Corso di Linguaggi di Programmazione

Lezione 7

Alberto Ceselli
alberto.ceselli@unimi.it

Università degli Studi di Milano

19 Marzo 2013

Programmazione dichiarativa

- Programmi = Algoritmi + Strutture Dati
- Algoritmi = "Logica" + Controllo
- **programmazione dichiarativa:** utilizza una *notazione simbolica* per specificare la logica del programma
- la parte di *controllo* è trascurata dal programmatore

Esempio: AMPL

- **AMPL:** A Modeling Language for Mathematical Programming
- Linguaggio simbolico per problemi di programmazione matematica (alla lavagna).
- Algoritmo = Modello Matematico + Metodo Risolutivo
- Algoritmo = Programmazione Matematica + Branch-and-bound

Esempio: un "programma" AMPL

1. SETS: entità coinvolte
2. PARAM: dati del problema
3. VARIABLES: variabili del problema
4. OBJECTIVE: funzione obiettivo
5. CONSTRAINTS: vincoli del problema

Esempio: `zaino.mod, zaino.dat`

Esempio: AMPL

- Vant: notazione molto simile al modello matematico
- Vant: "logica" completamente svincolata dal controllo
- Svant: molto legato all'ambito applicativo
- Svant: Turing Equivalence?

Programmazione logica

- Obiettivo: separare la logica dal controllo, conservando *Turing-Equivalence*.
- Logica: calcolo predicativo (del primo ordine).
- Controllo: motore di inferenza (*resolution*).

Sebesta Cap. 16,
L. Sterling, E. Shapiro (1994) The Art of Prolog (SE), MIT press

Programmazione logica in pratica

- Risultato: PROLOG
- Un **programma prolog** è un programma logico in cui è definito un ordine sia tra le clausole nel programma che nei goals nel corpo delle clausole.
- PROLOG è la *realizzazione concreta* di un *modello di calcolo astratto*.

SWI prolog (`http://www.swi-prolog.org/`)

SWI prolog interpreter

L'interprete SWI: consult, edit, help, apropos, ...
Esempio: interrogazione di un 'database' prolog `prologdb (1)`.

Definizioni di base

- **Proposizione:** un'istruzione logica che può essere *vera* o *falsa*.
- **Proposizione atomica:** collezione di *termini composti*.
- **Termine composto:** un elemento di una relazione matematica, composto da due parti
  - un *funtore:* il simbolo di funzione (nome della relazione)
  - una *lista di elementi* (tupla)
- la cardinalità della lista è detta **arietà** della proposizione.
- Le proposizioni possono essere sia **fatti** che **queries**.

Definizioni di base

- Le proposizioni possono essere *composte* tramite connettori logici (not, and, or, equivalenza, implicazione).
- Nelle proposizioni possono apparire **variabili**, ma solo se introdotte da un **quantificatore** (universale o esistenziale).

Esempio: `prologdb (2)`

Definizioni di base

- Ci sono *molti modi* di esprimere *la stessa proposizione* (es. $A \rightarrow B = \neg A \vee B$).
- Obiettivo: minimizzare la *ridondanza* $\rightarrow$
- tutte le proposizioni sono considerate in forma di clausole

$$B_1 \vee B_2 \ldots B_n \leftarrow A_1 \wedge A_2 \ldots A_m$$

- la parte *destra* è detta **antecedente**
- la parte *sinistra* è detta **conseguente**
- i quantificatori non sono richiesti (implicito l'universale)
- ogni proposizione può essere convertita (Nilsson '71)

Esempio: `prologdb (3)`

Risoluzione

- Risoluzione (resolution): *algoritmo* per inferire nuove proposizioni da proposizioni date (Robinson '65)
- Esempio:
  - Input: $P_1 \leftarrow P_2$; $Q_1 \leftarrow Q_2$ ...
  - ... e supponiamo P_1 e Q_2 identici (T)
  - $T \leftarrow P_2$; $Q_1 \leftarrow T$
  - Output: inferiamo $Q_1 \leftarrow P_2$
- Esempio: parentela (alla lavagna)

Esempio: `prologdb (4)`

Risoluzione

- Estensione: se P_1 e Q_2 non sono identici (es. una *variabile* ed una costante)?
  - Input: $X \leftarrow P_2$; $Q_1 \leftarrow' anna'$ ...
  - **unification:** processo di ricerca di opportune variabili
  - **instantiation:** assegnamento (temporaneo) di valori alle variabili
  - Output: inferiamo $Q_1 \leftarrow P_2$, a patto che $X =' anna'$
- Esempio: parentela (alla lavagna)

(Esempio: `prologdb (5)`)

Risoluzione

- Quando le proposizioni sono utilizzate per risoluzione ...
- si restringe l'insieme delle clausole valide alle **clausole di Horn**:
  - `nonna(lina, silvia) :- madre(lina, anna), madre(anna, silvia)`
    (un termine a sinistra, almeno un termine a destra: headed).
  - `madre(lina, anna) (:-).`
    (un termine a sinistra, nessun termine a destra: "headless").
- Notare: "analogia" tra risoluzione con clausole di Horn e chiamata a sottoprogrammi.

Ancora definizioni!

In PROLOG

- un **termine**: può essere
  - una **costante**: **atomo** (es. `anna`, `silvia`) oppure **intero**
  - una **variabile** (es. `Anna`, `X`)
  - una **proposizione**.
- un **fatto** corrisponde ad una *clausola di Horn headless* (proposizione vera).
- una **regola** corrisponde ad una *clausola di Horn headed* (proposizione vera, quantificata universalmente).
- un **goal** (o query) è una *clausola di Horn headless* (di cui si vuole controllare la verità, quantificate esistenzialmente).

N.B. In PROLOG le variabili non hanno tipo, ma ne possono *assumono uno* durante il processo di unificazione.

Interprete e termini 'ground'

- Un termine si dice *ground* se non contiene variabili (libere).
- Es. programma senza variabili `connect.pl (1)`
- Es. `trace`

Interprete e unification

- Esempio: con unification / instantiation
- Es. `connect.pl (2), trace, ispath(a,c).`
- Es. `connect.pl (3), trace, ispath(a,c).`

Funzionamento di un interprete

Ci sono due tipi di risoluzione:

- **bottom-up**: l'interprete prova a combinare fatti e regole del database, cercando di ottenere il goal
  (efficace quando il numero di risposte corrette è elevato)
- **top-down**: l'interprete parte dal goal e sostituisce i sottogoal utilizzando fatti e regole del database
  (efficace quando il numero di risposte corrette è ridotto)

Funzionamento di un interprete

- Esempio 1:

```
A.
B.
C :- A.
D :- B, C.

?- D.
```

- Esempio 2:

```
father(jack).
father(bob).
man(X) :- father(X).

?-man(bob).
```

Funzionamento di un interprete

- **depth-first search**: sostituzione prima nel primo subgoal.
- **breadth-first search**: sostituzione "in parallelo" di tutti i sottogoal di un determinato goal.
- **backtracking**: quando un sottogoal non può essere dimostrato, l'interprete riconsidera l'ultima unificazione del sottogoal precedente, e considera altre alternative. (`backtrack.pl`)
- N.B. il controllo del backtracking può *migliorare l'efficienza* di un programma PROLOG.
- Esempio:

```
male (bob).
male (jack).
male (eugene).
parent(eugene, shelley).

?-male(X), parent(X, shelley).
```