L'Intelligenza Artificiale

di Raffaello De Masi

I linguaggi d'elezione dell'Intelligenza Artificiale: il LISP

Terza parte

Un'altra funzione fondamentale di controllo di flusso di un programma è **[cond]**, l'equivalente in Lisp dello statement If-Then-Else di altri linguaggi. La forma più generale della funzione è:

(cond (*test$_1$ azione*)
(*test$_2$ azione*)
(*test$_3$ azione$_3$*)
.....
(*test$_n$ azione$_n$*)

I test vengono valutati tutti finché uno viene ritenuto vero. Allora le operazioni associate con il test (*azione*) verranno eseguite e il valore della condizione sarà il valore dell'azione eseguita. Una volta, infine, che viene trovato un test che dia condizione vera, ed eseguita la relativa *azione*, nessun'altra valutazione, nell'ambito di [cond] viene eseguita ed il flusso del programma abbandona la funzione. Se, ancora, nessuna delle condizioni è testata come vera, il valore finale di [cond] sarà **nil** (lo conosceremo meglio tra poco).

[cond] è una funzione, in LISP, estremamente versatile: non entreremo qui nei particolari, ma una delle forme per cui essa è utilizzata si basa [illegible] fatto che, al contrario di altri linguaggi, [illegible] per una condizione di falso viene restituita la funzione [nil], per la condizione di vero non restituisce solo una funzione di vero, [t]rue, ma in effetti il risultato del test significa espressamente *diverso da falso*. Può sembrare un particolare di scarsa importanza, ma non lo è.

Sempre a proposito di test condizionali, ne approfittiamo per introdurre due funzioni semplici, [equal] e [null]. Si tratta di due funzioni logiche che possiedono la forma:

(equal *espress.1 espress.2*)

e

(null *espress.1*)

La prima funzione restituisce t(rue) se espress.1 ed espress.2 sono eguali, mentre [null] dà vero se il suo argomento è () o [nil].

Atomi e Liste

Abbiamo già detto, in precedenza, che Lisp è un linguaggio adatto alla elaborazione di simboli. Cominciamo allora ad intenderci sul significato di tali termini. Si intende come simbolo una stringa di caratteri alfanumerici che inizia con una lettera e non ha spazi vuoti all'interno: istintivamente viene qui da pensare alla definizione di nome di variabile, ma non è la stessa cosa.

Lisp, abbiamo già ripetuto fino alla noia, è estremamente interattivo. In tale ottica consideriamo l'espressione:

(setq milano roma)

avremo come risposta un messaggio d'errore, ed il motivo è ovvio: il sistema tenterà di assegnare alla variabile milano il valore della *variabile* roma, non ancora definita. Ma se noi intendiamo assegnare a milano non già un valore numerico di variabile, ma la sequenza di simboli stessa, occorre seguire un'altra strada.

Ci viene in aiuto la funzione [quote] che va usata nel modo:

(quote "serie_di_caratteri_alfanumerici")

dove con serie_di_caratteri_alfanumerici si intende la stringa di caratteri, i simboli che si intendono rappresentare.

Battiamo così

(setq milano (quote roma))

ed avremo assegnato alla variabile milano il simbolo "roma" tout court; a semplice riprova batteremo (ricordate la prima parte del Lisp)

milano

per avere in risposta

roma.

La funzione [quote], e le parentesi connesse, sono state comunque qui usate per puro scopo dimostrativo. Sebbene universalmente riconosciuta ed implementata da tutti i fornitori di linguaggi Lisp, il suo uso è per lo meno seccante e viene generalmente sostituito dalla forma ['], per cui la forma precedente diviene molto più maneggevole presentandosi come

(setq milano 'roma)

Chi legge potrà aver avuto l'impressione che, con nome di simboli, si intenda far riferimento alle stringhe alfanumeriche del Basic: non è così. Infatti, tanto per fare un esempio, fanno parte dei simboli gli stessi nomi delle funzioni intrinseche del Lisp.

Cose come *roma* sono chiamate, in Lisp, atomi; ma atomi possono essere sia numeri che, come abbiamo appena visto, sequenze di caratteri alfanumerici. Semplicisticamente, potremo affermare che si intendono come atomi tutte le cose manipolabili dal linguaggio. Le poche regole che disciplinano l'ortografia degli atomi sono così riassumibili: ogni parola, composta di lettere, può essere usata in Lisp, vale a dire che qualsiasi parola non contenente caratteri estranei all'alfabeto sarà un atomo. Per una serie di ragioni che qui non è il caso di trattare, e che sono intrinseche alla filosofia stessa del linguaggio, non ci sono limiti alla lunghezza delle parole utilizzabili. In aggiunta, comunque, possono essere utilizzati, come avevamo accennato precedentemente, caratteri numerici, con l'unica eccezione rappresentata dal fatto che non possono comparire all'inizio della parola stessa. Il motivo di tale restrizione appare ovvio: il sistema tenta di interpretare quanto legge prima come numero; se l'atomo cominciasse con un numero cardinale, trovando come seconda o successivo carattere una lettera andrebbe in confusione, con risultati imprevedibili. Per cui la maggior parte dei dialetti di Lisp non accettano, come primo carattere di un atomo, un numero (a meno che l'implementatore del linguaggio non abbia previsto un token di sistema che verifichi che, perché una serie alfanumerica sia considerata come tale, debba contenere *almeno* una lettera; si tratta, comunque di una complicazione inutile, anche per rispettare il principio, appena possibile, di essere chiari, il che non guasta mai).

Ancora, una sequenza valida di definizione di un atomo non può includere parentesi (di qualsivoglia tipo, aperte o chiuse). La funzione delle pa-

rentesi è talmente precisa e circostanziata da non ammettere ambiguità, di alcun genere.

Restano i segni speciali, come @, #, $, % od altri. In generale si tratta di caratteri non utilizzabili (ma ancora, qui, la completa mancanza di standard del Lisp consente numerose variazioni); generalmente fa eccezione il simbolo [_] (le parentesi quadre non c'entrano; servono solo a delimitare il carattere); tale carattere è utile nella definizione di stringhe alfanumeriche, quando queste sono rappresentate da più di una parola. Così apparirà molto più chiara la stringa "sequenza_di_variabili", piuttosto che "sequenzadivariabili", senza alcun separatore interno.

La definizione di simboli è, comunque, abbastanza elastica: non si dimentichi che tutti i nomi di funzioni, come [setq] o [sin] sono simboli (e quindi atomi). In effetti ciò porta a considerare che, generalizzando, qualunque cosa (numeri, definizioni, istruzioni) utilizzata in Lisp è un atomo. E con ciò ci stiamo avvicinando all'assunto principale, del Lisp come manipolatore di liste.

Infatti, raggruppando atomi insieme ed inserendoli tra parentesi tonde si ottengono liste di "cose" o più precisamente, liste di atomi. Così

(A1 A2 A3)

è una lista composta da tre elementi (o membri). Ciò ci porta a considerare che tutto quanto utilizzato come esempio finora, come

(times n1 n2)
(times, lo si ricordi, rappresenta la moltiplicazione)

o (sum 44 52 12.5)

sono liste (rispettivamente di 3 e 4 atomi, in quanto si identifica come membro anche l'operatore numerico). Da ciò discendono molte cose, prima tra tutte il fatto che una lista può comprendere altre liste come

(sum (−5 3) (times 5 5))

in cui, attenzione, si ha una lista di *tre* elementi di cui il secondo ed il terzo sono ambedue liste di due elementi. In tal caso, in una accezione più globale, si fa uso del termine di S-espressione, che, formalmente, può essere definita come l'insieme di simboli ed atomi, ordinati da una serie di parentesi aperte e chiuse, in equilibrio tra di loro. Un esempio particolare di lista è quella vuota, il **nil**, o (): acutamente, come fanno osservare Charniac e MacDermott nel loro volume "Artificial Intelligence Programming", Eribaum, Hillsdale, N.J., 1980, (è alla bibliografia di tali autori, che verrà, alla fine, fornita esaurientemente, che fanno riferimento la maggior parte delle notizie espresse su questo argomento nei nostri articoli), () e (()) non sono, comunque la stessa cosa; mentre nel primo caso si ha una lista vuota, nel secondo si ha una lista di un solo elemento che, incidentalmente è vuoto. Così, ampliando l'esempio, ((())) è del tutto diverso dai precedenti.

È possibile esplorare una lista mediante due ordini: [car] e [cdr]. Nel primo caso viene restituito il primo elemento della lista, nel secondo la rimanente parte. Ancora, le due funzioni non possono essere applicate ad atomi, solo a liste. [car] e [cdr] possono essere, in espressioni dove sono presenti in gran numero e successivamente, abbreviate e riunite tra di loro, utilizzando le lettere rappresentative [a] e [d]. Così

(car(cdr(car(cdr lista))))

può essere abbreviata in

(cadadr lista)

dove i termini comuni "c" ed "r" sono presenti all'inizio ed alla fine e quelli rappresentanti le singole funzioni sono espressi solo dalla lettera significativa, "d" e "a".

Ma stiamo perdendo di vista l'obiettivo principale del nostro discorso, che è quello di inquadrare anche se in maniera non univoca e non del tutto corretta le liste e le S.espressioni. Il discorso è tutto basato sulla importanza delle parentesi tonde, che, per semplificare, possono essere intese con il valore di START e STOP. Ciò che appare tra le parentesi è, generalmente, il nome di un operatore, un'azione che si desidera che Lisp svolga per noi, ed una serie di valori, argomenti che l'operatore stesso dovrà utilizzare nel suo sviluppo. Il set d'istruzioni prende il nome di funzione. Così

(setq a '(b c d e))

assegnerà alla variabile a la lista (b c d e). Tutta la riga è formata da tre S.espressioni (abbiamo già accennato che significa espressione simbolica): [setq] (che ricordiamo può essere anche scritto come set"), a, e (b c d e). Le prime sono due. S.espressioni formate di un solo atomo, la terza è un esempio di S.espressione di lunghezza maggiore. A questo punto se battiamo:

a

il sistema risponderà

(b c d e).

Il valore "a" può essere ridefinito con una espressione analoga; tanto per intenderci

(setq a 'arrivederci)

setterà la variabile "a" al nuovo valore simbolico (e non numerico, per la presenza della quota ['']). Per inizializzare la variabile è possibile battere:

(setq a nil) o
(setq a ())

per ripulire il contenuto della variabile. Attenzione, però, non si tratta di cancellazione della variabile stessa. La variabile continuerà ad esistere fino al resettaggio del sistema e la variabile "a" conterrà sempre un valore, che, solo per inciso, è nullo . "a", cioè, ha un valore ben diverso da un'altra variabile mai inizializzata ed adoperata. Infatti battendo

zuzzerellone

il sistema, generalmente, risponderà, oltre che restituendo la parola "zuzzerellone" come abbiamo visto due puntate fa, con qualche messaggio d'errore, evidenziante la non definizione della variabile stessa. Qualche piccola precisazione per quanto attiene alle

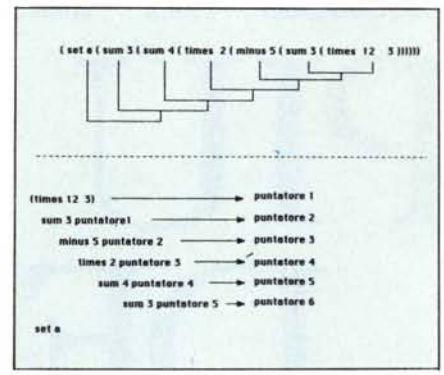

*Figura a) - Gerarchia di valutazione della S.espressione 3+(4+(2 * (5− 3+(12*3))))*

parentesi: Lisp esegue sempre una valutazione di ciò che gli viene fornito e tenta, in ogni momento di eseguire quello che gli viene presentato, tranne in casi particolari, in cui accetta supinamente quanto gli viene fornito (è il caso di quanto è indicato dalle virgolette ['']). L'ordine di analisi è determinato dalla gerarchia delle parentesi stesse. Perciò l'espressione algebrica

3 + (4 + (2 • (5 − 3 + (12•3))))

assume un aspetto piuttosto simile in Lisp, a parte la particolare notazione prefissa. La sequenza di valutazione parte dalla parentesi più interna per giungere a quella più esterna, questo sebbene il sistema legga una sola volta la riga, da sinistra a destra. Nel caso, poi, che il grado di gerarchia sia pari, e che quindi gli operatori siano allo stesso livello le operazioni seguono la normale sequenza algebrica, vale a dire da sinistra a destra, a meno di specifiche precedenze. La figura a) mostra come viene articolata l'analisi dell'espressione citata.

La precisazione, forse pedante, è importante per vedere come vengono costruite le funzioni non primarie (o come si dice in gergo built-in, precostruite); è questo uno degli argomenti più interessanti e delle caratteristiche più utili e pratiche del Lisp; e, ancora, è fondamentale per l'inquadramento del concetto generale di lista che qui abbiamo solo intravisto. Ne riparleremo! MC

É Honeywell PC Superteam

Da oggi tutti i Personal dovranno fare i conti con PC Superteam. Disponibile in tre versioni per adeguarsi nel modo più flessibile alle vostre esigenze, PC Superteam opera con disco, diskette e cartuccia nastro; ha una memoria di massa che si estende fino a 40 milioni di bytes; per lui sono già pronti oltre 10.000 programmi di software standard. Il complesso delle sue caratteristiche tecniche, fra le quali spicca l'eccezionale velocità, rende PC Superteam davvero straordinario.

Un Personal che è già entrato nella leggenda può entrare nel vostro ufficio.

IL PERSONAL PIÚ VELOCE DEL WEST

RSCG

Conoscere e risolvere insieme.

Honeywell

Honeywell Information Systems Italia