di Raffaello De Masi

Elementi di Prolog

terza parte

I componenti di un programma di Prolog

La prima pietra dell'uso di un linguaggio è la conoscenza dei mezzi che questo linguaggio mette a disposizione; perdonate la banalità, ma come si fa a guidare una macchina senza conoscere i comandi? Perciò non è un rimangiarsi le parole circa la mancanza di rigidità e la libertà di impostazione di un programma se questa puntata sarà dedicata ai componenti fissi di un programma ed alla loro identificazione ed esplicitazione; a dispetto infatti della forma più libera di programmazione e di impostazione dell'algoritmo, anche il nostro ha bisogno, nell'individuazione generale del programma, di seguire certe impostazioni di base e di adottare certi caposaldi cui non può rinunciare. Eccoci quindi a descrivere la componentistica di questo linguaggio

Turbo Prolog, a differenza degli altri prolog, impone al programmatore una organizzazione generale dei componenti di un programma che, seppur non estremamente rigida, è pur sempre vincolante; gli altri rappresentanti del linguaggio, circolanti su mini e mainframe, invece provvedono, generalmente, autonomamente, a setacciare il programma ed a riordinare la struttura secondo l'esatta gerarchia. Perché con Turbo Prolog non è così; la imposizione è in questo caso voluta, e devo congratularmi, ancora una volta, con i progettisti della Borland per questa scelta. Ho infatti eseguito un piccolo esperimento su un mini del mio istituto, introducendo una serie di regole, obiettivi, fatti, inseriti in maniera volutamente errata e capace di indurre in confusione, e la macchina è entrata immediatamente in stato confusionale. Si tratta di poca cosa (in effetti il tutto consiste nell'ordinare quattro sezioni fondamentali); perciò non andate dicendo che, dopo aver fatto l'apologia dell'anarchia programmatica, ci stiamo rimangiando la parola.

Il problema dell'ordine non è certo peregrino se la stessa culla del Prolog, la Francia, sta ritornando sui suoi passi ed il Prolog II, di recente pubblicazione, impone alcune, generali, regole di ordinamento nell'ambito del programma stesso; non si tratta di una «diminutio capitis», di una restrizione eccessiva, visto poi che questa restrizione è, all'atto pratico, ben poca cosa. Questo vuole solo dire che alcune piccole regole di saper vivere, anche nel campo dei linguaggi, non creano problemi a nessuno e facilitano certo molto il lavoro al linguaggio. Una curiosità; il tentativo di standardizzazione del Prolog II prevede, guarda caso, regole di ordinamento e di gerarchia del linguaggio identiche a quelle già previste dal Turbo Prolog; una coincidenza? non credo! O forse gli estensori del nuovo standard si erano letti prima il manuale del Turbo Prolog? non penso neppure questo; probabilmente i tecnici francesi e quelli della Borland sono giunti allo stesso risultato seguendo strade diverse; ed il fatto che le conclusioni siano state identiche nella maggior parte dei casi, non è, di per sé, testimonianza della qualità del procedimento e della bontà delle conclusioni?

Componentistica principale di un programma

Della componentistica di un programma, vale a dire delle parti principali che lo costituiscono, abbiamo già accennato la puntata scorsa. Ci torneremo adesso per descrivere, una per una, le sezioni costituenti, ognuna con la sua sintassi.

La figura A mostra lo schema generale della struttura di un programma in Prolog e Turbo Prolog; ogni programma, come già avemmo modo di dire, è composto da un massimo di 4 sezioni, rispettivamente definite «domini», «predicati», «obiettivo» e «regole».

I domini

Sotto questo nome va la parte di programma che raccoglie gli oggetti, i tipi di dati, manipolati da programma, con i relativi nomi utilizzati nella manipolazione. Ovviamente esistono tipi diversi di domini (dati) disponibili per la manipolazione; in default essi sono sei, ma possono essere definiti ulteriori tipi, senza limitazione, a piacere dell'utente-programmatore.

Letteralmente, un dominio è una categoria specifica in cui un oggetto, elemento, entità, può essere definito; in intelligenza artificiale il sostantivo assume significato ben più ampio, ma per

meglio intenderci, e per limitare volutamente il campo delle ipotesi ammissibili su cui saremo chiamati a lavorare successivamente, diremo che un dominio è un nome attraverso cui vengono definiti oggetti a caratteristiche e manipolabilità diverse; ad esempio sono domini un singolo carattere, una stringa di lettere, un numero, una quantità booleana, ecc.

Si potrebbe dire che domini equivale al termine «variabile» di altri programmi; la cosa è vera fino ad un certo punto essendo limitativa, visto che il concetto di variabile, in Prolog è più ampio di quanto avviene in altri programmi. Ad esempio, in Turbo Prolog (come anche in Prolog) il vecchio concetto di variabile è stato ampiamente allargato; indipendentemente dal fatto che Prolog, come Basic, non impone alcuna dichiarazione di variabile, Prolog ammette variabili non identificate (tipizzate); ad esempio la stessa variabile può essere utilizzata successivamente per contenere stringhe o numeri, non solo, ma addirittura (cosa che farà inorridire i pascalisti, e comunque i puristi dei linguaggi), lo stesso tipo di dato, contenuto in una variabile, può essere usato per elaborazioni di tipo diverso a seconda delle necessità del programma.

Come dicevamo precedentemente, i domini predefiniti dal linguaggio sono raggruppati in sei gruppi principali, secondo quanto esposto nella figura B; di essi alcuni ci sono familiari già: integer, char, real, string, ecc. hanno praticamente molto del significato che assumono negli altri linguaggi; ma uno è completamente nuovo e degno di un certo interesse. Sarà comunque il caso di descriverli tutti con ordine.

Descrizione dei domini

Cominciamo con la descrizione dei domini numerici. Finalmente sono sparite tutte quelle arzigogolazioni insulse relative al livello di precisione delle variabili; le parole «single precision», «double precision», «unsigned» e così via sono bandite dal vocabolario Prolog; solo due sono i tipi di variabile previsti, *[integer]* e *[real]* mentre tutto il resto sparisce nel nulla, anche perché tutta quella complicazione di numeri (come avemmo modo di essere d'accordo, in una discussione con Giustozzi) serviva per complicare la vita al programmatore e renderla facile al calcolatore ... Nel tipo *[integer]* vengono conservati i numeri interi, in quello *[real]* i decimali. Ambedue possono essere con o senza segno, anche se, come in tutti i linguaggi, caratteri numerici possono essere conservati in domini stringa o carattere, ovviamente senza possibilità di utilizzo in calcoli (è il tipico esempio di un numero telefonico, di un conto corrente bancario, o della targa di un'auto; in Prolog l'uso di stringhe destinate a trattare numeri non manipolabili aritmeticamente è incoraggiato, per una serie di considerazioni circa l'uso della memoria che avremo successivamente modo di verificare).

Il dominio *[char]* è destinato ad accogliere singoli caratteri (non a caso char è il troncamento di character), e contiene uno ed un solo carattere (lettera, numero, simbolo d'interpunzione, simbolo speciale, addirittura carattere di controllo). Il simbolo *char*, proprio per le precisazioni circa la memoria di cui si diceva in precedenza, si differenzia, in maniera elevata, dalla stringa; addirittura il metodo di individuazione è diverso. *[char]* viene racchiuso tra singole virgolette, per cui sono valide le assegnazioni 'B', '$', ':' '7', ma non lo sono "A" (usate le doppie virgolette, riservate alle stringhe), 3 (senza virgolette, in questo caso il carattere viene inteso come numero), e 'mcmicrocomputer' (più di una lettera).

Il dominio *[string]* amplia, se così si vuol dire, il concetto di *[char]*, visto che in questo dominio è possibile conservare più di una lettera, numero o simbolo. L'uso di questo dominio è molto simile al maneggio delle analoghe stringhe in Basic; in pratica una stringa è una serie

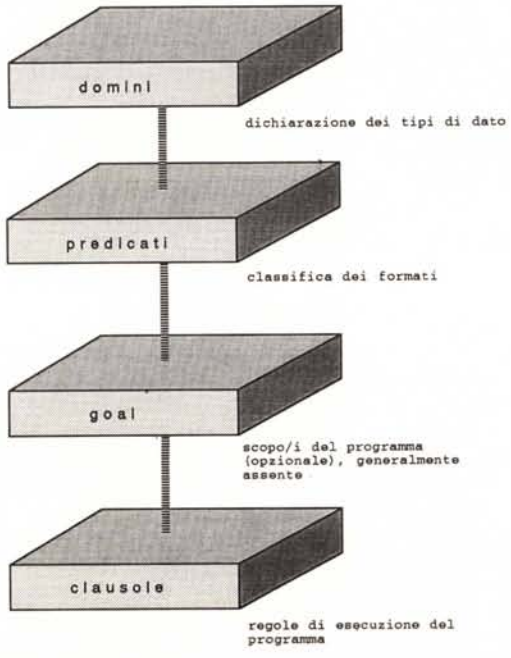

Figura A - Gerarchia generale di un programma in Prolog e Turbo Prolog.

domini	caratteristiche
char	lettera, numero o simbolo speciale, racchiuso tra singole virgolette
string	combinazione qualsiasi tra numeri, lettere e simboli, racchiusa tra doppie virgolette ["]
symbol	simile alla stringa, ma rappresentato da una sequenza senza spazi; non sono richieste virgolette
integer	numero intero
real	numero con cifre decimali
file	dati depositati su un supporto di memoria

Figura B - Tipi di domini e loro caratteristiche.

contigua di caratteri (si ricordi che anche lo spazio è un carattere) e, proprio come in questo linguaggio, essa viene racchiusa tra doppie virgolette. Saranno perciò esempi di stringhe valide le forme:

"mcmicrocomputer"
"1234567890"
"mannaggia 'a pressa, dicette 'a maruzza"

(a parte il significato dell'adagio delle mie parti, quest'ultimo esempio è stato fornito per evidenziare come il linguaggio accetti ogni sorta di carattere (tra cui anche il simbolo ['] destinato a manipolare il dominio *[char]*) fino alla chiusura della stringa stessa con il secondo ["]). Proprio nell'ottica della minore complicazione possibile per il programmatore, Prolog (e Turbo Prolog) ammettono la definizione di un dominio *[string]* anche senza l'uso delle virgolette, per cui assegnazioni del tipo:

precipitevolissimevolmente
bo_derek
il_nome_della_rosa_:_romanzo_di_Umberto_Eco

sono tutte valide, senza eccezioni. C'è però da far notare una particolarità, che sarà molto importante in seguito, che riguarda l'uso delle maiuscole. Esse, almeno in questa fase vanno abolite se presenti all'inizio della parola e se si desidera fare un'assegnazione di stringa senza usare le virgolette. Poiché Prolog utilizza le stringhe *inizianti* con lettera maiuscola per un suo scopo particolare, che vedremo tra qualche puntata, quando faremo un po' di programmazione logica un tanto avanzata, definizioni *[string]* del tipo.

MCmicrocomputer
GRATIS
MINISTERO_DELLA_DIFESA

non sono ammesse se non racchiuse tra ["].

Il dominio *[file]* è del tutto analogo all'omonima struttura presente in altri linguaggi, anche se, ovviamente le tecniche di manipolazione sono un tanto differenti; ne parleremo prossimamente.

Il dominio *[symbol]* è, in tutte le sue accezioni, e per quasi tutti gli usi di programmazione, abbastanza simile al dominio *[string]*, ed è, negli stessi casi, con esso intercambiabile; le differenze formali stanno nel fatto che *[symbol]* può essere costituito solo da una parola, senza cioè spazi separatori (l'underscore, il trattino di sottolinetura collegante due parole [_], non interrompe la continuità della parola) e senza, ancora una volta, iniziali maiuscole; quelle sostanziali consistono, invece, nell'importante principio che i simboli sono manipolati e valutati in maniera più rapida ed efficiente delle stringhe in quanto i simboli sono raccolti, dal programma, in una tavola di «lookup», supervisione, sempre a portata di mano, per modo di dire, del «main». Il rovescio della medaglia è rappresentato dal maggior uso della memoria, e nel fatto che la stessa operazione di inserzione di simboli, che pur rappresentano un mezzo per velocizzare l'esecuzione del programma, è di fatto più lenta nel processo di definizione.

Turbo Prolog ammette un altro tipo di dominio; le *[list]*. Pur se parleremo più diffusamente in seguito di questo tool fondamentale per un gran numero di applicazioni, possiamo adesso dire che può contenere ogni tipo di domino standard, ad eccezione (ovviamente) dei file. Ogni lista è rappresentata dall'equivalente dominio seguito da un asterisco [*] (es: string*).

In aggiunta ai domini fondamentali appena descritti, Turbo Prolog ammette la definizione di domini personali, creati secondo le specifiche del programmatore. Si tratta di una procedura piuttosto simile alla creazione dei [tipi] del Pascal, anche se con scopi e funzionalità diverse. Tanto per chiarire, nell'esempio della puntata scorsa era possibile individuare due domini nuovi, diversi tra di loro, come «persone_che_usano_un_calcolatore» e «macchina_preferita»; ogni dominio è, quindi, rappresentato da simboli custom.

Nella redazione di un programma, è tecnica comune di definire i domini, nella sezione «domini» del programma, riunendo insieme quelli dello stesso tipo. Viene, a tutti gli effetti, eseguita una dichiarazione, rappresentata da una lista di uno o più elementi, separati da virgole, ed affiancata, a destra, da un segno di = e dalla dichiarazione di tipo attinente; tanto per intenderci una sezione di dichiarazione è così rappresentabile:

```
domains
moto, automobili, targhe = symbol
nome = string
categoria = char
costo, spese_accessorie = integer
consumo = real
```

Comunque, in ossequio al principio che i linguaggi di programmazione sono fatti per semplificare la vita al programmatore e non alla macchina, non è assolutamente necessario che un programma in Turbo Prolog contenga una sezione dichiarativa. Ciononostante devo ammettere (a malincuore) che assegnare, all'inizio dei programmi, un'area riservata alle dichiarazioni di variabile, pardon, di dominio, magari accompagnato da remark esplicativi, consente di rendere più chiara la lettura del programma sorgente ed il suo debug. Turbo Prolog possiede, in questo caso, una diagnostica abbastanza efficace per la rilevazione di errori di assegnazione; tutto questo non fa altro che facilitare un po' la vita del programmatore, con ulteriore dimostrazione dell'assunto.

I predicati

Al contrario di quanto avviene, come abbiamo visto, per i domini, la sezione dei predicati è del tutto necessaria; i predicati rappresentano il cuore del programma, così come le regole ne rappresentano il motore; tanto per tentare una analogia (cosa sempre pericolosa tra linguaggi, ed ancora di più per un idioma tanto anticonvenzionale come il Prolog), i predicati rappresentano quelli che altrove sono chiamati specificatori, intendendo, con ciò, la fornitura di informazioni, al sistema, del tipo di regole che verranno successivamente eseguite.

Una definizione abbastanza calzante di predicato è quella espressa da D. Shafer nel suo volume dedicato al Turbo Prolog, dove dice che un predicato è come un progetto di massima, uno schema per le regole che seguiranno, contenute nella parte successiva del programma.

Una dichiarazione di predicato, che, lo ricordiamo, è parte necessaria del programma, per ovvi motivi, definisce univocamente il nome dei predicati e tipo, nome e numero degli argomenti che gli saranno assegnati. La sola regola circa i nomi dei predicati è che devono sempre iniziare con una lettera minuscola.

Non esistono limitazioni al nome dei predicati (tranne, ovviamente, che non sono ammesse più parole), ciò consente, secondo una regola di saper vivere informatico che non guasta mai, qui come altrove, di adottare nomi mnemonici per i predicati. Così

addiziona(addendo_1, addendo_2)

è certamente una buona regola di scrittura di un predicato.

Gli argomenti

Praticamente li abbiamo appena visti; [addendo_1] ed [addendo_2] sono argomenti, vale a dire valori forniti al programma (come costanti o variabili) perché questo, attraverso istruzioni o clausole, li possa adeguatamente manipolare. Espressioni del tipo

somma(spese,guadagno)
aggiungi(imponibile,iva)
dividi(12000,30000,somma_delle_altre_spese)

sono tutte forme di utilizzo di predicati. La definizione dei predicati è abbastanza semplice.

Il primo dei nostri esempi potrebbe essere

```
predicates
somma(integer,integer)
```

I goal

Il [goal] (obiettivo) è la seconda delle frazioni del programma ad essere opzionale (anzi, per essere precisi, quasi tutti i programmi in Prolog mancano di questa sezione). Il motivo è presto detto. Il [goal] è lo scopo del programma; se si desidera essere interattivi con questo, vale a dire se si vuole che il programma sia capace di seguire direttive esterne, a seconda delle esigenze dell'utente, generalmente non si inseriscono [goal] all'interno del programma.

Viceversa, se lo scopo del programma è sempre lo stesso la presenza di una sezione [goal] eviterà la seccatura di digitare alla tastiera richieste circa la fornitura del risultato.

Consideriamo l'esempio:

```
/* calcolo del costo totale di un oggetto */

predicates
        somma( real,real,real)
        stampa(real)
goal
        somma( imponibile,iva,totale),
        stampa(totale).
clauses
        imponibile = 20000.
        iva = imponibile *.18.
        somma(variabile1,variabile2,variabile3)
        if
        variabile3 = variabile2 + variabile1.

/* fine del programma*/
```

Il programma si spiega da solo; esaminati i predicati, il programma legge gli obiettivi e cerca di eseguirli secondo le regole trovate in [clauses] (si noti la *s* finale). Poiché Prolog consente di accettare variabili ed obiettivi da tastiera, si vede come è preferibile tenere il programma svincolato da formule di soluzione fisse.

Le "clauses"

La traduzione letterale (direttive, clausole) di questo termine evidenzia lo scopo ed il significato della sezione.

Esso rappresenta, come dicono oltre oceano «the genuine engine», il motore, il cuore, il fulcro vitale del programma; una sezione [clause] ben redatta e correttamente impostata significa aver svolto il 90% del lavoro di programmazione. Il bello sta nel fatto che, al contrario di quanto avviene in tutti gli altri linguaggi, il buon programmatore in Prolog (adesso dirò una cosa che farà venire le convulsioni ai pascalisti) ammucchia tutte assieme le regole di cui avrà bisogno nel programma, salvo poi a farvi riferimento nella sezione goal; vi sembra da folli? E noi che cosa facciamo quando diciamo, ad esempio, «Chiama un tassì» o «Fate il tagliando all'auto?»; esattamente enunciamo un [goal] che fa riferimento ad una serie di [regole], istruzioni, operazioni già definite ed ammucchiate nel cervello della nostra segretaria o del meccanico della concessionaria; capito che significa utilizzo del linguaggio naturale?

Le clausole, in Turbo Prolog, sono formalmente rappresentate da due componenti principali; i fatti [fact] e le regole [rule]. I primi esprimono delle relazioni tra oggetti; esempi corretti di fatti sono:

```
preferisce(mirella, yogurth)
parte(giovanni, milano)
eguale(base,altezza,area)
```

che, in linguaggio naturale significano

```
mirella preferisce lo yogurth
giovanni parte per milano
area è eguale a base per altezza
```

Ovviamente sarebbe assurdo pensare che questi [fatti] siano già predefiniti; niente dice al linguaggio che Mirella preferisca lo yogurth e non viceversa.

A definire i rapporti provvedono le regole, che servono a precisare come i fatti interagiscono nel loro interno e successivamente tra di loro per formare nuovi fatti. Esiste una forma generale di regola che può essere così definita:

```
AAAAAA      if      BBBBBB
```

con ovvio significato; BBBBBB rappresenta generalmente una serie di comandi, istruzioni o, addirittura una serie di regole (caso del tagliando dell'auto: cambiare l'olio, le candele, i filtri (comando), leggere il contachilometri per apporre il cartellino (istruzione), sostituire i freni se consumati (regola nidificata), ecc. È in base alle regole che Prolog riesce a trarre le sue conclusioni, e sono proprio le regole che governano le inferenze e rendono Prolog del tutto diverso dagli altri linguaggi.

Formalmente ogni clausola, sia essa fatto o predicato, deve possedere un corrispettivo nella sezione [predicati]. Eseguita questa indispensabile connessione, vediamo ora la differenza, a livello di sintassi, tra fatti e regole. Il costrutto formale di un fatto è del tutto simile a quello di un predicato (preferisce (corrado ibm)). tranne che questi terminano con un punto fermo [.]. Come si vede dall'esempio, un fatto è rappresentato da una denominazione, da argomenti (che possono anche mancare), separati da virgole [,] e da un punto finale.

Maggiore attenzione merita il costrutto della regola. Formalmente le regole si dividono in due parti, la testa [head] e la coda [tail] ruotanti attorno al simbolo [if]. Anche qui la fine della regola è segnalata da un punto ma, per dirla in gergo zoologico, una regola può essere pluricaudata; ogni coda è individuata e separata dalle altre da connettori logici ([and], [or]). Sebbene non sia obbligatorio è prassi comune che ogni coda, col suo relativo connettore, sia individuata in linea separata. Sotto questo punto di vista la cosa è perfettamente fattibile, visto che i [CR] non hanno alcuna importanza e il delimitatore di istruzione è rappresentato da un punto (che anche qui è obbligatorio).

I commenti

E siamo giunti alla cenerentola della sezione, il commento, che oltre tutto abbiamo già visto nell'esempio precedente. Un commento è, formalmente, una o più linee di testo presenti in un programma, escluse dalla esecuzione, e destinate ad evidenziare, nel corpo del programma, indicazioni informali del programmatore. Un commento può trovarsi dappertutto, nel codice sorgente, e viene automaticamente ignorato dal compilatore in fase di redazione del codice oggetto; esso inizia con il simbolo [/*] e termina con il suo reciproco [*/].

Inutile dire che i commenti rappresentano una delle pietre miliari della programmazione, qui come in qualunque altro linguaggio; abbondare in essi non è mai un difetto, visto che ad ognuno di noi sarà capitato di scrivere un codice chiarissimo (sic!) e ritrovarsi dopo una settimana a sbattere la testa per capire cosa intendevamo fare in quel punto. Termina così la nostra terza puntata, con un'esposizione più dettagliata della componentistica di un programma.

La cosa che viene più immediata da rilevare è che ci sembra di avere, almeno con quello che si è visto finora, ben poca cosa per poter costruire programmi complessi; invece è vero il contrario, tenendo conto che è inutile avere a disposizione una serie di regole da applicare in un mondo dove non esistono regole da seguire se non quelle della comune logica (che d'altro canto, almeno lo speriamo, non fa difetto a nessuno). Vedremo, fin dalla prossima puntata, come, con questi piccoli strumenti, è possibile realizzare costruzioni sempre più complesse ed efficienti (non a caso prolog sta divenendo lo stato dell'arte nella programmazione dei sistemi esperti, dove va rapidamente soppiantando il Lisp, afflitto da problemi di comprensibilità e di ordine interno); a risentirci! MC