AOTRED2.

Un programma basato sui nested array per la riduzione di alberi AND/OR.

Renzo Beltrame

'Rapporto interno C84-16

AOTRED2.
Un programma basato sui nested array per la riduzione di alberi AND/OR.

Renzo BELTRAME

CNUCE - Ist. del CNR
36, Via S. Maria
56100 PISA - (Italy)
tel. +39 (50) 593 237
telex 500 371

Doc. C84-16

Ottobre 1984

INDICE

LISTA DELLE ILLUSTRAZIONI

1.0 INTRODUZIONE

Il problema di semplificare alberi AND/OR ha, come e' noto, interesse sia nell'ambito della "faults tolerance", sia quale problema piu' generale di ridurre a forma normale una sottoinsieme delle possibili espressioni, anche complesse, del calcolo degli enunciati in logica.

In questa seconda accezione l'approccio qui descritto richiede che ci si limiti ad espressioni in cui non compaia l'operatore NOT. Il problema e' stato affrontato lasciandogli invece intera la sua generalita' con un altro approccio, utilizzando il meccanismo delle "classi" offerto dal linguaggio di programmazione SIMULA 67.

Per effettuare la riduzione di di espressioni AND/OR in passato era stato messo a punto un programma scritto in linguaggio APL, e quindi impiegando una schematizzazione matriciale del problema [R.2]. Si e' voluto riformularlo utilizzando l'estensione del linguaggio APL che prevede i nested array per valutare le semplificazioni possibili e, sopratutto, il prevedibile miglioramento delle prestazioni.

Infatti nella schematizzazione matriciale classica si ottenevano spesso matrici molto sparse, con bassi fattori di riempimento, e pertanto era inevitabile l'appesantimento dovuto alla necessita' di operare anche sugli 0 delle matrici.

Per problemi di ottimizzazione di flusso su reti alcuni primi risultati di impiego dei nested array al posto di array classici sono apparsi molto promettenti [R.5], di qui lo stimolo a trasformare l'approccio al problema, trasformazione che ha fornito risultati assai buoni.

Infatti e' notevolmente cresciuta, a parita' di condizioni, la mole dei problemi che possono venir trattati, ed i tempi di elaborazione si sono anch'essi ridotti. Questi effetti, poi, diventano piu' sensibili al crescere delle dimensioni del problema.

Un vantaggio accessorio, ma non per questo meno comodo, ci e' venuto dal continuare ad utilizzare un linguaggio di programmazione interpretato anziche' compilato. La cosa ci ha permesso di semplificare al massimo sia la generazione delle variabili che ricorrono nelle espressioni, sia l'interpretazione e l'esecuzione delle espressioni stesse. Infatti queste ultime si presentano formalmente identiche sia che se ne esegua la valutazione assegnando alle variabili valori presi nell'insieme {0,1}, sia che se ne effettui la semplificazione dando alle variabili significato simbolico. In entrambi i casi AND e OR designano funzioni diadiche, cioe' relazioni binarie.

2.0 SCHEMATIZZAZIONE DEL PROBLEMA IN TERMINI DI ARRAY

Dalla riduzione di una espressione del calcolo degli enunciati contenente solo gli operatori AND e OR si vuole ottenere una nuova espressione in cui al primo livello ritroviamo solo operatori OR e questi collegano fra loro espressioni contenenti solo operatori AND. Si vuole cioe' una riduzione a forma canonica congiunta/disgiunta.

Sotto forma di grammatica, in notazione BNF, si ha:

```
<reduct> ::= <expr1> OR <expr1> | <expr1> OR <reduct>;
<expr1> ::= <symb> AND <symb> | <symb> AND <expr1>;
```

dove <symb> e' un identificatore di variabile booleana.

2.1 SCHEMATIZZAZIONE IN TERMINI DI ARRAY CLASSICI

Detto U l'insieme dei simboli delle variabili che ricorrono nella riduzione in questione, la schematizzazione in termini di array classici puo' venir descritta nel modo seguente..

Ogni espressione del tipo <expr1> sia vista come un sottoinsieme Ei di U. Sia c(Ei) la sua funzione caratteristica.

Si rappresenti tale funzione caratteristica come un vettore di 0 e di 1 $\underline{v}$(Ei). Tale vettore avra' sempre per definizione lo stesso numero di elementi di U.

Su U, che contiene per definizione elementi tutti diversi e nei casi trattati anche in numero finito, e' possibile definire un ordinamento che ponga in corrispondenza biunivoca i suoi elementi con la successione 1,2,3,....,n.

Usando sempre lo stesso ordinamento per costruire il vettore $\underline{v}$(Ei), <reduct> puo' venir rappresentato come una matrice in cui ogni riga e' costituita dal vettore che rappresenta una particolare <expr1>.

Cosi' in un universo di variabili costituito da {a1,a2,a3,a4}, l'espressione:

a1 AND a2 AND a4

avra' la rappresentazione:

1 1 0 1

Le colonne, per quanto detto, rappresentano tutte le variabili booleane che intervengono nell'espressione da semplificare. Quindi la matrice, avendo tante colonne quante sono le variabili che intervengono nel sistema di espressioni da semplificare, sara' in generale una matrice molto sparsa.

Con questa schematizzazione l'operazione di OR ha come risultato una matrice ottenuta aggiungendo alle righe della matrice che rappresenta il termine di sinistra, le righe della matrice che rappresenta il termine di destra. Cioe':

exp1 OR exp2

avra' la rappresentazione:

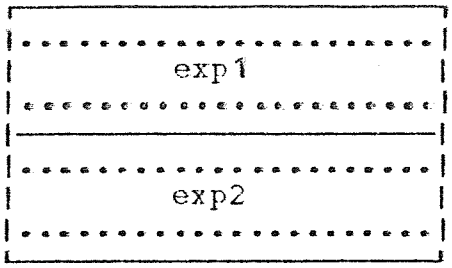


Fig. 1. Rappresentazione schematica di un OR

L'operazione di AND ha invece come risultato una matrice che, detto m1 il numero di righe della matrice che rappresenta il termine di sinistra ed m2 quello della matrice che rappresenta il termine di destra, avra' m1xm2 righe. Queste sono ottenute ripetendo m2 volte il blocco delle m1 righe della matrice che rappresenta il termine di sinistra, e ponendo in ogni blocco un 1 fisso nelle colonne in cui ricorre un 1 rispettivamente nella riga 1,2,...,m2 della matrice che rappresenta il termine di destra.

Supponendo che il termine di sinistra sia:

1 1 0 1
1 0 1 0

e che il termine di destra sia:

```
1 0 1 0
1 1 0 0
```

l'operazione di AND dara' come risultato:

```
1 1 1 1
1 0 1 0
1 1 0 1
1 1 1 0
```

Un semplice teorema (v. ad esempio [R.4]) che, quando f ed x siano gia' in forma canonica, stabilisce l'identita':

$$(f \text{ AND } x) \text{ OR } f \Longleftrightarrow f \qquad (2.1.1)$$

permette di eliminare le righe di cui un'altra sia sottoinsieme, semplificando cosi' la rappresentazione finale.

Nel caso sopra esemplificato otterremo:

```
1 0 1 0
1 1 0 1
```

Questa schematizzazione non e' in generale ottimale in senso assoluto, poiche', come si e' detto, porta ad avere matrici sparse con basso fattore di riempimento, e cio' e' tanto piu' probabile quanto piu' elevato e' il numero delle variabili coinvolte. Inoltre richiede una scansione delle espressioni di partenza per isolare le variabili che vi intervengono, stabilire l'universo U e la corrispondenza tra i suoi elementi e gli indici di colonna della matrice che rappresenta <reduct>. Essa e' quindi funzionale ad un linguaggio di programmazione che si basi sugli array classici come l'APL.

2.2 SCHEMATIZZAZIONE IN TERMINI DI NESTED ARRAY

In questa schematizzazione si puo' utilizzare una diversa rappresentazione di <reduct>. Precisamente <reduct> e' rappresentato da una matrice colonna, in cui ciascuna delle righe e' costituita da un solo elemento: un vettore "enclosed" che rappresenta a sua volta una <expr1>.

Ciascuno degli elementi di questo vettore e' a sua volta un vettore "enclosed", quest'ultimo costituito da una sequenza di caratteri: nel nostro caso il nome di una della variabile che intervengono in quella particolare <expr1>. Tale rappresentazione equivale a descrivere l'insieme $ci^{-1}(1)$, dove ci e' la funzione caratteristica dell'i-esima <expr1>.

Nel caso dell'espressione prima indicata:

a1 AND a2 AND a4

avremo ora la rappresentazione:

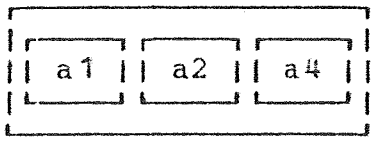


Fig. 2. Rappresentazione di una <expr1> come nested array

Cio' rende, come si vedra', estremamente semplice l'elaborazione, evitando, tra l'altro la noia di una tabella di simboli, di un analizzatore sintattico e di un esecutore delle espressioni non semplici, tutte cose che un linguaggio di programmazione interpretato consente tranquillamente di evitare mediante l'uso della funzione "execute".

Le operazioni AND e OR hanno invece una struttura estremamente simile a quella vista nel caso di una implementazione basata sugli array classici.

L'operazione di OR e' formalmente identica, poiche' il risultato e' una matrice ottenuta concatenando le righe della matrice che rappresenta il termine di sinistra a quelle della matrice che rappresenta il termine di destra (v. Fig. 1 pag. 3). Poi si eseguono le semplificazioni consentite dall'identita' (2.1.1) pag. 4.

L'operazione di AND si svolge ora in modo piu' vicino a quello di una manipolazione simbolica. Ha ancora come risultato una matrice che, detto m1 il numero di righe della matrice che rappresenta il termine di sinistra ed m2 quello della matrice che rappresenta il termine di destra, avra' m1xm2 righe. Queste sono ottenute ripetendo m2 volte il blocco delle m1 righe della matrice che rappresenta il termine di sinistra, e concatenando agli elementi di ciascuno dei vettori "enclosed" gli elementi del vettore "enclosed" che costituisce rispettivamente la riga 1,2,...,m2 della matrice che rappresenta il termine di destra.

A questo punto occorre eliminare in ogni riga all'interno del vettore "enclosed" i simboli ripetuti.

Ripetendo il caso esemplificato nella formulazione con array classici avremo per il termine di sinistra:

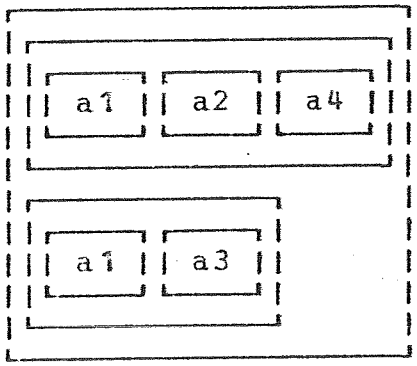
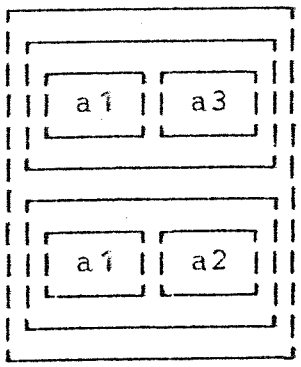


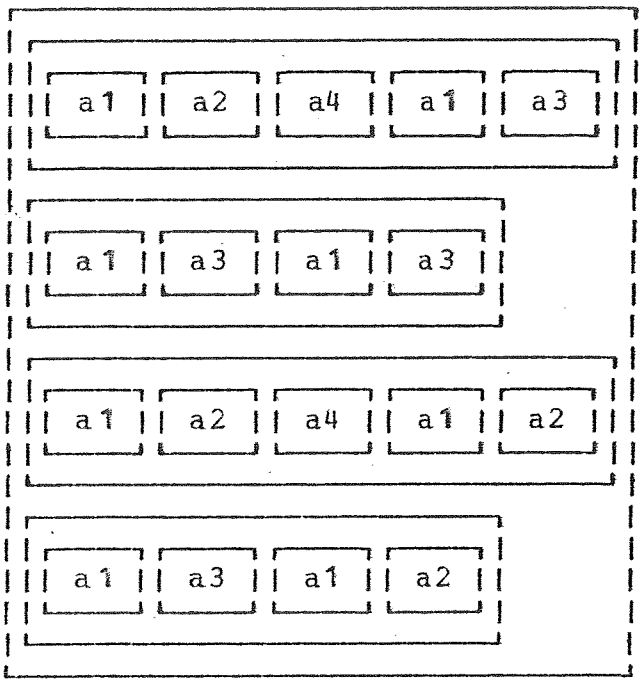
Fig. 3. Rappresentazione di una <reduct> come nested array

che illustra anche la rappresentazione di una <reduct> come general array.

Il termine di destra sia poi:



ottenendo formalmente come risultato intermedio:



vanno intanto eliminati tutti i termini ripetuti entro le varie <expr1>, ottenendo:

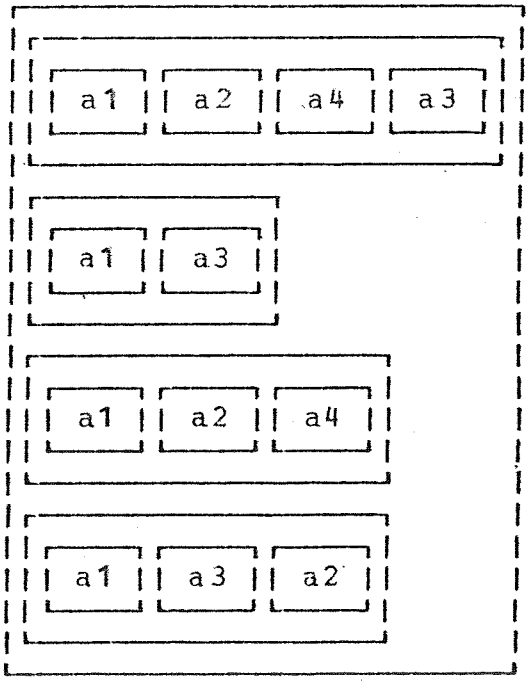
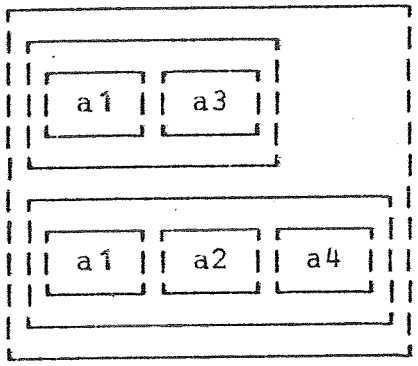


Fig. 4. Rappresentazione di un risultato intermedio

Bisogna poi bisogna applicare le semplificazioni consentite dall'identita' (2.1.1) pag. 4, eliminando in questo caso quei vettori "enclosed" che ne contengano un altro quale sottoinsieme.

Il risultato finale della struttura esemplificata in Fig. 4 pag. 7 e':



Nell'implementazione le semplificazioni sono eseguite in un modo un po' diverso da quello qui schematizzato, cosi' da avere migliori prestazioni nel caso di espressioni molto complesse. Va da se', infatti, che nel caso di espressioni complesse e' opportuno:

- sfruttare al massimo la proprieta' di due termini messi in OR di essere gia' semplificati ed usare come termine di confronto quello con minor numero di righe;
- utilizzare la possibilita' di eliminare la riga usata come termine di confronto quando se ne trovi un'altra che e' sottoinsieme di questa.

In tal modo si ottiene un miglioramento veramente notevole del tempo di elaborazione poiche' la semplificazione basata sull'identita' (2.1.1) pag. 4 e' l'operazione piu' critica da un punto di vista computazionale di tutto il processo di riduzione. Si spiega cosi' la stesura, abbastanza poco lineare, che hanno assunto le funzioni che effettuano tale semplificazione.

3.0 L'IMPLEMENTAZIONE IN APL2

L'implementazione che utilizza l'APL2, sviluppato dalla IBM [R.1], risente fortemente delle scelte fatte in questa sede, perche', come e' noto, la standardizzazione delle funzioni e degli operatori per manipolare i nested array e' ancora molto lontana dall'aver raggiunto il livello di consensi che funzioni ed operatori hanno in APL.

3.1 ESPRESSIONE DI INGRESSO

I dati di ingresso possono essere forniti in due modi.

Il primo modo e' ovviamente costituito da un'unica espressione, non importa quanto complessa, del tipo:

TOP = <espressione booleana in valida sintassi APL>

Le funzioni booleane debbono essere indicate con AND e OR, mentre per il resto - uso delle parentesi, ordine delle operazioni, etc. - valgono le usuali convenzioni dell'APL.

Il secondo modo, estremamente piu' comodo nel caso di una espressione complessa, consiste invece nel fornire una serie di espressioni del tipo:

TOP = <espressione booleana in valida sintassi APL>
<symb> = <espressione booleana in valida sintassi APL>
.....

dove a destra del segno "=" sta una espressione del tipo precedentemente descritto.

In questo secondo caso, pero', sostituendo ordinatamente a partire dal fondo ogni occorrenza dei simboli di sinistra con l'espressione di destra racchiusa tra parentesi, si deve sempre ottenere una espressione booleana valida nel linguaggio APL. La funzione PARSE (v. "3.2.3 La funzione SYMBDEF" pag. 12) esegue alcuni controlli sintattici sull'espressione stessa.

3.2 LE FUNZIONI BASE

Le funzioni base in APL2 sono molto semplici e verranno illustrate qui di seguito.

3.2.1 La funzione OR

La funzione OR ha la forma seguente:

```
Z←X OR Y
→(2=ρρY)/L3
Y←1 1ρ, Y
L3:→(2=ρρX)/L5
X←1 1ρ, X
L5:Z←X SUBS Y
```

Essa impiega strumentalmente la funzione SUBS che elimina le righe, ossia gli elementi delle disgiunzioni, che siano superset di altri. Come e' noto cio' e' possibile dal momento che ci troviamo gia' in forma canonica congiunta/disgiunta. La funzione SUBS ha la forma seguente:

```
Z←X SUBS Y;L L1 K K1 M N N1 I
X←X[⍋,⊃¨ρ¨X;]
Y←Y[⍋,⊃¨ρ¨Y;]
L3:L←0
→L8,(1↑ρX)≤1↑ρY
L1←X
X←Y
Y←L1
L8:→((L←1+L)>1↑ρX)/0,ρZ←X,[1]Y
K1←(L,1)⌷⍺[X
K←Y∈¨K1
N←,∨/¨K
→(0=,∨/N)/L8
M←,+/¨K
I←,⊃[1]N∧(M=ρ¨K1)∧(M≤,ρ¨Y)
→(0=∨/,I)/L18
Y←(~I)/[1]Y
→((~L1),L1←(1↑ρX)≤1↑ρY)/L3 L8
L18:I←,⊃[1]N∧(M<ρ¨K1)∧(M=,ρ¨Y)
→(0=∨/,I)/L8
X←(L≠⍳1↑ρX)/[1]X
→L8,ρL← 1+L
```

3.2.2 La funzione AND

La funzione AND ha la forma seguente:

```
Z←X AND Y;K K1 P
→(0≠1↑ρY←(2↑((ρY),1 1))ρY)/L3
→0,ρZ←X
L3:→(0≠1↑ρX←(2↑((ρX),1 1))ρX)/L5
→0,ρZ←Y
L5:→(1<1↑ρY)/L7
→0,ρZ←UNIQ X,[1],Y
L7:K1←(K←1)↑ρY
Z←X AND 1 1]α[Y
L9:→(K1<K←1+K)/0
P←X AND(K,1)]α[Y
→L9,ρZ←Z OR P
```

Essa impiega strumentalmente la funzione UNIQ che elimina all'interno di una stessa riga i doppioni dei simboli, ossia gli elementi delle congiunzioni che siano ripetizione di altri. Essa inoltre limita il numero degli elementi distinti al massimo prestabilito.

```
Z←UNIQ X
Z←(2↑(ρX),1)ρX
Z←(∩⌿Z)/⌿Z
Z←SUBS1 Z
Z←(∪GNCUT≥ρ⌿Z)/[1]Z
```

dove la funzione SUBS1 ha la forma seguente:

```
Z←SUBS1 X;L L1 K K1 M N N1 I
L←1↑0,ρX←X[4,⊃⍴⌿X;]
L2:→((L←1+L)≥M←1↑ρZ←X)/0
K1←(L,1)]α[X
L4:K←X∈⌿K1
N←,∧/⌿K
M←⌈/N/N1←,+/⌿K
→(L=I←(N×N1)⍳M)/L9
→L4,ρ((L,1)]α[X)←K1←(I,1)]α[X
L9:K←M=N1
(I⌿K)←0
→(0=+/K)/L2
X←(~K)/[1]X
→L2,ρL←⌈/0,L-+/L↑K
```

La funzione AND impiega strumentalmente, ove necessario, la funzione OR ed opera in questo caso recursivamente.

Le ultime due sono le funzioni che danno contenuto operativo ai simboli delle variabili ed eseguono la semplificazione delle espressioni.

3.2.3 La funzione SYMBDEF

La funzione SYMBDEF ha la forma seguente:

```
SYMBDEF X;H SY N M
X←X,'←','C','C',',',',''''',X,''''
X←'SY' CATF X
N←1↑ρX
M← 1↑ρX
H←0
L6:→(N<H←1+H)/L8
→L6,ρX[H;]←M↑(' '≠X[H;])/X[H;]
L8:H←⎕FX X
SY
```

Questa funzione assegna molto semplicemente al simbolo della variabile, preso eguale alla stringa di caratteri che la rappresenta nell'espressione di ingresso, appunto tale stringa di caratteri. Cio' avviene tramite una funzione, SY, locale alla SYMBDEF. L'operazione avviene dopo che un analizzatore sintattico ha isolato i nomi delle variabili. Per semplicita' si sono usate in questo caso le stesse funzioni APL messe a punto per il programma AOTRED. Esse hanno la forma seguente:

```
Z←PARSE X;X1 Z1 T TO CO
CO←1,Z←Z1←TO←⍳0
L2:X1←PAREL X
→(')(≠'=1↑X1)/L16,L18,L19
→(T←(∧/'AND'=3↑X1),(∧/'OR'=2↑X1),1)/L5,L5,L8
L5:→(~∧/TO=T←1↑T/1 0 0)/L10
→(1=CO)/L15
→L14,(CO←1),(TO←T),ρZ1←X1,' ',Z1
L8:→(0=CO)/L15
→L14,(CO←0),(ρZ1←X1,' ',Z1),ρGSYMB←GSYMB CATF X1
→(1=CO)/L15
Z←((T←'XYW',⍕GNR←1+GNR),'=',Z1)CATF Z
CO←1,Z1←TO←⍳0
⍴X,'←(⍞' '',X1,'' '',T),',X
L14:→(0≠ρ,⍴X)/L2
L15:Z←1 HALT 'ERROR COND:  ',⍞GX1
L16:Z←(PARSE X)CATF Z
→L2,(CO←0),ρZ1←'XYW',(⍕GNR),' ',Z1
L18:→0,ρZ←('XYW',(⍕GNR←1+GNR),'=',Z1)CATF Z
L19:Z←((PAREL X),X1,Z1)CATF Z
→L15×⍳∨/' '≠,⍴X
```

```
Z←PAREL X;T
Z←⍞(T←L/Z⍳' ')(≠')↑Z←,⍴X
Z←(-T←1⌈T+¯1×(1↑Z)∈')')(≠')↑Z
→(0≠ρZ←(' '≠Z)/Z)/0,ρ⍴X,'←T↓',X
Z←(∧/' '=,⍴X)HALT 'ERROR COND: ',⍞GX1
Z←PAREL X
```

Opzionalmente puo' venir eseguita la funzione EXPAND che provvede ad espandere una espressione logica in piu' espressioni contenenti o solo l'operatore booleano AND o solo l'operatore booleano OR. La funzione EXPAND ha la forma seguente:

```
Z←EXPAND X;GX1 GNR
Z←⍳GNR←0
L2:GX1←⍞' ' SEP1,X[1;]
→(0≠1↑ρX←1 0↓X)/L2,ρZ←Z CATF PARSE 'GX1'
```

Tale funzione si avvale della funzione SEP1 che ha la forma seguente:

3.2.4 La funzione ESEGUI

La funzione ESEGUI ha la forma seguente:

```
ESEGUI X;AX H
H←X⍳⍺_'=!
(H]⍺[X)←'←'
X←'AX' CATF⍵X
H←⎕FX X
AX
```

Essa provvede ad eseguire le semplificazioni dopo aver montato a partire dalle espressioni di ingresso una opportuna funzione. Questa, locale alla ESEGUI, e' stata chiamata AX.

3.3 RAPPRESENTAZIONE DELLA RIDUZIONE A FORMA CANONICA

La riduzione finale ha ovviamente la rappresentazione vista per un qualsiasi <reduct>, cioe' quella indicata in Fig. 3 pag. 6.

Il massimo numero di variabili che possono comparire congiunte in ciascun elemento della disgiunzione e' fissato dall'utente come dato di ingresso. E cio' perche' in problemi di "faults tolerance" si vuole di solito tener conto che il contemporaneo verificarsi di un numero crescente di eventi ha probabilita' sempre piu' bassa e, pertanto e' inutile superare un certo numero di eventi contemporanei.

Il risultato e' comunque conservato in una variabile globale di nome TOP.

4.0 UNA SEDUTA A TERMINALE

Si da' qui di seguito l'illustrazione di una seduta a terminale per il caso di una espressione composta, ma non molto complessa, di cui, volendo, si puo' verificare anche manualmente la correttezza del risultato.

Come si puo' vedere, lo spazio di lavoro e' stato dotato di una funzione che provvede a richiedere interattivamente all'utente le informazioni necessarie e gli fornisce poi, sempre interattivamente il risultato.

Non e' stata prevista una forma piu' sofisticata, perche' in APL2, come in APL, e' estremamente semplice interrompere la visualizzazione a terminale di una variabile senza interrompere l'interattivita'.

Nel caso illustrato la variabile da semplificare e' gia' contenuta in una variabile, PROVA1, e, pertanto, non deve essere fornita interattivamente. Per questo secondo caso sono state previste opportune funzioni che facilitano l'immissione interattiva.

E' poi possibile reiterare la semplificazione con parametri diversi, conservare la forma espansa dell'espressione da semplificare, e/o conservare i risultati intermedi della semplificazione, per analizzare, ad esempio, i vari passi del procedimento.

E' prevista anche una funzione di "reset" dello spazio di lavoro, capace di eliminare i risultati intermedi dell'ultima semplificazione eseguita. Essa si avvale della lista dei simboli delle variabili in gioco che e' stata usata durante la semplificazione. Di qui la limitazione all'ultima semplificazione eseguita.

```
      )LOAD AOTRED2
SAVED 19.10.10  9/27/84 (GMT)

      LREDUCTION

RESET OF WORKSPACE? (Y/N): Y
NEW CONDITIONS? (Y/N):  N
NAME OF VARIABLE CONDITIONS?:  PROVA1
MAX. ORDER OF CUT SET?:  5

  A F
  A G
  A C
  A D E


END OF SESSION? (Y/N):  Y
SAVE OF EXPANDED CONDITIONS? (Y/N):  N
SAVE OF INTERMEDIATE RESULTS? (Y/N):  N

      PROVA1
TOP=G1 AND (G2 OR G3) AND A
G1=A OR B OR C
G2=D AND E OR G3
G3=F OR G OR C

      )OFF HOLD
```

BIBLIOGRAFIA

R.1 - "APL2 - Program Description", IBM Manual SB21-2990, June 1982.

R.2 - R. Beltrame, "AOTRED - Un programma in APL per la riduzione di alberi AND-OR ricorrenti in problemi di faults-tolerance", Rapporto CNUCE, Giugno 1978.

R.3 - R. BELTRAME, "Un programma per la riduzione a forma normale di espressioni del calcolo degli enunciati in logica. Un approccio basato sulla programmazione per oggetti.", Rapporto CNUCE, (in preparazione).

R.4 - N.E. KOBRINSKI, B.A. TRAKHTENBROT, Automata Theory, North-Holland, Amsterdam, 1963.

R.5 - C. SANDI, "L'APL2 nell'analisi combinatoria", I Convegno Nazionale APL, C.N.R., Roma, 7-8 Novembre 1983.

Questo documento si compone di iv+18 pagine.

Stampato Mercoledi, 9 Gennaio 1985, - 1:14 p.m.