L'Intelligenza Artificiale

di Raffaello De Masi

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

Nello studio dell'intelligenza artificiale, il motivo dell'uso del Lisp è stato riassunto, con una battuta, da Eugene Charniak, nel suo articolo «On the use of framed knowledge in language comprehension» Artificial Intelligence, 11, pp. 225/265; si impara il Lisp per lo stesso motivo per cui si impara il francese prima di andare in Francia: è il linguaggio dei nativi. Per la verità, programmi occasionali sono stati talvolta scritti in linguaggi diversi, ma si può dire che, su cento programmi, 95 sono redatti in Lisp, e dei rimanenti 5 almeno quattro sono stati scritti, in altri linguaggi, prima che il Lisp fosse stato messo a punto: infine l'ultimo è probabilmente scritto in Prolog, un avversario recentemente comparso sulla scena degli idiomi intelligenti.

Ma la situazione non si risolve rispondendo in tal modo: stabilito che il Lisp è il best seller dei cultori di A.I., perchè viene così unanimamente preferito ad altri linguaggi, come il Fortran od il «C»? La risposta non è semplice, anzi, per meglio dire, è piuttosto sottile, in quanto alla scelta del Lisp concorrono diverse ragioni: per semplificare il problema diremo che i principali motivi della scelta di tale lingua sono riducibili a due.

Per prima cosa, il Lisp è molto più flessibile di altri linguaggi: l'utilizzatore ha un controllo pressochè totale sull'idioma, fino al punto che è possibile modificare la stessa sintassi del linguaggio, se questa non è di nostro gradimento.

La seconda ragione è che il Lisp è orientato alla manipolazione di simboli, invece che di numeri. Lisp consente numerose facility destinate ad associare informazioni e simboli: inoltre ha numerosi mezzi per costruire nuove strutture di dati, secondo i nostri desideri e direttive; le stesse strutture, una volta preparate, non sono rigidamente ancorate al programma, ma possono essere messe agevolmente da parte per destinare il loro spazio d'uso a qualcosa di diverso. Il tutto senza che il programmatore sia chiamato a sapere granché su quello che il computer, nel suo interno, compie (ciò forse in contrasto con la nouvelle vague della programmazione, che sta imponendo nuove e farraginose regole all'ars programmandi, uno fra tutti il rugginoso ed impastoiato Pascal e suo degno nipote Modula-2).

Il rovescio della medaglia, qui come altrove, esiste, ed è pesante. Il Lisp è forse il linguaggio meno documentato e standardizzato: vale a dire che non esiste un atto od una commissione ufficiale (od ufficiosa) che abbia stabilito uno standard di base cui potersi rifare; tanto per intenderci Lisp è meno standardizzato dello stesso Basic, il che è quanto dire. Per cui è possibile creare statement od informazioni (nel Lisp hanno diverso nome), del tutto incomprensibili da un'altra forma dialettale. Il rimedio sta nella stessa natura del nostro linguaggio: esso è estremamente interattivo; perciò è sempre possibile, in qualunque momento, testare la routine desiderata per vedere se funziona sulla nostra macchina: la natura stessa del Lisp, estremamente aperto al dialogo, consente questo ed altro.

Il Lisp è un linguaggio interpretato: tecnicamente ciò vuol dire che il programma, invece di essere tutto traslato in linguaggio macchina prima di essere eseguito, viene guardato linea per linea, interpretato per quello che gli viene richiesto, e eseguito ancora così, pezzo per pezzo (esistono sul mercato compilatori Lisp, ma si tratta di operazioni avanzate, e, per nostra opinione, snaturanti il carattere stesso del linguaggio).

Secondo il punto di vista di molti studiosi, inoltre, il Lisp non è solo un linguaggio, ma un vero e proprio ambiente, in questo essendo molto simile all'APL. Il legame tra linguaggio ed ambiente Lisp è così stretto, infatti, che, in gergo, molto spesso si usa la frase «entrare in Lisp».

Per penetrare in ambiente, appunto, generalmente è sufficiente battere alla tastiera:

```
lisp [enter]
```

dove l'[enter] rappresenta il tasto del ritorno carrello, operazione che da questo momento verrà sottintesa. Lo schermo apparirà pulito, ospitante solo il prompt, che può essere divero a seconda della marca: esso può essere rappresentato da un triangolino, da una lineetta lampeggiante, da un punto esclamativo: per noi sarà un rettangolino, come

```
□
```

Una volta entrati in ambiente, dicevamo, Lisp, per sua natura interprete fino all'assurdo, tenterà di valutare qualunque cosa si batterà alla tastiera, darà la sua brava risposta, ed attenderà di valutare qualcos'altro. In gergo ciò viene chiamato ciclo scrittura-valutazione-stampa.

La prima e l'ultima delle fasi sono ovvie: vediamo cosa succede nella seconda.

Per fare ciò partiamo con un esempio abbastanza semplice: facciamo valutare alla macchina un numero: se battiamo alla tastiera:

```
□ 5
```

avremo come risposta

```
5
□
```

vale a dire che la macchina ha analizzato la cifra, ne ha dedotto il suo valore, lo ha restituito ed ha visualizzato un nuovo prompt, attendendo di valutare qualcos'altro.

La prima sessione di scrittura di Lisp potrebbe, alla nostra macchina, essere:

```
lisp; si entra in ambiente
□ 5
5; viene chiesto a Lisp
□ 22; di valutare una serie di
22; numeri
□-55
55
□ 2.77
2.77
□ (exit)
```

Vediamo che la macchina non ha poi svolto un gran lavoro e per ora le performance matematiche del linguaggio non sono poi state brillanti. È invece ovvio il senso della valutazione di cui dicevamo precedentemente, ed in più compaiono due istruzioni nuove: intuiamo che [;] rappresenta il simbolo di un commento: tutto ciò che lo segue, fino al CR viene ignorato. Compare, alla fine il comando (exit), ben racchiuso tra parentesi tonde, che rapprensenta il comando per uscire dall'ambiente.

Tutto ciò non è stato finora eccitante: vediamo perciò come è possibile eseguire operazioni (anzi, per dirla alla Lisp, funzioni) aritmetiche: battiamo

```
□ (+ 5 2)
```

avremo

```
7
□
```

viene cioè eseguita l'addizione delle cifre fornite.

In generale, la sintassi della istruzione + è la seguente:

```
(+ -numeri-)
```

dove *-numeri-* (i trattini non significano il segno meno), rappresentano gli argomenti dell'operazione e possono essere più di due: così

```
(+ 12 3 4 12)
```

darà

```
31
□
```

la spaziatura tra i caratteri ed i simboli non è rigida. Possono essere inseriti più spazi bianchi ed anche un'intera riga sia tra gli argomenti che tra argomento ed operatore, ed ancora tra parentesi ed altro simbolo; il linguaggio non fa differenza. Ciononostante, per quel minimo di standard che il linguaggio non ha, ma che le abitudini gli hanno imposto (a proposito, stiamo lavorando in Franz Lisp, uno dei comuni dialetti del Lisp), è d'uso seguire la notazione appena vista, con un solo spazio tra operatore ed operandi e tra gli operandi stessi, mentre le parentesi non possiedono alcuno spazio di separazione con i simboli che le seguono o le precedono.

Altre funzioni aritmetiche, in Lisp, sono

```
(* -numeri-); operazione di moltiplicazione
(/ -numeri-); divisione
(— -numeri-); ed ovviamente sottrazione.
```

C'è da fare qualche precisazione: la prima impressione della mancanza di coordinamento tra vari Lisp l'abbiamo qui; addirittura manca uno standard per le stesse operazioni aritmetiche: molti Lisp adottano invece del simbolo [+] la notazione [plus], scritta proprio così: un esempio potrebbe essere:

```
(plus 4 5)
```

Tale è la confusione (peraltro più apparente che reale) e la poca portabilità del linguaggio. Generalmente, almeno in casi del genere, comunque, ambedue le versioni (+, plus, — minus, ecc., vale a dire la funzione in simbolo e scritta in lettere) sono ammesse da diversi linguaggi.

La tecnica di valutazione di una funzione, da parte del Lisp, è la seguente: innanzitutto esso valuta gli argomenti, quindi esegue la funzione. In tal modo è possibile inserire due funzioni l'una nell'altra: così avremo:

```
□ (/(+ 81 12) 3)
```

che darà come risposta

```
31
```

vale a dire che Lisp, leggendo la prima parentesi aperta, si aspetta che si stia per richiedere la valutazione di una funzione: trovatala [/] interpreta tutto ciò che gli perverrà successivamente come argomenti della funzione fino a trovare una parentesi tonda chiusa destinata al bilanciamento; incappa invece in una seconda parentesi aperta: mette così in disparte la prima e valuta gli argomenti successivi: esegue l'addizione e recupera la precedente funzione; si aspetta a questo punto un secondo operando (il primo l'ha ricavato dalla coppia interna di parentesi), lo trova, esegue la divisione, cerca la parentesi in chiusura; se, come in questo caso, la ritrova fornisce il risultato (la valutazione finale) altrimenti riprende l'operazione: non esiste limite teorico all'apertura ed alla chiusura di parentesi, tranne, forse, il punto di vacillamento mentale.

Prima di fermarci immaginiamo di aver battuto:

```
□ (* 2 (_ 15 8))
```

e ci accorgiamo immediatamente, dopo aver premuto [return], di aver scambiato [—] con [_]; nessun problema: Lisp, interattivo qual è, e valutatore al sommo grado, ha capito che c'è qualcosa che non va (_ non è una funzione definita): ci risponde:

```
Error: eval: undefined function _
<1>
```

Noteremo che il cursore [□] è cambiato in <1>: senza andare daccapo battiamo:

```
(return _)
```

avremo il risultato esatto

```
14
□
```

con il cursore ritornato alla forma abituale. Lisp, all'errore di valutazione entra in un «break loop», evidenziato dalla nuova forma del cursore, se si sbaglia ancora si passa in un altro «break loop» di secondo grado, rappresentato dal cursore <2>.

Per uscire da un «break loop» possono farsi varie cose: battendo «exit» si esce dall'ambiente stesso: è questo un caso davvero grave, quando altri mezzi non ci permettono di disincagliarci. «Reset» sblocca la situazione fermando l'operazione di valutazione e ritornando alla primitiva condizione «read-eval-print», lo status principale di valutazione (in altre parole, la operazione richiesta viene annullata e si ritorna alle condizioni di partenza: è come se si dicesse all'interprete: «Lascia perdere»). Infine «return» seguito da un argomento, dice al programma «Seguita, sostituendo al carattere incomprensibile quello che ti fornisco di seguito». Si noti la presenza del simbolo ['], il sigle quotation mark.; è importante, e deve esserci: ne vedremo prossimamente il perché.

Esiste, infine un rozzo comando di break, che varia a seconda delle tastiere, che ferma un programma, per esempio quando capita in un loop infinito. È opportuno, per definirlo, leggere le istruzioni della macchina (generalmente è rappresentato dalla combinazione ctrl-c).

Bene, fermiamoci qui; vedremo la prossima volta qualcosa di più interessante, anche per quanto riguarda la vera e propria programmazione. MC