di Raffaello De Masi

Prolog & Turbo Prolog

seconda parte

Una premessa.
Fermiamoci per un momento a considerare l'affermazione principale della puntata scorsa, e questo per due buoni motivi; innanzi tutto perché chi ben comincia è alla metà dell'opera, secondo perché mai come qui occorre entrare bene nel concetto di linguaggio descrittivo. Tanto per intenderci, Prolog è molto più prossimo al nostro modo di pensare di qualunque altro linguaggio. Come avviene in un romanzo, in cui vengono descritti i fatti lasciando sovente al lettore il processo di collegamento tra le parti in gioco, il programmatore in Prolog trascura tutto il processo di descrizione delle relazioni intercorrenti tra i fatti, per descrivere i fatti stessi. Inoltre, il che non dispiace ad uno che apprezza e vanta la libertà e la scioltezza di programmazione del Basic, è discretamente non lineare; non è detto (e d'altro canto non è stato mai dimostrato) che le cose debbano essere sempre esattamente al loro posto (con buona pace dei Pascalisti & C.). Generalmente, se si dimentica qualcosa in un programma, è sufficiente, nella maggior parte dei casi, battere l'istruzione o il fatto mancante, senza preoccuparsi di ridisegnare completamente tutto il flusso del programma stesso

Secondo una pittoresca ed efficace definizione di Dan Shafer comparsa su un numero di Byte di un paio di anni or sono, in Prolog i «processi» prendono il posto delle informazioni e le relazioni divengono parti («pezzi») del processo stesso. La mente umana non ragiona per procedimenti, mette solo insieme cose, informazioni: ma probabilmente un suggestivo esempio di questo autore chiarirà tutta la faccenda. Se leggendo un giornale scopriamo che c'è stato un incidente automobilistico lungo la strada che frequentiamo per andare a lavoro, la nostra mente collega questa informazione ad altre già in suo possesso, immagina noi stessi alla guida della nostra automobile in quella strada, e ne trae conclusioni (attenzione! la nostra macchina abbisogna di gomme nuove, occorre pagare la tassa automobilistica, bisognerà chiedere in prestito ad Andrea la Land Rover per portar via quei mobili dal garage) che, di certo, non erano nelle intenzioni di chi ha scritto l'articolo. La mente umana ragiona e segue un filo logico proprio cosi; non a caso si dice «Seguire il filo dei pensieri»; siamo di fronte ad una serie (solo apparentemente) casuale di connessioni tra eventi altrettanto casuali, ma capaci di giungere a decisioni, opinioni, operazioni tra le cose. Se la nostra vita di tutti i giorni è così ben regolata da un processo tanto aleatorio, vuol dire che questo modo di procedere non è poi tanto errato. Prolog funziona allo stesso modo; perché, quindi non usare un mezzo tanto vicino alla nostra mente anche su calcolatore, visto che poi il cervello umano non è altro che un calcolatore?

Premesso (ed ormai accertato) che Prolog è il miglior linguaggio disponibile per risolvere problemi coinvolgenti oggetti e relazioni tra gli oggetti, e considerato che la maggior parte dei programmi utilizzati su un calcolatore (word processing, basi di dati, sistemi esperti, ecc) coinvolge oggetti e relazioni, ne consegue che Prolog è uno dei migliori linguaggi per lo sviluppo di applicazioni.

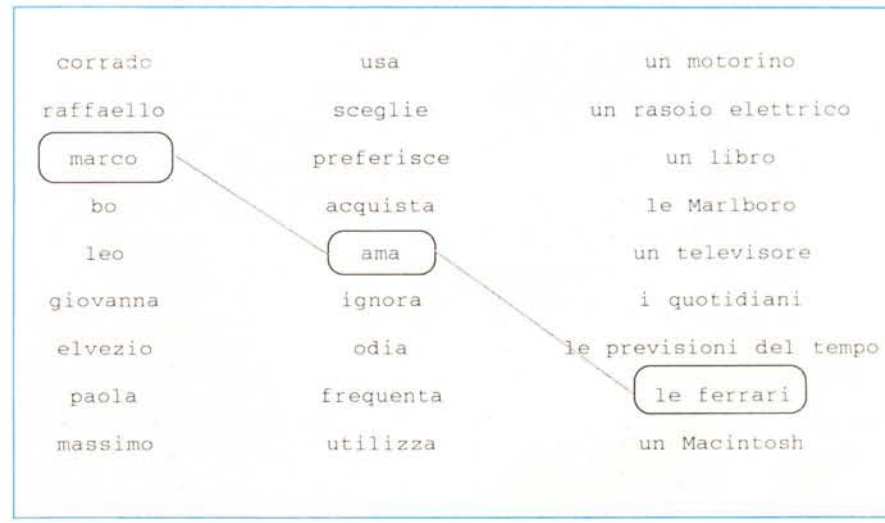

Figura 1 - Tabella di relazione tra oggetti.

Oggetti e relazioni tra di essi

Un «oggetto» (è l'ultima volta che lo inseriremo tra virgolette), in Prolog è quanto, in un costrutto grammaticale, in un discorso, occupa il posto del soggetto o del complemento oggetto; una relazione, invece, è il predicato che lega tra di loro gli oggetti; la similitudine col costrutto letterario non è peregrina e ci sarà utile diverse volte in futuro per chiarire certi passaggi che potrebbero essere non ovvi per chi è abituato a programmare in altri linguaggi.

Consideriamo la frase: «Marco ama le Ferrari» (ogni tanto bisogna pur incensare il proprio datore di lavoro!) e colleghiamo il tutto alla figura 1. Questa mostra una serie di oggetti, legati tra loro da relazioni; come si vede alla categoria oggetto possono appartenere cose concrete ed astratte, senza differenza e una relazione, come quella appena espressa ed evidenziata nella figura, può esistere tra oggetto e oggetto, tra oggetto ed idea e tra due idee tout court. Allo stesso modo la frase «Mazzini sognava la libertà d'Italia» è un collegamento tra idee e persone-oggetti. Ma spesso può accadere che quanto si esprime in un costrutto logico non sia eccessivamente chiaro; ad esempio la frase «Le Ferrari sono bellissime» coinvolge un oggetto, la bellezza, non facilmente esprimibile neppure nel linguaggio normale; in linguaggi programmatori convenzionali, pertanto, idee di tal genere sono pressoché impossibili da esprimere e manipolare; Prolog consente invece di realizzare connessioni di tal genere in maniera efficace e determinante.

La frase «Corrado possiede un IBM» è, come abbiamo visto, un collegamento tra oggetti (Corrado, IBM) legati tra di loro da una relazione di possesso. Nella lingua parlata, questo tipo di espressione si definisce dichiarativa, in contrapposizione ad una espressione interrogativa («Corrado possiede un IBM?») od

Figura 2.

```
10 '        questo programma consente di ricavare
20'         cosa preferisce marco
22 '        ricavato con modifiche da un esempio
24 '        di Ralph Roberts (1966,opera citata)
30 '
40 DATA  "CORRADO PREFERISCE  IBM"
50 DATA  "RAFFAELLO PREFERISCE MACINTOSH"
60 DATA  "BO PREFERISCE APOLLO"
70 DATA  "LEO PREFERISCE HEWLETT-PACKARD"
80 DATA  "ELVEZIO PREFERISCE OLIVETTI"
90 DATA  "MASSIMO PREFERISCE AMDHAL"
100 '
110 FOR K = 1 TO 6
120   READ A$(K)
130 NEXT K
140 INPUT "NOME" ; B$
150 IF B$ = "MARCO" THEN B$ = "CORRADO" ELSE GOTO
240
160 FOR K1 = 1 TO 5
170   C$ = ""
180   FOR K2 = 1 TO LEN(A$(K1))
190     IF MID$(A$(K1),K2,1) = "" THEN 220
200     C$ = C$ + MID$(A$(K1),K2,1)
210   NEXT K2
220   IF C$ = B$ THEN 250
230 NEXT K1
240 PRINT " DATI INSUFFICIENTI O NON CORRETTI" : GOTO 260
250 PRINT " MARCO " ; MID$(A$(K1),K2,LEN(A$(K1)) -K2+1)
260 END
```

Listato in Basic.

```
domains
     persona, calcolatore    = symbol

predicates
     preferisce(persona,calcolatore)

clauses
     preferisce(corrado,ibm)
     preferisce(raffaello,macintosh)
     preferisce(bo,apollo)
     preferisce(leo,hewlett-packard)
     preferisce(elvezio,olivetti)
     preferisce(massimo,amdhal)
     preferisce(marco,x)  if preferisce(corrado,x)
```

Listato in Turbo Pascal.

imperativa («Corrado, compra un IBM!»). Prolog intende tutte e tre i tipi di espressione (sebbene il primo tipo rappresenti almeno il 90% di un discorso). Le righe di un programma in Prolog sono rappresentate, appunto, da una grande quantità di espressioni dichiarative, organizzate da alcune frasi imperative; le frasi interrogative sono, invece, usate dall'operatore per interrogare il programma sulle sue conoscenze. Una serie di frasi come quelle precedenti appare, in Prolog così:

```
ama(marco,ferrari)
possiede(corrado,ibm)
```

e così via. Frasi di questo tipo sono completamente accette da Prolog; quale altro linguaggio accetterebbe ordini-frasi di questo tipo?

Facciamo un esempio, visualizzato in figura 2 (esempio derivato da un programma di Ralph Roberts presente nel volume «The power of Turbo Prolog»; una completa bibliografia ragionata sarà fornita alla fine di queste note); viene presentato un programma in cui si esegue una verifica ragionata delle preferenze, in fatto di calcolatori, di diverse persone. Si nota subito la differenza di lunghezza e complessità tra i due listati, in Basic ed in Turbo Prolog. Ma la cosa non finisce qui; proprio perché il Basic mal si adatta ad una situazione descrittiva come quella prevista (di procedurale, cioè di regole risolutive, come si vede, c'è ben poco) il programma, in questo linguaggio, non è cosa ben misera e debole; questo listato può rispondere solo alla richiesta: «Che cosa preferisce Marco se Marco preferisce quello che preferisce Corrado?» (scusate la cacofonia del tutto!); qualsiasi altra relazione tra gli oggetti-cose-persone presenti nel listato è sconosciuta al programma e genera solo una risposta di dati incorretti o non manipolabili. L'equivalente listato redatto in Prolog è più breve della versione Basic, è sicuramente molto più intuitivo, ordinato ed immediato nella comprensione del suo compito, oltre che notevolmente più potente. Ma consentite ad un entusiasta di questo linguaggio di evidenziare qualcosa di più: vi accorgete che, in questo listato, non ci sono comandi coinvolgenti procedure? Non è necessario, al Prolog, fornire regole od istruzioni destinate a leggere o comparare variabili o parti di esse. Prolog già sa, per conto suo, che cosa fare. Occorre solo dichiarare gli oggetti e le relazioni che tra essi intercorrono. Sempre continuando con l'entusiasmo, esiste un altro linguaggio che permette di descrivere una situazione così articolata in termini tanto chiari e succinti?

Le parti principali di un programma

Il listato mostra chiaramente che per stabilire cosa piace a marco (al contrario di quanto avviene in Basic, il Prolog utilizza preferenzialmente le lettere minuscole), occorrono tre categorie: la prima costituisce i domini, che specifica i tipi di valori che gli oggetti possono assumere tra di loro; più semplicemente potremmo dire, anche se in modo limitativo, che in questa fase vengono descritti i personaggi dell'azione, i tipi di oggetto che entreranno in gioco (persona; calcolatore), e viene inoltre indicata la loro natura (symbolo; avremo modo di essere più chiari in proposito tra breve).

La seconda è rappresentata dai predicati, le regole che governano i rapporti tra i domini [nel caso particolare viene specificato che la persona (soggetto) preferisce un calcolatore (oggetto)]. Anche qui si evidenzia la struttura descrittiva del programma; viene specificata una correlazione, senza stare troppo a precisare di che tipo di correlazione si tratta; avremmo potuto specificare «odia», «mangia», «segue», «getta_dal_balcone», il principio sarebbe stato altrettanto efficientemente rispettato.

L'ultima «area» del programma è rappresentata dalle clausole; è questa generalmente l'area più estesa, visto che raccoglie le regole spicce che coinvolgono ogni particolare predicato. Si tratta della più ampia (e se vogliamo la vera e propria) area dichiarativa del programma; corrado e ibm, raffaello e macintosh, bo ed apollo formano binomi indissolubili. L'ultima frase è un po' particolare; condizionando le scelte di marco a quelle di corrado, esegue un distinguo, che indica, in un certo qual modo, a Prolog la strada da imboccare.

Bene, abbiamo inserito le informazioni necessarie; Prolog farà il resto. Lanciamo il programma e vediamo cosa succede. Beh! non succede proprio niente; il programma non contiene procedure da seguire, quindi non esegue un bel nulla. Ed allora?

Per inquadrare bene il problema occorre fare mente locale ad un ragionamento del genere: con il nostro listato, abbiamo fornito al programma una serie di informazioni circa oggetti e relazioni tra essi intercorrenti. Il programma adesso è come se dicesse: «Bene, mi hai fornito certe informazioni: cosa desideri sapere?». In pratica sullo schermo si apre un riquadro, il «dialog window», la finestra di dialogo, che ci consente di parlare a Prolog per proporgli una serie di richieste. In gergo tecnico Prolog chiede un «goal», un obiettivo.

Il «goal»

Il «goal» è la quarta sezione da definire in un programma, ma a differenza delle prime tre può o no risiedere fisicamente nel programma stesso. Nel primo caso essa si presenta sotto la forma

```
goal
preferisce(marco,hewlett-packard)
```

ed in effetti avvisa direttamente il programma che il suo scopo è verificare se «marco, tenendo conto che preferisce ciò che piace a corrado, potrebbe gradire un HP». Se ciò non è, il programma diviene interattivo e chiede all'utente di fargli qualche domanda.

Poiché non abbiamo inserito alcun obiettivo nel programma la finestra di dialogo è pronta a ricevere ordini; battiamo

```
preferisce(marco,olivetti)
```

che, a tutti gli effetti equivale alla domanda «marco (soggetto) preferisce un calcolatore olivetti (complemento oggetto)?».

Avremo una risposta negativa, ovviamente («false»); ma la cosa può essere più raffinata, vale a dire che è possibile battere la richiesta:

```
preferisce(marco,Che_cosa)
```

incaricando la macchina di eseguire per conto suo lo scanning delle preferenze di marco. Il linguaggio, infatti, analizza la domanda, e, non ritrovando l'oggetto [Che_cosa] tra quelli presenti nel listato interpreta questo come una «variabile» che dovrà contenere l'oggetto preferito da marco; la risposta sarà

```
Che_cosa = ibm
```

Esiste una terza possibilità peraltro facilmente intuibile; se «Che_cosa» viene trasferito al primo posto con una richiesta del tipo:

```
preferisce(Che_cosa,ibm)
```

ci verranno restituite due risposte diverse del tipo:

```
Che cosa_= marco
Che cosa_= corrado
```

e certo tutto ciò, col programmino in Basic a fianco, non è proprio possibile eseguire.

Credo che sia il caso di fermarci qui con l'introduzione al Prolog. La prossima volta vedremo di illustrare le caratteristiche strutturali principali di un programma in Turbo Prolog, il vero pilastro su cui si basa la corretta individuazione e realizzazione di un efficiente programma. A risentirci! MC

RICORDI presenta:

Archimedes

La potenza del RISC nel personal computer più veloce del mondo

▷ Dalla Acorn di Cambridge, U.K., una nuova rivoluzione nell'informatica personale ▷ Archimedes, un computer (o meglio, un'intera serie) dalle altissime prestazioni ▷ Basato su un'unità centrale RISC (Reduced Instruction Set Computer) a 32 bit, Archimedes mette a vostra disposizione una potenza di calcolo finora sconosciuta nel campo dei personal computer ▷ Potenza per eseguire programmi in BBC BASIC a una velocità superiore a quella del linguaggio macchina di molti microcomputer tradizionali ▷ Potenza per accedere a diversi sistemi operativi, dall'ADFS all'MS-DOS* ad altri ancora ▷ Potenza per supportare linguaggi ad alto livello come C, FORTRAN, LISP, PROLOG, PASCAL (oltre a un BASIC formidabile) ▷ Potenza per generare un suono stereofonico di qualità digitale, e una grafica ad altissima definizione con migliaia di colori ▷ Potenza per collegare le più varie periferiche: digitalizzatori, interfacce MIDI, modem, eccetera ▷ Vincitore del Microcomputer Of The Year Award 1987 ▷ Archimedes, il personal computer più veloce del mondo, a un prezzo eccezionale: presso il vostro rivenditore o nei negozi RICORDI.

*MS-DOS è un marchio della Microsoft Corp.

Distributore esclusivo: G. RICORDI & C.
Settore Informatico
Via Salomone, 77
20138 MILANO
tel. 02/5082-315

DOPPIOUNI

Acorn
The choice of experience.
Un'azienda del gruppo Olivetti

Per maggiori informazioni, inviate questo coupon a G. RICORDI & C. Settore Informatico, Via Salomone, 77, 20138 MILANO

Desidero avere maggiori informazioni su Archimedes

Nome:

Cognome:

Qualifica professionale:

Ditta, Ente o Scuola:

Indirizzo: