

Sistemi a regole [di produzione]

Maria Simi
a.a. 2014/2015

Sistemi a regole

- Fin qui: sistemi a regole come caso particolare di inferenza logica
- I sistemi "a regole" sono storicamente i più utilizzati nella costruzione di sistemi esperti
- OPS5, poi XCON, sono sistemi esperti che hanno prodotto CLIPS [Gary Riley]
- Sono sistemi a regole "in avanti"

Sistemi a regole di produzione

- Uno dei primi paradigmi di rappresentazione della conoscenza (di sapore "procedurale") in I.A.
- Un modello del processo di risoluzione dei problemi da parte dell'uomo.
 - è naturale esprimere competenze sotto forma di regole del tipo "se ... allora ...".
 - i *sistemi esperti* di prima generazione sono quasi tutti sistemi a regole.

Modello di computazione

- Programmazione (o regime di controllo) *guidata da pattern*: quello che viene eseguito al passo successivo è determinato da una attività di *pattern matching*.
- Modello di computazione del tutto generale (stessa potenza della macchina di Turing).

Pattern matching e unificazione

- *Unificazione*: le variabili possono essere in entrambe le espressioni da unificare
 - lo abbiamo visto con i linguaggi logici
- *Pattern matching*
 - si tratta di identificare una sostituzione che rende un *pattern* e un enunciato identici.
 - le variabili sono solo nei pattern, non negli enunciati (componenti dello stato)
 - è un caso particolare di unificazione

Pattern matching

- Se ha successo viene calcolata una "lista di legami per le variabili"
- Esempio:

```
p=(P ?x 0 ?x)
s=(P 3 0 3)
Risultato: {{?x 3}}

p=(P ?x ?y ?x)
s=(P 3 0 4)
Risultato: fail
```

Si può usare lo stesso algoritmo usato per unificazione

Definizione di sistema di produzione

(sistema a regole con concatenazione "in avanti")

Un sistema con 4 componenti:

1. un insieme di *regole di produzione* (o *regole*, o *produzioni*) nella forma:

condizione \Rightarrow *azione*

antecedente \Rightarrow *conseguente*

dove:

- l'*antecedente* è uno schema (o *pattern*) che esprime una condizione che determina l'applicabilità della regola;
- il *conseguente* determina il passo di risoluzione del problema da effettuare: l'aggiunta di un nuovo fatto allo stato corrente, una modifica dello stato, ... una qualunque azione

Definizione (cont.)

2. Una *memoria di lavoro* (WM - Working Memory) che contiene una descrizione dello stato corrente della computazione.

Nei sistemi "in avanti" la WM contiene all'inizio una descrizione dello stato iniziale.

Definizione (cont.)

3. Un *interprete* che esegue un ciclo "riconosci-agisci". Ad ogni passo:

- individua l'insieme delle *regole applicabili* (mediante *pattern matching*); questo è l'*insieme dei conflitti*;
- ne sceglie una, mediante una *strategia di risoluzione dei conflitti*;
- Attiva | "fa scattare" la regola (*fire*), cioè esegue la parte azione (tipicamente istanziata).

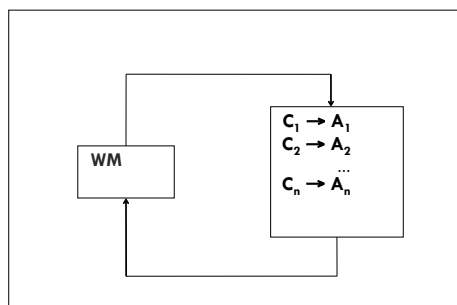
Questo cambia il contenuto della WM e il ciclo si ripete fino ad una condizione di terminazione (successo) o finché non ci sono più regole applicabili (fallimento).

Definizione (cont.)

4. *Strategia di risoluzione dei conflitti* per decidere quale regola applicare tra quelle applicabili:

- "prendi la prima regola applicabile"
- euristiche di utilità generale (le vediamo dopo)
- euristica sofisticata dipendente dal dominio

Schema di funzionamento



Un esempio

- Un semplice sistema di produzioni per ordinare una stringa fatta di "a", "b" e "c".

- *Insieme di regole:*

1. $ba \Rightarrow ab$
2. $ca \Rightarrow ac$
3. $cb \Rightarrow bc$

- *Memoria di lavoro:* la stringa in una fase intermedia del processo di ordinamento (all'inizio la stringa da ordinare).

- *Strategia:* la prima regola applicabile

Esempio (cont.)

Ciclo	Memoria di lavoro	Insieme dei conflitti	Regola selezionata
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	0	ALT

Tanto per fissare le idee ...

- Definiamo un linguaggio a regole minimale, à la *CLIPS*
 - I fatti e la memoria di lavoro (WM)
 - Le regole
 - L'interprete (il motore di inferenza)
 - L'agenda
 - Le strategie di risoluzione dei conflitti

I fatti e la memoria di lavoro

Sintassi: (<atom> <atomic-object>*)

Esempi:

(lista-spesa latte pane biscotti)

(altezza 1,80)

(luce accesa)

WM: lista di fatti (senza variabili)

Le regole

(defrule <rule-name>
 <condition>* => <action>*)

- l'antecedente è una lista di condizioni, da considerare in congiunzione tra di loro. Tipicamente hanno la stessa struttura dei fatti ma contengono variabili (sono *pattern con variabili*).
- il conseguente è una lista di azioni che possono contenere variabili

Le azioni

- Aggiunta di un fatto alla WM:
 - (assert <fact>)
- Rimozione di un fatto: (retract <fact>)
- Per stampare: (print <string>)
- Per terminare: (return)
- Una qualunque altra azione prevista ...

Esempio: agenzia matrimoniale

(persona nome sesso
 libero status avvenenza)

- nome: nome della persona
- sesso: M, F
- libero: T, NIL (vero, falso)
- status: 1, 2, 3, 4 a seconda di quanti soldi ha
- avvenenza: scarsa media buona ottima

Agenzia matrimoniale: una regola

```
(defrule marry-money
  (persona ?nome-lei F T ?status-lei ottima)
  (persona ?nome-lui M T ?status-lui scarsa)
  (< ?status-lei 2) (> ?status-lui 2)
  =>
  (retract (persona ?nome-lei F T ?status-lei ottima))
  (retract (persona ?nome-lui M T ?status-lui scarsa))
  (assert (persona ?nome-lei F NIL ?status-lui ottima))
  (assert (persona ?nome-lui M NIL ?status-lui scarsa))
  (print ?nome-lei " e " ?nome-lui " si sposano per soldi")
```

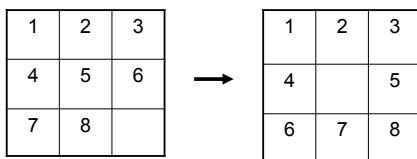
WM: (persona mary F T 1 ottima)
(persona john M T 3 scarsa)

L'interprete

L'interprete esegue un ciclo:

- **Determina regole applicabili (costruisce agenda)**
 - Mediante pattern-matching si controlla quali condizioni sono soddisfatte da fatti nella WM
 - Le "attivazioni" vengono aggiunte all'agenda: nome della regola, fatti usati, legami, eventuale priorità
- **Ordina l'agenda in base ad una strategia di risoluzione dei conflitti**
- **Esegue la prima regola dell'agenda**
 - Valuta la lista di azioni nella parte destra, istanziate dalla lista dei legami.
 - Se l'azione è (return) termina.

Il gioco dell'8 a regole



Il gioco dell'8: rappresentazione immediata

Rappresentazione: (B 1 2 3 4 0 5 6 7 8)

Le regole per ↑:

```
(B ?x1 ?x2 ?x3 0 ?x4 ?x5 ?x6 ?x7 ?x8) =>
  (assert (B 0 ?x2 ?x3 ?x1 ?x4 ?x5 ?x6 ?x7 ?x8))
  (retract (B ?x1 ?x2 ?x3 0 ?x4 ?x5 ?x6 ?x7 ?x8))
(B ?x1 ?x2 ?x3 ?x4 0 ?x5 ?x6 ?x7 ?x8) =>
  (assert (B ?x1 0 ?x3 ?x4 ?x2 ?x5 ?x6 ?x7 ?x8))
  (retract (B ?x1 ?x2 ?x3 ?x4 0 ?x5 ?x6 ?x7 ?x8))
(B ?x1 ?x2 ?x3 ?x4 ?x5 0 ?x6 ?x7 ?x8) =>
  (assert (B ?x1 ?x2 0 ?x4 ?x5 ?x3 ?x6 ?x7 ?x8))
  (retract (B ?x1 ?x2 ?x3 ?x4 ?x5 0 ?x6 ?x7 ?x8))
(B ?x1 ?x2 ?x3 ?x4 ?x5 ?x6 0 ?x7 ?x8) => (assert (B ?x1 ?x2 ?x3 0 ?x4 ?x5 ?x6 ?x7 ?x8))...
(B ?x1 ?x2 ?x3 ?x4 ?x5 ?x6 ?x7 0 ?x8) => (assert (B ?x1 ?x2 ?x3 ?x4 0 ?x5 ?x6 ?x7 ?x8))...
(B ?x1 ?x2 ?x3 ?x4 ?x5 ?x6 ?x7 ?x8 0) => (assert (B ?x1 ?x2 ?x3 ?x4 ?x5 0 ?x6 ?x7 ?x8 ?x8))...
```

1	2	3
4	0	5
6	7	8

... + 6 regole per ↓ + 6 regole per → + 6 regole per ← = 24

Il gioco dell'8: un po' meno regole

Rappresentazione: (P cifra riga colonna)

Regole per ↑:

```
(P 0 2 ?y) (P ?c 1 ?y) =>
  Assert (P 0 1 ?y), Assert (P ?c 2 ?y),
  Retract (P 0 2 ?y), Retract (P ?c 1 ?y)
(P 0 3 ?y) (P ?c 2 ?y) =>
  Assert (P 0 2 ?y), Assert (P ?c 3 ?y),
  Retract (P 0 3 ?y), Retract (P ?c 2 ?y)
```

Totale: 8 regole

Il gioco dell'8: ancora meno regole

Rappresentazione: (P cifra riga colonna)

Regola per ↑:

```
(P 0 ?x ?y) (?x≠1) (P ?c (?x - 1) ?y) =>
  Assert (P 0 (?x - 1) ?y), Assert (P ?c ?x ?y),
  Retract (P 0 ?x ?y), Retract (P ?c (?x - 1) ?y)
```

Totale: 4 regole

Il gioco dell' 8: considerazioni

- La rappresentazione scelta può fare la differenza nel numero delle regole necessarie
- Più espressivo il linguaggio dei pattern (e delle regole) più compatta è la rappresentazione e più complicato è il pattern-matching.
- Istanza del problema più generale: più espressivo il linguaggio di rappresentazione, più compatta la rappresentazione e più complesso il meccanismo inferenziale.

Strategie di risoluzione dei conflitti

1. *Basate sulle regole*
 - la prima regola applicabile
 - la più specifica o con condizioni più stringenti.
Es. $c1 \wedge c2 \wedge c3 < c1 \wedge c2$; $(P \ 0 \ 1 \ 2) < (P \ 0 \ ? \ 2)$
 - 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*)
 - la più plausibile (*pattern matching approssimato*)

Strategie di risoluzione dei conflitti

2. *Basate sugli oggetti*: su una graduatoria di importanza degli oggetti che compaiono nei pattern.

Esempio 1 (da Eliza):

"I know everybody laughs at me"
everybody > I

Esempio 2:

(Stanza in-fiamme)(Bambino in-pericolo) \Rightarrow
(Salva-bambino) (Esci)
(Luce accesa) \Rightarrow (Spengi-luce)(Esci)
Bambino > Luce

Strategie di risoluzione dei conflitti

3. *Basate sull'effetto delle regole*: si applica una funzione di valutazione agli stati risultanti e si sceglie il migliore.

Strategie di risoluzione dei conflitti

4. *Meta-regole*: la strategia di controllo è definita tramite altre regole, che essendo regole che trattano di regole vengono dette *meta-regole*.

Esempio:

Sotto le condizione A e B

le regole che [non] menzionano X

{del tutto | nell'antecedente | nel conseguente}

sono {del tutto inutili | probabilmente utili |

forse utili | molto utili}

Meta-regole in SOAR e PRODIGY.

Complessità del pattern matching

Il problema è quello di fare un pattern-matching molti a molti tra:

- una serie di r regole (in OR)
- una serie di n antecedenti delle regole (in AND)
- una serie di w componenti dello stato (in OR)

$(OR \dots (AND \dots (OR \dots)))$

↑
sulle regole sulle precondizioni sulle comp. di stato

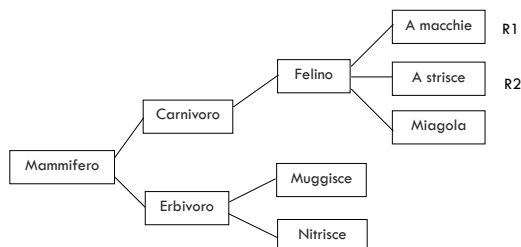
Il tutto ripetuto per c cicli:

$r \times n \times w \times c$ operazioni di pattern matching!

Ottimizzazioni: rete di discriminazione

- Assunzione 1: regole diverse possono condividere molte delle precondizioni. Esempio:
 R1: (Mammifero ?x) (Felino ?x) (Carnivoro ?x)
 (A-Macchie ?x) \Rightarrow (assert (Leopardo ?x))
 R2: (Mammifero ?x) (Felino ?x) (Carnivoro ?x)
 (A-Strisce ?x) \Rightarrow (assert (Tigre ?x))
- Idea : codificare gli antecedenti delle regole sotto forma di rete di discriminazione

Esempio di rete di discriminazione



Ottimizzazioni: calcolo incrementale

- Assunzione 2: l'applicazione di una regola influenza solo pochi elementi dello stato e quindi l'insieme delle regole applicabili, tipicamente, varia di poco da un ciclo all'altro
- Idea: restringersi alle regole la cui applicabilità è influenzata dall'ultima modifica allo stato.

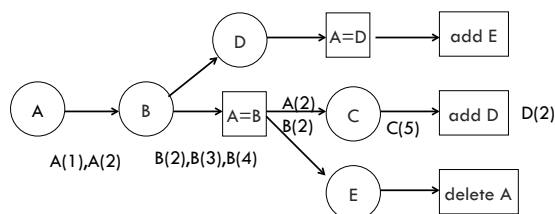
Algoritmo RETE [Forgy 1982]

Esempio: WM = {A(1), A(2), B(2), B(3), B(4), C(5)}

A(x) B(y) D(x) \Rightarrow add E(x)

A(x) B(x) C(y) \Rightarrow add D(x)

A(x) B(x) E(z) \Rightarrow delete A(x)



Vantaggi del paradigma a regole

- È un modello plausibile del ragionamento umano e comunque ...
- C'è una certa naturalezza nel modellare, sotto forma di regole, il tipo di competenza di natura "euristica" dell'esperto umano
- Implementazione naturale del paradigma di risoluzione dei problemi come ricerca

Vantaggi del paradigma a regole

- Separazione della conoscenza (lo stato e le regole) dal controllo (l'interprete o "motore inferenziale" \Rightarrow possibilità di cambiare una delle due parti indipendentemente)
- Modularità della base di regole: poca interazione tra le regole (solo tramite la WM) \Rightarrow supporto per lo sviluppo incrementale
- Possibilità di traccia e giustificazione
- Il modello di computazione è generale, può essere adattato a linguaggi diversi (logici o meno).

Per provare CLIPS

- <http://clipsrules.sourceforge.net/>