di Raffaello De Masi

Il concetto di eguaglianza

In questa puntata ci intratterremo sul concetto di eguaglianza-equivalenza e sui relativi operatori disponibili in Prolog. Quando si pongono delle domande, quando si formulano dei «goal», quando si definiscono regole, non si fa altro (qui come in altri linguaggi, d'altro canto), che eseguire delle comparazioni di eguaglianza o equivalenza tra i componenti delle nostre basi di conoscenza.
Il vantaggio di Prolog rispetto ad altri linguaggi è che, proprio perché coinvolge soprattutto concetti logici (oltre che matematici, algebrici, ecc.), è stato dotato dagli implementatori di operatori di comparazione molto efficienti e raffinati. È di ciò che parleremo in questa puntata

Discorrere di eguaglianza e diseguaglianza non è così facile, come non lo è nel mondo degli uomini; per essere più precisi potremmo dire che esistono diverse forme, qualità, quantità di eguaglianza e di converso, di diseguaglianza. Ad esempio una cosa può essere più grande, minore, o addirittura non comparabile o contraria. I mezzi per eseguire questi confronti sono chiamati operatori logici, e sono riassunti nella tabella A, che evidenzia anche il loro significato.

Parlare di eguaglianza e diseguaglianza tra numeri è cosa abbastanza intuitiva; 5 è senz'altro più grande di 4; così:

```
Goal:4>1
True
Goal:
```

darà una risposta senz'altro poco sorprendente.

Un poco più sottile è la domanda:

Operatore	Significato	Esempio		Simbolo mat.
>	maggiore	5>2	(true)	>
		5<2	(false)	
<	minore	10<12	(true)	<
		10>12	(false)	
=	eguale	5=3+2	(true)	=
		5=2+2	(false)	
<=	minore o eguale	5<=6+7	(true)	≤
		5<=3+2	(true)	
		5<=3	(false)	
>=	maggiore o eguale	5>=3	(true)	≥
		5>=5	(true)	
		5>=5+2	(false)	
<> [><]	diverso	5<>4	(true)	≠
		5<>3+2	(false)	

Tabella A - Gli operatori logici in Turbo Prolog.

```
Goal: "Maria"+4"maria"
False
Goal:
```

Come è possibile la cosa o, in altri termini, quale è la tecnica del sistema per stabilire che «maria» è più grande di «Maria»? Il fatto si giustifica facilmente per chi ha una certa tecnica di programmazione anche in linguaggi diversi dal Prolog in particolare. In pratica il sort delle stringhe tiene conto, come al solito, di una tabella di ordinamento che consente di «mettere in fila» non solo numeri, ma lettere, segni particoli, od altro. Lo standard di ordinamento ormai accettato universalmente è l'ASCII (tranne alcuni residui colpi di coda dell'Holleryt), l'American Standard Code for Information Interchange, sistema di rappresentazione numerico-formale destinato a rappresentare e trasformare in numero qualsiasi segno (lettere, numeri, punteggiatura, operatori logici e numerici, caratteri speciali) ottenibile con la tastiera. In tale codice, ormai di dominio incontrastato e riconosciuto, le lettere maiuscole vengono prima delle minuscole, vale a dire che possiedono un numero identificativo inferiore; in questo modo, poiché il sorting viene eseguito, preferenzialmente, in base alla prima lettera delle stringhe da comparare, «M» di Maria viene prima di «m» di maria. In altri termini Prolog esamina le stringhe che gli vengono fornite in base al primo carattere a sinistra (qualcosa di simile a quello che noi facciamo cercando in un vocabolario o in un elenco telefonico); ove mai ci fosse eguaglianza tra di essi, passa al secondo e così via. È questo il motivo per cui Maria viene prima di Mario, ma Mare viene prima di Maremma (non perché, dopo la «e» non ci sia più nulla, ma perché lo spazio dopo questa lettera ha un codice ASCII inferiore a quello della «m»).

Al contrario, se le stringhe da comparare hanno la stessa lunghezza e sono eguali, carattere per carattere, il risultato finale è [True]. Gli operatori mostrati nella tabella A sono disponibili in qualunque versione del Prolog, e tramite essi e i connettivi di cui abbiamo già parlato è possibile porre Goal anche molto complessi e raffinati.

Anche i numeri possono essere confrontati tra di loro con questi operatori.

La cosa è abbastanza diretta specie se si parla di interi, numeri senza parte frazionaria o decimale, dove il confronto è abbastanza diretto.

Il terzo operatore considerato nella tabella, [=], può, nella maggior parte dei linguaggi, significare due cose diverse.

Il suo uso più diffuso, anche se limitato al solo mondo dell'informatica, è quello di assegnazione; in Basic, come in Fortran o in C, il costrutto.

```
A=5
```

vuol dire, tout court, «Assegna il valore 5 ad A».

Il secondo significato, più vicino alle regole della matematica formale, tiene conto delle funzioni di comparazione, il segno di eguale consente di confrontare due quantità tra loro: ovviamente in questo caso occorre qualcos'altro, rappresentato generalmente da un «IF», un «WHEN» o un «UNTIL». Ad esempio, potrebbe essere necessario evidenziare un messaggio o scegliere certe conclusioni, in base ad un confronto tra valori, come un conto in banca o una lista di messaggi. Il discorso ci porta, qui, un po' lontano e riprende un concetto, sviluppato qualche tempo fa, che riguardava l'istanziazione (l'inizializzazione delle variabili in Prolog). Come ricorderemo, l'assegnazione di valori a variabili, in questo linguaggio, è cosa molto più labile di quella che avviene, ad esempio, in Basic o C. Il fatto è che l'istanziazione è qualcosa di molto meno fermo della inizializzazione, come possiamo vedere dal seguente esempio:

```
Goal: Variabile="maria".
Variabile=maria
1 solution
Goal : Variabile="Maddalena".
Variabile=Maddalena
Goal:
```

Secondo logica, e secondo i dettami e le regole di qualunque altro linguaggio, al rigo segnato dagli asterischi avremmo dovuto avere una risposta diversa, al massimo «False». Se, cioè, questo tipo di operazione fosse stato eseguito in Basic il sistema avrebbe inteso la riga precedente come una richiesta di valutazione circa la eguaglianza delle variabili «Variabile» e «Maddalena». In Prolog è diverso. La riga 1 istanzia la variabile «Variabile» solo per il tempo destinato ad eseguire il confronto e dare la risposta della riga 2, dopo di che «Variabile» ridiviene non istanziata e disponibile per un nuovo compito; perciò la riga 4 non viene considerata come un confronto, ma come una nuova istanziazione. Come fare, quindi, per eseguire una comparazione sfuggendo al cappio della istanziazione? Occorre adottare una scappatoia, eseguendo una istanziazione congiuntamente ad una assegnazione di regola. Un esempio potrebbe essere:

```
Goal:Variabile1="a"and Variabile2="b" and
Variabile1<Variabile2.
1 Solution
Goal:
```

che, ad un minimo di esame, diviene abbastanza intuitivo, specie se si converte il tutto in linguaggio naturale. Esso, in pratica diviene:

```
Variabile1 ha il valore "a"; Variabile2 ha il
valore "b". Il valore "a" è maggiore del
valore "b"?
```

Il sistema compara i valori delle due variabili nello stesso momento della loro istanziazione, eseguendo la comparazione voluta. Se, dopo l'operazione, si batte alla tastiera:

```
Goal:Variabile1=Variabile2
```

avremo, dal sistema, un messaggio d'errore (error 2201 - Free variable in expression), a causa del tentativo di comparare due variabili ormai non istanziate (Variabile1 e Variabile2 non hanno più alcun valore, proprio nel momento stesso che, col [Return] è stato eseguito lo scopo del precedente Goal).

Ritorniamo un momento alla comparazione delle stringhe; posto nei termini di cui prima, il problema si presenta piuttosto semplice e facile da risolvere (la comparazione bruta di due stringhe non è certo cosa da richiedere intelligenza artificiale). Ma la tecnica di comparazione delle stringhe non è così limitata; la fonte dei termini di comparazione può essere, d'altro canto, diversa (tastiera, memoria di massa, input da periferiche diverse, come scanner, videoterminali, ecc.).

Un esempio di quanto appena detto è rappresentato da quanto esposto nel seguente programmino, che mostra anche alcune tecniche di I/O dati.

Analizziamolo un momento:

```
Predicates
si_no(string).
Clauses
si_no(Risposta) if
    (Risposta="No" or Risposta="Si") and
    print("Negativo").
```

Il listato appena battuto esegue una serie di cosette simpatiche che così possono essere riassunte:

```
Goal:si_no("no").
Negativo
True
Goal:si_no("No").
Negativo
True
Goal:si_no("si").
False
Goal:
```

Semplice capirne la tecnica: eseguire comparazioni di questo tipo determina una azione diretta sul blocco di istruzioni. Prolog sviluppa il suo piano di comparazione basato sull'input (attraverso il Goal) dell'utente o sulla parola «Negativo» quando qualcuno inserisce la risposta appropriata. L'eccellente lavoro finora svolto nella comparazione delle stringhe può essere adattato senza traumi anche alla comparazione tra numeri; per essere più corretti diremo che comparazione, istanziazione, e verifica attraverso Prolog dei valori numerici seguono le stesse regole formali previste nell'organizzazione delle stringhe. Anche qui ci serviremo di un programmino ad hoc:

```
Predicates
maggiore(integer,integer,integer,integer)
Clauses
maggiore(A,B,C,D) if
    A>B and
    A>C and
    A>D.
```

semplice, no? Chi riuscirebbe a fare la stessa cosa in Pascal o in Basic? Ma non basta; la stessa clausola può essere modificata per avere nuove funzioni come ad esempio:

```
Predicates
maggiore(integer,integer,integer,integer)
Clauses
maggiore(A,B,C,D) if
    A>B and
    A>=C and
    A<D.
```

Questo sì che è controllo! Il programma verifica che A sia superiore a B, eguale o maggiore di C e minore di B. Interrogare, in questo come nel caso precedente, è cosa da ragazzi. La domanda:

```
Goal:maggiore(7,5,3,1)
```

darà, nel caso del primo programma «True» e nel caso del secondo «False». Non si tratta di qualcosa di estremamente elastico? Chi può fornire di meglio?

In questa puntata abbiamo esaminato più da vicino alcune tecniche di comparazione, semplici ma abbastanza efficienti. Ci siamo comunque resi conto che abbiamo a questo stato di cose più mezzi che materiale da usare.

Una grossa mancanza, sentita proprio dopo le parole di oggi, è quella di approfondimento più avanzato dell'aritmetica del sistema e delle tecniche operative numeriche. È di questo che ci interesseremo nella prossima puntata. A risentirci! MC