di Raffaello De Masi

# Ancora sulle liste; come e dove si utilizzano

*Parlare di liste significa, a tutti gli effetti, parlare di basi di conoscenza; le due cose sono indiscutibilmente connesse, non fosse altro per il fatto che sono praticamente nate insieme. D'altro canto sarebbe difficile immaginare un linguaggio di Intelligenza Artificiale senza il supporto di queste strutture (si immagini ad esempio un sistema esperto che manipola la sua base di dati con le strutture del Pascal o del Cobol).*
*Fino alla volta scorsa abbiamo parlato, in diverse riprese, delle tecniche di definizione delle liste e delle proprietà delle stesse. In questa puntata è giunto il momento di usare le liste; vediamo come e dove si può far uso della loro potenza e versatilità*

## L'uso delle liste

Usare le liste significa innanzitutto usare elementi delle strutture; daremo quindi di seguito, attraverso un esempio, una dimostrazione della potenza delle liste in caso di un utilizzo dedicato delle stesse. Si tratta di una applicazione piuttosto semplice (tanto, mai come in questo caso, vista una, viste tutte), ma che ci permetterà di affrontare a cuor leggero le successive applicazioni del linguaggio a più ampie e articolate basi di conoscenza.

Immaginiamo di dover manipolare una base di conoscenza contenente i dati di una corsa, poniamo di Formula 1. Finora, nel campionato, sono state svolte cinque corse, con una serie di risultati, ed è nostro intendimento manipolare le informazioni in nostro possesso, per ricavarne alcune notizie di base. La nostra base di conoscenza è stata così organizzata:

```
Domains
piloti = symbol
Predicates
corsa(corridori)
Clauses
corsa(["1",prost,senna,alboreto,nannini,fabi,
                              nakajima,piquet]).
corsa(["2",senna,alboreto,nannini,prost,
                                  fabi,piquet).
corsa(["3",prost,alboreto,piquet,fabi,
                              nakajima,nannini).
corsa(["4",prost,alboreto,senna,fabi,
                              nannini,piquet]).
corsa(["5",nannini,prost,alboreto,
                              senna,piquet]).
```

Il primo membro di ogni lista è racchiuso tra parentesi, come certamente si ricorderà, per forzare Prolog a riconoscere il termine come stringa e non come numero. Le altre stringhe, ovviamente, sono accettabili senza le virgolette. Le corse successive saranno, ovviamente, aggiunte, sotto forma di altre liste, alla base di conoscenza come di strutture «corsa([ecc...», come quelle che si vedono nell'esempio.

Con queste informazioni già presenti nel data base, è possibile imporre goal del tipo:

```
Goal:corsa(["4",X|T])
X=prost
1 solution
Goal:corsa(["5",Vincitore|T])
Vincitore=Nannini
1 solution
Goal:corsa(["4",_,_,Terzo|T])
Terzo=senna
1 solution
Goal:
```

Semplice, ma anche poco pratico; d'altro canto non c'è mica bisogno di una base di dati e di un linguaggio di Intelligenza Artificiale se poi siamo costretti a battere queste cose complicatissime alla tastiera; esiste però un sistema più semplice e pratico per organizzare la nostra ricerca; immaginiamo di voler cercare tutti i vincitori delle corse finora disputate e tutti i corridori classificatisi fino al terzo posto. Definiremo una serie di regole, adatte allo scopo; la prima la chiameremo, ad esempio, «vincitore» e sarà definita come segue:

```
vincitore(X) if
    corsa([Numero,X|T]) and
    print([Numero]).
```

È già possibile, a questo punto, sapere se, ad esempio, «nannini» ha mai vinto una corsa; basterà battere:

```
Goal:vincitore(nannini).
4
1 solution
Goal:
```

o chiedere una lista di tutti i vincitori:

```
Goal:Vincitore(Chi).
1
Chi=prost
2
Chi=senna
3
Chi=prost
4
Chi=prost
5
Chi=nannini
5 Solution
Goal:
```

Per soddisfare alla seconda richiesta, è possibile «costruire» una successiva regola, che, per analogia, chiameremo «primi_classificati», che tradotta in codice sarà:

```
primi_classificati(X)                    if
    (corsa([N,X|T])                      or
    (corsa([N,_,X|T])                    or
    (corsa([N,_,_,X|T])                  and
    print(N,«»,X)
```

Vediamo l'enunciazione di qualche richiesta, basata su questa regola nuova di zecca:

```
Goal:primo_classificato(prost)
1 prost
3 prost
4 prost
5 prost
4 Solutions
Goal:
```

Si tratta ancora una volta di un esempio semplice, ma che evidenzia una tecnica sofisticata degli elementi delle basi di conoscenza. La cosa di maggiore utilità è rappresentata, come si vede, dalla struttura [H|T] che può essere costruita nella maniera più elastica possibile per la ricerca dell'elemento nella base di dati. Questo è importante anche perché, come si vede dall'esempio precedente, il buon prost (non me ne voglia se lo scrivo con la minuscola, lo esige Prolog), viene cercato e compare nelle soluzioni anche se compare in secondo e terzo (e anche più, ovviamente) posto.

Un'altra cosa interessante illustrata da quest'ultimo se pur breve esempio è costituita dal fatto che è possibile esplorare la base di dati attraverso adeguate combinazioni di operatori [or]. Ciò permette, attraverso una oculata analisi dei risultati, di eseguire, ad esempio, ricerche mirate nell'ambito della base di dati.

Infatti, l'operazione di ricerca viene eseguita in una maniera un poco particolare; prost viene cercato nelle liste 1, 3 e 4 come vincitore, poi la ricerca ricomincia daccapo per passare al secondo posto, e così via. Le liste vengono, cioè, scandite *tutte nei primi posti, poi nei secondi e infine nei terzi.* Per maggior chiarezza esaminiamo quanto segue:

```
Goal:primo_classificato(senna)
2 senna
1 senna
3 senna
3 Solutions
Goal:
```

vale a dire che prima viene cercato senna come vincitore (corsa 2), poi al secondo posto (corsa 1) e infine al terzo posto (corsa 3). Niente male davvero!

## *Le liste nelle strutture*

Come abbiamo già visto precedentemente, le strutture, in Prolog, sono tool potenti nella costruzione e nell'organizzazione delle basi di conoscenza. All'interno di una struttura, come d'altro canto ricorderemo, è possibile rappresentare un elemento come una lista; è lecito senz'altro manipolare questa lista come una lista separata, applicando quanto descritto precedentemente, nei nostri esempi.

Immaginiamo di creare una struttura, costituita da una base di conoscenza, in cui sia possibile immagazzinare le preferenze di un gruppo di persone, sotto forma di abitudini sportive alimentari, o di lettura. Alcune basi di conoscenza sono eccellenti palestre per l'uso delle liste e delle subliste; in effetti l'esempio che esponiamo di seguito è forse, su queste pagine, il primo caso di manipolazione complessa di liste, organizzate in basi di conoscenza efficienti.

La nostra base di conoscenza è, inizialmente, così articolata:

```
persona(vincenzo,preferisce([cioccolata,
mandorle]),evita([corsa,cognac]),rifiuta([]),
                        legge([buzzati,eco,rea])).
persona(stefano,preferisce([cioccolata]),
evita([corsa,mandorle]),rifiuta([]),legge
                        ([buzzati,pitigrilli])).
persona(mirella,preferisce[corsa,cognac],
evita([]),rifiuta([cioccolata,mandorle]),
                        legge([scerbanenco,tolstoi])).
```

Da notare che ognuna delle tre strutture usa liste vuote «[]» per «tenere il posto» di un elemento strutturale per il quale la base di conoscenza non ha elementi particolari da inserire (o per il quale ci si riserva successivamente di inserire valori). D'altro canto, al di fuori del significato sostanziale che mirella cerca di non evitare niente nelle sue abitudini, esiste anche un fatto formale, per cui il Turbo Prolog, in particolare, esige che la struttura generale della base di conoscenza sia ordinata e precisa. La presenza delle «empty list», delle liste vuote, è quindi assolutamente necessaria.

A questo punto realizziamo un costrutto anche complesso, per la costruzione della nostra base di dati. Definiamo, quindi, una regola generale in base alla quale, successivamente, potremo trovare una «persona» che legge buzzati. Creeremo una regola che chiameremo «lettore», che ci permetterà di trovare i lettori di un particolare autore; batteremo:

```
lettore(Autore)if
    persona(Chi,_,_,_,legge(X)) and
    membro(Autore,X) and
    print(Chi,«legge»,Autore).
```

Il senso della regola, tradotto in linguaggio umano, è: «Una persona è un lettore di un autore se legge qualcosa e l'autore è un membro della lista (degli autori, ovviamente) letta dalla persona». Forse un po' circonvoluto, ma speriamo abbastanza chiaro e corrispondente alla regola citata. L'ultima linea permette di inviare in output una risposta simile ad un messaggio, un poco più efficiente del solito messaggio asettico finora visto nelle risposte.

Usiamo, come al solito, la nostra base dati e la nostra regola bell'e pronta per eseguire una scansione della nostra base di dati: quanto vediamo sullo schermo è:

```
Goal:lettore(buzzati)
vincenzo legge buzzati
stefano legge buzzati
2 Solutions
Goal:lettore(X)
vincenzo legge buzzati, X=buzzati
vincenzo legge eco, X=eco
stefano legge buzzati, X=buzzati
stefano legge pitigrilli, X=pitigrilli
mirella legge scerbanenco, X=scerbanenco
mirella legge tolstoi, X=tolstoi
7 Solutions
Goal:
```

È particolarmente interessante il secondo Goal, in base al quale Prolog stampa una risposta, formattata secondo i canoni imposti dalla nostra regola. Senza liste, l'immagazzinamento di tutte le informazioni e delle regole per la loro manipolazione avrebbe richiesto ben altra complessità e volume.

Tutto quello che abbiamo appena visto è stato, volutamente, semplice. Se si considera che ogni lista può manipolare subliste, e che ogni regola può essere complicata e strutturata all'infinito ci si rende conto della possibilità di accedere a strutture di controllo estremamente sofisticate e diversificate. Per un puro caso, in altre pagine della rivista parlo dei sistemi esperti e della tecnica «rule-based» di manipolare la conoscenza in essi custodita. In altre parole il motore inferenziale di cui si parla in quelle pagine con la tecnica descritta, è rappresentato da una serie di regole strutturate come quelle dell'ultimo esempio.

E finalmente abbiamo finito di parlare di liste; ci sono volute tre puntate per sviluppare le nozioni generali del problema; all'utente passa ora il testimone della sperimentazione su strutture più complesse (sotto questo punto di vista il Turbo Prolog Borland, con il suo blocco di esercizi-esempi è una palestra eccezionale di sperimentazione).

E siamo giunti a una svolta nella nostra trattazione: finora ci siamo limitati a descrivere le strutture dal punto di vista generale, senza diversificare il nostro dire in esempi più complessi; ma dalla prossima puntata scenderemo un poco più in dettaglio, ricuperando eventualmente argomenti già trattati in precedenza; tanto per intenderci, la prossima volta parleremo di stringhe e di conversione di tipi, agganciandoci a quanto già detto nei mesi di marzo-aprile. A risentirci! MC