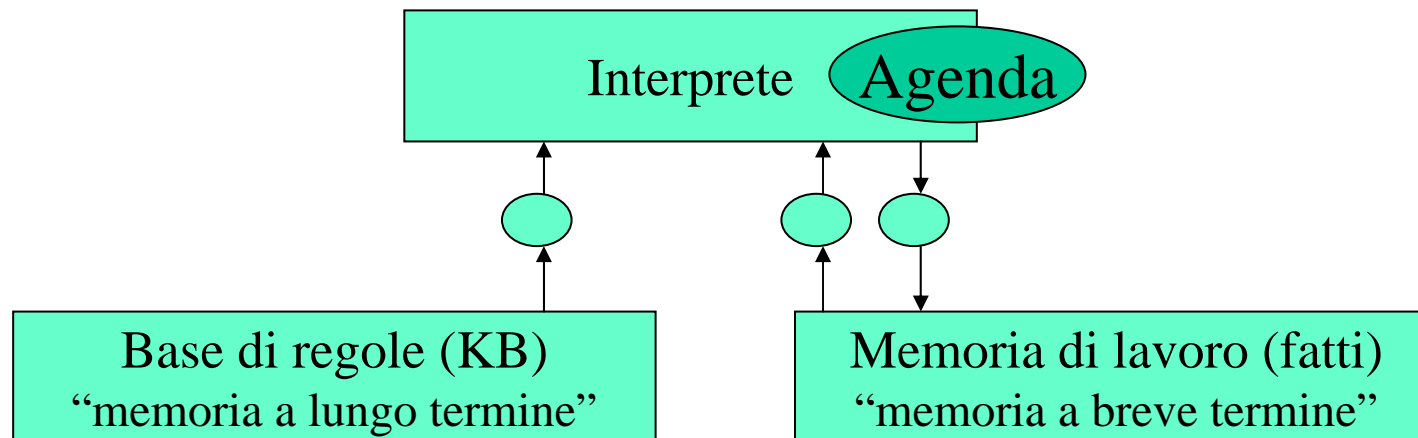
Assegnazioni

$?x = \text{mario}$ $?y = \text{luca}$ $?z = \text{alberto}$

Fatto aggiunto dopo l'esecuzione della regola istanziata
(nonno mario alberto)

Architettura di un sistema a regole

- Agenda
 - Il sistema mantiene un'agenda delle regole istanziate
 - Da cui seleziona le regole da attivare che producono nuovi fatti nella memoria di lavoro



Architettura di un sistema a regole

- Ciclo di esecuzione (fino a quando l'agenda è vuota)
 - determina l'insieme delle regole potenzialmente eseguibili
 - aggiorna l'agenda
 - sceglie ed attiva una regola
 - aggiorna la memoria di lavoro

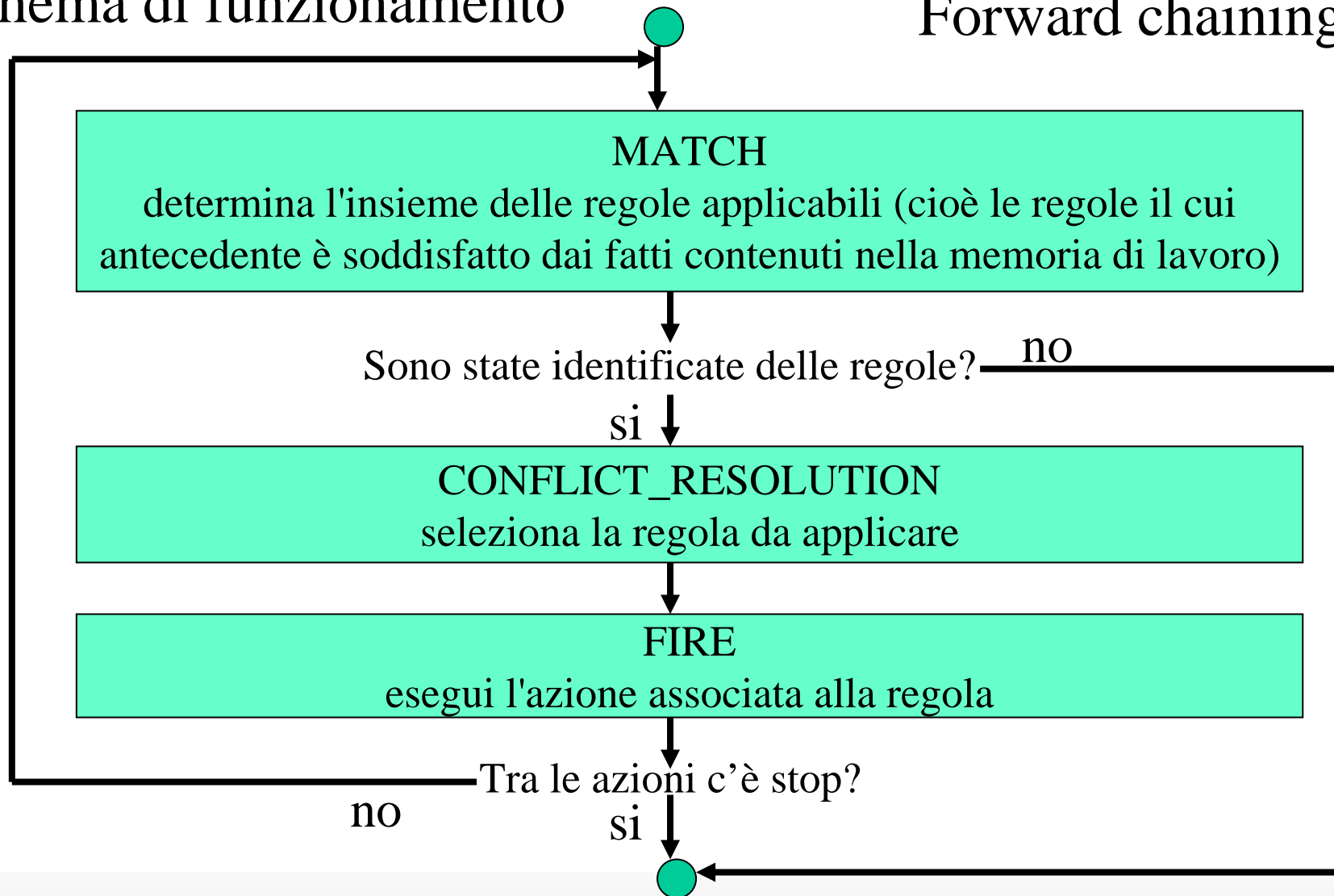
Architettura di un sistema a regole

- Problemi:
 - Evitare i cicli infiniti riattivando istanziazioni di regole già attivate (le regole devono essere inserite nell'agenda solo in presenza di fatti nuovi)
 - Inefficienza del confronto diretto, ad ogni ciclo, di tutte le regole con tutti i fatti (algoritmi efficienti: RETE)

Architettura di un sistema a regole

- Schema di funzionamento

Forward chaining



Architettura di un sistema a regole

- Match
 - Unificazione / algoritmi efficienti (RETE): determina l'insieme delle regole il cui antecedente è soddisfatto dai fatti contenuti nella memoria di lavoro
 - Produce un insieme di regole potenzialmente attivabili
 - Possono essere individuate più regole attivabili
 - Una regola può dar luogo anche a più istanziazioni attivabili
- Fire
 - Esegue il conseguente della regola

Architettura di un sistema a regole

- Conflict resolution
 - Sono potenzialmente eseguibili più regole
- Prima soluzione: sono eseguite tutte
Significato: esegue tutte le inferenze basate su fatti conosciuti anche se sono irrilevanti per gli scopi che si vogliono raggiungere
- Seconda soluzione: si adotta una strategia di controllo (un criterio per scegliere quale delle regole potenzialmente eseguibili - conflict set – deve essere effettivamente eseguita)

Architettura di un sistema a regole

- Conflict resolution

- Nessuna duplicazione.

Non si esegue la stessa regola sullo stesso argomento due volte

- Accensione in ordine di attivazione

La regola attivata più di recente è accesa per ultima (FIFO, Breadth First)

- Vicinanza temporale

La regola attivata più di recente è accesa per prima (LIFO, Depth First)

Architettura di un sistema a regole

- Conflict resolution

- Specificità

Si preferiscono regole più specifiche

$Mammifero(x) \Rightarrow \text{add Gambe}(x, 4)$

→ $Mammifero(x) \wedge Umano(x) \Rightarrow \text{add Gambe}(x, 2)$

- Priorità

Si preferiscono regole con priorità più alta assegnata esplicitamente

$PannelloDiControllo(p) \wedge Polveroso(p) \Rightarrow \text{Azione}(Polvere(p))$

→ $PannelloDiControllo(p) \wedge SpiaFusioneAccesa(p) \Rightarrow \text{Azione}(Evacuazione)$

Architettura di un sistema a regole

- Strategie di esplorazione della base di regole
 - Forward chaining
(in avanti) o controllo guidato dai dati
 - Backward chaining
(all'indietro) o controllo guidato dal goal G.

Forward e backward chaining

Strategia **forward** (in avanti) o controllo guidato dai dati

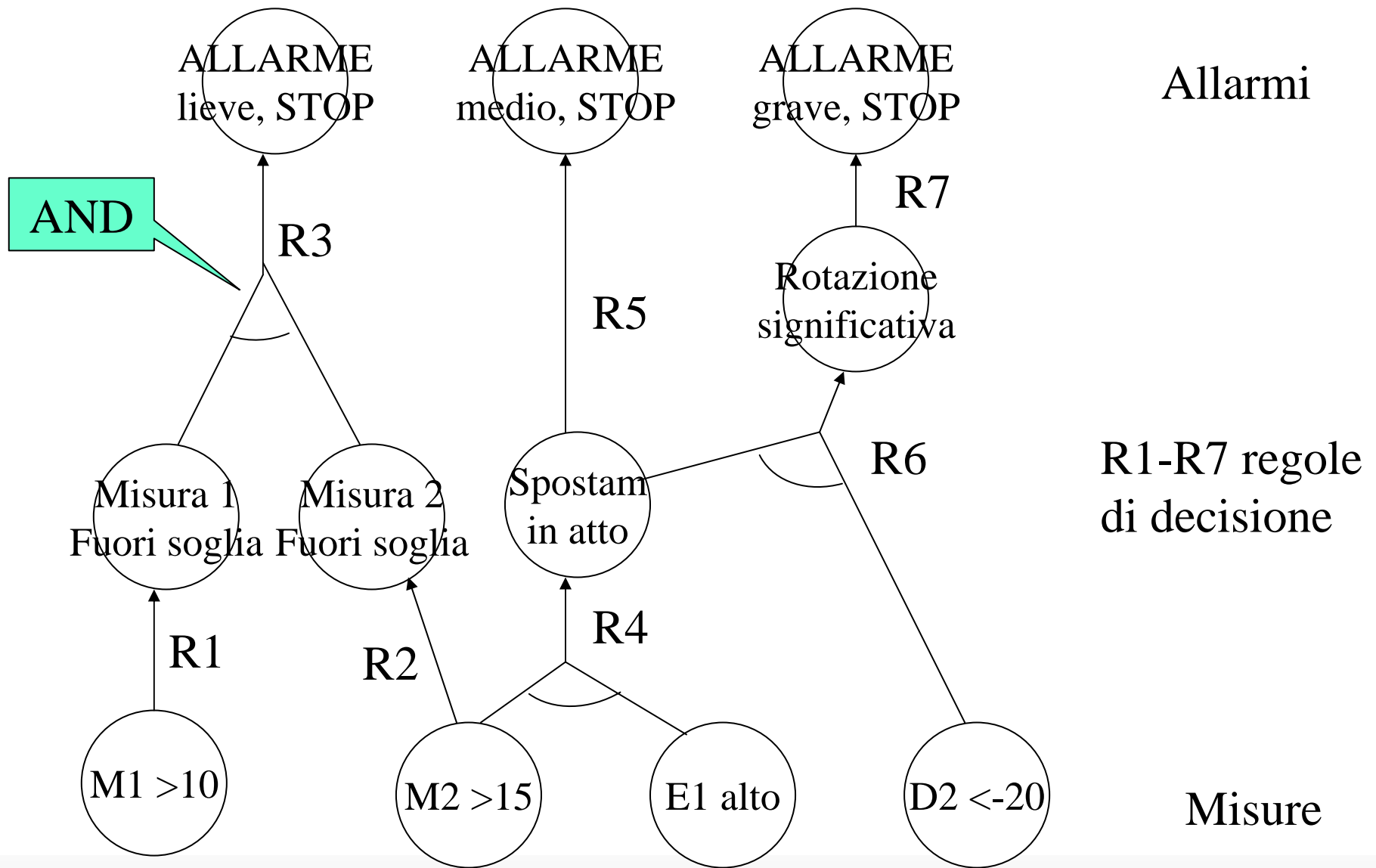
```
while <obiettivo non raggiunto> do  
  begin  
    <MATCH: determina l'insieme delle  
    regole applicabili (cioè le regole il  
    cui antecedente è soddisfatto dai  
    fatti contenuti nella memoria di  
    lavoro) >;  
    <CONFLICT_RESOLUTION: seleziona la  
    regola da applicare>;  
    <FIRE: esegui l'azione associata alla  
    regola>  
  end.
```

Forward e backward chaining

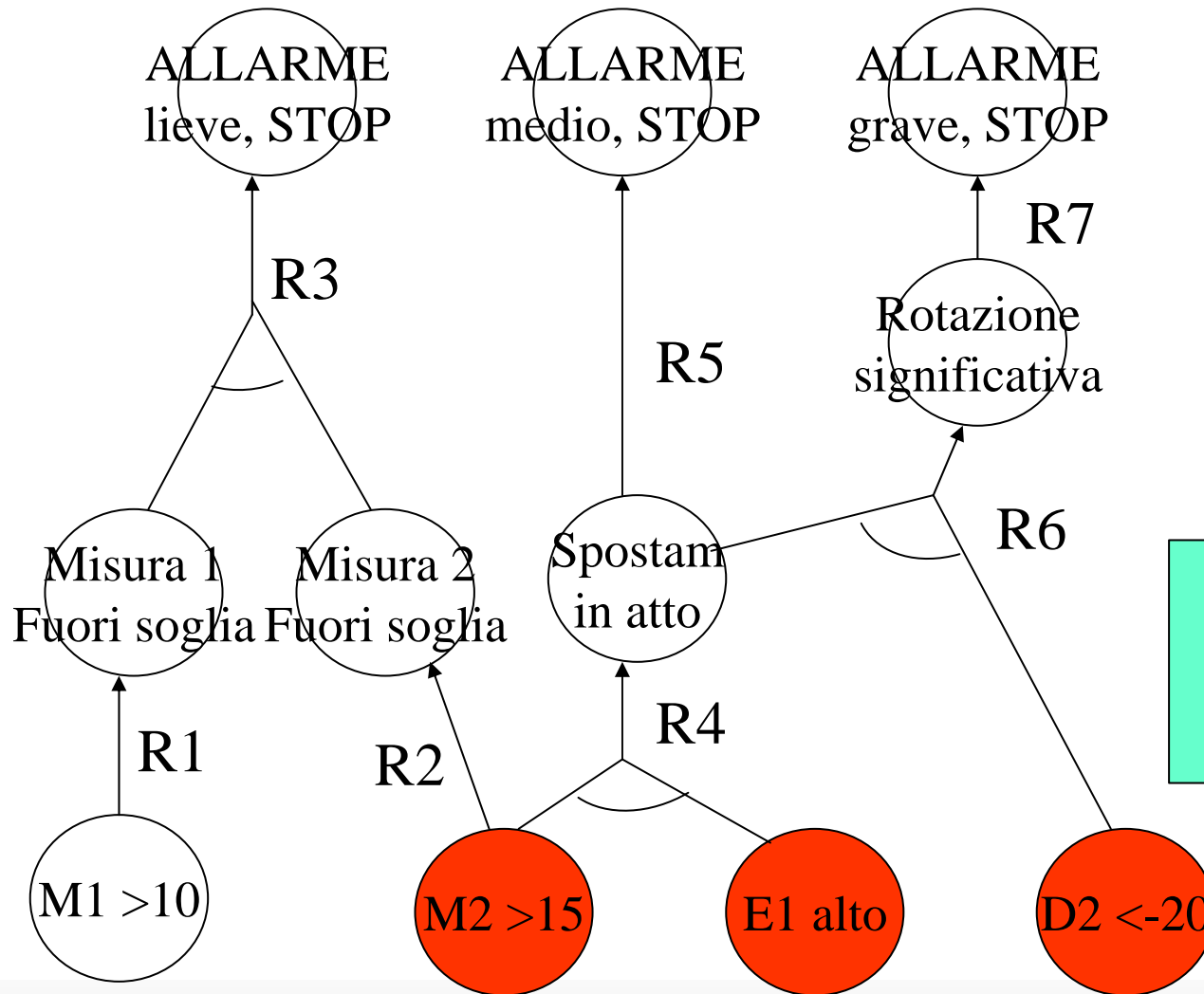
Strategia backward
(all'indietro) o controllo
guidato dal goal G.

```
if <G è un fatto nella memoria di lavoro>  
then <G è dimostrato>  
else  
begin  
  <MATCH: seleziona le regole il cui  
  conseguente può essere unificato con G>  
  <CONFLICT RESOLUTION: seleziona la  
  regola da applicare>  
  <l'antecedente della regola selezionata  
  diventa il nuovo obiettivo (congiunzione  
  di obiettivi) da verificare>  
end.
```

Forward e backward chaining



Forward e backward chaining

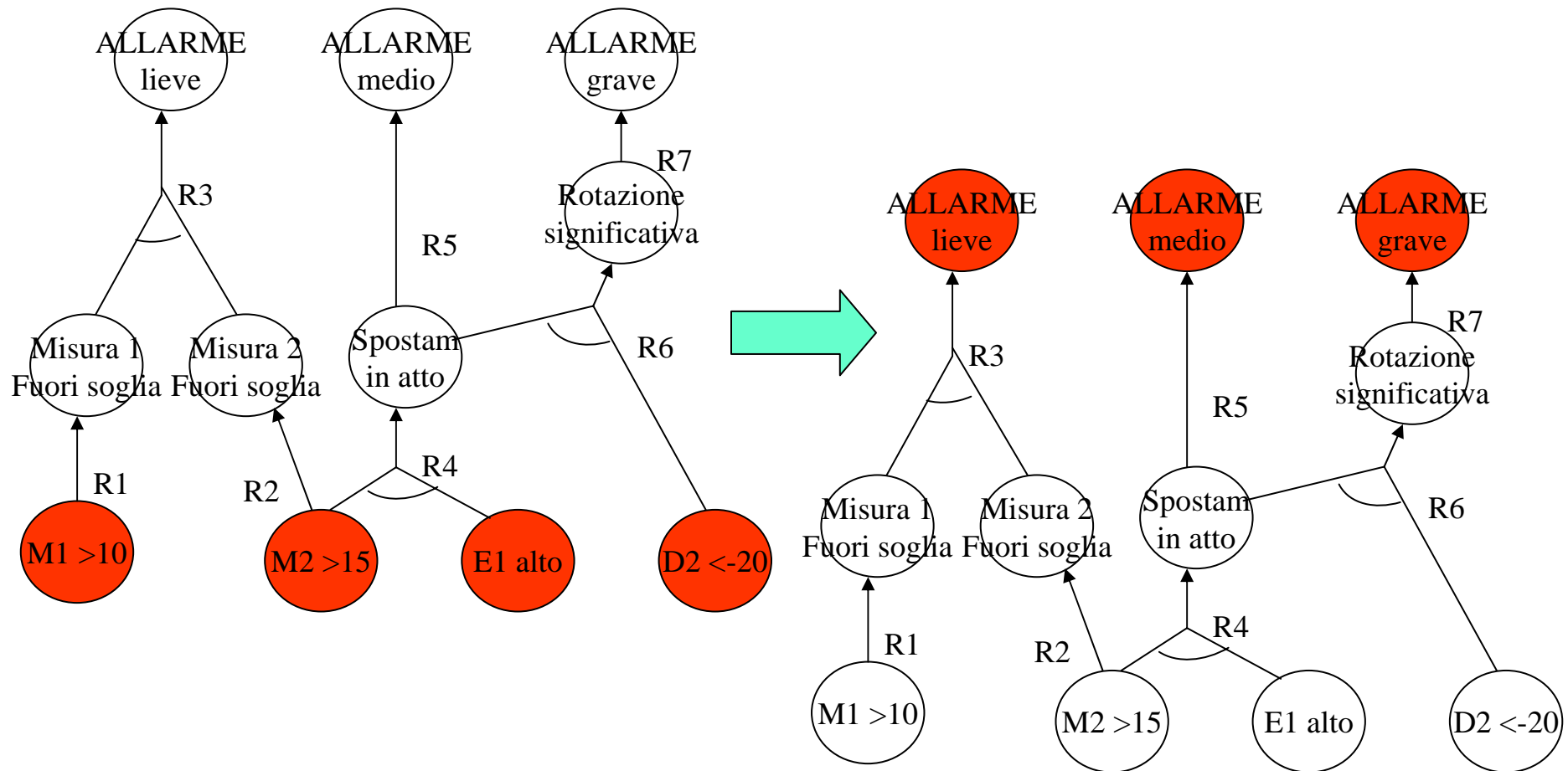


Idea: Il processo di ragionamento è diretto dai dati che sono propagati attraverso la base di conoscenza, deducendo nuovi fatti

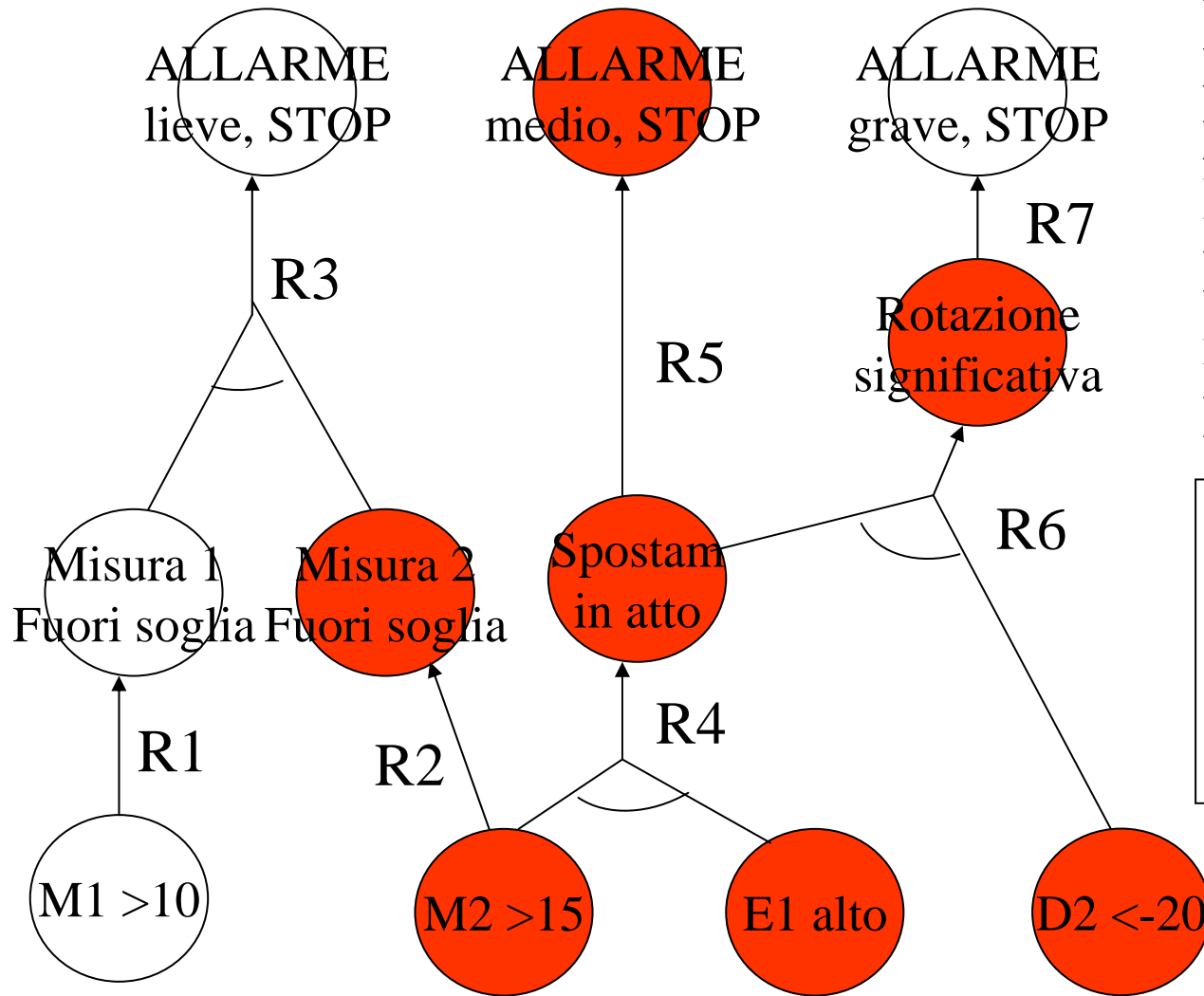
Strategia **forward** (in avanti) o controllo guidato dai dati

Forward e backward chaining

Gestione della propagazione di fatti irrilevanti



Forward e backward chaining

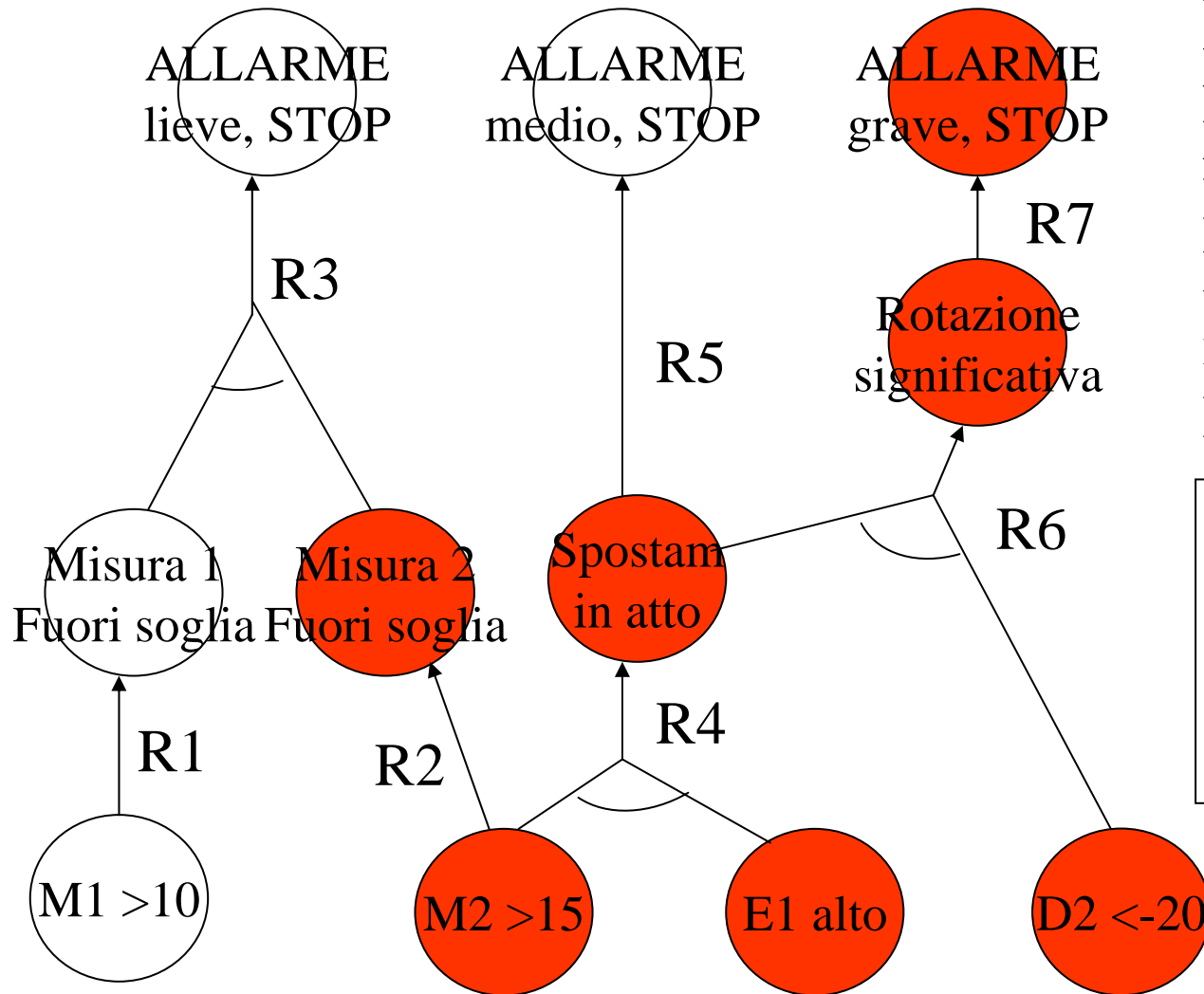


La regola attivata più di recente è accesa per ultima (FIFO, Breadth First)
 Es. Sono attivate nell'ordine R2 e R4
 E' accesa per prima R2

Conflict set	Regola accesa
R4 R2	R2
R4	R4
R6 R5	R5
R6	stop



Forward e backward chaining

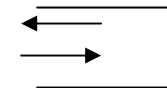


La regola attivata più di recente è accesa per prima (LIFO, Depth First)

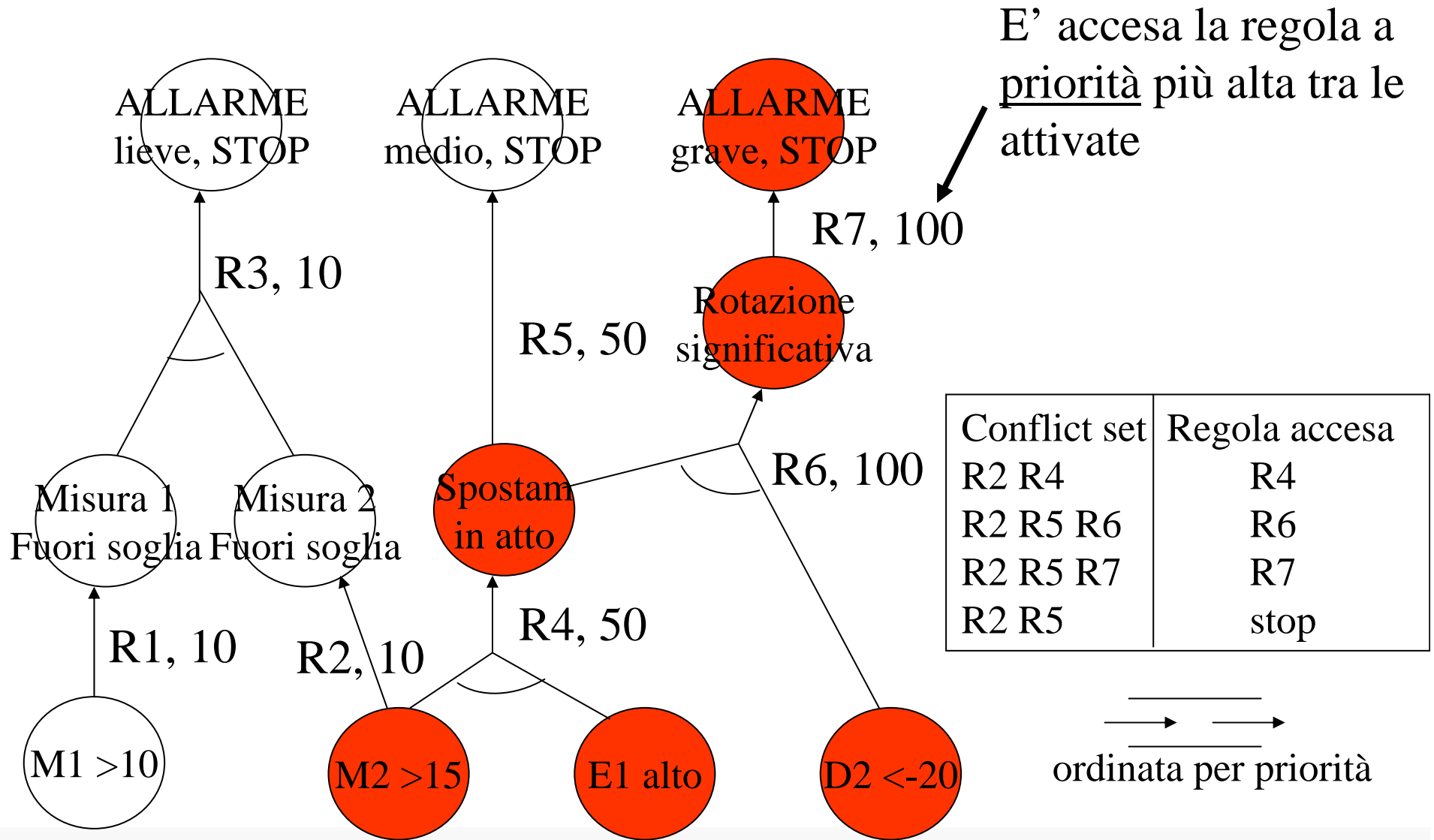
Es. Sono attivate nell'ordine R2 e R4

E' accesa per prima R4

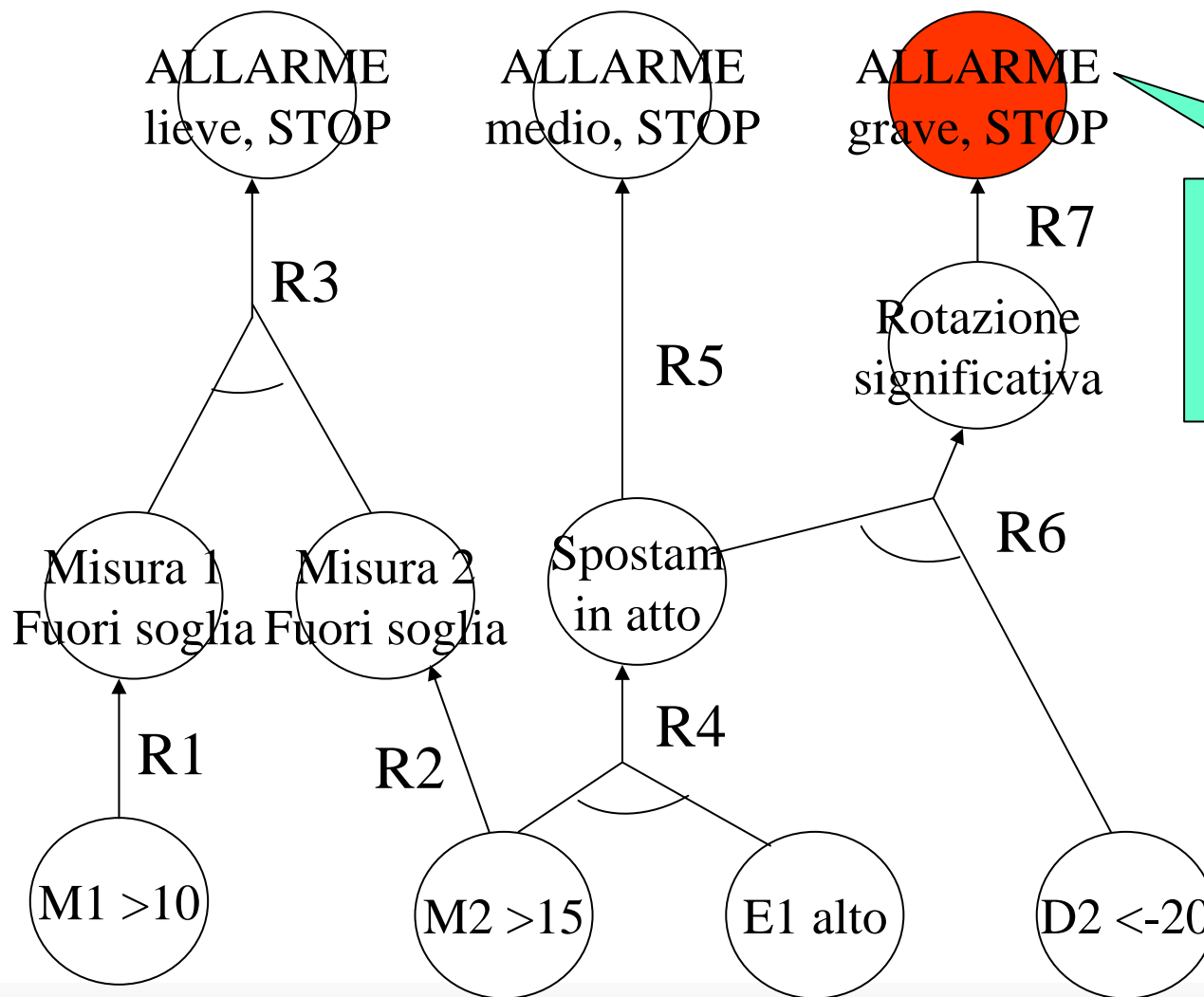
Conflict set	Regola accesa
R4 R2	R4
R6 R5 R2	R6
R7 R5 R2	R7
R5 R2	stop



Forward e backward chaining



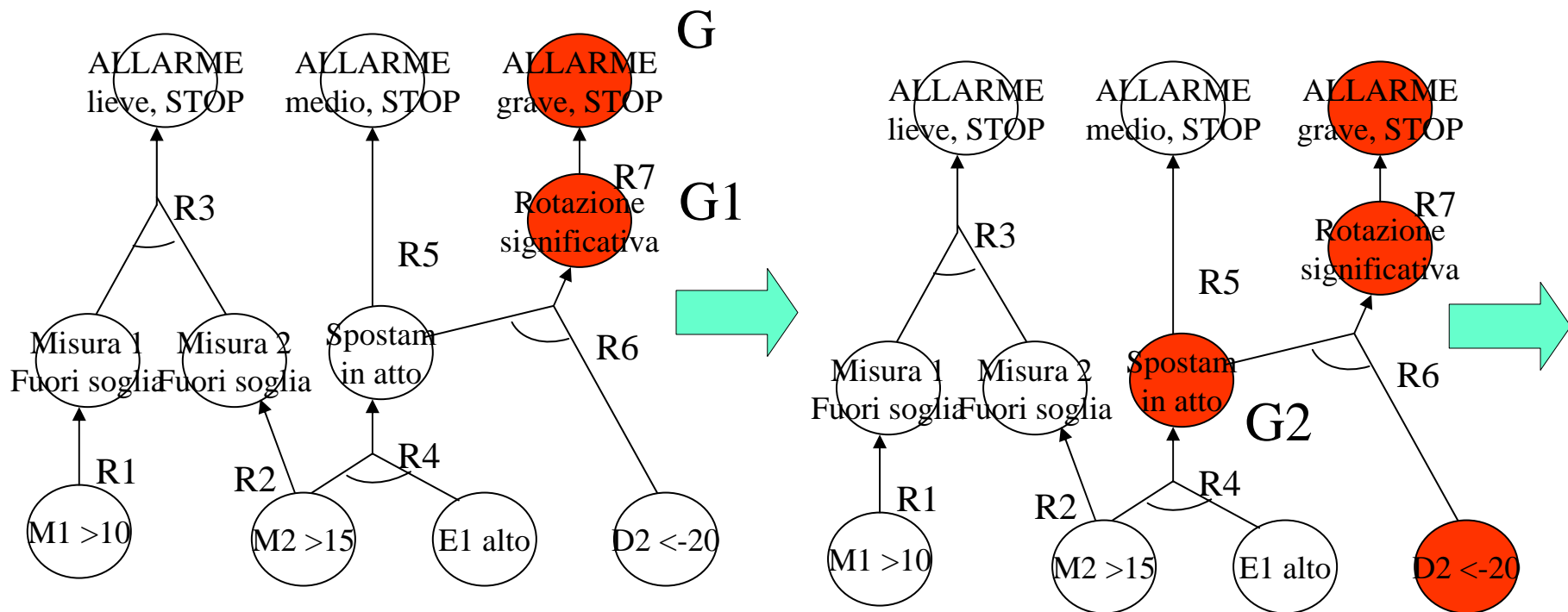
Forward e backward chaining



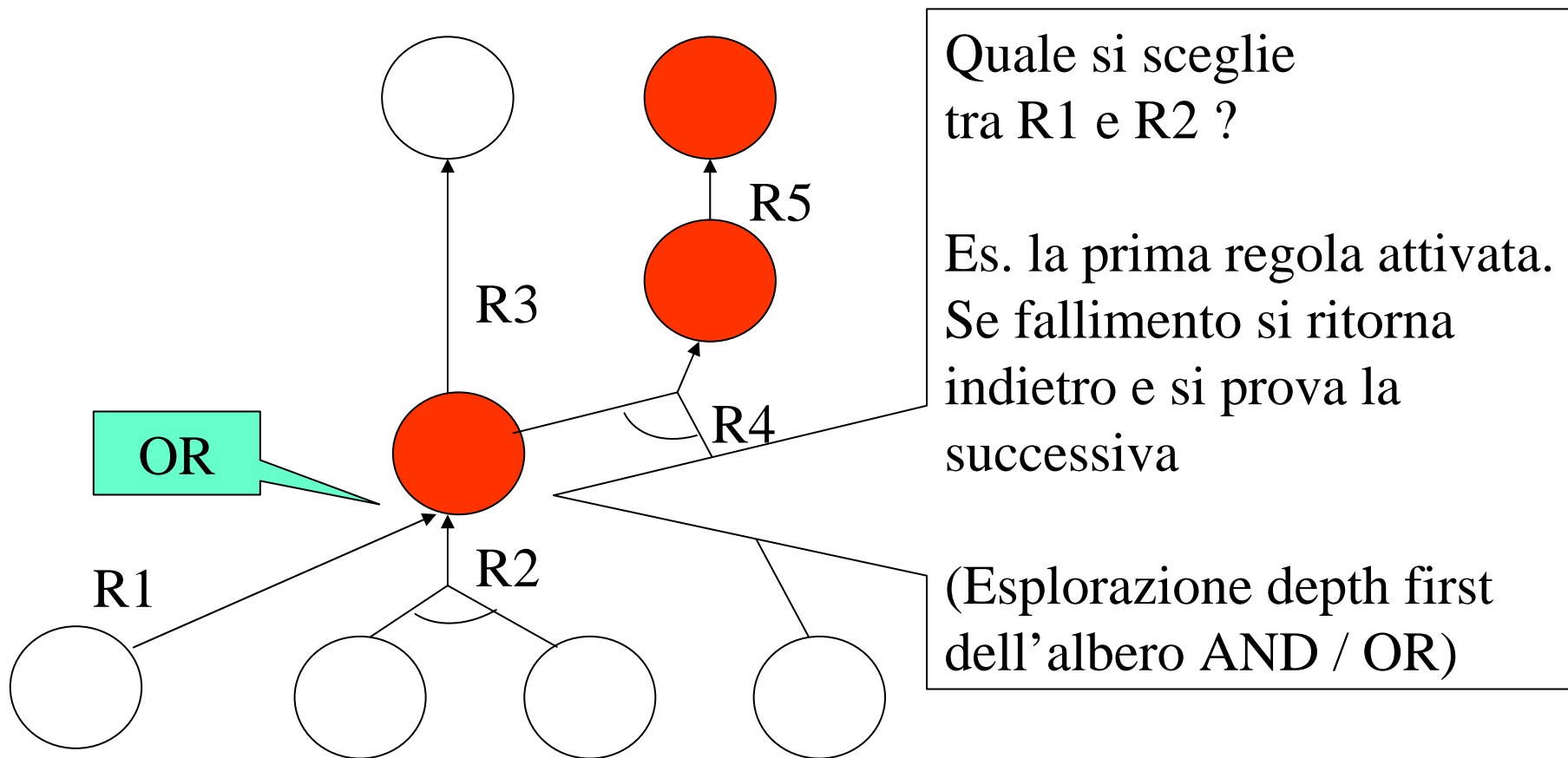
Strategia backward
(all'indietro) o controllo
guidato dal goal G.

Forward e backward chaining

Idea: procedere all'indietro dall'obiettivo G:
per dimostrare G tramite KB, verifica se G è già dimostrato.
Se non lo è, dimostra tramite KB tutte le premesse di una
regola che ha G come conclusione



Forward e backward chaining



Sviluppo di sistemi a regole

- Motivazioni per utilizzare un sistema a regole
 - E' richiesta una significativa capacità di prendere decisioni
 - Le regole decisionali sono potenzialmente complesse
 - Le regole cambiano nel tempo
 - Il codice è mantenuto nel tempo

Sviluppo di sistemi a regole

- Note di Ingegneria del Software relativamente ai sistemi a regole

“Ingegneria della conoscenza” ?????????

- Analisi dei requisiti / Specifiche incerte
- Processo di sviluppo iterativo
- Componente da integrare con altre tecnologie
- Linguaggio dichiarativo e motore di inferenza: piacevole
- Base di regole grande e meccanismi di controllo embedded nel motore di inferenza e nei suoi parametri (es. priorità di accensione delle regole nel conflict set): spiacevole; codice complesso difficile da capire
- Sistemi basati su modelli