

P
R
O
G
R
E
S
S
O

E
L
E
T
T
R
O
N
I
C
O

A
D
V
A
N
C
E

I
N
E

XXII CONGRESSO INTERNAZIONALE PER L'ELETTRONICA

ROMA, 12-15 MARZO 1975

**ELABORATORI ELETTRONICI:
PROGRAMMAZIONE STRUTTURATA E STRUTTURA DI DATI**

**PROBLEMI RELATIVI AI SERVIZI MOBILI
DI TELECOMUNICAZIONI**

**RASSEGNA INTERNAZIONALE ELETTRONICA E NUCLEARE
00193 ROMA - VIA CRESCENZIO, 9 - TEL. 65.69.343/4/5**

C.MONTANGERo-G.PACINI-F.TURINI

Istituto di Scienze dell'Informazione

Univ. di Pisa

Istituto di Elaborazione dell'Informazione

C.N.R. Pisa

ND-LISP: UN SISTEMA PER LA PROGRAMMAZIONE STRUTTURATA
IN INTELLIGENZA ARTIFICIALE

ABSTRACT

Recent experience has shown that nondeterminism is adequate and attractive to deal with A.I. programming. However, because of the complexity of A.I. problems, means to guide the selection of the alternatives at each choice points must be explicitly introduced. Otherwise, the combinatorial explosion of the computation would become unbearable.

The paper describes the philosophy of ND-Lisp, a language for nondeterministic programming, that allows a very high degree of intervention in the search policy. This goal is achieved without loss of clarity and intuitiveness -that is, structure- of programs, which is the most likely consequence of the introduction of heuristics.

ND-Lisp forces the user to write programs in two conceptually distinct phases: first, a properly nondeterministic algorithm is encoded; then, choice points are exploded into generative functions expressing the associated heuristics and search policies.

This programming method allows to keep conceptually secluded and well localized the heuristic facets of programs. Heuristics can be introduced -may be, by successive refinements- without destructively affecting the original nondeterministic structure of the algorithm.

INTRODUZIONE

Tecniche di programmazione non deterministica sono state ampiamente usate nell'ambito dell'Intelligenza Artificiale (I.A.) <2,4,6,10>. Infatti, i programmi in I.A. tendono a raggiungere un obiettivo (goal) tramite l'esame di diverse possiblita', molte delle quali possono, a posteriori, risultare insoddisfacenti. I linguaggi non deterministici consentono una chiara e concisa formulazione di questo tipo di procedure.

In generale un programma si dice non deterministico <5> se prevede situazioni, punti di scelta, in cui la successiva azione da intraprendere non e' univocamente definita. L'aspetto piu' attrattivo del non determinismo e' che il programmatore puo' ragionare supponendo che ad ogni punto di scelta sia disponibile immediatamente l'alternativa migliore. Questa "forma mentis" porta alla formulazione di programmi semplici e concettualmente chiari.

In pratica, e' necessario che l'alternativa migliore venga in qualche modo selezionata. Qualora la selezione sia lasciata completamente a carico del sistema, ad esempio tramite un meccanismo incorporato di "backtracking", cio' si traduce in una ricerca esaustiva. L'esperienza in I.A. ha mostrato come siano necessari strumenti per intervenire nella selezione, in modo da poter programmare una politica delle scelte che permetta lo sfolcimento delle possibilita' esistenti <1,11>. In caso contrario, per problemi reali, l'esplosione combinatoria del calcolo porta verso tempi di esecuzione praticamente inaccettabili.

Sfortunatamente, l'introduzione esplicita nei programmi non deterministici delle euristiche necessarie per sfoltire l'insieme delle possibilita', sembra cozzare contro l'atteggiamento mentale, che, come abbiamo detto, costituisce la maggiore attrattiva del non determinismo. La conseguenza piu' immediata e' infatti la tendenza verso programmi in cui risulta difficile riconoscere l'originaria natura non deterministica dell'algoritmo, in quanto essa viene mascherata dall'introduzione di componenti euristiche.

ND-Lisp e' stato progettato per fornire un ambiente di programmazione in cui le due esigenze, innegabilmente contrastanti, trovino una ragionevole conciliazione. Cio' e' stato ottenuto proponendo una tecnica in cui la stesura del

programma viene affrontata in due fasi concettualmente distinte, in modo da tenere per quanto possibile separate e localizzate le componenti euristiche che si vogliono introdurre.

FUNZIONI GENERATRICI

Il comportamento di un sistema non deterministico può essere modellato come segue:

ad ogni punto di scelta il sistema genera tanti nuovi ambienti di calcolo quante sono le alternative che si vogliono considerare.

Secondo una terminologia largamente diffusa chiameremo contesto <10,11> l'ambiente di calcolo creato per esaminare una alternativa. Il calcolo procede indipendentemente in ogni nuovo contesto, producendo un risultato soddisfacente oppure un fallimento.

Lo stato del sistema è sintetizzabile con un albero, che diremo albero dei contesti. Ogni nodo non terminale rappresenta un punto di scelta e ha tanti figli quante sono le alternative attualmente in esame; esso memorizza l'insieme delle alternative ancora disponibili e lo stato in cui eventualmente iniziarne l'esplorazione. I nodi terminali corrispondono invece alle alternative attualmente in esame.

Se in un sistema non deterministico è stata incorporata una ben determinata tecnica per la selezione delle alternative, ogni volta che uno dei comandi disponibili per il nondeterminismo viene incontrato, il controllo passa ad un apposito modulo programmato in modo da realizzare la tecnica prescelta. Quest'ultimo, che noi chiameremo generatore, prende in esame lo stato dell'albero dei contesti e lo modifica in obbedienza alla tecnica incorporata. Per esempio, supponiamo che il generatore realizzi una tecnica di backtracking e che nel contesto A (fig.1a) venga eseguito un comando di tipo FAIL. In questo caso

il generatore elimina il nodo A, prende poi in esame il padre di A, cioe' B, e aggiunge un nuovo contesto (fig.1b) in cui tentare la successiva alternativa associata al nodo B. In uno schema di questo tipo, il programmatore puo' condizionare la politica delle scelte solo attraverso i mezzi che il linguaggio fornisce per intervenire sull' operato del generatore, ad es. messaggi di fallimento <7>.

Contrapposta a questa organizzazione caratterizzata da una tecnica incorporata a priori e centralizzata nel generatore, noi proponiamo la struttura schematizzata in fig. 2. Come la figura illustra, ad ogni nodo non terminale e' associata una funzione generatrice, destinata a contenere la strategia che si vuole associare al corrispondente punto di scelta. Le funzioni generatrici devono essere definite dal programmatore il quale puo' servirsi di esse per associare ad ogni singolo punto di scelta la strategia che egli ritiene piu' opportuna.

ND-Lisp non fornisce alcun particolare comando per segnalare la posizione dei punti di scelta. La presenza di punti di scelta e' segnalata dalla chiamata di funzioni generatrici. Quando una funzione generatrice viene attivata, la funzione stessa rimane associata nell'albero dei contesti al nodo corrispondente all'alternativa in cui l'attivazione e' avvenuta. Sarà poi compito della funzione dare vita a nuovi tentativi, usando un apposito comando fornito dal linguaggio <8,9>, e curare la loro gestione secondo la politica che il programmatore intende perseguire.

Ricollegandoci ora a quanto accennato nell'introduzione, quello che noi proponiamo e' una programmazione articolata in due fasi concettualmente distinte, in cui le funzioni generatrici giocano un ruolo logicamente diverso. Nella prima fase il programmatore assume un atteggiamento propriamente non deterministico e inserisce nel suo programma chiamate a funzioni

generatrici unicamente per segnalare la posizione dei punti di scelta. In questa fase non ci si preoccupa affatto della struttura interna delle funzioni generatrici, ma ci si limita a pensare che esse selezionino ogni volta direttamente l'alternativa migliore. Nella seconda fase le funzioni generatrici vengono effettivamente definite, introducendo in tal modo nel programma le euristiche che si ritengono piu' opportune. Le due fasi sono concettualmente indipendenti, cosicche' il programmatore puo' sofisticare gradualmente le euristiche introdotte, senza turbare la struttura del programma base.

CONCLUSIONI

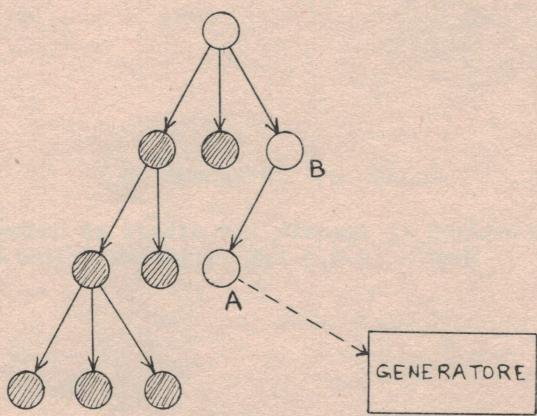
La metodologia suggerita dal linguaggio ND-Lisp puo' essere riguardata come una diretta derivazione della idea di base della programmazione strutturata, cioe' programmazione per raffinamenti successivi (top-down) <3>. In effetti, quello che si suggerisce e' la scrittura di un programma dove certe funzioni (le funzioni generatrici) rimangono da definire, essendo specificato solamente lo scopo che esse debbono ottenere. Le funzioni generatrici saranno da realizzare in tempi logicamente successivi; la loro definizione appare, da questo punto di vista, come il raffinamento di un programma scritto in una fase precedente.

La gerarchia tra i due livelli di programmazione non e' pero' una gerarchia genericamente temporale. A ciascuno dei due livelli corrisponde un ben preciso significato, in quanto e' ben definito quello che nella prima fase viene rimandato ai raffinamenti successivi: nella prima fase si codifica un algoritmo propriamente non deterministico; successivamente, i punti di scelta vengono espansi in vere e proprie procedure che diventano i contenitori dove il programmatore mette l'euristica che intende associare ai singoli punti di scelta.

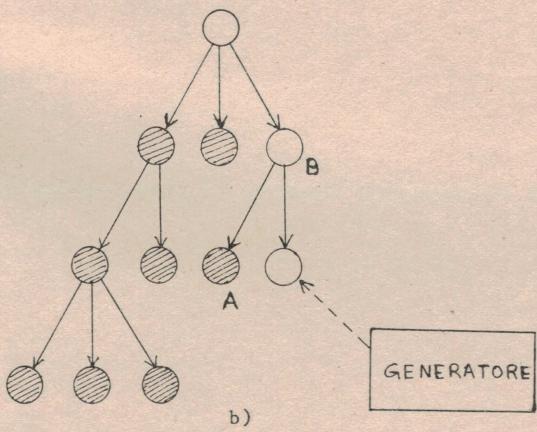
I due livelli di programmazione consentono di mantenere concettualmente separati e praticamente localizzati gli aspetti euristici del programma. Le euristiche possono essere introdotte senza che cio' interferisca distruttivamente con la originaria natura non deterministica dell'algoritmo.

BIBLIOGRAFIA

- 1-Bobrow, D.G. e D.V. Wegbreit. "A model and stack implementation of multiple environments". Comm. ACM, vol. 16, n. 10, pp.591-603.
- 2-Confield Smith, D. e H.J. Enea. "Backtracking in MLISP2". Proc. 3.o IJCAI, Stanford 1973, pp.671-685.
- 3-Dahl, O.J., Dijkstra, E. e C.A.R.Hoare. "Structured programming". New York, Academic Press, 1972.
- 4-Davies, D.J.M. "POPLER: a POP2 PLANNER". MIP-89, School of A.I. University of Edinburgh.
- 5-Floyd, R.W. "Nondeterministic algorithms". JACM, vol.14, n.4, Ottobre 67, pp.636-644.
- 6-Hewitt, C. "Procedural embedding of knowledge in PLANNER". Proc. 2.o IJCAI, London 1971.
- 7-Hewitt, C. "PLANNER: a language for manipulating models and proving theorems in a robot". A.I. memo 168, MIT, 1970.
- 8-Montangero, C., Pacini, G. e F.Turini. "Two-level control structure for nondeterministic programming". Nota Interna I.E.I. B74-37, Pisa, ottobre 74.
- 9-Montangero, C., Pacini, G. e F.Turini. "ND-Lisp reference manual". Nota Tecnica I.E.I., Pisa, (in preparazione).
- 10-Rulifson, J.F., Waldinger, R.J. e J.A.Derkson. "QA4: procedural calculus for intuitive reasoning". A.I. Center, Tech.Note 73, S.R.I., Stanford, novembre 73.
- 11-Sussman, G.J. e D.V. McDermott. "From PLANNER to CONNIVER, a genetic approach". Proc. AFIPS FJCC 1972, pp.1171-1179.



a)



b)

Fig. 1. I nodi tratteggiati indicano tentativi falliti.

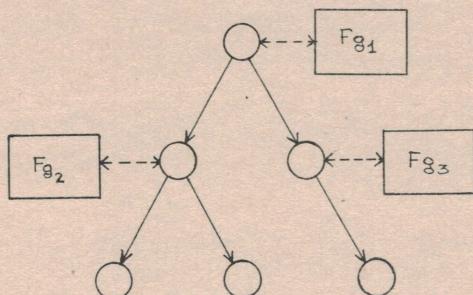


Fig. 2