Sintesi logica: reti combinatorie

M. Favalli

Engineering Department in Ferrara

Sommario su reti a 2 livelli SOP

- Minimizzazione del numero di implicanti o letterali
- Problema NP-completo
  - calcolo degli implicanti primi
  - calcolo della copertura di costo minimo
- **Metodo di Quine-McCluskey**: generazione iterattiva degli implicanti primi e branch-and-bound nella tabella di copertura con il supporto di essenzialitá e dominanza
- **Espresso**: tecniche esatte con miglioramenti rilevanti rispetto a QM (generazione di implicanti primi tramite consenso, bound migliorati) ed euristici (che hanno il vantaggio di non dover considerare tutti gli implicanti primi)
- **Scherzo**: basato sui BDD e sui set combinatori, due ordini di grandezza piıú veloce di Espresso

Sintesi logica per reti combinatorie

- **RTL ⇒ livello logico**
  - costo, prestazioni, consumo di potenza, affidabilitá
- **Problematiche relative alla funzione**
  - caso generale
  - funzioni aritmetiche
- **Problematiche relative alla tecnologia obbiettivo**
  - custom, ASIC, FPGA, new nano-technologies

Scherzo

- Nella sintesi a due livelli si ha il problema di rappresentare insiemi di termini prodotto
- Un termine prodotto puó essere rappresentato come una configurazione su un insieme di $2n$ (n numero di ingressi) variabili binarie, una per ogni letterale
  - si consideri ad esempio $P = ab'e'$ come implicante di $F(a, b, c, d, e, f)$, allora tale termine prodotto é rappresentato da: $[a, a', b, b', c, c', d, d', e, e', f, f'] = 100100000100$
- Questo insieme é sparso sia per il ridotto numero di termini prodotto rispetto a quello totale, sia per il ridotto numero di uni
- Si potrebbe usare un BDD che peró avrebbe un numero elevato di variabili

Combinational sets

- **Combinational set** CS: utilizzato per rappresentare l'insieme degli implicanti
- Il CS ha una struttura simile a un BDD:
  - ogni nodo rappresenta una variabile di decisione associata a un letterale, dal nodo escono due archi, quello denotato con 0 corrisponde al caso in cui il letterale non compare in un termine prodotto e quello con 1 se compare
  - le foglie valgono 1 se i cammini che portano ad esse corrispondo a implicanti
  - La differenza con un BDD sta nelle regole di riduzione

Regole di riduzione

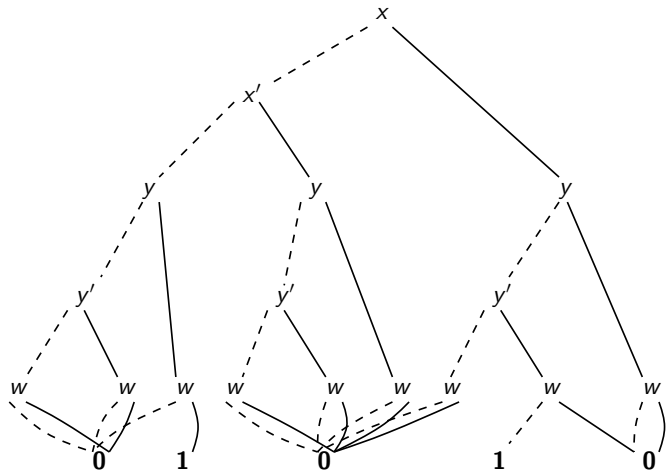
- isomorfismo: non si hanno sottografi uguali (comune ai BDD)
- vengono eliminati i nodi in cui l'arco denotato con 1 é connesso a una foglia con il valore 0 (zero suppressed BDD)
- si noti che i nodi con i due figli uguali non vengono eliminati come negli ROBDD
- questo aspetto é fondamentale per descrivere in maniera compatta l'insieme dei termini prodotto e sin generale insiemi di sottoinsiemi

Note

- un combinational set (definito in questo modo) non denota direttamente una funzione booleana
- puó essere associato a una funzione booleana se si assume che l'insieme di implicanti che descrive sia utilizzato in un espressione ***g*** di tipo SP

Esempio - I

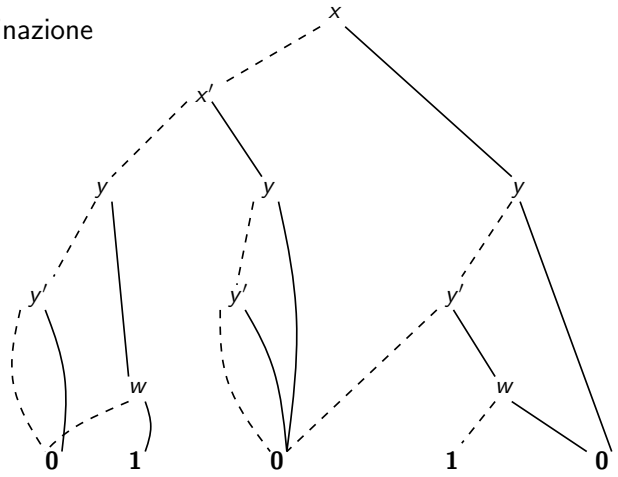
Esepressione: ***xy'*** + ***yw***



Esempio - I

Esepressione: ***xy′ + yw***

eliminazione



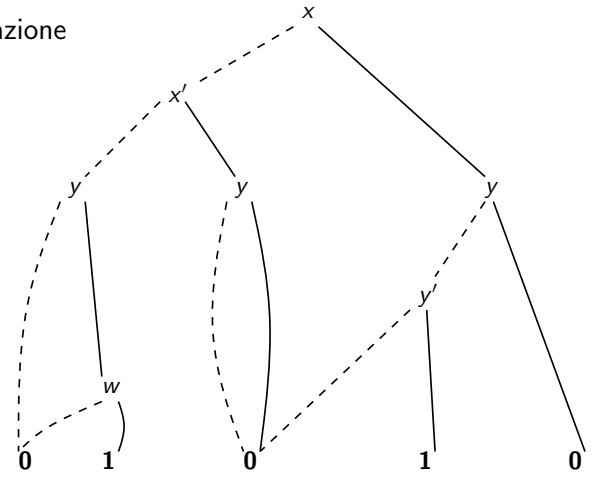
Sintesi logica

Introduzione
Reti a 2 livelli
Reti multilivello
Ottimizzazione technology independent
Don't cares
Risultati

Esempio - I

Esepressione: ***xy′ + yw***

eliminazione



8/68

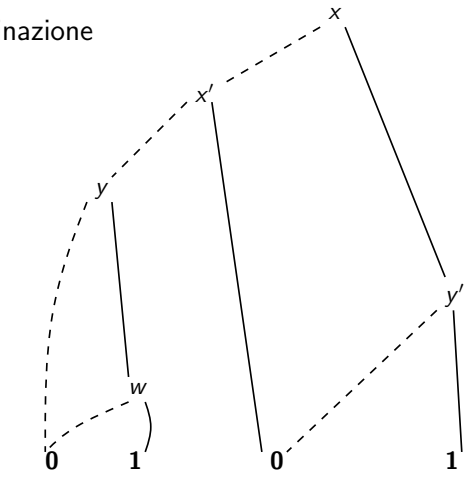
Sintesi logica

Introduzione
Reti a 2 livelli
Reti multilivello
Ottimizzazione technology independent
Don't cares
Risultati

Esempio - I

Esepressione: ***xy′ + yw***

eliminazione



8/68

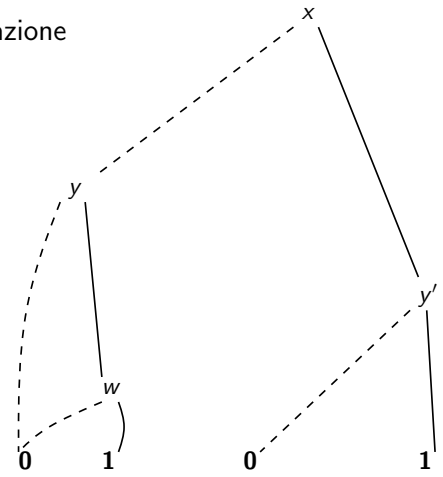
Sintesi logica

Introduzione
Reti a 2 livelli
Reti multilivello
Ottimizzazione technology independent
Don't cares
Risultati

Esempio - I

Esepressione: ***xy′ + yw***

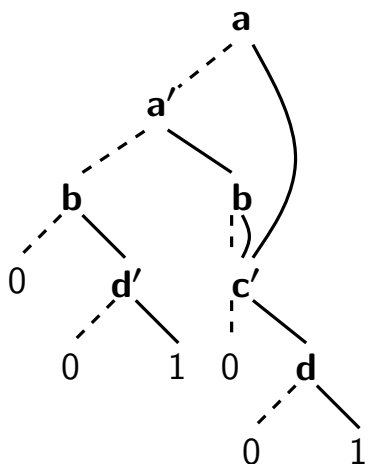
eliminazione



8/68

Esempio - II

set of implicants $\{\mathbf{bd'}, \mathbf{ac'd}, \mathbf{a'bc'd}\}$



Utilizzo dei combinational set per la sintesi di reti a due livelli

- Uno dei problemi nella sintesi ottima di reti a due livelli é dato dal numero di implicanti primi che in alcuni casi puó essere molto grande e non rappresentabile esplicitamente
- L'uso dei combinational set consente di rappresentarli in maniera implicita con tutti i vantaggi di manipolazione tipici dei BDD

Esercizi

Es. 1

- si consideri l'espressione SP $\boldsymbol{ab} + \boldsymbol{a'c} + \boldsymbol{b'c}$
- si costruisca il ROBDD per la funzione corrispondente
- si consideri poi l'espressione SP $\boldsymbol{ab} + \boldsymbol{c}$ e si costruisca il ROBDD per tale espressione
- si costruiscano poi i CS per le due espressioni in esame
- si faccia qualche considerazione sulla differenza fra le due strutture dati

Es. 2

- si determini se é possibile valutare la funzione corrispondente all'espressione SP descritta da un CS per una certa configurazione delle variabili di ingresso

Reti combinatorie multilivello

- Implementazioni piú compatte e con migliori prestazioni per la maggior parte delle funzioni Booleane
  - $\Rightarrow$ la maggior parte delle reti combinatorie nei componenti VLSI sono di questo tipo
- Piú gradi di libertá
  - $\Rightarrow$ maggiori dimensioni dello spazio di soluzioni da esplorare ai fini dell'ottimizzazione
  - $\Rightarrow$ utilizzo di euristici
- Sistemi a regole (IBM - 80s)
- Approcci sistematici (algoritmici)

Approcci algoritmici

- Technology independent optimization
  - metodi algebrici
  - metodi Booleani
- Technology dependent optimization
  - metodi basati su graph-covering
- Questa divisione costituisce di per se un approccio euristico: anche supponendo di disporre di metodologie ottimali in entrambi i casi, non si ha alcuna garanzia che la loro composizione dia luogo a una soluzione ottima

Metodologie technology independent

- Inizialmente vedremo aspetti relativi alla minimizzazione dell'area
  - numero di letterali come stima dell'area
- L'ottimizzazione avviene applicando alla rete una serie di trasformazioni in maniera iterattiva
  - un aspetto chiave nella riduzione del costo é l'utilizzo del fan-out tramite l'individuazione di sotto funzioni comuni
- Come primo esempio vedremo le seguenti trasformazioni:
- **Decomposizione** di una funzione Booleana: il processo di rappresentare la funzione tramite la composizione di un certo numero di funzioni piú semplici
- **Estrazione** da piú funzioni Booleane: il processo di estrarre nuove sottoespressioni comuni da un insieme di funzioni

Decomposizione

- Puó essere basata su diverse tecniche di tipo Booleano o algebrico
- In ambito Booleano abbiamo diversi tipi di decomposizioni basate su cofactoring:
  - Shannon expansion: $f(x_1, ..., x_i, ..., x_n) = x_i' f|_{x_i'} + x_i f|_{x_i}$
  - Davio positive expansion: $f = f|_{x_i'} \oplus x_i \frac{\partial f}{\partial x_i}$ dove $\frac{\partial f}{\partial x_i} = f|_{x_i'} \oplus f|_{x_i}$
  - Generalized cofactor: espansione rispetto a una funzione anziché rispetto a una variabile
  - ....
- Esempio: $f = ab + a'c'd' + ad'$

$$\begin{aligned} f &= a'x + ay \\ x &= c'd' \\ y &= b + d' \end{aligned}$$

- `decomp` in SIS

Utilizzo della decomposizione

Estrazione

- Non é detto che la decomposizione dei nodi di una Boolean network produca sottoespressioni comuni
- L'estrazione punta ad ottenere questo risultato per un insieme di funzioni (tipicamente due per motivi computazionali)
  - costruzione di nuovi nodi intermedi che possano essere utilizzati da piú funzioni
  - é un processo soggetto a ottimizzazione rispetto ad area, ritardo e consumo di potenza
- Esempio

$$f = acd + bcd + e \qquad f = xy + e$$
$$g = ae' + be' \qquad g = xe'$$
$$h = cde \qquad h = ye$$
$$x = a + b$$
$$y = cd$$

Strumenti utilizzati nell'ottimizzazione iterattiva di reti multilivello

- Fattorizzazione:
- Sostituzione ed eliminazione:
- Estrazione

Factoring

- La forma fattorizzata di un espressione SOP e POS

$$f = ac + ad + bc + bd + e = (a + b)(c + d) + e$$

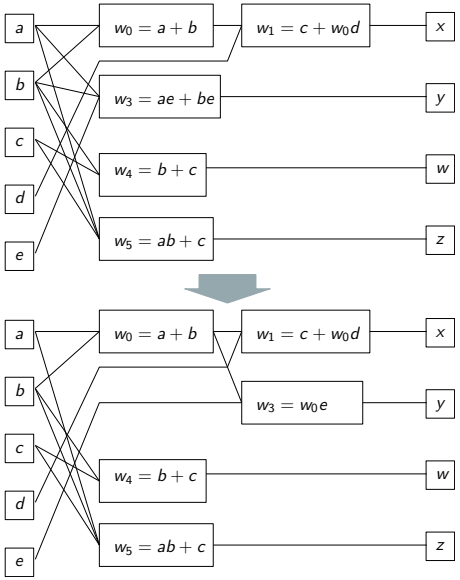
- Mentre il conteggio dei letterali in un espressione SOP predice il costo di un implementazione con PLA, quello di una forma fattorizzata riflette il costo di un gate CMOS complesso
- Sono piú compatte delle rappresentazioni SOP, ma mancano metodi di manipolazione Booleana
- Esistono numerose possibili fattorizzazioni per ogni espressione
  - non si possono considerare tutte
  - non si tratta quindi dello strumento adatto per l'individuazione di sottoespressioni comuni
- Criteri di ottimalitá e restrizioni

Sostituzione ed eliminazione

- La sostituzione di una funzione g in f consiste nell'esprimere f in funzione dei suoi ingressi e g
- Esempio: $f = ab + ac$, $g = b + c \Rightarrow f = ag$
- L'eliminazione consiste nell'eliminare una variabile in f sostituendola con la funzione associata ovunque essa compaia
- Esempio: $f = ab + x$, $g = cx$ e $x = y + w \Rightarrow f = ab + y + w$ e $g = cy + cw$
- Si tratta di un operazione che puó portare a una crescita esponenziale del numero di letterali
  - si associa un valore a ciascun nodo dato dalla differenza fra il numero di letterali della rete col nodo meno il numero di letterali nella rete senza nodo (7-8=-1 nell'esempio)
  - si elimina il nodo solo se tale differenza non é troppo alta

Esempio di applicazione della sostituzione `resub`

```
.model prova
.inputs a b c d e
.outputs x y w z
.names a b w0
1- 1
-1 1
.names c w0 d x
1-- 1
-11 1
.names a b e y
1-1 1
-11 1
.names b c w
1- 1
-1 1
.names a b c z
11- 1
--1 1
.end
```



Semplificazione

- Utilizzo di metodi di minimizzazione del costo per reti a due livelli per semplificare le espressioni dei nodi
- Semplificazione locale
- Semplificazione non locale: l'espressione viene semplificata tenendo conto dei vincoli di controllabilitá dati dai nodi nel TFI e TFO del nodo in esame
  - tali vincoli vengono espressi come don't care
- In SIS `simplify` e `full_simplify`

Metodologie euristiche

- Dal punto di vista dell'ottimizzazione la sostituzione e la semplificazione rappresentano qualcosa di simile a un gradiente decrescente in quanto minimizzano il costo
- L'eliminazione serve per uscire dai minimi locali
- Decomposizione ed estrazione giocano un ruolo molto rilevante perché evidenziano sottoespressioni comuni e verranno discusse in seguito
- Nella rete c'é un compromesso fra il numero di nodi e la loro dimensione: una rete con pochi nodi complessi da margini per la semplificazione, una con molti nodi semplici mette a disposizione piú sottoespressioni comuni

Esempio con SIS

- SIS mette a disposizione diversi comandi che possono essere utilizzati per realizzare queste trasformazioni
- Questi vengono tipicamente utilizzati in script di tipo euristico
- Come prima applicazione vedremo `script.rugged` applicato a una rete multilivello [De Micheli 1994]
- Questo script presenta buone caratteristiche di scalabilitá

script.rugged

- Prima di eseguire lo script si da il comando `set autoexec print_stats` che abilita la stampa delle statistiche dopo ogni comando
- da SIS una qualsiasi sequenza di comandi in un file (script) puó essere messa in esecuzione con `source -x`

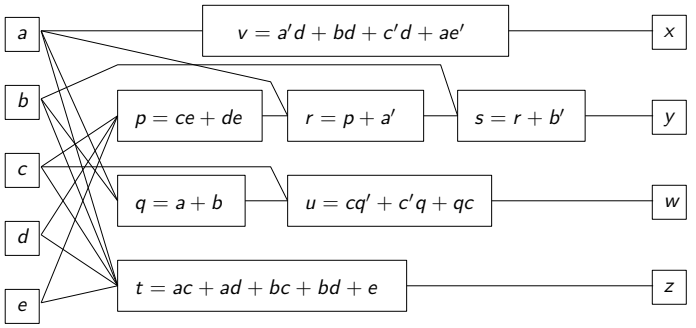
```
sweep; eliminate -1
simplify -m nocomp
eliminate -1

sweep; eliminate 5
simplify -m nocomp
resub -a

fx
resub -a; sweep

eliminate -1; sweep
full_simplify -m nocomp
```


Esempio di rete



Divisione e ricerca dei divisori comuni

- A meno di non compiere costose ricerche nello spazio delle possibili decomposizioni e fattorizzazioni gli strumenti visti non sono utilizzabili efficientemente
- La divisione é utile nella sostituzione in quanto fornisce un risultato indipendentemente dalla fattorizzazione
- La ricerca di divisori comuni é alla base dell'estrazione

Divisione e ricerca dei divisori comuni

- **Divisione** di f per p: $f = pq + r$, ove p é detto **divisore**, q **quoziente** e r **resto**
  - se $r = 0$, p é chiamata **fattore** di f

Proposizione 1

Una funzione p é un fattore Booleano di f se e solo se $fp' = 0$ ($f_{ON} \subseteq p_{ON}$)

Proposizione 2

Se $fp' \neq 0$, allora p é un divisore Booleano di f

Esempio

- Esempio di divisione Booleana
  - $f = a + bc$, $p = a + b$
  - p é un fattore Booleano di f perché $fp' = (a + bc)(a + b)' = (a + bc)a'b' = 0$
- Si noti che un espressione é un fattore Booleano di se stessa e lo é pure il valore 1
- La verifica di fattori e divisori puó essere fatta utilizzando i BDD

Calcolo del quoziente

- Sia f che g siano espresse come SOP e siano $f = \{b_1, b_2, ..., b_{|f|}\}$ e $g = \{a_1, a_2, ..., a_{|g|}\}$ i rispettivi insiemi di cubi
- Sia $h_i = \{c_{i,j} \mid a_i c_{i,j} \in f\}$ per ogni $i = 1, 2, ..., |g|$ e $j = 1, 2, ..., |f|$
- Si puó verificare che

$$f/g = \bigcap_{i=1}^{|g|} h_i = h_1 \cap h_2 \cap .... \cap h_{|g|}$$

- Esempio
  - $f = abc + abd + de$, $|f| = 3$
  - $g = ab + e$, $|g| = 2$
  - $c_{1,1} = c$, $c_{1,2} = d$ e $c_{2,3} = d$
  - $h_1 = \{c, d\}$ e $h_2 = \{d\} \Rightarrow f/g = d$
  - quindi $f = (ab + e)d + abc$
- L'algoritmo ha un costo quadratico $O(|f| \cdot |g|)$

Divisione algebrica

- Per un espressione ci sono diversi divisori e i risultati possono dipendere dalla rappresentazione
- La divisione algebrica é una restrizione della divisione che porta a risultati unici in maniera efficiente utilizzando descrizioni SOP
- Sia $sup(f)$ il supporto (semantico) di f, ovvero l'insieme di variabili da cui f dipende
- Due funzioni f e g sono ortogonali ($f \perp g$) se $sup(f) \cap sup(g) = \emptyset$

Definizione

La funzione g é un **divisore algebrico** di f se esitono h e r tali che: 1) $f = gh + r$, 2) $h \neq 0$, 3) $g \perp h$, 4) il resto é minimo (ovvero contiene il minimo numero di cubi)

Kernel e divisori algebrici

- Si tratta di trovare dei divisori efficaci da utilizzare nella sintesi
- Insieme dei divisori di f: $D(f) = \{g \mid f/g \neq 0\}$
- Insieme dei divisori primari di f: $P(f) = \{f/c \mid c \text{ é un cubo}\}$

Proposizione 3

Ogni divisore di f é contenuto in un divisore primario, ovvero se g divide f, allora $g \subseteq p \in P(f)$ (in termini Booleani $g \rightarrow p = 1$)

Esempio

Sia $f = ab + bc + ac + d$,
$P(f) = \{ab + bc + ac + d, a + c, b + c, a + b, 1\}$, il divisore $g = a$ é contenuto in $a + c$

Kernel e divisori algebrici

- Una funzione g é cube-free se il solo cubo che divide g con resto 0 é 1
- I **kernel** di f sono definiti come $K(f) = k \mid k \in P(f)$ é cube-free
- Per $k \in K(f)$ il suo **cokernel** é il cubo c tale che $f/c = k$
- Un kernel di un espressione é un quoziente cube-free dell'espressione divisa da un cubo chiamato cokernel $\Rightarrow$ un espressione cube-free é il cokernel di se stessa

Esempio

- $f = ace + bce + de + g$
- divisore a : $f/a = ce$ che é un cubo singolo e non é un kernel
- divisore e : $f/e = ac + bc + d$ che é cube free e quindi é un kernel
- divisore ce : $f/ce = a + b$ che anch'esso un kernel
- f stessa é un kernel (diviso da 1)

Kernel e divisori

Teorema [Brayton e Mc Mullen]

Due espressioni f e g hanno un divisore comune cube-free se e solo se esistono dei kernel $k_f \in K(f)$ e $k_g \in K(g)$ tali che $k_f \cap k_g$ ha due o piú termini (non é un cubo)

- Algoritmo per il calcolo dei kernel di f
  1. si rende l'espressione cube-free dividendola per il suo fattore piú grande
  2. si selezionano ordinatamente i letterali di f e la si divide per questi
  3. il risultato é un kernel se é cube-free, se non é lo si divide per il fattore piú grande
  4. si procede fino a quando non si trovano piú kernel

Esempio

- $\varphi = abcd + abce + abef$
- l'espressione non é cube-free e va divisa per $c_\varphi = ab \Rightarrow \gamma = cd + ce + ef$ (che é un kernel)
- si divide γ per c, d, e, f:
  - $\gamma/c = d + e$ che é un kernel
  - $\gamma/d = c$ che non é un kernel
  - $\gamma/e = c + f$ che é un kernel
  - $\gamma/f = e$ che non é un kernel
- Quindi: $K(f) = \{cd + ce + ef, d + e, c + f\}$

Utilizzo dei kernel nella extract

$\mu = \mathbf{ace + bce + de + f}$ $\qquad$ $\eta = \mathbf{ad + bd + cde + ef}$

kernels:
$\mathbf{k}_{\mu,0} = \mathbf{ace+bce+de+f}$
$\mathbf{k}_{\mu,1} = \mathbf{a + b}$
$\mathbf{k}_{\mu,2} = \mathbf{ac + bc + d}$

kernels:
$\mathbf{k}_{\eta,0} = \mathbf{ad+bd+cde+ef}$
$\mathbf{k}_{\eta,1} = \mathbf{a + b + ce}$
$\mathbf{k}_{\eta,2} = \mathbf{cd + f}$

l'unica intersezione non nulla é
$\mathbf{k}_{\mu,1} \cap \mathbf{k}_{\eta,1} = \mathbf{a + b = w}$

extract

$\mu = \mathbf{wce + de + f}$ $\qquad$ $\mathbf{w = a + b}$ $\qquad$ $\eta = \mathbf{wd + cde + ef}$

Tecniche booleane

- Le operazioni algebriche viste fino ad ora si basano sulla rappresentazione di funzioni come espressioni SP
- Nel caso booleano non esiste un operazione di divisione, ma solo una generalizzazione a qualsiasi operazione che date due funzioni ***f*** e ***p*** produce:

$$f \circ p = p \cdot q + r$$

- Tale definizione non é unica
  - si consideri $f = x'w' + yw + xw$ e $p = x' + w$
  - si puó verificare che
    $f = (x' + w)(x + w') + yw = (x' + w)(y + w') + xw$
  - si noti che queste espressioni non sono in alcun modo ottenibili per via algebrica

Fattorizzazione booleana

- Siano ***f*** e ***g*** due funzioni booleane tali che $f \to g$, allora ***f*** puó essere scritta come $f = gh$ dove $f \to h$ e $h \to f + g'$
- Si noti che $f \to g$ puó essere scritta come $f \leq g$ perché l'ONset di ***f*** é contenuto in quello di ***g***
- Quindi si ha $f \leq h \leq f + g'$
- ***g*** si definisce come **fattore Booleano** di ***f***

Fattorizzazione Booleana: esempio

- Siano $f = a'bc + abd + ac'd$ e $g = a' + d$ da cui $f + g' = ad' + ac' + bc$
- Per selezionare ***h*** guardiamo a una mappa di Karnaugh di $f + g'$ (in cui gli uni di ***f*** sono in grassetto)
- Chiaramente ***h*** deve contenere tutti gli uni in grassetto piú eventualmente qualcuno non in grassetto
- Un risultato possibile é $h = bc + ac'$
- Scegliendo il primo si ha $f = (a' + d)(bc + ac')$

ab \ ***cd***	00	01	11	10
00	0	0	0	0
01	0	0	**1**	**1**
11	1	**1**	**1**	1
10	1	**1**	0	1

Divisione Booleana

- Se invece di avere $f \to g$ si ha $fg \neq 0$, f puó essere riscritta come $f = gh + r$ con $r \to f \ \wedge f \neq r$ ovvero $r < f$
- In questo caso, $fg' \to r$ e quindi $fg' \leq r < f$
- Dati r e g, h deve soddisfare la seguente relazione: $(fr' \to h) \ \wedge (h \to f + g')$ (ovvero, $fr' \leq h \leq f + g'$)
- g viene definito un **divisore Booleano** di f

Divisione Booleana: esempio

- Siano $f = a + bc + d$ e $g = a + b$
- Per selezionare r guardiamo a una mappa di Karnaugh di f (in cui gli uni di fg' sono in grassetto)
- Chiaramente r deve contenere tutti gli uni in grassetto piú eventualmente qualcuno non in grassetto
- Il risultato piú sensato é $r = d$

	cd			
ab	00	01	11	10
00	0	**1**	**1**	0
01	0	1	1	1
11	1	1	1	1
10	1	1	1	1

Divisione Booleana: esempio

- Ora rimane da scegliere h di modo che $fd' \leq h \leq f + (a + b)'$ ovvero $ad' + bcd' \leq h \leq a + bc + d + a'b'$
- La mappa di Karnaugh mostra l'estremo superiore (uni) e quello inferiore (uni in grassetto per h)
- h deve contenere tutti gli uni in grassetto ed eventualmente qualcuno degli altri
- La scelta ottimale é $h = a + c$ e quindi $f = (a + c)(a + b) + d$

	cd			
ab	00	01	11	10
00	1	1	1	1
01	0	1	1	**1**
11	**1**	1	1	**1**
10	**1**	1	1	**1**

Operazioni Booleane

- Come si puó capire, i gradi di libertá per questo tipo di operazioni sono moltissimi
- In circuiti piccoli si possono ottenere risultati notevoli
- Tali tecniche risultano peró troppo onerose computazionalmente in circuiti di grandi dimensioni

Satisfiability e observability don't cares

- Utilizzati per migliorare la simplify: ovvero per ottimizzare la funzione di singoli nodi senza modificare la topologia della rete
- Sintesi ottima di reti a due livelli: la presenza di condizioni di indifferenza sugli ingressi puó consentire di ridurre il costo dell'espressione di uscita.
- Nel caso di reti multilivello, le condizioni di indifferenza si possono produrre all'interno della rete anche se non sono presenti sugli ingressi.
- Se $\boldsymbol{X}$ é l'insieme degli ingressi e $\boldsymbol{Y}$ quello dei segnali intermedi esistono in teoria $\mathbf{2}^{|\boldsymbol{X}|+|\boldsymbol{Y}|}$ possibili configurazioni di cui solo $\mathbf{2}^{|\boldsymbol{X}|}$ sono veramente possibili $\Rightarrow$ gradi di libertá

Idea

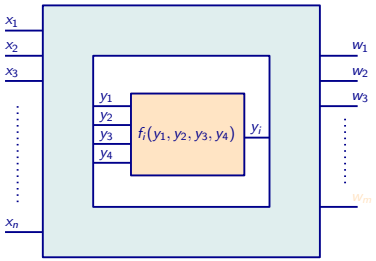
Dipendentemente dalla struttura del circuito é possibile che si presentino le seguenti condizioni:

1. La controllabilitá degl ingressi non é completa: alcune configurazioni delle variabili del supporto locale non si possono presentare
2. L'osservabilitá dell'uscita non é completa: per alcune configurazioni delle variabili del supporto locale, l'uscita del blocco considerato non é osservabile (le uscite non dipendono da tale segnale)

Conseguenza

In entrambi i casi si creano delle condizioni di indifferenza nella funzione $\boldsymbol{f}$ che possono essere utilizzate per semplificarla.

Esempio



Si consideri un blocco logico all'interno di una rete combinatoria

- il blocco realizza una funzione $\boldsymbol{y_i = f_i(y_1, y_2, y_3, y_4)}$ delle variabili del supporto locale
- le variabili del supporto locale dipendono dagli ingressi del circuito
- anche la propagazione di $\boldsymbol{y_i}$ alle uscite é condizionata al resto della rete
- é possibile che oltre a $\boldsymbol{f_i}$ siano permesse altre funzioni senza cambiare la funzione della rete

Satisfiability Don't cares (SDC)

- Il SDC set rappresenta l'insieme di configurazioni del supporto locale che non si possono presentare.
- Il SDC set puó essere calcolato a partire dalla funzione caratteristica della relazione Booleana che descrive la consistenza della rete
  - sia $\boldsymbol{y_i}$ l'uscita di un nodo $\boldsymbol{i}$ e $\boldsymbol{Y_i}$ il suo supporto locale la sua consistenza é data da $\boldsymbol{y_i = f_i(X, Y_i)}$
  - quindi per tutta la rete $\boldsymbol{\bigwedge_{\forall i}(y_i = f_i(X, Y_i))}$
  - negando tale condizione e applicando De Morgan abbiamo le configurazioni che non possono mai capitare $\boldsymbol{SDC(X, Y) = \bigvee_{\forall i}(y_i \neq f_i(X, Y_i))}$
- Tale espressione puó essere calcolata solo per il TFI di $\boldsymbol{i}$ (meno $\boldsymbol{i}$ stesso)

Esempio

- Rete di partenza: $y_1 = x_1x_2$, $y_2 = x_2 + x_3$ e $y_3 = y_1 \oplus y_2 = x_1x_2' + x_1'x_2$
- $SDC(X, Y) = (y_1 \neq (x_1x_2)) + (y_2 \neq (x_2 + x_3)) + (y_3 \neq (y_1 \oplus y_2))$
- Limitando la relazione al TFI del nodo 3 si ha: $SDC_{y_3} = (y_1 \neq (x_1x_2)) + (y_2 \neq (x_2 + x_3))$
- Quantificando universalmente rispetto agli ingressi primari si ha: $\forall x_1, x_2, x_3\ (y_1 \neq (x_1x_2)) + (y_2 \neq (x_2 + x_3)) = y_1y_2'$
- Sfruttando tale DC per l'espressione di y_3, si ha $y_3 = y_1'y_2$
- Tali operazioni possono essere eseguite utilizzando i BDD

y_1 \ y_2	0	1
0	0	1
1	1	0

y_1 \ y_2	0	1
0	0	1
1	-	0

Calcoli dell'esempio

Sia $\varphi = SDC_{y_3}$

$$\begin{aligned}\varphi &= (y_1 \neq (x_1x_2)) + (y_2 \neq (x_2 + x_3)) \\ &= y_1(x_1x_2)' + y_1'(x_1x_2) + y_2(x_2 + x_3)' + y_2'(x_2 + x_x3) \\ &= y_1x_1' + y_1x_2' + y_1'x_1x_2 + y_2'x_2 + y_2'x_3 + y_2x_2'x_3\end{aligned}$$

dove
$\forall x_1, x_2, x_3\ \varphi = \varphi|_{000} \cdot \varphi|_{001} \cdot \varphi|_{010} \cdot \varphi|_{011} \cdot \varphi|_{100} \cdot \varphi|_{101} \cdot \varphi|_{110} \cdot \varphi|_{111}$
In cui $\varphi|_{000} = \varphi|_{100} = y_1 + y_2$, $\varphi|_{001} = 1$,
$\varphi|_{010} = \varphi|_{011} = \varphi|_{101} = y_1 + y_2'$, $\varphi|_{110} = \varphi|_{111} = y_1' + y_2'$
Da cui $\forall x_1, x_2, x_3\ \varphi = y_1y_2'$

Observability Don't Cares (ODC)

- L'ODC rappresenta l'insieme di configurazioni del supporto locale per cui l'uscita di un nodo non é osservabile ad alcun PO
- Per calcolarli si considera y_i come un PI, e si calcola la funzione gloable $g_j(X, y_i)$ di ciascun PO
- y_i é osservabile all'uscita j se é vera:

$$\frac{\partial g_j}{\partial y_i} = g_j(X, y_i = 0) \neq g_j(X, y_i = 1)$$

- Quindi y_i non é osservabile ad alcuna uscita se:

$$ODC_i = \bigwedge_{\forall j \in PO} \left(\frac{\partial g_j}{\partial y_i}\right)'$$

Esempio

- Sia data la rete: $y_1 = b'c$, $y_2 = ab + a'c$ e $y_3 = y_1 + y_2 + d$
- Si vuole semplificare y_2
- La funzione globale di y_3 é $g_3(a, b, c, d, y_2) = b'c + y_2 + d$
- Da cui:

$$ODC_3 = \left(\frac{\partial g_3}{\partial y_2}\right)' = ((b'c + 0 + d) \oplus 1)' = b'c + d$$

- In questo semplice caso l'ODC puó essere usato direttamente (perché gli ingressi del nodo da semplificare coincidono con i PI), dando luogo a $y_2 = ab + c$

a \ bc	00	01	11	10
0	0	1	1	0
1	0	0	1	1

a \ bc	00	01	11	10
0	0	-	1	0
1	0	-	1	1

DC set complessivo

- Ci sono due problemi di carattere pratico
- L'ODC é funzione dei PI e quindi puó non essere utilizzabile direttamente
- A SDC e ODC si deve aggiungere l'insieme XDC dei DC esterni per ciascuna funzione di uscita

Esempio

- Sia $y_1 = a \oplus c$, $y_2 = ab'$, $y_3 = a + b$, $y_4 = (y_2 \oplus y_3) + c$ e $y_5 = y_1 y_4$
- Si vuole semplificare y_4, quindi esprimo y_5 in funzione di y_4 da cui $y_5 = (a \oplus c) y_4$ da cui

$$DC_i(a,b,c)' = ODC_i(a,b,c)' = \frac{\partial (a \oplus c) y_4}{\partial y_4} = (a \oplus c) \oplus 0 = (a \oplus c)$$

- Il DC set per y_4 risulta

$$D_4(y_2, y_3) = (\exists a, b, c\, (y_2 = ab')(y_3 = a + b)(a \oplus c))' = y_2 y_3'$$

- Situazione per y_4 rappresentata su mappe senza e con DC

c \ y_2y_3	00	01	11	10
0	0	1	0	1
1	1	1	1	1

c \ y_2y_3	00	01	11	10
0	0	1	0	-
1	1	1	1	-

- Quindi: $y_4 = y_2' y_3 + c$

DC set complessivo

- Si definisce don't care set globale di un nodo i $DC_i(X) = \left(\wedge_{\forall j \in PO} XDC_j(X)\right) \vee ODC_i(X)$ (non tiene conto del SDC)
- Sia Y_i il supporto locale di y_i, l'insieme D_i di DC locale per i é

$$D_i(Y_i) = \left(\exists X \bigwedge_{y_j \in Y_i} (y_j = g_j(X)) \wedge DC_i(X)' \right)'$$

- In pratica, si calcola il care set per il supporto locale e lo si interseca con il complemento del don't care set di i, poi si quantifica esistenzialmente (proiezione di X su Y_i) e si nega
- In teoria ci si potrebbe aspettare di sommare SDC_i a D_i, ma questa operazione é inutile in quanto l'operazione é giá compresa nella formula precedente
- Anche in questo caso si utilizzano i BDD

Problemi e prospettive

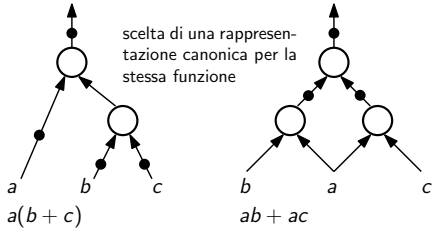
- Si tratta di tecniche molto potenti dal punto di vista della semplificazione
- Hanno tutti i problemi relativi all'uso di ROBDD
- Possibili soluzioni
  - calcolo dei DC set basandosi su una parte della rete
  - utilizzo di tecniche approssimate che calcolano un sottoinsieme del DC set

Ottimizzazione di reti combinatorie tramite AIG

- Si sono viste diverse tecniche utili dal punto di vista dell'ottimizzazione
- Dobbiamo descriverne di nuove per la sintesi di reti descritte con AIG ?
- No, perché in realtá un AIG puó essere visto come un formalismo che descrive diverse Boolean networks
- L'idea base é quella del clustering: i nodi dell'AIG vengono ragruppati in sottoreti con uscita e con come ingressi un insieme di segnali detto cut
- Queste reti corrispondono in qualche modo ai nodi di una Boolean network
- Essendo possibili diversi clustering, l'AIG corrisponde a diverse Boolean network alle quali possono essere applicate alcune delle tecniche viste oltre a tecniche specifiche per AIG

Hashing strutturale

- Introduce una canonicitá parziale della rappresentazione in cui risultano canoniche alcune sottoreti
- Hashing strutturale a un livello, tutte le volte in cui si aggiunge un AND, si verifica se esiste un nodo con lo stesso fan-in
- Hashing strutturale a due livelli
  - in una fase preliminare si analizzano tutte i possibili AIG a due livelli e per ogni funzione implementabile in tale modo si sceglie una solo rappresentazione
  - quando si costruisce l'AIG e si aggiunge un AND si costruisce la forma canonica a due livelli avete tale gate come radice, questo puó cambiare i nodi di fan-in

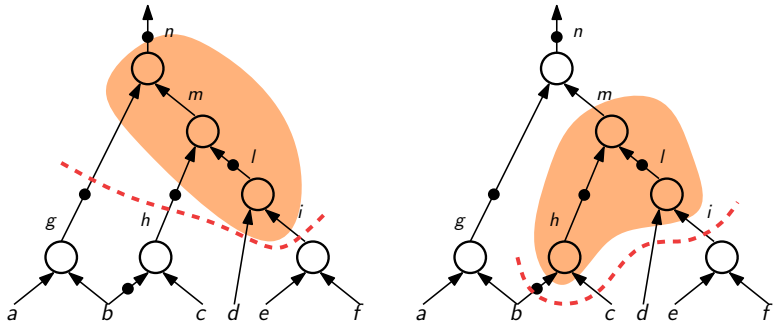


Costruzione dell'AIG

- A partire da una boolean network si parte dall'espressione fattorizzata dei nodi e si utilizza De Morgan per la somma logica
- Il numero di nodi nell'AIG é miore o uguale al numero di letterali nella rete fattorizzata
- A partire da un BDD si sostituisce ogni nodo del BDD con un MPX a due ingressi dati descritto come AIG
- Il numero di nodi dell'AIG é minore o uguale al numero di nodi del BDD per tre

AIG e clustering

Due esempi di 4-cut fattibili e dei i relativi clustering

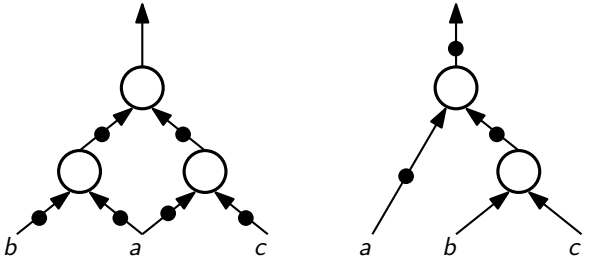


Riduzione funzionale (rewrite)

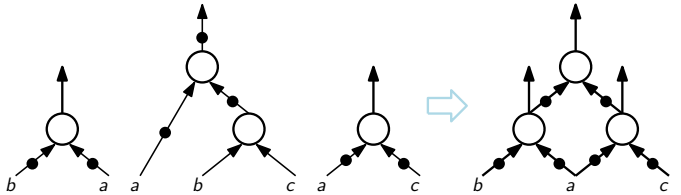
- Per ogni nodo della rete individua un insieme di tagli fattibili (4-cut)
- Per ciascun taglio calcola la funzione (una fra $\mathbf{2^{2^4}}$) e la memorizza in una parola di 16 bit
- Per ciascuna funzione cerca un sotto grafo AIG piú compatto all'interno di una hash-table memorizzata
  - la ricerca avviene all'interno di 222 classi di funzioni equivalenti per trasformazioni NPN (negazione degli ingressi, permutazione degli ingressi, negazione dell'uscita)
- Si tiene traccia anche della possibile presenza di nodi giá presenti nella rete per una risostituzione
- Per realizzare queste operazioni possono essere utilizzate simulazione, BDD e SAT

Esempi

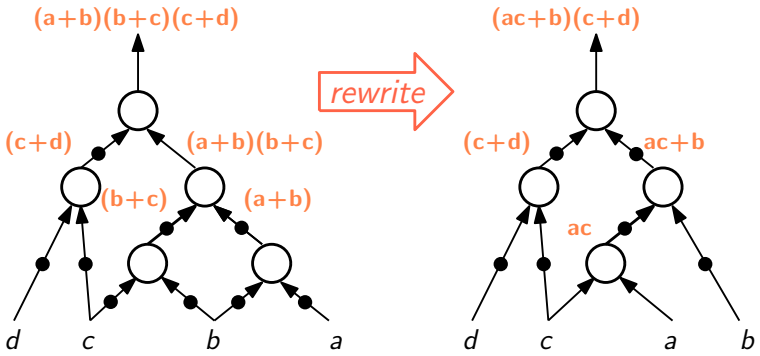
Possibile sotto grafo della rete e AIG ad esso equivalente ($(a+b)(a+c)$ e $a+bc$)



Rewrite in presenza di nodi giá esistenti nella rete



Esempio



Altre operazioni su AIG

- **refactor**: un sottografo viene trasformato di nuovo in un espressione SP e rifattorizzzato con una strategia diversa
- **balance**: un sottografo che ha su un ramo molti nodi puó essere bilanciato per poi avere un minor ritardo nella rete risultante
- **riduzione funzionale**
  - gli AIG non sono forme canoniche e quindi possono essere presenti nodi che realizzano la stessa funzione come radice di sottografi diversi
  - si puó usare la boolean satisfiability per verificare se due nodi sono equivalenti
  - oppure si puó costruire i grafo in modo semi-canonico (FRAIG)

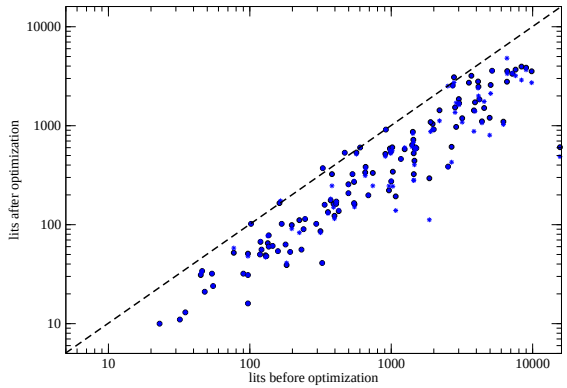
Sintesi con AIG

- I vantaggi della sintesi con AIG sono dati da:
  - compattezza della rappresentazione in memoria;
  - risultati equivalenti a SIS con tempi di calcolo decisamente inferiori;
  - rappresentazione della rete piú adatta a agli algoritmi di technology mapping che intervengono nella fase successiva;

SIS e `script.rugged`

- Sia n_0 il numero di letterali prima dell'applicazione di `script.rugged` e n_1 e n_2 il numero di lettarali dopo la sua applicazione per una o due volte
- Guadagno medio su tutti i benchmark $avg(n_1/n_0) = 0.48$ e $avg(n_2/n_0) = 0.46$ mostrando che l'applicazione dell'euristico per due volte da luogo a vantaggi marginali

simplified script.rugged optimization results



Risultati con SIS e ABC

Risultati ottenuti utilizzando SIS e ABC per ottimizzare una serie di benchmark combinatori tratti dal set LGSynth93

- analisi dei risultati ottenuti con `script.rugged` e `script.algebraic`
- risultati technology independent e dopo il technology mapping

SIS e `script.algebraic`

- `script.algebraic` utilizza `gcx` e `gkx` in maniera iterattiva
- Guadagno medio su tutti i benchmark **0.49**

simplified script.rugged vs. script.algebraic optimization results