di Raffaello De Masi

I linguaggi d'elezione dell'intelligenza artificiale: il LISP

SESTA E ULTIMA PARTE

La funzione [for]

Poiché la lista è la funzione fondamentale del Lisp, ed è in essa che confluiscono e si raggruppano gli elementi fondamentali del linguaggio, è necessario avere a disposizione un operatore che esegua in maniera del tutto automatica, su tutti gli elementi della lista stessa, qualcosa, ad esempio una operazione stabilita dall'utente. La serie di operazioni raccolte sotto tale tipologia è riunita sotto il nome di funzioni di mappatura. Queste operazioni sono piuttosto complesse, e richiedono conoscenze ben più avanzate di quelle finora raccolte. Ciononostante è possibile qui introdurre una funzione, [for], che, sebbene non standardizzata nella forma e nelle sue potenzialità, risulta di notevole utilità per la manipolazione di elementi multipli, come quelli presenti in una lista.

[For] è un'altra delle forme del Lisp prive di qualsiasi standard; in analogia con quanto evidenziato da Chamiek e McDermott nel loro volume «Artificial Intelligence» che, come abbiamo più volte detto, è fonte prima di quanto stiamo narrando in queste note, mostreremo le tre principali forme di [for]. Esse possono così essere schematicamente in figura 1.

Al contrario di quanto avviene nella maggior parte dei linguaggi di programmazione, la variabile viene sempre riferita agli elementi della lista successiva all'operatore [in]. Non ha perciò senso il comune operatore [step] presente, ad esempio, in Basic. Per ogni elemento della lista, «espressione» viene valutata. La differenza tra le tre forme sta in ciò che accade al valore di «espressione». Nel primo caso quanto presente in [espressione] viene brutalmente eseguito, senza influenzare il contenuto delle liste e variabili iniziali; nel secondo, [save], i valori vengono inseriti e conservati nella lista stessa, nel terzo, infine, essi sono collegati insieme nella lista. Ad esempio, immaginiamo di voler manipolare una lista di numeri, derivandone una seconda lista che sia eguale alla prima tranne che ogni membro è pari al corrispondente +2. Avremo una serie di istruzioni come in figura 2 a). Da [splice] si interrompa allorché, in una delle liste manipolate, viene intercettato l'ultimo atomo.

Ritorniamo un momento alla funzione [do] per meglio esemplificarla; in pratica è questo il caso in cui sia [for] che [do] risultano molto più somiglianti alle analoghe strutture di linguaggi più evoluti. Si tratta di un loop nella più semplice accezione della parola, universalmente usato per eseguire cicli ripetuti un certo numero di volte; ci sembra inutile continuare su questo argomento.

La ricursione in Lisp

Una delle tecniche più efficaci nella programmazione è la ricursione (o ricorsività). Senza voler esaminare a fondo tale importante aspetto della più avanzata tecnica programmatoria diremo, per sbrigarcela in due parole, che una funzione è ricorsiva se essa si definisce in termini di se stessa. Per fare un esempio banale potremo dire che un esempio di definizione ricorsiva è quella che indica la pioggia come «quella cosa che cade quando c'è pioggia».

Posta in questi termini la cosa sembra priva di significato (ed infatti lo è), ma in pratica si tratta di una delle strutture più potenti di certi linguaggi informatici. Lisp supporta la ricursione nella maniera più ampia e completa; una funzione può richiamare se stessa sia in forma diretta (la funzione [a] chiama la funzione [a], nella sua definizione e sviluppo) che indiretta ([a] chiama [b] che a sua volta richiama [a]). A titolo di pura notizia, notevole potenza della funzione di riscursione è data, al Lisp, dal possedere operatori come [car] e [cdr], di cui abbiamo già parlato. Strutture ricorsive che utilizzano tali funzioni, in una con [defun], sono pane quotidiano del Lisp, specie se riunite nelle «macro», di cui parleremo adesso.

```
1]   ( for ("variabile" in list)
       (do "espressione))

2]   ( for ("variabile" in list)
       (save "espressione"))

3]   ( for ("variabile" in list)
       (splice "espressione))
```

Figura 1 - Tre tipologie d'uso della funzione [for].

Le «macro»

Si definisce come «macro» la possibilità di riunire sotto un unico operatore-funzione una serie di operazioni-manipolazioni predefinite, che, per il fatto di essere correntemente utilizzate nel corso di un programma, può essere agevole riunire e richiamare con un unico nome. Possono essere intese in senso lato come macro le procedure del Pascal, ma anche le funzioni del «C» {————}, le word del Forth, i subprogrammi del Fortran, ecc. Il Basic originariamente ne è stato privo (quale difetto non ha questo povero linguaggio!) ma le sue variazioni più

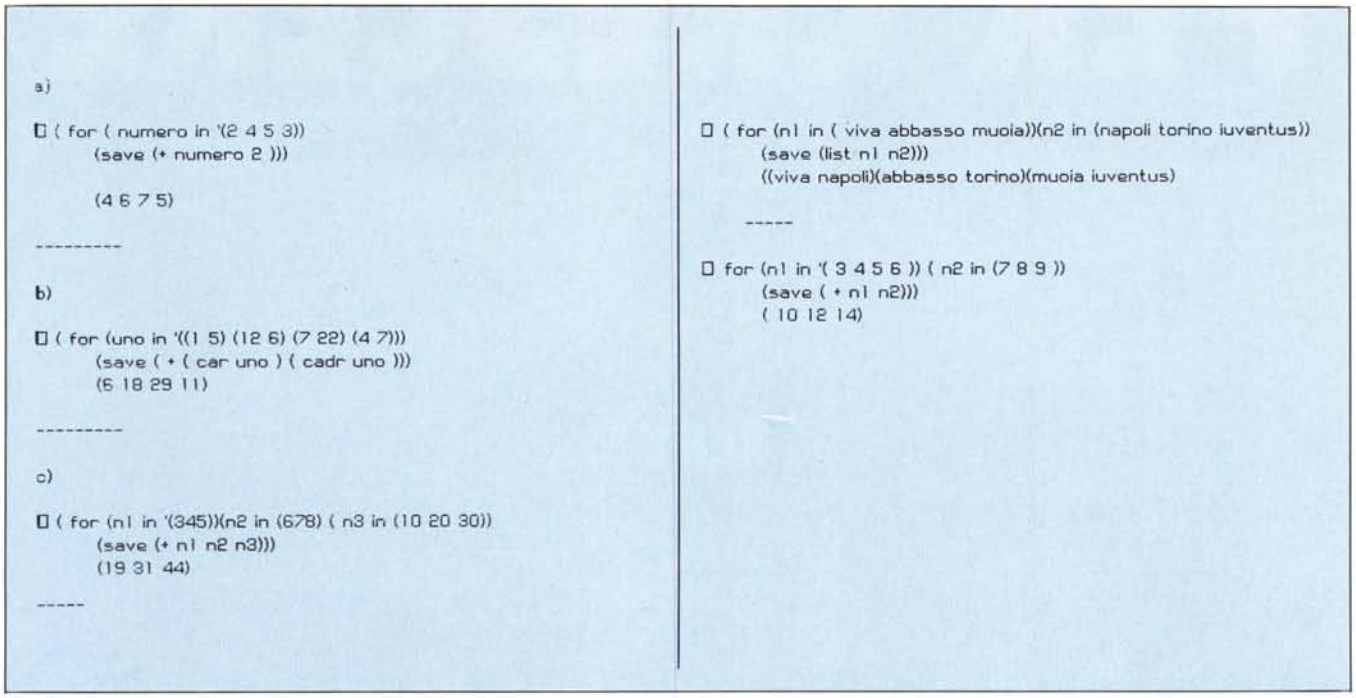


Figura 2 - Esempi della funzione [for] su casi numerici e di liste (nota: indovinate di che squadra è De Masi... e, soprattutto, di che squadra NON è...).

evolute la ammettono, sia sotto forma di semplici [DEFFN] (define function, che debolucce all'inizio, sono divenute sempre più elastiche ed efficienti), sia sotto le vesti di veri e propri sottoprogrammi-funzioni, come nella più recente release del MSBasic della Microsoft per il Mac.

Lisp consente la creazione di macroistruzioni attraverso l'operatore [defmacro] (si ricordi che operatore e funzione, in Lisp, assumono lo stesso significato). Tanto per fare un esempio, banale quanto si vuole, definiamo una macro, [addizione] che equivalga all'operatore [+]; avremo:

```
(defmacro addizione (x1 x2)
(+ x1 x2))
(addizione)
```

basterà a questo punto battere

```
(addizione 35)
```

per avere il risultato di 7.

Come numerosi altri linguaggi, anche il Lisp consente che macro possono confluire in altre, per così dire di gerarchia maggiore. Si tratta di strutture piuttosto potenti ed elastiche, facili da maneggiare, specie se ben commentate. Linguaggi ben fatti consentono inoltre di creare macro in cui risulta variabile addirittura il numero di variabili da manipolare. È così possibile creare veri e propri nuovi statement, come con le word del Forth; in ciò è di notevole aiuto la presenza delle liste che, con la loro struttura, risultano ideali per generalizzare la funzionalità di una macro; è infatti possibile far riferimento generico ad una lista, ancorché indefinita, pur non conoscendone i limiti, cosa che non è sempre possibile in altri linguaggi, dove le array, lontane cugine delle liste stesse, sono di esse molto meno manipolabili e parecchio più rigide.

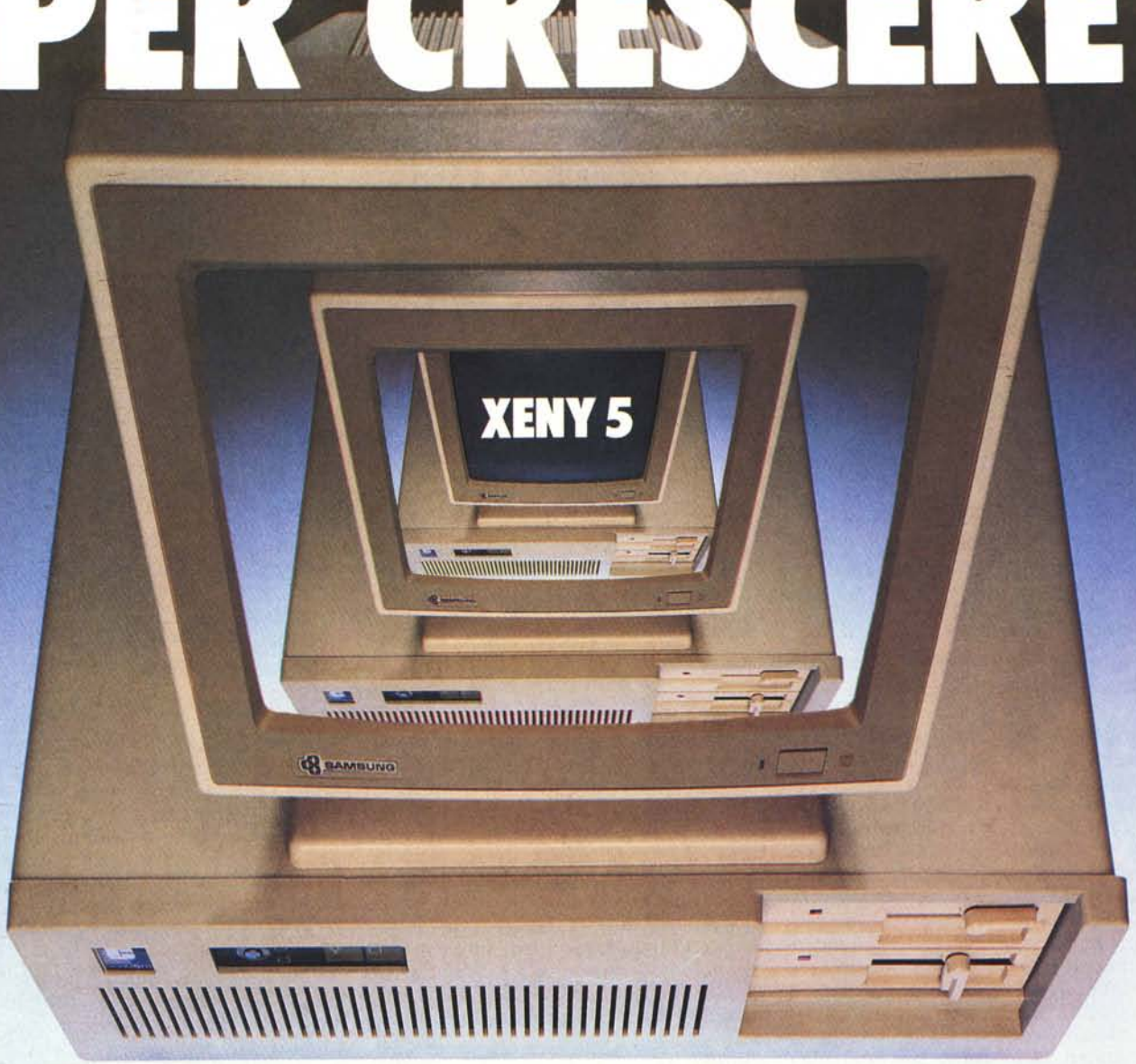
Termina così la nostra rapida occhiata al linguaggio Lisp. Sebbene sviluppata in diverse puntate, la trattazione ha toccato solo i principi fondamentali della programmazione in tal idioma, ingiustamente relegato nel campo pur affascinante della intelligenza artificiale. Si tratta, come speriamo di aver mostrato, di un idioma abbastanza intuitivo e semplice da usare; il suo problema, se ci è consentito dirlo, è nella sua anticonvenzionalità molto spinta, che lo rende strano a chi è abituato a pensare in linguaggi molto più conseguenziali, con costrutti soggetto — predicato — complemento abbastanza codificati, come avviene in Fortran, Basic, Pascal. Il suo grave difetto, la mancanza di uno standard, è molto pesante e lo rende scarsamente trasportabile; peccato, perché si tratta di un linguaggio discretamente elastico, tanto da poter essere usato con disinvoltura ed efficacia sia in applicazioni numeriche che non.

La sua estrema interlocutorietà ne fa uno strumento ideale per lo sviluppo di software (quante volte, scrivendo una serie di righe in Basic od una procedura in Pascal, ci siamo chiesti se davvero funziona correttamente?), ma ad essa occorre fare la mano, altrimenti si resta per lo meno sconcertati da certe risposte talvolta ambigue. Tutto sommato, comunque, risulta, per chi lo volesse utilizzare, piuttosto facile da imparare, non imponendo, ad un attento esame, alcuna particolare modifica di regole programmatorie già preacquisite.

Che sia un linguaggio d'elezione per la I.A., come dice il nostro titolo, lo dimostra il fatto che il suo più giovane concorrente, il Prolog (di cui comunque parleremo) stenta a prendere quota anche in certi ambienti più qualificati, segno che, il rapporto pregi-difetti è nettamente in attivo. Noi ne abbiamo trattato solo alcuni aspetti, utili per quanto diremo appresso; per chi volesse approfondire l'argomento forniremo, al termine di queste note, una bibliografia (eventualmente annotata) più che sufficiente anche per un non novizio.

Esaminato l'argomento Lisp che, come abbiamo già detto, ci ha fornito alcuni tool per proseguire il nostro discorrere sui veri aspetti dell'intelligenza artificiale, affronteremo la prossima volta un argomento oltremodo affascinante; la visione ed i problemi ad essa connessi. Vedremo, è il caso di dirlo, come un computer, dotato di opportune periferiche, possa guardare, e distinguere oggetti. A risentirci, quindi! MC

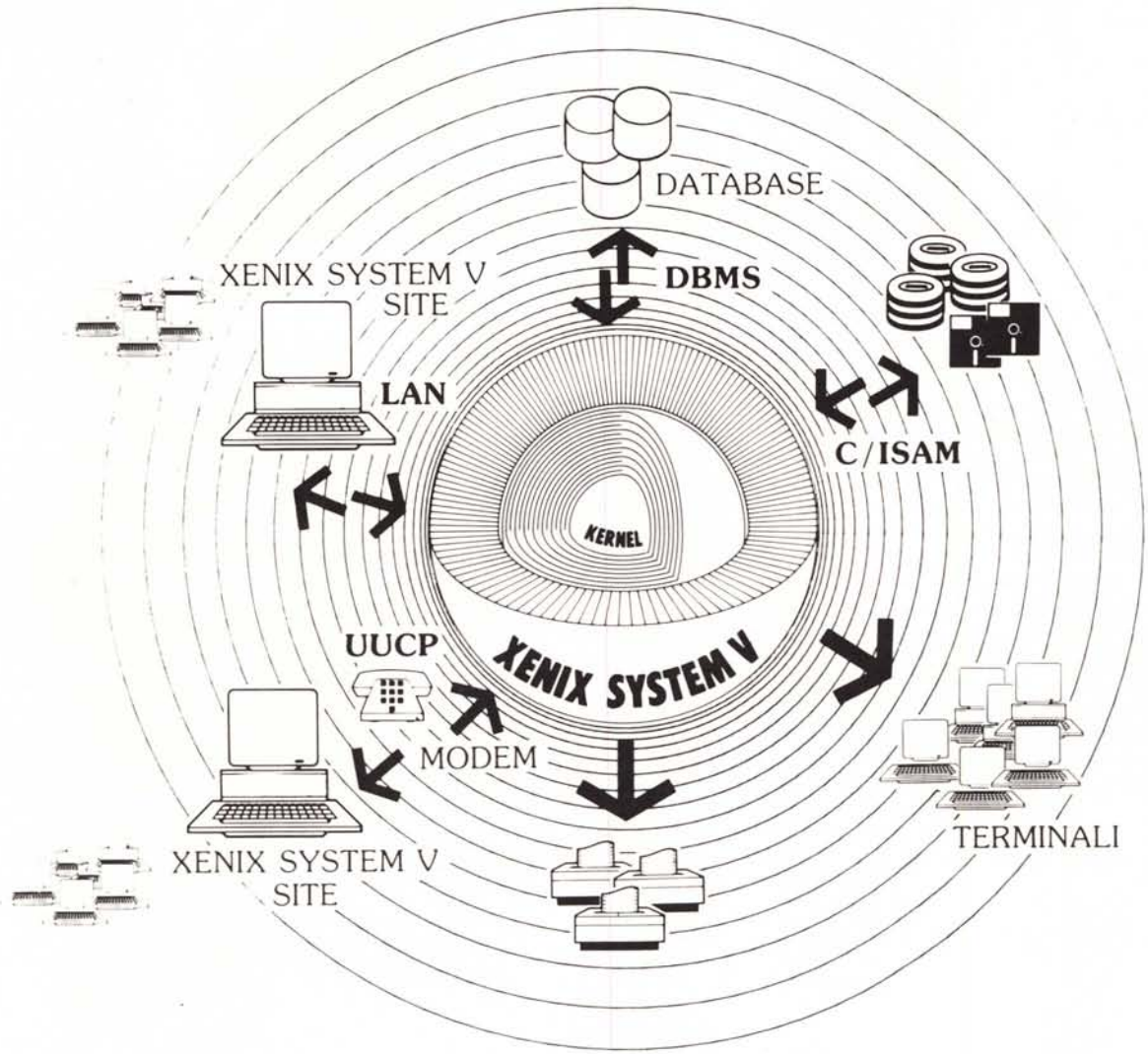
PER CRESCERE



COMPUTERLINE

KALEIDO/Roma

XENY 5



XENY5 vi porta nel mondo dei sistemi multiutente, multitasking, nelle reti per Office Automation.
XENY5 è un sistema integrato di hardware e software: un hardware PC AT compatibile, un software XENIX Sys V S.C.O. nel pieno rispetto della «System V Interface Definition» AT&T.
XENY5 vi dà la possibilità di leggere e scrivere floppy disk da 360 Kbyte o 1.2 Mbyte anche in formato MS-DOS per consentire scambio dati in maniera trasparente. XENY5 permette il collegamento di almeno 8 posti di lavoro indipendenti, la connessione con altri sistemi XENY5 attraverso la rete locale MICNET, il collegamento con altri computer in ambiente XENIX/UNIX o in altri ambienti operativi, tutto con il software standard XENIX Sys. V. XENY5 viene fornito con il software e documentazione relativa a partire dalla configurazione base.

COMPUTERLINE

COMPUTERLINE srl - 00173 Roma - Via U. Comandini, 49 - tel. 06/6133025-6130164-6130912 - Telex 621166 Compul-I - Telefax 39 6 7970559