SAPIENZA Università di Roma

A.A. 2008-2009

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Metodi Formali per il Software e i Servizi

Fabio Patrizi

www.dis.uniroma1.it/~patrizi

Introduzione ai BINARY DECISION DIAGRAMS

(maggio 2009)

Riferimenti Bibliografici

Slides based on the following references:

1. Michael Huth, Mark Ryan, *Logic in Computer Science*. Cambridge University Press, 2000

2. H. R. Andersen, *An Introduction to Binary Decision Diagrams*. Lecture Notes, 1999, IT University of Copenhagen. (`http://www.configit.com/fileadmin/Configit/Documents/bdd-eap.pdf`)

3. Randall E. Bryant, *Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams*. ACM Comput. Surv. 24:3. 1992.

4. Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, L. J. Hwang, *Symbolic Model Checking:* 10^{20} *States and Beyond*. Inf. Comput., 98:2. 1992.

Binary Decision Diagrams: Motivazioni

Gli algoritmi per il Model Checking necessitano di memorizzare **insiemi di stati**

Il numero di stati cresce esponenzialmente con le proposizioni di un transition system

Nella pratica, centinaia di proposizioni rappresentano situazioni frequenti

Binary Decision Diagrams: Motivazioni

Esempio: verifica formale di una macchina a stati con 100 bit (comune microchip)

Tempo:

- 2^{100} ($\simeq 10^{30}$) stati
- età dell'universo $\simeq 2^{34}$ anni
- cominciando la visita dal Big Bang, ad una velocità di 2 milioni stati/sec, non avremmo ancora finito!

Spazio:

- Giga: 2^{30} (Wikipedia arriva "solo" allo "Yotta": 2^{80}), $2^{100} > 2^{30} \cdot 2^{30} \cdot 2^{30}$;
- Dove troviamo lo spazio per memorizzare (eventualmente tutti) gli stati?

Nozioni Base: Funzioni Booleane

Funzione Booleana: funzione di n variabili Booleane

$$f : \{0,1\}^n \rightarrow \{0,1\}$$

Le funzioni Booleane sono tipicamente rappresentabili come *formule proposizionali* o *tabelle di verità*

Es.:

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	0

$$f(x_1, x_2) = \neg(x_1 \vee x_2)$$

Distinguiamo tra l'**entità matematica** e la sua **rappresentazione**

Binary Decision Diagrams: Motivazioni

Spazio: anziché rappresentare gli stati esplicitamente, usiamo una rappresentazione simbolica, mediante funzioni booleane

Esempio:

- proposizioni: p_1, p_2
- Spazio degli stati $S = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$

Il sottoinsieme $S' = \{\langle 0,0\rangle, \langle 0,1\rangle\}$ può essere rapprentato dalla formula proposizionale $\phi'_S = \neg p_1$, la quale individua tutti gli stati di S tali che $p_1 = 0$

Quanto più è grande S, tanto più spazio risparmiamo per rappresentare *il sottoinsieme di S i cui stati hanno* $p_1 = 0$

Binary Decision Diagrams: Motivazioni

Tempo: abbiamo bisogno di effettuare operazioni tra insiemi di stati (intersezione, unione, complementazione, test d'insieme vuoto, etc.), ovvero sulle loro rappresentazioni simboliche. Ciò corrisponde a combinare (or, and, not) funzioni booleane ed esguire test (soddisfacibilità, validita) su di esse.

Soddisfacibilità e *validità* sono molto inefficienti per funzioni rappresentate in forma tabellare

Soddisfacibilità e validità di formule proposizionali notoriamente difficile (NP-hard e coNP-hard, rispettivamente)

Se ci limitiamo a CNF/DNF, soddisfacibilità e validità diventano duali:
molto semplice in una forma $\rightarrow$ molto costosa nell'altra (conversioni CNF $<->$ DNF tempo-esponenziali)

Rappresentazione	compatta?	test		operazioni		
		SAT	VALID	and	or	not
formule proposizionali	spesso	difficile	difficile	semplice	semplice	semplice
DNF	talvolta	facile	difficile	difficile	facile	difficile
CNF	talvolta	difficile	facile	facile	difficile	difficile
tabelle di verità ordinate	mai	difficile	difficile	difficile	difficile	difficile

Binary Decision Diagrams: Motivazioni

OBIETTIVO: Forma di rappresentazione **compatta** che migliori l'**efficienza**

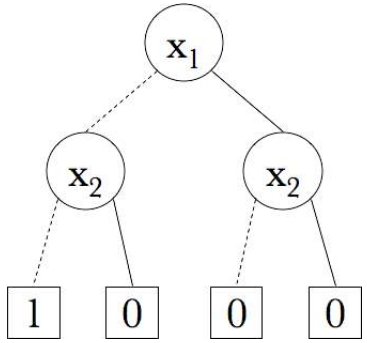
- ridurre lo spazio per la rappresentazione
- ridurre il tempo necessario alle operazioni

Ordered Binary Decision Diagrams (OBDD):

		test		operazioni		
Rappresentazione	compatta?	SAT	VALID	and	or	not
...	...	...	...	...	...	...
OBDD	spesso	facile	facile	media	media	facile

Alberi Binari di Decisione

Esempio: $f(x_1, x_2) = \neg(x_1 \vee x_2)$



Nodi quadrati: **terminali**. Nodi tondi: **non terminali**

Ogni nodo non terminale ha sempre un arco tratteggiato ed un arco pieno verso i nodi figlio (terminali o non)

I nodi terminali non hanno archi uscenti

Gli archi tratteggiati (pieni) rappresentano l'assegnazione del valore $\bot$ ($\top$) alla variabile contenuta nel nodo padre

Le assegnazioni alle variabili ottenute seguendo un percorso dalla radice alla foglia dànno come valore della funzione quello contenuto nel nodo foglia

Es.: per calcolare $f(1,0)$, seguo l'arco pieno uscente da x_1 e, successivamente l'arco tratteggiato uscente da x_2, ottenendo quindi il valore 0 Infatti: $\neg(\top \vee \bot) = \bot$

Alberi Binari di Decisione (2)

Quanto risparmiamo rappresentando una funzione con un Albero Binario di Decisione?

Funzione booleana di n variabili in forma tabellare: 2^n righe

Funzione booleana di n variabili come Albero: almeno $2^{n+1} - 1$ nodi (eventualmente di più, se ci sono occorrenze multiple di variabili)

Non è molto conveniente:

n	tabella	albero
0	1	1
1	2	3
2	4	7
...	...	...
10	1024	2047
...	...	...

Possiamo eliminare delle ridondanze significative, **trasformando l'albero in un grafo**

Il grafo ottenuto è detto **Diagramma Binario di Decisione (BDD)**

Esercizio 1

Si consideri la funzione booleana descritta dalla seguente tabella di verità

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

a. Costruire un albero binario di decisione per la funzione $f(x_1, x_2, x_3)$ dove la radice è è etichettata con x_1 ed è seguita, in ogni cammino radice-foglia, da un nodo etichettato con x_2 quindi da uno etichettato con x_3

b. Costruire un altro albero binario di decisione, dove la radice è un nodo x_3, seguito da un nodo x_2 quindi da x_1

Esercizio 2

Sia T un generico albero binario di decisione, per una generica funzione $f(x_1, \ldots, x_n)$ di n variabili booleane. Assumendo che ogni variabile occorra esattamente una volta lungo ciascun cammino radice-foglia, dimostrare che T contiene $2^{n+1} - 1$ nodi.

Binary Decision Diagrams

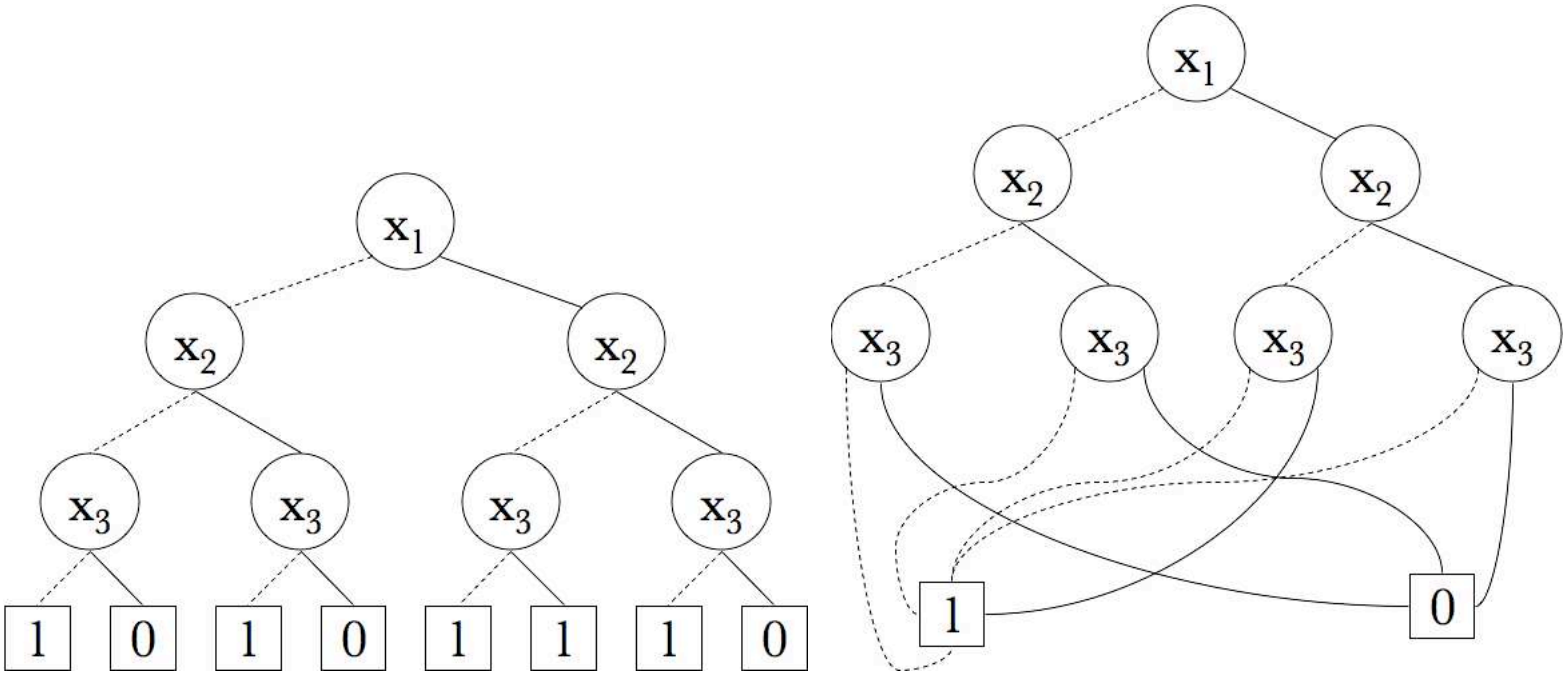
Binary Decision Diagram (BDD):

- grafo diretto aciclico (DAG)
- un solo nodo iniziale (cioè senza archi entranti)
- nodi terminali (cioè senza archi uscenti) etichettati con 0 o 1
- nodi nonterminali etichettati con una variabile booleana
- nodi nonterminali con due archi uscenti: uno etichettato con 0 (tratteggiato) ed uno con 1 (tratto pieno)

NOTA: Ogni Albero Binario di Decisione è un (caso particolare di) BDD

BDD: Ottimizzazioni

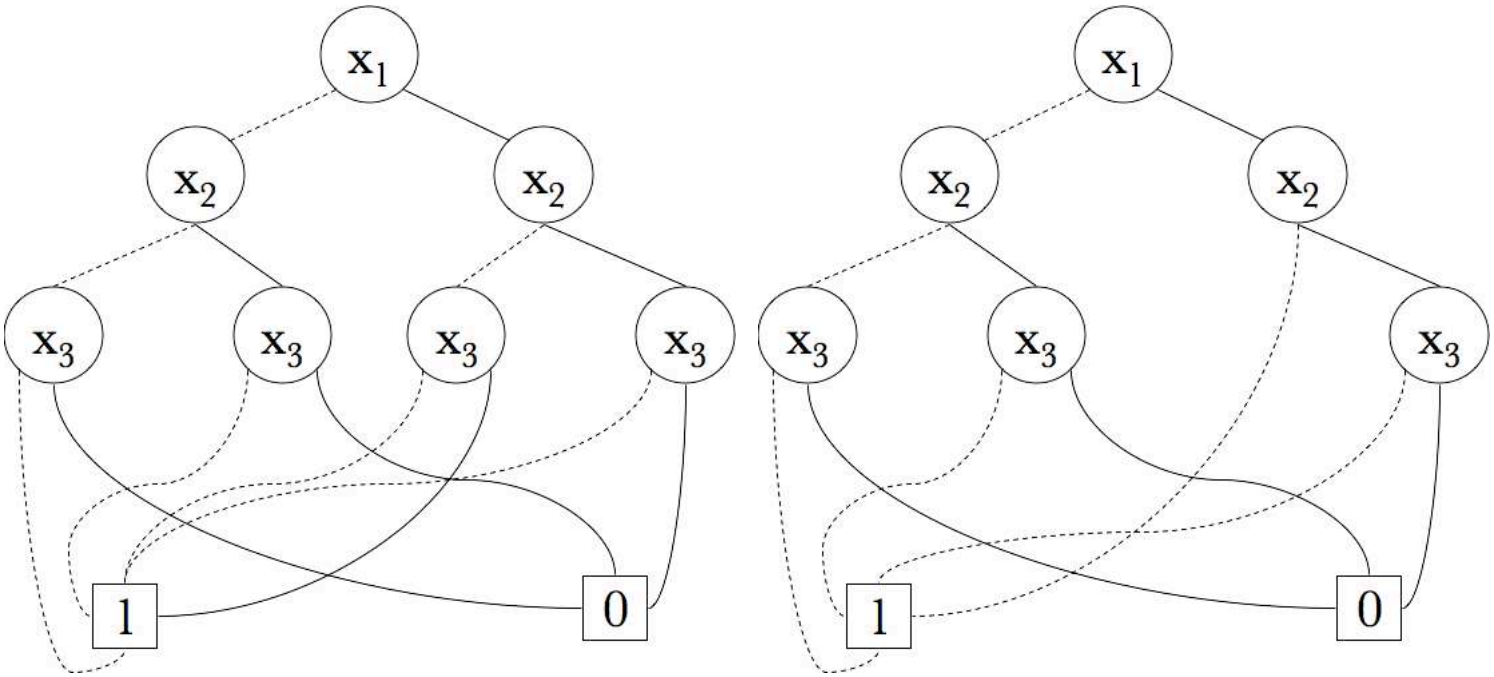
Possiamo usare **solo 2 terminali** anziché 2^l (l: profondità dell'albero)



NOTA: risparmiamo spazio per $2^l - 2$ simboli terminali, ma abbiamo lo **stesso numero di archi**

BDD: Ottimizzazioni (2)

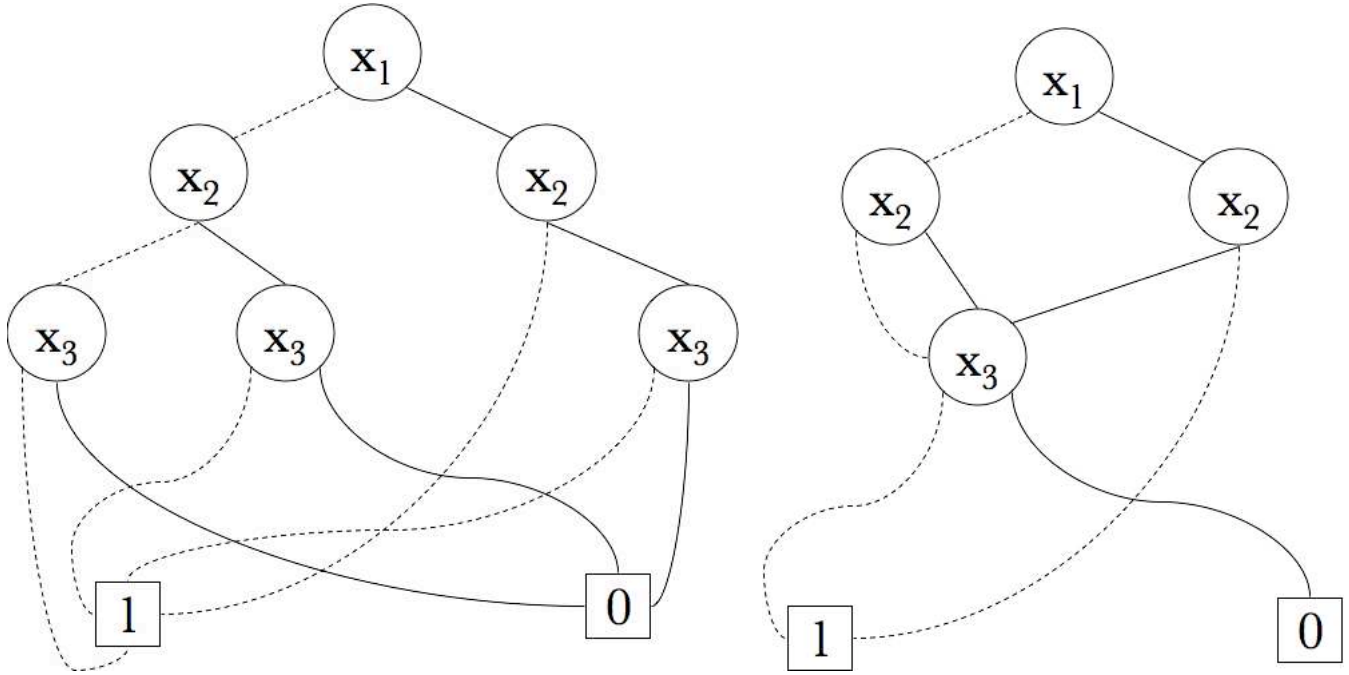
Rimozione dei **punti di scelta inutili**



(Il valore della funzione non dipende dal valore di x_3, se $x_1 = 1$ ed $x_2 = 0$)

BDD: Ottimizzazioni (3)

Condivisione (o sharing) dei sotto-BDD



(Tutti i sotto-BDD aventi nodo iniziale etichettato con x_3 sono uguali)

BDD: Ottimizzazioni (4)

Tre ottimizzazioni per rendere un BDD più compatto:

C1: Rimozione di terminali duplicati

C2: Rimozioni dei test ridondanti

C3: Rimozione dei non-terminali duplicati

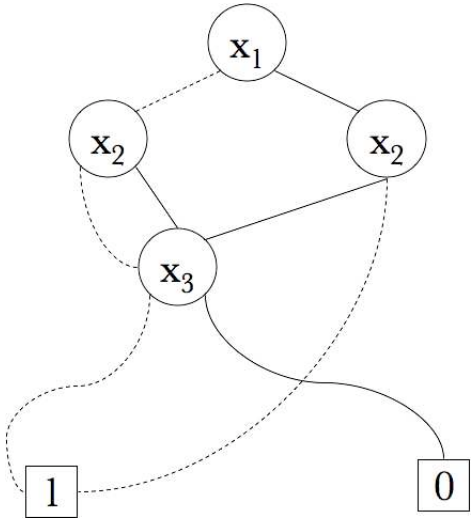
BDD Ridotti

Applicando una delle ottimizzazioni C1-C3 ad un BDD, si ottiene ancora un BDD

L'applicazione delle ottimizzazioni può quindi essere iterata

Un BDD su cui non è possibile applicare ulteriori ottimizzazioni è detto **ridotto**

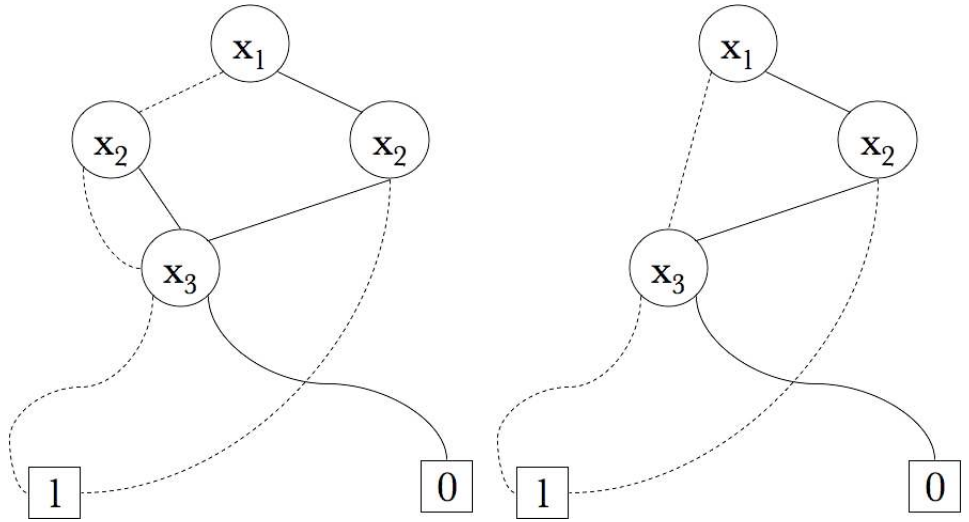
Il BDD ottenuto è ridotto?



BDD Ridotti: Esempio

Il BDD ottenuto è ridotto?

No: ottimizzazione C2!



BDD: Soddisfacibilità

La formula rappresentata da un BDD è **soddisfacibile**

$\Leftrightarrow$

Il BDD contiene un path consistente dalla radice ad una foglia etichettata con 1

BDD: Validità

La formula rappresentata da un BDD è **valida**

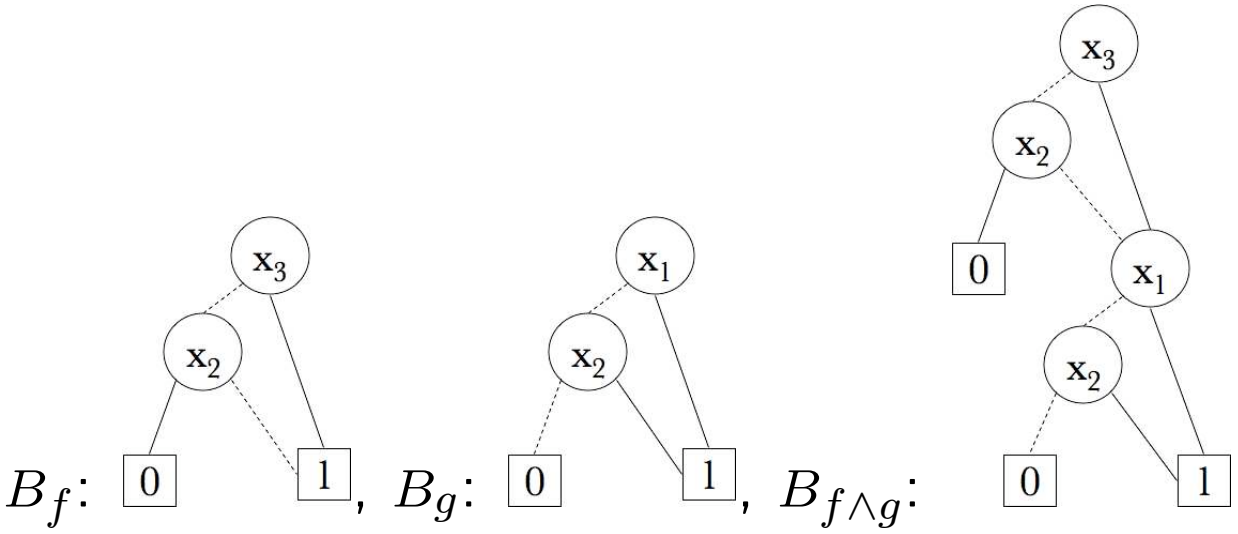
$\Leftrightarrow$

Il BDD non contiene alcun path consistente dalla radice ad una foglia etichettata con 0

BDD: Operatore AND

Rimpiazzamento dei terminali 1 con BDD

Funzioni booleane: $f = x_3 \vee \neg x_2$, $g = x_1 \vee x_2$

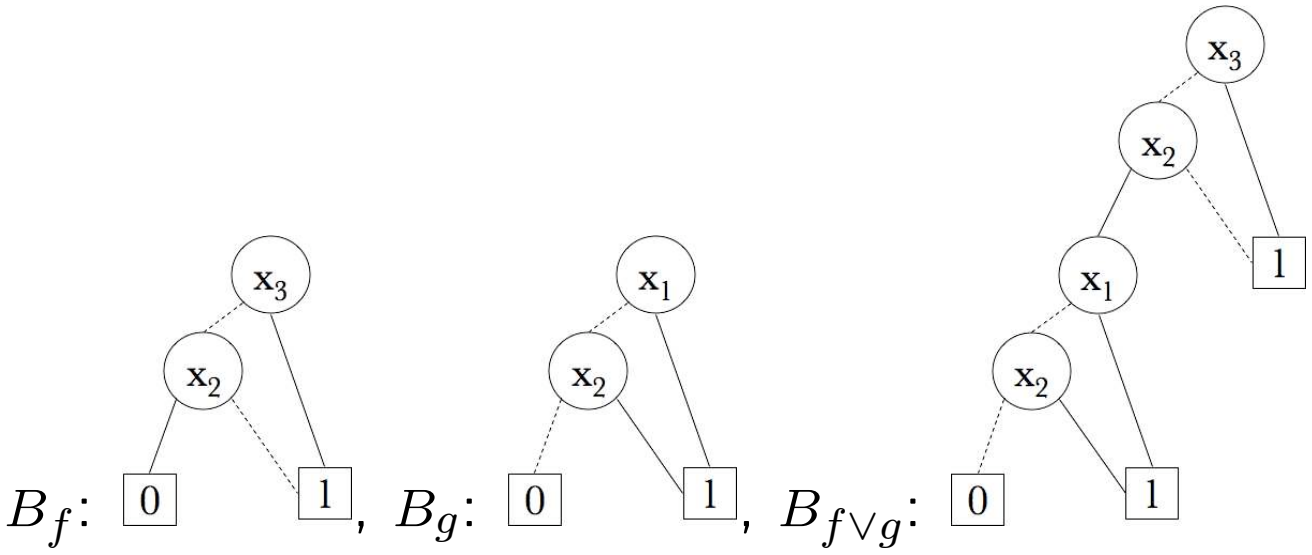


Compattezza **non preservata**

BDD: Operatore OR

Rimpiazzamento dei terminali 0 con BDD

Funzioni booleane: $f = x_3 \vee \neg x_2$, $g = x_1 \vee x_2$

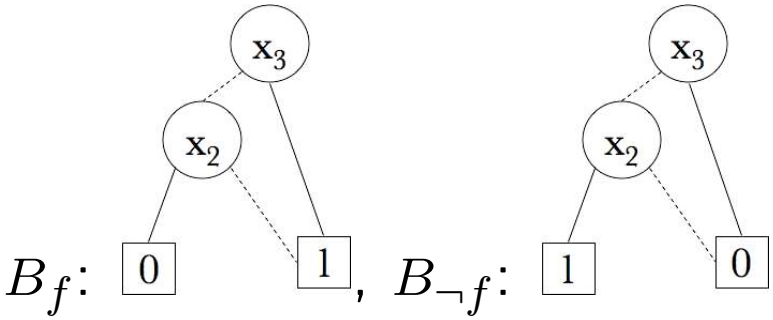


Compattezza **non preservata**

BDD: Operatore NOT

Scambio di terminali 0 con terminali 1

$f = x_3 \vee \neg x_2$



Compattezza **preservata**

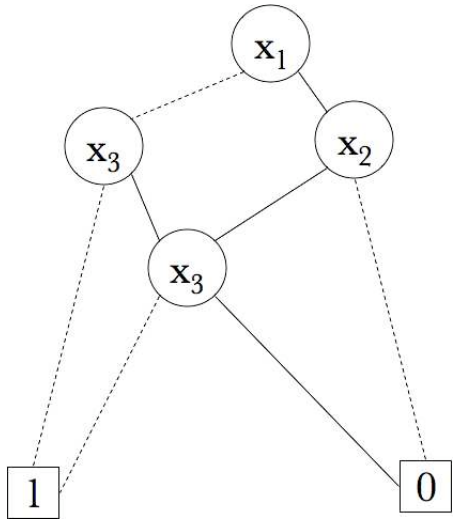
Esercizio 3

Costruire i BDD associati alle seguenti funzioni booleane:

- $f(x, y) = x \wedge y$
- $f(x, y) = x \vee y$
- $f(x, y) = x \oplus y$ ($\oplus$ = XOR)
- $f(x, y, z) = (x \oplus y) \wedge (\neg x \vee z)$

BDD: Ordinamento delle Variabili

Path radice-foglia possono contenere diverse occorrenze di una stessa variabile



Spreco di spazio: basterebbe connettere x_1 al nodo x_3 più profondo

Inconsistenze: il path $x_1 - x_3 - x_3 - 1$ è **inconsistente**

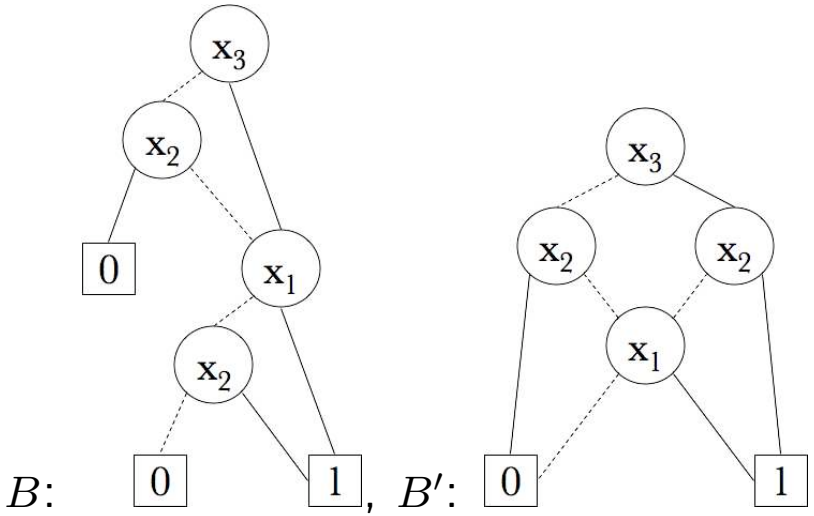
BDD: Ordinamento delle Variabili (2)

L'**ordinamento** delle variabili lungo i cammini radice-foglia gioca un ruolo importante

BDD con occorrenze multiple di una stessa variabile lungo i cammini (radice-foglia) sono inefficienti:

- **più spazio** rispetto a rappresentazioni con occorrenza singola
- efficienza del **test d'equivalenza** compromessa

Esempio: B,B' rappresentano la stessa funzione $f = (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2)$



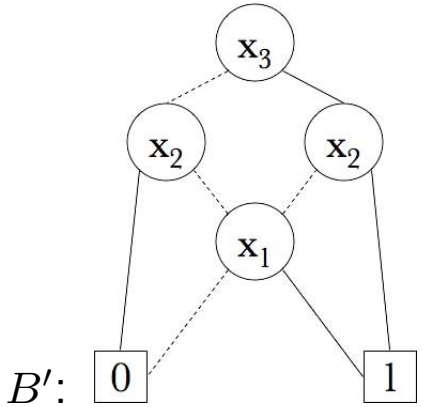
ESERCIZIO: verificare l'equivalenza di B e B'

BDD Ordinati (OBDD)

Ordine: insieme di variabili booleane su cui è definito un ordine (formalmente: stretto e totale)

Ordered BDD: tutti i cammini radice-foglia rispettano uno stesso ordinamento delle variabili

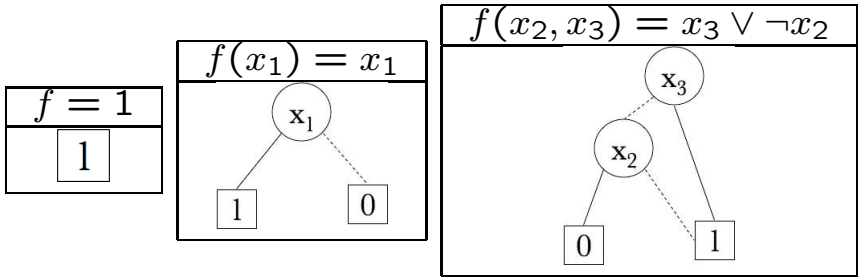
Esempio:

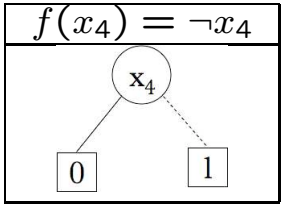


Ordine delle variabili: $[x_3, x_2, x_1]$

BDD Ordinati (OBDD) (2)

I seguenti BDD sono ordinati secondo l'ordine $O = [x_3, x_2, x_1]$:

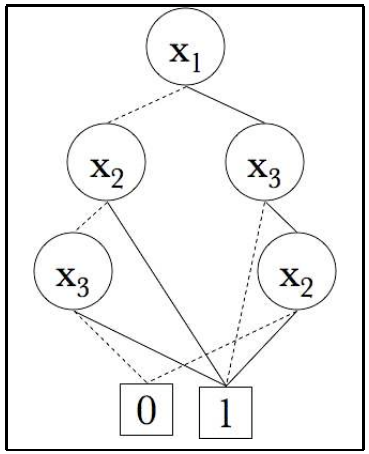


Il BDD  non è ordinato secondo O perché $x_4 \notin O$. Tuttavia, **è anch'esso un OBDD**, poiché

è ordinato secondo, ad es., $O' = [x_4]$ (ma anche $O'' = [x_3, x_4]$, $O''' = [x_4, x_3, x_1]$, etc...)

Ovviamente, BDD con occorrenze multiple nei cammini **non possono essere ordinati**

Esistono anche BDD non ordinati, **senza occorenze multiple**:



ATTENZIONE: gli operatori AND e OR, definiti sopra, tra BDD possono introdurre **occorrenze multiple** e quindi **non preservano l'ordinamento**

BDD Ordinati: Compatibilità

Ordini Compatibili: non esistono due variabili x, y tale che $x < y$ in un ordine e $x > y$ nell'altro

Teorema: L'OBDD ridotto (ROBDD) che rappresenta una data funzione f **è unico** (per un ordine fissato)
Dimostrazione: Induzione sul numero di argomenti della funzione

Ovvero: dati due ROBDD con ordini compatibili, essi rappresentano la stessa funzione booleana sse hanno **identica struttura**

Quindi: la verifica dell'equivalenza di due funzione rappresentate da ROBDD (con ordini compatibili) è IMMEDIATA!

BDD Ordinati: Forma Canonica

Dato un OBDD, se lo riduciamo usando le ottimizzazioni C1-C3, otteniamo **sempre lo stesso OBDD ridotto**, indipendentemente dall'ordine di applicazione delle ottimizzazioni

Data una funzione booleana, il suo OBDD ridotto (ROBDD) ne è una rappresentazone in **forma canonica**

Esercizio: costruire i ROBDD B_0, B_1, B_x, $B_{\neg x}$, $B_{x \wedge y}$, $B_{x \vee y}$

ROBDD: Vantaggi

Permettono la rappresentazione compatta di molte funzioni che in altre rappresentazioni occuperebbero **spazio esponenziale**

Esistono algoritmi efficienti (diversi da quelli visti sopra) per costruire combinazioni booleane di ROBDD che siano ancora ROBDD

I test di soddisfacibilità, validità, falsità ed equivalenza sono eseguiti in maniera molto efficiente (rispetto alle dimensioni dei ROBDD)

L'introduzione dei ROBDD nel Model Checking ha reso trattabili, **nella pratica**, problemi che prima non lo erano

ROBDD: Vantaggi (2)

Esempio Stabilire se una funzione booleana f, rappresentata come ROBDD, è valida ha costo $O(1)$

È sufficiente verificare che il ROBDD sia B_1=$\boxed{1}$

ROBDD: Vantaggi (3)

Esempio: funzione *parità* a 3 argomenti (1 sse il numero di variabili x_1, x_2, x_3 con valore 1 è pari)

$$parita(x_1, x_2, x_3) = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3)$$

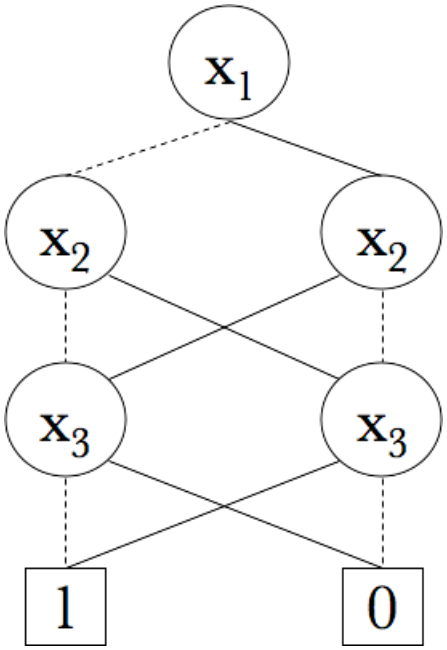
In generale, come formula proposizionale, contiene $n2^{n-1}$ letterali

Come ROBDD, contiene solo $2n + 1$ nodi

Esercizio: Costruire il ROBDD che rappresenta la funzione booleana *parità* a 3 argomenti

Soluzione

$$parita(x_1, x_2, x_3) = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3)$$



Esercizio 4

Data la funzione booleana $f(x_1, x_2, x_3) = x_1 \wedge (x_2 \vee \neg x_3)$, costruire il suo ROBDD per i seguenti ordinamenti delle variabili:

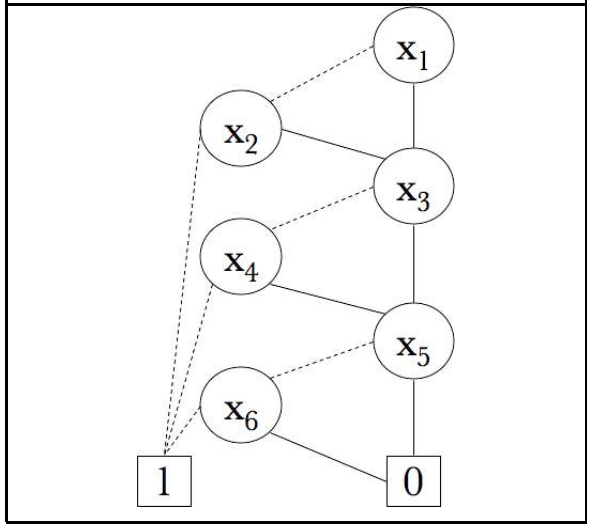
- $[x_1, x_2, x_3]$
- $[x_3, x_1, x_2]$
- $[x_3, x_2, x_1]$

Impatto dell'Ordine

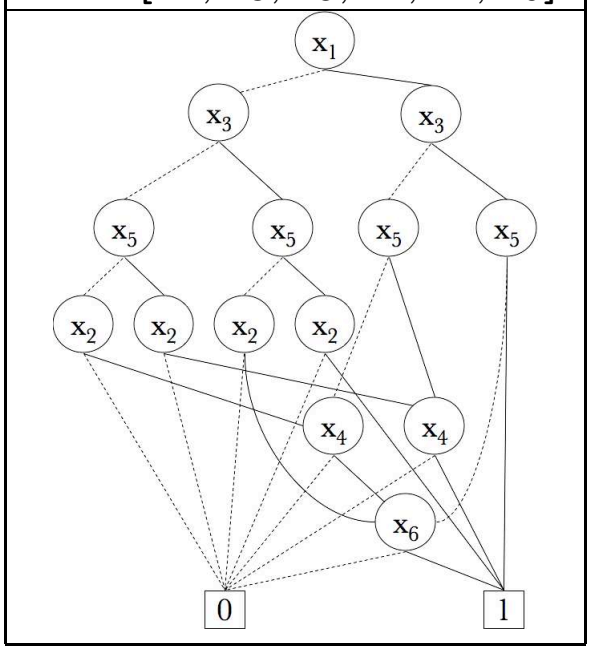
Data una funzione booleana f, la dimensione del ROBDD **dipende dall'ordinamento scelto**

Esempio: $f = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6)$

$O = [x_1, x_2, x_3, x_4, x_5, x_6]$



$O = [x_1, x_3, x_5, x_2, x_4, x_6]$



In generale, per $f = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \ldots \wedge (x_{2n-1} \vee x_{2n})$:

- $O = [x_1, x_2, \ldots, x_{2n-1}, x_{2n}] : N = 2n + 2$
- $O = [x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n}] : N = 2^{n+1}$

Impatto dell'Ordine (2)

In generale, un OBDD può avere dimensione esponenziale nel numero delle variabili (v. esempio precedente)

La sensibilità degli OBDD all'ordine è Il prezzo che paghiamo per ottenere i vantaggi degli OBDD rispetto ai BDD

Trovare l'ordinamento ottimo dele variabili è un problema **NP-completo**

Nella pratica, si adottano delle **euristiche** che generano "buoni" ordinamenti in molti casi

Sommario

Per effettuare MC efficientemente, abbiamo bisogno di rappresentazioni efficienti di insiemi di stati

Usiamo le funzioni booleane, rappresentate da ROBDD

Offrono una forma canonica (fissato un ordinamento) per la rappresentazione di funzioni

La forma canonica ci permette di eseguire le seguenti verifiche efficientemente:

Assenza di variabili ridondanti: se una funzione $f(x_1, \ldots, x_n)$ non dipende dal valore di qualche variabile x_i allora x_i non appare nel ROBDD

Equivalenza semantica: f e g equivalenti se i rispettivi ROBDD hanno struttura identica (con ordinamento compatibile)

Validità: f è valida sse è rappresentata dal ROBDD B_1

Implicazione: $f \rightarrow g$ sse il ROBDD $B_{f \wedge \neg g}$ è B_0

Soddisfacibilità: f è soddisfacibile sse l'ROBDD B_f è diverso da B_0

Algoritmi per ROBDD

Le riduzioni C1-C3 sono alla base di ogni manipolazione degli OBDD

Abbiamo già visto degli algoritmi (naïve), molto efficienti, per comporre BDD con operatori AND, OR e NOT

Purtroppo, questi algoritmi non preservano l'ordinamento né la compattezza quindi, se applicati a (R)OBDD, non restituiscono, in generale, OBDD

Esistono algoritmi efficienti per la composizione di ROBDD

Algoritmi per ROBDD: `reduce`

Ad ogni OBDD B, associamo le seguenti funzioni:

- $e(n)$: per ogni nodo n, restituisce l'etichetta (variabile se non-terminale, costante se terminale)
- $l(n)$: per ogni nodo n, restituisce il figlio connesso tramite un arco tratteggiato
- $h(n)$: per ogni nodo n, restituisce il figlio connesso tramite un arco a tratto pieno

Inoltre, se $\{x_1, \ldots, x_n\}$ sono tutte le variabili che etichettano i nodi di B, definiamo strato i-esimo di B l'insieme dei nodi di B che hanno stessa etichetta x_i, per $i \leq n$ e strato $(n+1)$-esimo l'insieme di tutti i nodi terminali di B

L'algoritmo **`reduce`** procede bottom-up, dapprima costruendo una funzione $id(n)$ che associa ad ogni nodo un identificatore intero, e successivamente fondendo, ancora bottom-up, i nodi con stesso identificatore e redirezionando opportunamente gli archi

Algoritmi per ROBDD: `reduce` (2)

```
For each nodo terminale n
  id(n):=e(n); // riduzione C1
let cur_id := 2;
For each strato S, partendo dal più profondo
  For each nodo n in S
    if id(l(n))=id(h(n))
      then
        id(n)=id(l(n))=id(h(n)); // riduzione C2: test ridondante
      else
        if esiste un nodo m tale che e(n)=e(m) and id(m) è definita and
                          id(l(n))=id(l(m)) and id(h(n))=id(h(m))
         then
           id(n)=id(m); // riduzione C3
         else{
           id(n)=cur_id;
           cur_id++;
         }
```

La procedura termina con un'iterazione bottom-up che:

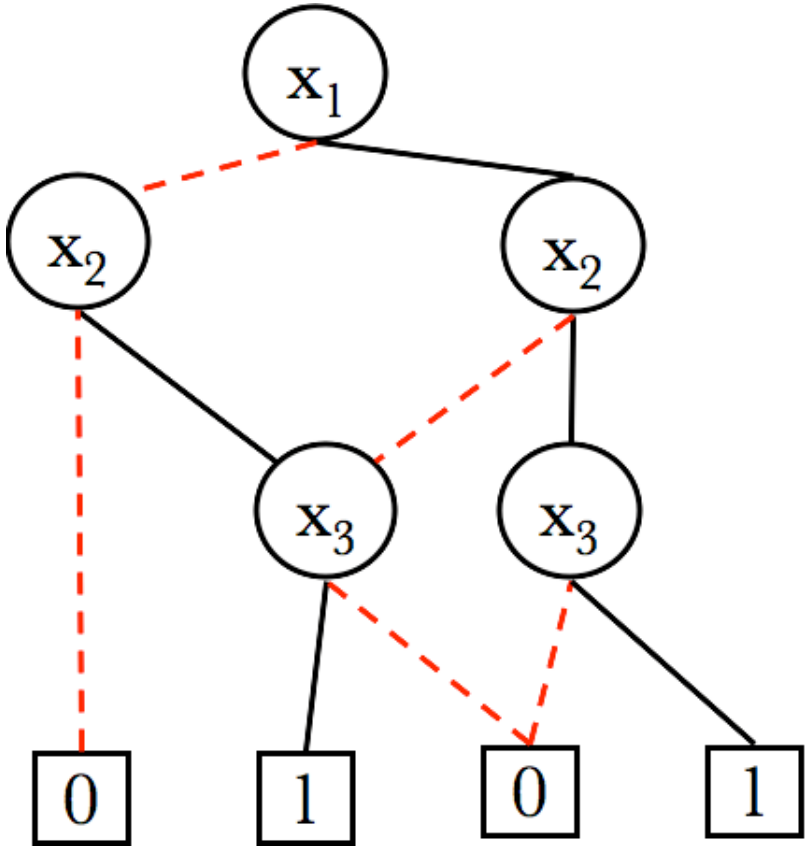
1. unisce i nodi di uno stesso livello aventi stesso identificatore
2. redireziona conseguentemente gli archi
3. redireziona l'arco entrante nei nodi aventi stesso identificatore dei nodi figlio verso il sottoBDD condiviso, ottenuto unendo i nodi figlio.

Complessità: $O(|B| \cdot log|B|)$ (ottenuta con l'uso di hash tables)

Nella versione mostrata: $O(|B|^2 \cdot log|B|)$

`reduce`: Esempio

Applicare l'algoritmo `reduce` all'OBDD in figura, associando ad ogni nodo il suo intero identificativo



Algoritmi per ROBDD: `apply`

Applica un operatore tra OR,AND,XOR a due OBDD ($B_{\neg f} = B_{f\ XOR\ 1}$) e restituisce l'(R)OBDD

Input: $op \in \{OR, AND, XOR\}$, ROBDD B_f, ROBDD B_g; OUTPUT: (R)OBDD $B_{(f\ op\ g)}$

Basato sull'applicazione ricorsiva dell'**espansione di Shannon**

$$f(x) \equiv \neg x \wedge f[x \leftarrow 0] \vee x \wedge f[x \leftarrow 1]$$

In particolare:

$$f\ op\ g = \neg x \wedge (f[x \leftarrow 0]\ op\ g[x \leftarrow 0]) \vee x \wedge (f[x \leftarrow 1]\ op\ g[x \leftarrow 1])$$

Diverse ottimizzazioni vengono applicate on-the-fly

In generale, è necessaria un'applicazione finale di `reduce` per ottenere riduzione

Complessità: $O(|B_f| \cdot |B_g|)$ (nella versione naïve la complessità è esponenziale, se non si usa *memoization*)

Algoritmi per ROBDD: `apply` (2)

B_f e B_g: OBDD con ordinamenti compatibili; $op \in \{\wedge, \vee, \oplus\}$ operatore da applicare

Chiamiamo r_f ed r_g i nodi radice di B_f e B_g

- Se entrambi r_f ed r_g sono terminali, allora calcoliamo $e(r_f)$ op $e(r_g)$ e restituiamo l'OBDD corrispondente (B_0 o B_1)

- Negli altri casi, almeno uno tra r_f ed r_g è non-terminale. Assumiamo che entrambi siano tali che $e(r_f) = x_i$ ed $e(r_g) = x_i$. Creiamo un nodo n tale che $e(n) = x_i$, con $l(n) = apply(op, l(r_f), l(r_g))$ e $h(n) = apply(op, h(r_f), h(r_g))$

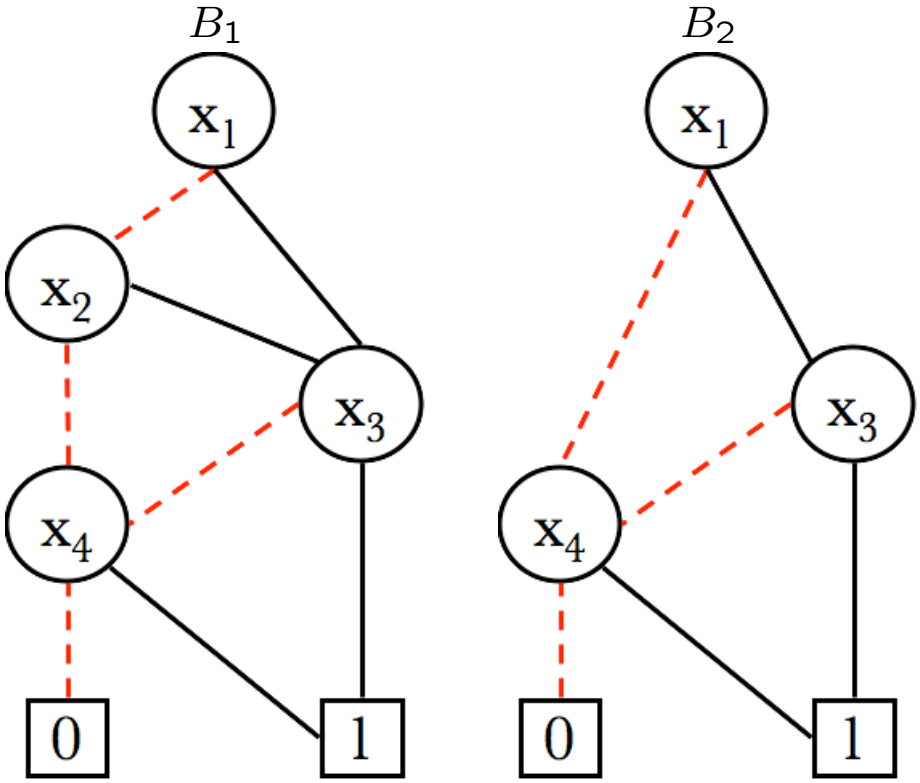
- se, invece, $e(r_f) = x_i$ ed r_g è terminale o $e(r_g) = x_j$ con $j > i$, allora, per la compatibilità degli ordinamenti, B_g non contiene nodi n t.c. $e(n) = x_i$ e, quindi g non dipende da x_i. Quindi, creiamo un nodo n t.c. $e(n) = x_i$ con $l(e) = apply(op, lo(r_f), r_g)$ e $h(e) = apply(op, hi(r_f), r_g)$

- il caso in cui r_f ed r_g verificano le condizioni complementari è gestito in maniera simmetrica.

La procedura si conclude con una chiamata a **`reduce`**

`apply`: Esempio

Applicare l'algoritmo `apply`$(\vee, B_1, B_2)$ agli OBDD in figura, mettendo in evidenza le successive chiamate ricorsive effettuate



Altri Algoritmi per ROBDD

Altri algoritmi necessari nelle implementazioni reali:

- `restrict`: dato un OBDD B associato ad una funzione f, una sua variabile x ed una costante $c \in \{1, 0\}$, resituisce il ROBDD B', associato alla funzione $f' = f[x \leftarrow c]$:
  - `restrict`$(0, x, B)$: per ogni nodo n di B tale che $e(n) = x$ redireziona gli archi entranti verso $l(n)$ e rimuove n
  - `restrict`$(1, x, B)$: simmetrico (redireziona gli archi entranti verso $h(n)$)

  Complessità: $O(|B_f| \cdot log|B_f|)$

- `exists`: dato un OBDD B associato ad una funzione f ed una sua variabile x, costruisce il ROBDD associato alla funzione

$$\exists x.f \doteq f[x \leftarrow 0] \vee f[x \leftarrow 1]$$

  Usato nel calcolo della pre-immagine

  Può essere implementato come `apply`$(+,$`restrict`$(0, x, B),$`restrict`$(1, x, B))$

  Si possono applicare diverse ottimizzazioni

Algoritmi per OBDD: Complessità

Algoritmo	Input OBDD	Output OBDD	Complessità temporale
`reduce`	B	B ridotto	$O(\|B\| \cdot log\|B\|)$
`apply`	B_f, B_g	$B_f \ _{op} \ _g$ (ridotto)	$O(\|B_f\| \cdot \|B_g\|)$
`restrict`	B_f	$B_{f[x \leftarrow 0]}$ o $B_{f[x \leftarrow 1]}$ (ridotto)	$O(\|B_f\| \cdot log\|B_f\|)$
`exists`	B_f	$B_{\exists x_1 \ldots \exists x_n f}$ (ridotto)	NP-completo

Algoritmi per ROBDD: Considerazioni

Gli algoritmi `reduce` e `apply` sono efficienti **nella dimensione dei ROBDD in input**

Se una funzione f non ha una rappresentazione compatta come ROBDD B_f (es., ha dimensione esponenziale nel numero delle variabili), allora le computazioni che coinvolgono B_f saranno costose

Un esempio di simili funzioni è l'insieme delle funzioni per la moltiplicazione degli interi: hanno dimensione esponenziale nel numero di bit di input

QUINDI

Nel caso peggiore non cambia nulla!

Model Checking Simbolico

Abbiamo visto l'algoritmo `Labeling` per il MC in CTL

Dati un KM ed una formula CTL φ, l'algoritmo calcola l'insieme degli stati di KM che soddisfano φ

Per fare ciò, è necessario eseguire delle operazioni su insiemi intermedi di stati (si pensi al calcolo di un fixpoint)

In un'implementazione naïve, gli insiemi sono rappresentati esplicitamente

In un'implementazione efficiente sono rappresentati in maniera simbolica

L'algoritmo di Labeling può essere, e **tipicamente è**, realizzato efficientemente sfruttando i ROBDD

Insiemi di Stati come OBDD

Consideriamo un insieme (generico) finito S

Vogliamo rappresentarne i sottoinsiemi T come OBDD

1. Codifica binaria di ogni stato, come vettore di $n = \lceil log_2 \ |S| \ \rceil$ bit

2. Ad ogni sottinsieme $T \subseteq S$ associamo la sua funzione caratteristica $f_T : \{0,1\}^n \rightarrow \{0,1\}$, pari ad 1 sse lo stato $s \in S$ associato al vettore argomento di f è tale che $s \in T$

Insiemi di Stati come OBDD: Esempio

$$S = \{a, b, c, d, e, f, g\}$$

$$T = \{a, b, f\}$$

Codifica binaria degli stati

$$n = \lceil\ log_2(7)\ \rceil = 3$$

Stato	$v_1 v_2 v_3$
a	0 0 0
b	0 0 1
c	0 1 0
d	0 1 1
e	1 0 0
f	1 0 1
g	1 1 0

Funzione Caratteristica (tabella di verità)

$v_1 v_2 v_3$	f_T
(a) 0 0 0	1
(b) 0 0 1	1
(c) 0 1 0	0
(d) 0 1 1	0
(e) 1 0 0	0
(f) 1 0 1	1
(g) 1 1 0	0
(-) 1 1 1	-

Scelta della Codifica Binaria

Nel caso del MC, usiamo la funzione di etichettatura del KM $L : S \to 2^{AP}$ per la codifica degli stati

Le variabili booleane che usiamo sono esattamente le proposizioni atomiche associate al TS

Ciascuno stato $s \in S$ è associato all'insieme $L(s)$
(per semplicità, assumiamo che se $L(s) = L(s')$ allora $s = s'$. È comunque sempre possibile applicare trasformazioni semplici al KM per soddisfare l'assunzione)

Rappresentazione di Stati

Possiamo rappresentare ogni stato $s \in S$ con la funzione booleana

$$f_s = \bigwedge_{i=1}^{|AP|} l_i$$

dove $l_i = p_i$ se $p_i \in L(s)$ e $l_i = \neg p_i$ se $p_i \notin L(s)$

Esempio

Come visto in precedenza, un contatore modulo 8 ha 8 stati, $S = \{s_0, \ldots, s_7\}$. La funzione di etichettatura $L : S \to 2^{AP}$ associa ad ogni stato un sottoinsieme delle proposizioni atomiche $AP = \{b_0, b_1, b_2\}$ (tutte e sole quelle vere in quello stato).

$L(s_0) = \emptyset$ (tutte le proposizioni sono false: i bit sono tutti a 0)
$L(s_1) = \{b_0\}$ (solo b_0 è vera: il bit 0 è pari ad 1 e tutti gli altri a 0)
...
$L(s_6) = \{b_2, b_1\}$ (b_1 e b_2 sono vere: i bit 1 e 2 sono pari ad 1 ed il bit 3 pari a 0)
$L(s_7) = \{b_0, b_1, b_2\}$ (tutte le proposizioni vere: tutti i bit hanno valore 1)

Quindi abbiamo:
$f_{s_0} = \neg b_0 \wedge \neg b_1 \wedge \neg b_2$;
$f_{s_1} = b_0 \wedge \neg b_1 \wedge \neg b_2$;
$f_{s_6} = \neg b_0 \wedge b_1 \wedge b_2$;
$f_{s_7} = b_0 \wedge b_1 \wedge b_2$

Rappresentazione di Insiemi di Stati

Un insieme di stati T può essere facilmente rappresentato dalla disgiunzione delle funzioni caratteristiche degli stati in esso contenuti

Esempio

$S = \{s_0, \ldots, s_7\}$

$T = \{s_0, s_1, s_6\}$

$f_T = f_{s_0} \vee f_{s_1} \vee f_{s_6} = (\neg b_0 \wedge \neg b_1 \wedge \neg b_2) \vee (b_0 \wedge \neg b_1 \wedge \neg b_2) \vee (\neg b_0 \wedge b_1 \wedge b_2)$

Operazioni su Insiemi di Stati

Le operazioni di **intersezione, unione e complementazione** tra insiemi corrispondono agli operatori $\wedge$, $\vee$, $\neg$, rispettivamente, applicati alle funzioni booleane rappresentative degli insiemi

Esempio

$T_1 = \{s_0, s_1\}$, $T_2 = \{s_1, s_2\}$

$f_{T_1} = f_{s_0} \vee f_{s_1} = (\neg b_0 \wedge \neg b_1 \wedge \neg b_2) \vee (b_0 \wedge \neg b_1 \wedge \neg b_2) = \neg b_1 \wedge \neg b_2$

$f_{T_2} = f_{s_1} \vee f_{s_2} = (b_0 \wedge \neg b_1 \wedge \neg b_2) \vee (\neg b_0 \wedge b_1 \wedge \neg b_2) = \neg b_2 \wedge (b_0 \vee b_1) \wedge (\neg b_1 \vee \neg b_0)$

$\mathbf{f_{T_1 \cap T_2}} = f_{T_1} \wedge f_{T_2} = ((\neg b_0 \wedge \neg b_1 \wedge \neg b_2) \vee (b_0 \wedge \neg b_1 \wedge \neg b_2)) \wedge ((b_0 \wedge \neg b_1 \wedge \neg b_2) \vee (\neg b_0 \wedge b_1 \wedge \neg b_2)) = b_0 \wedge \neg b_1 \wedge \neg b_2 = \mathbf{f_{s_1}}$

Rappresentando queste funzioni mediante OBDD riusciamo ad ottenere significativi guadagni di efficienza nella pratica

Rappresentazione della Relazione di Transizione

Vogliamo rappresentare anche la relazione di transizione δ tra stati come un OBDD

La relazione di transizione è un insieme. Precisamente: **è un sottoinsieme di** $S \times S$

Quindi possiamo associare ad ogni coppia $\langle s, s' \rangle \in \delta$ una codifica binaria e procedere come per gli stati

Mostriamo come farlo usando funzioni booleane, ma il passaggio ai ROBDD è immediato, se pensiamo che ogni funzione booleana è rappresentabile nella sua forma canonica come ROBDD

Rappr.ne della Relazione di Transizione (2)

Data una coppia di stati $\langle s, s' \rangle \in S \times S$, vogliamo costruire una funzione booleana f_δ che:

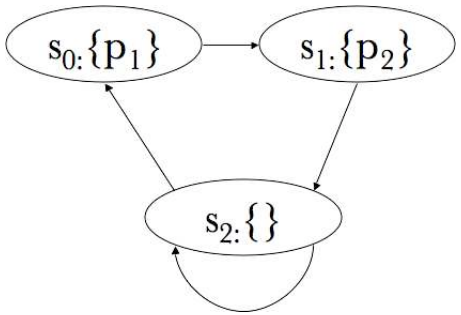
- prende in input la codifica binaria degli stati s ed s';
- restituisce 1 se e solo se $\langle s, s' \rangle \in \delta$

Usiamo la codifica degli stati indotta dalla funzione di labeling

Abbiamo bisogno di duplicare le variabili: una per il valore corrente ed una per il valore al passo successivo

Rappr.ne della Relazione di Transizione (3)

Esempio



p_1	p_2	p'_1	p'_2	δ
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	-
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	-
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	-
1	1	1	1	-

$$f_\delta = (\neg p_1 \wedge \neg p_2 \wedge \neg p'_1 \wedge \neg p'_2) \vee (\neg p_1 \wedge \neg p_2 \wedge p'_1 \wedge \neg p'_2) \vee (p_1 \wedge \neg p_2 \wedge \neg p'_1 \wedge p'_2) \vee (\neg p_1 \wedge p_2 \wedge \neg p'_1 \wedge \neg p'_2) =$$
$$(\neg p_1 \wedge \neg p'_1 \wedge \neg p'_2) \vee (\neg p_1 \wedge \neg p_2 \wedge p'_1) \vee (p_1 \wedge p'_2)$$

Rappr.ne della Relazione di Transizione (4)

Ovviamente, vogliamo evitare la costruzione della tabella di verità di δ

In pratica, la costruzione di f_δ viene fatta in maniera diretta a partire dalla specifica del KM

In SMV, la relazione di transizione viene specificata nella forma:

```
next(stato) :=
    case
        stato = nonInteressato : {nonInteressato, richiede};
        stato = richiede & !sem : acquisito;
        ...
    esac;

next(sem) :=
    case
        stato = richiede : 1;
        stato = rilasciato : 0;
        1 : sem;
    esac;
```

Per ogni istruzione `case`, possiamo scrivere una funzione relativa alla variabile assegnata nel corpo del case:

$f_{\texttt{stato}} = ((\texttt{stato=nonInteressato}) \wedge (\texttt{stato'=nonInteressato} \vee \texttt{stato' = richiede})) \wedge (\texttt{stato=richiede} \wedge \texttt{sem!=1} \wedge \texttt{stato'=acquisito}) \ldots$

$f_{\texttt{sem}} = (\texttt{stato=richiede} \wedge \texttt{sem'=1}) \wedge (\texttt{stato=rilasciato} \wedge \texttt{sem'=0}) \wedge (1 \wedge (sem' = sem))$

f_δ è ottenuta come congiunzione di tutte queste funzioni, una per ciascuna variabile