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

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.

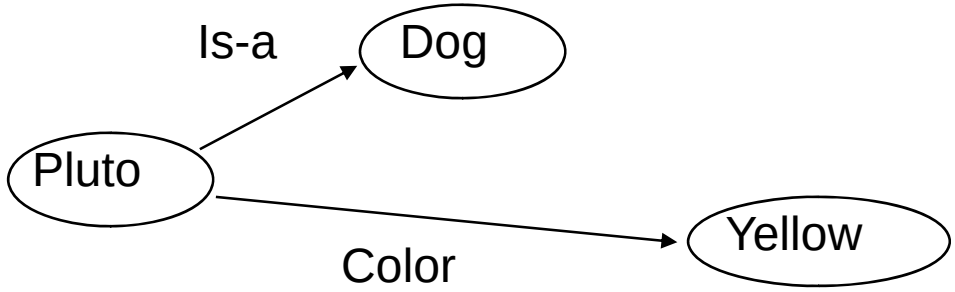
Descrizione procedurale.

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

Reti semantiche

Nodi = informazione atomica

Legami = relazioni tra nodi



Problemi con le reti semantiche

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

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))
```

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

Come definiamo i frames?

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.**

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.**

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**

Definizione di demoni

Ad <u>esempio</u>, 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)))))
```

Cosa si puo’ rappresentare con i demoni?

Conoscenza procedurale

Problemi

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

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**

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)))
```

Definizione di esemplari

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

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

Definizione di metodi (1)

```
(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)
```

Definizione di metodi (2)

```
(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

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.**

Logica

Logica dei predicati del primo ordine, ma non solo.

Apparato formale:

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

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**

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")

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)

Pattern-matching

Pattern:

sequenza di simboli contenente anche variabili

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

Pattern-matching:

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

Es:

Fatti**:**

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

Pattern-matching**:**

(?a #Apple1)

Inferenza**:**

=> (#Apple1 (ReadyP 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**

Quando usare le regole?

Quando si ha conoscenza euristica

Problemi

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

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