Sintesi logica: strumenti

M. Favalli

Engineering Department in Ferrara

Strumenti per la manipolazione di funzioni Booleane

- Introduzione
- Algebra di Boole
- Forme normali
- BDD

Gli strumenti che verranno descritti in questo ambito trovano un impiego piú generale della semplice sintesi di reti combinatorie, sia riguardo alla EDA che riguardo alla CS

Sintesi logica

- **Produzione automatica di componenti logici a partire da una descrizione al livello RTL**
  - analisi
  - strumenti per la manipolazione e la trasformazione di reti e funzioni
  - ottimizzazione
- Queste fasi devono essere supportate da opportuni modelli matematici
- Reti combinatorie

Sintesi RTL → logic level

- Modello matematico: **algebra di Boole** (logica e algebra)
- Modello del tempo: **sincrono**
- Target tecnologico: **ASIC** o FPGA

Rappresentazione delle funzioni (strutture dati)

- SOP e POS
- BDD
- AIG
- Boolean networks

Tecniche di Boolean reasoning (package)

- BDD (librerie)
- SAT (solvers)
- AIG (tools)

Algebra di Boole - I

- Variabile Booleana $x \in \mathbb{B} = \{0, 1\}$
- Letterale: variabile Booleana (x) o il suo complemento (x')
- Spazio Booleano a n dimensioni: $\mathbb{B}^n = \mathbb{B} \times .... \times \mathbb{B}$
  - mintermine: un vertice $a \in \mathbb{B}^n$ (possibile candidato a un assegnamento a 1 della funzione)
- Funzione Booleana completamente specificata: $f : \mathbb{B}^n \rightarrow \mathbb{B}$
- Funzione Booleana a piú uscite: $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$

Algebra di Boole - II

- Funzione Booleana non completamente specificata: $f : \mathbb{B}^n \rightarrow \{0, 1, \text{-}\}$
  - **ON-set**: $f_{ON} = \{a \in \mathbb{B}^n : f(a) = 1\}$
  - **OFF-set**: $f_{OFF} = \{a \in \mathbb{B}^n : f(a) = 0\}$
  - **DC-set**: $f_{DC} = \{a \in \mathbb{B}^n : f(a) = \text{-}\}$
- Se $h = f \wedge g \Rightarrow h_{ON} = f_{ON} \wedge g_{ON}$, $h_{OFF} = f_{OFF} \vee g_{OFF}$ e $h_{DC} = \neg(h_{ON} \vee h_{OFF})$
- Funzione Booleana non completamente specificata a piú uscite: $f : \mathbb{B}^n \rightarrow \{0, 1, \text{-}\}^m$

Formula Booleana

- $\varphi := 0 \mid 1 \mid x \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \leftrightarrow \varphi_2$
- precedenza (ordine crescente) $\leftrightarrow \rightarrow \vee \wedge \neg$
- alle volte si utilizzeranno $+, \cdot, '$ per motivi compattezza
- qui si dará per scontata la conoscenza delle principali proprietá di tali operatori

Funzioni $\Leftrightarrow$ Formule

- A ogni espressione φ corrisponde una e una sola funzione f
- A ogni funzione f corrisponde un numero non finito di espressioni $\Rightarrow$ **spazio di ricerca per la sintesi logica**
- Potere espressivo di un insieme di operatori $\Rightarrow$ insiemi funzionalmente completi
  - $\neg, \wedge, \vee$ (algebra di commutazione)
  - $\neg, \vee$ (NOR logic), $\neg, \wedge$ (NAND logic)
  - $\neg, \rightarrow$
  - $\wedge, \leftrightarrow$ oppure $\vee, \leftrightarrow$

Teorema di espansione di Shannon

- Data una funzione $f(x_1, ..., x_i, ..., x_n)$ si definiscono
  - cofattore negativo: $f|_{x_i=0} = f|_{x_i'} = f(x_1, ..., 0, ..., x_n)$
  - cofattore positivo: $f|_{x_i=1} = f|_{x_i} = f(x_1, ..., 1, ..., x_n)$
- Il teorema di espansione consente di esprimere una funzione utilizzando i suoi cofattori:

$$f = x_i' f|_{x_i=0} + x_i f|_{x_i=1}$$

Formule Booleane Quantificate (QBF) - I

- Quantificatore esistenziale: $\exists$
- Quantificatore universale: $\forall$
- Esempio: $\varphi = \forall x_1, x_2 \, \exists \, x_3, x_4 \; f(x_1, x_2, x_3, x_4, x_5, x_6)$
  - x_1, x_2, x_3, x_4 sono variabili di tipo bound
  - x_5, x_6 sono variabili di tipo free
- Ogni formula QBF puó essere trasformata in una formula libera da quantificatori mediante le seguenti proprietá
  $\forall x \; f(x, y) = f(0, y) \wedge f(1, y) \qquad \exists x \; f(x, y) = f(0, y) \vee f(1, y)$
- La regola va applicata dall'interno
- Si noti che $\forall x \; \exists y \neq \exists y \forall x$

Esempio - I

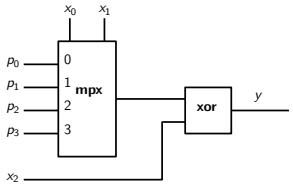
$\forall x \; \exists y \; f(x, y, w)$

$$\Rightarrow \; \forall x \; f(x, 0, w) \vee \forall x \; f(y, 1, w)$$

$$\Rightarrow \; (f(0, 0, w) \wedge f(1, 0, w)) \vee (f(0, 1, w) \wedge f(1, 1, w))$$

Esempio - II

- Ipotetica cella programmabile di FPGA, siano $p_{0..3}$ i bit di programmazione e $x_{0..2}$ le variabili di ingresso
- É possibile realizzare la funzione definita da $x_0 + x_1 x_2$ mediante tale cella?
- L'uscita della cella é:
  $y = (p_0 x_0' x_1' + p_1 x_0 x_1' + p_2 x_0' x_1 + p_3 x_0 x_1) \oplus x_2$



- Si vuole imporre il vincolo dato dall'espressione QBF:

$$\exists p_0, p_1, p_2, p_3 \forall x_0, x_1, x_2 \; , \; y \leftrightarrow x_0 + x_1 x_2$$

- In pratica, si vuole verificare se esiste una configurazione delle variabili di programmazione tale che l'uscita della cella sia equivalente all'espressione da realizzare

Esempio - II

- Questo risulta nella formula quantizzata

$$\varphi = \exists p_0, p_1, p_2, p_3 \forall x_0, x_1, x_2 \ , \ f(p_0, p_1, p_2, p_3, x_0, x_1, x_2)$$

- ove $f(p_0, p_1, p_2, p_3, x_0, x_1, x_2) = x_0{}' x_1{}' (p_0 x_2 + p_0{}' x_2{}') + x_0 x_1 (p_3 x_2{}' + p_3{}' x_2) + x_0 x_1{}' (p_1 x_2{}' + p_1{}' x_2) + p_2{}' x_0{}' x_1$
- Un solver per espressioni QBF mostra che la formula é sempre falsa e quindi non é possibile sintetizzare tale funzione
- La cosa puó essere verificata usando le proprietá introdotte precedentemente

$$\begin{aligned} \varphi = \quad & \exists p_0, p_1, p_2, p_3 \ , \ f(p_0, p_1, p_2, p_3, 0, 0, 0) \wedge \\ & \exists p_0, p_1, p_2, p_3 \ , \ f(p_0, p_1, p_2, p_3, 1, 0, 0) \wedge \\ & \ldots\ldots \\ & \exists p_0, p_1, p_2, p_3 \ , \ f(p_0, p_1, p_2, p_3, 1, 1, 1) \end{aligned}$$

- Ma $f(p_0, p_1, p_2, p_3, 0, 0, 0) = p_0{}'$ e $f(p_0, p_1, p_2, p_3, 1, 0, 0) = p_0$, per cui si capisce immediatamente come φ sia falsa

Rappresentazione di funzioni e reti Booleane

- Il dominio di una funzione Booleana ha dimensioni esponenziali nel numero di ingressi
- Questo problema rimane in ogni tipo di rappresentazione
  - se rappresentiamo direttamente la funzione si hanno problemi di memoria
  - se rappresentiamo la funzione tramite una rete si hanno problemi computazionali nella manipolazione di tale modello
- É evidente come il modo di rappresentare queste funzioni sia critico per sintesi, collaudo e verifica al livello logico

Solvers

- Per determinare se un espressione quantificata é vera o falsa si puó usare la trasformazione in una formula priva di quantificatori e usare un SAT solver
  - la trasformazione é peró evidentemente esponenziale nel numero di variabili bounded
  - non viene fornita alcuna informazione sulle variabili quantificate esistenzialmente
- Un differente approccio é invece basato sulla costruzione di BDD
- Piú recentemente sono stati sviluppati solver SAT per formule QBF

Forme normali SOP o POS

- Le consideriamo dal punto di vista della manipolazione di funzioni Booleane
  - Prodotto: il prodotto di due espressioni SOP puó essere fatto moltiplicando tutti i termini prodotto della prima per tutti quelli della seconda $\Rightarrow$ costo quadratico
  - Somma: costo lineare
  - Complementazione: si utilizza De Morgan ottenendo un espressione POS e si con la proprietá distributiva si calcola poi un espressione SOP
    - Il costo é esponenziale in quanto il prodotto di due termini S da luogo a un numero di termini prodotto pari al prodotto del loro numero di letterali
    - $\varphi = ab + cd + ef \Rightarrow$ $\varphi' = (ab + cd + ef)' = (ab)'(cd)'(ef)' = (a' + b')(c' + d')(e' + f') = (a'c' + a'd' + b'c' + b'd')(e' + f') = (a'c'e' + a'c'f' + a'd'e' + a'd'f' + b'c'e' + b'c'f' + b'd'e' + b'd'f')$

Binary Decision Diagrams

- Uno dei principali strumenti per la manipolazione di funzioni Booleane
- Data una funzione $f(x_1, ..., x_n)$, l'applicazioni iterattiva del teorema di espansione di Shannon da luogo a un albero binario (BDD) con un insieme di nodi V
  - nodi terminali o foglie v con un valore $value(v) \in \{0, 1\}$
  - nodi non-terminali con: un indice $index(v) \in \{1, ..., n\}$ e due figli:

    0-child : $low(v) \in V$

    1-child : $high(v) \in V$
  - se $index(v) = i$, x_i é la variabile di decisione di v
- Un nodo si dice radice se non ha genitori

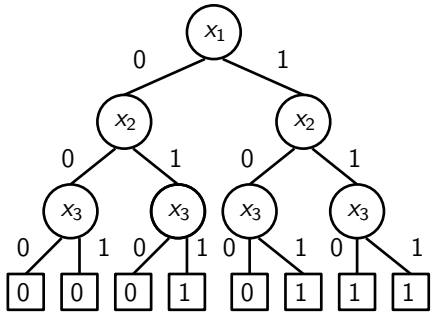
Funzione associata a un BDD

- Ogni nodo di un BDD corrisponde a una funzione Booleana
- Definizione induttiva:
  1. nodo é terminale
     - if $value(v) = 1$ then $f_v = 1$
     - if $value(v) = 0$ then $f_v = 0$
  2. nodo non-terminale con $index(v) = i$

     $f_v = x_i' f[low(v)](x_1, ..., x_n) + x_i f[high(v)](x_1, ..., x_n)$

BDD ordinato (OBDD)

- Un BDD si dice ordinato se per ogni cammino dalla radice alle foglie gli indici (le variabili) compaiono sempre nello stesso ordine
- Fissato un orientamento delle variabili un OBDD é una forma canonica



x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	1	1

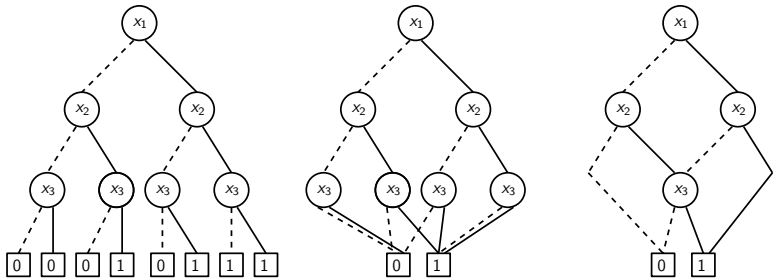
Isomorfismo

- Due OBDD D_1 e D_2 con lo stesso ordinamento delle variabili sono isomorfi se esiste una funzione iniettiva π dai nodi di D_1 a quelli di D_2 tale che:
  - $\forall v \in V(D_1)$ if $\pi(v) = w$ $(w \in V(D_2))$, allora o:
    1. v e w sono foglie con lo stesso valore ($value(v) = value(w)$) o
    2. v e w sono nodi non-terminali con lo stesso indice ($index(v) = index(w)$) e $\pi(low(v)) = low(w)$ e $\pi(high(v)) = high(w)$
- La verifica di isomorfismo puó essere fatta partendo dal nodo radice in un tempo lineare nel numero di nodi

Reduced Ordered BDD (ROBDD)

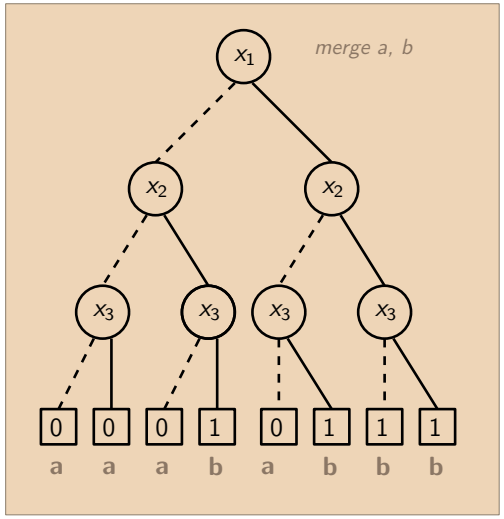
- Un OBDD D é ridotto se
  1. non contiene alcun nodo v con $high(v) = low(v)$ e
  2. non contiene alcuna coppia di nodi v, w tali che i sottografi aventi v e w come radici sono isomorfi
- Un ROBDD puó essere costruito in maniera iterattiva a partire da un OBDD mediante le seguenti regole
  1. due foglie con lo stesso valore sono fuse in unico nodo
  2. due nodi non terminali u e v con la stessa variabile di decisione, lo stesso 0-child ($low(u) = low(v)$) e lo stesso 1-child ($high(u) = high(v)$) sono fusi
  3. un nodo non-terminale v con $low(v) = high(v)$ viene rimosso ridirigendo l'arco che parte dal suo genitore verso il suo figlio

Esempio



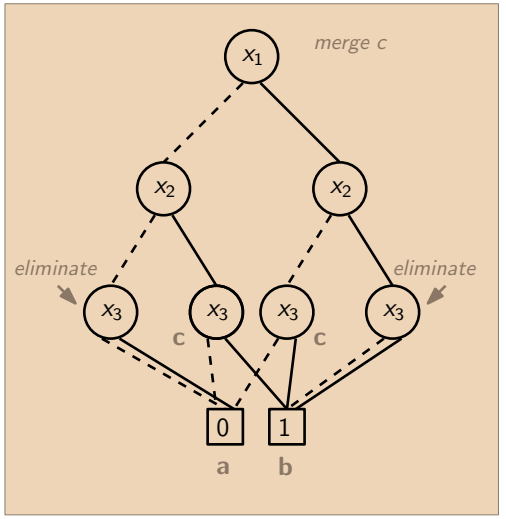
Algoritmo di riduzione

- L'algoritmo attraversa l'albero dall'alto (foglie, quindi basso in figura) assegnando un etichetta a ciascun nodo diverso dai precedenti che incontra



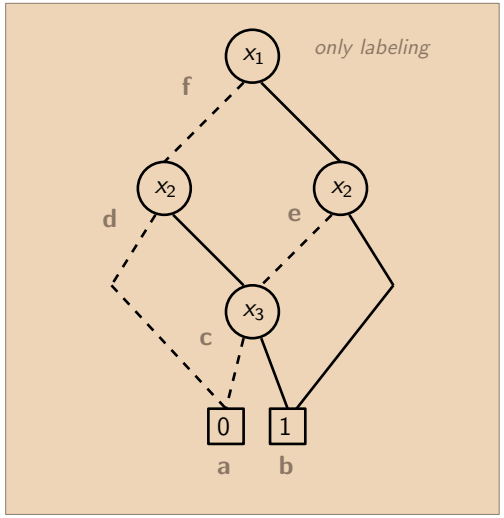
Algoritmo di riduzione

- L'algoritmo attraversa l'albero dall'alto (foglie, quindi basso in figura) assegnando un etichetta a ciascun nodo diverso dai precedenti che incontra



Algoritmo di riduzione

- L'algoritmo attraversa l'albero dall'alto (foglie, quindi basso in figura) assegnando un etichetta a ciascun nodo diverso dai precedenti che incontra



Complessitá dell'algoritmo di riduzione

- L'algoritmo ha una complessitá pari a $O(|V| log(|V|))$
- Peró in worst case $|V|$ é esponenziale nel numero di variabili di ingresso
- Un possibile approccio basato su costruzione di OBDD e successiva riduzione a ROBDD non é pratico perché l'OBDD tenderá ad occupare troppa memoria
- La soluzione consiste nel costruire direttamente il ROBDD

ROBDD come forme canoniche

- Per ogni funzione Booleana f e ordinamento delle variabili di decisione, esiste un unico ROBDD corrispondente a f e qualsiasi altro OBDD contiene piú nodi
- Di conseguenza il problema di verificare se due funzioni f e g sono equivalenti si riduce a verificare se i due grafi sono isomorfi
- Come applicazione vedremo il **combinational equivalence checking**: date due reti combinatorie C_1 e C_2 definite sullo stesso supporto, si vuole determinare se sono equivalenti
- L'operazione é possibile se si costruiscono i corrispondenti ROBDD

ROBDD come strumento per Boolean Reasoning

- I BDD rendono molto semplici alcune operazioni di manipolazione Booleana sulla funzione
  - calcolo di $\neg f$: si tratta semplicemente di scambiare di valore le foglie (tempo costante)
  - calcolo dei cofattori rispetto a una variabile x_i: si attraversa il BDD determinando tutti i nodi u il cui 0-child o 1-child ha indice i ($v : index(v) = i$) e si sostituisce tale nodo (v) con $high(v)$ ($f|_{x_i=1}$) o $low(v)$ ($f|_{x_i=0}$)
- La realizzazione di $f \wedge g$ o $f \vee g$ é piú interessante

Operatore `BddApply`

- Siano D_1 e D_2 due ROBDD definiti sullo stesso supporto e con lo stesso ordinamento di variabili corrispondenti a due funzioni f e g
- Si vuole calcolare il ROBDD $D_1 \langle op \rangle D_2$ corrispondente alla funzione $f \langle op \rangle g$
- Proprietá (teorema di espansione di Shannon):

$$f\langle op\rangle g = x_i'(f|_{x_i'}\langle op\rangle g|_{x_i'}) + x_i(f|_{x_i}\langle op\rangle g|_{x_i})$$

- Per due generici nodi $v \in V(D_1)$ e $w \in V(D_2)$ dei ROBDD

$$f[v]\langle op\rangle g[w] = x_i'(f[low(v)]\langle op\rangle g[low(w)]) + x_i(f[high(v)]\langle op\rangle g[high(w)]) \quad (1)$$

- Dove $f[v]$ é la funzione associata al nodo v

Casi da considerare

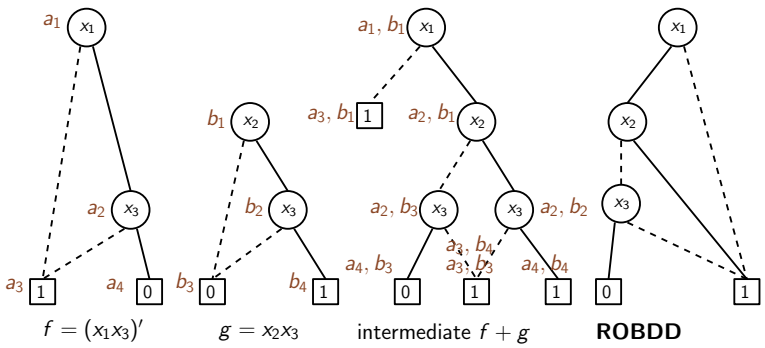
1. v e w sono foglie: si genera un nuovo nodo u con valore $value(u) = value(v)\langle op\rangle value(w)$
2. se $index(v) = index(w) = i$, si crea un nuovo nodo u con indice i e poi si applica recursivamente Eq. 1 su $low(v)$ e $low(w)$ per gerare $low(u)$ e su $high(v)$ e $high(w)$ per generare $high(u)$
3. se $index(v) = i$, ma $index(w) > i$ si crea un nuovo nodo u con indice i e si applica recursivamente Eq. 1 su $low(v)$ e w per generare $low(u)$ e su $high(v)$ e w per generare $high(u)$
4. se $index(v) > i$ e $index(w) = i$ si crea un nuovo nodo u con indice i e si applica recursivamente Eq. 1 su v e $low(w)$ per generare $low(u)$ e su v e $high(w)$ per generare $high(u)$

L'indice della radice é quello piú basso.

Algoritmo

- L'algoritmo attraversa i grafi di f e g a partire dalle loro radici
- Complessitá esponenziale: ogni chiamata puó generare 2 chiamate recursive
- Miglioramenti:
  - se uno dei due nodi é una foglia, si possono applicare direttamente le regole dell'algebra di Boole (es. $f \wedge 1 = f$)
  - per evitare di valutare la stessa coppia di nodi piú di due volte, si utilizza una hash table indirizzata da (v, w, u)
  - prima di applicare l'algoritmo a una nuova coppia di nodi viene verificato che non sia giá stata valutata
- La complessitá diventa $O(|V(D_1)| \cdot |V(D_2)|)$ (non si dimentichi che queste quantitá possono essere esponenziali)

Esempio



Esempio: passi svolti dall'algoritmo

step	v	w	u	rule	$low(u)$	$high(u)$
1	a1	b1	a1b1	3	$f[low(a1)] + g[b1] =$ $= f[a3] + g[b1] = 1$ inserisce la foglia 1 ($a3b1$)	$f[high(a1)] + g[b1] =$ $= f[a2] + g[b1]$ itera
2	a2	b1	a2b1	4	$f[a2] + g[low(b1)] =$ $= f[a2] + g[b3] = f[a2]$ itera	$f[a2] + g[high(b1)] =$ $= f[a2] + g[b2]$ itera
3	a2	b3	a2b3	3	$f[low(a2)] + g[b3] =$ $= f[a3] + g[b3] = 1 + 0$ inserisce la foglia 1 ($a3b3$)	$f[high(a2)] + g[b3] =$ $= f[a4] + g[b3] = 0$ inserisce la foglia 0 ($a4b3$
4	a2	b2	a2b2	2	$f[low(a2)] + g[low(b2)] = f[high(a2)] + f[high(b2)]$ $= f[a3] + f[b3] = 1 + 0$ inserisce la foglia 1 ($a3b3$)	$= f[a4] + f[b4] = 0 + 1$ inserisce la foglia 1 ($a4b4$

Operatore ITE

- É una generalizzazione di `BddApply`
- $\texttt{ITE}(f, g, h) = fg + f'h$ dove f, g e h sono funzioni definite sullo stesso supporto
- L'operatore puó essere applicato recursivamente a partire dal livello piú basso

$$\texttt{ITE}(f, g, h) = x_i' \texttt{ITE}(f|_{x_i'}, g|_{x_i'}, h|_{x_i'}) + x_i \texttt{ITE}(f|_{x_i}, g|_{x_i}, h|_{x_i})$$

- Dimostrazione:
  - $(fg)|_x = f|_x g|_x$ e $(f + g)|_x = f|_x + g|_x$
  - quindi

$$\begin{aligned}
&\texttt{ITE}(f, g, h) = fg + f'h = x(fg + f'h)|_x + x'(fg + f'h)|_{x'} = \\
&x((fg)|_x + (f'h)|_x) + x'((fg)|_{x'} + (f'h)|_{x'}) = \\
&x(f|_x g|_x + f'|_x h|_x) + x'(f|_{x'} g|_{x'} + f'|_{x'} h|_{x'}) = \\
&\qquad x\texttt{ITE}(f|_x, g|_x, h|_x) + x'\texttt{ITE}(f|_{x'}, g|_{x'}, h|_{x'})
\end{aligned}$$

ITE come operatore universale

Ogni funzione con uno o due operandi puó essere realizzata tramite la `ITE` selezionando opportunamente i suoi argomenti

espressione	forma equivalente	espressione	forma equivalente
0	0	$(f + g)'$	$\texttt{ITE}(f, 0, g')$
1	1	$(f \oplus g)'$	$\texttt{ITE}(f, g, g')$
$f \cdot g$	$\texttt{ITE}(f, g, 0)$	g'	$\texttt{ITE}(g, 0, 1)$
$f \cdot g'$	$\texttt{ITE}(f, g', 0)$	$f + g'$	$\texttt{ITE}(f, 1, g')$
f	f	f'	$\texttt{ITE}(f, 0, 1)$
g	g	$f' + g$	$\texttt{ITE}(f, g, 1)$
$f' \cdot g$	$\texttt{ITE}(f, 0, g)$	$(f.g)'$	$\texttt{ITE}(f, g', 1)$
$f \oplus g$	$\texttt{ITE}(f, g', g)$	$f + g$	$\texttt{ITE}(f, 1, g)$

Accorgimenti

Nella realizzazione degli algoritmi `BddApply` o `ITE`, vengono utilizzati alcuni accorgimenti che non cambiano la complessitá computazionale, ma che rendono fattibile l'utilizzo di ROBDD in molti casi di interesse pratico

- Utilizzo di hash table: struttura dati che memorizza un oggetto in una locazione identificata da una chiave calcolabile a partire dall'oggetto stesso

  unique table una locazione per ogni nodo identificata da una chiave data da (i, g, h) dove i é l'indice della variabile, e g e h sono dei puntatori ai figli di tale nodo

  computed table una locazione per ogni (f, g, h) (o $(f, g, \langle op \rangle)$) a ogni livello di recursione si verifica se i parametri di `AplyBdd` o `ITE` esistono nella tabella

Utilizzo di unique table

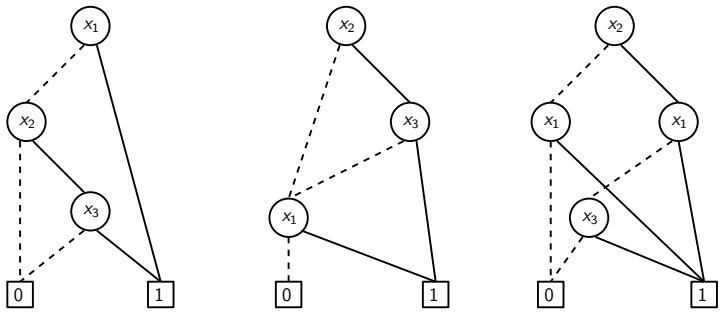
- la funzione di hashing mappa la chiave su una locazione
  - si prendono i, g e h e si shiftano di diversi numeri di bit, si somma il risultato e si calcola i resto della divisione per le dimensioni della table
  - non é una funzione iniettiva e si usano collision chain
- Ogni volta che si deve decidere se inserire un nuovo nodo nel BDD:
  1. si calcolano recursivamente g e h
  2. si calcola la chiave e si verifica se giá esiste nella unique table (o nella collision chain associata a tale chiave)
  3. se non esiste lo si calcola e lo si aggiunge
  4. altrimenti si aggiornano solo dei puntatori

Algoritmo `ITE`

```
ITE(f,g,h) {
  (result,terminal)=TERMINAL_CASE(f,g,h);
  if (terminal)
    return result;
  (result,is_in_comp_table)=COMP_TABLE_HAS_ENTRY(f,g,h);
  if (is_in_comp_table)
    return result;
  i=TOP_VAR(f,g,h);
  t=ITE(high(f),high(g),high(h));
  e=ITE(low(f),low(g),low(h));
  if (t=e)
    return t;     /* equal childs */
  r=FIND_OR_ADD_UNIQUE_TABLE(i,t,e);
  return r;
}
```


Variable ordering

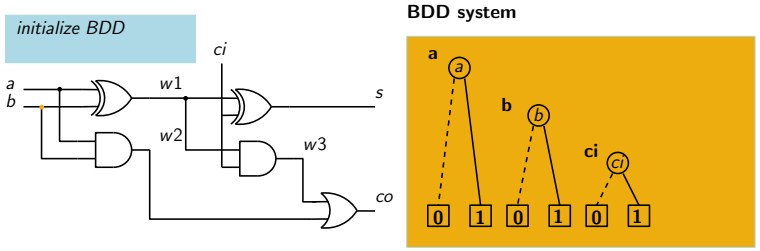
- Il numero di nodi in un ROBDD dipende dall'ordinamento delle variabili
- Utilizzo di euristici
- Riordinamento dinamico
- Esempio: $f = x_1 + x_2 x_3$



ordering x_1, x_2, x_3 ordering x_2, x_3, x_1 ordering x_2, x_1, x_3

Calcolo del BDD di una rete combinatoria

- Si inizializza il BDD package creando le variabili di decisione
- Poi si attraversa la rete eventualmente livellizzata calcolando il ROBDD di ciascun segnale

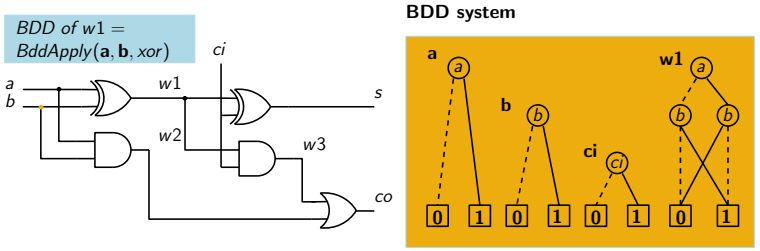


Sintesi logica

Introduzione
Boolean algebra
Rappresentazioni
BDD
Boolean networks
AIG

Calcolo del BDD di una rete combinatoria

- Si inizializza il BDD package creando le variabili di decisione
- Poi si attraversa la rete eventualmente livellizzata calcolando il ROBDD di ciascun segnale

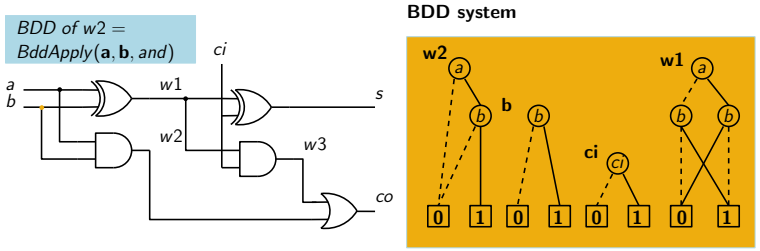


Sintesi logica

Introduzione
Boolean algebra
Rappresentazioni
BDD
Boolean networks
AIG

Calcolo del BDD di una rete combinatoria

- Si inizializza il BDD package creando le variabili di decisione
- Poi si attraversa la rete eventualmente livellizzata calcolando il ROBDD di ciascun segnale

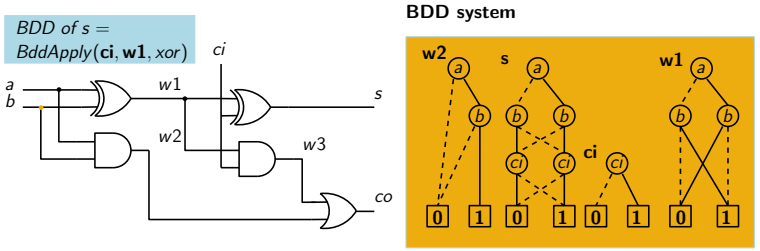


Sintesi logica

Introduzione
Boolean algebra
Rappresentazioni
BDD
Boolean networks
AIG

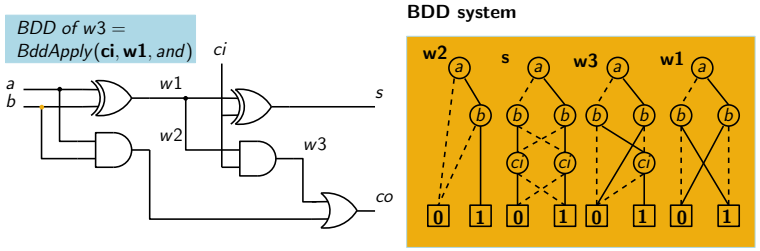
Calcolo del BDD di una rete combinatoria

- Si inizializza il BDD package creando le variabili di decisione
- Poi si attraversa la rete eventualmente livellizzata calcolando il ROBDD di ciascun segnale



Calcolo del BDD di una rete combinatoria

- Si inizializza il BDD package creando le variabili di decisione
- Poi si attraversa la rete eventualmente livellizzata calcolando il ROBDD di ciascun segnale



Sintesi logica

Introduzione
Boolean algebra
Rappresentazioni
BDD
Boolean networks
AIG

Applicazioni

SAT checking si guarda semplicemente che il ROBDD non sia uguale a 0 (***O*(*const*)**)

tautology checking si guarda che sia uguale a 1 (***O*(*const*)**)

SAT assignment si attraversa il ROBDD dalla radice (depth-first) annotando le variabili incontrate come 0 o 1 dipendentemente dall'arco seguito, se si raggiunge una foglia a 1 si é risolto il problema, se si raggiunge uno 0 si torna sull'ultima decisione che porterá necessariamente a 1 (***O*(*num.dec.var* + 1)**)

BDD of a combinational network

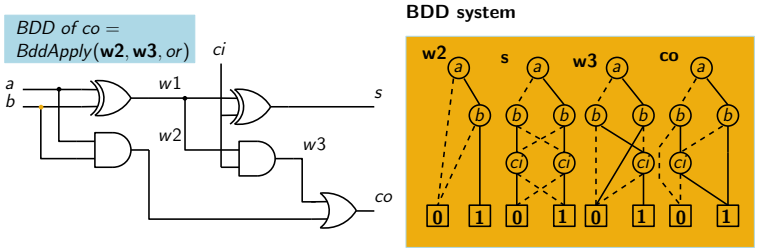
....

Sintesi logica

Introduzione
Boolean algebra
Rappresentazioni
BDD
Boolean networks
AIG

Calcolo del BDD di una rete combinatoria

- Si inizializza il BDD package creando le variabili di decisione
- Poi si attraversa la rete eventualmente livellizzata calcolando il ROBDD di ciascun segnale



Sintesi logica

Introduzione
Boolean algebra
Rappresentazioni
BDD
Boolean networks
AIG

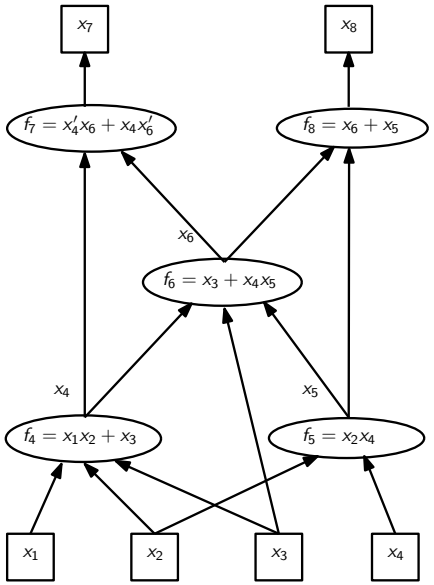
Boolean networks

- DAG $G = (V, E)$:
  - ogni vertice $v \in V$ ha associata una funzione f_v e una variabile di uscita x_v tale che $x_v = f_v$
  - un arco $e \in E$ che connette u con v significa che x_u é una variabile della funzione f_v
    - x_u appare quindi in ogni espressione di f_v
    - x_u appartiene al supporto locale strutturale di f_v
    - se $\partial f_v / \partial x_u \neq 0$ allora x_u appartiene al supporto locale funzionale di f_v
  - l'insieme degli archi entranti in un vertice viene detto fan-in (***FI*(*v*)**) e l'insieme degli archi uscenti fan-out (***FO*(*v*)**)
  - l'insieme dei nodi $i \in V$ con fan-in vuoto é detto insieme degli ingressi primari (primary-inputs, PIs)
  - l'insieme dei nodi $i \in V$ con fan-out vuoto é detto insieme delle uscite primarie (primary-outputs, POs)

Boolean networks

- Transitive fan-in:
  $TFI(v) = \{i \in V \mid i = v \vee i \in FI(j) \text{ for } j \in TFI(v)\}$
- Transitive fan-in:
  $TFO(v) = \{i \in V \mid i = v \vee i \in FO(j) \text{ for } j \in TFO(v)\}$
- $TFI(v) \cap TFO(v) = v$
- f_v (funzione locale) $\Rightarrow$ g_v (funzione globale): si sostituisce recursivamente in f_v la funzione f_u di ogni variabile $x_u \mid u \in TFI(v)$
- Possibili rappresentazioni per f_v: SOP, forme fattorizzate, BDD o AIG

Esempio



Utilizzo delle Boolean Network

- Storicamente si tratta del primo formato utilizzato nella sintesi di reti multilivello
- Formato piuttosto compatto che peró non offre particolari caratteristiche rilevanti dal punto di vista della manipolazione
- Gli AIG hanno dei vantaggi marginali nella sintesi e vantaggi addizionali nella verifica

And-Inverter Graphs (AIG)

- Grafo orientato i cui vertici possono essere AND a 2 ingressi (ingressi primari e uscite primarie visti eventualmente come pseudo-nodi) e gli archi possono essere di tipo invertente o meno
- Consente di rappresentare qualsiasi funzione Booleana
- Consente una rappresentazione compatta di molte funzioni che danno luogo a problemi nel caso dei BDD
- Formato utilizzato in ABC

Costruzione

- Trasformazione SOP ⇒ AIG: inserimento di parentesi per avere operazioni binarie e utilizzo della legge di De Morgan (fino a quando non si hanno solo prodotti)
- Nel caso in cui il punto di partenza sia dato da una rete combinatoria, si tratta di sostituire a ciascun gate l'equivalente AIG

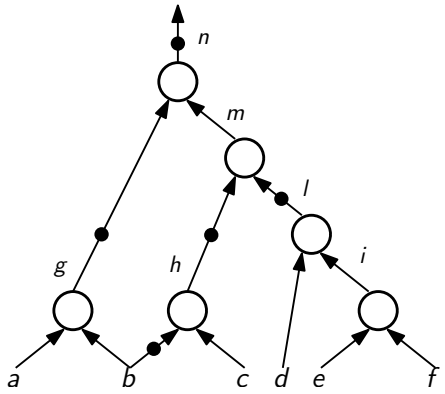
Esempio

Esempio: $y = ab + b'c + def$

$y = ((ab) + ((b'c) + (d(ef))))''$

$y = ((a \cdot b)'((b' \cdot c) + (d \cdot (e \cdot f)))')'$

$y = ((a \cdot b)'((b' \cdot c)'(d \cdot (e \cdot f))')')'$

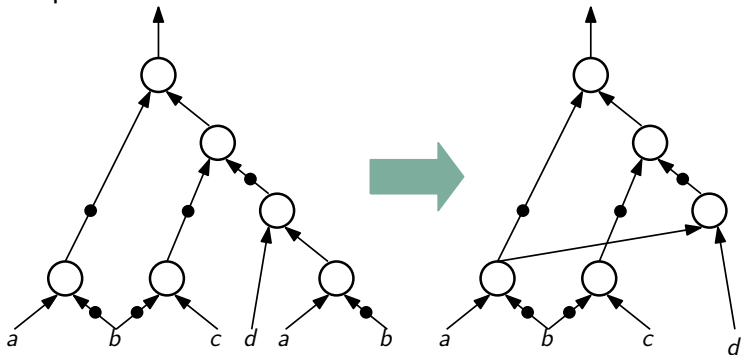


Caratteristiche degli AIG

- Gli AIG vanno visti come strutture dati e non come reti logiche
- Structural hashing: i nodi con gli stessi ingressi vengono riuniti durante la costruzione del grafo in cui ogni nodo risulta strutturalmente unico
- Gli invertitori sono rappresentati come archi complementati e non come porte logiche
- Ogni nodo del grafo occupa lo stesso spazio in memoria, questo puó essere sfruttato da un memory manager a basso livello per rendere efficiente l'attraversamento del grafo

Structural Hashing

- Esempio:



- Nota: non é un hashing funzionale, $a + bc$ e $(a + b)(a + c)$ danno luogo ad AIG diversi

Utilizzo

- Le operazioni di manipolazione sugli AIG possono essere fatte semplicemente connettendo le uscite degli AIG di partenza in maniera opportuna a nuovi nodi
  - se si vuole calcolare $f = f_1 \vee f_2$ ove f_1 é rappresentata da AIG_1 e f_2 da AIG_2, si collegano i nodi di uscita di AIG_1 e AIG_2 a un nodo AND tramite archi invertenti e poi si inverte anche l'uscita
- Un AIG non é una forma canonica, con qualche trasformazione si hanno proprietá locali di canonicitá
- Per questo motivo necessitano di essere utilizzati insieme a programmi SAT

Funzione caratteristica

- Dato un sottoinsieme $U \subseteq \mathbb{B}^n$, si dice funzione caratteristica di U, una funzione $\xi : \mathbb{B}^n \to \mathbb{B}$ tale che se $e \in U$ allora $\xi(e) = 1$ e 0 altrimenti
- Corrispondenza fra le operazioni sugli insiemi ($\cup$, $\cap$ e $\setminus$) e le operazioni Booleane ($\vee$, $\wedge$ e $\neg$)
- Esempio: descrizione di una funzione non completamente specificata $f : \mathbb{B}^n \to \{0, 1, -\}$ come funzione caratteristica
  - $f_{OFF}(m)$if and only if$f(m) = 0$
  - $f_{ON}(m)$if and only if$f(m) = 1$
  - $f_{DC}(m)$if and only if$f(m) = -$

Boolean relations

- Si consideri una funzione $f : \mathbb{B}^n \to \mathbb{B}^m$ e siano:
  - $x \in \mathbb{B}^n$ il vettore delle variabili di ingresso
  - $y \in \mathbb{B}^m$ il vettore delle variabili di uscita
- La relazione é definita come

$$R(x, y) = \bigwedge_{i=1}^{m} (y_i = f_i(x))$$

- Tutte le configurazioni di x e y che sono consistenti con f appartengono alla relazione
- La relazione puó quindi essere descritta dalla sua funzione caratteristica definita su $\mathbb{B}^{n+m}$
- Sono state utilizzate nell'ambito della sintesi ad esempio per ottimizzare celle combinatorie con piú uscite

Esempio

full-adder function:
$f : \mathbb{B}^3 \to \mathbb{B}^2$

a	b	cin	cout	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

characteristic function $\rho : \mathbb{B}^5 \to \mathbb{B}$ of the Boolean relation R

a	b	cin	cout	s	ρ
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	1
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	1
0	1	1	1	1	0

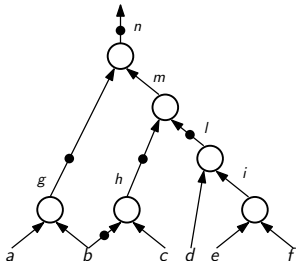
a	b	cin	cout	s	ρ
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	1
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	1

Funzione caratteristica ⇒ consistenza dei segnali di un circuito

- Data una rete combinatoria, si consideri l'insieme di variabili binarie costituito da ingressi, uscite e segnali interni
- Una configurazione binaria di tale insieme si dice consistente con la rete se per ogni gate della rete, l'uscita é quella corrispondente ai valori di ingresso di tale gate
- Vedremo che questo problema é descrivibile e risolubile utilizzando tecniche SAT

Funzione caratteristica ⇒ consistenza dei segnali di un circuito

- Rete combinatoria trasformata in un AIG
- La consistenza di un gate AND (con ingressi a, b e uscita c) é data da: $(ab \rightarrow c)(a' \rightarrow c')(b' \rightarrow c') = (a'+b'+c)(a+c')(b+c')$



- Le inversioni corrispondenti agli archi possono essere inserite complementando la variabile di ingresso
- Consistenza AIG ⇒ prodotto di quelle degli AND (CNF) $(a' + b' + g)(a + g')(b + g')(b + c' + h)(b' + h')(c + h')(e' + f' + i)(e + i')(h + i')(d' + i' + l)(d + l')(i + l')(h + l + m)(h' + m')(l' + m')(g + m' + n')(g' + n)(m + n)$
- La CNF vale 1 solo se ingressi e segnali hanno valori consistenti

Conclusioni

- Sono state analizzate alcune dei modelli matematici e delle conseguenti strutture dati utilizzabili per descrivere e manipolare funzioni booleane
- Non esiste un modello che possa essere definito come superiore agli altri, ma esistono applicazioni in cui un modello puó risultare preferibile