Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

GRAFI AND/OR GENERALIZZATI COME
LINGUAGGIO DI PROGRAMMAZIONE

P. Asirelli

Nota Interna B76-12

Maggio 1976

ABSTRACT

The paper deals with the interpretation of a programming language, based on the Generalized AND/OR Graphs formalism. A program in the language is a set of productions. Program execution is performed by searching a solution of the corresponding Generalized AND/OR graph.

1. INTRODUZIONE

La generalizzazione dei grafi AND/OR (GAG) come modello di risoluzione di problemi non numerici (problem solving) attraverso la riduzione in sottoproblemi e l'equivalenza tra i GAGs e le grammatiche di tipo Ø dimostrata in (1),hanno suggerito l'idea di definire un linguaggio basato su tale formalismo, i cui programmi sono insiemi di produzioni, come avviene per gli schemi di programmi FOLP in (2).

In sostanza, avendo a disposizione:

1)- un insieme di produzioni che definiscono uno schema di programma e le primitive usate dallo stesso;

2)- un meccanismo di valutazione (applicazione delle produzioni), e' possibile interpretare un qualunque programma, espresso nel linguaggio cui appartiene lo schema di programma definito dall'insieme di produzioni, cercando una soluzione del grafo AND/OR generalizzato corrispondente al programma dato.

In questo lavoro viene descritto un interprete per il linguaggio di programmazione basato sui grafi AND/OR generalizzati.

La sintassi delle relazioni funzionali, o produzioni, e del programma (problema) vengono date nella parte terza.

Il grafo AND/OR generalizzato, viene costruito partendo dal problema dato e ottenendo, dall'applicazione ad esso delle relazioni funzionali mediante l'operatore di riduzione (che in questo caso e' un meccanismo di pattern matching), tanti nodi del grafo quante sono le relazioni funzionali applicabili. Ogni nodo del grafo cosi' ottenuto, detto PSG (Potential Solution Graph) e' a sua volta un grafo il cui nodo iniziale e' un nodo AND.

Un PSG costituisce un tentativo, o contesto, di risoluzione del problema e, quando tutti i nodi discendenti dal nodo AND sono risolti, la soluzione e' data dalla combinazione lineare di tutti i discendenti. Da ogni PSG vengono generati altri tentativi, sempre con l'applicazione delle regole funzionali; il PSG che viene espanso per primo e' l'ultimo generato al passo precedente, cioe', i PSG vengono espansi seguendo una tecnica LIFO.

Nella parte quarta verra' descritto l'algoritmo di interpretazione (valutazione) del GAG, con la definizione di che cosa si intende per relazione funzionale applicabile e che cosa si intende per nodi risolti di un PSG o meglio quand'e' che un PSG puo' essere considerato una soluzione.

Si mostrera' anche come l'interprete sia in grado di funzionare come esecutore simbolico.

Il programma che realizza l'interprete e' stato scritto inizialmente in LISP 1.5 sul calcolatore I.B.M. 360/67 del C.N.U.C.E., quindi, in seguito alla sostituzione del 360/67 con il 370/168, il programma e' stato riadattato al MAGMA LISP (4) , di cui viene usata solo la parte LISP pura.

Infine nella parte quinta verranno dati alcuni esempi del

funzionamento del sistema che e' stato realizzato per lavorare in modo conversazionale con l'utente.

2. SEMANTICA

Con un calcolatore, il metodo piu' semplice per affrontare il problema della ricerca di una soluzione di un problema iniziale dato, nello spazio di quelle possibili, e' quello di tentare varie vie finche' casualmente non se ne ottiene una.

Un modo di risolvere il problema e' quello della 'riduzione in sottoproblemi'.

Analizzando il problema iniziale, si puo' concludere che esso e' suddivisibile in sottoproblemi. Se si e' in grado di risolvere i sottoproblemi, si ha anche una soluzione del problema iniziale. I sottoproblemi possono a loro volta essere scomposti in altri sottoproblemi e continuando cosi' si possono raggiungere dei problemi considerati 'terminali', la cui soluzione e' banale.

Esistono anche degli operatori, detti di riduzione, che applicati alla descrizione di un problema, forniscono un insieme di descrizioni di sottoproblemi.

Il procedimento sopra descritto puo' essere schematizzato con l'uso dei grafi AND/OR che hanno la struttura di Fig. (1), dalla quale si deduce che, detto A il problema iniziale, esso sara' risolto se almeno uno dei problemi alternativi, M, N ed F e' risolto. Per questa ragione il nodo A viene detto nodo OR.

Risolvere il problema M significa risolvere entrambi i problemi B e C, per questa ragione il nodo M viene detto nodo AND, cosi' pure per il nodo N, che sara' risolto se, sia D che E sono risolti.

I grafi AND/OR possono essere usati per rappresentare metodi di riduzione di problemi mediante l'applicazione degli operatori di riduzione.

L'applicazione di un singolo operatore di riduzione ad una descrizione di problema produce un nodo AND intermedio e i suoi nodi OR successori.

In un grafo AND/OR si ha un nodo S che e' associato alla descrizione del problema iniziale. I nodi che corrispondono a descrizioni di problemi primitivi, vengono chiamati 'nodi terminali'.

Il processo di ricerca su un grafo AND/OR, corrispondente alla ricerca di una soluzione nello spazio di quelle possibili, ha lo scopo di mostrare che il nodo S iniziale e' 'risolto'. La definizione di nodo 'risolto' e' la seguente:

a)- i nodi terminali sono risolti (perche' associati a problemi primitivi);

b)- se un nodo non terminale e' AND, esso e' risolto se, e solo se, tutti i suoi successori sono risolti;

c)- se un nodo non terminale e' OR, e' risolto se, e solo se, almeno uno dei successori e' risolto.

Un grafo soluzione e' un sottografo di nodi risolti, tale che secondo le definizioni a), b) e c) dimostra che il nodo iniziale S

e' risolto.

I nodi 'irrisolvibili' possono essere definiti nel modo seguente:

a)- un nodo non terminale senza successori e' 'irrisolvibile';

b)- se un nodo e' AND, esso e' irrisolvibile se almeno uno dei successori e' irrisolvibile;

c)- se un nodo e' OR, esso e' irrisolvibile se tutti i successori sono irrisolvibili.

In generale si vuole dimostrare che il nodo iniziale (problema iniziale) e' risolto, in questo caso il processo di soluzione e' molto simile ad un processo di dimostrazione di teoremi.

Sempre in generale i grafi AND/OR sono definiti implicitamente dalla descrizione del problema iniziale e dagli operatori di riduzione, in questo caso conviene introdurre l'operatore successori Γ (la cui applicazione corrisponde all'applicazione di tutti gli operatori di riduzione applicabili), il quale, applicato ad una descrizione di problema, produce tutti gli insiemi di descrizioni di problemi successori. Risolvere il problema consiste nel generare una parte del grafo AND/OR, che sia sufficiente a dimostrare che il nodo iniziale e' risolto.

E' pero' da notare che l'uso dei grafi AND/OR come modello di risoluzione di problemi attraverso la riduzione degli stessi, presuppone che i singoli sottoproblemi possano essere risolti indipendentemente l'uno dall'altro, cosa che in generale puo' non avvenire come per la classe dei problemi la cui soluzione richiede l'uso di risorse di cui si ha disponibilita' limitata. (Un esempio di tali problemi e' dato in (5)).

Altri esempi sono dati dai problemi in cui la soluzione ad un sottoproblema puo' modificare altri sottoproblemi; problemi di questo tipo si verificano quando per la descrizione del problema sono necessarie delle variabili comuni. Ci sono infine problemi la cui formulazione richiede l'uso di concetti definiti come l'unione di termini non indipendenti, di questi problemi viene data una descrizione dettagliata in (5), e anche piu' avanti in questo capitolo.

Le difficolta' ora viste, di usare grafi AND/OR per la riduzione di problemi, quando non esista una indipendenza tra i sottoproblemi, sono state superate con la generalizzazione dei grafi AND/OR.

Un grafo AND/OR e' completamente definito da un insieme D di problemi e un insieme F di operatori di riduzione. Indichiamo poi con $\perp$ (bottom) e $\top$ (top) rispettivamente la 'soluzione del problema indefinito' e 'problema terminale'.

Ogni operatore di riduzione e' rappresentato da produzioni del tipo:

f_1 : $\perp \Longrightarrow H_1 , H_2 , \ldots H_n , B_1 , B_2 , \ldots B_n$

f_2 : $\perp \Longrightarrow B_1 , B_2 , B_3 , \ldots\ldots B_n$

f_3 : $B_1 \Longrightarrow H_2$

f_4 : $B_2 \Longrightarrow H_3 , H_4 , B_1$

f_j : $B_j \Longrightarrow T$

in cui H_1 ,H_2 ,... e B_1 ,B_2 .... rappresentano i sottoproblemi in cui puo' essere scomposto il problema iniziale $\perp$. (L'ultima produzione afferma che B_j e' terminale).

Una tale rappresentazione di operatori, puo' essere usata solo nel caso in cui i sottoproblemi sono indipendenti l'uno dall'altro. Nel caso invece in cui sono dipendenti fra loro, e' necessaria una generalizzazione dei grafi AND/OR per permettere che gli operatori di riduzione vengano rappresentati con produzioni nella cui parte sinistra possano esistere piu' di una informazione.

Un grafo AND/OR generalizzato e' definito come $\gamma = (\text{'O'}, \text{'T'}, \text{'A'}, \text{'E'}, \text{'S'})$ in cui 'O' e' un insieme di nodi OR non terminali, 'T' e' un insieme di nodi OR terminali, 'A' e' un insieme di nodi AND, 'E' e' un sottinsieme di $(O \cup T) \times A \cup A \times (O \cup T)$ i cui elememnti sono archi orientati, 'S' appartiene ad 'O' ed e' chiamato nodo iniziale, o nodo di partenza.

Il grafo in Fig. (2), e' un GAG in cui : i nodi etichettati con le lettere maiuscole sono nodi OR non terminali (problemi); i nodi etichettati con lettere minuscole sono nodi OR terminali (problemi risolti), infine i nodi etichettati con le cifre sono nodi AND e indicano gli operatori di riduzione.

Gli operatori di riduzione si applicano generalmente ad un insieme di problemi, mentre, come abbiamo visto precedentemente nel caso di grafi AND/OR, gli operatori si applicano ad un solo problema, cioe', nel caso di GAG si prevede un 'contesto' in cui il problema deve essere ridotto.

La ricerca di un grafo AND/OR generalizzato si propone di trovare un grafo soluzione che mostri che il nodo iniziale e' risolto.

I grafi soluzione sono definiti sull'albero AND/OR generalizzato γ' ottenuto duplicando quei nodi OR del grafo γ originale, che hanno piu' di un nodo entrante.

La definizione ricorsiva di nodi risolti e irrisolvibili e' analoga a quella data per i grafi AND/OR.

Un PSG (Potential Solution Graph) e' ogni sottografo s di δ' tale che:

i)- il nodo S e' in s;

ii)- se il nodo n e' in s, allora :
se n e' un nodo OR non terminale, allora in s c'e' un successore di n;
se n e' un nodo AND, allora tutti i successori di n sono in s.

Un PSG s in cui il nodo S e' etichettato come risolto si dice 'grafo soluzione'.

Abbiamo visto che in un GAG, gli operatori di riduzione si applicano anche ad un insieme di problemi, cioe' sono dati da produzioni del tipo:

$$f_1 : d_1^1 , d_2^1 , d_3^1 \ldots . d_h^1 \Longrightarrow d_1^2 , d_2^2 , d_3^2 \ldots . . d_h^2 .$$

Nel linguaggio che qui introdurremo i d_i^j sono formule atomiche del calcolo dei predicati del primo ordine oppure delle relazioni di uguaglianza, che in sostanza sono dei letterali.

f_1 : $\perp \Longrightarrow$ CAR(y,z),CDR(x,y), x=(a.(b.c))

f_2 : CAR(x,y), x=(w1.w2) $\Longrightarrow$ y=w1

f_3 : CDR(x,y), x=(w1.w2) $\Longrightarrow$ y=w2

Come si vede le produzioni f_2 ed f_3 definiscono, mediante relazioni, le funzioni un certo linguaggio. La produzione f_1 (programma principale) richiede di calcolare il (car (cdr (a.(b.c)))). In sostanza quindi le produzioni possono essere viste come dei sottoprogrammi dall'applicazione dei quali si puo' ottenere la soluzione del programma iniziale in cui i nodi terminali sono dati dalle relazioni di uguaglianza. Cioe' si definisce 'PSG soluzione' quel PSG i cui nodi terminali sono tutti relazioni di uguaglianza.

Vediamo ora in Fig. (3) il GAG corrispondente alla computazione del problema generico $\perp$ definito dall'insieme delle produzioni sopra dette.

Come si vede la computazione suddetta corrisponde al calcolo del (car (cdr (a. (b.c)))) del LISP.

3. SINTASSI

La Sintassi dei programmi o sottoprogrammi (produzioni) e' basata sulla sintassi del calcolo dei predicati del prim'ordine.

Nel calcolo dei predicati del prim'ordine si definisce come 'termine' una qualunque espressione che sia un simbolo di variabile, un simbolo di costante oppure un simbolo di funzione n-adica seguito da n termini separati da virgola e chiusi tra due parentesi, ad esempio:

1) $f^4\ (a,x,f^3(b,b,y),z)$

e' un termine con f^4 ,f^3 simboli di funzione; a,b costanti e x,y e z simboli di variabile.

Si definisce poi come 'formula atomica' una qualunque espressione formata da un simbolo di predicato n-adico, seguito da n termini separati da virgole e chiusi tra parentesi, ad esempio

2) $p^2\ (f^4\ (a,x,g^3\ (b,b,y),z),a)$.

La precedente puo' anche essere scritta come :

3) $P(h,a),F(a,x,m,z,h),G(b,b,y,m)$

con P, F, G simboli di predicato, a,b costanti e h,x,m,z,y simboli di variabile. In questo modo si da' alle funzioni una particolare rappresentazione e cioe': tutte le funzioni sono rappresentate da relazioni, che devono avere un argomento in piu', che e' la variabile cui viene assegnato il risultato della computazione, cioe' una funzione g(x,y), deve essere scritta nella forma: G(x,y,z) con z tale che g(x,y)=z. Ancora si puo' vedere facilmente che:

4) $P(h,r),F(r,x,m,z,h),G(s,s,y,m),\ r=a,\ s=b$

e' equivalente alla 3), in cui r=a ed s=b vengono detti 'legami di variabile'.

In questo lavoro , con:

-'predicato' si intende una formula atomica in cui non compaiono ne' simboli di costante ne' simboli di funzione come argomenti, ma soltanto simboli di variabile;

-ad una formula atomica in cui compaiono simboli di funzione come argomento, ma non simboli di costante, corrisponde una 'lista di predicati' del tipo della 3);

-infine ad una formula atomica in cui compaiono simboli di funzione, simboli di costante e simboli di variabile corrisponde :

'lista di predicati' + 'legami di variabile'

in cui 'legami di variabile' sono relazioni di uguaglianza come nella 4).

In (6) vengono fatte delle considerazioni sul calcolo dei predicati del prim'ordine come linguaggio di programmazione.

Le formule atomiche usate sono del tipo della 2), in cui quindi i termini che seguono l'identificatore di predicato, possono essere anche costanti o funzioni.

In (6), il linguaggio considerato e' del tipo AND/OR e quindi viene lasciata fuori tutta quella classe di problemi la cui risoluzione richiede uno speciale trattamento delle relazioni di dipendenza.

Mentre in (6) vengono sviluppati solo i predicati, in questo lavoro si possono interpretare anche le funzioni, cioe', se le funzioni non devono essere interpretate , esse compaiono nella struttura 4) come assegnate a delle variabili, altrimenti, vuol dire che esistono delle produzioni che ne prevedono lo sviluppo, allora compaiono come 'predicati'. Facciamo un esempio riprendendo in considerazione la 2); se per f e g non ci sono produzioni che ne prevedono lo sviluppo la 2) verra' scritta:

$$P(x,y),\ x=f^{4}(a,x,g^{3}(b,b,y),z),\ y=a$$

altrimenti la 2) viene riscritta come la 4).

Supposto quindi che F e G siano interpretate, una riscrittura della 2) come la 4), permette di mettere in evidenza il fatto che alla 4) puo' essere applicata una produzione che, ad esempio, riduce soltanto uno o piu' 'predicati' nel senso in cui qui vengono intesi, in qualunque ordine essi compaiano, senza dover calcolare prima g, poi f e quindi p, poiche' la dipendenza di P da F e da G risultera' evidenziata dai nomi delle variabili e dai loro legami, che quindi vengono tenuti separati.

Si puo' anche osservare che la trasformazione prima vista, da 2) a 4), permette (in casi in cui, argomenti diversi dello stesso predicato, o, argomenti uguali di predicati diversi, siano dati dal risultato del calcolo della medesima funzione) di evitare di ricalcolare piu' volte la stessa funzione. Supponiamo infatti di avere la seguente formula atomica:

5) $$p^{4}(a,f(x,y),g(a,f(x,y),z))$$

la f(x,y) deve essere calcolata due volte, se la 5) viene riscritta come:

6) $$P(m,n,q),F(x,y,n),G(m,n,z,q),\ m=a$$

la F(x,y,n), puo' essere calcolata una sola volta, e il suo valore essere usato da P e da G tramite il legame della variabile n.

Inoltre in 4) r=a ed s=b, specificano, essendo a e b costanti, che le variabili r ed s non sono contestuali, sono cioe' indipendenti dal modo in cui viene risolto il problema.

Le produzioni che l'utente puo' definire sono del tipo:

S := S'

dove:
S e' una lista il cui CAR e' una lista di predicati e il cui CDR e' una lista di legami di variabile.
S' e' una lista del tipo di S oppure una lista di legami di variabile.

Il sistema accetta le produzioni espresse secondo il seguente formalismo :

```
<produzione>                :=(<parte sinistra>.<parte destra>)¦
                            (LAMBDA(<lista di variabili>)

<parte sinistra>            :=(<lista di predicati>.<lista di leg.
                              di var. >)

<parte destra>              :=<parte sinistra>¦
                              <lista di leg. di var.>

<lista di predicati>:=<predicato>¦(<predicato><lista di
                             predicati>)

<predicato>                 :=<identificatore>    ¦
                            (<identificatore> <lista di variabili>)

<identificatore>            :=<lettera>¦<lettera><identificatore>
<lista di variabili>:= NIL  ¦  <variabile>    ¦
                              (<variabile>.<lista di variabili>)

<variabile>                 :=(VAR <identificatore>)
<lista di leg. di var.>:=<lista vuota>     ¦
                               ((<variabile> <valore>).<lista di
                               leg. di var.>)

<lista di identificatori di variabile>:=NIL                ¦
                                        <identificatore>¦
                                       (<identificatore>.
                                  <lista di identif. di variabile>)

<valore>                    :=po ¦ NIL ¦ <atomo> ¦<variabile>¦
                            (<valore>.<valore>)

<atomo>                     :=<lettera>¦<numero>¦<atomo><atomo>
```

Le produzioni sono delle relazioni funzionali del tipo:

$$F(x,y),x=\forall \Longrightarrow H(z,j),j=x \qquad \text{oppure}$$

$$F \Longrightarrow H,D$$

dove nel primo caso F ed H sono funzioni, quindi le produzioni possono essere viste come delle subroutines che possiedono delle variabili formali e locali. Perche' il sistema sia in grado di distinguere le variabili formali dalle altre l'utente deve far precedere le produzioni dal 'LAMBDA' seguito dalla lista di variabili che devono essere considerate formali.

E' stato stabilito di dare alle variabili la forma: (VAR X), per evitare confusioni con i nomi dei predicati o con i <valori> eventualmente assegnati alle variabili, l'unica eccezione a questa regola si ha nel caso della lista di variabili che si trova nella <parte sinistra> di una produzione preceduta da 'LAMBDA' che indica che la lista che segue e' una lista di atomi che rappresentano nomi di variabili.

Se una variabile puo' assumere un qualunque valore imprecisato, si dice che il valore della variabile e' $\forall$, che all'interno del sistema viene rappresentato dall'atomo 'PO'.

Qualora una variabile, argomento di una funzione o che compare come <valore> di un'altra variabile, non compaia nella lista di legami di variabile come legata ad un certo valore, si intende avere valore 'PO', e' comunque permesso esplicitare che il valore di tale variabile e' 'PO'. Per esempio e' una lista di legami di variabile la seguente:

```
(((VAR x) a)((VAR y) (VAR x))((VAR z) ((VAR x)(VAR y))))
```

dove si intende che:

- il valore della variabile x e' a;
- il valore della variabile y e' il valore della variabile x, cioe' a;
- il valore della variabile z e' una lista in cui il 'car' e' il valore di x e il 'cdr' e' il (<valore di y>.NIL).

E' una produzione accettata la seguente:

```
((((TIMES (VAR X) (VAR Y) (VAR Z))
   (PLUS (VAR H) (VAR L) (VAR Y)))
  ((VAR H) 3) ((VAR L) 4))
  (((TIMES (VAR X) (VAR H) (VAR B))
   (PLUS (VAR B) (VAR M) (VAR R)))
  ((VAR H) 3)((VAR L) 4)))
```

oppure la stessa preceduta da:(LAMBDA (x y z).....

Il problema che l'utente puo' presentare al sistema perche'

questo sia risolto deve essere espresso secondo il seguente formalismo:

<problema> := <parte sinistra>

e' un problema il seguente:

```
((APPEND (VAR x) (VAR y) (VAR z))
        ((VAR x)(a b))((VAR y)(c d))))
```

4. ALGORITMO DI INTERPRETAZIONE

4.1 - Il sistema e' stato realizzato per lavorare in forma conversazionale con l'utente, al quale e' consentito di operare nei modi seguenti:

a)- fornire un insieme di produzioni e richiedere che queste vengano stabilmente conservate su di un file su disco;

b)- dare un insieme di produzioni da conservare su disco e successivamente il problema che deve essere risolto, specificando il nome del file che contiene le produzioni da applicare al problema;

c)- specificare una profondita' massima oltre la quale le espansioni dei PSG devono essere interrotte, oppure, qualora si voglia dare una profondita' di espansione infinita, rispondere con un 'NIL' alla richiesta del sistema. In quest'ultimo caso i PSG vengono espansi fino a che non viene raggiunta una soluzione o fino a che tutti i tentativi sono falliti.

4.2 - E' bene a questo punto precisare alcune assunzioni fatte, e cioe':

a)- per produzione applicabile si intende la produzione la cui parte sinistra e' formata da uno o piu' predicati che figurano tra i predicati del problema, cioe', indicando con U l'insieme dei predicati che compaiono nella parte sinistra della produzione in esame, e con A, l'insieme dei predicati che compaiono nel problema, si definisce produzione applicabile, la produzione per cui risulta:

$$U \subseteq A.$$

b)- Un nodo si dice risolto, se e' della forma:

(<variabile> <valore>)

quindi, poiche' come abbiamo visto un PSG e' un sottoalbero, vedi Fig.(4), un PSG e' una soluzione se $N_1, N_2, N_3, \ldots\ldots, N_h$ sono risolti, ossia se sono legami di variabile. Infatti se cosi' non fosse vorrebbe dire che un nodo contiene almeno un predicato e

---

N.B. La relazione $U \subseteq A$, deve valere per la parte <identificatore> dei predicati. Gli argomenti, qualora questi siano presenti, devono essere in numero uguale, nel problema e nella produzione, relativamente allo stesso predicato.

quindi il PSG in esame deve essere ulteriormente espanso, cio' significa che la sequenza $N_1\ N_2\ N_3\ \ldots\ N_h$ non e' ancora una soluzione.

c)- Infine, prima di passare ad una descrizione piu' dettagliata, definiamo che cosa e' il risultato dell'applicazione dell'operatore di riduzione al problema. Abbiamo detto che l'operatore di riduzione e' un meccanismo di pattern matching applicato ad una produzione ed al problema, allora:

il risultato del matching del problema con una produzione e' la <parte destra> della produzione stessa, nella quale vengono aggiunti dei legami di variabile, o vengono cambiati i valori delle variabili che gia' risultano legate, questo a causa dei valori effettivi assegnati alle variabili dal problema.
Infatti, come gia' abbiamo detto, le produzioni sono degli schemi che quindi valgono in generale; quando poi lo schema viene usato per risolvere un problema specifico, le variabili che compaiono nelle produzioni devono assumere di volta in volta i valori specificati dal problema. Proprio per questo, ogni volta che una produzione viene usata, le variabili locali devono essere ridenominate per evitare che ci siano delle variabili locali della produzione di nome uguale ad alcune variabili del problema.

4.3 L'INTERPRETE

p.1)- Si leggono dal file specificato dall'utente, le produzioni che vengono poste in una lista LISPROD.

p.2)- Si esegue il matching del problema (PROB) con la prima produzione che compare in LISPROD;
se il matching non fallisce p.3, altrimenti : p.5.

p.3)- Se il risultato del matching e' una soluzione : p.4.
altrimenti si pone il PSG ottenuto in testa alla lista OPENL e quindi p.5.(*1)

p.4)- Il PSG ottenuto viene semplificato in modo da eliminare eventuali legami di variabile superflui e restituito come risultato del problema fornito dall'utente.

p.5)- Si mette la prima produzione di LISPROD in LISPROD1,
si elimina da LISPROD la prima produzione;
se LISPROD e' la lista vuota si assegna a LISPROD LISPROD1 e quest'ultima diviene la lista vuota, quindi p.6, altrimenti p.2.

p.6)- Se la lista OPENL e' vuota, si ha come risultato un fallimento poiche' nessuna delle produzioni date riesce a ridurre il problema iniziale.(*2)
Se la lista OPENL non e' vuota contiene dei PSG il primo dei quali viene assegnato a PROB e tolto da OPENL, quindi si ripete da p.2.

(*1)
L'inserimento del PSG in una pila OPENL, permette di salvare lo stato del programma, sia per quanto riguarda la memoria, sia per quanto riguarda il controllo, realizzando quindi un meccanismo dei contesti per la risoluzione non-deterministica del problema iniziale.

(*2)
Cio' si puo' verificare perche' :
1)- non c'e' soluzione;
2)- oppure non ci sono le informazioni necessarie; vedi gli esempi della parte quinta.

L'algoritmo puo' essere facilmente modificato in modo da funzionare come un esecutore simbolico. Infatti, se invece di interrompere il processo appena viene trovata una soluzione, continuassimo a tentare le altre possibilita' rimaste inesplorate, potremmo arrivare ad un insieme di soluzioni corrispondenti all'esecuzione del programma per specifici insiemi di dati iniziali.
Inoltre le soluzioni utilizzano uno stato simbolico; cioe', le variabili possono essere associate ad espressioni contenenti simboli di variabile e costruttori di tipi di dati astratti, come ad esempio:

x=(cons a b).

4.4 ALGORITMO DI MATCHING

Gli argomenti di questo algoritmo, che realizza l'operatore di riduzione, sono il problema stesso (PROB) e la parte sinistra della produzione con la quale deve essere verificato il matching.

p.1)- Se i predicati del problema e della produzione soddisfano alla condizione a) del 4.2, si esegue p.2, altrimenti viene dichiarato 'FAIL'.

p.2)- Si controlla che esista una corrispondenza tra le variabili di uno stesso predicato che compare nella produzione e nel problema, questo controllo viene effettuato per ogni predicato della produzione.
Se tale corrispondenza esiste per tutte le variabili di tutti i predicati della produzione si dichiara 'T' (che indica il successo del matching), altrimenti si dichiara 'FAIL'.

Per quanto riguarda la corrispondenza che deve esistere tra le variabili di uno stesso predicato e nel problema e nella produzione, e' da notare che non hanno importanza i nomi delle variabili, quanto l'ordine con cui esse seguono l'identificatore di predicato e i valori cui sono legate, cioe', per ogni variabile di un predicato della produzione, il <valore> a cui risulta legato nella <lista di legami di variabile>, deve coincidere con il <valore> a cui e' legata la variabile corrispondente del problema. La coincidenza dei valori anzi detti non e' tanto semplice da trovare, come invece potrebbe sembrare a prima vista, poiche' come si puo' vedere nella definizione sintattica di <valore> nel cap. 3, un <valore> puo' essere dato da una <variabile>, allora in questo caso, per verificare il matching si deve ricercare il <valore> della variabile, ricercando il <valore> della variabile a cui risulta legata nella <lista di legami di variabile>, si puo' quindi generare un processo di ricerca non troppo semplice. Da quanto ora detto emerge il problema della sicurezza di non avere nelle liste di legami di variabile dei cosi' detti 'cicli', ossia variabili che risultano legate l'una all'altra senza una via di uscita.

Vediamo adesso alcuni semplici esempi che possono chiarire meglio quand'e' che un matching fallisce o meno.

Sia :

FF (x,y,z), x=(a b), y=(c d), z=$\forall$

il problema da risolvere e sia :

FF (h,l,m), h=∀ ⟹ m=l

la produzione con la quale deve essere verificato il matching. In questo caso l'algoritmo di matching registra un 'T', poiche' :

1)- i nomi dei predicati coincidono;
2)- le variabili corrispondenti nel problema e nella produzione sono :

h≡x ; l≡y ; m≡z ;

inoltre i valori a cui sono legate h, l ed m e' ∀.

Se invece la produzione era:

FF(h,l,m), l=(c d) ⟹ m=l

avremmo avuto ancora 'matching' poiche' il valore di h e m sarebbe stato ∀, e il valore di l avrebbe coinciso con il valore di y. Il matching avrebbe invece registrato un 'FAIL' se la produzione fosse stata:

FF(h,l,m), l=(a b) ⟹ m=l.

Vediamo, schematicamente, come viene trattato un problema particolare applicando gli algoritmi di interpretazione e di pattern matching.

siano date le seguenti produzioni:

1) TIMES (X,Y,Z), X=Ø $\Longrightarrow$ Z=Ø;

2) TIMES (X,Y,Z), X=(S W1) $\Longrightarrow$ TIMES(W1,Y,W2),PLUS(Y,W2,Z);

3) PLUS(X,Y,Z), X=Ø $\Longrightarrow$ Z=Y;

4) PLUS(X,Y,Z), X=(S W1), Z=(S W2) $\Longrightarrow$ PLUS(W1,Y,W2), Z=(S W2).

con X,Y e Z variabili globali; W1 e W2 variabili locali; S simbolo di una funzione che calcola il successore di un numero e che in questo caso non e' interpretata.

Il problema sia:

TIMES(X1,X2,X3), X1=(S Ø), X2=(S (S Ø));

vediamo in base al grafico di Fig. (5), come questo viene sviluppato.

L'applicazione della prima produzione fallisce poiche' prevede che il primo argomento del predicato 'TIMES' abbia valore Ø; la seconda produzione e' l'unica che genera una scomposizione del problema. Poiche' i tentativi fatti con la terza e quarta produzione falliscono (non coincidono i nomi dei predicati nel problema e nelle produzioni), il problema attuale diviene quello ottenuto dall'applicazione della seconda produzione.

Le uniche produzioni che, applicate al problema attuale, restituiscono una ulteriore scomposizione sono la prima e la quarta; ancora il PSG che viene espanso e' quello generato con l'applicazione della quarta produzione poiche' e' l'ultimo in ordine di generazione.

Si procede ancora applicando in ordine le produzioni, fino a che non si ottiene un nodo terminale che e' W2=Ø; a questo punto il processo termina.

Il risultato sara' dato dalla combinazione lineare di tutti i nodi terminali (relazioni di uguaglianza), che, con un algoritmo opportuno, vengono semplificati ottenendo :

W5=Ø, X3=(S(S Ø)), W1=Ø, X2=(S(S Ø)).

RINGRAZIAMENTI

Ringrazio vivamente il Prof. G. Levi per avermi seguita ed aiutata nello studio e nella realizzazione di questo lavoro.

```
magmalisp
EXECUTION BEGINS...
***  MAGMA - LISP  ***
F: load
A: (gagip)
GAGLP
F: gagip
A: ()
(VUOI SOLIANTO DEITARE PRODUZ, ?-NUMEF | NIL)
()
(QUAL'E' IL PROBLEMA)
((gffa (var x))((var x) socrates>
(OK? -Y | NIL)
y
(VUOI DETTARE DELLE PRODUZ, ?-NOMEF | NIL)
gffal
+
(lambda (x)((gffa (var x)))(((fallible (var x))
(greek (var x)))>
(VUOI SALVARLA ?-Y | NIL)
y
+
(lambda (x)((fallible (var x)))((human (var x))>
(VUOI SALVARLA ?-Y | NIL)
y
+
(lambda (x)((human (var x))((var x) turing))
(((var x) turing)>
(VUOI SALVARLA ?-Y | NIL)
y
+
(lambda (x)((human (var x))((var x) socrates))
(((var x) socrates>
(VUOI SALVARLA ?-Y | NIL)
y
+
(lambda (x)((human (var x))((var x) socrates))
(((var x) socrates>
(VUOI SALVARLA ?-Y | NIL)
()



+
(lambda (x)((greek (var x))((var x) socrates))
(((var x) socrates>
(VUOI SALVARLA ?-Y | NIL)
y
+
()
(VUOI LA RISTAMPA DEL FILE ORA SCRITTO ? -Y | NIL)
y
(LAMBDA (X) ((GFFA (VAR X))) (((FALLIBLE (VAR X)) (GREEK (VAR X)))))
(LAMBDA (X) ((FALLIBLE (VAR X))) ((HUMAN (VAR X))))
(LAMBDA (X) ((HUMAN (VAR X)) ((VAR X) TURING)) (((VAR X) TURING)))
(LAMBDA (X) ((HUMAN (VAR X)) ((VAR X) SOCRATES)) (((VAR X) SOCRATES)))
(LAMBDA (X) ((GREEK (VAR X)) ((VAR X) SOCRATES)) (((VAR X) SOCRATES)))
EOF
(VUOI SALVARLO ?-Y | NIL)
```

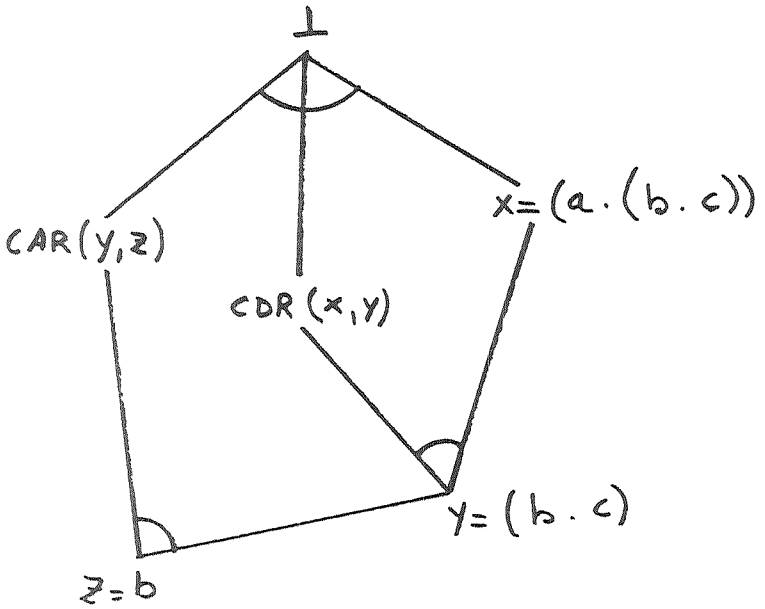

Fig. 3

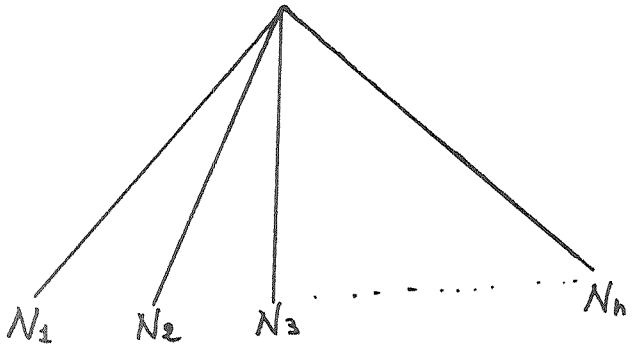

Fig. 4

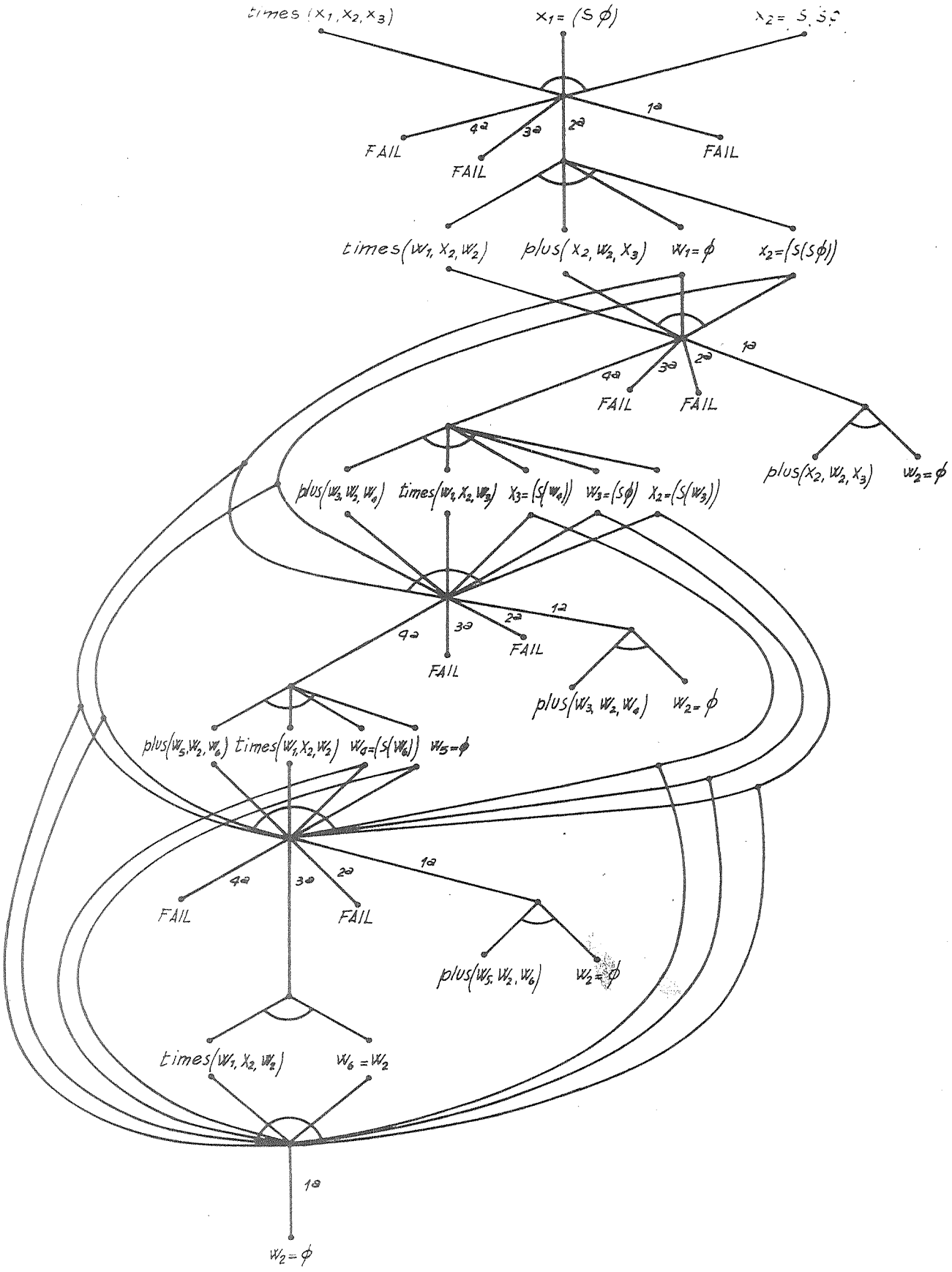
times (X1, X2, X3)
X1 = (S φ)
X2 = S S φ
1a
2a
3a
4a
FAIL
FAIL
FAIL
times(W1, X2, W2)
plus(X2, W2, X3)
W1 = φ
X2 = (S(S φ))
plus(W3, W2, W4)
times(W1, X2, W3)
X3 = (S(W4))
W3 = (S φ)
X2 = (S(W3))
plus(X2, W2, X3)
W2 = φ
plus(W5, W2, W6)
times(W1, X2, W4)
W4 = (S(W6))
W5 = φ
plus(W5, W2, W6)
W2 = φ
times(W1, X2, W2)
W6 = W2
W2 = φ

FIG. 5

BIBLIOGRAFIA

1) - Levi,G.,Sirovich,F., Generalized AND/OR Graphs and their Relation to Formal Grammars.I.E.I. - N.I. B73-15 Novembre '73.

2) - Degano, P., Una Classe di Schemi Paralleli Non Deterministici.I.E.I. - N.I. B74-39.

3) - McCarthy, J., LISP 1.5 Programmer's Manual. (Cambridge, Mass.? The M.I.T. Press).

4) - Montangero, C., Pacini, G.,Turini,F.; Magma Lisp: A "Machine Language" for Artificial Intelligence. Proc. IV Int.'l Joint Conf. on Artificial Intelligence. Tbilisi.(Settembre '75).

5) - Levi,G.,Sirovich,F., A problem Redustion Model for Non Indipendent Subproblems. Proc. IV Int'l Joint Conf. on Artificial Intelligence. Tbilisi. (Settembre '75)

6) - Kowalski,R-A, Predicate logic as programminglanguage. Information Proceding 1974- pag. 569-574.