

Capitolo 4

Generazione automatica di vettori di collaudo

Michele Favalli

4.1 Introduzione

Il problema della generazione delle sequenze di collaudo per un circuito digitale rappresenta un punto centrale nell'ambito del *testing*, sia per le rilevanti problematiche teoriche che si incontrano nello sviluppo di adeguati algoritmi, sia perchè pone importanti condizionamenti sulla progettazione stessa dei circuiti (in termini di architettura e di sintesi).

Nel descrivere le procedure di generazione di vettori di collaudo, si farà riferimento allo schema concettuale introdotto nel Cap. sulla simulazione di guasti, in cui si verifica che il circuito sia privo di guasti.

In particolare, il punto di partenza per la maggior parte degli algoritmi di generazione automatica di vettori di collaudo (definiti come orientati ai guasti, *fault-oriented*) è una lista di guasti da rivelare, che è ottenibile dalla descrizione del circuito digitale e da un insieme di modelli di guasto. Nel generare vettori di collaudo per questi guasti, si ricerca un compromesso ottimale fra il costo delle operazioni necessarie per generare ed applicare i vettori di *test* e l'efficacia di tali sequenze, misurata in termini di copertura.

Più in dettaglio, il costo può essere ricondotto a due termini, il primo dei quali è costituito dai tempi di calcolo necessari per generare la sequenza di collaudo, mentre il secondo è dato dal tempo necessario per collaudare un componente durante la produzione, che è tipicamente proporzionale al numero di vettori di collaudo.¹

¹Nella teoria classica del collaudo la seconda componente del costo è sempre stata considerata come

Da un punto di vista teorico, quindi, l'obiettivo principale da raggiungere è quello di generare sequenze di lunghezza minima per un dato valore di copertura sull'insieme di guasti considerato. A causa della grande complessità dei circuiti che ha senso considerare, questa operazione deve essere svolta in modo automatico, mediante algoritmi che nel loro insieme costituiscono gli strumenti per la generazione automatica di sequenze di collaudo (*ATPG, Automatic Test Pattern Generator*).

A questo proposito, occorre rilevare come la ricerca di sequenze di lunghezza minima sia teoricamente concepibile, ma in pratica non affrontabile dal punto di vista dei costi di calcolo anche per circuiti di dimensioni medie.

Se quindi la ricerca della sequenza di collaudo di lunghezza minima non viene, in generale, perseguita, vengono comunque presi accorgimenti per ridurre la lunghezza dell'insieme di vettori generato. Fra questi accorgimenti merita menzione l'uso delle ipotesi introdotte nel capitolo sui modelli di guasto come, ad esempio, quella di guasti singoli e permanenti, nonché delle condizioni di dominanza fra i guasti. Da questo punto di vista, è l'utilizzo della simulazione di guasto che consente una notevole compattazione delle sequenze di collaudo generate.

Infatti, per ogni guasto non ancora rivelato, gli algoritmi che verranno illustrati nel seguito generano uno o più vettori di collaudo in grado di rivelarlo; tali vettori saranno tipicamente in grado di rivelare anche guasti diversi dall'obiettivo per cui sono stati generati, guasti che possono essere individuati mediante simulazione in modo da non richiedere la generazione di ulteriori vettori.

Nel loro complesso, algoritmi che operano nel modo descritto in precedenza vengono definiti come "deterministici", in quanto ogni vettore viene generato sulla base di un preciso obiettivo. Negli algoritmi di tipo casuale (*random*), invece, i vettori vengono generati in maniera casuale fino al raggiungimento di un dato valore di copertura, determinato mediante la simulazione di guasto. Un tipico schema per la generazione di vettori di collaudo è illustrato in Fig. 4.1.

Per quanto riguarda gli algoritmi che verranno illustrati nel seguito, si nota che questi fanno riferimento a un insieme di guasti di partenza ottenuti da diversi modelli di guasto, e a un livello di descrizione del circuito.

A quest'ultimo riguardo, sinora sono stati sviluppati algoritmi di generazione automatica di vettori di collaudo a partire dal livello *switch* fino a quello comportamentale.

Poiché i principi base di tali algoritmi sono comuni per i diversi livelli di descrizione circuitale, conviene considerare il caso più semplice costituito da guasti di tipo *stuck-at* in circuiti descritti a livello *gate*. che, analogamente a quanto visto nel caso della simulazione di guasto, è anche il più diffuso.

la più importante. Di qui notevoli sforzi per minimizzare il numero di vettori di collaudo. In realtà, questo rimane senz'altro vero in circuiti prodotti su larga scala, ma in determinate applicazioni (ASIC), in cui sono i tempi di sviluppo del prodotto ad essere critici, la prospettiva può cambiare radicalmente.

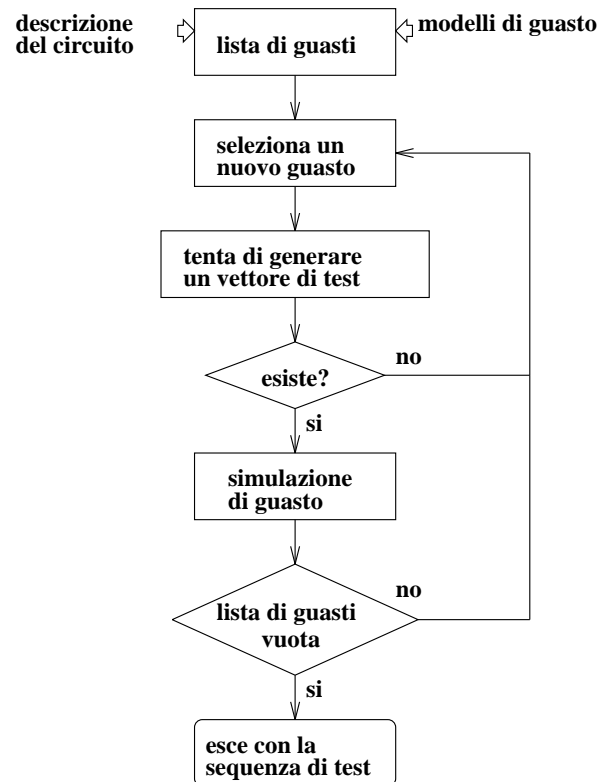


Figura 4.1: Tipico schema di flusso per un algoritmo di generazione di vettori di collaudo orientato ai guasti. Dopo che è stata generata una sequenza di collaudo per il guasto "obiettivo", la simulazione determina se altri guasti sono stati rivelati dalla stessa sequenza. Come si vede la generazione di vettori di collaudo per un dato guasto può non avere successo perchè: a) il guasto non è rivelabile; b) il costo di tale operazione risulta eccessivo.

In questa trattazione verrà considerato prima il caso di circuiti combinatori per poi passare quello, più complesso, delle reti sequenziali.

4.2 Algoritmi deterministici per circuiti combinatori

Le condizioni necessarie per la rivelazione di un guasto di tipo *stuck-at* in un circuito combinatorio sono date dall' *attivazione* (per cui, nel circuito privo di guasti, una linea sede del guasto deve assumere un valore opposto a quello a cui è bloccata per la presenza di uno *stuck-at*) e di *propagazione* (per cui gli effetti del guasto devono essere resi osservabili ad almeno un uscita del circuito).

Compito degli algoritmi di generazione automatica di vettori di collaudo è quello di ricercare fra le possibili configurazioni di ingresso al circuito le condizioni logiche che

permettano l'attivazione e la propagazione degli effetti del guasto. I valori assegnati alle linee del circuito per creare tali condizioni devono poi essere “*giustificati*” mediante opportuni assegnamenti alle linee del circuito. Gli effetti di ciascun assegnamento, nel caso di linee con $fan-out > 1$ devono poi essere propagati all'interno del circuito (*implicazione*) per determinare i vincoli posti ai passi successivi dell'algoritmo.

In questo caso, poi, a ogni passo dell'algoritmo devono essere compiute delle scelte (ad esempio su quale ramo di $fan-out$ propagare l'errore, oppure quale configurazione di ingresso a un *gate* utilizzare per giustificarne l'uscita), le quali possono dare luogo a delle incoerenze (ovvero ad assegnamenti contraddittori di un segnale).

In presenza di tali incoerenze, gli algoritmi si trovano a dover cambiare scelte fatte in precedenza, questo è il maggiore problema in termini di costi di calcolo richiesti dagli algoritmi di generazione di vettori di collaudo in quanto il numero di iterazioni necessarie per trovare un assegnamento coerente degli ingressi può anche essere molto grande.

A questo riguardo, sono stati sviluppati degli approcci euristici che a partire da un'analisi del circuito forniscono parametri utilizzabili dagli algoritmi per attuare le scelte caratterizzate da una maggiore probabilità di successo. Questi parametri sono spesso dati da misure di controllabilità (facilità con cui un segnale può essere portato al valore logico 0 o 1) e osservabilità (facilità con cui si possono osservare i segnali alle uscite del circuito) dei vari nodi del circuito.

Per esempio, se si vogliono propagare gli effetti di un guasto attraverso un nodo con $fan-out > 1$, conviene farlo attraverso il ramo caratterizzato dai migliori valori di osservabilità.

Per analizzare al meglio i problemi connessi a queste operazioni consideriamo due semplici esempi, nei quali non si farà riferimento a un preciso metodo di generazione automatica di vettori di collaudo, ma a una serie di operazioni che sono comunque comuni a diversi tipi di algoritmo (algoritmo-D, PODEM, FAN) e sono rappresentative dei problemi tipici di tali algoritmi.

Nel primo caso (Fig. 4.2) si considera un circuito combinatorio privo di riconvergenze di $fan-out$. Come si può vedere le condizioni per l'attivazione e la propagazione del guasto sono immediatamente identificabili, mentre l'assegnamento dei valori delle altre linee del circuito e degli ingressi per giustificare tali condizioni è molto semplice.

Il secondo esempio (Fig. 4.3), invece, contiene delle riconvergenze di $fan-out$ ed è possibile che alcune scelte fatte per propagare gli effetti del guasto (oppure per giustificare dei valori) diano luogo a conflitti in passi successivi di giustificazione. Questo problema, essenzialmente dovuto alla presenza di riconvergenza di $fan-out$ che rende fra loro correlati i diversi segnali del circuito, è particolarmente importante nella generazione di vettori di collaudo ed è in pratica quello che dà luogo al maggior costo

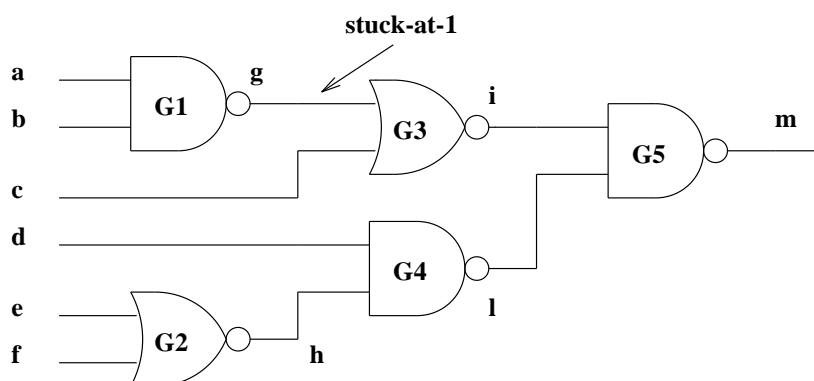


Figura 4.2: Esempio di generazione di un vettore di collaudo in un circuito combinatorio privo di riconvergenze di *fan-out*. Per attivare il guasto è necessario portare la linea **g** al valore 0 nel circuito privo di guasti, mentre la propagazione richiede $c = 0$ e $l = 1$. $g = 0$ può essere giustificato imponendo $a, b = 1$, mentre $l = 1$ è giustificabile imponendo ad esempio $d = 0$. Quindi applicando il vettore 1100XX si riesce a rivelare il guasto.

computazionale.

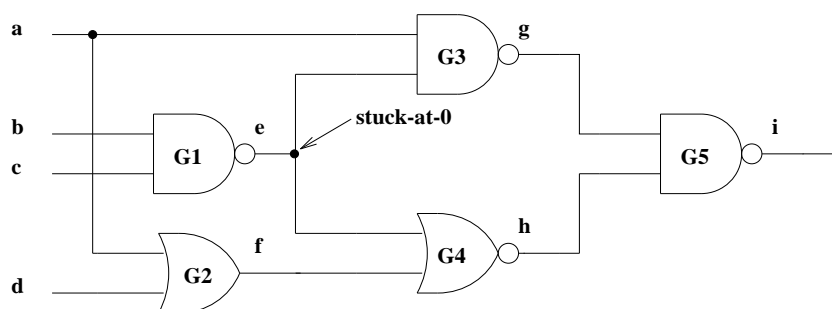


Figura 4.3: Esempio di generazione di un vettore di collaudo in un circuito combinatorio con riconvergenze di *fan-out*. Per attivare il guasto è necessario portare la linea **e** al valore 1 nel circuito privo di guasti, tale valore è giustificabile dall'assegnamento $b = 0$. Mentre la propagazione può essere fatta o attraverso **G3** o **G4**: nel primo caso è richiesto $a = 1$, ma tale assegnamento implica $b = 1$ e $h = 0$, quest'ultimo valore impedisce però la propagazione degli effetti del guasto attraverso **G5**. È quindi necessario cambiare la scelta fatta e propagare gli effetti del guasto attraverso **G4**, ciò richiede $f = 0$ che è giustificato da $a, d = 0$. $a = 0$ implica $g = 1$; tale valore consente comunque la propagazione degli effetti del guasto attraverso **G5**.

Inoltre, dopo ogni assegnamento di valore di una linea (sia nella fase di propagazione, che in quella di giustificazione), è necessario propagarne gli effetti in tutto il circuito.

Nel seguito verranno descritti alcuni dei principali metodi per la generazione di vettori di collaudo.

4.2.1 Metodo delle differenze booleane

Questo metodo sfrutta la conoscenza della dipendenza funzionale delle uscite di un circuito dagli ingressi e dai segnali interni ad esso per generare algebricamente i vettori di collaudo.

Siano, $\mathbf{x} = \{x_1, x_2, \dots, x_p\}$ e $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \{y_1, y_2, \dots, y_q\}$ i vettori delle variabili di ingresso e delle uscite di un circuito combinatorio in cui si vuole rivelare la presenza di un guasto di tipo *stuck-at* su un segnale interno g . Tale segnale, sarà a sua volta funzione degli ingressi $g = g(\mathbf{x})$.

Sotto queste ipotesi, si può trovare l'espressione che individua l'insieme dei vettori di collaudo in grado di rivelare tale guasto, traducendo in espressioni algebriche le condizioni per l'attivazione del guasto stesso e la propagazione dei suoi effetti.

In particolare, ai fini dell'attivazione un vettore di collaudo dovrà soddisfare l'equazione $g(\mathbf{x}) = 1/0$ (nel caso di guasto di tipo *stuck-at-0/1*).

Per quello che riguarda la propagazione degli effetti, è necessario che la linea g sia osservabile ad almeno un uscita y_k del circuito per la quale deve valere la relazione $y_k(\mathbf{x}, g = 0) \oplus y_k(\mathbf{x}, g = 1) = 1$ (dove $\mathbf{y} = \mathbf{f}(\mathbf{x}, g)$) rappresenta la dipendenza delle uscite da g che viene vista come variabile indipendente).

Quindi, l'insieme dei vettori di collaudo in grado di rivelare, per esempio, un guasto di tipo *stuck-at-0* è costituito dai vettori che soddisfano la relazione:

$$g(\mathbf{x}) \cdot (y_k(\mathbf{x}, g = 0) \oplus y_k(\mathbf{x}, g = 1)) = 1 \quad (4.1)$$

Nella sua versione originale, questo metodo era limitato a circuiti di dimensioni estremamente ridotte a causa dell'impossibilità di rappresentare ed elaborare in maniera efficiente espressioni booleane. Esso riveste attualmente maggiore importanza in quanto la rappresentazione di espressioni booleane mediante BDD (*Binary Decision Diagram*) ha, in parte, risolto questo tipo di problemi.

Come esempio si può considerare il circuito di Fig. 4.3. Con il guasto di tipo *stuck-at-1* sulla linea e , associando una variabile booleana a ciascun segnale l'uscita realizza la funzione $i = e + a + d$, mentre $e = (bc)'$. Quindi l'equazione 4.1 assume la forma

$$(bc)'((a + d) \oplus 1) = (b' + c')a'd' , \quad (4.2)$$

che rappresenta proprio l'insieme dei vettori di collaudo ($abcd = 0000, 0100, 0010$) in grado di rivelare guasto ipotizzato.

4.3 L'algoritmo-D

L'algoritmo-D è stato il primo metodo in grado di fornire un vettore di collaudo, qualora esistente, per un dato guasto obiettivo di tipo *stuck-at* (sempre nel caso di circuiti combinatori). Per la determinazione delle condizioni di attivazione e propagazione, l'algoritmo utilizza un formalismo in cui il simbolo D rappresenta i cambiamenti di valore delle linee dovuti alla presenza del guasto. In particolare, una linea assume il valore D se nel circuito privo di guasti vale 1 e in quello guasto 0. La situazione opposta viene invece denotata dal simbolo \overline{D} .

Oltre ai simboli D e \overline{D} vengono utilizzati i valori logici 0 e 1 e il simbolo X per denotare linee il cui valore non è ancora stato assegnato dall'algoritmo.

Quest'ultimo inizia assegnando al valore D (nel caso di un *stuck-at-0*) alla linea sede del guasto e prosegue propagando tale valore verso le uscite primarie, risolvendo contemporaneamente i problemi di giustificazione che si pongono.

Nel fare questo, l'algoritmo-D utilizza un approccio formale che ben si presta ad essere implementato in un efficiente programma di calcolo e che verrà illustrato in seguito con maggiore dettaglio.

A questo proposito, conviene introdurre alcune definizioni che fanno riferimento a cubi, ovvero a n-ple di valori dell'algebra $(0, 1, X, D, \overline{D})$, ciascuno dei quali corrispondente a un segnale del circuito considerato:

- il D-cubo di propagazione (*propagation D-cube*) di un generico *gate* è dato dall'insieme delle configurazioni di ingresso in grado di riportare la presenza di un simbolo D in ingresso. Come esempio, è riportata di seguito la tabella della verità di un *gate* AND a due ingressi:

	0	1	X	D	\overline{D}
0	0	0	0	0	0
1	0	1	X	D	\overline{D}
X	0	X	X	X	X
D	0	D	X	D	0
\overline{D}	0	\overline{D}	X	0	\overline{D}

gli incroci con i simboli D e \overline{D} corrispondono ai *propagation D-cube* del *gate*:

ingressi	uscita
1 D	D
1 \overline{D}	\overline{D}
D 1	D
\overline{D} 1	\overline{D}
D D	D
\overline{D} \overline{D}	\overline{D}

- il D-cubo primitivo (*primitive D-cube*) per un *gate* e un guasto rappresenta le condizioni di attivazione del guasto. Sempre nel caso di un *gate* AND a due ingressi, i D-cubi primitivi per i possibili guasti sono dati da:

ingressi	uscita	guasto
0 0	\overline{D}	uscita <i>stuck-at-1</i>
0 1	\overline{D}	
1 0	\overline{D}	
1 1	D	uscita <i>stuck-at-0</i>
\overline{D} 1	\overline{D}	ingresso <i>stuck-at-1</i>
1 \overline{D}	\overline{D}	ingresso <i>stuck-at-1</i>

- La copertura singolare (*singular cover*) di un *gate* rappresenta le condizioni sugli ingressi che possono dare luogo ai valori logici 0 e 1 e serve durante la giustificazione. Ad esempio, nel caso di un AND *gate* si ha:

ingressi	uscita
0 X	0
X 0	0
1 1	1

Queste definizioni vengono utilizzate dall'algoritmo-D nel seguente modo: per prima cosa vengono calcolati i cubi di propagazione (sia per D che per \overline{D}) di tutti i *gate* presenti nel circuito (ponendo a X le linee non interessate da tali condizioni locali di propagazione). Per quello che riguarda la fase di giustificazione, invece, viene creata una tabella che contiene la *singular cover* di ciascun *gate*.

In pratica, queste due tabelle riassumono tutte le informazioni sulla struttura del circuito. Per seguire meglio i passi più rilevanti nell'esecuzione dell'algoritmo-D, si consideri l'esempio di Fig. 4.4, per i quali la tabella con i cubi di propagazione e quella con la *singular cover* sono illustrate in Tab. 4.1 e 4.2, rispettivamente.

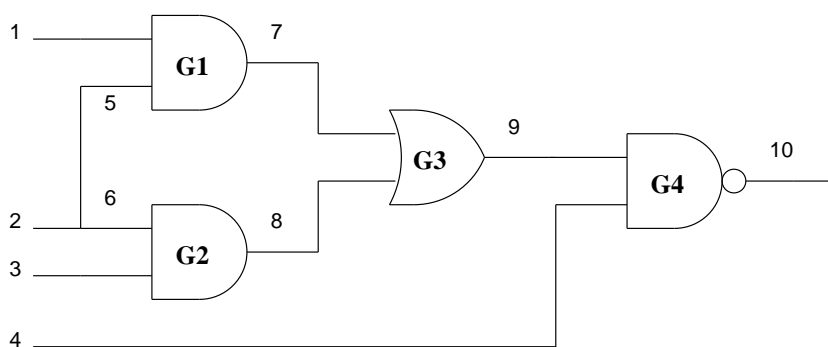


Figura 4.4: Esempio di rete combinatoria contenente riconvergenze di *fan-out*, per l'applicazione dell'algoritmo D.

gate	cubo	linea									
		1	2	3	4	5	6	7	8	9	10
G1	c_0	1				D		D			
	c_1	D				1		D			
G2	c_3			1			D		D		
	c_4			D			1		D		
G3	c_5							0	D	D	
	c_6							D	0	D	
G4	c_7				1					D	\overline{D}
	c_8				D					1	\overline{D}

Tabella 4.1: Tabella con i D-cubi di propagazione per ciascun gate del circuito di Fig. 4.4. Per rendere leggibile la tabella i valori X sono rappresentati da spazi bianchi. Per compattezza, nella tabella sono rappresentati solo i cubi di propagazione per il simbolo D , è comunque immediato dedurre quelli per il simbolo \overline{D} .

A questo punto, il processo di generazione automatica di vettori di collaudo parte da un cubo che ha tutte le linee a un valore X tranne quelle coinvolte nel cubo primitivo del guasto considerato.

Per illustrare questi concetti, si consideri, a titolo di esempio, di voler rivelare l'uscita di **G1** *stuck-at-0*. In questo caso il cubo iniziale che rappresenta le condizioni (locali) per l'attivazione del guasto è dato da:

gate	cubo	linea									
		1	2	3	4	5	6	7	8	9	10
G1	c_8	0				X		0			
	c_9	X				0		0			
	c_{10}	1				1		1			
G2	c_{11}			X			0		0		
	c_{12}			0			X		0		
	c_{13}			1			1		1		
G3	c_{14}							X	1	1	
	c_{15}							1	X	1	
	c_{16}							0	0	0	
G4	c_{16}				0					X	1
	c_{17}				X					0	1
	c_{18}				1					1	0

Tabella 4.2: Tabella con le *singular cover* per ciascun gate del circuito di Fig. 4.4. Per rendere leggibile la tabella i valori X sono rappresentati da spazi bianchi.

cubo	linea									
	1	2	3	4	5	6	7	8	9	10
c_{19}	1	1	X	X	1	X	D	X	X	X

L'algoritmo procede poi assegnando a 0, 1, D , \overline{D} , i valori ancora indeterminati fino a far apparire un valore D o \overline{D} su una delle uscite primarie del circuito. A questo punto i valori assegnati alle linee di ingresso costituiscono un vettore di collaudo per il guasto di partenza.

In questo processo, si sceglie un cammino attraverso cui propagare gli effetti del guasto. Il primo *gate* attraverso cui viene propagato il simbolo D (o \overline{D}) fornisce un cubo di propagazione, che viene combinato con quello corrispondente al cubo primitivo del guasto attraverso un operazione chiamata intersezione di cubi (*D-cube intersection*). Che viene attuata linea per linea secondo le regole fornite nella tabella seguente.

	0	1	X	D	\overline{D}
0	0	\emptyset	0	\emptyset	\emptyset
1	\emptyset	1	X	\emptyset	\emptyset
X	0	1	X	D	\overline{D}
D	\emptyset	\emptyset	D	D	λ
\overline{D}	\emptyset	\emptyset	\overline{D}	λ	\overline{D}

Chiaramente, quando si combinano due cubi si possono avere condizioni di incoerenza quando, ad esempio, la stessa linea compare in un cubo con il valore 1 e in un altro con il valore 0. Nella tabella precedente questo è rappresentato dal simbolo \emptyset (che indica un insieme vuoto). In questo caso, l'algoritmo D deve ritornare su l'ultima scelta attuata e cambiarla. Il simbolo λ rappresenta, in linea di principio un'incoerenza che può però essere eliminata in taluni casi utilizzando un D -cubo di propagazione per \overline{D} anziché D .

Nell'esempio, il primo *gate* attraverso cui propagare gli effetti del guasto è **G3**, per cui il cubo c_{19} deve essere intersecato con quello ottenuto dalla tabella dei cubi di propagazione (c_6) ottenendo così :

cubo	linea									
	1	2	3	4	5	6	7	8	9	10
c_{20}	1	1	X	X	1	1	D	0	D	X

Al passo successivo la propagazione avviene attraverso **G4** (cubo c_7) e si ottiene:

cubo	linea									
	1	2	3	4	5	6	7	8	9	10
c_{21}	1	1	X	1	1	1	D	0	D	D

Operazioni analoghe vengono svolte per giustificare i valori assegnati durante la propagazione utilizzando la tabella di copertura singolare del circuito.

Nell'esempio, l'unico valore da giustificare è l'uscita di **G2** (linea 8), mediante l'ingresso (3), il cui valore si ottiene intersecando il cubo c_{21} con c_{13} :

cubo	linea									
	1	2	3	4	5	6	7	8	9	10
c_{22}	1	1	0	1	1	1	D	0	D	D

L'esempio svolto è particolarmente semplice. Nella realtà, tuttavia, spesso le scelte che occorre operare si rivelano incompatibili, per cui è necessario iterativamente ritornare all'ultima scelta attuata e cambiarla fino ad ottenere un assegnamento coerente.

4.3.1 PODEM

L'algoritmo D presenta notevoli svantaggi quando viene utilizzato per reti che presentano un numero notevole di possibili alternative per la giustificazione e la propagazione, molte delle quali possono dare luogo a configurazioni incompatibili e a un numero molto grande di iterazioni. In particolare, questo è il caso delle reti che sono realizzate mediante *gate* di tipo EXOR (ad esempio le reti ECAT che sono tipicamente generatori rivelatori di parità) e che presentano un numero notevole di riconvergenze di *fan-out*. Il problema specifico dell'algoritmo D in questi casi, consiste nel fatto che esso esplora tutte le possibili alternative in fase di giustificazione prima di correggere le scelte fatte durante la fase di propagazione.

Per risolvere questi problemi, è stato realizzato l'algoritmo PODEM (*Path Oriented DEcision Making*) che risulta, in generale, più efficiente dell'algoritmo D. PODEM è il primo algoritmo che imposta il problema della generazione di vettori di collaudo come un'esplorazione dello spazio delle possibili configurazioni di ingresso. Tale esplorazione viene fatta mediante classiche metodologie di tipo *branch-and-bound*, di cui in Fig. 4.5 è riportato un esempio.

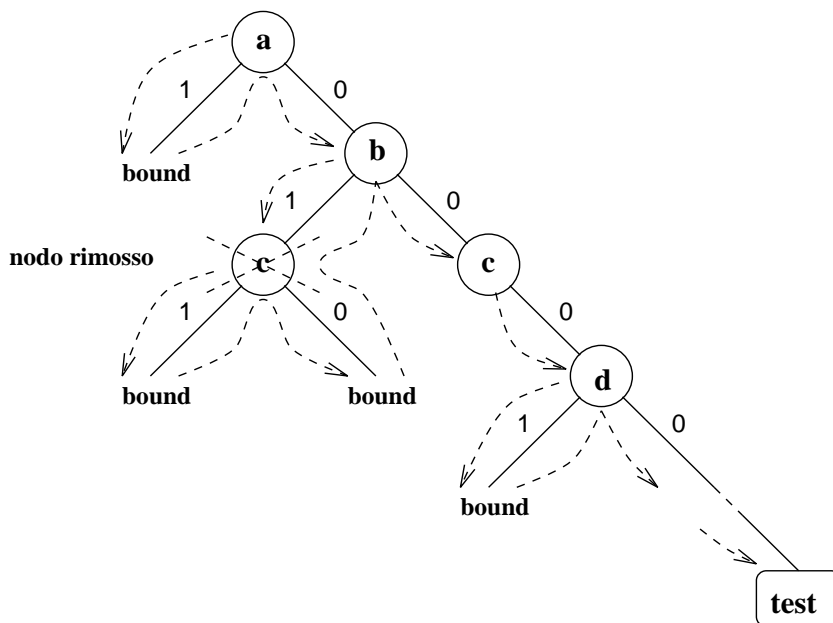


Figura 4.5: Possibile albero di decisione corrispondente all'esecuzione di un algoritmo di *branch-and-bound*. L'albero viene esplorato nella sequenza indicata dalle frecce.

In pratica, PODEM inizia assegnando tutte le linee di ingresso al valore X, poi una di queste viene assegnata a un valore (0 o 1) determinando (in maniera simile alla simulazione logica) gli effetti di tale assegnamento sullo stato del circuito e in particolare

sulla propagazione dei valori D o \overline{D} . In questa maniera eventuali informazioni su assegnamenti incoerenti con gli obiettivi di attivazione e propagazione del guasto si rendono disponibili molto prima che nell'algoritmo D .

Come si vedrà in seguito, ciascun assegnamento può dare luogo a condizioni che rendono impossibile la rivelazione del guasto. Se questo non avviene si procede assegnando altri ingressi. Questo processo, denominato *branching*, viene ripetuto fino a quando: a) l'attivazione o la propagazione non sono più possibili; b) il valore D (o \overline{D}) viene giustificato e propagato ad almeno un uscita primaria del circuito.

Gli assegnamenti possono risultare incompatibili con la propagazione del guasto (condizione a)), perchè: a.a) il guasto non viene attivato (perchè la linea sede del guasto assume un valore uguale a quello del circuito corretto); a.b) non c'è alcun cammino fra la sede del guasto e le uscite primarie attraverso cui sia possibile propagare il valore D . In questi casi, si torna all'ultimo ingresso assegnato e (se questa strada non è ancora stata percorsa) se ne complementa il valore (*bounding*), altrimenti si procede risalendo il processo di *branching* fino a trovare una scelta non ancora attuata, per poi ripartire. In pratica, il processo di *branch-and-bound* si configura come l'esplorazione di un albero binario.

Chiaramente, se durante il processo di *branching* la scelta della linea di ingresso da assegnare e del relativo valore viene attuata in maniera casuale, il numero di iterazioni dell'algoritmo risulta enorme essendo pari a 2^n (ove n è il numero di ingressi). Per risolvere questo problema, ciascuna scelta cerca di soddisfare un obiettivo, tentando in pratica di avvicinare lo stato del circuito a quello in cui si ha la rivelazione del guasto.

In particolare, viene prima perseguita l'attivazione del guasto e solamente in seguito si cerca di propagare D verso le uscite primarie del circuito. In pratica, ciascun obiettivo richiede, inizialmente, di portare a dati valori alcune linee del circuito. Ad esempio, se l'obiettivo iniziale consiste nel cercare di avvicinare D alle uscite del circuito, si analizza la D frontiera e si sceglie il *gate* più vicino (in termini di numero di livelli di logica, vedi Fig. 4.6) alle uscite del circuito. Per propagare D attraverso tale *gate* è necessario assegnare un certo valore alle sue linee di ingresso su cui non è presente il simbolo D (ad esempio nel caso di un AND esse devono venire assegnate al valore 1).

A questo punto, viene attivata una procedura di *backtrace*, che procede dal segnale "obiettivo" verso gli ingressi primari del circuito. Tale procedura prosegue fino a determinare un ingresso primario e un suo valore tali da massimizzare la probabilità di soddisfare l'obiettivo di partenza (si noti che questa procedura non tenta di giustificare l'obiettivo iniziale, ma si limita a cercare di indirizzare convenientemente il processo di *branch-and-bound*).

In pratica, a ogni passo della procedura di *backtrace* si ha come punto di partenza (o obiettivo corrente) l'uscita di un gate a un dato valore. Si tratta, quindi, di asseg-

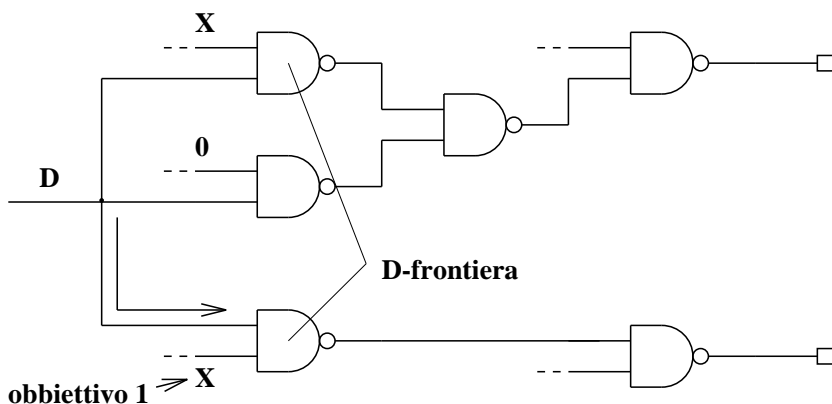


Figura 4.6: Esempio di scelta degli obiettivi da parte di PODEM per la propagazione degli effetti del guasto. Fra i *gate* che compongono la D-frontiera, si tenta di propagare *D* attraverso quello più vicino alle uscite primarie del circuito assegnando a 1 l'ingresso che si trovava a *X*. Tale valore è poi l'obiettivo iniziale per la procedura di *backtrace*, che tenta di individuare un assegnamento per la fase di *branching*.

nare opportunamente uno degli ingressi del *gate* che costituirà poi l'obiettivo del passo seguente.

Le scelte operate durante il *backtrace* utilizzano opportune misure di controllabilità, ovvero valori numerici che quantificano la facilità con cui un dato segnale può essere portato al valore logico 0 o 1 (queste possono essere date, ad esempio, dalla probabilità che un segnale si porti a 1 o a 0 con un assegnamento casuale degli ingressi del circuito).

Per vedere come le misure di controllabilità vengono applicate durante il processo di *backtrace* si considerino i seguenti esempi. Si supponga che l'obiettivo sia un 1 sull'uscita di un AND, in questo caso un algoritmo di giustificazione tenterebbe di portare tutti gli ingressi a 1, la procedura di *backtrace* di PODEM è, invece, molto più semplice e assegna un segnale per volta. In particolare, seleziona quello più difficile da portare a 1 (*"hardest"*, cioè quello con il minore valore di controllabilità a 1 fra i diversi ingressi del *gate*) e lo pone come nuovo obiettivo. Questo, perchè conviene tentare l'assegnamento più problematico rivelando così eventuali incoerenze prima di avere assegnato tutti gli altri ingressi del *gate*. Se, invece, l'uscita del *gate* AND deve essere portata a 0, PODEM sceglie l'ingresso più facile da controllare a 0, questo perchè diversamente dal caso precedente è sufficiente portare un ingresso al valore logico 0 per determinare il valore dell'uscita.

Come esempio di procedura di *backtrace* si consideri Fig. 4.7, ove come misura di controllabilità si è utilizzata la probabilità di segnale. Come si vede, l'obiettivo iniziale è di portare a 0 l'uscita del NAND **GR**. Per fare questo si sceglie l'ingresso con il valore più basso di controllabilità a 1, che in questo caso corrisponde all'uscita di **GQ**. Per

portare a 1 l'uscita di quest'ultimo *gate* è sufficiente portare a 0 uno degli ingressi: in questo caso si sceglie la linea meglio controllabile (ingresso **P**).

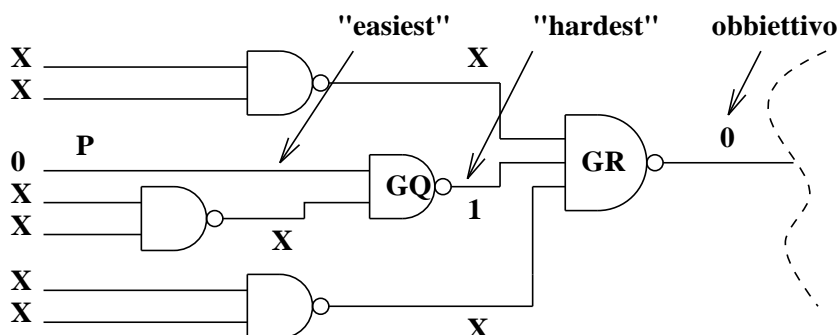


Figura 4.7: Esempio della procedura di *backtrace* di PODEM.

Di seguito è riportato un esempio di applicazione di PODEM (Fig. 4.8). Tale esempio è particolarmente semplice e quindi non permette di vedere i passi svolti durante il *branching* nella scelta degli ingressi e dei loro valori, per cui gli ingressi verranno semplicemente esplorati in un ordine arbitrario. L'esempio rimane comunque significativo per quello che riguarda la struttura dell'albero di decisioni utilizzato per esplorare le possibili configurazioni di ingresso Fig. 4.9.

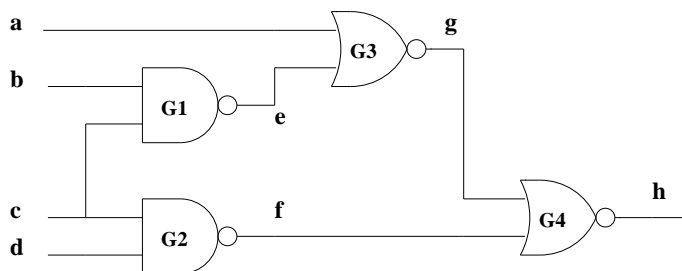


Figura 4.8: Esempio di circuito per l'applicazione di PODEM.

4.3.2 FAN

L'esplorazione dello spazio delle possibili configurazioni di ingresso con un algoritmo di *branch-and-bound* costituisce il maggiore vantaggio di PODEM, in quanto, rispetto all'algoritmo-D, consente di verificare molto più rapidamente la coerenza degli assegnamenti eseguiti. D'altra parte, gli approcci euristici utilizzati durante la fase di *branching* sono eccessivamente semplici, perchè non tengono conto di diverse condizioni in grado di ridurre notevolmente il numero delle iterazioni necessarie per arrivare alla definizione di un vettore di collaudo.

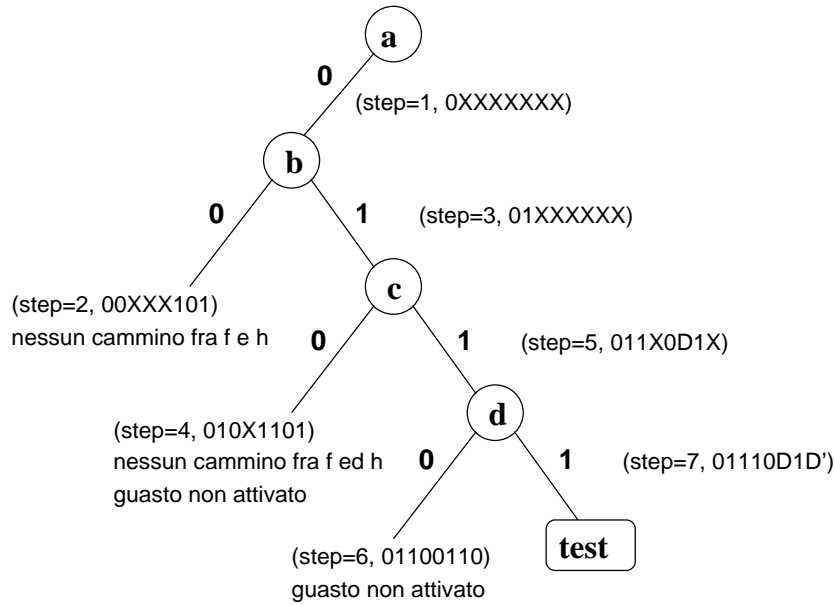


Figura 4.9: Rappresentazione grafica (albero di decisioni) dei passi svolti da PODEM per generare un vettore di collaudo per un guasto *stuck-at-1* sulla linea **f** del circuito di Fig. 4.8. Come si vede, i passi di *branching* 2, 4 e 6 danno luogo a condizioni di *bound* sia per ciò che riguarda l'attivazione che la propagazione.

Per sfruttare al meglio le possibilità del *branch-and-bound* è stato sviluppato FAN, che raggiunge una migliore efficienza di PODEM per diversi circuiti, in quanto tenta di sfruttare al meglio la conoscenza della topologia del circuito per ridurre il numero di iterazioni, semplificare i passaggi intermedi e determinare l'inesistenza della soluzione il più presto possibile.

Per comprendere i vantaggi di FAN, è conveniente introdurre alcune definizioni:

1. una linea viene definita come *bound-line* se può essere raggiunta da un punto di *fan-out* (ovvero un nodo con *fan-out* > 1);
2. le linee non di tipo *bound* sono definite come *free-line*;
3. una *free line* che è in ingresso allo stesso *gate* di una *bound line*, oppure l'uscita di un *gate* che pilota dei rami di *fan-out* (i quali sono *bound line*) si dice adiacente a una *bound line* e viene definita come *head line*.

La Fig. 4.10 illustra l'applicazione di queste definizioni al caso di un semplice circuito.

Come si può vedere le *head line* sono le uscite di reti prive di *fan-out*, per cui il valore di ciascuna di esse può essere assegnato senza cambiare il valore di linee nel

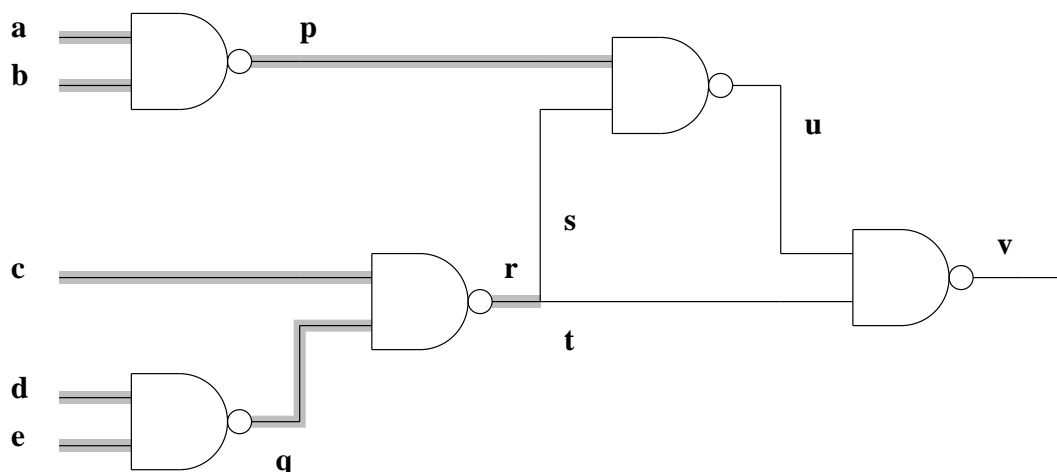


Figura 4.10: Esempio delle definizioni utilizzate da FAN per caratterizzare un circuito combinatorio dal punto di vista topologico: le *free-line*, indicate in tratteggio, corrispondono a regioni prive di *fan-out* del circuito (**a**, **b**, **p**, **c**, **r**, **d**, **e**, **q**); le linee a valle di nodi con *fan-out* > 1 si dicono *bound-line* (**s**, **t**, **u**, **v**); le free-line **p** ed **r** sono *head-line* in quanto, rispettivamente, linee di *fan-in* di gate con *bound-lines* come altri ingressi e linee a *fan-out* > 1.

circuito diverse da quelle comprese nel cono di *fan-in* della *head line*. Pertanto, una volta assegnato il valore a una *head-line*, le operazioni di *backtrace* (*branch and bound*) nell'albero di decisioni necessarie per giustificarlo, si possono fermare alla *head line* senza bisogno di iterare sugli ingressi che possono essere assegnati in maniera univoca.

Si supponga, ad esempio, di volere controllare la (*head-line*) **r** al valore logico 0, questo può essere fatto con un semplice algoritmo che partendo dalla linea in esame, procede assegnando valori agli ingressi di ciascun gate tali da giustificarne il valore dell'uscita fino a raggiungere gli ingressi del circuito che controllano tale linea (in pratica, si assegna **c** = 1, **q** = 1 e poi **d** = 0). In questo modo, si ha un assegnamento coerente degli ingressi, invece, in una procedura di *branch and bound*, se il valore 0 fosse stato assegnato all'ingresso **c** si sarebbe avuta un'incoerenza.

Un ulteriore vantaggio di FAN è il modo in cui tratta i cammini attraverso cui vengono propagati gli effetti del guasto. Per capire questo aspetto del problema, si consideri un tipico esempio delle operazioni svolte da PODEM nel caso della parte di circuito illustrata in Fig. 4.11

Si supponga che la D-frontiera sia costituita da **a** (**a** = *D*). Per propagare il segnale *D* attraverso **GA** (**c** = \overline{D}), è necessario avere **b** = 1. A questo punto, PODEM esegue la procedura di *backtrace* per individuare un opportuno assegnamento degli ingressi. Se tale assegnamento, oltre a giustificare **b** = 1, implica anche **f** = 0, gli effetti del guasto

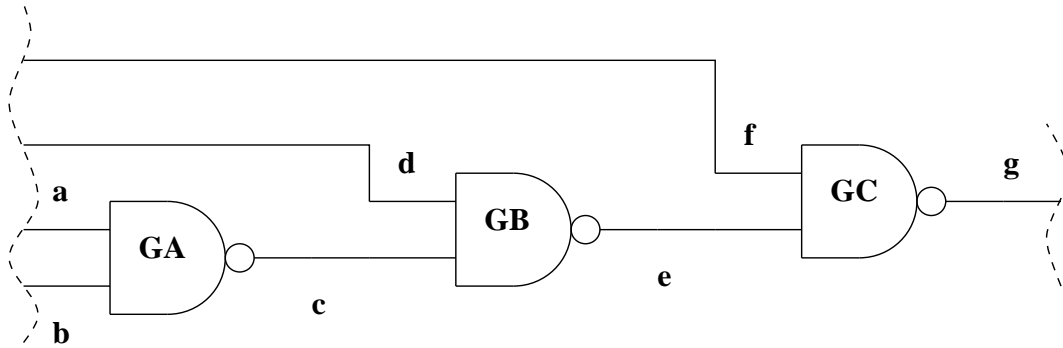


Figura 4.11: Circuit used to instantiate D value propagation in FAN.

non possono essere propagati oltre **GC**. PODEM non si accorge, però del problema e pertanto tenta inutilmente di propagare \bar{D} attraverso **GB** ($e = D$).

Per risolvere questo problema, FAN individua il più lungo cammino attraverso cui D può essere propagato senza che si abbiano delle scelte (ovvero dei nodi con *fan-out* > 1) e impone subito le condizioni (uniche) per la propagazione degli effetti del guasto lungo tale cammino (*unique sensitization*).

Nel caso di Fig. 4.11, infatti, FAN individua immediatamente che i *gate* **GA**, **GB**, **GC** costituiscono un *unique sensitization path* e, pertanto, assegna contemporaneamente $\mathbf{b}, \mathbf{d}, \mathbf{f} = 1$ così da avere $\mathbf{g} = \bar{D}$. A questo punto viene eseguita la procedura di *backtrace* (che come si vedrà nel seguito è diversa da quella utilizzata da PODEM) per assegnare nuovi valori agli ingressi. In questo modo eventuali incompatibilità vengono immediatamente individuate risparmiando operazioni inutili.

Per quello che riguarda, invece, le procedure di *backtrace* necessarie per individuare le linee di ingresso da assegnare, mentre PODEM a partire da un dato obiettivo seleziona una linea per volta, FAN esegue un *backtrace* multiplo individuando più linee per volta ed assegnando possibilmente più ingressi.

Per capire i vantaggi di questo approccio si consideri il circuito di Fig. 4.12 e si supponga che l'obiettivo sia quello di portare la linea **l** a 0.

Le operazioni svolte da PODEM nel *backtrace* sono illustrate in Fig. 4.12a e risultano nell'assegnamento dell'ingresso **b** a 0.

Questo assegnamento non è tale da giustificare il valore di **l** ed è possibile, che nei passi seguenti, altre linee di ingresso vengano assegnati a valori incoerenti con il valore di **l**.

L'algoritmo FAN, invece, (Fig. 4.12b) assegna tutti i valori univocamente determinati dall'obiettivo iniziale (eseguendo delle scelte ove invece queste condizioni non sono verificate, come nel caso di un NAND con uscita 1): pertanto queste incoerenze non possono essere generate.

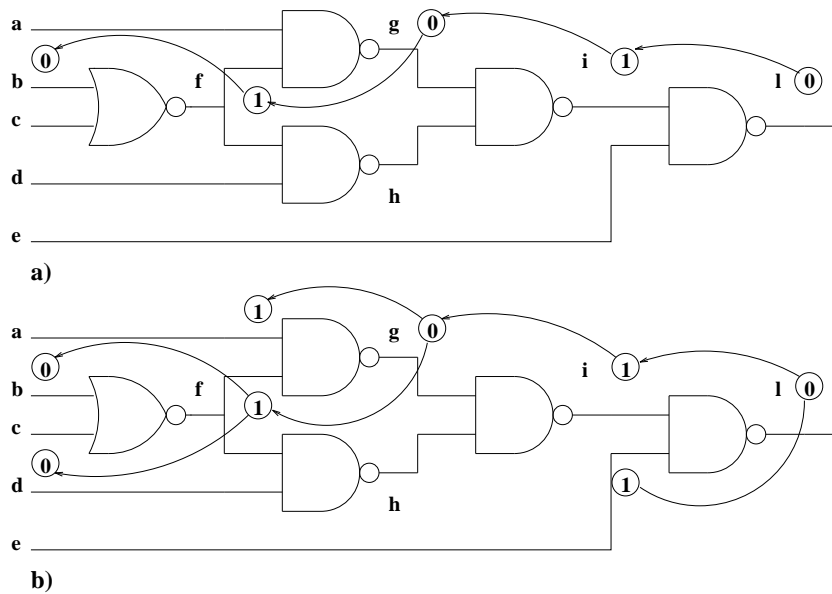


Figura 4.12: Esempio delle operazioni svolte per giustificare il valore 0 della linea l da: a) PODEM, che per ciascun *gate* determina un solo valore di ingresso (il più facile da controllare se l'uscita del *gate* NAND è a 1 e il più difficile viceversa 0); b) FAN, che, quando per un dato valore di uscita del *gate* (0 nel caso del NAND) tutti gli ingressi sono determinati, li assegna dando così luogo a cammini multipli di *backtrace*.

4.4 Pseudorandom Testing

Nelle metodologie di collaudo di tipo casuale, i vettori di collaudo vengono generati in maniera casuale o (pseudocasuale, la distinzione fra i due termini verrà chiarita nel seguito di questo paragrafo) a partire da una data distribuzione di probabilità. In particolare, se tale distribuzione è uniforme (ovvero se ogni vettore di collaudo ha la stessa probabilità di essere generato, e quindi ciascun ingresso ha una distribuzione di probabilità di valere 1 indipendente da quella degli altri ingressi e con valore atteso pari a 0.5), si parla propriamente di generazione di vettori di collaudo di tipo *random*, se invece tale distribuzione non è uniforme (ad esempio se ciascun ingresso ha probabilità diversa da 0.5) si parla di generazione *random* pesata (*weighted random testing*).

Rispetto alla generazione di vettori di collaudo di tipo deterministico, la generazione casuale presenta il vantaggio di una maggiore semplicità degli algoritmi utilizzati e lo svantaggio di una minore efficienza, che risulta in maggiori lunghezze delle sequenze di collaudo a parità di copertura (o in minori valori di copertura a parità di lunghezza della sequenza di collaudo).

Una delle maggiori applicazioni di questa metodologia è data dalla possibilità di integrare direttamente sul chip generatori di vettori di collaudo di questo tipo che

risultano molto più compatti (essendo tipicamente costituiti da registri a scorrimento retroazionati linearmente) di quelli richiesti da tecniche di tipo deterministico (in pratica, delle ROM in cui memorizzare i vettori di collaudo generati algebricamente).

Le sequenze generate in questo caso si dicono più propriamente di tipo pseudocasuale, in quanto, gli algoritmi di generazione di vettori di collaudo che possono essere integrati sono in grado di fornire sequenze che solo in parte approssimano le caratteristiche di sequenze casuali. Infatti, seppure la probabilità di valere 1 di ciascun segnale nelle sequenze pseudocasuali è approssimativamente pari a 0.5, esiste sia correlazione fra i diversi segnali di ingresso che correlazione fra i diversi vettori che vengono generati. In pratica, una volta generato un vettore, questo non può ripresentarsi e le caratteristiche del processo di generazione dei vettori di collaudo diventano quindi simili a quelle di un esperimento senza reintroduzione, se poi la lunghezza della sequenza è maggiore di quella massima di conteggio del circuito utilizzato per generarla, si hanno anche caratteristiche di periodicità.

Comunque, se la sequenza utilizzata è sufficientemente più corta di quella massima ottenibile dal registro utilizzato per la generazione di vettori, allora le caratteristiche statistiche sono molto simili a quelle di una sequenza *random*. Per questo motivo e per rendere più semplice la trattazione, quando non diversamente specificato, si farà riferimento al collaudo di tipo casuale.

Anche se particolarmente utile nel caso di circuiti in grado di autocollaudarsi, la generazione di vettori di collaudo di tipo casuale può essere utilizzata in alternativa agli algoritmi di tipo deterministico anche nel caso in cui i vettori siano generati con programmi di simulazione. In questo caso, si vedrà come l'efficienza rispetto ai programmi deterministici sia piuttosto ridotta e quindi siano necessari ulteriori accorgimenti.

Per quello che riguarda il soddisfacimento di obiettivi di qualità da parte di sequenze di tipo casuale (punto chiave per questo tipo di metodologie), anche in questo caso l'obiettivo da soddisfare è tipicamente dato da un valore di copertura di guasti, mentre il costo è quasi unicamente dato dalla lunghezza della sequenza da generare, perchè il contributo della parte computazionali è estremamente ridotto.

Eseguendo la simulazione di guasto per ciascun vettore di collaudo generato (sia da un programma che da un circuito) si conosce immediatamente il valore corrente di copertura di guasti e la generazione di vettori casuali può essere interrotta una volta raggiunto un obiettivo prefissato.

Questa procedura, in apparenza semplice, può presentare dei notevoli problemi. In particolare, per alcuni circuiti, all'aumentare del numero di vettori *random* applicati la copertura aumenta molto lentamente, per cui possono essere necessarie sequenze molto lunghe e conseguentemente costi di simulazione e collaudo molto alti.

A questo riguardo, Fig. 4.13 illustra un caso tipico dove la copertura cresce lenta-

mente e rimane bassa anche per sequenze di vettori casuali estremamente lunghi.

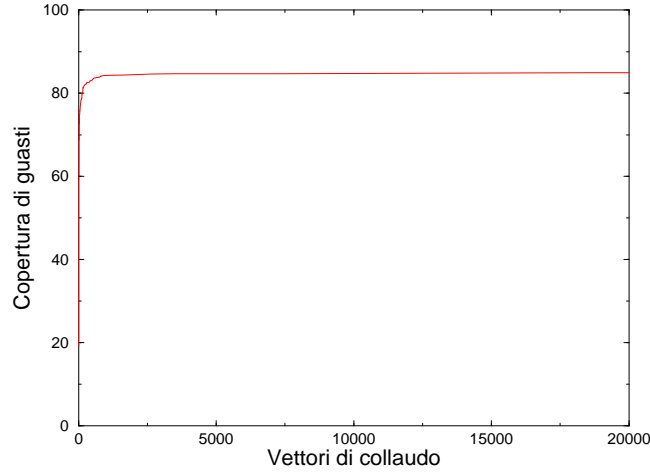


Figura 4.13: Tipico andamento della copertura di guasti in funzione del numero di vettori di una sequenza casuale in un circuito difficilmente collaudabile con queste tecniche. La copertura, una volta raggiunto un certo valore cresce molto lentamente.

Per comprendere meglio questo tipo di problemi, è necessario considerare gli aspetti probabilistici della rivelazione di guasti mediante sequenze di tipo pseudocasuale. Questa trattazione sarà ristretta al caso di guasti di tipo *stuck-at* in reti combinatorie (nel caso di reti sequenziali il collaudo di tipo pseudocasuale è abbastanza inefficace). A questo scopo, è necessario introdurre alcune definizioni:

- d_k probabilità di rivelare il guasto considerato (k) con un vettore generato casualmente (*detection probability*): $d_k = \frac{N_k}{2^n}$ dove N_k è il numero di vettori di collaudo in grado di rivelare il guasto k e n è il numero di ingressi del circuito considerato;

La probabilità di rivelare il guasto k per la prima volta al vettore di collaudo j risulta uguale al prodotto di quella di non avere rivelato il guasto con i primi $j - 1$ vettori di collaudo con quella di rivelarlo al successivo:

$$D(k, j) = (1 - d_k)^{j-1} d_k \quad (4.3)$$

La probabilità di rivelare il guasto con l vettori di collaudo (detta anche *test confidence*) risulta essere espressa da:

$$G(k, l) = \sum_{j=1}^l (1 - d_k)^{j-1} d_k = 1 - (1 - d_k)^l. \quad (4.4)$$

La seconda eguaglianza può essere spiegata semplicemente notando che tale probabilità (poichè le prove sono fra loro indipendenti) è uguale al complemento della probabilità che nessuno dei vettori applicati riveli il guasto.

Se il valore di d_k è molto piccolo, per avere un valore di *test confidence* elevato sono necessari valori sequenze molto lunghe; in particolare, se il prodotto fra d_k e l è $\ll 1$ allora, $G(k, l)$ può essere approssimata dalla seguente espressione:

$$G(k, l) = 1 - e^{-d_k l} . \quad (4.5)$$

Se si vuole, quindi, determinare il numero di vettori di collaudo necessario per rivelare il guasto con una data probabilità G si ha dall'eq. 4.4:

$$l = \frac{\ln(1 - G)}{\ln(1 - d_k)} . \quad (4.6)$$

A questo punto, è interessante ricavare l'espressione del valore atteso della copertura di guasti per un dato insieme di guasti e una lunghezza l di sequenza *random*. Sia $r_k(l)$ una variabile aleatoria che vale 0 se il guasto non è rivelato dalla sequenza applicata e 1 se è rivelato. La copertura di guasti ($C(l)$) è anch'essa una variabile aleatoria (che dipende dalla specifica sequenza applicata) esprimibile come:

$$C(l) = \frac{N_d(l)}{N_{tot}} = \sum_{k=1}^{N_{tot}} r_k(l) \quad (4.7)$$

Tenendo conto che il valore atteso di $r_k(l)$ è dato dalla probabilità ($G(k, l)$) di rivelare il guasto, con una sequenza di lunghezza l , si ha:

$$E[C(l)] = E\left[\sum_{k=1}^{N_{tot}} r_k\right] = \frac{1}{N_{tot}} \sum_{k=1}^{N_{tot}} E[r_k] = \frac{1}{N_{tot}} \sum_{k=1}^{N_{tot}} G(k, l) \quad (4.8)$$

tenendo conto dell' eq. 4.4 si ha:

$$E[C(l)] = \frac{1}{N_{tot}} \sum_{k=1}^{N_{tot}} 1 - (1 - d_k)^l = 1 - \frac{1}{N_{tot}} \sum_{k=1}^{N_{tot}} (1 - d_k)^l \quad (4.9)$$

in cui il termine $\sum_{k=1}^{N_{tot}} (1 - d_k)^l$ che rappresenta la percentuale attesa di guasti che non sono rivelati da una sequenza casuale di lunghezza l è dominato dai guasti caratterizzati dai valori più bassi di *detection probability* ed è tanto maggiore, tanto più i guasti di questo tipo che costituiscono il principale problema del collaudo pseudocasuale (e vengono definiti come *random resistant*) sono numerosi.

La conoscenza del valore di *detection probability* per ciascun guasto consente, mediante l'eq. 4.9, di determinare la copertura di guasti per una data lunghezza della sequenza di collaudo, senza bisogno quindi eseguire lunghe e costose simulazioni di guasto. Inoltre, eliminando dal circuito, mediante tecniche di DFT, i guasti *random resistant* è possibile ridurre in maniera significativa la lunghezza delle sequenze di collaudo

sia nel caso di circuiti BIST che di generazione software di vettori di collaudo (come alternativa, in questo caso si può ricorrere a una generazione di tipo deterministico per i guasti non rivelati dalla sequenza casuale).

D'altra parte, la valutazione esatta di d_k per ciascun guasto implicherebbe la simulazione esaustiva del circuito che è spesso non fattibile. Queste considerazioni hanno dato luogo alla ricerca di metodi approssimati per determinare la *detection probability* di ciascun guasto. Questi metodi, che rientrano fra quelli denominati misure di collaudabilità approssimate (*approximate testability measures*), sono tipicamente basati su algoritmi che trattano il circuito in maniera probabilistica.

Sono stati sviluppati diversi metodi di questo tipo, tutti caratterizzati dal tentativo di raggiungere sufficienti livelli di accuratezza con costi di calcolo estremamente ridotti.

Per introdurre i concetti fondamentali utilizzati da tali algoritmi descriveremo il più semplice di questi denominato COP (*Circuit Observability Program*) che si basa su un'analisi probabilistica del circuito. Tale analisi trascura le correlazioni fra i diversi segnali del circuito (e quindi fornisce un risultato esatto solo nel caso in cui la rete sia priva di riconvergenze di fan-out).

In questa ottica, la probabilità che un guasto sia rivelato da un vettore di tipo *random* è data dal prodotto di quella che il guasto sia attivato con quella (s_k) di propagarne gli effetti alle uscite del circuito (tale quantità è detta osservabilità).

Per determinare queste quantità, è necessario determinare la probabilità di valere 1 o 0 di ciascun segnale del circuito o controllabilità (per una linea g si definisce $c_g = Prob\{g = 1\}$). Si consideri, ad esempio, un AND con ingressi A e B , la probabilità di avere 1 in uscita (supponendo i segnali non correlati) è data dal prodotto delle due probabilità di avere 1 in ingresso; risulta quindi $c_{OUT} = c_{ACB}$.

Per determinare la controllabilità di ciascun segnale nel circuito, trattandosi di sequenze casuali, la probabilità degli ingressi viene assegnata a 0.5 e, quindi, l'algoritmo procede verso le uscite primarie del circuito assegnando il valore di controllabilità dell'uscita di ciascun gate a partire da quelli degli ingressi utilizzando le regole illustrate in Tab. 4.3 che rappresentano la descrizione probabilistica di ciascun gate nel caso in cui si considerano gli ingressi indipendenti.

L'esempio di Fig. 4.14 illustra il calcolo della controllabilità nel caso di un semplice circuito combinatorio con riconvergenza di fan-out. Come si può vedere la stima della controllabilità non è più esatta nel caso dell'uscita, infatti, tale valore viene stimato da COP come $1 - c_F c_G$, espressione che è valida solo se i segnali F e G sono indipendenti, ma tale ipotesi non è verificata nel circuito in esame in quanto questi segnali dipendono da C .

L'osservabilità delle diverse linee, invece, viene calcolata a partire dalle uscite del circuito cui viene attribuito il valore 1 e viene riportata verso gli ingressi determinando

gate	c_{OUT}
AND	$\prod_{\forall IN} c_{IN}$
OR	$1 - \prod_{\forall IN} (1 - c_{IN})$
NOT	$1 - c_{IN}$
NAND	$1 - \prod_{\forall IN} c_{IN}$
NOR	$\prod_{\forall IN} (1 - c_{IN})$

Tabella 4.3:

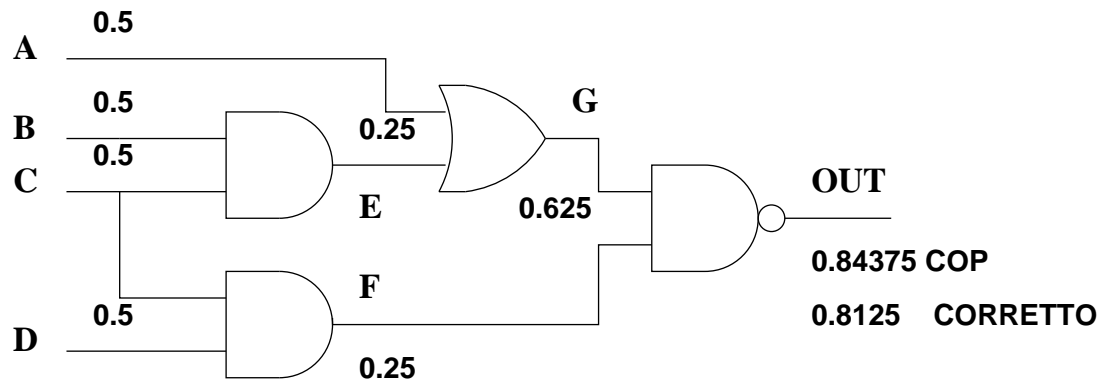


Figura 4.14: Valori di controllabilità (probabilità di avere 1 su un segnale) calcolati con COP per un semplice circuito combinatorio. Come si può vedere COP commette un errore nel caso del segnale di uscita su cui si ha riconvergenza di fan-out.

l'osservabilità di ciascun ingresso di gate secondo regole di tipo probabilistico. Quindi, con le usuali ipotesi di segnali non correlati, l'osservabilità di una linea di ingresso di un gate è uguale alla probabilità che gli effetti di una variazione di tale linea siano tali da cambiare l'uscita del gate per l'osservabilità della linea di uscita di tale gate. Le regole per calcolare l'osservabilità per l'ingresso J di un gate con un uscita G sono riportate in Tab. 4.4.

In questo modo si ha una stima della probabilità di propagare gli effetti di un guasto.

Si consideri ad esempio un gate AND la cui linea di uscita G ha osservabilità 0.6, e le controllabilità degli ingressi A e B sono rispettivamente 0.5 e 0.7; l'osservabilità della linea B è data dalla probabilità che A valga 1 per la probabilità che l'uscita sia osservabile ed è pari a 0.3.

Nel caso di una linea P con fan-out > 1 , la sua osservabilità può essere ricavata da

gate	c_{OUT}
AND, NAND	$s_J = s_G \prod_{\forall IN \neq J} c_{IN}$
OR, NOR	$s_J = s_G \prod_{\forall IN \neq J} (1 - c_{IN})$
NOT	$s_J = s_G$

Tabella 4.4: Regole di COP per calcolare l'osservabilità di un segnale di ingresso a una *gate*.

quella dei suoi rami assumendo che questi siano indipendenti e che la quindi la linea è osservabile se lo è almeno uno dei suoi rami di fan-out; da cui si ha:

$$s_P = 1 - \prod_{\forall j} (1 - s_j) \quad (4.10)$$

dove j rappresenta un generico ramo di fan-out.

Le osservabilità dei segnali dello stesso circuito utilizzato nel caso della controllabilità sono riportate in Fig. 4.15.

La probabilità di rivelare i guasti può essere immediatamente ottenuta dal prodotto dell'opportuna controllabilità per il valore di osservabilità di tale linea: nel caso di un guasto di tipo *stuck-at-0* si ha quindi $d_k = c_k s_k$ e $d_k = (1 - c_k) s_k$ per un guasto di tipo *stuck-at-1*.

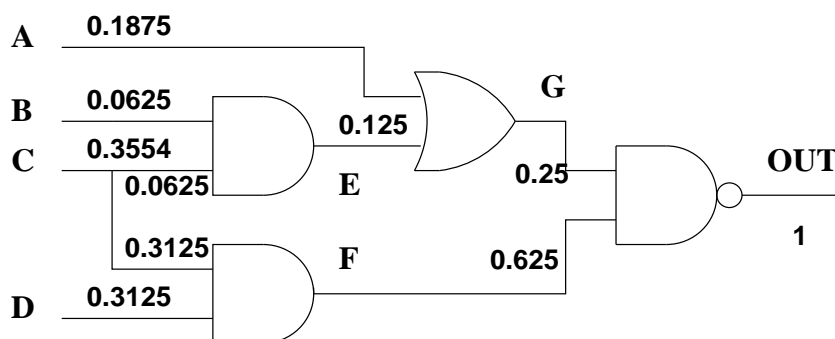


Figura 4.15: Valori di osservabilità stimati da COP per lo stesso circuito di Fig. 4.14.

Le probabilità di rivelare i guasti nel circuito utilizzato come esempio di applicazione di COP sono confrontati in Tab. 4.5, con quelli esatti. Come si può osservare, COP nel calcolo della *detectability* dei guasti commette errori che possono essere anche notevoli, per risolvere questo problema sono stati sviluppati diversi algoritmi che cercano di ottenere misure più accurate tenendo in conto, seppure in maniera approssimata, della correlazione fra i segnali del circuito.

guasto	COP	corretto
A <i>stuck-at-0</i>	0.09375	0.0625
B <i>stuck-at-1</i>	0.03125	0.0625
C <i>stuck-at-0</i>	0.1777	0.1875
C <i>stuck-at-1</i>	0.1777	0.1875
C.1 <i>stuck-at-1</i>	0.03125	0.0
C.2 <i>stuck-at-1</i>	0.1562	0.125
D <i>stuck-at-1</i>	0.1562	0.1875
E <i>stuck-at-0</i>	0.03125	0.0625
F <i>stuck-at-1</i>	0.46875	0.4375
OUT <i>stuck-at-0</i>	0.84375	0.8125
OUT <i>stuck-at-1</i>	0.15625	0.1875

Tabella 4.5: Probabilità di rivelare i guasti nel circuito di Fig. 4.14.

Mediante questo tipo di misure di collaudabilità è quindi possibile determinare la probabilità di rivelare i guasti e quindi la lunghezza delle sequenze casuali da applicare al circuito per ottenere dati valori di copertura, d'altra parte è anche possibile individuare i guasti che danno luogo a problemi di copertura (*random resistant*) ed eliminarli mediante tecniche di DFT.

4.4.1 Generazione random pesata

I ragionamenti fatti fino a questo momento riguardano distribuzioni uniformi dei segnali di ingresso, esistono casi in cui, invece, è preferibile avere ingressi con valori diversi della probabilità di avere un 1 o uno 0. Si consideri ad esempio il circuito di Fig. 4.16, dove è evidente che per massimizzare la probabilità di rivelare i guasti nel sottocircuito *C* i cui effetti si devono propagare attraverso il gate di uscita, è necessario che linea di ingresso *X* abbia una probabilità di valere 1 più grande di 0.5.

Chiaramente, questa linea non potrà avere una probabilità di valere 1 troppo grande, altrimenti la rivelabilità dello *stuck-at 1* sulla linea stessa diventerebbe troppo piccola. Si tratta quindi di determinare i "pesi" (le probabilità di valere 1) degli ingressi cercando di migliorare la rivelabilità di guasti *random resistant* senza danneggiare quella di guasti che sarebbero facilmente rivelabili con sequenze casuali, ma che potrebbero non esserlo per date distribuzioni degli ingressi. Gli algoritmi che ricavano i pesi ottimali degli ingressi sono tipicamente basati su misure di collaudabilità. In pratica, a partire dai

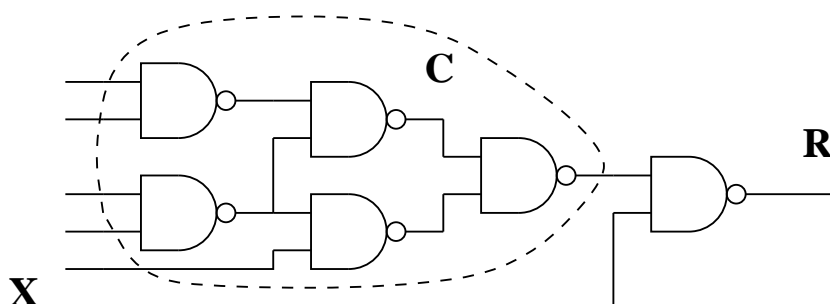


Figura 4.16: Circuito utilizzato per mostrare l'esigenza di disporre di probabilità di valere 1 diverse da 0.5 nella generazione di vettori di collaudo casuale.

valori di rivelabilità dei guasti ottenuti con pesi uniformi si variano i pesi dei diversi ingressi misurando ogni volta una cifra di merito (tipicamente la lunghezza di collaudo per una data copertura) fino ad ottenere una o più distribuzioni di probabilità tali da minimizzarne il valore. Per comprendere come siano convenienti più distribuzioni di pesi, si consideri un circuito costituito da un AND e un OR che condividono un numero alto di ingressi. Nel caso dell' AND, conviene avere un'altra probabilità di avere degli 1 e viceversa nel caso dell'OR (in questo modo, si migliora notevolmente l'osservabilità delle linee di ingresso). Se si adotta una singola distribuzione, allora risulterebbero dei pesi pari a 0.5 se invece si applicano due sequenze casuali la prima con pesi > 0.5 e viceversa la seconda, la rivelabilità dei guasti aumenta significativamente.

4.5 ATPG per circuiti sequenziali

La generazione di vettori di collaudo nel caso di circuiti sequenziali è un problema molto più complesso che nel caso di circuiti combinatori, perchè sia per attivare un guasto, sia per propagarne gli effetti alle uscite del circuito, sono necessarie sequenze di più vettori di collaudo. Questo problema è facilmente intuibile (si pensi al caso in cui si voglia rivelare un guasto *stuck-at 0* sull'uscita del k mo flip-flop di un registro a scorrimento a n stadi: per attivare il guasto sono necessari k vettori all'ingresso seriale e per propagarne gli effetti all'uscita seriale del registro servono $n - k$ vettori).

La maggiore complessità della generazione di vettori di collaudo per circuiti sequenziali si spiega col fatto che diversamente dal caso dei circuiti combinatori, la ricerca si svolge nell'ambito di tutte le possibili sequenze di vettori di ingresso e non in quello di tutti i possibili vettori. Questa difficoltà ha dato luogo alla ricerca di soluzioni di DFT tali da ricondurre il collaudo di un circuito sequenziale a quello della sua parte combinatoria come le tecniche di progetto basate sullo *scan-path*. Queste tecniche quali presentano degli inconvenienti e quindi rappresentano solo una soluzione parziale del

problema, quindi diverse linee di ricerca hanno affrontato i temi legati alla generazione di vettori di collaudo per circuiti sequenziali sviluppando algoritmi in grado di trattare circuiti anche abbastanza complessi.

Nel discutere queste tecniche si utilizzerà, il caso di circuiti sequenziali sincroni con guasti di tipo *stuck-at* che consente alcune semplificazioni, rispetto al caso di circuiti asincroni. Un'altra ipotesi è quella che i circuiti abbiano uno stato di reset e che quindi la generazione della sequenza di collaudo avvenga a partire da uno stato noto. Questa ipotesi è giustificata dal fatto che sia per problemi di collaudo che di corretto comportamento conviene progettare i circuiti in questo modo.

Esistono tecniche, che non verranno esaminate, per ricondurre un circuito sequenziale a uno stato noto. Per comprendere i problemi legati ai circuiti sequenziali è interessante vedere come sia possibile estendere gli algoritmi studiati nel caso di circuiti combinatori a quello dei circuiti sequenziali. Per fare questo, il circuito sequenziale viene trasformato in una sequenza iterativa (*iterative array*) di repliche della parte combinatoria del circuito ciascuna delle quali corrisponde allo stato del circuito in un certo intervallo di sincronismo (*frame*).

Ciascuna di queste repliche del circuito ha come ingressi gli ingressi primari del circuito e i segnali corrispondenti allo stato presente in quel *frame*. Come uscite si hanno le uscite primarie del circuito e i segnali di stato futuro che sono poi in ingresso alla replica del circuito corrispondente al *frame* seguente (Fig. 4.17).

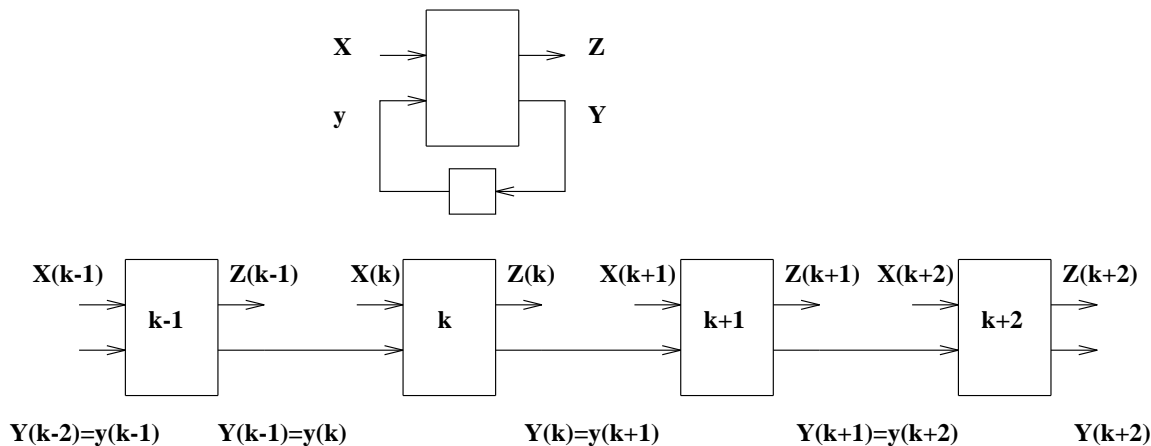


Figura 4.17: Rappresentazione di tipo *iterative array* dell'evoluzione di un circuito sequenziale.

Questo schema generale è quindi equivalente a un circuito combinatorio e pertanto possono essere applicati gli algoritmi di ATPG visti in precedenza. Per fare questo, bisogna chiarire alcuni punti che riguardano lo stato iniziale che si suppone noto e il numero di repliche del circuito da considerare per generare una sequenza di collaudo

per un dato guasto. Questa quantità, che condiziona l'efficienza dell'algoritmo, dipende dal guasto considerato. Si consideri infatti il circuito di Fig. 4.18, che è un semplice *adder* seriale che riceve in ingresso, a partire da quelli di minor peso i bit di due parole e fornisce serialmente l'uscita, questo circuito ha un solo flip-flop che si suppone essere inizialmente nello stato 0.

L'algoritmo di ATPG inizia con il generare un certo numero di repliche iniziali del circuito (in questo caso si parte da una) e tenta di generare una sequenza di collaudo per un circuito che ha come ingressi primari gli ingressi delle diverse repliche e come uscite primarie le loro uscite. Se l'algoritmo fallisce vengono aggiunte altre repliche.

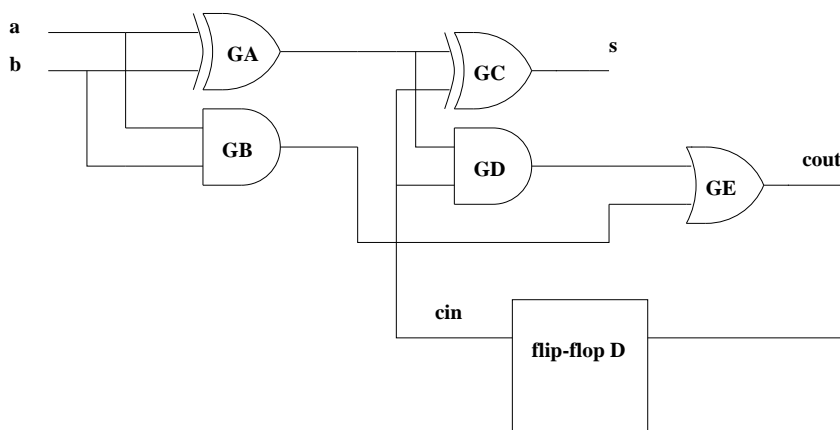


Figura 4.18: Addizionatore seriale utilizzato come esempio per gli algoritmi di ATPG per circuiti sequenziali.

Per capire questo processo, si considerino i guasti sulle uscite dei gate: a) **GA** *stuck-at-0*; b) **GB** *stuck-at-1*; c) **GD** *stuck-at-0*.

Nel caso a) il guasto può essere attivato (controllando a 1 l'uscita del gate **GA** tramite gli ingressi **a** e **b**) e reso osservabile all'uscita **s** con un singolo vettore di collaudo.

Nel caso b), invece, il guasto può essere attivato controllando a 0 l'uscita del gate **GB**, ma i suoi effetti possono essere propagati solamente alla linea di stato futuro **cout**, per renderli osservabili all'uscita è necessaria un'altra replica del circuito, in cui lo stato presente **cin** che porta gli effetti del guasto è facilmente osservabile in uscita.

Nel caso più complesso c), essendo *cin* a 0, il guasto non è attivato con il primo vettore di collaudo, per attivarlo **GD** dovrebbe valere 1 e non 0, per risolvere questo problema, bisogna aggiungere un'altra replica del circuito in cui si vede che bisogna avere **a**, **b** e **cin** a un valore logico alto, il valore di **cin** si giustifica assegnando **a** e **b** a 1 nel primo *frame*. Un altro *frame* serve per rendere osservabili gli effetti del guasto in uscita.

Il modello di tipo *iterative array* per la generazione di sequenza di collaudo per i guasti considerati è rappresentato in Fig. 4.19.

Da questi esempi risulta chiaro che all'aumentare della complessità dei circuiti il numero di repliche richieste per generare una sequenza di collaudo può dare luogo a un circuito troppo grande per essere trattato efficientemente dai programmi di ATPG, per questo motivo la tecnica illustrata non trova un largo utilizzo nella pratica.

Per risolvere questo tipo di problemi, sono state sviluppate diverse tecniche alternative di generazione di vettori di collaudo. Fra questi algoritmi si hanno metodi che non sfruttano la topologia del circuito sequenziale, ma utilizzano un modello di tipo funzionale che consiste nella rappresentazione del circuito come macchina a stati.

Questi algoritmi, detti di tipo funzionale, rappresentano il circuito sequenziale con il grafo di transizione dello stato e per generare la sequenza di collaudo costruiscono la macchina prodotto di quella corretta e di quella in cui è stato iniettato il guasto secondo lo schema di Fig. 4.20 in cui si vede che gli ingressi sono gli stessi delle due macchine a stati di partenza, mentre lo stato è dato dal prodotto cartesiano dei due insiemi di stati di partenza e l'uscita vale 1 solo se le uscite delle due macchine sono diverse.

È evidente, che se l'uscita della macchina prodotto è sempre nulla per qualsiasi sequenza di ingresso le due macchine sono equivalenti e quindi il guasto non è rivelabile, se invece esistono stati per cui la risposta della macchina prodotto è non nulla per dati valori degli ingressi allora il guasto è rivelabile.

Per trovare la sequenza di collaudo si costruisce, a partire da quelli delle macchine di partenza, il grafo di transizione dello stato della macchina prodotto e si cercano, con tecniche di attraversamento del grafo (il problema presenta notevoli costi di calcolo, per cui viene spesso risolto mediante approcci di tipo simbolico - *symbolic traversal* - che non richiedono di costruire esplicitamente il grafo di transizione dello stato), quegli stati per cui l'uscita della macchina prodotto è non nulla (e quindi la risposta del circuito in assenza di guasti e di quello guasto differiscono). Una volta trovata questo stato e i valori di ingresso tali da "dimostrare" che le due macchine non sono equivalenti, si tratta di trovare una sequenza di ingressi che a partire dallo stato di reset consenta di raggiungere tale stato.

Per illustrare questo tipo di tecniche, si consideri il guasto (c) dell'esempio precedente (coorspondente all'uscita del *gate GD* del circuito di Fig. 4.18 *stuck-at-0*). Le tabelle delle transizioni del circuito nel caso privo di guasti e in presenza del guasto, sono riportate in Tab. 4.6 e 4.7.

La tabella delle transizioni della macchina prodotto è riportata in Tab. 4.8, dove si può vedere che l'unico stato per cui la macchina prodotto ha uscita diversa da 0 è lo stato 10. Individuato questo stato, una sequenza di collaudo deve portare dallo stato

cin	a,b			
	00	01	10	11
0	0, 0	0, 1	0, 1	1, 0
1	0, 1	1, 0	1, 0	1, 1

Tabella 4.6: Stato futuro (cout) e uscita (s) del circuito di Fig. 4.18 in assenza di guasti in funzione degli ingressi (a, b) e dello stato presente.

cin*	a,b			
	00	01	10	11
0	0, 0	0, 1	0, 1	1, 0
1	0, 1	0, 0	0, 0	1, 1

Tabella 4.7: Stato futuro (cout) e uscita (s) del circuito di Fig. 4.18 in presenza del guasto (c) in funzione degli ingressi (a, b) e dello stato presente.

di reset (00) allo stato 10 ed applicare un qualsiasi vettore di ingresso (in questo caso, infatti, si ha errore per tutte le configurazioni di ingresso).

Questa sequenza può essere facilmente individuata nel diagramma di transizione degli stati della macchina prodotto (Fig. 4.21), dove, a partire dallo stato di reset (00), la sequenza di ingressi 11,01 serve per raggiungere lo stato 10 e l'ingresso successivo consente di rivelare i guasti.

cin, cin*	a,b			
	00	01	10	11
00	00, 00, 0	00, 11, 0	00, 11, 0	11, 00, 0
01	00, 01, 1	00, 10, 1	00, 10, 1	11, 01, 1
10	00, 10, 1	10, 01, 1	10, 01, 1	11, 10, 1
11	00, 11, 0	10, 00, 0	10, 00, 0	11, 11, 0

Tabella 4.8: Stato futuro (cout, cout*), uscite dei due blocchi combinatori e uscita della macchina prodotto in funzione degli ingressi e dello stato presente. Lo stato 01 non è raggiungibile e quindi l'unico stato per cui l'uscita della macchina prodotto vale 1 per qualsiasi configurazione degli ingressi (questo è un caso particolare).

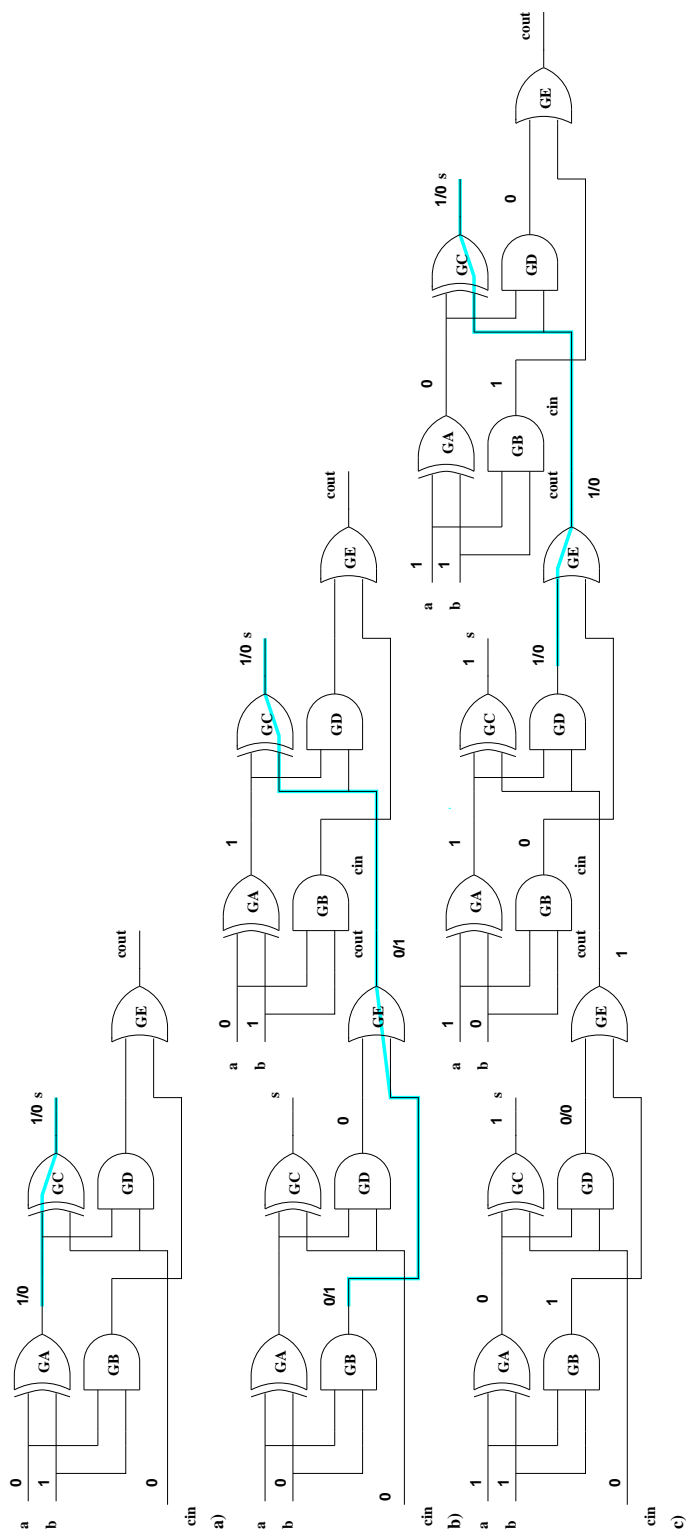


Figura 4.19: Esempio di generazione di vettori di collaudo per un semplice circuito sequenziale.

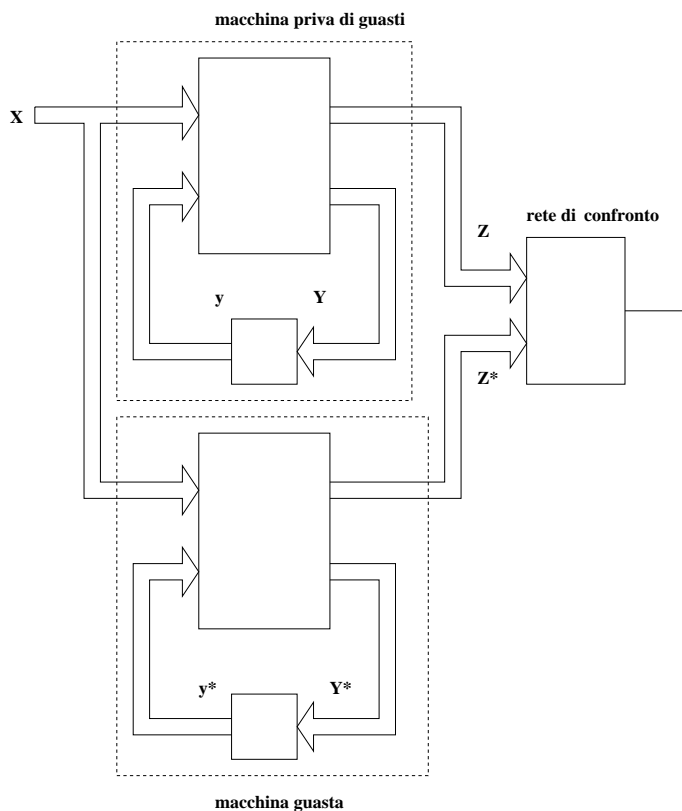


Figura 4.20: Schema di partenza utilizzato dagli algoritmi di tipo funzionale per la generazione di vettori di collaudo in circuiti sequenziali sincroni.

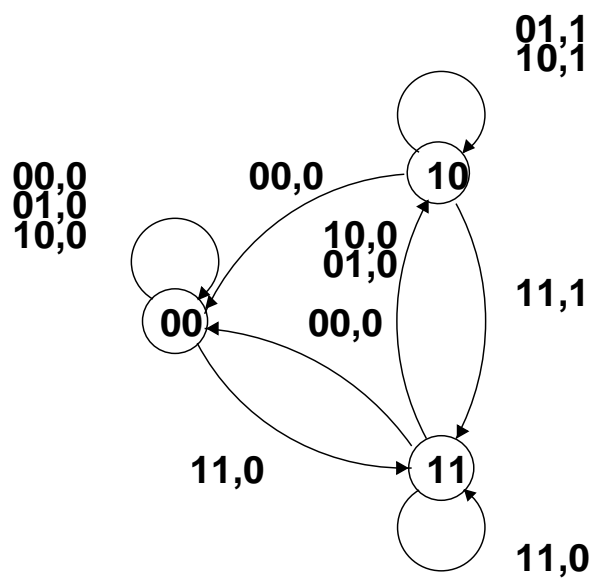


Figura 4.21: Diagramma degli stati della macchina prodotto.