Corso di Ingegneria della Conoscenza

Tecniche di rappresentazione della conoscenza

Andrea Bonarini

Politecnico di Milano

Dipartimento di Elettronica e Informazione
Artificial Intelligence and Robotics Project
Via Ponzio 34/5 - 20133 Milano

Tel. (02) 2399 3525

In questa parte tratteremo il secondo livello nel processo della rappresentazione della conoscenza. Abbiamo visto finora le caratteristiche generali della conoscenza; ora vedremo quali sono le tecniche a disposizione per rappresntarla in modo opportuno per un sistema esperto.

La controversia dichiarativo-procedurale

Descrizione dichiarativa.

Es.: Dato un punto A su un piano P, una circonferenza é il luogo dei punti di P equidistanti da A.

Descrizione procedurale.

Es.: Si ottiene un cerchio ponendo un bicchiere su di un foglio di carta e girandoci in torno con una matita

Negli anni settanta esplose nel settore della rappresentazione della conoscenza, la cosiddetta controversia dichiarativo-procedurale: c'era chi sosteneva l'importanza di avere solo rappresentazioni della conoscenza dichiarative, che cioè descrivessero gli elementi di conoscenza, mentre c'erano i fautori delle descrizioni procedurali, che mettevano l'accento su come si opera sulla conoscenza.

Si noti che la controversia, anche se ha fondamento, é in realtà articolata e coinvolge diversi aspetti della rappresentazione, non ultimo il suo livello di astrazione. Nel caso riportato a lato, ad esempio, abbiamo descrizioni a diversi livelli di dettaglio, di diverse cose (una circonferenza e come fare a disegnare una circonferenza) ed ognuna ragionevole per un dato task.

Le dichiarazioni tendono ad essere più brevi e vanno meglio per reasoning e spiegazioni. Le procedure tendono ad essere specifiche (possono contenere dettagli irrilevanti), ma producono risultati che le descrizioni dichiarative non producono

In realtà entrambe le modalità sono importanti ed è necessario usare tecniche che ne permettano l'integrazione.

Come esercizio, provate a rendere più generale la descrizione procedurale. Cosa si ottiene?

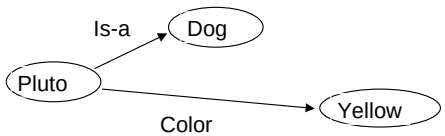
Quali sono le informazioni che cambiano?

Nel seguito esamineremo alcune tecniche di rappresentazione della conoscenza, evidenziandone problemi e campi applicativi

Reti semantiche

Nodi = Informazione atomica

Archi = legami tra i nodi



Le reti semantiche furono introdotte negli anni sessanta per rappresentare il significato di concetti. Ne vediamo a lato una rappresentazione grafica, ma è ovviamente possibile darne una rappresentazione formale con un linguaggio testuale.

Abbiamo in sostanza due elementi: nodi e archi. I nodi contengono informazioni atomiche, cioè, sostanzialmente, gli identificatori di concetti. Gli archi collegano due nodi, sono orientati, e sono associati al nome di una relazione.

Ad esempio, la rete riportata a lato rappresenta la conoscenza descritta dalla seguente frase: “Pluto è un cane giallo”

Problemi con le reti semantiche

- Descrizione del significato dei link
- Classificazione dei nodi
- Nessuna descrizione procedurale

Tecniche KR

Tra i problemi di questa rappresentazione abbiamo quelli citati a lato.

Innanzitutto i legami forniti dagli archi sono indifferenziati e non possono essere descritti. Ad esempio, solo il nome distingue il legame “Is_A” (che nell’esempio ha un ruolo analogo a quello che aveva #InstanceOf), dal legame “Color”, che non è altro che un modo per rappresentare una proprietà.

Altro problema consiste nel fatto che i nodi sono tutti dello stesso tipo, e non è possibile distinguere all’interno del formalismo tra nodi che rappresentano cassi e nodi che rappresentano esemplari.

Ultimo problema e’ che ogni descrizione procedurale che operi su conoscenza rappresentata da reti semantiche e’ fatta con formalismi diversi.

Frames

Strutture dati:

<frame> ::=(<nome-frame> <is-a> <slot>*)

<is-a> ::= (is-a <nome-frame>)

<slot> ::= (<nome-slot> <valore slot>)

<valore slot> ::= <simbolo> | <nome-frame>

Es:

```
(dog     (is-a mammal)
         (number-of-legs 4)
         (eats meat))
```


I frames sono strutture dati che permettono di raggruppare come dentro una cornice (in inglese “frame”) le informazioni relative ad un’entità.

E’ caratteristico dei frames avere uno slot is-a che collega il frame ad uno che fornisce una descrizione più generale. Lo stesso slot è usato per implementare sia le relazioni di generalizzazione tra classi, sia le relazioni di appartenenza di un esemplare ad una classe.

Come indicato a lato, un frame è identificato da un nome, seguito dallo slot Is-a e da uno o più slot. I valori degli slot possono essere sia dei simboli (numeri o parole), sia dei riferimenti ad altri frames.

Nell’esempio riportato si è rappresentato il fatto che i cani sono mammiferi, hanno 4 gambe e mangiano carne

Caratteristiche e problemi dei frames

Cosa si puo' rappresentare con i frames?

- Proprietà di elementi della conoscenza
- Eredità esplicita solo con gerarchia is-a

Problemi

- Nessuna specifica circa gli aspetti procedurali
- Scarsa caratterizzazione epistemologica

I frames sono quindi adatti a rappresentare proprietà, classi, esemplari. All'interno di questo formalismo non sono previste modalità per rappresentare conoscenza procedurale, ma spesso questo è ottenuto associando ai frames delle diverse tecniche orientate alla rappresentazione di aspetti procedurali.

Tra i problemi dei frames si ricorda anche che non prevedono la possibilità di distinguere classi da esemplari, che sono rappresentati in maniera uniforme.

Come definiamo i frames?

DIamo qui a lato una possibile definizione formale di frames, in una forma CLIPS-like.

Definizione CLIPS-like

(defframe <nome-frame>
(is-a <is-a-ref>*)
(<slot> <value>)*
)

E' solo <u>una</u> possibile definizione

Es.: **(defframe Pluto**
(is-a dog cartoon)
(has-legs 4)
(color yellow)
(owned-by Mickey)
(eats bones))
)

Test #2: frames

Rappresentare con frames la conoscenza contenuta nelle frasi seguenti:

- **Un ingegnere e' una persona.**
- **Aristide e' un ingegnere.**
- **Gli ingegneri sono laureati.**
- **Aristide si e' laureato il 20 luglio 1999.**
- **Ad Aristide piace ballare.**
- **Agli ingegneri piace il computer e non piace ballare.**

Si richiede qui di rappresentare la conoscenza contenuta nelle stesse frasi utilizzate nel Test #1. Lo scopo è di mostrare come le caratteristiche della conoscenza emerse in quella sede possano essre implementate (in modo abbastanza diretto) con il formaismo dei frames.

Test #3: frames

Rappresentare con frames la conoscenza contenuta nelle frasi seguenti:

- **La pompa P342 e' collegata al condotto C32.**
- **Il condotto C32 e' collegato al serbatoio S321.**
- **Il serbatoio S321 e' vuoto.**
- **La portata del condotto C32 e' 2 mc/h.**
- **Il flusso in uscita dalla pompa P342 e' 1 mc/h.**
- **La pompa P342 non funziona correttamente.**

Nello svolgere questo esercizio, lo studete è invitato in un primo tempo a rappresentare le caratteristiche della conoscenza coinvolta, attraverso il formalismo delle unit, e, quindi, a ricavare dal modello concettuale così ottenuto una rappresentazione attraverso il formalismo dei frames.

Demoni

Procedure associate ai dati => data driven programming

Tre tipi di demoni

- **Demoni** IF-ADDED**: scatta se il dato è modificato**
- **Demoni** IF-NEEDED**: scatta se il dato è letto**
- Notifiers**: scatta se il dato assume certi valori**

Il formalismo dei ***demoni*** permette di definire delle procedure che vengono attivate quando si verificano certe condizioni sulle strutture dati ad esse associate.In molte applicazioni sono associati a slot in strutture dati tipo frames. A queste ci riferiamo negli esempi che seguono.

Tre sono i tipi di demoni che esaminiamo.

I demoni ***IF-ADDED*** scattano ogni volta che viene fatta una modiffica al valore dello slot a cui sono associati. Ad esempio, potremmo avere un demone IF-ADDED associato ad uno slot che indica il valore della pressione all'interno di una parte di un impianto. Tutte le volte che questo viene aggiornato, il demone scatta ed aggiorna anche il valore della pressione in un'altra parte dell'impianto, che sappiamo essere legato al precedente da un legame inferenzale di tipo "constraint".

I demoni ***IF-NEEDED*** scattano ogni volta che il valore dello slot a cui sono associati viene letto.

I demoni di tipo "notifiers" scattano ogni volta che lo slot assume un certo valore. AD esempio, potrebbero essere usati per far scattare un allarme quando una certa pressione supera un valore critico.

Definizione di demoni

Ad esempio, associati a proprietà dei frame

(defframe <nome-frame>
(is-a <is-a-ref>*)
(<slot> <value>
((<demon-type> <demon-ref><arguments>)*))*)

Es.:
(defframe login-env
(is-a security-env)
(owner user)
(last-login date
((if-added #'inform-sys-manager (last-login owner)))))

Riportiamo a lato un esempio di possibile definizione di demoni associati a frames.

In questo caso vediamo come aumentano le possibilità di definizione degli slots di un frame, attraverso l'associazione di uno o più demoni. I demoni sono identificati da un tipo, seguito dal riferimento ad una funzione lisp che implementa gli associati aspetti procedurali, seguito da una lista di possibili valori. Questi valori possono essere valori atomici (termini, o numeri) riferimenti ad altri frames, o nomi di slost del frame in cui e' contenuto lo slot a cui e' associato il demone.

Cosa si puo' rappresentare con i demoni?

Conoscenza procedurale

Problemi

- Impossibilità di controllare il flusso
- Necessità di appoggiarsi ai dati

I maggiori problemi di questa tecnica di rappresentazione di conoscenza procedurale sono legati al fatto che i demoni, per loro natura, scattano su condizioni che dipendono dai dati e che sono difficilmente prevedibili a priori. Quindi, se è vero che i demoni permettono di definire procedure in modo modulare e di attaccarle ai dati a cui servono, è pure vero che questo fa perdere di vista l'andamento globale del processo inferenziale.

Oggetti

Strutture di conoscenza che si scambiano messaggi

- **Classi e esemplari**
- **Eredità is-a**
- **Procedure come <u>metodi</u> per rispondere a <u>messaggi</u>**
- **Possibilità di tipizzare le proprietà**

Una rappresentazione ad ***oggetti*** permette di unire aspetti procedurali ad aspetti dichiarativi, in un unico formalismo. La parte ***dichiarativa*** è trattata in modo analogo ai frames, anche se qui abbiamo i concetti di classe ed esemplare distinti, ed abbiamo la possibilità di qualificare gli slot con alcune caratteristiche importanti, quali il tipo di valori e il valore di default.

Gli aspetti procedurali sono realizzati attraverso ***metodi*** che ripondono a ***messaggi*** inviati all'oggetto. L'oggetto, quando riceve un messaggio cerca tra i suoi metodi se ne ha uno che può rispondere al messaggio giunto. In caso affermativo lo attiva, rispondendo con questo al messaggio. I metodi, a loro volta, possono contenere messaggi indirizzati ad altri oggetti.

In altri termini è come se l'oggetto possedesse della conoscenza e sapesse come operare su di essa e come interagire con altri oggetti.

Definizione di classi

```
(defclass <class_name>
    (is_a <class>+)
    (slot <slot_name> <facet>*)*)

(defclass CAR
       (is-a USER)
       (slot max-speed (access read-only) (default 200))
        (slot speed (default 0)))
```


Qui abbiamo un possibile formalismo per la rapprentazione di oggetti. Quattro sono gli elementi fondamentali:

la definizione di classi

la definizione di esemplari delle classi

la definizione di metodi

l'attivazione dei metodi

Le classi sono collegate in una gerarchia di generalizzazione e specializzazione. Le proprieta' sono definite da un nome ed eventualmente caratterizzate da un tipo (che determina i possibili valori della proprieta') e da un valore iniziale, che viene assegnato nel momento della creazione degli esemplari, quando non vengano forniti altri valori esplicitamente. Possono inoltre essere associate molte altre caratteristiche alle proprieta'.

Definizione di esemplari

Gli esemplari sono definiti indicandone la classe ed eventuali valori iniziali per le proprieta'.

(make-instance <instance_name> of <class_name>
  (<slot_name><slot_value>)*)

(make-instance my-car of CAR
    (speed 100))

Definizione di metodi (1)

I metodi sono definiti indicando la classe a cui fanno riferimento, le eventuali variabili ed un corpo procedurale.

```
(defmessage-handler <class-name><method_name>
  (<var_name>*)
  <body>)
```

CLIPS>

```
(defmessage-handler CAR accellerate
    ((?percentual-speed ZERO-ONE))
   (bind ?self:speed (+ ?self:speed
         (* ?self:speed   ?percentual-speed)))
```

Esempio metodi (1)

CLIPS>
(make-instance my-car of CAR
    (speed 100))
[mycar]
CLIPS>
(send [mycar] accellerate 0.1)
110
CLIPS> (send [mycar] print)
[mycar] of CAR
(max-speed 200)
(speed 110)

I metodi vengono inviati con una primitiva (send) agli esemplari di una classe.

Una variabile definita automaticamente (?self) si riferisce sempre all'esemplare a cui vengono inviati i metodi.

Definizione di metodi (2)

Esiste anche un altro modo di definire implicitamente i metodi, che permette di associare un metodo ad un insieme di classi che ne tipizzano gli argomenti.

```
(defmethod <method_name>  ((<var_name><var-type>*)*)
  <body>)

CLIPS>
(defmethod sum  ((?a INTEGER)(?b INTEGER))
  (+ ?a ?b))
CLIPS> (sum 2 3)
5
CLIPS>
(defmethod sum  ((?a STRING)(?b STRING))
  (str-cat ?a ?b))

CLIPS> (sum "reggi" "libro")
reggilibro
```

Assegnamento e lettura di valori di slot

(send <instance-name> put-<nomeslot> <new-value>)
(send <instance-name> get-<nomeslot>)

CLIPS> (send [mycar] put-speed 180)
180
CLIPS> (send [mycar] get-speed)
180

Con questi metodi di default si può fare riferimento a slot anche all'interno di regole

Sono usati anche dei metodi, generati autometicamente nel momento in cui viene definita la classe) che permettono di accedere in lettura o in scrittura agli slot di esemplari.

Cosa si può rappresentare con gli oggetti?

- Proprietà di elementi della conoscenza
- Eredità esplicita solo con gerarchia gen/spec ed esemplare/classe
- Operazioni sulle proprietà e sugli oggetti

Problemi

- Scarsa caratterizzazione epistemologica
- I metodi sono comunque procedure

Test #4: oggetti

Rappresentare con oggetti la conoscenza contenuta nelle frasi seguenti:

- **La pompa P342 e' collegata al condotto C32.**
- **Il condotto C32 e' collegato al serbatoio S321.**
- **Il serbatoio S321 e' vuoto.**
- **La portata del condotto C32 e' 2 mc/h.**
- **Il flusso in uscita dalla pompa P342 e' 1 mc/h.**
- **La pompa P342 non funziona correttamente.**
- **Quando un serbatoio e' vuoto suona l'allarme relativo.**
- **L'indicatore PIRCA321 da' la pressione nel serbatoio S231 su richiesta.**
- **L'indicatore LEV321 da' il livello nel serbatoio 321 sempre.**

Anche qui, si invita lo stdente a rappresentare dapprima le caratteristiche della conoscenza conenute nelle frasi riportate, e quindi a tradurre il modello concettuale nel formalismo ad oggetti.

Logica

Logica dei predicati del primo ordine, ma non solo.

Apparato formale:

- **assiomi**
- **teoremi**
- **operatori di combinazione**
- **quantificatori**

L'approccio logico, trattato in dettaglio in altri corsi, e' basato sull'utilizzo di un apparato formale che spesso è quello della logica dei predicati del primo ordine. Si tratta quindi di avere degii assiomi e dei teoremi sui quali opera un meccanismo inferenziale con regole predefinite (modus ponens, ...). E' possibie combinare elementi della rappresentazione con opportuni operatori (AND, OR, NOT). E' inoltre possibile quantificare esistenzialmente ed universalmente le variabili che compaiono.

Problemi con la Logica

- **Aspetti procedurali definiti una tantum e non sempre adeguati al problema**
- **Deduce tutte le conseguenze**
- **Spesso è necessaria una logica di ordine superiore, o con semantica diversa**

Gli aspetti procedurali sono definiti unicamente dal meccanismo inferenziale adottato, che è prefissato. Talvolta questo non è adatto ad affrontare un particolare problema.

L'apparato inferenziale della logica opera per dedurre tutte le conseguenze possibili, dato un insieme di assiomi e di asserzioni. Questo può essere a volte computazionalmente proibitivo, o, comunque, non desiderato.

Spesso la logica dei predicati del primo ordine non è sufficiente per rappresentare tutto ciò che si desidera, e diviene quindi necessaria una logica di ordine superiore, o comunque particolare.

Regole

<rule> ::= IF <premise> THEN <conclusion>

<premise> ::= <clause>|
<clause> OR <clause>

< conclusion > :: = <clause>

<clause> ::= <s-clause> | NOT <clause> |
<clause> AND <clause>

<s-clause> ::= <fact> | <pattern>

< fact > ::= (" sequenza atomica di simboli")

<pattern> ::= (" sequenza di simboli e nomi di variabili")

Il formalismo a regole è forse il più diffuso modo per rappresentare aspetti inferenziali in un sistema esperto.

A lato si riporta una grammatica che permette di rappresentare il tipo di regole più diffuse in cui possiamo avere AND, OR e NOT negli antecedenti e solo AND e NOT nei conseguenti.

Si noti che possiamo avere clausole composte da sole stringhe atomiche di simboli, che vanno confrontate in maniera diretta con fatti della stessa forma, oppure pattern contenenti nomi di variabili che partecipano al processo di pattern-matching.

Forward e backward

Modi per concatenare le regole

Es.:

if (circuito luci aperto) then (luci non si accendono)
if (fusibile luci fuso) then (circuito luci aperto)
if (corto circuito su circuito luci) then (fusibile luci fuso)

if (fusibile luci fuso) then (cambia fusibile luci)

Le regole possono essere concatenate forward o backward, a seconda degli scopi che si vogliono ottenere. Le prime tre regole presentate a lato, ad esempio, possono esser concatenate backward partendo dall'osservazione "le luci dell'auto non si accendono", per ricavare le cause di questo malfunzionamento. Le stesse regole possono essere concatenate forward, invece, per simulare cosa succede se c'è un corto circuito su circuito delle luci.

L'ultima regola è una regola che indica un'azione da fare, data una certa situazione corrispondente ai suoi antecedenti. Questo tipo di regole si chiamano "regole di produzione" e possono evidentemente essere concatenate solo forward.

Pattern-matching

Pattern:

sequenza di simboli contenente anche variabili

Es: if **($Color ?a #red)** and **($InstOf ?a #apple)** then **($ReadyP ?a True)**

Pattern-matching:

confrontare un pattern con una sequenza di simboli (ad es.: un fatto), assegnando alle variabili i valori corrispondenti

Es:

Fatti**:**

($Color #Apple1 #red) ($InstOf #Apple1 #apple)

Pattern-matching**:**

(?a #Apple1)

Inferenza**:**

=> (ReadyP #Apple1 True)

Conflict resolution

Cosa succede se più regole possono scattare?

- **scatta la prima che è entrata nel conflict set, oppure**
- **scatta la più recente, cioè si segue la linea di ragionamento corrente, oppure**
- **scatta la più ricca, cioè si sfrutta la conoscenza al massimo**

Accanto riportiamo alcune delle strategie usate per risolvere il conflitto che sorge quando più regole possono potenzialmente essere attivate, perchè tutte coprono la situazione descrittadalla base dei fatti. Una decisione è necessaria, in quanto i nuovi fatti introdotti potrebbero far scattare regole diverse.

Può capitare che le tre strategie riportate siano attivate in cascata.

Quando usare le regole?

Quando si ha conoscenza euristica

Problemi

- **Meccanismo di concatenazione fisso**
- **Non descrivono in modo naturale strutture dati (le usano)**

Le regole sono nate per catturare la cosiddetta conoscenza euristica, cioè il modo di operare di un esperto che affronta un problema.

Tra i problemi ad esse associati ricordiamo il fatto che su una base di regole di solito opera un solo meccanismo inferenziale definito a priori (e quindi potremmo avere problemi analoghi a quelli della logica).

Inoltre, le regole non descrivono strutture dati, ma possono esser applicate a strutture dati.

Test #5: Tecniche di rappresentazione

Rappresentare con frames, oggetti e regole la conoscenza contenuta nel seguente testo in modo che sia possibile rispondere alla domanda: come fa CAT ad andare dall'ufficio di Andrea alla Segreteria?

- Il robot CAT:
  - può andare avanti e indietro
  - può girare a destra e a sinistra di 30 gradi
  - sente oggetti avanti, dietro, a destra e a sinistra
- La porta di Andrea è sul lato Nord del corridoio
- La porta della Segreteria è sul lato Sud del corridoio
- La porta della Segreteria è a Est della porta di Andrea
- CAT è nell'ufficio di Andrea

Viene qui proposto un esercizio riassuntivo sulle tecniche principali di rappresentazione della conoscenza.

Si invita ancora a sviluppare dapprima il modello concettuale della conoscenza coinvolta, per poi giungere alla sua rappresentazione con una tecnica particolare.

Non si desidera che venga risolto il problema concettuale della pianificazione del muovimento, ma che si identifichi la conoscenza necessaria perche' un sistema di pianificazione possa operare