di Raffaello De Masi

Elementi di Prolog

Avevamo annunciato, nella scorsa puntata, il trattamento delle tecniche numeriche del Prolog.
Cosa strana per la verità, in un linguaggio di A.I., destinato più a manipolazione simbolica che algebrico-matematica. Ma tant'è, visto che, ormai, la specializzazione è preferita da pochi e i linguaggi tendono ad essere i più articolati, polivalenti ed efficienti possibile.
Ne abbiamo, una prova con il LISP, linguaggio molto più datato del Prolog e, legato a tecniche e vedute estremamente tradizionali anche per quanto riguarda l'utenza specifica. Motori inferenziali Lisp sono oggi adottati nelle più avanzate tecniche di elaborazione numerica e, addirittura di grafica; nessuna sorpresa; proprio su queste pagine, nella rubrica dedicata al Mac, abbiamo un paio di mesi or sono provato un Logo (l'object, per la precisione) che darebbe filo da torcere ai più dinamici «C» o ai più sussiegosi Pascal.
Merito delle macchine e dei loro processori, certo, ma merito anche di più efficienti software di gestione dei linguaggi, che, tanto per rimanere nello stesso esempio, mettono a disposizione almeno duemila primitive (statement) tra le più efficienti e specializzate.
Non a caso la Coral, produttrice del logo in parola, ha realizzato un Lisp con cui alcune software house si sono cimentate nella realizzazione di applicazioni di gran pregio, tra le quali un word processor che può dare punti all'eccezionale Word 4 della Microsoft. Perciò niente problemi se vediamo il nostro bel Prolog alle prese con problemi numerici

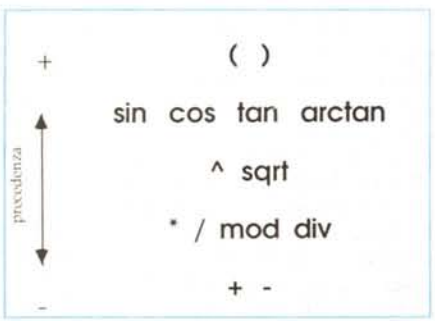

Tabella degli operatori numerici e delle loro precedenze relative.

Ovviamente, dopo questa sviolinata iniziale, ci si aspetterebbe di trovare, nel nostro Turbo, fior di operatori algebrici e matematici; vero, fino a un certo punto. Inutile cercare comunque la prestazione sofisticata (che magari non troverete neppure nel bellissimo MacFortran della Microsoft, linguaggio splendido per la realizzazione di ambienti di elaborazione numerica; ad ognuno il suo, tanto per continuare con le citazioni); perciò accontentiamoci di quello che vedremo in questa puntata, anche se non è moltissimo!

Nonostante quanto abbiamo appena detto occorre precisare che il Prolog, e in particolare il Turbo Prolog, non è un linguaggio adatto a sviluppare avanzate applicazioni in campo matematico. Tanto per intenderci è poco pratico chiamare il nostro per la realizzazione di uno spreadsheet ad avanzate caratteristiche di calcolo, come un Trapeze, un Full Impact o il mirabile Excel. Il problema non sta tanto nella potenza o meno degli operatori a disposizione, quanto nella maniera per così dire inusuale con cui gli operatori numerici si comportano in Prolog. Ciononostante è inutile recriminare; ogni linguaggio ha i suoi buoni e cattivi lati, le sue «preferenze», ed è inutile piangerci su. D'altro canto a nessuno verrebbe in mente di sviluppare applicazioni di A.I. usando linguaggi come il Pascal o il Cobol.

Ciononostante è opportuno ricordare che numerose versioni di Prolog, tra cui anche il Turbo, possiedono implementati abbastanza efficaci operatori algebrici, trigonometrici e, più in generale, matematici.

In questa puntata esamineremo le funzioni più comuni presenti nel Prolog della Borland, evidenziando, alla fine, come (ma questo è possibile in tanti altri idiomi) sia possibile ampliare la gamma degli stessi ove ce ne fosse bisogno.

Agli inizi, sui Prolog più anziani, una espressione del tipo:

```
A=2+3+5+7?
```

avrebbe dato una risultanza ben strana: ci saremo aspettati 17 come risposta, mentre avremmo avuto un imperturbabile risultato del tipo:

```
A=2+3+5+7
```

Che cosa è successo?

Niente di grave; non dimentichiamo lo spirito simbolico del linguaggio; poiché Prolog non sa che cosa sia «X» non ha fatto altro che assimilare la variabile all'espressione che gli abbiamo assegnata, tal quale, senza tentare su di essa alcuna valutazione. Tutti i Prolog, per superare il problema, disponevano di un operatore ad hoc, [is] che andava così usato:

```
A is 2+3+5+7
A=17
```

che eseguiva la forzatura dell'operazione insita nella espressione stessa. Turbo Prolog conserva questa istruzione, in rispetto alle regole del gioco ed ai suoi più vecchi genitori-fratelli, ma, a una espressione del più classico tipo, risponde eseguendo esso stesso la forzatura e fornendo il risultato esatto, ammesso, ovviamente, cosa che avevamo dimenticato di accennare, che ad [A] sia stata assegnata la esatta collocazione nei domini numerici.

Come già avevamo accennato nella discussione relativa alle stringhe, gli operatori aritmetici vanno usati nelle regole [rule] con i connettori operativi onde poter ottenere i risultati e usarli nei rispettivi [goal]. Vediamo subito qualche esempio.

Gli operatori aritmetici

In Prolog sono disponibili sei operatori aritmetici; attraverso di essi è possibile eseguire addizioni, sottrazioni, moltiplicazioni e divisioni, oltre il modulus (calcolo del resto della divisione) e l'elevamento a potenza (con tutti i suoi risvolti). Vediamone qualche esempio immediatamente.

In Turbo Prolog (come abbiamo accennato precedentemente), per eseguire l'addizione basta semplicemente scrivere un goal; ad esempio:

```
Goal:A=+4
A=9
1 Solution
Goal:
```

è la serie di operazioni che consente di eseguire una somma e di ottenere il risultato. Occorre notare che non è possibile battere semplicemente:

```
4+5
```

in quanto avremmo un errore (Turbo Prolog immagina che l'utente stia tentando una ridefinizione dell'operatore [+], e, ovviamente, blocca l'operazione. Ovviamente lo stesso ragionamento va-

le per gli altri operatori). Resta pertanto confermata la regola indissolubile che tutti gli operatori aritmetici vanno utilizzati sotto forma di istruzione in un [Goal], o, anche, in una regola, così come era avvenuto anche nel caso degli operatori letterari e di stringa.

Allo stesso modo funziona la sottrazione:

```
Goal:A=11-5
A=6
1 Solution
Goal:
```

e la relativa composizione tra somme e differenze:

```
Goal:A=5+6-10
A=1
1 Solution
Goal:
```

Per i noti motivi di confusione con la lettera «x» la moltiplicazione, come è noto, viene espressa dal simbolo [*], scelta che rappresenta forse una delle rare coincidenze tra linguaggi informatici.

La moltiplicazione, in Prolog viene manipolata come segue:

```
Goal:A=3*5
A=15
1 Solution Goal:
```

Lo stesso problema di notazione compare per la divisione che abbandona, nel gergo delle macchine, i due punti [:] per passare alla barra inclinata [slash, /].

Anche qui niente di differente rispetto a prima ma vedremo subito alcune importanti differenziazioni che coinvolgono, manco a dirlo, il mondo degli interi, dei reali e delle virgole fluttuanti.

Partiamo col primo esempio:

```
Goal:A=12/4.
A=3
1 Solution
Goal:
```

niente di difficile, anche perché fortunatamente il rapporto tra 12 e 4 è senza resto.

Ma cosa succede quando si divide 14 per 4?

```
Goal:A=14/4
A=3.5
1 Solution
Goal:
```

Il segno di divisione forza, ove, ovviamente, la relativa variabile sia stata opportunamente dimensionata, il risultato in virgola fluttuante (numeri reali). Per la divisione intera intervengono due nuovi operatori, [div] e [mod].

Il primo esegue la divisione riportando solo il quoziente in numero intero, il secondo riporta il resto della divisione. Così in tutti e tre i casi avremo:

```
Goal:A=14/4
A=3.5
1 Solution
Goal:
Goal:A=14 div 4
A=3
1 Solution
Goal:
Goal:A=14 mod 4
A=2
1 Solution
Goal:
```

chiaro, no? Ma osservate l'esempio seguente:

```
Goal:A=14 div 4 and B=14 mod 4
A=3, B=2
1 Solution
Goal:
```

Ambedue le operazioni vengono effettuate contemporaneamente. Ma se usiamo per bene quello che abbiamo finora imparato, possiamo scrivere:

```
Predicates
diviso(integer,integer)
Clauses
diviso(A,B)if
Quoziente=A div B and
Resto=A mod B and
Write(Quoziente),nl,Write(Resto),nl
```

da cui ponendo la questione:

```
Goal: diviso(15,4)
3
2
1 Solution
Goal: diviso(12,3)
4
0
1 Solution
Goal:
```

il tutto essendo abbastanza chiaro, ancora una volta.

L'elevazione a potenza non contiene in sé alcuna complicazione concettuale; l'esponenziazione è operazione simile a quella che avviene negli altri linguaggi (con la solita necessità di evidenziare la definizione delle variabili). Ad esempio il cubo di 4 (4 alla terza potenza), che vale 64, viene rappresentato in Prolog, dall'espressione:

```
Goal:A=4^3
A=64
1 Solution
Goal:
```

Ovviamente l'operazione di esponenziazione può essere utilizzata in maniera abbastanza raffinata, ad esempio per calcolare radici. La sola radice quadrata, in Prolog, è comunque rappresentata da un operatore separato, [sqrt], operatore che esegue la forzatura e trasforma in numero reale il risultato. Avremo così:

```
Goal:A=sqrt(16)
A=4
1 Solution
Goal:A=sqrt(67).
A=8.185352772
1 Solution
Goal:
```

questo ovviamente in base al numero di cifre dopo la virgola consentito dall'implementazione del Prolog che si sta usando.

Non potevamo terminare questa puntata senza aver ovviamente parlato della precedenza nelle operazioni di calcolo. È la solita questione della priorità, già trattata su queste pagine, e comune alla maggior parte degli altri linguaggi, e che presenta le stesse soluzioni; vale a dire che esiste una priorità nello sviluppo degli operatori, e che è possibile forzare queste operazioni mediante l'uso di parentesi.

La scala delle priorità è quella ben nota (e che viene evidenziata in figura). Si va dalle operazioni a più alta priorità, come l'elevazione a potenza, a quelle più basse, come l'addizione e la sottrazione. Un esempio dei risultati ottenibili è:

```
Goal:A=(2^3*4-22+1
A=11
1 Solution
Goal: A=2+3^3*4^2-22/ .5.
A=390
1 Solution
Goal:
```

L'uso delle parentesi consente di complicare ancora di più le cose; il loro uso è particolarmente prezioso se si considera che senza di esse certe forzature sarebbero per lo meno impossibili; consideriamo l'esempio:

```
Goal:A=(3+2)*5
A=25
1 Solution
Goal:A=((3+2)*4-(3+3^2)*5)/2.
A=20
1 Solution
```

esso è abbastanza chiarificatore dell'uso delle parentesi.

Infine un breve sguardo alle funzioni trigonometriche disponibili in Prolog; in questo caso occorre ricordare che molte di queste funzioni possono essere ricavate per manipolazione di quelle già esistenti, e, in particolare, è lecito, stringando il tutto, ridurre le «primitive» a due, seno [sin] e coseno [cos]; Turbo Prolog fornisce altre due funzioni built in, la tangente [tan] e l'arcotangente [arctan]. Resta invece la pesante limitazione di dover fornire l'angolo in radianti, cosa assurda quando una semplice routine implementata anche sulle più banali calcolatrici, permette di settare, dall'inizio, l'unità di misura considerata. Misteri dell'informatica, ma tant'è, e ce lo dobbiamo tenere.