SVILUPPO DI FORMALISMI PER ALBERI DEI GUASTI CON NODI DIPENDENTI E RIPARABILI

Candidato: Daniele Codetta Raiteri
Relatore: Prof. Andrea Bobbio
Contro-Relatore: Prof. ssa Giuliana Franceschinis

Alessandria, 16 Luglio 2002

Indice

Introduzione

L'argomento della tesi è l'albero dei guasti [1], un modello stocastico per la rappresentazione della struttura di un sistema e per l'analisi della sua affidabilità.

Dopo un esame iniziale del concetto di affidabilità di un sistema [2] e del significato logico dell'albero dei guasti, si è passati allo studio delle fasi dell'analisi del modello [3], concentrandosi sul calcolo della probabilità di guasto del sistema ad un certo tempo (**Cap. 1**).

L'attenzione si è quindi rivolta su due strumenti informatici, *Sharpe* [4] e *Astra* [5], per l'analisi automatica di alberi dei guasti, osservandone i formalismi di rappresentazione del modello, le tecniche di analisi, gli indici calcolabili e valutando la qualità dei risultati (**Cap. 2**).

Per fare questo si è reso necessario effettuare delle analisi con entrambi gli strumenti sugli stessi alberi dei guasti; entrambi gli strumenti richiedono la creazione di un file che contiene la struttura dell'albero espressa nel relativo formalismo; per semplificare l'operazione di confronto dei risultati, si è sviluppato un traduttore (*sharpe2astra*) che, dato un albero dei guasti rappresentato nel formalismo di *Sharpe*, genera un albero logicamente equivalente nel formalismo di *Astra* (**Cap. 3**).

L'esame di sistemi grandi dimensioni, quali quelli reali, ha richiesto un modo più semplice e rapido per realizzare la stesura dei relativi alberi dei guasti, piuttosto che scrivere "a mano" i file per *Sharpe* o *Astra*; perciò si è passati ad una variante del modello che consente una rappresentazione più compatta del sistema, l'albero dei guasti parametrico [6], e al disegno del modello attraverso uno strumento grafico, *DrawNet* [7] [8].

A questo punto si è sviluppata un'applicazione (*drawnet2sharpe*) che, dato un albero dei guasti parametrico disegnato con *DrawNet*, ne genera il relativo file per Sharpe dopo aver effettuato un'operazione di *unfolding*, cioè di trasformazione dell'albero dalla forma parametrica a quella esplicita (**Cap. 4**).

Grazie a questa applicazione, è ora possibile disegnare un albero dei guasti in forma parametrica attraverso uno strumento grafico ed analizzarlo con

Sharpe e quindi con *Astra* (tramite *sharpe2astra*).

La parte successiva della tesi riguarda lo studio dell'analisi per moduli [8] [9] di un albero dei guasti e l'introduzione di nuovi tipi di nodi, quali le scatole di riparazione [8] e le porte logiche dinamiche [10] [11].

Per quanto riguarda l'analisi per moduli, cioè l'analisi dell'albero fatta non nella sua interezza, ma esaminando una per volta le sue parti indipendenti (moduli), si è esaminato ed applicato un algoritmo per l'individuazione dei moduli [12] [9] (**Cap. 5**) e lo si è adattato per gli alberi con scatole di riparazione e in seguito con porte logiche dinamiche.

Per quanto riguarda gli alberi dei guasti con scatole di riparazione, dopo lo studio del significato logico della riparazione, delle sue possibili semantiche e delle dipendenze che questa determina su alcuni nodi dell'albero, *DrawNet* è stato modificato per inserire le scatole di riparazione all'interno degli alberi dei guasti (**Cap. 6**).

L'esame delle parti dell'albero in cui sono presenti scatole di riparazione non può avvenire con le tecniche utilizzate per gli alberi più primitivi, ma richiede un'analisi nello spazio degli stati facendo ricorso alle *Reti di Petri* [13] [14].

Quindi si è sviluppata un'applicazione che, dato un albero con nodi riparabili disegnato con *DrawNet*, ne effettua l'analisi individuandone i moduli e analizzandoli opportunamente (dal punto di vista combinatorio (*Sharpe*) o traducendoli in *Reti di Petri* [8] [15]) (**Cap. 7**).

La tesi termina con l'introduzione di una nuova porta logica ($k : n_{cons}$) [16] e con lo studio delle porte logiche dinamiche, riportando per ognuna il significato logico, la traduzione nel *Modello di Markov* [10], in *Rete di Petri* e come è stato aggiornato *DrawNet* in loro funzione (**Cap. 8**).

Capitolo 1

Introduzione al Fault Tree

1.1 Dallo studio dell'affidabilità del sistema al modello

Valutare l'affidabilità e la sicurezza di un sistema, specialmente se di grandi dimensioni, ha richiesto nel corso degli anni lo sviluppo, sia a livello industriale che accademico, di metodi e tecniche per generare modelli e quindi analizzare il comportamento nel tempo di un sistema.

I fattori che solitamente si osservano per valutare la qualità di un sistema riguardano aspetti quali i costi, la velocità, i consumi, le dimensioni, ecc. a livello di progettazione, mentre a livello di funzionamento l'attenzione è maggiormente concentrata sul concetto di *affidabilità* (reliability) [2].

Per *affidablità* si intende la capacità del sistema di far fronte ai guasti che ne potrebbero compromettere il fuzionamento e viene misurata attraverso la probabilità di assenza o presenza di malfunzionamenti a livello operativo in un determinato periodo di tempo.

Affidabilità è quindi sinonimo di "comportamento desiderato del sistema" e aumentando le esigenze di affidabilità è stato necessario ideare un modello che sia in grado allo stesso tempo di fornire una rappresentazione compatta e di essere facilmente specificato e compreso dal progettista.

L'*albero dei guasti* (Fault Tree) [1] [3] ha queste caratteristiche ed è stato largamente usato nell'*analisi di affidabilità* (reliability analysis) fin dai primi anni '60 ed è stato oggetto di numerose ricerche fino ad oggi.

Lo studio di questo modello prende il nome di *Fault tree analysis* (**FTA**) e ha come concetto fondamentale la traduzione della struttura fisica del sistema in un diagramma logico strutturato, l'albero dei guasti, in cui partendo dalle foglie che rappresentano i dispositivi fisici, si stabiliscono delle relazioni

logiche che indicano come i guasti si trasmettono dai componenti di base ai sottosistemi fino al guasto del sistema in generale.

La figura 1.1 è un esempio di albero dei guasti disegnato con lo strumento grafico *DrawNet* citedrawnet [8] .

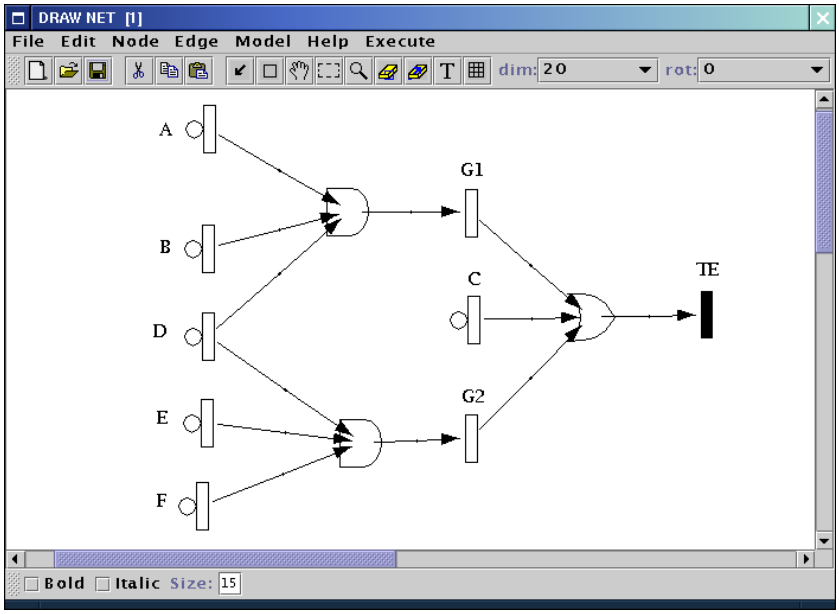


Figura 1.1: Esempio di Albero dei guasti

1.2 Il modello Albero dei guasti

Matematicamente parlando, il modello descrive le interazioni tra componenti di base del sistema come funzioni booleane. I nodi di un albero dei guasti sono di due tipi:

- **Eventi di base** (basic events): rappresentano componenti fisici del sistema; corrispondono alle "foglie" dell'albero (Fig. 1.2 (a));
- **Eventi Interni**: corrispondono all'output di una relazione tra eventi di base o interni e rappresentano dei sottosistemi (Fig. 1.2 (b)).

Un evento è una variabile booleana per cui il valore 1 indica il fatto che quel componente o quel sottosistema è guasto. Gli eventi possono essere collegati tra loro tramite delle relazioni che assumono il nome di *porte logiche*

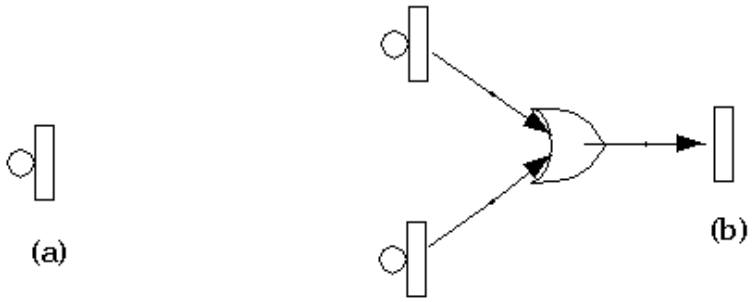


Figura 1.2: Evento di base (a) e interno (b)

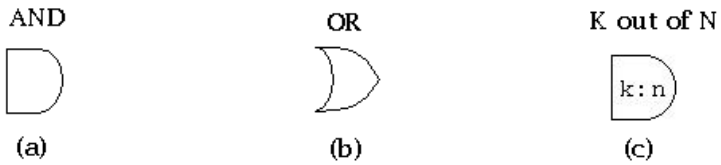


Figura 1.3: Porte logiche

(gate) . Esistono vari tipi di porte logiche; le più comuni nella Fault tree Analysis sono quelle indicate in Fig. 1.3.

Ogni porta logica ha un insieme di eventi di input che possono essere di base o interni ed un unico evento di output di tipo interno (Fig. 1.2 b). Il valore che assume l'evento di output dipende dal valore degli eventi di input e dal tipo di *porta logica* (relazione):

- **AND:** (Fig. 1.3 (a)) dati gli eventi di input $X_1, X_2, ..., X_n$ l'evento di output della porta logica ha valore

$$Y = X_1 \wedge X_2 \wedge ... \wedge X_n$$

(l'output è un guasto se tutti gli input sono guasti)

- **OR:** (Fig 1.3 (b)) dati gli eventi di input $X_1, X_2, ..., X_n$ l'evento di output della porta logica ha valore

$$Y = X_1 \vee X_2 \vee ... \vee X_n$$

(l'output è guasto se almeno uno degli input è guasto)

- **K out of N (k:n)** : (Fig. 1.3 (c)) dati gli eventi di input l'evento di output della porta logica ha valore 1 se almeno k dei suoi n input sono guasti, 0 altrimenti; la porta **k:n** può essere espressa come una combinazione di porte and e or come mostra la Fig. 1.4.

  Esempio di k:n relativo alla Fig. 1.4:

  $k = 2, n = 3$

  input: X_1, X_2, X_3

  output: $Y = (X_1 \wedge X_2) \vee (X_1 \wedge X_3) \vee (X_2 \wedge X_3)$

Esiste inoltre un evento finale chiamato *Top Event* (in Fig. 1.1 indicato con una barra nera) che indica lo stato dell'intero sistema (1 se guasto, 0 altrimenti).

Nell'albero di Fig. 1.1 appaiono le porte logiche **or** e **and**; il sistema così rappresentato è composto al primo livello dai sottosistemi $G1$ e $G2$ e dal dispositivo C; il sistema si guasta se almeno una di queste parti si guasta (**or**).

$G1$ è formato dai componenti A, B, D, mentre $G2$ da D (condiviso con $G1$), E e F; $G1$ e $G2$ si guastano se si guastano tutti i relativi componenti (**and**).

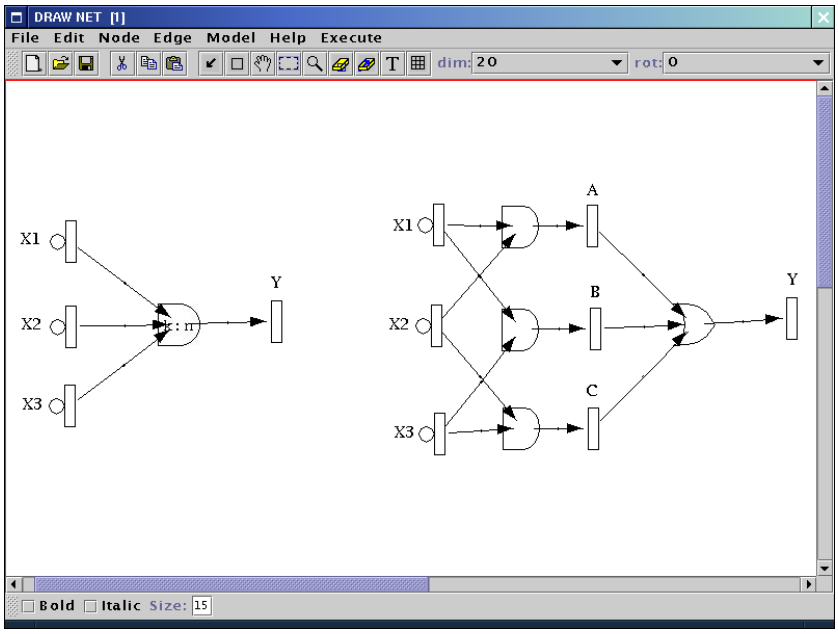


Figura 1.4: Combinazione di porte logiche corrispondente alla porta k:n

Esempi: assegnando dei valori booleani agli eventi di base dell'albero in Fig. 1.1, che rappresentano i componenti fisici del sistema, il guasto si può propagare fino al *Top Event* (indicato con TE) oppure soltanto fino ad un evento interno.

- per $A = 1, B = 1, C = 0, D = 1, E = 0, F = 1$

  $G1 = 1, C = 0, G2 = 0$

  quindi $TE = 1$ (guasto)

- per $A = 1, B = 0, C = 0, D = 1, E = 0, F = 1$ si ha che

  $G1 = 0, C = 0, G2 = 0$

  quindi $TE = 0$ (non guasto)

- per $C = 1$ e qualsiasi valore per gli altri eventi di base

  $TE = 1$

1.3 Analisi dell'albero dei guasti

1.3.1 La distribuzione esponenziale

Ad ogni evento di base viene assegnata una determinata distribuzione di probabilità, ad esempio

$$P(X = 1) = 0.00025, P(X = 0) = 1 - P(X = 1).$$

Si può legare la probabilità di guasto al tempo usando una distribuzione *esponenziale* per il tempo del primo guasto e specificando per ogni evento di base il fattore λ detto *tasso di guasto*.

Se per esempio $\lambda = 0.001$ guasti/ore, il tempo medio per il verificarsi di un guasto sarà $1/\lambda$, in questo caso 1000 ore.

Questo tipo di distribuzione è quella più largamente usata nella FTA e viene indicata come $exp(\lambda)$.

Funzione di densità:

$$f(t) = \lambda e^{-\lambda t} =$$

= Probabilità che il componente si guasti nell'intervallo di tempo compresto tra t e $t + dt$.

Funzione di ripartizione:

$$F(t) = 1 - e^{-\lambda t} =$$

= Probabilità che il componente sia già guasto al tempo $T = t$.

1.3.2 Analisi qualitativa

Prima di essere in grado di calcolare fattori quali tempi medi o probabililta' di guasto (*analisi quantitativa*) è necessario eseguire l'*analisi qualitativa*.

Questa consiste nel determinare i *cutset minimi* (**MCS**) dell'albero; per cutset minimo si intende un insieme minimale di componenti il cui guasto contemporaneo determina il guasto dell'intero sistema.

Si individuano i cutset come i termini dell'espressione booleana dell'albero in *forma normale disgiuntiva* (**DNF**); per arrivare a questa bisogna derivare dall'albero la sua espressione booleana visitando i nodi in profondità.

Facendo riferimento all'albero di Fig. 1.1 è possibile ricavare la formula booleana dell'intero albero e cioè:

$G1 = (A \wedge B \wedge D)$

$G2 = (D \wedge E \wedge F)$

$TE = G1 \vee C \vee G2 = (A \wedge B \wedge D) \vee C \vee (D \wedge E \wedge F)$

A questo punto attraverso una serie di trasformazioni e semplificazioni secondo l'Algebra booleana si giunge alla *forma normale disgiuntiva* (**DNF**), cioè una disgiunzione di congiunzioni di variabili; l'espressione booleana dell'albero di Fig. 1.1 è già in DNF.

Se un cutset minimo corrisponde a $x_1 \wedge x_2 \wedge \ldots \wedge x_n$ si dirà che ha ordine n (perché comprende n elementi) e dato che si tratta di una congiunzione di variabili che rappresentano eventi che si assume siano indipendenti tra loro, la sua probabilità di verificarsi sarà il prodotto delle probabilità di guasto dei singoli eventi:

$$P(MCS) = P(x_1 \wedge x_2 \wedge \ldots \wedge x_n) = \prod_{i=1}^{n} P(x_i)$$

Ritornando all'esempio di Fig. 1.1 i MCS sono i seguenti:

$MCS_1 = A, B, D$
$MCS_2 = C$
$MCS_3 = D, E, F$

1.3.3 Analisi quantitativa

Basta che si verifichi il guasto di tutti gli elementi di un MCS qualunque per determinare il guasto generale del sistema, quindi la sua probabilità di guasto corrisponde alla probabilità dell'unione dei MCS; in questo caso

$$P(MCS_1 \vee MCS_2 \vee MCS_3) =$$

$$P(MCS_1)+P(MCS_2)+P(MCS_3)-P(MCS_1 \wedge MCS_2)-P(MCS_1 \wedge MCS_3)-$$

$$P(MCS_2 \wedge MCS_3) + P(MCS_1 \wedge MCS_2 \wedge MCS_3)$$

dove l'intersezione tra due o più cutset minimi corrisponde al prodotto delle loro probabilità se sono indipendenti (senza eventi in comune), altrimenti al prodotto delle probabilità dei loro eventi considerati una volta sola.

Comp.	λ
A	0.0000001
B	0.0000001
C	0.0000002
D	0.0000002
E	0.0000003
F	0.0000003

Tabella 1.1: Parametri dei componenti

La probabilità di guasto del sistema se i MCS sono tutti indipendenti si può anche calcolare come:

$$1 - (1 - P(MCS_1))(1 - P(MCS_2)) \dots (1 - P(MCS_m))$$

dove $MCS_1, MCS_2, \dots, ..., MCS_m$ sono i cutset minimi, $(1 - P(X_i))$ è la probabilità di funzionamento del MCS i-esimo e il risultato finale corrisponde a $1 -$ Prob.(funzionamento del sistema).

Se gli eventi di base hanno una probabilità costante, basterà sostituirla nelle formule precedenti; se invece gli eventi hanno una distribuzione di tipo esponenziale di parametro λ, la probabilità di guasto del componente dipenderà dal tempo.

Quest'ultima è pari alla *funzione di ripartizione* $F(t)$ secondo il valore del fattore λ del componente e del valore di tempo assegnato.

1.3.4 Esempio

Facendo riferimento all'albero di Fig. 1.1, stabiliamo che i componenti A, B, C, D, E, F abbiano tutti distribuzione esponenziale con i parametri mostrati in Tab. 1.1.

Calcoliamo per ogni componente la probabilità che sia guasto al tempo 500000.

$$F_A(5000) = 1 - e^{-0.0000001 \cdot 500000} = 1 - e^{-0.05} = 0.0487706 = F_B(500000)$$
$$F_C(5000) = 1 - e^{-0.0000002 \cdot 500000} = 1 - e^{-0.10} = 0.0951626 = F_D(500000)$$
$$F_E(5000) = 1 - e^{-0.0000003 \cdot 500000} = 1 - e^{-0.15} = 0.1392920 = F_F(500000)$$

I cutset minimi sono già stati individuati nel paragrafo 1.3.2; calcoliamo per ognuno la probabilità di guasto al tempo 500000.

$$P(MCS_1) = F_A(500000) \cdot F_B(500000) \cdot F_D(500000) = 0.0002264$$

$P(MCS_2) = F_C(500000) = 0.0951626$
$P(MCS_3) = F_D(500000) \cdot F_E(500000) \cdot F_F(005000) = 0.0018464$

Determiniamo la probabilità di guasto del sistema come la probabilità dell'unione dei MCS:

$P(S) = P(MCS_1) + P(MCS_2) + P(MCS_3) - (P(MCS_1) \cdot P(MCS_2)) - (F_A \cdot F_B \cdot F_D \cdot F_E \cdot F_F) - (P(MCS_2) \cdot P(MCS_3)) + (F_A \cdot F_B \cdot F_C \cdot F_D \cdot F_E \cdot F_F) = = 0.0002264 + 0.0951626 + 0.0018464 - 0.0000215 - 0.0000044 - 0.0001757 + 0.0000004 = 0.0970342$

1.4 Albero dei guasti parametrico

Osservando l'albero dei guasti di sistemi di grandi dimensioni e che prevedono componenti simili tra loro, si possono notare parti del diagramma che vengono replicate più volte. Se si tratta di interi sottoalberi che vengono ridisegnati o se il numero di componenti di un certo tipo è molto elevato la stesura dell'albero si complica ulteriormente, rendendo inoltre la successiva analisi più complessa.

Si è reso quindi necessario utilizzare una notazione compatta per indicare parti del sistema identiche, ma distinte tra loro, nota come *Albero dei guasti parametrico* (**PFT**) [6] [17].

In questo modello, un insieme di dispositivi uguali tra loro viene rappresentato da un unico oggetto grafico a cui corrisponde un *parametro* che determina il numero delle repliche. Tale oggetto può essere un *evento di base replicato* o un *evento interno replicato*.

Per dispositivi uguali si intendono dei dispositivi che hanno la stessa funzione e la stessa probabilità di guastarsi; nel caso della distribuzione esponenziale hanno lo stesso λ.

La Fig. 1.5 è un esempio di albero dei guasti la cui rappresentazione in PFT ne semplifica decisamente la struttura a livello grafico come si vede in Fig. 1.6 in cui un insieme di eventi di base uguali tra loro viene indicato da un evento di base replicato.

La parametrizzazione può riguardare anche eventi interni e quindi interi sottoalberi come nelle Fig. 1.7 e 1.8 in cui la parametrizzazione riguarda il sottoalbero $G(i)$ e al suo interno a sua volta l'evento di base $BE(i, j)$.

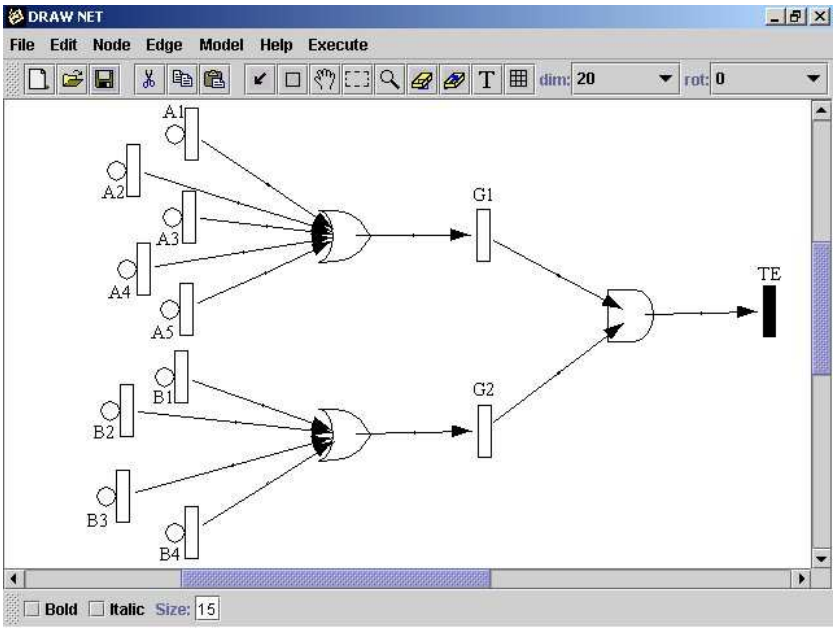


Figura 1.5: Albero con componenti identici, ma distinti

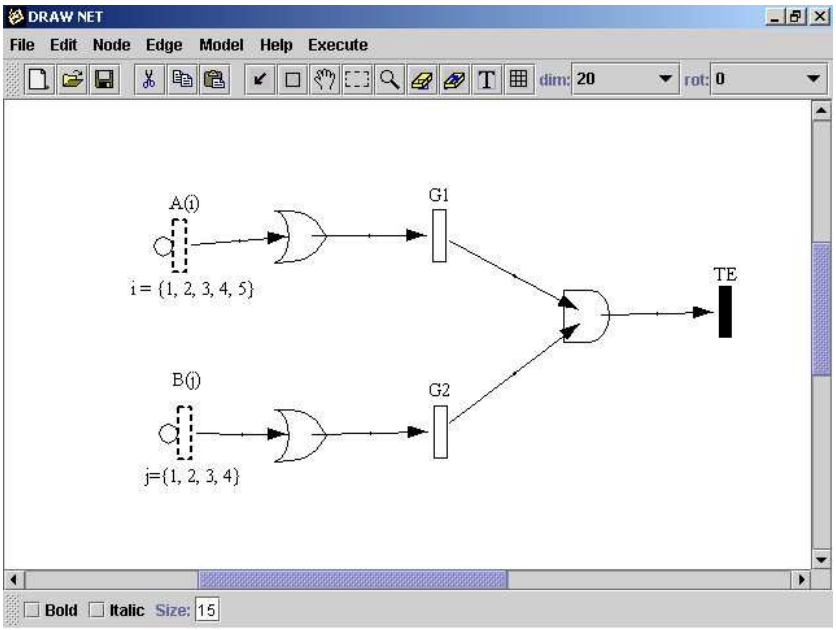


Figura 1.6: L'albero di Fig. 1.5 in forma di PFT

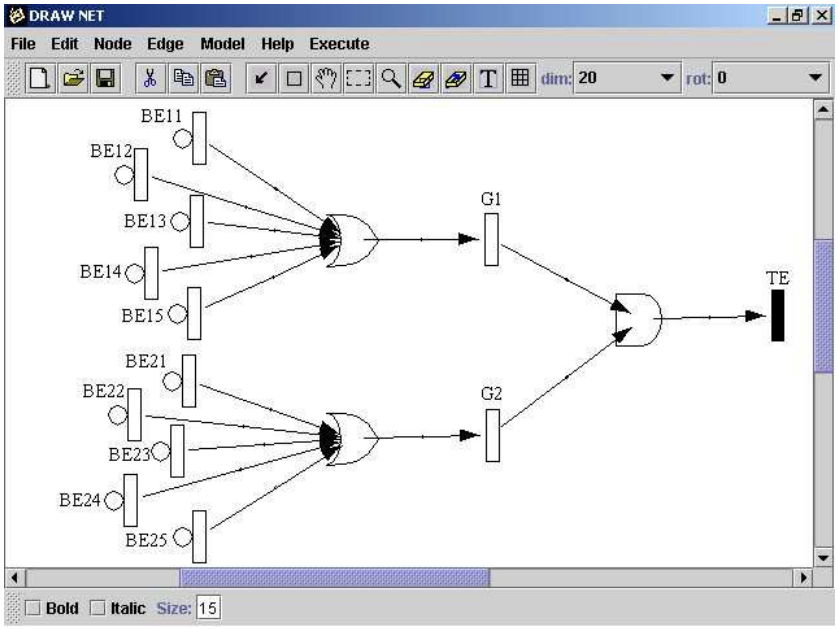


Figura 1.7: Albero con sottoalberi identici, ma distiniti (G1 e G2)

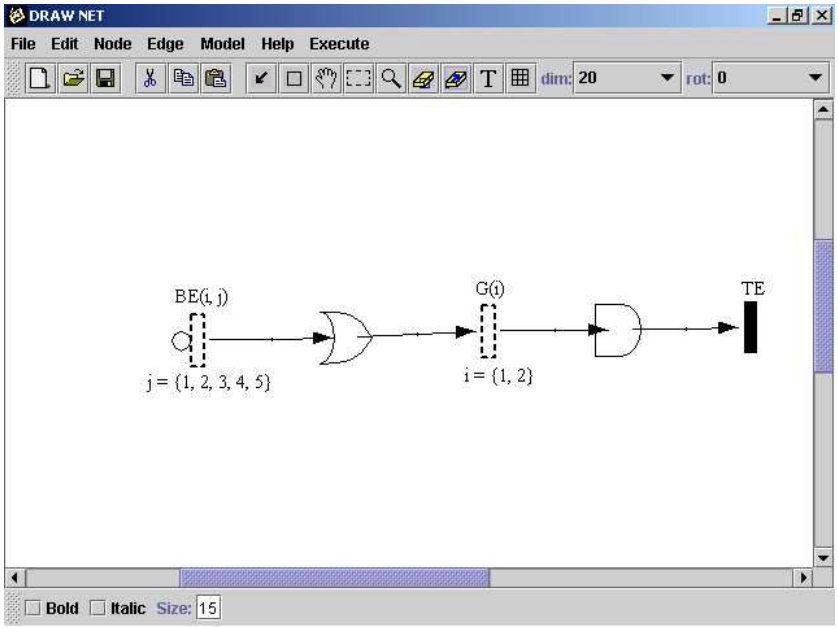


Figura 1.8: L'albero di Fig. 1.7 in forma PFT

Capitolo 2

Strumenti per l'analisi dell'albero dei guasti

2.1 Introduzione

Quando si modellano dei sistemi reali, la corrispondente rappresentazione sotto forma di albero dei guasti è di solito notevolmente estesa; fare l'analisi di questo genere di alberi richiede un numero elevato di operazioni dato che le combinazioni di eventi sono numerose; il calcolo dei cutset minimi e degli indici di affidabilità si rivela quindi particolarmente complicato.

Per questo esistono diversi programmi per la stesura e l'analisi automatica degli alberi dei guasti; in questo capitolo sono trattati due strumenti di questo tipo: *Sharpe* e *Astra*; per ognuno saranno illustrati il formalismo di rappresentazione, gli indici che sono in grado di calcolare e alcuni esempi di applicazione.

2.2 Sharpe

Sharpe [4] è uno strumento sofware per analizzare vari tipi di modelli stocastici tra cui le *catene di Markov*, *le reti di code*, *le reti di Petri* e appunto gli alberi dei guasti.

La versione di *Sharpe* utilizzata durante lo sviluppo della tesi è eseguibile in ambiente *Sun Solaris* e funziona dal prompt dei comandi con una chiamata del tipo:

```
sharpe.sun4 <file>
```

Il file che Sharpe riceve come parametro deve contenere l'indicazione del tipo di modello (albero dei guasti), la sua struttura (eventi di base e porte

logiche) e i risultati che l'analisi deve restituire (ad esempio il tempo medio di guasto del sistema rappresentato dall'albero). Ne segue un esempio:

```
* albero di esempio

ftree albero

repeat a exp(v1)
repeat b exp(v1)
repeat c exp(v1)

or G1 a c
or G2 c b
and TOP G1 G2

end

bind
v1 0.00000077
end

expr mean(albero)
expr variance(albero)

end
```

Questo file descrive un albero composto da tre porte logiche e che ha tre eventi di base; la fig. 2.1 è la sua rappresentazione grafica. Di questo albero vengono richiesti la media e la varianza del tempo di guasto. L'output dato da Sharpe per questo file è il seguente:

```
 mean(albero):   8.6580e+05

-----------------------------------------

 variance(albero):   5.6221e+11

-----------------------------------------
```

La Fig. 2.1 è la versione grafica dell'albero realizzata con *DrawNet*.

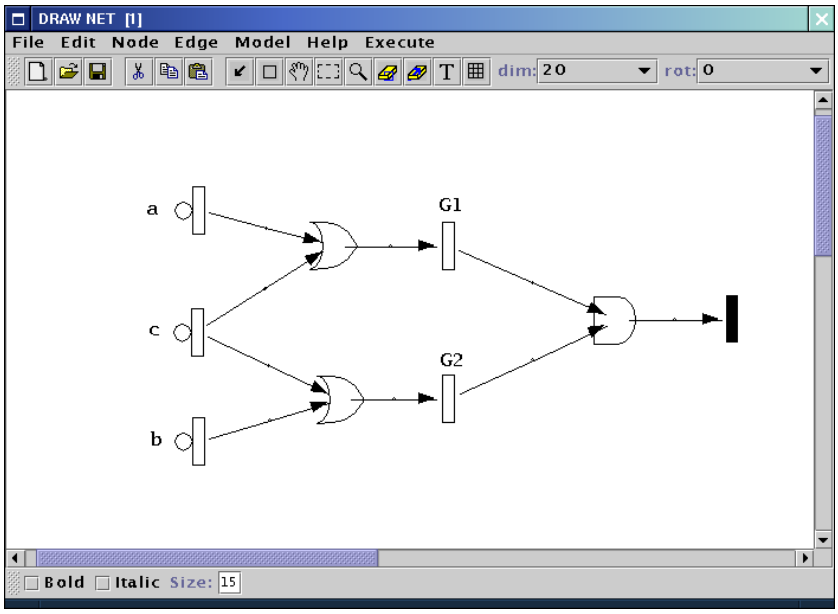


Figura 2.1: Albero di esempio

2.2.1 Il formalismo di Sharpe

Le principali regole per descrivere alberi dei guasti con Sharpe sono le seguenti:

- **Commenti:** si possono inserire commenti in qualsiasi punto della descrizione dell'albero preceduti da *; di solito si inserisce un commento sulla prima linea della descrizione per dare un titolo all'albero.
- **Eventi di base:** la dichiarazione di un evento di base è la seguente:

  ```
  <tipo> <nome> <distribuzione>
  ```

  Il tipo può essere *repeat* o *basic*: se un evento di base viene dichiarato *repeat*, ogni volta che il suo nome apparirà nella descrizione dell'albero, sarà interpretato sempre come lo stesso componente fisico; se invece, l'evento di base viene dichiarato come *basic*, ogni volta che il suo nome apparirà più avanti nel file sarà considerato come una copia distinta dell'evento dichiarato in precedenza.

  Nel file per Sharpe di esempio ci sono due riferimenti all'evento **c**; essendo questo dichiarato *repeat* ogni suo riferimento riguarda un unico componente fisico.

Sharpe consente di associare ad un evento di base diversi tipi di distribuzione di probabilità; quelle che sono state principalmete utilizzate durante il lavoro di tesi sono $exp(\lambda)$ (distribuzione esponenziale di parametro λ) e $prob(p)$ (probabilità costante di guasto p).

- **Porte logiche:** la dichiarazione per le porte logiche è la seguente:

  ```
  <tipo> <nome output> <nome input1> <nome input2> ...
  ```

  I tipi di porta logica supportati da Sharpe sono: **and**, **or** e **kofn**; quest'ultima ha un formalismo particolare che prevede anche l'indicazione dei parametri k ed n; se poi la porta logica ha come input n eventi uguali tra loro se ne può indicare uno solo; ad esempio:

  ```
  kofn 2,4, TOP in
  ```

  La porta logica in questo caso ha $n = 4$ input uguali e distinti tra loro e corrispondenti all'evento di nome **in** e ha come output il *Top Event*.

  Sharpe impone che in dichiarazioni di questo tipo tutti gli eventi di base siano di tipo *basic* dato che si tratta di repliche; in questo caso l'evento **in** deve essere di tipo *basic*.

- **Costanti:** Sharpe permette anche di dichiarare delle costanti che possono essere usate per indicare ad esempio il parametro di una distribuzione esponenziale comune a più eventi di base; la loro definizione avviene dopo la parola chiave **bind** nella forma

  ```
  <nome> <valore>
  ```

Non è necessario che vengano dichiarati prima tutti gli eventi di base e poi le porte logiche; si richiede solo che gli eventi di input di una porta logica siano definiti nelle righe precedenti alla riga della dichiarazione della porta logica stessa. L'ordine delle dichiarazioni deve quindi rispettare un ordine di tipo *bottom-up*; la dichiarazione del *Top Event* sarà di conseguenza l'ultima. L'ordine generale delle dichiarazioni è quindi il seguente:

1. **ftree** nome albero
2. dichiarazione di eventi di base e porte logiche in ordine tipo bottom-up

3. **end**
4. **bind**
5. dichiarazione di costanti
6. **end**
7. richiesta dei risultati di output
8. **end**

Sharpe è in grado di fornire vari risultati di output tra cui media e varianza del tempo di guasto come nell'esempio sopra. Se aggiungiamo la linea

```
eval(albero) 0 500000 50000
```

Sharpe ritorna anche i valori della probabilità di guasto al tempo t ($F(t)$) per **t** che va da 0 a 500000 con un passo di 50000:

```
 mean(albero):   8.6580e+05

-----------------------------------------

 variance(albero):   5.6221e+11

-----------------------------------------

      system albero
      t         F(t)

 0.0000 e+00  0.0000 e+00
 2.5000 e+04  1.9422 e-02
 5.0000 e+04  3.9141 e-02
 7.5000 e+04  5.9086 e-02
 1.0000 e+05  7.9195 e-02
 1.2500 e+05  9.9411 e-02
 1.5000 e+05  1.1968 e-01
 1.7500 e+05  1.3995 e-01
 2.0000 e+05  1.6019 e-01
 2.2500 e+05  1.8035 e-01
 2.5000 e+05  2.0040 e-01
 2.7500 e+05  2.2030 e-01
```

```
3.0000 e+05  2.4003 e-01
3.2500 e+05  2.5956 e-01
3.5000 e+05  2.7886 e-01
3.7500 e+05  2.9793 e-01
4.0000 e+05  3.1673 e-01
4.2500 e+05  3.3525 e-01
4.5000 e+05  3.5348 e-01
4.7500 e+05  3.7141 e-01
5.0000 e+05  3.8903 e-01
```

2.3 Astra

Il nome *Astra* è una sigla che sta per *"**A**dvanced **S**oftware **T**ool for **R**eliability **A**nalysis"*. *Astra* [5] è uno strumento per l'analisi di modelli stocastici ed è costituito da un insieme di moduli, uno o più per ogni modello; per gli alberi dei guasti sono stati utilizzati i moduli *Astra-**FTA*** e *Astra-**PTD***, rispettivamente per l'analisi qualitativa e probabilistica e per l'analisi probabilistica dipendente dal tempo.

Astra funziona in ambiente MS-Windows ed è stato sviluppato presso la società **Infocon s. n. c.** di Lavagna (GE) nell'ambito del Centro Richerche della Commissione Europea.

Astra dispone di un'interfaccia grafica che permette più facilmente operazioni quali il caricamento (Fig. 2.2) e la modifica dell'albero (Fig. 2.3), la generazione della sua rappresentazione grafica (Fig. 2.4), l'avvio delle procedure di analisi (Fig. 2.5, 2.6, 2.7), la lettura dei relativi risultati (Fig. 2.10, 2.11, 2.12, 2.13), la generazione di grafici cartesiani o istogrammi.

Astra necessita di un file con estensione **.tfx** che contiene la descrizione dell'albero che va comunque scritto a mano o creato attraverso i traduttori che ho implementato (Astra consente solo di modificare l'albero con operazioni di copia/incolla o di rinominare gli eventi).

La descrizione dell'albero deve ovviamente seguire un determinato formalismo che verrà descritto nel paragrafo successivo.

Inoltre Astra prevede di collezionare un insieme di alberi in progetti e di creare per ogni progetto versioni diverse.

2.3.1 Il formalismo di Astra

Il precedente esempio di albero descritto invece con il formalismo di Astra ha questo aspetto:

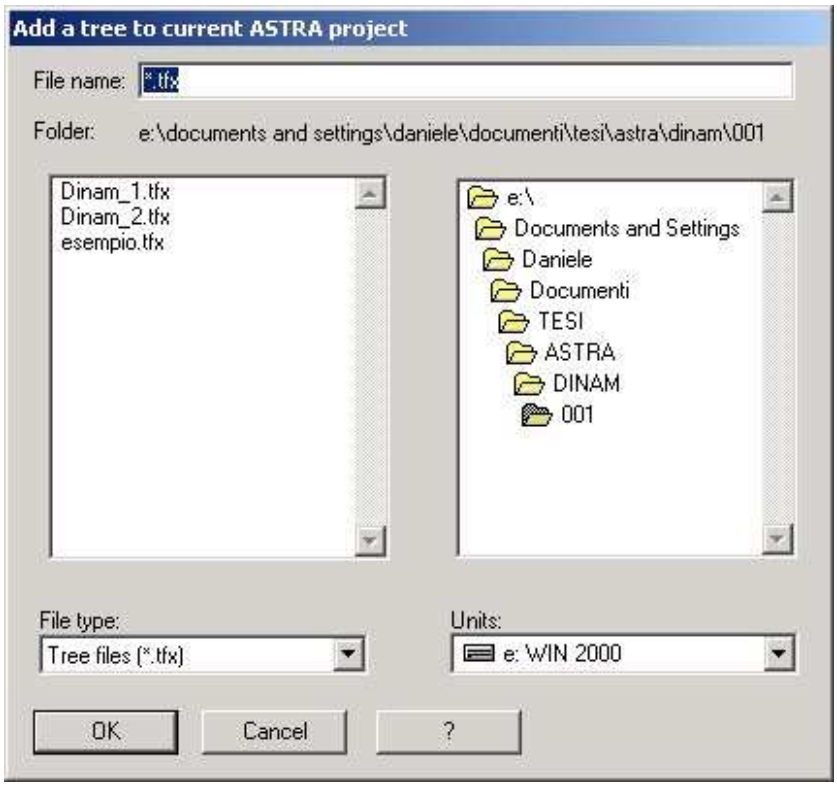


Figura 2.2: Caricamento dell'albero

Figura 2.3: Editor per la modifica dell'albero

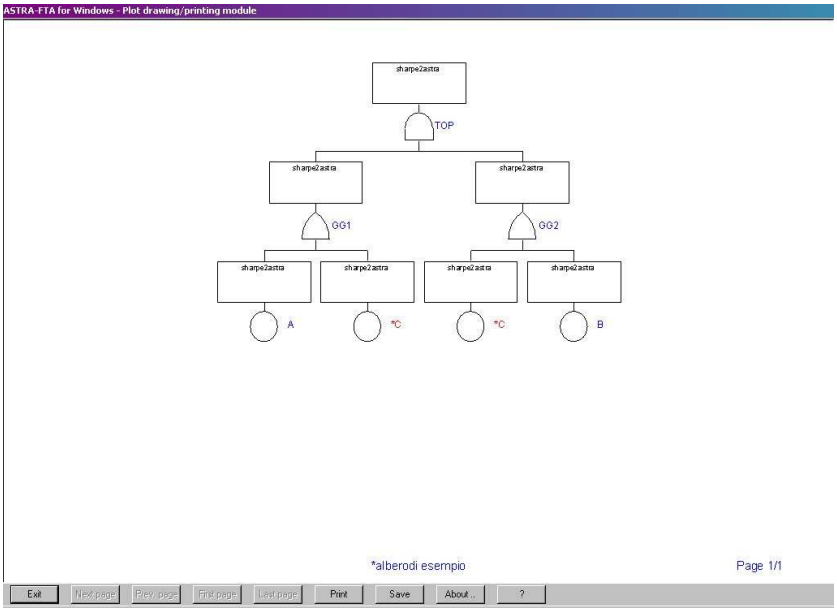


Figura 2.4: Rappresentazione grafica dell'albero (plot)

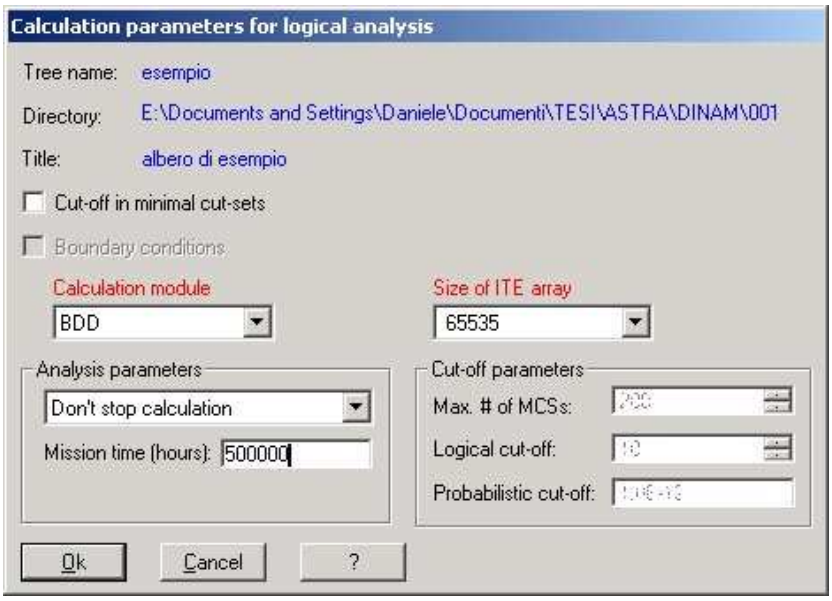


Figura 2.5: Avvio dell'analisi qualitativa

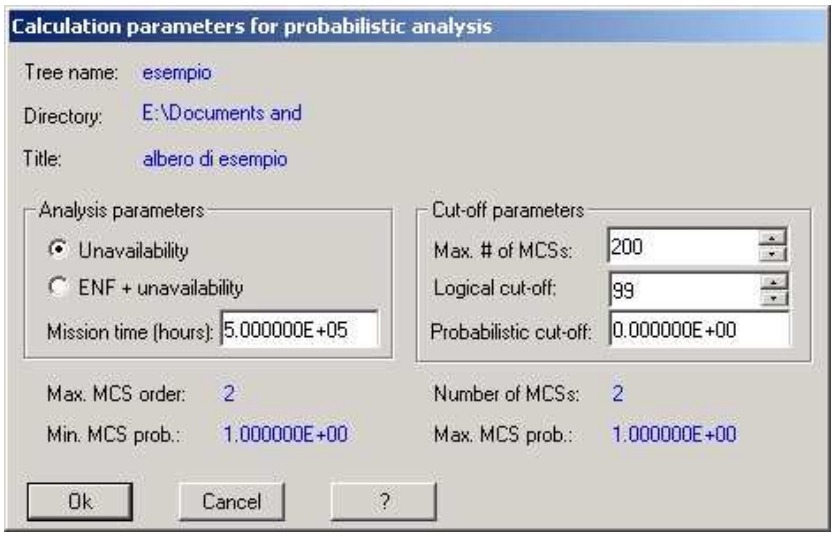


Figura 2.6: Avvio dell'analisi quantitativa

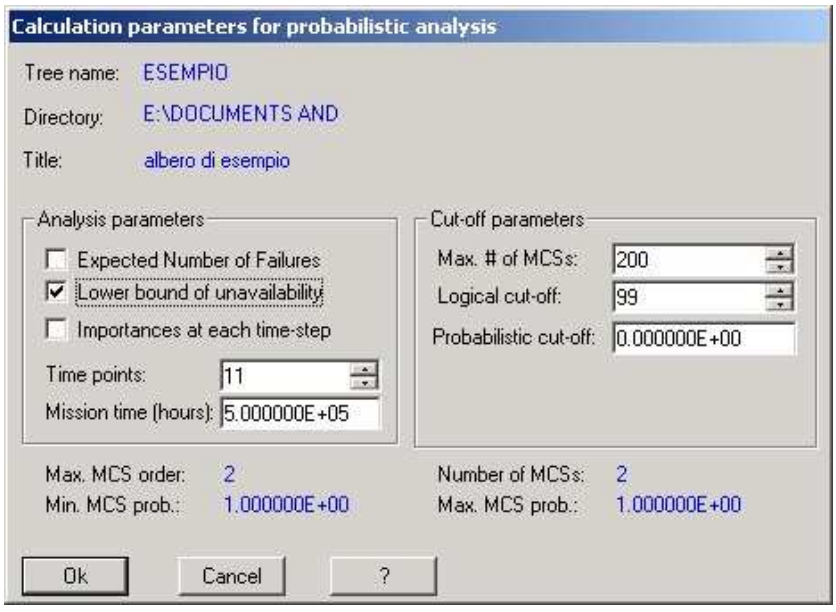


Figura 2.7: Avvio dell'analisi dipendente dal tempo

```
*albero di esempio

[EVENTS]

A 0.00000077 0 0 0 0 "sharpe2astra"
B 0.00000077 0 0 0 0 "sharpe2astra"
C 0.00000077 0 0 0 0 "sharpe2astra"

[GATES]

TOP AND GG1 GG2 "sharpe2astra"
GG2 OR C B "sharpe2astra"
GG1 OR A C "sharpe2astra"

$END
```

Questo file per Astra è stato creato con il traduttore *sharpe2astra* che ho implementato.

Il formalismo di Astra prevede questo ordine di dichiarazioni:

1. titolo o commento
2. dichiarazione eventi di base
3. dichiarazione porte logiche
4. $END

Per quanto riguarda la dichiarazione delle porte logiche, la prima che deve essere dichiarata è quella che ha il *Top Event* come output, mentre le altre possono apparire in qualunque ordine. Astra associa ad ogni nome un evento, per cui se un nome di evento appare più volte nel file, viene sempre inteso come riferimento allo stesso componente o sottosistema.

I nomi degli eventi devono essere composti da caratteri tutti maiuscoli e i nomi degli eventi interni (output di porte logiche) devono cominciare con il carattere G. Astra inoltre non accetta che una porta logica abbia come input degli eventi con lo stesso nome, dato che c'è corrispondenza univoca tra nomi ed eventi.

Le regole per le dichiarazioni sono le seguenti:

- **Eventi di base:**

  `<nome> <val_1> <val_2> <val_3> <val_4> <val_5> <commento>`

dove il nome è quello dell'evento di base a cui seguono cinque valori che sono rispettivamente

- *Failure rate:* tasso di guasto se la distribuzione dell'evento di base è esponenziale;
- *Repair time:* tempo di riparazione se il componente è riparabile;
- *Initial unavailability:* probabilità costante di guasto
- *Test interval e First test time:* rispettivamente intervallo di tempo tra le verifiche di funzionamento del componente e tempo in cui avviene la prima verifica (se il componente è riparabile).

L'ultimo campo è un commento all'evento: se il file per Sharpe è stato generato con *sharpe2astra* apparirà questo nome come commento.

In un albero possono essere presenti fino a 2048 eventi di base secondo Astra.

- **Porte logiche:**

```
<nome evento ouput> <tipo porta logica> <nomi input> <commento>
```

I tipi di porta logica supportati da Astra sono i seguenti:

- *AND*
- *OR*
- *k:n* (k out of n) con indicazione esplicita di k e n, ad esempio

  ```
  G1 2/3 A B C
  ```

- *INH* (And con due eventi di input di cui uno si deve verificare prima dell'altro)
- XOR (Or esclusivo)

Astra permette ad una porta logica di avere fino a 8 eventi di input e il numero massimo di porte logiche presenti in un albero dei guasti è 2048.

2.3.2 Struttura di Astra-FTA

*Astra-**FTA*** consente di eseguire in successione l'analisi *qualitativa*, in cui determina i *minimal cutset* (**MCS**), e l'analisi *quantitativa* per il calcolo dei valori numerici.

L'analisi *qualitativa* consiste delle seguenti fasi:

- **Semplificazione dell'albero:** in questa fase l'albero di input viene trasformato in un nuovo albero logicamente identico a quello iniziale, ma più semplice, basato esclusivamente sulle porte logiche AND, OR e NOT. In particolare le porte *K out of N*, *XOR* e *INH* sono tradotte nell'equivalente espressione booleana con AND, OR e NOT. Successivamente i sottoalberi negati dalla porta NOT vengono complementati usando le leggi di **De Morgan**.

- **Modularizzazione (o decomposizione per moduli):** a questo punto l'albero viene decomposto in moduli; con questo termine si intendono sottoalberi indipendenti dal resto dell'albero ovvero sottoalberi che non hanno parti condivise con altri. Questa operazione consente di dividere alberi di grandi dimensioni in diversi alberi più piccoli ed indipendenti tra loro che possono essere analizzati individualmente e i cui risultati possono essere combinati tra loro per ottenere l'insieme di **MCS** dell'albero originale.

- **Individuazione dei MCS:** Astra-FTA dispone di due tecniche per determinare i MCS tra cui l'utente può scegliere tramite l'interfaccia grafica al momento di lanciare l'analisi qualitativa. I due metodi sono i seguenti:

  - *metodo BDD*: basato su diagrammi di decisione binaria;
  - *metodo bottom-up*: basato sull'Algebra booleana.

  Questi metodi e l'analisi *quantitativa* saranno approfonditi nel paragrafo successivo.

- **Unificazione dei risultati:** i vari risultati ottenuti per ogni modulo vengono combinati per ottenere l'insieme di MCS dell'intero albero dato in input.

2.3.3 Analisi qualitativa tramite Diagrammi di decisione binaria

Il metodo basato sui *diagrammi di decisione binaria* (**BDD**) rappresenta la funzione booleana corrispondente all'albero dei guasti come *grafo diretto aciclico* (**DAG**), il quale viene a sua volta ridotto ad un grafo più semplice da cui si ricavano finalmente tutti i cutset minimi.

Lo spazio di memoria occupata per eseguire il procedimento può crescere rapidamente, in particolare quando ci sono molti eventi presenti in più punti dell'albero e molti di questi sparsi, cioè presenti vicino sia al *Top Event* sia alla frontiera dell'albero.

Prendiamo come esempio l'albero di Fig. 2.4 (o di Fig. 2.1); l'espressione booleana corrispondente è la seguente:

$$(a \vee c) \wedge (c \vee b)$$

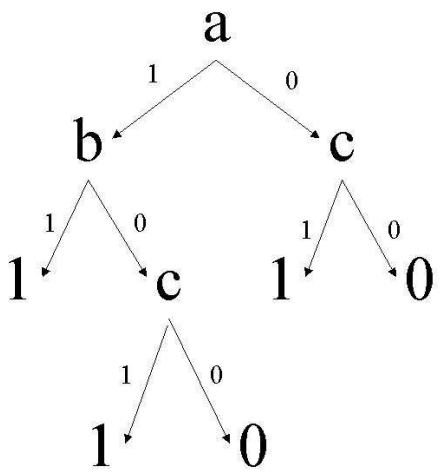


Figura 2.8: Albero di decisione corrispondente all'albero dei guasti di Fig. 2.4

L'albero di decisione di Fig. 2.8 è stato costruito applicando all'espressione booleana la *decomposizione di* ***Shannon*** che consiste nel ricavare un'altra espressione booleana corrispondente all'originale quando una delle

variabili assume valore 1 o 0:

$$F(x_1, x_2, ..., x_i - 1, x_i, x_i + 1, ..., x_n) = x_i F_1 + \neg x_i F_0$$

dove
$F_1 = f(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$
$F_0 = f(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n)$
F_1 rappresenta F quando x_i è pari a 1, mentre F_0 quando x_i è uguale a 0.

La decomposizione di Shannon può essere rappresentata da un grafo i cui vertici non terminali rappresentano le variabili; ogni vertice ha due archi uscenti, il sinistro per F_1, il destro per F_0; i vertici terminali indicano il valore finale dell'espressione dopo aver assegnato un valore a tutte le variabili. L'albero di decisione si ottiene quindi applicando la decomposizione di Shannon rispetto ad una variabile, dopodichè si decompone ricorsivamente F_1 e F_0 risultanti rispetto ad una seconda variabile e così via fino a raggiungere un valore finale, 1 o 0.

L'ordine con cui si considerano le variabili rispetto alle quali decomporre ha un effetto rilevante sulla complessità del grafo e sul numero di operazioni necessarie poi per ridurlo; nell'esempio in questione l'ordine seguito è a, b, c.

Una volta creato il diagramma di decisione bisogna ridurlo ad una forma più semplice applicando le seguenti regole:

1. fondere i nodi terminali che hanno lo stesso valore (1 o 0);
2. fondere i nodi non terminali che rappresentano la stessa variabile e che hanno i medesimi discendenti;
3. rimuovere i vertici ridondanti, cioè quei vertici che hanno entrambi gli archi uscenti diretti verso lo stesso vertice.

Il diagramma di Fig. 2.8 diviene, secondo queste regole, quello di Fig. 2.9; ogni vertice si può rappresentare da una tripla (x, L, R) che indica il nome del vertice, il nome del vertice discendente sinistro e il nome del discendente destro; questo tipo di rappresentazione prende il nome di **ite** dato che la tripla si può interpretare come

if x then L else R.

Ad esempio la rappresentazione per il vertice a del diagramma è $ite(a, b, c)$.

Componendo le varie triple si può dare la rappresentazione **ite** dell'intero diagramma di Fig. 2.9; nel caso di esempio si ricava:

$$ite(a, ite(b, 1, ite(c, 1, 0)), ite(c, 1, 0))$$

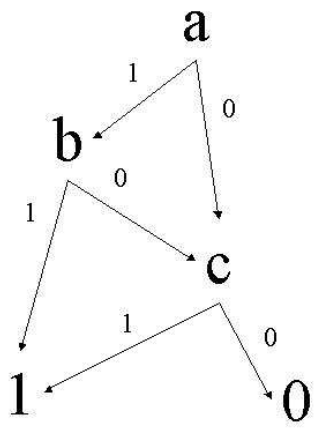


Figura 2.9: Albero di decisione semplificato

in modo più sintetico $Z = ite(x, F, G)$; Z indica l'intero grafo, x è la radice (a), F il sottografo discendente sinistro di a, G il sottografo discendente destro. Da questa notazione si possono ricavare i cutset minimi di Z dati da:

MCS di $Z =$ ($x \wedge$(*tutti i cammini di* F *che non contengono nessun cammino di* G))$\cup$ (*tutti i cammini di* G)

Nell'esempio in considerazione, $MCS = ab, c$.

Infatti i cammini di F che portano ad una soluzione sono b e bc, mentre quello di G è c e x corrisponde al vertice a; quindi l'unico cammino di F che non ha cammini in comune con G è b che viene concatenato a a formando ab, cioè il primo MCS. L'altro MCS è l'unico cammino di G, cioè c.

2.3.4 Analisi qualitativa tramite il metodo bottom-up

Il metodo **BDD** consente di calcolare rapidamente e con esattezza i *cutset minimi*, ma ha il difetto che lo spazio di memoria richiesto per memorizzare tutte le rappresentazioni **ite** di un albero di grandi dimensioni con parti ripetute può eccedere la memoria a disposizione.

In questi casi l'utente può ricorrere all'utilizzo del metodo *bottom-up* (**BU**); questo prevede la conversione in *forma normale disgiuntiva* (**DNF**)

dell'espressione booleana dell'albero attraverso le regole dell'Algebra booleana.

Il metodo prende la denominazione di *bottom-up* in quanto l'espressione booleana risultante dall'albero dei guasti in esame viene derivata valutando in ordine "bottom-up" gli eventi dell'albero facendone una visita in profondità.

Il metodo è lo stesso con cui si effettuava l'analisi *qualitativa* nel paragrafo 1.3.2.

L'utente ha la possibilità di richiedere ad Astra-FTA di calcolare solo i cutset minimi più significativi (**SMCS**); per fare questo vengono applicate delle tecniche di *"cut-off"* per selezionare dai MCS i SMCS:

- **cut-off logico n_{lim}:** solo i cutset minimi di ordine $n \leq n_{lim}$ sono considerati significativi; si basa sulla considerazione che la probabilità di un MCS di verificarsi è inversamente proporzionale al suo ordine; questo è vero se si assume che tutti gli eventi di un MCS abbiano tutti uguale probabilità, ipotesi che non è molto realistica; per questo il **cut-off logico** è utile giusto per analisi preliminari;

- **cut-off probabilistico** P_{lim}: sono significativi quei cutset minimi la cui probabilità di verificarsi supera o eguaglia il valore P_{lim}; con questo caso la rilevanza di un MCS è legata alla sua probabilità, rendendo questo criterio molto più utile del precedente.

2.3.5 L'analisi quantitativa di Astra

Astra richiede all'utente un tempo rispetto al quale calcolare i vari valori di probabilità detto **tempo di missione** che verrà indicato con T.

Considerando eventi non riparabili, Astra calcola i seguenti indici:

- **Probabilità di guasto al tempo T dell'evento di base E**:

$$P^E(T) = 1 - e^{\lambda T} + P^E(0)e^{-\lambda T}$$

  dove λ è il parametro della distribuzione esponenziale e $P^E(0)$ è la probabilità di guasto al tempo 0; $P^E(0)$ è opzionale e di default è impostato a 0; per indicare un evento con probabilità iniziale p si imposta $\lambda = 0$ e $P^E(0) = p$.

- **Probabilità di guasto al tempo T dell'evento di base E con probabilità costante P_0**:

$$P^E(T) = P_0 \text{ (indipendente dal tempo)}$$

- **Probabilità del verificarsi al tempo T del cutset minimo C di ordine n**:

$$P_k^C(T) = \prod_{i=1}^{n} P_i^E(T)$$

- **Probabilità di guasto al tempo T del sistema S (*Top Event*)**: dato un insieme di MCS, ad esempio A, B, C, se si volesse calcolare questo indice esattamente si dovrebbe tener conto del principio di inclusione-esclusione dell'unione, cioè:

  $P(A \vee B \vee C) = P(A) + P(B) + P(C) - P(A \wedge B) - P(A \wedge C) - P(B \wedge C) + P(A \wedge B \wedge C)$

  Se il numero di cutset minimi è elevato la lunghezza di questa espressione può crescere notevolmente; in particolare se il numero di MCS è N_{MCS} il numero dei termini sarà $2^{N_{MCS}} - 1$ per questo Astra-FTA non calcola di questo indice il valore esatto, ma si limita a definirne un limite superiore, più precisamente:

$$P^S(T) \leq P_U^S(T) = \sum_{k=1}^{N} P_k^C(T)$$

Astra-FTA inoltre calcola degli indici opzionali (se indicati dall'utente), cioè il numero medio di guasti (**ENF**) dei componenti di base, dei MCS e del sistema durante un certo tempo T; inoltre determina il *grado di importanza* di eventi e cutset, cioè il grado con cui contribuiscono al guasto dell'intero sistema.

Mentre il modulo Astra-FTA calcola i suoi indici al tempo T indicato dall'utente, il modulo *Astra-**PTD*** determina gli stessi indici non solo al tempo T, ma per un certo numero di valori di tempo compresi tra 0 e T; ad esempio se l'utente ha indicato T (tempo di missione) pari a 1000 e desidera l'analisi in 11 momenti dell'intervallo di tempo da 0 a T, il modulo *Astra-**PTD*** farà una valutazione per t uguale a $0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000(T)$.

Le formule precedenti non saranno applicate solo rispetto a T, ma rispetto a t, per il suo valore in ogni momento indicato.

Quindi Astra-PTD restituirà, ad esempio, la probabilità di guasto del sistema per $t = 0, t = 100, t = 200, ..., t = 1000$. In questo caso Astra-PTD non fornisce solo il limite superiore indicato sopra per il guasto del sistema, ma anche un ulteriore limite superiore calcolato diversamente e un limite inferiore (quest'ultimo se richiesto dall'utente):

- **limite superiore di *Esary-Proschan***:

$$P^S(t) \leq 1 - \prod_{k=1}^{N} (1 - P_k^C(t))$$

Questa formula è conveniente se le probabilità di guasto dei MCS sono alte; inoltre se i MCS sono indipendenti tra loro, questo limite diventa il valore esatto della probabilità di guasto del sistema e comunque non può avere un valore superiore a 1, a differenza del precedente limite superiore.

- **limite inferiore**:

$$P^S(t) \geq \sum_{k=1}^{N} P_k^C(t) - \sum_{j=1}^{N-1} \sum_{j=1}^{N} Q_{ij}(t)$$

dove $Q_{ij}(t)$ rappresenta la probabilità di intersezione del i-esimo e j-esimo cutset minimo; se i due cutset sono indipendenti tra loro corrisponde al prodotto delle loro probabilità di verificarsi, altrimenti viene calcolato come

$$Qij(t) = \frac{P_i^C(t) \cdot P_j^C(t)}{P_{ij}(t)}$$

in cui $P_{ij}(t)$ è il prodotto delle probabilità di guasto degli eventi comuni ai due MCS.

2.3.6 La lettura dei risultati

Seguono le figure riguardanti i risultati calcolati con Astra sull'albero di esempio di Fig. 2.4 (e di Fig. 2.1).

Confrontando i risultati sullo stesso albero dati da Sharpe e Astra, si può notare che i valori di $F(t)$ restituiti da Sharpe (esatti) sono compresi nella Fig. 2.13, relativa ad Astra, tra i valori di *Unavailability* (limite superiore) e il limite inferiore (*lower bound*) per lo stesso t.

Il limite superiore di *Esary-Proshan* corrisponde ai valori dati da Sharpe in quanto i MCS di questo esempio sono indipendenti (non hanno eventi in comune); anche il limite inferiore corrisponde ai valori esatti in quanto, essendo i cutset minimi, solo due, c e ab, la relativa formula applicata all'esempio equivale al calcolo esteso della probabilità del sistema.

Al termine del Cap. 3 ci saranno ulteriori casi di esempio per confrontare più in dettaglio i risultati forniti da Sharpe e Astra.

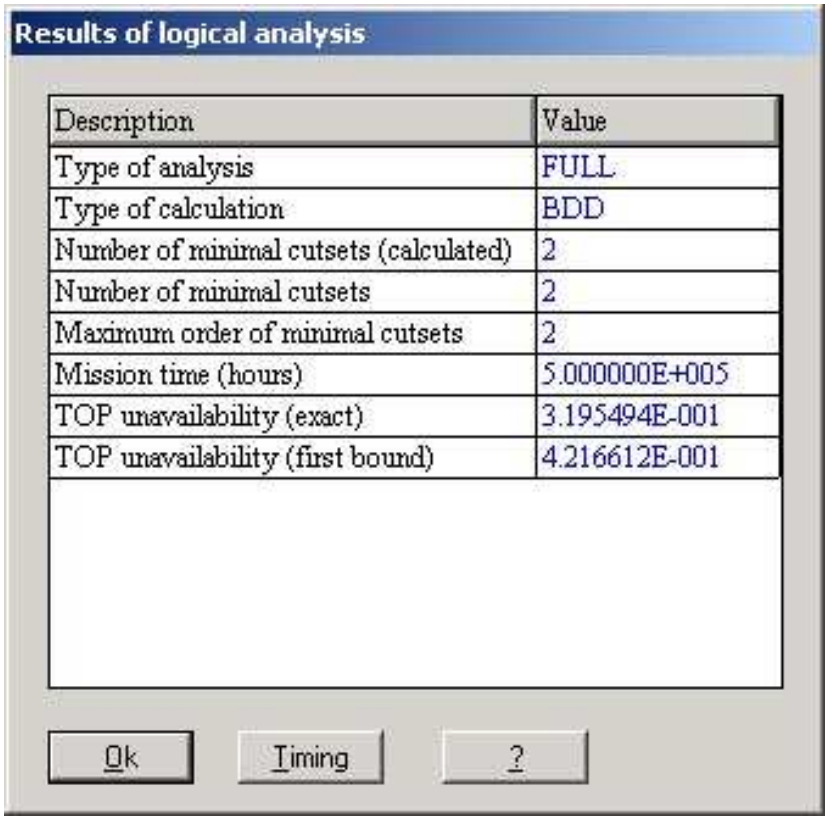


Figura 2.10: Risultati dell'analisi logica

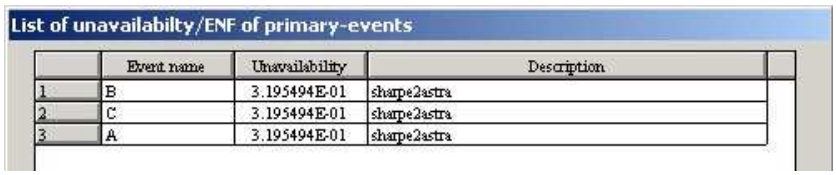


Figura 2.11: Probabilità di guasto dei componenti al tempo di missione

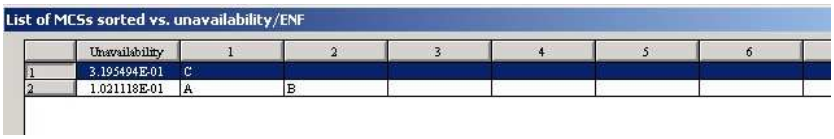
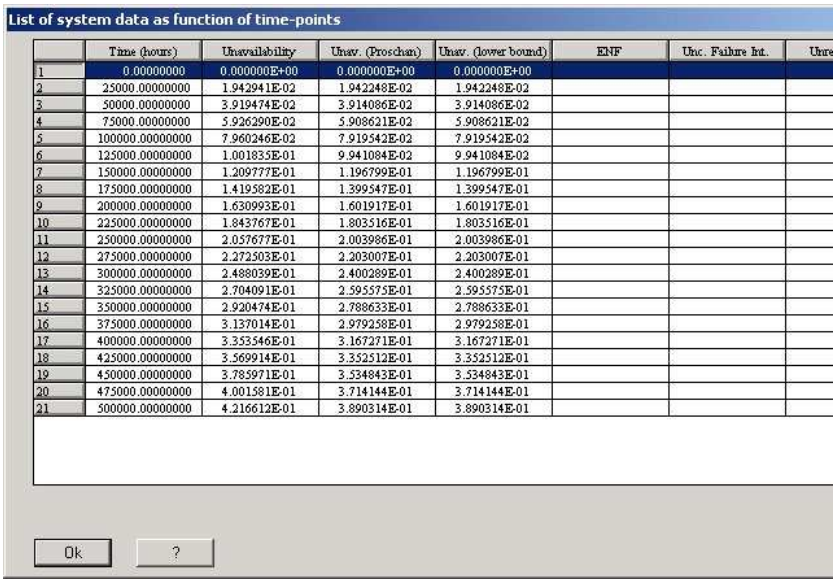


Figura 2.12: MCS e relativa probabilità al tempo di missione



List of system data as function of time-points

	Time (hours)	Unavailability	Unav. (Proschan)	Unav. (lower bound)	ENF	Unc. Failure Int.	Unre
1	0.00000000	0.000000E+00	0.000000E+00	0.000000E+00			
2	25000.00000000	1.942941E-02	1.942248E-02	1.942248E-02			
3	50000.00000000	3.919474E-02	3.914086E-02	3.914086E-02			
4	75000.00000000	5.926290E-02	5.908621E-02	5.908621E-02			
5	100000.00000000	7.960246E-02	7.919542E-02	7.919542E-02			
6	125000.00000000	1.001835E-01	9.941084E-02	9.941084E-02			
7	150000.00000000	1.209777E-01	1.196799E-01	1.196799E-01			
8	175000.00000000	1.419582E-01	1.399547E-01	1.399547E-01			
9	200000.00000000	1.630993E-01	1.601917E-01	1.601917E-01			
10	225000.00000000	1.843767E-01	1.803516E-01	1.803516E-01			
11	250000.00000000	2.057677E-01	2.003986E-01	2.003986E-01			
12	275000.00000000	2.272503E-01	2.203007E-01	2.203007E-01			
13	300000.00000000	2.488039E-01	2.400289E-01	2.400289E-01			
14	325000.00000000	2.704091E-01	2.595575E-01	2.595575E-01			
15	350000.00000000	2.920474E-01	2.788633E-01	2.788633E-01			
16	375000.00000000	3.137014E-01	2.979258E-01	2.979258E-01			
17	400000.00000000	3.353546E-01	3.167271E-01	3.167271E-01			
18	425000.00000000	3.569914E-01	3.352512E-01	3.352512E-01			
19	450000.00000000	3.785971E-01	3.534843E-01	3.534843E-01			
20	475000.00000000	4.001581E-01	3.714144E-01	3.714144E-01			
21	500000.00000000	4.216612E-01	3.890314E-01	3.890314E-01			

Ok ?

Figura 2.13: Probabilità di guasto del sistema per t che va da 0 a 500000

Capitolo 3

Il traduttore da Sharpe a Astra

3.1 Introduzione

Esaminare un sistema rappresentato da un albero dei guasti tramite strumenti di analisi diversi, *Sharpe e Astra*, ha necessitato la creazione di un traduttore che, data la rappresentazione di un albero nel formalismo di Sharpe, creasse un file contenente un albero logicamente uguale, ma descritto nel formalismo di Astra.

Impiegando due strumenti di analisi per analizzare lo stesso sistema, è possibile confrontarli valutando ad esempio quale dei due strumenti ha maggiore grado di precisione o quale mette a disposizione più possibilità di personalizzazione del risultato.

3.2 Differenze tra i formalismi di Sharpe e Astra

Sintetizzando quando già descritto nel capitolo precedente, si possono individuare le principali differenze tra i due formalismi nei seguenti aspetti:

- **ordine delle dichiarazioni** di eventi e porte logiche
  - *Sharpe*: bottom-up;
  - *Astra*: prima gli eventi di base, poi la porta con output sul Top Event seguita dalle altre porte in qualunque ordine;
- **nomi degli eventi**
  - *Sharpe*: caratteri maiuscoli e minuscoli;

  - *Astra*: solo caratteri maiuscoli e per gli eventi interni la prima lettera deve essere G;

- indicazione del **tipo di distribuzione** di probabilità

  - *Sharpe*: esistono vari tipi di distribuzione e si indicano esplicitamente dopo il nome dell'evento, ad esempio

    ```
    repeat BE prob(0.000025)
    ```

  - *Astra*: sono disponibili la distribuzione esponenziale e quella costante che vengono assegnate eplicitando rispettivamente il primo e il terzo valore successivi al nome dell'evento di base; se si indicano entrambi si può specificare una distribuzione di tipo esponenziale con definizione della probabilità di guasto al tempo 0; si può inoltre specificare un tasso di riparazione per i componenti riparabili.

- posizione dei **commenti**

  - *Sharpe*: possono essere inseriti ovunque preceduti da *;
  - *Astra*: si mette un commento all'inizio del file e un commento per ogni evento di base o porta logica.

La traduzione da Sharpe a Astra non sarebbe che una semplice modifica dell'ordine e della forma delle dichiarazioni di eventi di base e porte logiche se non fosse per il fatto che Sharpe consente di dichiarare un unico input per una porta **K out of N** intendendo che tale porta ha n input uguali e distinti tra loro.

Infatti Astra non permette che una porta logica abbia un unico input o più input con lo stesso nome; per questo è stato necessario in situazioni di questo genere duplicare l'input in un certo numero di copie.

Esempio:

```
*Sharpe
kofn 2,4, TOP in
```

diventa

```
TOP 2/4 IN1 IN2 IN3 IN4 "Astra"
```

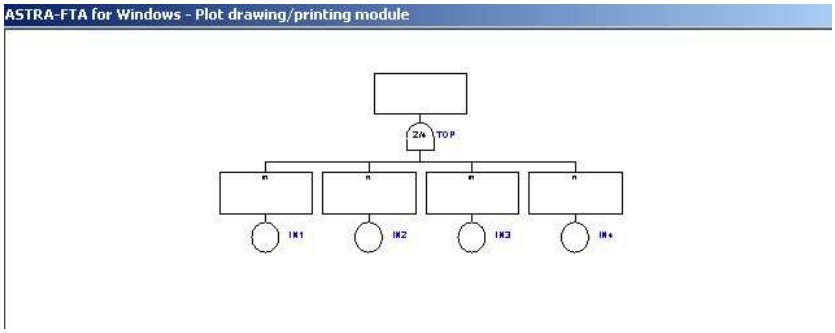


Figura 3.1: Albero con eventi duplicati

La Fig. 3.1 mostra questo semplice albero come viene visualizzato in Astra.

Se l'evento da duplicare è un evento di base la soluzione è semplice, mentre se si tratta di un evento interno bisogna duplicare l'intero sottoalbero sottostante tale evento.

Ogni copia prende il nome dell'originale seguito da un indice il cui valore può variare da 1 al numero di copie necessarie; se l'evento duplicato è interno si aggiungerà ad ogni nome di evento contenuto nel sottoalbero duplicato lo stesso indice dell'evento interno alla radice del sottoalbero.

Un nome di evento può essere seguito anche da più di un indice se un sottoalbero duplicato si trova all'interno di uno o più sottoalberi duplicati.

Esempio:

File Sharpe:

```
* esempio k:n con 1 input

ftree duplica

basic be exp(v1)

or int be be be

kofn TOP 2,3, int

end

bind v1 0.0000001
```

```
end

expr mean(duplica)

expr variance(duplica)

end
```

File Astra corrispondente:

```
*esempio k:n con 1 input

[EVENTS]

BE11 0.0000001 0 0 0 0 "sharpe2astra"
BE21 0.0000001 0 0 0 0 "sharpe2astra"
BE31 0.0000001 0 0 0 0 "sharpe2astra"
BE12 0.0000001 0 0 0 0 "sharpe2astra"
BE22 0.0000001 0 0 0 0 "sharpe2astra"
BE32 0.0000001 0 0 0 0 "sharpe2astra"
BE13 0.0000001 0 0 0 0 "sharpe2astra"
BE23 0.0000001 0 0 0 0 "sharpe2astra"
BE33 0.0000001 0 0 0 0 "sharpe2astra"

[GATES]

TOP 2/3 GINT1 GINT2 GINT3 "sharpe2astra"
GINT3 OR BE13 BE23 BE33 "sharpe2astra"
GINT2 OR BE12 BE22 BE32 "sharpe2astra"
GINT1 OR BE11 BE21 BE31 "sharpe2astra"

$END
```

La Fig. 3.2 mostra come appare tale albero in Astra.

3.3 Struttura del traduttore

Il traduttore da Sharpe a Astra è stato implementato in linguaggio C ed è composto dalle seguenti parti:

1. Parser

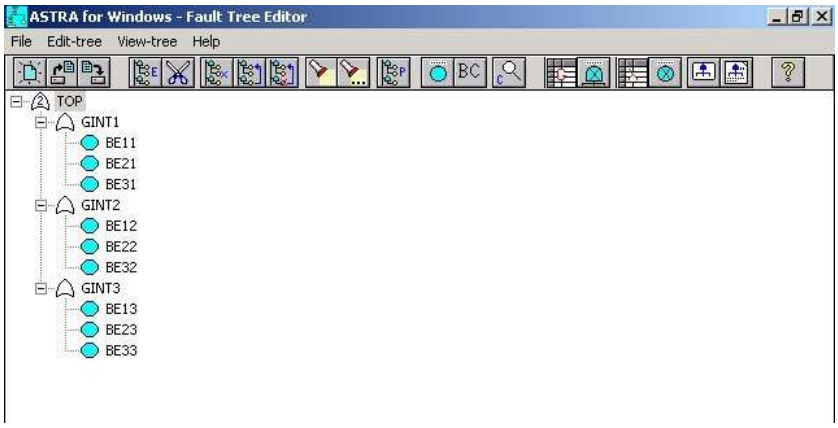


Figura 3.2: Albero con eventi duplicati

2. Duplicazione di eventi
3. Conversione dei nomi degli eventi nella forma di Astra
4. Controllo di compatibilità con Astra
5. Generazione del file in formato Astra (estensione .tfx)

Viene lanciato da prompt dei comandi con la seguente sintassi:

```
sharpe2astra <nome del file per Sharpe> <nome del file per
              Astra con estensione .tfx>
```

3.3.1 Parser

Il parser opera in due fasi:

1. Individuazione delle costanti
2. Caricamento di eventi di base e porte logiche

Nella prima fase vengono ricercate le dichiarazioni di costanti all'interno del file, comprese tra la parola chiave *bind* e il relativo *end*, e vengono caricati in memoria i loro nomi e i rispettivi valori. Le costanti vengono individuate prima del caricamento della struttura dell'albero in modo tale che ogni riferimento ad una delle costanti venga immediatamente sostituito con il valore corrispondente.

La fase successiva, che è il parser vero e proprio, carica eventi di base e porte logiche; per ognuno di questi elementi, nel file per Sharpe corrisponde una riga; la prima parola della riga deve essere una parola chiave che indica il tipo di porta logica o di evento.

- Se la parola chiave è *basic* o *repeat* si tratta di un **evento di base**; a questo punto viene salvato il nome dell'evento e se ne deve individuare la distribuzione di probabilità assegnatagli; il parser è in grado di riconoscere la distribuzione esponenziale, quella costante e i relativi parametri; non sono state prese in considerazione altri tipi di distribuzione di Sharpe in quanto solo i suddetti sono supportati da Astra.

- Le **porte logiche** sono caratterizzate dalle parole chiave *and, or, kofn*; se la porta è di tipo *and* oppure *or*, ne viene salvato il nome, anticipato dal carattere G, e il tipo; se, invece, si tratta di *kofn* vengono caricati anche i valori dei parametri k ed n; tutte le stringhe di caratteri che seguono sulla stessa riga vengono considerate come i nomi degli eventi di input della porta logica. Se due o più eventi di input hanno lo stesso nome, essi vengono rinominati aggiungendo un indice al loro nome ed eventualmente generando una copia distinta del sottoalbero sottostante se si tratta di eventi interni.

  Nell'esempio di Fig. 3.2 l'evento *be* viene duplicato in 3 copie distinte: $BE1, BE2, BE3$.

- I **commenti** sono preceduti da *; il primo di questi che si incontra viene copiato nel file per Astra di output come titolo dell'albero, mentre tutti gli altri vengono ignorati; tipicamente in un file per Sharpe sulla prima riga si scrive un commento per descrivere l'albero.

La Fig. 3.3 mostra il diagramma a stati del parser.

3.3.2 Duplicazione di eventi

Già durante la fase di *parsing* possono essere duplicati degli eventi se si presentano più input di una porta logica con lo stesso nome. Questo è possibile perché essendo l'ordine delle dichiarazioni in Sharpe di tipo bottom-up, gli input di una porta logica devono essere già definiti nelle righe precedenti e quindi al momento già caricati, conoscendone la struttura o i parametri.

Dopo la fase di *parsing*, quando si conosce completamente la struttura dell'albero, si verifica, per ogni porta logica di tipo *kofn*, se questa ha un

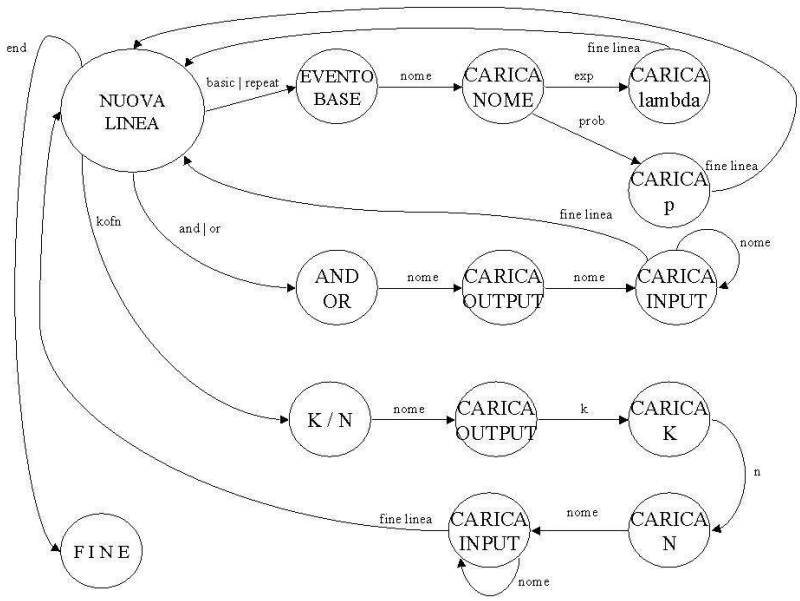


Figura 3.3: Diagramma a stati del parser

unico evento di input dichiarato; in questo caso nel formalismo di Sharpe si intende che la porta ha n input distinti e uguali all'evento dichiarato. Bisogna quindi duplicare tale evento n volte e, se si tratta di un evento interno, anche il sottoalbero sottostante a tale evento.

La procedura di duplicazione per gli eventi di base semplicemente crea una copia dell'originale che ha la stessa distribuzione di probabilità e che ha come nome quello dell'evento originale seguito da un indice numerico; per gli eventi interni, si ricerca la definizione della porta logica con quell'evento come output e si crea una nuova definizione di porta in cui tutti i nomi sono seguiti dal valore dell'indice; ricorsivamente si fa una duplicazione allo stesso modo degli eventi di input della porta logica in questione con lo stesso valore di indice.

Nell'esempio di Fig. 3.2 vengono generate 3 copie dell'evento **int** e dato che si tratta di un evento interno viene duplicato anche ciò che si trova sotto; quindi $BE1, BE2, BE3$ vengono ulteriormente replicati in $BE11, BE21, BE31$ per $GINT1$ e analogamente per $GINT2$ e $GINT3$.

3.3.3 Conversione dei nomi nel formato di Astra

Astra esige nomi di evento con tutti i caratteri maiuscoli; per quanto riguarda i nomi di eventi interni, già in fase di parsing si era inserita la lettera G all'inizio dei loro nomi; in questa fase per ogni nome di evento, ogni carattere minuscolo viene convertito nel corrispondente maiuscolo; fa eccezione il *Top Event* al quale viene assegnato arbitrariamente il nome TOP secondo il formalismo di Astra.

3.3.4 Controllo di compatibilità con Astra

A questo punto l'albero è stato caricato, alcune sue parti, se necessario, sono state duplicate e i nomi rispettano le regole di Astra; bisogna ancora però fare dei controlli perchè il formalismo di Astra sia completamente rispettato; questi controlli riguardano:

1. la lunghezza del commento, al massimo 40 caratteri;
2. la lunghezza dei nomi degli eventi, al massimo 10 caratteri;
3. se vi sono dei parametri numerici espressi nella forma $1 - p$, dove p è una probabilità, si deve verificare che tale differenza non sia nulla o negativa;
4. solo i nomi di eventi interni, output di porte logiche, possono cominciare per G.

Se si riscontrano degli errori viene chiesto all'utente se vuole comunque generare il file per Sharpe; questo perchè in alcuni casi, anche se vi sono degli errori di forma segnalati dal traduttore, il file per Astra può andare bene comunque. Ad esempio, se il commento è troppo lungo, esso verrà troncato oppure se il nome di un evento supera i 10 caratteri, sarà anch'esso troncato; in quest'ultimo caso però tagliare gli ultimi caratteri, se questi servono a distinguere un evento da un altro, potrebbe generare un albero logicamente diverso dall'originale!

3.3.5 Generazione del file per Astra

Questa è la fase finale; in un file che deve avere estensione **.tfx** vengono scritti nell'ordine:

1. **Il commento**;

2. **gli eventi di base** preceduti dalla parola chiave [EVENTS]; dopo il nome di ogni evento seguono i suoi parametri; dato che le distribuzioni supportate da Astra sono solo due e che Sharpe non prevede riparazioni di componenti, i parametri degli eventi di base che possono essere specificati possono solo essere il primo o il terzo, mentre tutti gli altri verranno indicati con valore nullo; Astra richiede un commento per ogni evento: il commento è lo stesso per tutti gli eventi ("sharpe2astra") e indica che il file è stato generato dal traduttore.

3. **Le porte logiche**, precedute dalla parola chiave [GATES], indicando evento di output, tipo di porta, eventi di input e relativo commento; la prima porta logica che viene dichiarata è quella con TOP come evento di output, dopodiché seguono tutte le altre.

4. La parola chiave $END che indica la fine del file.

La Fig. 3.4 è un riepilogo delle fasi del traduttore.

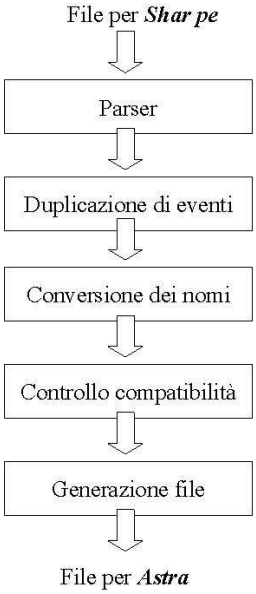


Figura 3.4: Input, fasi del traduttore e output

3.4 Strutture dati utilizzate

Le principali strutture dati utilizzate riguardano la memorizzazione di eventi di base e porte logiche; gli eventi di base sono contenuti in due vettori di strutture (un vettore per gli eventi con probabilità costante ed uno per quelli con distribuzione esponenziale) che hanno i seguenti campi:

- **nome**: stringa che contiene il nome dell'evento;
- **valore**: stringa che contiene il parametro di probabilità; è una stringa anziché un numero reale per poter contenere espressioni nella forma $1 - p$.

Le porte logiche sono invece memorizzate in un array di strutture con i campi seguenti:

- **nome**: stringa che contiene il nome dell'evento di output;
- **tipo**: stringa che indica il tipo di porta logica;
- **nodi**: stringa che contiene i nomi degli eventi di input;
- **num_nodi**: numero di eventi di input della porta.

Ci sono poi delle variabili che indicano il numero di porte logiche, il numero di eventi, il numero di errori, ecc.

3.5 Confronto tra risultati

In questo paragrafo verranno esaminati alcuni alberi dei guasti con Sharpe e con Astra per confrontare i risultati forniti dai due strumenti di analisi e spiegare le cause delle eventuali differenze.

3.5.1 Esempio 1

L'albero è quello di Fig.1.1; il relativo file per Sharpe è il seguente:

```
*esempio1

ftree primo

repeat a exp(v1)
repeat b exp(v1)
```

```
repeat c exp(v2)
repeat d exp(v2)
repeat e exp(v3)
repeat f exp(v3)

and G1 a b d
and G2 d e f
or TOP G1 c G2

end

bind
v1 0.0000001
v2 0.0000002
v3 0.0000003
end

format 8

eval(primo) 0 500000 25000

end
```

Viene richiesta la probabilità di guasto del sistema per t che varia tra 0 e 500000 con un passo di 25000; questi sono i risultati restituiti da Sharpe:

```
      system primo
        t             F(t)

 0.00000000 e+00  0.00000000 e+00
 2.50000000 e+04  4.98782881 e-03
 5.00000000 e+04  9.95259480 e-03
 7.50000000 e+04  1.48961388 e-02
 1.00000000 e+05  1.98202000 e-02
 1.25000000 e+05  2.47264195 e-02
 1.50000000 e+05  2.96163442 e-02
 1.75000000 e+05  3.44914300 e-02
 2.00000000 e+05  3.93530455 e-02
 2.25000000 e+05  4.42024748 e-02
 2.50000000 e+05  4.90409213 e-02
 2.75000000 e+05  5.38695103 e-02
 3.00000000 e+05  5.86892924 e-02
```

```
 3.25000000 e+05  6.35012460 e-02
 3.50000000 e+05  6.83062807 e-02
 3.75000000 e+05  7.31052395 e-02
 4.00000000 e+05  7.78989016 e-02
 4.25000000 e+05  8.26879853 e-02
 4.50000000 e+05  8.74731502 e-02
 4.75000000 e+05  9.22549997 e-02
 5.00000000 e+05  9.70340834 e-02
```

Calcoliamo gli stessi indici con Astra; prima di tutto bisogna tradurre il file per Sharpe nel formalismo di Astra:

```
*esempio1

[EVENTS]
A 0.0000001 0 0 0 0 "sharpe2astra"
B 0.0000001 0 0 0 0 "sharpe2astra"
C 0.0000002 0 0 0 0 "sharpe2astra"
D 0.0000002 0 0 0 0 "sharpe2astra"
E 0.0000003 0 0 0 0 "sharpe2astra"
F 0.0000003 0 0 0 0 "sharpe2astra"

[GATES]
TOP OR GG1 C GG2 "sharpe2astra"
GG2 AND D E F "sharpe2astra"
GG1 AND A B D "sharpe2astra"

$END
```

I risultati restituiti da Astra sono quelli di Fig. 3.5; i risultati di Sharpe sono compresi tra i limiti superiori e il limite inferiore di Astra per lo stesso valore di t; in questo esempio i MCS non sono indipendenti (vedi paragrafo 1.3.2) perciò il limite di *Esary-Proshan* non equivale ai risultati esatti di Sharpe, anche se si avvicina molto dato che il cutset che ha maggiore influenza è **c**, mentre gli altri hanno un peso scarso. Lo stesso discorso vale per il limite inferiore (*lower bound*).

3.5.2 Esempio 2

Esaminiamo ora l'albero di Fig. 3.2 che è stato usato in questo capitolo come esempio di traduzione da Sharpe a Astra; il contenuto dei relativi file si trova nel paragrafo 3.2.

List of system data as function of time-points

	Time (hours)	Unavailability	Unav. (Proschan)	Unav. (lower bound)
1	0.00000000	0.000000E+00	0.000000E+00	0.000000E+00
2	25000.00000000	4.987830E-03	4.987829E-03	4.987829E-03
3	50000.00000000	9.952619E-03	9.952595E-03	9.952595E-03
4	75000.00000000	1.489626E-02	1.489614E-02	1.489614E-02
5	100000.00000000	1.982058E-02	1.982020E-02	1.982020E-02
6	125000.00000000	2.472734E-02	2.472642E-02	2.472642E-02
7	150000.00000000	2.961824E-02	2.961636E-02	2.961634E-02
8	175000.00000000	3.449491E-02	3.449146E-02	3.449143E-02
9	200000.00000000	3.935891E-02	3.935309E-02	3.935304E-02
10	225000.00000000	4.421177E-02	4.420256E-02	4.420247E-02
11	250000.00000000	4.905494E-02	4.904106E-02	4.904091E-02
12	275000.00000000	5.388981E-02	5.386973E-02	5.386950E-02
13	300000.00000000	5.871773E-02	5.868963E-02	5.868927E-02
14	325000.00000000	6.354000E-02	6.350173E-02	6.350121E-02
15	350000.00000000	6.835784E-02	6.830697E-02	6.830623E-02
16	375000.00000000	7.317245E-02	7.310619E-02	7.310516E-02
17	400000.00000000	7.798497E-02	7.790019E-02	7.789879E-02
18	425000.00000000	8.279649E-02	8.268969E-02	8.268782E-02
19	450000.00000000	8.760806E-02	8.747537E-02	8.747292E-02
20	475000.00000000	9.242068E-02	9.225785E-02	9.225469E-02
21	500000.00000000	9.723530E-02	9.703768E-02	9.703367E-02

Figura 3.5: I risultati di Astra per l'albero di Fig. 1.1

Vediamo quali sono i risultati restituiti dai due strumenti di analisi richiedendo sempre la probabilità di guasto del sistema per t che varia tra 0 e 500000 con un passo pari a 25000; cominciamo da Sharpe:

```
      system duplica
      t          F(t)

 0.0000 e+00   0.0000 e+00
 2.5000 e+04   1.6666 e-04
 5.0000 e+04   6.5836 e-04
 7.5000 e+04   1.4630 e-03
 1.0000 e+05   2.5688 e-03
 1.2500 e+05   3.9642 e-03
 1.5000 e+05   5.6383 e-03
 1.7500 e+05   7.5801 e-03
 2.0000 e+05   9.7791 e-03
 2.2500 e+05   1.2225 e-02
 2.5000 e+05   1.4909 e-02
 2.7500 e+05   1.7819 e-02
 3.0000 e+05   2.0948 e-02
 3.2500 e+05   2.4287 e-02
```

```
3.5000 e+05  2.7825 e-02
3.7500 e+05  3.1555 e-02
4.0000 e+05  3.5469 e-02
4.2500 e+05  3.9558 e-02
4.5000 e+05  4.3815 e-02
4.7500 e+05  4.8232 e-02
5.0000 e+05  5.2802 e-02
```

I risultati di Astra sono in Fig. 3.6; anche in questo casi i risultati esatti di Sharpe sono compresi entro i limiti di Astra, anche se questi hanno valori che si discostano maggiormente da quelli esatti, in quanto i MCS sono molti (27, tutti di ordine 2) e indipendenti tra loro.

List of system data as function of time-points

	Time (hours)	Unavailability	Unav. (Proschan)	Unav. (lower bound)
1	0.00000000	0.000000E+00	0.000000E+00	0.000000E+00
2	25000.00000000	1.683287E-04	1.683151E-04	1.662189E-04
3	50000.00000000	6.716348E-04	6.714177E-04	6.547522E-04
4	75000.00000000	1.507409E-03	1.506316E-03	1.450419E-03
5	100000.00000000	2.673157E-03	2.669719E-03	2.538048E-03
6	125000.00000000	4.166398E-03	4.158051E-03	3.902476E-03
7	150000.00000000	5.984667E-03	5.967454E-03	5.528555E-03
8	175000.00000000	8.125513E-03	8.093803E-03	7.401153E-03
9	200000.00000000	1.058650E-02	1.053271E-02	9.505158E-03
10	225000.00000000	1.336520E-02	1.327955E-02	1.182548E-02
11	250000.00000000	1.645921E-02	1.632944E-02	1.434705E-02
12	275000.00000000	1.986614E-02	1.967727E-02	1.705482E-02
13	300000.00000000	2.358360E-02	2.331774E-02	1.993380E-02
14	325000.00000000	2.760922E-02	2.724531E-02	2.296899E-02
15	350000.00000000	3.194066E-02	3.145426E-02	2.614545E-02
16	375000.00000000	3.657557E-02	3.593867E-02	2.944827E-02
17	400000.00000000	4.151164E-02	4.069248E-02	3.286258E-02
18	425000.00000000	4.674654E-02	4.570941E-02	3.637355E-02
19	450000.00000000	5.227798E-02	5.098309E-02	3.996640E-02
20	475000.00000000	5.810368E-02	5.650696E-02	4.362636E-02
21	500000.00000000	6.422137E-02	6.227436E-02	4.733876E-02

Figura 3.6: I risultati di Astra per l'albero di Fig. 3.2

3.5.3 Esempio 3

L'albero considerato è quello di Fig. 3.7; questo albero ha la particolarità che un evento interno ($G1$) appare come input di due porte logiche diverse; inoltre, per come è strutturato l'albero, risultano due MCS, $B1$ e $B2$, che riguardano due soli eventi.

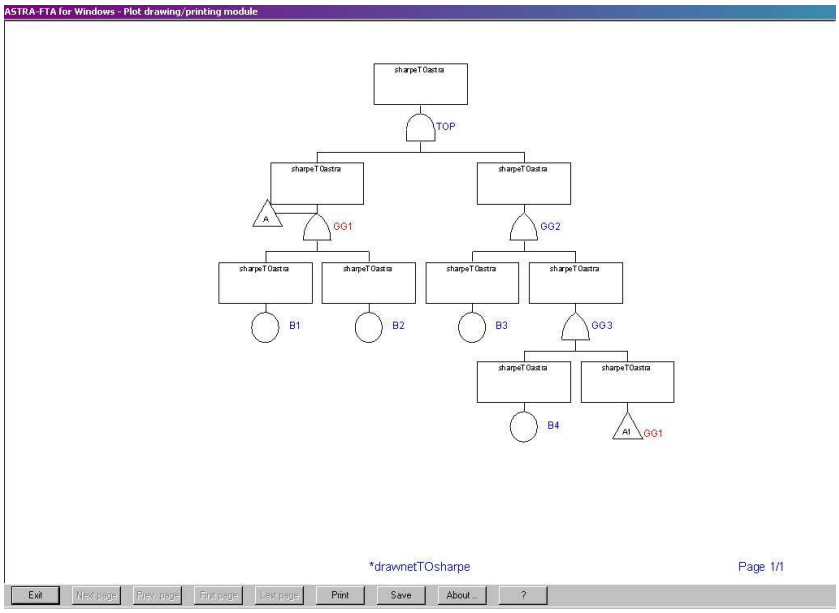


Figura 3.7: Albero di esempio con evento interno condiviso

Infatti il sistema si guasta se si verifica $B1$ o $B2$, mentre gli altri eventi di base non sono determinanti in questo senso e quindi non appaiono nei MCS.

Questo è il listato Sharpe:

```
*esempio 3

ftree test_02

repeat B1 exp(v1)
repeat B2 exp(v2)
repeat B3 exp(v3)
repeat B4 exp(v4)

or G1 B1 B2
or G3 B4 G1
or G2 B3 G3
and TOP G1 G2

end

bind
```

```
v1 0.000001
v2 0.000002
v3 0.000003
v4 0.000004

end

format 8

eval(test_02) 0 525000 25000
end
```

Segue quello di Astra:

```
*esempio 3

[EVENTS]
B1 0.000001 0 0 0 0 "sharpe2astra"
B2 0.000002 0 0 0 0 "sharpe2astra"
B3 0.000003 0 0 0 0 "sharpe2astra"
B4 0.000004 0 0 0 0 "sharpe2astra"

[GATES]
TOP AND GG1 GG2 "sharpe2astra"
GG1 OR B1 B2 "sharpe2astra"
GG3 OR B4 GG1 "sharpe2astra"
GG2 OR B3 GG3 "sharpe2astra"

$END
```

Essendo i MCS indipendenti il limite superiore di *Esary-Proshan* corrisponde ai valori esatti di Sharpe e al limite inferiore, dato che i MCS sono solo due.

Seguono i risultati restituiti da Sharpe, quelli di Astra sono in Fig. 3.8.

```
      system test_02
        t             F(t)

 0.00000000 e+00  0.00000000 e+00
 2.50000000 e+04  7.22565137 e-02
 5.00000000 e+04  1.39292024 e-01
 7.50000000 e+04  2.01483781 e-01
```

```
1.00000000 e+05  2.59181779 e-01
1.25000000 e+05  3.12710721 e-01
1.50000000 e+05  3.62371848 e-01
1.75000000 e+05  4.08444636 e-01
2.00000000 e+05  4.51188364 e-01
2.25000000 e+05  4.90843579 e-01
2.50000000 e+05  5.27633447 e-01
2.75000000 e+05  5.61765008 e-01
3.00000000 e+05  5.93430340 e-01
3.25000000 e+05  6.22807646 e-01
3.50000000 e+05  6.50062251 e-01
3.75000000 e+05  6.75347533 e-01
4.00000000 e+05  6.98805788 e-01
4.25000000 e+05  7.20569032 e-01
4.50000000 e+05  7.40759739 e-01
4.75000000 e+05  7.59491537 e-01
5.00000000 e+05  7.76869840 e-01
```

List of system data as function of time-points

	Time (hours)	Unavailability	Unav. (Proschan)	Unav. (lower bound)
1	0.00000000	0.000000E+00	0.000000E+00	0.000000E+00
2	25000.00000000	7.346066E-02	7.225651E-02	7.225651E-02
3	50000.00000000	1.439332E-01	1.392920E-01	1.392920E-01
4	75000.00000000	2.115485E-01	2.014838E-01	2.014838E-01
5	100000.00000000	2.764318E-01	2.591818E-01	2.591818E-01
6	125000.00000000	3.387023E-01	3.127107E-01	3.127107E-01
7	150000.00000000	3.984738E-01	3.623718E-01	3.623718E-01
8	175000.00000000	4.558549E-01	4.084446E-01	4.084446E-01
9	200000.00000000	5.109492E-01	4.511884E-01	4.511884E-01
10	225000.00000000	5.638556E-01	4.908436E-01	4.908436E-01
11	250000.00000000	6.146686E-01	5.276334E-01	5.276334E-01
12	275000.00000000	6.634781E-01	5.617650E-01	5.617650E-01
13	300000.00000000	7.103701E-01	5.934303E-01	5.934303E-01
14	325000.00000000	7.554269E-01	6.228076E-01	6.228076E-01
15	350000.00000000	7.987266E-01	6.500622E-01	6.500622E-01
16	375000.00000000	8.403442E-01	6.753475E-01	6.753475E-01
17	400000.00000000	8.803510E-01	6.988058E-01	6.988058E-01
18	425000.00000000	9.188153E-01	7.205690E-01	7.205690E-01
19	450000.00000000	9.558022E-01	7.407597E-01	7.407597E-01
20	475000.00000000	9.913739E-01	7.594915E-01	7.594915E-01
21	500000.00000000	1.000000E+00	7.768698E-01	7.512799E-01

Figura 3.8: I risultati di Astra per l'albero di Fig. 3.7

Capitolo 4

Traduttore da DrawNet a Sharpe

4.1 Introduzione

Ora è possibile analizzare lo stesso albero sia con Sharpe sia con Astra grazie all'apposito traduttore; anche se Astra dispone di un'articolata interfaccia grafica, la scrittura del file contenente la struttura dell'albero da esaminare va comunque fatta "a mano" attraverso qualche editor di testo.

Questo modo di creare l'albero può risultare scomodo, visto che la sua rappresentazione testuale lo rende poco intuitivo e comprensibile a prima vista; più facile ed immediato sarebbe disegnare l'albero con qualche strumento grafico, salvarlo in un file e convertirlo nel corrispondente formalismo di Sharpe e Astra.

Per poter fare questo si è scelto uno strumento grafico che fosse abbastanza adatto alla logica degli alberi dei guasti, flessibile ad eventuali evoluzioni, quali il PFT, e che salvasse l'albero in un formato di file accessibile direttamente.

4.2 DrawNet come strumento grafico

DrawNet [7] [8] è uno strumento grafico implementato in Java per disegnare grafi che possono essere di ogni natura; infatti è possibile indicare a *DrawNet* la tipologia di grafo che si intende poter disegnare creando in linguaggio XML un formalismo che descrive il tipo di grafo e che viene caricato all'avvio del programma dopo essere stato tradotto in un formato interno a DrawNet.

Fondamentalmente si devono indicare nel formalismo quali sono i tipi di nodi e i tipi di archi che il grafo descritto può contenere e le loro caratteris-

tiche; ogni tipo di elemento del grafo, nodo o arco, deve avere specificato un nome e degli eventuali attributi.

DrawNet è stato inizialmente utilizzato durante il lavoro di tesi per disegnare alberi dei guasti anche in forma parametrica; il formalismo in XML utilizzato è il seguente:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?> <!DOCTYPE
formalism SYSTEM "formalism.dtd"> <formalism parent="" name="PFT">
    <propertyType name="Title" default=""/>

    <nodeType parent="" name="Event0">
        <propertyType name="Label" default=""/>
        <propertyType name="Parameters" default=""/>
    </nodeType>
    <nodeType parent="Event0" name="Event"/>
    <nodeType parent="Event0" name="BasicEvent">
        <propertyType name="Distribution" default="ALL EXP 1.0"/>
    </nodeType>
    <nodeType parent="Event" name="ReplicatorEvent">
        <propertyType name="PList" default=""/>
        <propertyType name="DeclaredParameters" default=""/>
        <propertyType name="PredSWN" default=""/>
    </nodeType>
    <nodeType parent="BasicEvent" name="BasicReplicatorEvent">
        <propertyType name="PList" default=""/>
        <propertyType name="DeclaredParameters" default=""/>
        <propertyType name="PredSWN" default=""/>
    </nodeType>
    <nodeType parent="" name="TopEvent">
        <propertyType name="Label" default=""/>
    </nodeType>

    <nodeType parent="" name="Gate"/>
    <nodeType parent="Gate" name="And"/>
    <nodeType parent="Gate" name="Or"/>
    <nodeType parent="" name="G2of3"/>
    <nodeType parent="Gate" name="KofN">
        <propertyType name="K" default="2"/>
        <propertyType name="N" default="3"/>
    </nodeType>
```

```
    <nodeType parent="" name="ColorClass">
        <propertyType name="SubClassList" default=""/>
        <propertyType name="Ordered" default="false"/>
    </nodeType>

    <nodeType parent="" name="ColorSubClass">
        <propertyType name="ElementList" default=""/>
    </nodeType>

    <nodeType parent="" name="ParameterType">
        <propertyType name="ParameterName" default=""/>
        <propertyType name="ParameterColor" default=""/>
    </nodeType>


    <edgeType parent="" name="Arc">
        <constraint fromType="Gate" fromCardinality="1"
         toType="Event" toCardinality="1"/>
        <constraint fromType="G2of3" fromCardinality="1"
         toType="Event" toCardinality="1"/>
        <constraint fromType="Gate" fromCardinality="1"
         toType="TopEvent" toCardinality="1"/>
        <constraint fromType="G2of3" fromCardinality="1"
         toType="TopEvent" toCardinality="1"/>
        <constraint fromType="Event0" fromCardinality=""
         toType="Gate" toCardinality=""/>
        <constraint fromType="Event0" fromCardinality=""
         toType="G2of3" toCardinality="3"/>
    </edgeType>
</formalism>
```

Il formalismo definisce nell'ordine questi tipi di nodo:

- **Event**: evento interno (output di una porta logica);
- **Basic Event**: evento di base (componente fisico del sistema, nodo foglia dell'albero);
- **Replicator Event**: evento interno replicato (per PFT);

- **Basic Replicator Event**: evento di base replicato (per PFT); indexbasic replicator event
- **Top Event**: evento radice dell'albero;
- **And**: porta logica di tipo and;
- **Or**: porta logica di tipo or; indexor
- **G2of3**: porta logica di tipo *K out of N* con $k = 2$ e $n = 3$ (2/3);
- **KofN**: porta logica di tipo *K out of N* con k ed n variabili che vengono indicati dall'utente in due specifici campi.

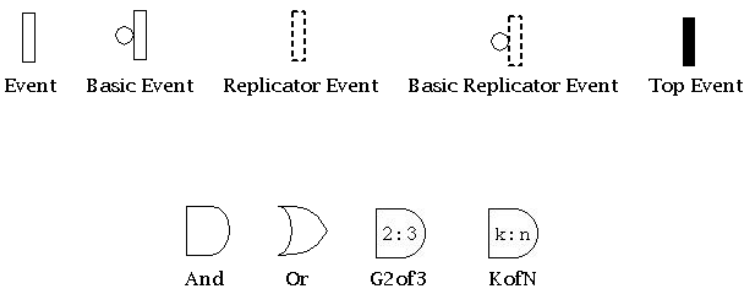


Figura 4.1: I simboli di DrawNet per FT e PFT

La Fig. 4.1 mostra le icone usate in DrawNet per distinguere i vari nodi; un esempio di come possono essere collegate tra loro è la Fig. 4.2, l'albero dei guasti parametrico el sistema **Multiproc**.

Tutti gli eventi, ad eccezione del Top Event, hanno i seguenti campi che descrivono le loro carratteristiche (Fig. 4.3 e 4.4):

- **name**: è il nome del nodo; non deve essere condiviso con altri nodi e viene assegnato automaticamente da DrawNet a meno che l'utente non lo specifichi direttamente;
- **Label**: è l'etichetta del nodo, cioè un nome che non identifica univocamente il nodo come il campo precedente, ma che ne dà una descrizione;
- **Parameters**: contiene l'elenco dei parametri se l'evento fa parte di un sottoalbero replicato.

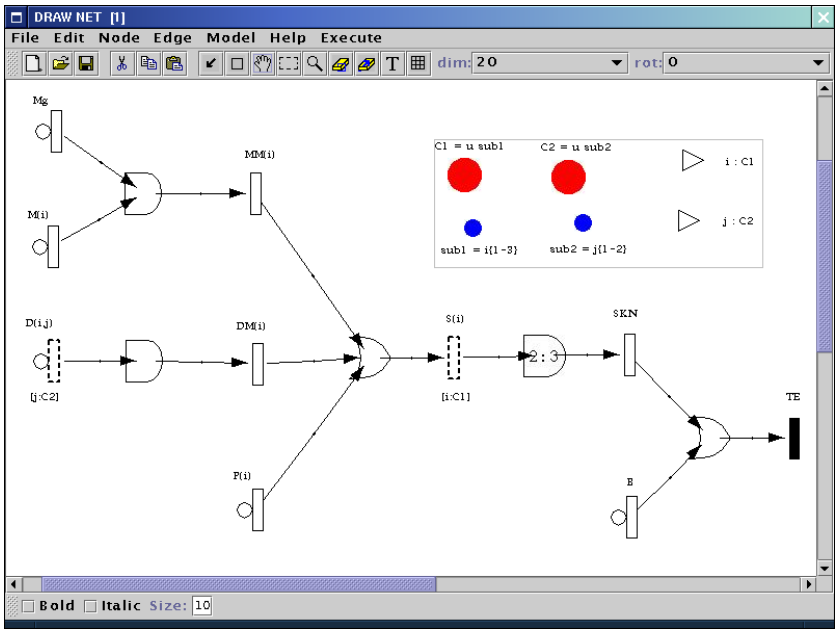


Figura 4.2: Multiproc (PFT)

Gli eventi di base, replicati o meno, hanno un campo Distribution dove si indica il tipo di distribuzione di probabilità e i relativi parametri.

Nel caso della distribuzione esponenziale, quella principalmente utilizzata, si usa la notazione ALL EXP λ; questa distribuzione viene impostata come quella di default e con $\lambda = 1.0$ se non indicato diversamente dall'utente.

Gli eventi replicati, interni o di base, contengono inoltre il campo **Declared Parameters** che contiene i parametri secondo i quali viene replicato l'evento e la parte di albero sottostante (Fig. 4.5 e 4.6).

Per quanto riguarda gli archi di connessione tra i nodi, il formalismo ne contiene un solo tipo, di nome **Arc**; comunque in DrawNet qualunque tipo di arco ha di default, anche se non indicati nel formalismo, due campi, **From** e **To**, che indicano rispettivamente il nome dei nodi di partenza e di arrivo dell'arco di collegamento e che vengono assegnati automaticamente da DrawNet nel momento in cui l'arco viene disegnato.

Nel formalismo quando si specifica un tipo di arco si devono scrivere delle regole che stabiliscono quali tipi di nodi un arco può collegare, con quale verso e in quale modo.

In questo caso un arco può collegare:

- più eventi in input alla stessa porta logica (3 input se la porta è di tipo 2/3);

Property Page

Property	Value
type	BasicEvent
name	BasicEvent7
visibility	false
Distribution	ALL EXP 0.00003
Label	M
Parameters	i

Figura 4.3: Campi di DrawNet per un evento di base

Property Page

Property	Value
type	Event
name	Event17
visibility	false
Label	SKN
Parameters	

Figura 4.4: Campi di DrawNet per un evento interno

Property Page

Property	Value
type	BasicReplicatorEvent
name	BasicReplicatorEvent6
visibility	false
PredSWN	
DeclaredParameters	j
Distribution	ALL EXP 0.08
Label	D
PList	
Parameters	i,j

Figura 4.5: Campi di DrawNet per un evento di base replicato

Property Page

Property	Value
type	ReplicatorEvent
name	ReplicatorEvent15
visibility	false
PredSWN	
DeclaredParameters	i
Label	S
PList	
Parameters	i

Figura 4.6: Campi di DrawNet per un evento interno replicato

- una sola porta logica ad un solo evento interno o al *Top Event*.

Quindi una porta può avere diversi input e deve avere un unico output; il verso dell'arco nel primo caso va dall'input alla porta, nel secondo dalla porta all'evento interno o al *Top Event*.

Una volta scritto in XML il formalismo, questo va tradotto in un formato interno di DrawNet tramite un apposito comando; a questo punto, grazie ad una interfaccia grafica di nome *TableConfig* (anche questa realizzata in Java) si può associare ad ogni tipo di nodo un'icona che lo rappresenterà a livello grafico in DrawNet (Fig. 4.7).

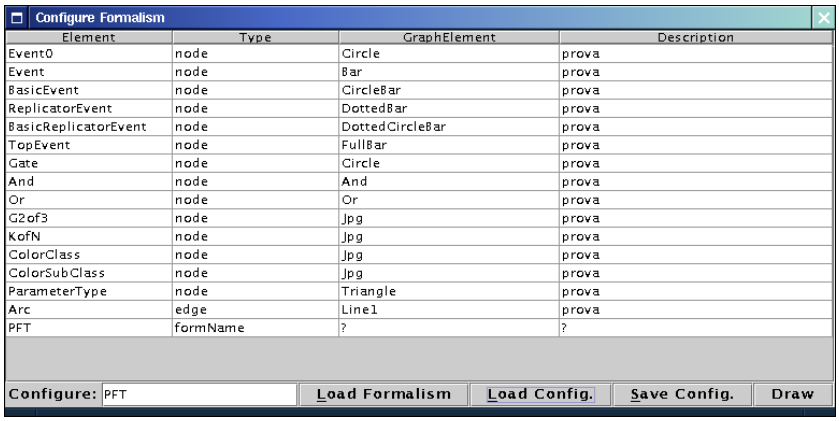


Figura 4.7: TableConfig

TableConfig ne mette a disposizione alcune, ma se ne possono creare di proprie in formato JPG o GIF con lo stesso nome del tipo di nodo a cui sono destinate.

Finalmente DrawNet è pronto per disegnare un certo tipo di grafo (albero dei guasti) con una certa rappresentazione grafica.

4.3 Rappresentazione in DrawNet di PFT

Per indicare un evento di base replicato o un sottoalbero replicato si usano rispettivamente i nodi di tipo *Basic Replicator Event* e *Replicator Event*.

Nel primo caso si deve specificare sia nel campo **Declared Parameters** sia nel campo **Parameters** il parametro rispetto al quale avviene la duplicazione dell'evento di base; in pratica il parametro indica quali sono gli indici di ciascuna copia (Fig 4.5).

Nel caso di eventi interni replicati si intende la duplicazione dell'intero sottoalbero; in questo caso si deve specificare nel *Replicator Event* il parametro sia nel campo **Declared Parameters** sia nel campo **Parameters**, dopodiché lo stesso parametro va indicato nel campo **Parameters** per ogni evento sottostante al *Replicator Event*.

In pratica, il campo **Parameters** contiene il parametro per cui avviene la duplicazione per tutti gli eventi del sottoalbero duplicato, indicato originariamente nel campo **Declared Parameters** del *Replicator Event* alla radice del sottoalbero.

Un sottoalbero duplicato potrebbe a sua volta contenere un ulteriore sottoalbero duplicato con radice in un *Replicator Event*; in questa situazione ci sono due possibilità:

- quest'ultimo sottoalbero viene replicato un certo numero di volte per ogni copia del sottoalbero a livello superiore; in questo caso il Replicator Event più interno avrà dichiarato nel campo Declared Parameters il proprio parametro, mentre nel campo Parameters avrà il parametro superiore e il proprio. Gli eventi sottostanti al Replicator Event più interni dovranno avere anch'essi nel campo Parameters entrambi i parametri.

  E' il caso del sistema *Multiproc* di Fig. 4.2, costituito da tre sottosistemi S_1, S_2, S_3, ognuno dei quali è composta da un processore $P(i)$, una memoria $M(i)$ e da due dischi fissi $D(i, j)$; c'è poi una memoria comune Mg e il bus B che connette i tre sottosistemi.

- Si desidera che le copie del sottoalbero replicato più interno siano le stesse per tutte le copie del sottoalbero superiore; in questo caso i campi Declared Parameters e Parameters del Replicator Event più interno conterrano soltanto il proprio parametro e soltanto questo sarà indicato negli eventi sottostanti.

  Un esempio è il sistema **PLC** di Fig. 4.8 in cui i sottosistemi $INPUT(z)$ sono gli stessi per ogni SUB(y).

Gli stessi principi si applicano nel caso si tratti di Basic Replicator Event anziché di Replicator Event; l'unica differenza consiste nell'assenza di un sottoalbero sottostante e quindi non c'è la propagazione del parametro o dei parametri agli eventi più in profondità.

In ogni caso il parametro di duplicazione serve per indicare il numero delle copie e i relativi indici; ad esempio se l'evento replicato RE ha come parametro i a cui corrispondono gli indici $\{a, b, c, d\}$, il numero delle copie sarà 4 e queste saranno RE_a, RE_b, RE_c, RE_d.

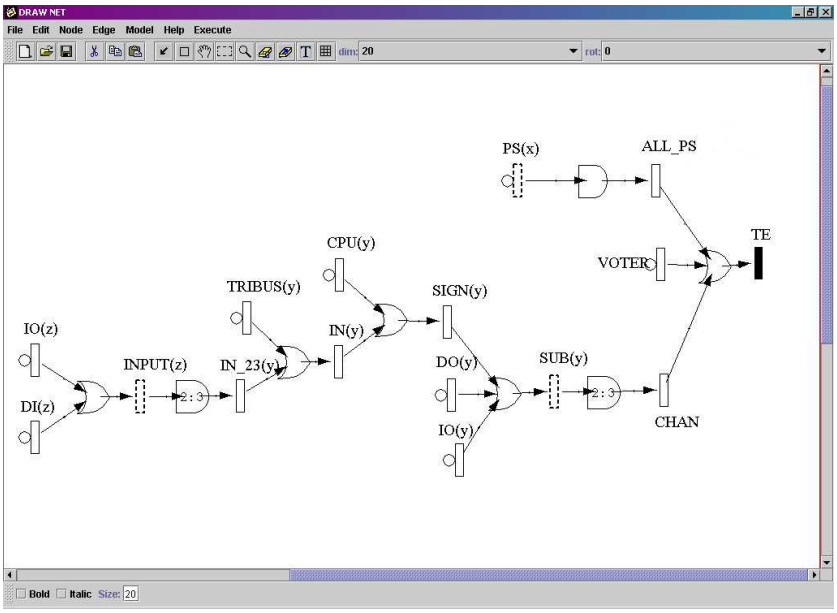


Figura 4.8: Albero PFT del sistema PLC

L'indicazione della natura dei parametri in DrawNet si realizza utilizzando tre tipi di nodi indicati nel formalismo di sopra; essi sono:

- **Parameter Type**: serve per indicare il nome del parametro; per l'insieme dei suoi valori si fa riferimento ad una classe di colore indicata nell'apposito campo; in DrawNet è indicato con un triangolo;
- **Color Class**: una classe di colore serve per indicare i possibili valori del parametro; questi possono essere contenuti in una o più liste il cui nome viene indicato nel campo SubClassList; viene indicato con un cerchio rosso;
- **Color SubClass**: è una lista di valori i cui elementi può essere indicato in modo esplicito, ad esempio $\{1, 2, 3, 4\}$, o in modo implicito, $\{1 - 4\}$; si indica con un cerchio blu più piccolo.

4.4 La traduzione da DrawNet a Sharpe

Attraverso il comando **Solve** di DrawNet, si genera un file in formato XML che contiene la struttura dell'albero secondo il formalismo e secondo il contenuto dei campi dei vari nodi e archi disegnati dall'utente.

Questo file XML viene preso in input dal traduttore che ho implementato in linguaggio C e che genera la rappresentazione dello stesso albero nel formalismo di Sharpe.

Nel caso il file XML riguardi un albero parametrizzato il traduttore esegue anche un'operazione di *unfolding*, cioè di trasformazione da PFT a FT, in quanto Sharpe non prevede la parametrizzazione.

In questo modo è possibile disegnare un albero dei guasti, eventualmente parametrizzato, con uno strumento grafico, DrawNet, e analizzarlo con Sharpe, anziché scriverlo "a mano" secondo il formalismo di Sharpe. Successivamente, attraverso il traduttore da Sharpe ad Astra, è possibile passare lo stesso albero a quest'ultimo strumento di analisi.

La traduttore da DrawNet a Sharpe avviene nelle seguenti fasi:

1. Parsing
2. Analisi dei parametri
3. Linking
4. Individuazione delle costanti
5. Trasformazione da PFT a FT (Unfolding)
6. Ordinamento di eventi e porte logiche
7. generazione del file per Sharpe.

Durante la spiegazione delle fasi del traduttore si farà riferimento come esempio al PFT del sistema **Multiproc** di Fig. 4.2 il cui listato XML è il seguente:

```
<?xml  version='1.0' encoding='UTF-8' standalone='no'?>
<PFT name='Multiproc' visibility='false' Title=''>

   <ColorClass name='C1' visibility='false' SubClassList='sub1'
    Ordered='false'/>

   <ColorClass name='C2' visibility='false' SubClassList='sub2'
    Ordered='false'/>

   <ColorSubClass name='sub1' visibility='false'
    ElementList='i{1-3}'/>
```

```
<ColorSubClass name='sub2' visibility='false'
 ElementList='j{1-2}'/>

<ParameterType name='ParameterType4' visibility='false'
 ParameterColor='C1' ParameterName='i'/>

<ParameterType name='ParameterType5' visibility='false'
 ParameterColor='C2' ParameterName='j'/>

<BasicReplicatorEvent name='BasicReplicatorEvent6' visibility='false'
 PredSWN='' DeclaredParameters='j' Distribution='ALL EXP 0.000008'
 Label='D' PList='' Parameters='i,j'/>

<BasicEvent name='BasicEvent7' visibility='false'
 Distribution='ALL EXP 0.000003' Label='M' Parameters='i'/>

<BasicEvent name='BasicEvent8' visibility='false'
 Distribution='ALL EXP 0.000003' Label='Mg' Parameters=''/>

<And name='And9' visibility='false'/>

<And name='And10' visibility='false'/>

<Event name='Event11' visibility='false' Label='MM' Parameters='i'/>

<Event name='Event12' visibility='false' Label='DM' Parameters='i'/>

<BasicEvent name='BasicEvent13' visibility='false'
 Distribution='ALL EXP 0.000005' Label='P' Parameters='i'/>

<Or name='Or14' visibility='false'/>

<ReplicatorEvent name='ReplicatorEvent15' visibility='false'
 PredSWN='' DeclaredParameters='i' Label='S' PList='' Parameters='i'/>

<G2of3 name='G2of316' visibility='false'/>

<Event name='Event17' visibility='false' Label='SKN' Parameters=''/>

<BasicEvent name='BasicEvent18' visibility='false'
 Distribution='ALL EXP 0.000002' Label='B' Parameters=''/>
```

```
   <Or name='Or19' visibility='false'/>

   <TopEvent name='TopEvent20' visibility='false' Label='TE'/>

   <Arc name='Arc1' visibility='false' from='BasicEvent8' to='And9'/>

   <Arc name='Arc2' visibility='false' from='BasicEvent7' to='And9'/>

   <Arc name='Arc3' visibility='false' from='BasicReplicatorEvent6'
    to='And10'/>

   <Arc name='Arc4' visibility='false' from='And10' to='Event12'/>

   <Arc name='Arc5' visibility='false' from='And9' to='Event11'/>

   <Arc name='Arc6' visibility='false' from='BasicEvent13' to='Or14'/>

   <Arc name='Arc7' visibility='false' from='Event12' to='Or14'/>

   <Arc name='Arc8' visibility='false' from='Event11' to='Or14'/>

   <Arc name='Arc9' visibility='false' from='Or14'
    to='ReplicatorEvent15'/>

   <Arc name='Arc10' visibility='false' from='ReplicatorEvent15'
    to='G2of316'/>

   <Arc name='Arc11' visibility='false' from='G2of316' to='Event17'/>

   <Arc name='Arc12' visibility='false' from='Event17' to='Or19'/>

   <Arc name='Arc13' visibility='false' from='BasicEvent18' to='Or19'/>

   <Arc name='Arc14' visibility='false' from='Or19' to='TopEvent20'/>
</PFT>
```

4.4.1 Parser

Nel file XML il tag all'inizio di ogni riga indica il tipo di nodo o arco che quella riga descrive; seguono degli attributi il cui contenuto corrisponde ai

campi descrittivi di ogni nodo o arco. Secondo il tag che incontra, il parser lancia una procedura opportuna che carica i dati contenuti su quella riga nell'apposita struttura dati; gli eventi, replicati o meno, vengono memorizzati in due vettori di strutture, uno per gli eventi di base ed uno per quelli interni, i cui campi principali sono:

- **nome**: nome non significativo dell'evento di base, assegnato da DrawNet; identifica univocamente il nodo;
- **label**: nome, dato dall'utente, descrittivo dell'evento di base;
- **esp**: contiene i parametri della distribuzione di probabilità; se questa è esponenziale contiene il valore di λ;
- **param**: vettore che contiene i nomi dei parametri per cui dovrà essere duplicato l'evento di base; se l'evento di base non è un Basic Replicator Event, i parametri saranno quelli di un Replicator Event a livello superiore, altrimenti saranno gli stessi contenuti nel vettore **local**.
- **local**: vettore che contiene i nomi dei parametri per cui dovrà essere duplicato l'evento di base; questo vettore sarà riempito se si tratta di un Basic Replicator Event, altrimenti resterà vuoto.
- **num_param**: numero di elementi del vettore **param**;
- **num_local**: numero di elementi del vettore **local**;
- **copie**: indica il numero di copie da generare dell'evento di base; il suo valore viene assegnato successivamente in base all'analisi dei parametri;
- **origine**: indica l'indice nel vettore dell'evento da cui è stato copiato l'evento di base corrente; riguarda le copie e viene inizializzato nella fase di duplicazione.

Le porte logiche sono memorizzate in un vettore di strutture, i cui campi sono:

- **nome**: nome della porta, identificativo assegnato da DrawNet;
- **tipo**: tipo di porta; dipende dal tag;
- **arg**: contiene i valori di k ed n per le porte *K out of N*;
- **figli**: vettore di puntatori agli eventi di input della porta;

- **tipo_figli**: stringa in cui ogni carattere indica la natura di uno degli eventi di input (di base o interno); la stringa ha lunghezza pari al numero di input della porta;
- **num_figli**: numero di eventi di input della porta;
- **evento**: puntatore all'evento interno output della porta;

Gli archi sono invece inseriti in un vettore in cui ogni elemento è una struttura composta da **From** e **To**; questi contengono rispettivamente il nome di un evento e il nome di una porta logica se si tratta di un arco di input, altrimenti, per gli archi di output, **From** contiene il nome di una porta logica e **To** il nome di un evento interno.

Dato che i nomi assegnati da DrawNet individuano in modo univoco i nodi o gli archi dell'albero e dato che tra due nodi ci può essere un solo arco (si tratta di un albero), per distinguere un arco da un altro sono sufficienti i campi **From** e **To**.

La struttura per la memorizzazione dei parametri contiene il nome del parametro e il nome del suo colore, quella per i colori è composta dal nome del colore e dal nome della lista (assumento che ad ogni colore corrisponda una sola lista di valori), mentre quella per la lista di valori ha come campi:

- **nome**: nome della lista;
- **indici**: vettore contenente i valori della lista (numeri o stringhe); se la lista è stata dichiarata in DrawNet in modo implicito il parser determinerà ogni suo valore;
- **card**: numero di elementi della lista.

Ci sono poi delle variabili che indicano il numero di eventi di base, eventi interni, porte logiche, parametri, ecc.

Il parser ha quindi il compito di individuare nel file XML i vari tipi di nodo o arco, caricarne gli attributi e memorizzarli nelle opportune strutture dati illustrate sopra.

L'ordine delle dichiarazioni nel file XML generato da DrawNet rispetta l'ordine con cui le varie parti dell'albero sono state disegnate dall'utente, quindi può essere qualunque: porte logiche, eventi, archi, parametri, ecc. nel codice XML possono comparire a qualunque riga e possono essere mescolati tra loro.

Per questo si è reso necessario memorizzare tutte le parti dell'albero per poi collegarle tra loro, anziché determinare la struttura dell'albero direttamente dalle dichiarazioni come fatto nel traduttore da Sharpe a Astra.

DrawNet quindi non fornisce la definizione diretta di eventi e porte logiche, ma essendo uno strumento grafico adatto a disegnare ogni tipo di grafo, descrive nel file XML degli oggetti grafici che compongono l'albero, piuttosto che degli oggetti logici.

La Fig. 4.9 mostra il diagramma a stati del parser per i principali tipi di elemento dell'albero.

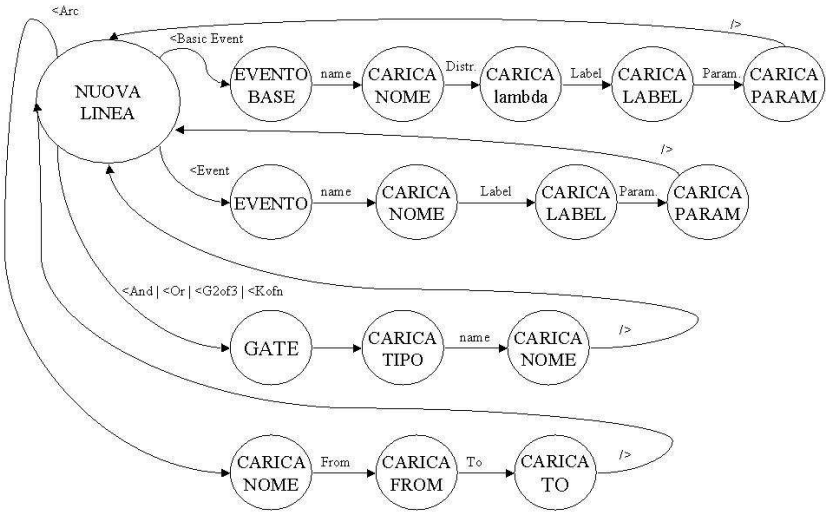


Figura 4.9: Diagramma a stati del parser

Caso di esempio: Multiproc

Dopo aver caricato il file **Multiproc.xml** che contiene il listato XML indicato sopra, risultano come eventi di base $D(i,j), M(i), Mg, P(i), B$ (tutti con distribuzione esponenziale) e come eventi interni $MM(i), DM(i), S(i), SKN$, oltre al *Top Event* indicato con TE; risultano poi due porte logiche *And*, due Or e una porta 2 out of 3.

I parametri sono i e j, i colori sono $C1$ e $C2$, mentre $sub1$ e $sub2$ sono le liste di valori; gli archi sono 14 e il nome dell'albero è *Multiproc* e il relativo campo **Title** per il commento è vuoto.

4.4.2 Analisi dei parametri

L'analisi dei parametri avviene in due fasi; nella prima, si deve associare ad ogni parametro un insieme di valori; si esaminano tutti i parametri e per ognuno si risale al colore associato facendo una ricerca nel vettore dei colori di quello con il nome indicato nella descrizione del parametro; una volta trovato il colore, da questo si risale a sua volta alla lista di valori corrispondente, facendo sempre una ricerca, questa volta nel vettore delle liste, in base al nome di lista indicato dal colore; quindi si inizializza un puntatore che collega direttamente il parametro alla lista di valori.

Un colore può anche riguardare più parametri ed una lista può essere associata a più di un colore; quindi due parametri, ad esempio, potrebbero avere lo stesso insieme di valori.

Nella seconda fase, per ogni evento di base o interno di tipo Replicator, si determina il numero di copie da generare successivamente in base ai suoi parametri, in particolare secondo il numero di valori associati a ciascuno di questi indicati nel campo **local**; infatti il numero di copie dell'evento sarà uguale al prodotto delle cardinalità degli insiemi di valori legati ad ogni parametro dell'evento indicato nel campo **local**.

Questo valore viene calcolato per ogni evento con parametri nel campo local e memorizzato nell'apposito campo della struttura che descrive l'evento di tipo Replicator.

Caso di esempio

Al parametro i corrisponde il colore $C1$ al quale corrisponde a sua volta la lista $sub1$ che ha come elementi $\{1, 2, 3\}$; il parametro j è legato al colore $C2$ quindi alla lista $sub2$ di elementi $\{1, 2\}$.

Gli eventi di tipo Replicator in *Multiproc* sono $D(i, j)$ (di base) e $S(i)$(interno); $D(i, j)$ è replicato localmente rispetto al parametro j, mentre il parametro è ereditato da un evento replicato a livello superiore ($S(i)$); il campo *copie* di $D(i, j)$ assume quindi il valore 2, mentre quello di $S(i)$ il valore 3.

4.4.3 Linking

In questa fase si determinano per ogni porta logica gli eventi di input, la loro natura (di base o interni) e l'evento di output della porta in base agli archi entranti e uscenti rispetto a questa.

Viene esaminato un arco per volta, cioè si determina la natura di ciò descritto nei campi **From** e To; in questo senso sono possibili tre situazioni:

- **From** evento di base **To** porta logica: la porta logica ha come input un evento di base;
- **From** evento interno **To** porta logica: la porta logica ha come input un evento interno;
- **From** porta logica **To** evento interno: l'evento interno è l'output della porta logica.

Nei primi due casi si collega la porta logica agli eventi assegnando degli elementi del vettore dei puntatori ai nodi figli, mentre nel terzo caso si inizializza il puntatore all'evento di output.

Se uno degli eventi di input è di tipo Replicator non gli verrà assegnato un solo puntatore, ma tanti quanti il numero di copie che se ne dovrà fare successivamente; in questo modo si prepara già lo spazio per le copie dell'evento e si evita di dover spostare verso destra gli eventi di input della porta successivi a quello replicato.

Una volta esaminati tutti gli archi e quindi dopo aver collegato opportunamente porte ed eventi si ha la struttura completa dell'albero. A questo punto è possibile eseguire su di esso le operazioni necessarie per la traduzione vera e propria nel formalismo di Sharpe.

Caso di esempio

La Fig. 4.10 mostra come appare al momento l'albero in memoria; si nota che la porta logica relativa all'evento interno SKN ha tre puntatori verso lo stesso evento di input, cioè $S(i)$ di tipo Replicator.

Analogamente $D(i, j)$ appare due volte come input della porta con evento di output $DM(i)$.

4.4.4 Individuazione delle costanti

Come spiegato nel Cap. 2, Sharpe consente di indicare delle costanti utilizzabili nelle dichiarazioni degli eventi di base ed in particolare nell'indicazione della loro probabilità di guasto.

Il traduttore crea a questo punto un vettore in cui vengono inseriti tutti i valori dei parametri λ per la distribuzione esponenziale degli eventi di base; questi valori saranno poi indicati nel file per Sharpe dopo la parola chiave *bind* con i nomi: $v1, v2, v3, \ldots$ e saranno utilizzati nelle dichiarazioni delle distribuzioni; se due o più eventi di base hanno lo stesso λ, questo nel vettore comparirà una sola volta e nel file per Sharpe si farà riferimento più volte alla stessa costante.

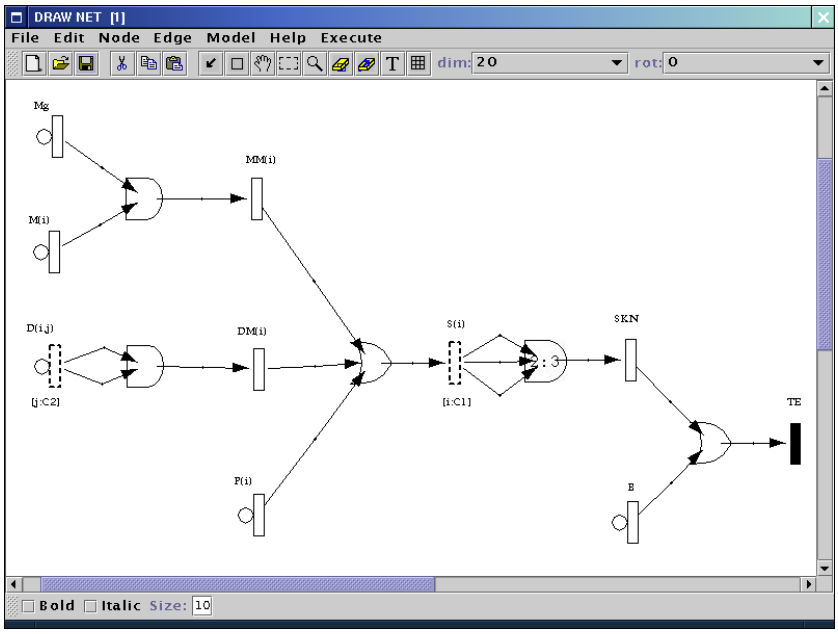


Figura 4.10: Struttura dell'albero dopo la fase di linking

Caso di esempio

Le costanti che appariranno nel file Sharpe sono:
$0.000008, 0.000003, 0.000005, 0.000002$.

4.4.5 Unfolding

Per *unfolding* si intende la trasformazione da PFT a FT, mantenendo però lo stesso significato logico dell'albero.

L'algoritmo di *unfolding* parte dalla parte bassa dell'albero per poi salire verso il *Top Event* esaminando i nodi dell'albero in un ordine di tipo bottom-up. Per ottenere questo si è associato ad ogni evento output di una porta logica un numero d'ordine che indica a quale passo di una visita in profondità è stato attraversato tale evento: 1 per il primo evento interno visitato, 2 per il secondo e così via.

Seguento questo ordine si esaminano tutti gli eventi interni; per ognuno si verifica se la porta logica associata ha come input degli eventi replicati; in caso affermativo, per ognuno di questi genera tante copie quante il numero indicato nell'apposito campo dell'evento replicato, inizializzato durante l'analisi dei parametri.

Si aggiornano ora i puntatori indirizzandoli verso le copie appena create; fare una copia di un evento replicato significa più precisamente creare un nuovo evento che eredita dall'originale l'etichetta (campo **label**), la probabilità di guasto se si tratta di un evento di base, il puntatore alla porta logica di cui esso è output se si tratta di eventi interni.

Per distinguere a questo punto una copia da un'altra si devono assegnare dei valori ai loro parametri; in pratica si fa corrispondere ad ogni parametro presente nel campo **local** un valore che viene salvato nel campo **param** al posto del parametro in questione.

Se la duplicazione dell'evento è avvenuta secondo un solo parametro, alla prima copia si assegna come indice il primo valore della lista, alla seconda il secondo e così via; se invece, la duplicazione è avvenuta secondo due o più parametri si costruisce un nuovo insieme di valori dato dal prodotto cartesiano degli insiemi di valori legati a ciascun parametro indicato nel campo local. Ciascun elemento di questo nuovo insieme viene assegnato come indice a ogni copia dell'evento generata.

Se l'evento duplicato è di base il procedimento termina, altrimenti se l'evento è interno si deve anche duplicare il sottoalbero sottostante a ciascuna copia.

In questo caso l'algoritmo scende lungo ciascun input della porta logica associata all'evento in base ai puntatori; questi sono stati ereditati dall'evento originale e quindi puntano a parti dell'albero ancora in forma parametrica.

Per ciascun evento input della porta logica si verifica se questo ha nel campo **param** dei parametri in comune con quelli del campo **local** dell'evento duplicato; in caso affermativo si stacca l'evento di input, se ne fa una copia sostituendo al parametro in comune lo stesso valore assunto nell'evento alla radice del sottoalbero e la si attacca al posto dell'originale. Se la copia è un evento interno a sua volta, si ripete ricorsivamente la stessa procedura. Se un evento sottostante non ha invece parametri in comune con la radice del sottoalbero, viene lasciato invariato.

Esaminando gli eventi replicati in ordine bottom-up, quando si duplica il sottoalbero si è certi che all'interno di questo non ci sono più altri eventi di tipo Replicator in quanto sarebbero già stati duplicati in precedenza se ce ne fossero stati.

Infatti l'ordine di valutazione bottom-up è utile proprio quando ci sono eventi replicati discendenti di altri eventi replicati. In questo caso viene duplicato prima l'evento a livello inferiore e il suo eventuale sottoalbero che non contiene eventi replicati; quindi, più tardi, si duplica il sottoalbero sottostante l'evento replicato a livello superiore contenente già una duplicazione avvenuta.

Caso di esempio

Le Fig. 4.11, 4.12, 4.13 mostrano i vari passi con cui avviene l'*unfolding* dell'albero *Multiproc*.

L'ordine con cui si visiterebbero i nodi in profondità è:
$Mg, M(i), MM(i), D(i,j), DM(i), P(i), S(i), SKN, B, TE$; gli eventi di tipo Replicator sono $D(i,j)$ e $S(i)$.

Nella Fig. 4.11 l'evento $D(i,j)$ è il primo di tipo Replicator che subisce una duplicazione dato che è quello più esterno; al suo posto ci sono ora $D(i,1)$ e $D(i,2)$, i due dischi fissi del sottosistema $S(i)$; dato che si tratta di un evento di base, la sua duplicazione non comporta ulteriori modifiche all'albero.

Nella Fig. 4.12 si vede come l'input replicato della porta logica relativa a SKN è divenuto un insieme di eventi; al posto di $S(i)$ ora ci sono $S(1), S(2), S(3)$.

La Fig. 4.13 mostra la copia del sottoalbero che si trovava sotto $S(i)$ per $S(3)$; al posto di ogni riferimento al parametro i ora c'è l'indice 3; i componenti D che avevano già subito una duplicazione, sono stati ulteriormente duplicato come si nota dal doppio indice, in cui la prima cifra indica il sottosistema e la seconda il numero del disco fisso.

Per $S(1)$ e $S(2)$ sono stati generati dei sottoalberi analoghi, solo che al posto di i c'è 1 per $S(1)$ e 2 per $S(2)$.

L'evento Mg non ha parametro quindi non subisce alcuna duplicazione; esso sarà quindi condiviso da $S(1)$, $S(2)$ e $S(3)$ e farà sempre riferimento allo stesso dispositivo.

4.4.6 Ordinamento di eventi e porte logiche

Dato che Sharpe richiede che le dichiarazioni degli elementi dell'albero seguano un'ordinazione per cui ogni evento prima di essere indicato come input di una porta logica deve essere stato definito in precedenza, vengono a questo punto ricalcolati i numeri d'ordine secondo una visita in profondità, questa volta però tenendo conto anche dei nuovi nodi dell'albero.

4.4.7 Generazione del file per Sharpe

La procedura per la generazione del file per Sharpe prevede la scrittura nel file delle seguenti parti:

1. **commento**;

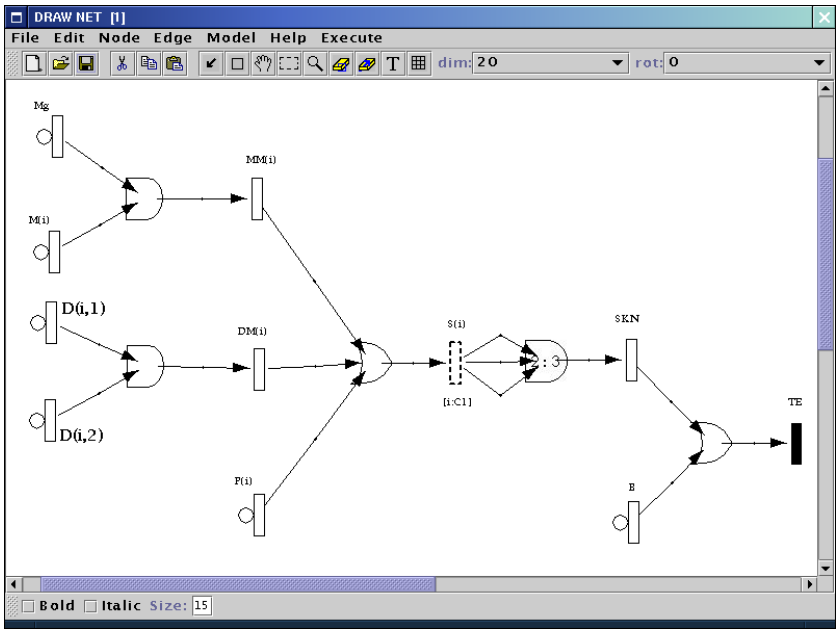


Figura 4.11: Duplicazione di D(i,j) rispetto a j

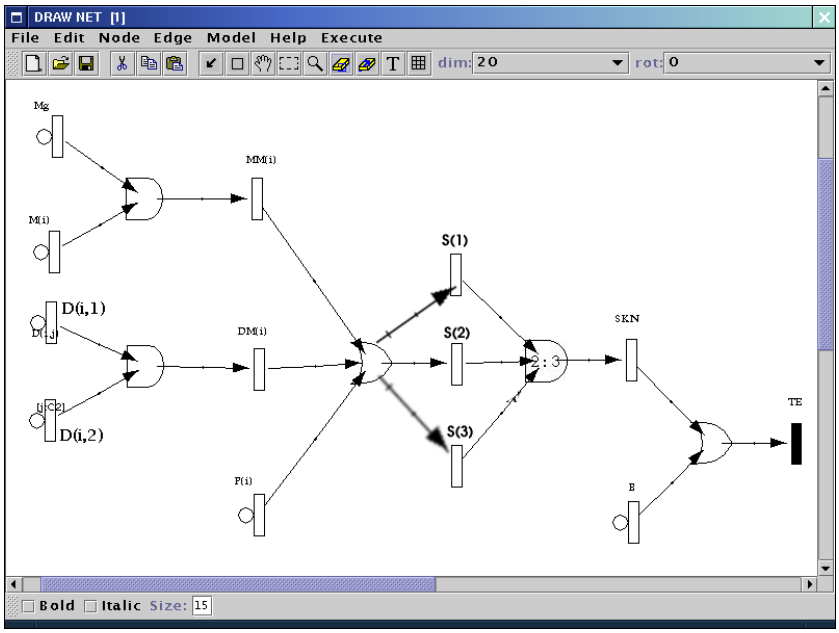


Figura 4.12: Duplicacazione dell'evento S(i) rispetto a i

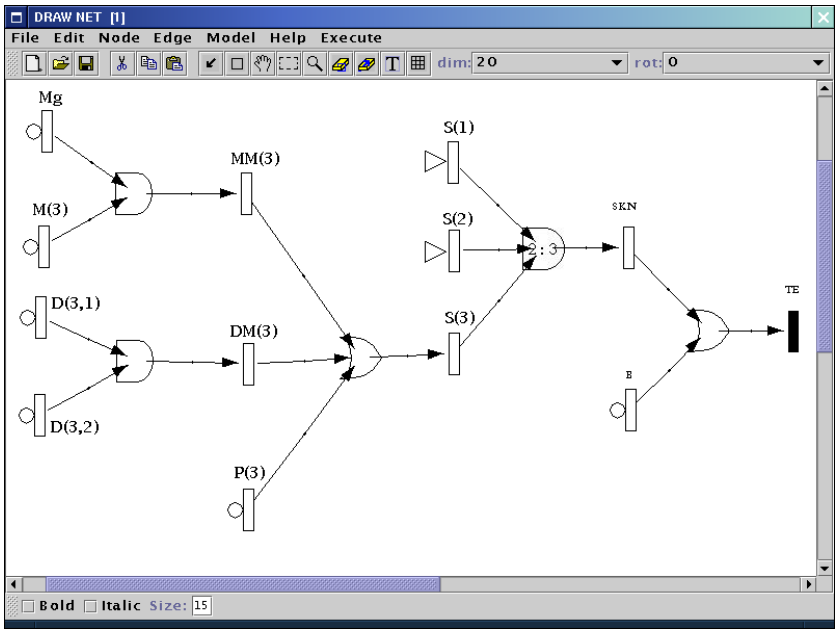


Figura 4.13: Duplicazione del sottoalbero S(i)

2. **nome dell'albero** preceduto dalla parola chiave *ftree* che indica a Sharpe che si tratta di un albero dei guasti;

3. dichiarazione degli **eventi di base**;

4. dichiarazione delle **porte logiche**;

5. la parola chiave **end** che indica la fine delle dichiarazioni relative alla struttura dell'albero;

6. dichiarazione delle **costanti** tra la parola chiave *bind* e il relativo *end*;

7. dichiarazione degli **indici** da calcolare;

8. **end** finale.

Il primo elemento inserito è un commento all'inizio del file preceduto da *; DrawNet permette di indicare un commento all'albero che viene indicato nel file XML generato dopo il tag PFT nell'attributo Title; questo se specificato viene trasferito nel file per Sharpe come commento.

Sulla stessa riga del file XML, nel campo name è indicato il nome dell'albero che viene anch'esso scritto nel file per Sharpe dopo la parola chiave ftree.

Seguono le dichiarazioni degli eventi di base con la rispettiva distribuzione di probabilità (esponenziale), le dichiarazione delle porte logiche con il relativo eventi (output e input) e le costanti (dopo la parola chiave bind). L'ordine delle dichiarazione delle porte logiche rispecchia il numero d'ordine degli eventi di output corrispondenti.

L'ordine degli eventi di base è quello con cui sono memorizzati, quindi nessuno in particolare; questo perchè, dato che sono dichiarati tutti all'inizio, prima delle porte logiche, è sicuro che ogni riferimento nei loro confronti avverrà dopo la loro dichiarazione.

A questo punto dovrebbe seguire la richiesta di quali dati di output si richiedono da Sharpe; il traduttore indica la media e la varianza del tempo di guasto del sistema.

Gli eventi di base sono dichiarati tutti di tipo *repeat*: in questo modo un nome di evento riguarda un solo dispositivo fisico; dato che è stata fatta un'operazione di *unfolding* si è sicuri che a nomi diversi corrispondono eventi diversi e quindi utilizzare il tipo *basic* per dichiarare gli eventi di base sarebbe inutile: è più adeguato dal punto di vista logico usare **repeat**.

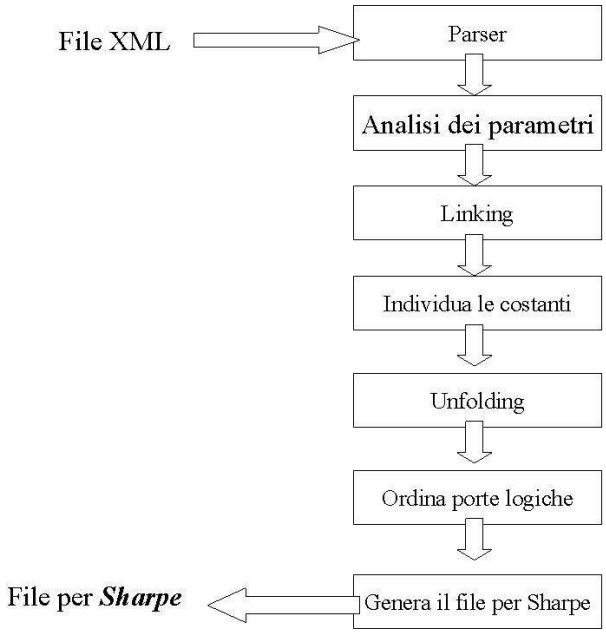


Figura 4.14: Fasi del traduttore

Caso di esempio

L'output del traduttore è il seguente file per Sharpe:

```
* drawnet2sharpe

ftree Multiproc

repeat Mg exp(v2)
basic B exp(v4)
basic P1 exp(v3)
basic D11 exp(v1)
basic D12 exp(v1)
basic M1 exp(v2)
basic P2 exp(v3)
basic D21 exp(v1)
basic D22 exp(v1)
basic M2 exp(v2)
basic P3 exp(v3)
basic D31 exp(v1)
basic D32 exp(v1)
basic M3 exp(v2)

and MM1 Mg M1
and DM1 D11 D12
or S1 P1 DM1 MM1
and MM2 Mg M2
and DM2 D21 D22
or S2 P2 DM2 MM2
and MM3 Mg M3
and DM3 D31 D32
or S3 P3 DM3 MM3
kofn SKN 2,3, S1 S2 S3
or TOP SKN B
end

bind
v1 0.000008
v2 0.000003
v3 0.000005
v4 0.000002
end
```

```
expr mean(Multiproc)
expr variance(Multiproc)
end
```

Il commento presente sulla prima riga indica che il file è stato generato dal traduttore; viene scritta questa stringa se nel listato XML generato da DrawNet, il campo **Title** è vuoto.

Se si chiede a Sharpe di analizzare l'albero di questo file richiedendo la probabilità di guasto del sistema al tempo t che varia tra 0 e 5000 con un passo di 250, Sharpe restituisce i seguenti valori:

```
      system Multiproc
        t              F(t)

 0.00000000 e+00  0.00000000 e+00
 2.50000000 e+02  5.04585778 e-04
 5.00000000 e+02  1.01843528 e-03
 7.50000000 e+02  1.54168464 e-03
 1.00000000 e+03  2.07446715 e-03
 1.25000000 e+03  2.61691326 e-03
 1.50000000 e+03  3.16915058 e-03
 1.75000000 e+03  3.73130386 e-03
 2.00000000 e+03  4.30349500 e-03
 2.25000000 e+03  4.88584307 e-03
 2.50000000 e+03  5.47846425 e-03
 2.75000000 e+03  6.08147190 e-03
 3.00000000 e+03  6.69497653 e-03
 3.25000000 e+03  7.31908577 e-03
 3.50000000 e+03  7.95390444 e-03
 3.75000000 e+03  8.59953452 e-03
 4.00000000 e+03  9.25607513 e-03
 4.25000000 e+03  9.92362258 e-03
 4.50000000 e+03  1.06022704 e-02
 4.75000000 e+03  1.12921092 e-02
 5.00000000 e+03  1.19932268 e-02
```

Se si volesse esaminare questo albero anche con Astra, sarebbe sufficiente passarlo al traduttore *sharpe2astra* che in questo caso genera il seguente file:

```
*drawnet2sharpe
```

```
[EVENTS]
B   2.000000E-006 0 0 0 0 "sharpe2astra"
D11   8.000000E-006 0 0 0 0 "sharpe2astra"
D12   8.000000E-006 0 0 0 0 "sharpe2astra"
D21   8.000000E-006 0 0 0 0 "sharpe2astra"
D22   8.000000E-006 0 0 0 0 "sharpe2astra"
D31   8.000000E-006 0 0 0 0 "sharpe2astra"
D32   8.000000E-006 0 0 0 0 "sharpe2astra"
M1    3.000000E-006 0 0 0 0 "sharpe2astra"
M2    3.000000E-006 0 0 0 0 "sharpe2astra"
M3    3.000000E-006 0 0 0 0 "sharpe2astra"
MG    3.000000E-006 0 0 0 0 "sharpe2astra"
P1    5.000000E-006 0 0 0 0 "sharpe2astra"
P2    5.000000E-006 0 0 0 0 "sharpe2astra"
P3    5.000000E-006 0 0 0 0 "sharpe2astra"
[GATES]
TOP OR GSKN B "sharpe2astra"
GSKN 2/3 GS1 GS2 GS3 "sharpe2astra"
GS1 OR P1 GDM1 GMM1  "sharpe2astra"
GS2 OR P2 GDM2 GMM2  "sharpe2astra"
GS3 OR P3 GDM3 GMM3  "sharpe2astra"
GDM1 AND D11 D12     "sharpe2astra"
GMM1 AND MG M1       "sharpe2astra"
GDM2 AND D21 D22     "sharpe2astra"
GMM2 AND MG  M2      "sharpe2astra"
GDM3 AND  D31 D32    "sharpe2astra"
GMM3 AND  MG M3      "sharpe2astra"
$END
```

La Fig. 4.15 mostra come appare l'albero in Astra, mentre la Fig. 4.16 contiene i risultati dati da Astra per gli stessi valori di tempo richiesti a Sharpe.

Il divario tra i risultati di Sharpe (esatti) e quelli di Astra (limiti superiori e limite inferiore) è abbastanza grande a causa del numero elevato di MCS (28 e dipendenti) anche se i risultati di Sharpe sono comunque all'interno dell'intervallo di valori tra i limiti di Astra.

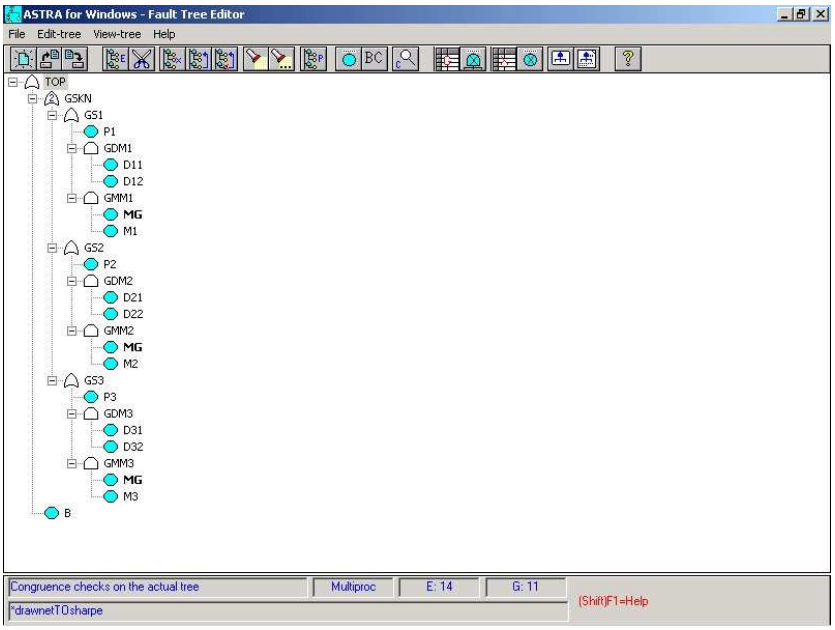


Figura 4.15: L'albero di Multiproc come appare in Astra

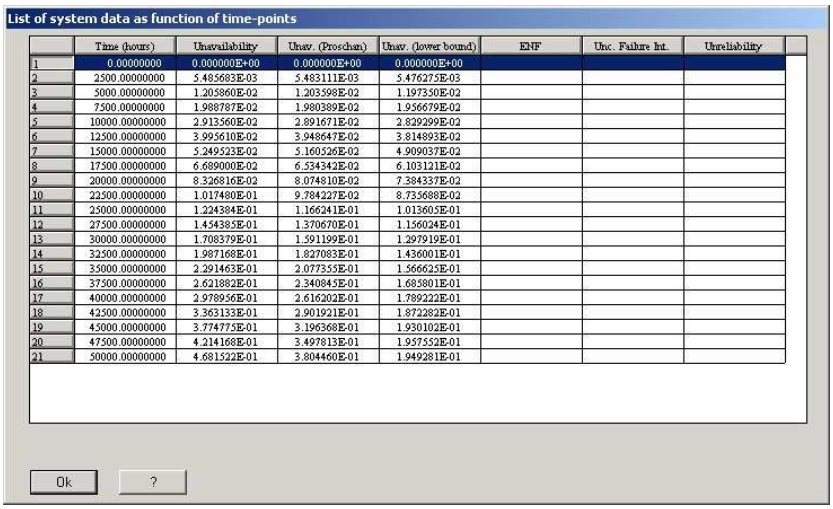


Figura 4.16: I risultati calcolati da Astra su Multiproc

Capitolo 5

Individuazione dei moduli

5.1 Introduzione

L'analisi di un albero dei guasti di grandi dimensioni può richiedere un numero elevato di operazioni; per ridurre i costi computazionali si può ricorrere alla scomposizione dell'albero in varie parti e all'analisi di ciascuna di queste; raccogliendo e sintetizzando i vari risultati si determina poi una valutazione dell'intero albero.

Le parti in cui l'albero viene suddiviso prendono il nome di **moduli** e corrispondono a parti dell'albero indipendenti tra loro cioè parti i cui eventi non appaiono altrove nell'albero.

In particolare un modulo viene indicato con il nome dell'evento alla radice del sottoalbero indipendente.

Questo permette di non considerare l'albero nelle sua interezza, ma di esaminare uno per volta i suoi moduli; in questo modo non solo l'analisi è più semplice, ma anche i risultati sono maggiormente comprensibili; i MCS di un albero di grandi dimensioni, ad esempio, potrebbero essere molti e di ordine elevato, magari dipendenti tra loro, rendendo difficile la comprensione del loro ruolo nell'affidabilità del sistema.

Alberi di grandi dimensioni non sono rari se si tiene conto di sistemi reali o di alberi che sono il risultato di operazioni di *unfolding* su PFT.

La procedura di analisi di un albero dei guasti per moduli dovrebbe seguire questi passi:

1. determinare i moduli tramite un algoritmo;
2. risolvere ciascun modulo, cioè calcolarne la probabilità di guasto;
3. sostituire nell'albero originale ogni modulo con un evento di base la cui probabilità dipende dal risultato del modulo corrispondente;

4. risolvere l'albero originale ora semplificato.

5.2 Un algoritmo lineare per determinare i moduli

Si considera come esempio l'albero di Fig. 5.1; l'espressione booleana corrispondente è la seguente:

$$(((e_1 \wedge e_2) \vee (e_3 \wedge e_4)) \wedge ((e_3 \wedge e_4) \vee (e_5 \wedge e_6)))$$

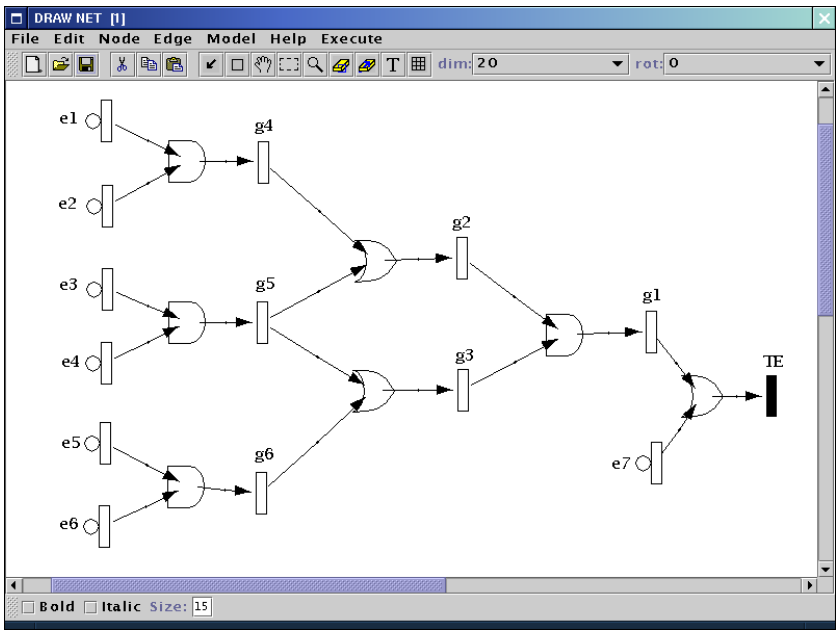


Figura 5.1: Albero di esempio

L'algoritmo [12] [9] determina i moduli eseguendo due visite in profondità; ad ogni evento dell'albero vengono associati tre valori di tempo (timestamp): t_1, t_2, t_{last}.

La Tab. 5.1 indica l'ordine con cui vengono visitati i nodi e a quali valori di tempo.

La parte di albero al di sotto di un evento non viene mai attraversato due volte; ad esempio il sottoalbero sotto g_5 viene visitato quando si proviene da g_2, ma non quando si proviene da g_3; si rivisita solo l'evento interno.

Nella prima visita in profondità partendo dal *Top Event* vengono inizializzati per ogni nodo dell'albero t_1, t_2 e t_{last} che indicano rispettivamente il

t	1	2	3	4
nodo	TE	g_1	g_2	g_4
t	5	6	7	8
nodo	e_1	e_2	g_4	g_5
t	9	10	11	12
nodo	e_3	e_4	g_5	g_2
t	13	14	15	16
nodo	g_3	g_5	g_6	e_5
t	17	18	19	20
nodo	e_6	g_6	g_3	g_1
t	21	22		
nodo	e_7	TE		

Tabella 5.1: Tempi di visita dell'albero

tempo della prima, della seconda e dell'ultima volta in cui il nodo è stato visitato.

Infatti ogni nodo non terminale viene visitato almeno due volte: la prima volta scendendo dal nodo genitore, la seconda ritornando dall'ultimo figlio.

Un nodo foglia viene visitato un numero di volte pari al numero di archi che lo toccano; se un nodo foglia è collegato da un solo arco t_1 e t_{last} saranno uguali e t_2 non sarà definito.

Nella seconda visita si individua per ogni nodo interno il minimo dei valori t_1 (min_{t_1}) e il massimo dei valori t_{last} ($max_{t_{last}}$) tra tutti i nodi sottostanti.

Un sottoalbero è un modulo se per l'evento g_i alla sua radice valgono entrambe le condizioni:

- $g_i.min_{t_1}(sottalbero) > g_i.t_1$
- $g_i.max_{t_{last}}(sottoalbero) < g_i.t_2$

Sono inoltre moduli per definizione il *Top Event* e gli eventi di base (nodi foglia).

La prima condizione stabilisce che in un modulo tutti gli eventi contenuti al suo interno devono essere visitati per la prima volta dopo la prima visita del nodo alla radice; la seconda condizione invece, afferma che gli eventi sottostanti devono essere visitati per l'ultima volta prima della seconda visita dell'evento alla radice.

Se entrambe le condizioni sono verificate, significa che tutti gli elementi del sottoalbero sono visitati solo passando dall'evento alla radice e quindi non sono condivisi con altri sottalberi.

La linearità dell'algoritmo è facilmente dimostrabile: consiste di due visite in profondità che richiedono un tempo lineare rispetto al numero dei nodi dell'albero.

L'algoritmo ha più precisamente questa forma:

```
VISITA1(nodo)
{
  tempo:=tempo+1;
  nodo.num_visite:=nodo.num_visite+1;
  nodo.last:=tempo;

  if (nodo.tipo=foglia)
    if (nodo.num_visite=1)
       nodo.t1:=tempo;
    else
      if (nodo.num_visite=2)
        nodo.t2:=tempo;


  if (nodo.tipo=interno)
    if (nodo.num_visite=1)
    {
      nodo.t1:=tempo;
      for i:=1 to nodo.num_figli
        VISITA1(nodo.figlio(i));
      nodo.num_visite:=nodo.num_visite+1;
      nodo.t2:=tempo;
      nodo.last=tempo;
    }
}



VISITA2(nodo)
{

  for i:=1 to nodo.num_figli
  {

    if (i=1)
      if (nodo.figlio(1).tipo = foglia)
```

```
      {
        nodo.min_t1:=nodo.figlio(1).t1;
        nodo.max_last:=nodo.figlio(1).last;
      }
      else
      {
        VISITA2(nodo.figlio(1));
        nodo.min_t1:=nodo.figlio(1).t1;
        nodo.max_last:=nodo.figlio(1).last;
      }

    else

      if (nodo.figlio(i).tipo = foglia)
      {
        if (nodo.figlio(i).t1 < nodo.min_t1)
          nodo.min_t1:=nodo.figlio(i).t1;
        if (nodo.figlio(i).last > nodo.max_last)
          nodo.max_last:=nodo.figlio(i).last;
      }
      else
      {
        VISITA2(nodo.figlio(i));
        if (nodo.figlio(i).t1 < nodo.min_t1)
          nodo.min_t1:=nodo.figlio(i).t1;
        if (nodo.figlio(i).last > nodo.max_last)
          nodo.max_last:=nodo.figlio[i]].last;
      }
  }

  if ((nodo.min_t1 > nodo.t1) and (nodo.max_last < nodo.t2))
    nodo \‘e un modulo

  if (nodo.min_t1 < nodo.t1)
    nodo.t1:=nodo.min_t1;

  if (nodo.max_last > nodo.last)
    nodo.last:=nodo.max_last;

}
```

```
begin

  tempo:= 0;

  VISITA1(Top Event);
    //Imposta i timestamp

  VISITA2(Top Event);
    //Calcola min_t1 e max_last per ogni nodo interno
        relativamente al suo sottoalbero

end
```

La procedura ricorsiva $VISITA1$ verifica al momento della visita di un nodo il numero di visite che esso ha subito, compresa quella corrente; quindi aggiorna il valore di t_{last} relativo al nodo visitato con il valore del tempo corrente; quest'ultimo viene incrementato ad ogni visita.

Se il nodo visitato è una foglia dell'albero, la procedura semplicemente assegna il valore di t_1 se si tratta della prima visita oppure assegna il valore di t_2 se si tratta della seconda.

Se invece il nodo visitato è interno e questo viene visitato per la prima volta, sarà necessario visitare anche i nodi discendenti; perciò si assegna il valore di t_1 e la procedura viene rilanciata ricorsivamente per ogni figlio del nodo corrente; il valore di t_2 verrà assegnato dopo aver visitato tutto il sottoalbero ed essere quindi ritornati a questo nodo.

Se il nodo interno viene nuovamente visitato dopo la prima e la seconda visita, si aggiorna solo il valore del suo t_{last} e il relativo sottoalbero non viene più visitato.

La procedura $VISITA2$ deve determinare per ogni nodo interno, min_{t_1} e $max_{t_{last}}$; la procedura scende ricorsivamente fino a un nodo interno che ha come figli solo foglie dell'albero; per questo nodo interno determina min_{t_1} e $max_{t_{last}}$ confrontando tra loro i valori t_1 e t_{last} relativi a ciascun nodo figlio; quindi verifica che il nodo interno identifichi un modulo.

A questo punto, al valore t_1 dell'evento interno viene assegnato il relativo min_{t_1}, mentre il valore t_2 diventa pari a max_{tlast}; questi assegnamenti vengono effettuati in modo tale che, valutando i nodi a livello superiore, si possano determinare min_{t_1} e $max_{t_{last}}$ relativi a tutti i nodi discendenti, anziché ai soli figli.

Infatti per i nodi interni a livello superiore, si può così determinare min_{t_1} e $max_{t_{last}}$ confrontando tra loro i valori t_1 e t_{last} dei soli nodi figli.

L'ordine con cui vengono valutati i nodi interni dell'albero di esempio di Fig. 5.1, per verificare se identificano dei moduli, è il seguente: g_4, g_5, g_2, g_5, g_6, g_3, g_1; prima di valutare un nodo interno, devono essere stati valutati tutti i nodi interni suoi figli.

Può capitare che un nodo interno venga esaminato due volte, come nel caso di g_5, portando comunque allo stesso risultato; per evitare questo inconveniente, prima di esaminare un modulo, si potrebbe verificare che questo non sia già avvenuto in precedenza, attraverso una variabile di *flag* che indica l'avvenuta o meno valutazione del nodo.

La Tab. 5.2 e la Fig. 5.2 mostrano i valori di t_1, t_2, t_{last} per gli eventi di base e per quelli interni; gli eventi di base hanno il primo e il terzo valore uguali in quanto nell'esempio gli eventi di base non sono condivisi e quindi vengono visitati una sola volta; per questo t_2 non è definito.

I nodi interni hanno t_1 diverso da t_2, dato che vengono visitati almeno due volte, e tutti hanno t_2 uguale a t_{last} ad eccezione di g_5 che è l'unico evento raggiunto da due archi e quindi visitato 3 volte.

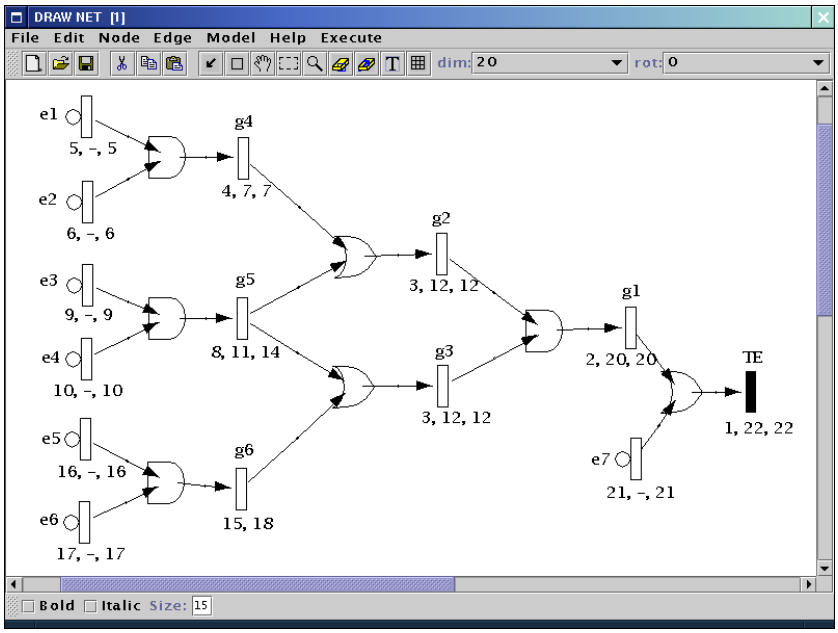


Figura 5.2: Albero di esempio con timestamp

La Tab. 5.3 mostra quali eventi interni sono alla radice di moduli.

nodo	t_1	t_2	t_{last}
e_1	5	-	5
e_2	6	-	6
e_3	9	-	9
e_4	10	-	10
e_5	16	-	16
e_6	17	-	17
e_7	21	-	21
g_1	2	20	20
g_2	3	12	12
g_3	13	19	19
g_4	4	7	7
g_5	8	11	14
g_6	15	18	18

Tabella 5.2: Tempi della prima, seconda ed ultima visita dei nodi dell'albero

nodo	min_{t_1}	$max_{t_{last}}$	modulo?
g_1	3	19	SI
g_2	4	14	NO
g_3	8	18	NO
g_4	5	6	SI
g_5	9	10	SI
g_6	16	17	SI

Tabella 5.3: Risultati dell'algoritmo

t	1	2	3	4
nodo	TE	g_1	g_2	g_4
t	5	6	7	8
nodo	e_1	e_2	e_3	g_4
t	9	10	11	12
nodo	g_5	e_3	e_4	g_5
t	13	14	15	16
nodo	g_2	g_3	g_5	g_6
t	17	18	19	20
nodo	e_4	e_5	e_6	g_6
t	21	22	23	24
nodo	g_3	g_1	e_7	TE

Tabella 5.4: Tempi di visita dell'albero

Nell'esempio, sono dei moduli, oltre al *Top Event* e agli eventi di base, i sottoalberi sotto g_1, g_4, g_6; se consideriamo g_1 e immaginiamo di staccarlo dall'albero, i suoi eventi di base non sono collegati a nessun altra parte dell'albero.

Possiamo quindi staccarlo e sostituirlo con la sua soluzione; lo stesso concetto vale per gli altri moduli.

g_2 e g_3 non sono moduli in quanto condividono g_5 che viene visitato passando da entrambi; infatti i suoi valori di tempo mettono in evidenza che g_5 è stato attraversato dopo aver ultimato la visita di g_2 e prima di iniziare la visita di g_3.

Nel caso di g_2, che non è un modulo, se lo staccassimo dall'albero, si perderebbe g_5 compromettendo g_3 che condivide dei nodi con g_2.

Un modulo può comunque contenere al suo interno altri moduli, come nel caso di g_1 che racchiude g_4 e g_6; in questa situazione si possono staccare sia i moduli più interni sia quello esterno senza rimuovere eventi condivisi. Si potrebbe staccare prima i moduli più interni, sostituirli e poi risolvere il modulo esterno.

In questo esempio tutti i nodi foglia sono visitati una volta sola e perciò hanno t_1 e t_{last} uguali e t_2 non definito; questo perché sono toccati da un solo arco.

Modifichiamo lievemente questo albero aggiungendo degli archi per far condividere degli eventi di base da alcune porte logiche; il risultato è quello di Fig. 5.3; la Tab. 5.4 mostra i tempi di visita dei nodi.

In Fig. 5.3 e_3 viene condiviso da g_4 e g_5, mentre e_4 è condiviso da g_5 e g_6.

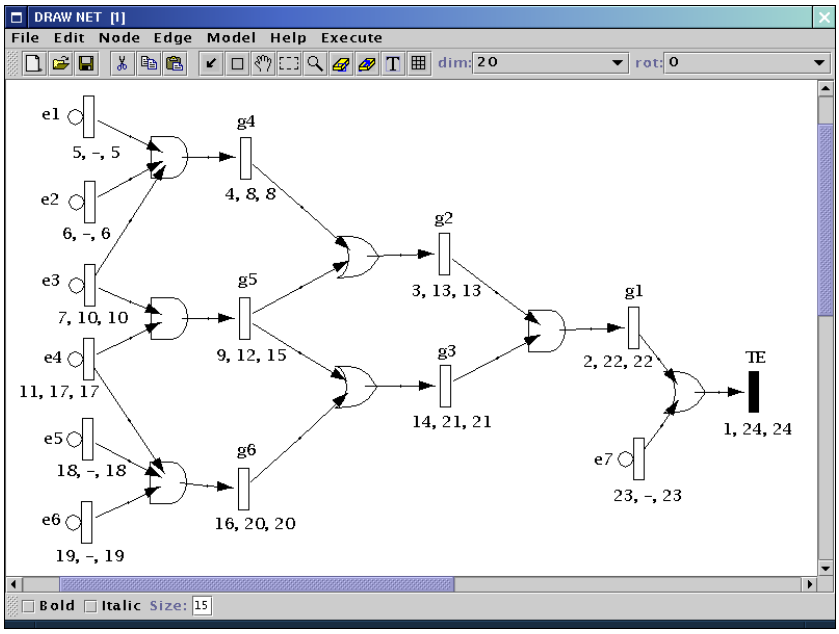


Figura 5.3: Nuovo albero di esempio con timestamp

La Tab. 5.5 mostra i valori di t_1, t_2 e t_{last} per i nodi dell'albero; si nota che per gli eventi di base e_3 e e_4 è definito anche t_2 dato che questi vengono visitati due volte.

Dai risultati della Tab. 5.6 si vede che l'unico modulo è g_1 oltre a TE; infatti e_3 viene visitato anche dopo g_4 e prima di g_5 e perciò g_4 e g_5 non sono moduli; g_6 non è un modulo a causa di e_4 che viene visitato anche prima di g_6.

g_1 è un modulo in quanto tutta la parte sottostante viene visitata in tempi compresi tra i valori di t_1 e t_2 relativi a g_1.

5.3 Conclusioni

L'utilità di individuare i moduli di un albero non riguarda soltanto l'aspetto dell'analisi; è utile anche quando si vuole sostituire un sottosistema: un modulo può essere eliminato dall'albero e sostituito da un sottoalbero completamente diverso senza compromettere altre parti.

L'algoritmo, fatto girare sull'albero relativo al sistema *Multiproc* dopo aver effettuato l'*unfolding* (Fig. 4.13) ha determinato che i moduli sono: $DM(1)$, $DM(2)$, $DM(3)$, SKN e TE.

nodo	t_1	t_2	t_{last}
e_1	5	-	5
e_2	6	-	6
e_3	7	10	10
e_4	11	17	17
e_5	18	-	18
e_6	19	-	19
e_7	23	-	23
g_1	2	22	22
g_2	3	13	13
g_3	14	21	21
g_4	4	8	8
g_5	9	12	15
g_6	16	20	20

Tabella 5.5: Tempi della prima, seconda ed ultima visita dei nodi dell'albero

nodo	min_{t_1}	$max_{t_{last}}$	modulo?
g_1	3	21	SI
g_2	4	15	NO
g_3	7	20	NO
g_4	5	10	NO
g_5	7	17	NO
g_6	11	19	NO

Tabella 5.6: Risultati dell'algoritmo

Infatti $MM(1)$, $MM(2)$ e $MM(3)$ non possono essere moduli in quando condividono l'evento Mg e per lo stesso motivo non sono moduli $S(1)$, $S(2)$ e $S(3)$.

Per provare l'algoritmo, questo è stato in un primo momento inserito all'interno del traduttore da DrawNet a Sharpe; dopo aver effettuato l'eventuale *unfolding* e la traduzione, si applica l'algoritmo all'albero in memoria.

Capitolo 6

Alberi con eventi riparabili

6.1 Introduzione

La trattazione fatta fino ad ora a proposito di alberi dei guasti ha considerato soltanto la modellazione di sistemi in cui il guasto di un componente fisico (*evento di base*), di un sottosistema (*evento interno*) o dell'intero sistema (*Top Event*) portava ad uno stato da cui non era possibile uscire; infatti secondo il modello considerato fin qui, dal momento in cui si verifica il guasto di una parte, questa non funzionerà mai più.

Si vorrebbe invece creare un modello in cui sia possibile riportare una parte dell'albero che si è guastata allo stato funzionante; per realizzare ciò è stato aggiunto nel modello un nuovo tipo di nodo detto **scatola di riparazione** (Repair Box) che viene collegato tramite un apposito arco ad un qualsiasi evento volendo significare che, se si verifica un guasto in quel componente o in quel sottosistema, lo si può riparare, facendolo tornare in funzione.

La Fig. 6.1 mostra un esempio di albero con eventi riparabili disegnato con DrawNet; si tratta dell'esempio di Fig. 5.1 a cui sono sono state aggiunte due scatole di riparazione: la prima collegata al sottosistema g_2 e la seconda al dispositivo e_7.

6.2 Aggiornamento di DrawNet alle scatole di riparazione

Per disegnare le scatole di riparazione con DrawNet si è resa necessaria la modifica del formalismo che descrive i PFT e che viene passato a DrawNet al suo avvio.

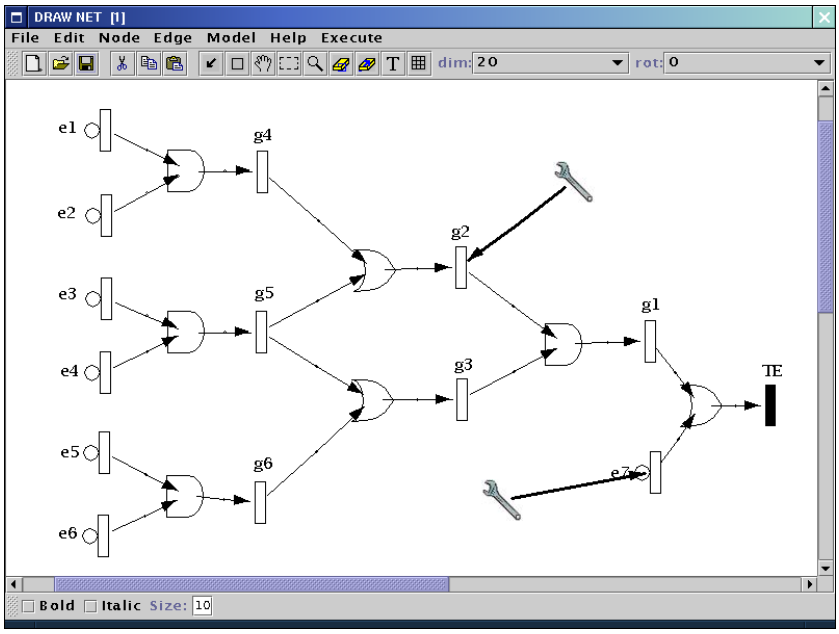


Figura 6.1: Albero di esempio con scatole di riparazione

In particolare si è aggiunto un nuovo tipo di nodo per la scatola di riparazione chiamato **Repair** che viene rappresentato a livello grafico dalla figura di una chiave inglese; per mettere ulteriormente in risalto la presenza di una scatola di riparazione è stato introdotto anche un nuovo tipo di arco denominato *Wrench* (in inglese significa appunto chiave inglese) con la specifica ed unica funzione di collegare la scatola con l'evento di guasto.

Nel formalismo così modificato si è anche stabilito che una scatola di riparazione può riguardare un unico evento (di base o interno) e che un evento può essere legato ad una sola scatola di riparazione.

Seguono le definizioni aggiunte al formalismo a tale scopo:

```
<nodeType parent="" name="Repair">
    <propertyType name="Repair_Rate" default="1.0"/>
</nodeType>

<edgeType parent="" name="Wrench">
    <constraint fromType="Repair" fromCardinality="1"
     toType="Event0" toCardinality="1"/>
    <constraint fromType="Repair" fromCardinality="1"
     toType="TopEvent" toCardinality="1"/>
</edgeType>
```

Ogni scatola ha associato un *tasso di riparazione* (Repair_Rate) pari all'inverso del tempo medio necessario per effettuare la riparazione.

L'arco di tipo *Wrench* (chiave inglese) può collegare una sola scatola ad un solo evento e viceversa.

6.3 Semantica di riparazione

Prima di poter eseguire la riparazione di un componente o di un sottosistema si deve stabilire con maggiore precisione cosa significa e cosa comporta questa operazione; gli aspetti da valutare in questo senso sono diversi e influenzano non solo la modellazione del sistema, ma anche la sua analisi.

Un primo aspetto riguarda gli eventi coinvolti: se la scatola viene collegata ad un evento di base, la riparazione riguarderà soltanto un componente fisico; se la scatola è invece collegata ad un evento interno, la riparazione avverrà per tutti i componenti dell'albero sottostante che sono guasti, dato che se si verifica un guasto a livello di evento interno, sarà dovuto alla propagazione, attraverso le porte logiche, dai componenti fisici guasti (eventi di base) all'evento interno.

Bisogna anche stabilire in quale momento la riparazione può avvenire:

- immediatamente dopo il guasto;
- dopo aver verificato la presenza del guasto; questo controllo verrebbe fatto dalla scatola di riparazione ad intervalli di tempo regolari; in caso affermativo la scatola eseguirebbe la riparazione;
- quando un sottosistema non è più funzionante e allora si mettono in funzione tutte le scatole di riparazione contenute al suo interno;
- quando l'intero sistema è guasto.

Si può inoltre associare alla scatola di riparazione un tempo necessario per effettuare la riparazione; inoltre bisogna stabilire se durante una riparazione il resto del sistema continua a funzionare oppure si arresta in attesa che la riparazione venga effettuata.

C'è poi il problema relativo al verificarsi di un guasto all'interno di un sottosistema durante la sua riparazione: se nel sottosistema c'è più di un componente guasto, durante la riparazione di uno di questi, se ne potrebbe guastare un altro appena riparato.

Un sottosistema riparabile potrebbe poi contenere al suo interno delle ulteriori parti riparabili; in questo caso alcuni componenti potrebbero essere riparati in corrispondenza della presenza di guasti in punti diversi dell'albero.

E' necessario quindi decidere con precisione cosa è in grado di fare una scatola di riparazione per poterla applicare con efficacia e fare un'analisi del sistema appropriata.

6.4 I moduli in presenza di riparatori

A livello di determinazione dei moduli di un albero dei guasti, la presenza di scatole di riparazione genera la presenza di ulteriori vincoli di dipendenza tra eventi dell'albero, in particolare la presenza di scatole collegate ad eventi interni come nel caso di Fig. 6.1.

Tutto ciò che si trova al di sotto di un evento riparabile non può essere un modulo in quanto la scatola di riparazione determina una dipendenza di tutti i componenti del sottoalbero nei confronti del nodo riparabile; infatti se si propaga il guasto fino a quest'ultimo e comincia la riparazione, questa opererà su tutti gli eventi di base sottostanti.

Se al di sotto dell'evento interno riparabile ci fosse un modulo e questo venisse analizzato separatamente, la sua valutazione non terrebbe in considerazione il fatto che i componenti del modulo sono riparabili in caso di guasto. Per questo tutto il sottoalbero con radice in un evento riparabile deve essere analizzato nella sua interezza.

L'algoritmo per l'individuazione dei moduli descritto nel capitolo precedente è stato modificato per tener conto delle scatole di riparazione; più precisamente al momento di valutare se un evento interno contraddistingue un modulo confrontando i valori di tempo, si tiene conto del fatto che l'evento si trovi o meno al di sotto di un evento riparabile; in caso affermativo non si tratta di un modulo.

Per capire se un evento si trova al di sotto di un altro riparabile, durante la seconda visita in profondità una variabile di nome *rep* viene impostata al valore 1 quando si attraversa un evento riparabile e rimane tale fino a quando la visita del sottoalbero sottostante non è stata ultimata; se un evento è un potenziale modulo secondo i valori di tempo, allora si verifica il valore della variabile *rep*: se *rep* vale 1 ci troviamo sotto un evento riparabile, quindi non si tratta di un modulo, altrimenti sì.

```
VISITA2(nodo, rep)
{


  if (nodo \‘e collegato a una scatola di riparazione)
    rep2:=1;
```

```
else
  rep2:=rep;

for i:=1 to nodo.num_figli
{

  if (i=1)
    if (nodo.figlio(1).tipo = foglia)
    {
      nodo.min_t1:=nodo.figlio(1).t1;
      nodo.max_last:=nodo.figlio(1).last;
    }
    else
    {
      VISITA2(nodo.figlio(1), rep2);
      nodo.min_t1:=nodo.figlio(1).t1;
      nodo.max_last:=nodo.figlio(1).last;
    }

  else

    if (nodo.figlio(i).tipo = foglia)
    {
      if (nodo.figlio(i).t1 < nodo.min_t1)
        nodo.min_t1:=nodo.figlio(i).t1;
      if (nodo.figlio(i).last > nodo.max_last)
        nodo.max_last:=nodo.figlio(i).last;
    }
    else
    {
      VISITA2(nodo.figlio(i), rep2);
      if (nodo.figlio(i).t1 < nodo.min_t1)
        nodo.min_t1:=nodo.figlio(i).t1;
      if (nodo.figlio(i).last > nodo.max_last)
        nodo.max_last:=nodo.figlio[i]].last;
    }
}

if ((nodo.min_t1 > nodo.t1) and (nodo.max_last < nodo.t2) and (rep=0))
  nodo \‘e un modulo
```

nodo	min_{t_1}	$max_{t_{last}}$	rep	modulo?
g_1	3	19	0	SI
g_2	4	14	0	NO
g_3	8	18	0	NO
g_4	5	6	1	NO
g_5	9	10	1	NO
g_6	16	17	0	SI

Tabella 6.1: Risultati dell'algoritmo aggiornato alle riparazioni

```
  if (nodo.min_t1 < nodo.t1)
    nodo.t1:=nodo.min_t1;

  if (nodo.max_last > nodo.last)
    nodo.last:=nodo.max_last;

}
```

In sintesi, le condizioni per cui un evento g_i è alla radice di un modulo sono:

- $g_i.min_{t_1}(sottoalbero) > g_i.t_1$
- $g_i.max_{t_{last}}(sottoalbero) < g_i.t_2$
- l'evento g_i non discende da eventi riparabili.

Nonostante la presenza delle scatole di riparazione, i tempi di visita dell'albero di Fig. 6.1 restano quelli indicati nelle Tab. 5.1 e 5.2; la Tab. 6.1 è un'estensione della Tab. 5.3: indica anche se il nodo si trova in una parte riparabile secondo il valore corrente della variabile *rep*; in base a questa e ai valori di tempo si decide se si tratta di un modulo o meno.

Se non ci fossero scatole di riparazione collegate, i moduli sarebbero g_1, g_4, g_5 e g_6; in questo esempio invece, g_4 e g_5 non sono moduli, in quanto sono discendenti del nodo g_2 a cui è collegata una scatola di riparazione che genera una dipendenza su tutto il suo sottoalbero.

Se si collegasse una scatola di riparazione al *Top Event*, si determinerebbe una dipendenza sull'intero albero e quindi ci sarebbe un unico modulo corrispondente all'albero stesso.

6.4.1 Riparazioni e nodi foglia

Non è più valida l'affermazione che i nodi foglia (eventi di base) sono moduli per definizione; infatti la dipendenza generata da una scatola di riparazione si estende fino alle foglie del relativo sottoalbero e quindi queste non possono essere moduli.

I nodi foglia sono quindi moduli se non discendono da eventi riparabili; potrebbero però avere delle scatole di riparazione collegate direttamente a loro; in questo caso vale comunque il principio per cui nessun modulo può discendere da eventi riparabili.

L'evento di base e_7 in Fig. 6.1 ha collegata una scatola, ma non discende da eventi riparabili: è quindi un modulo (riparabile); e_1, e_2, e_3 e e_4 non sono moduli perché discendono da g_2, mentre sono moduli, ma non riparabili, e_5 e e_6.

6.4.2 Riparazioni in cascata

Cosa succederebbe se si collegasse una scatola di riparazione anche a g_4 (Fig. 6.2)? In questo caso un sottoalbero riparabile farebbe parte di un ulteriore sottoalbero riparabile, ci sarebbero cioè due riparazioni in cascata: la scatola collegata a g_4 entrerebbe in funzione in caso di propagazione del guasto fino a questo nodo, cioè se si guastano e_1 e e_2; la scatola di riparazione di g_2 invece, opererebbe se si guasta g_4 o g_5 o se si guastano entrambi.

I componenti e_1 e e_2 potrebbero essere riparati quindi da entrambe le scatole; in questa situazione e_1 e e_2 non sono moduli in quanto discendono dall'evento riparabile g_4; questo a sua volta non è un modulo perchè discende da g_2; g_2 non è un modulo perchè condivide degli eventi con g_3.

I moduli sono g_6, g_1 e TE, oltre agli eventi di base e_5, e_6 e e_7, quest'ultimo riparabile.

Più complesso è il discorso da fare per stabilire quando e_1 e e_2 vengono riparati da una scatola e quando dall'altra.

- **Caso 1**: supponiamo che g_4 sia funzionante e g_5 guasto; il guasto si propagherà attraverso la porta *Or* a g_2, abilitando la relativa scatola che riparerà ogni componente sottostante a g_2 e quindi anche e_1 e e_2.

- **Caso 2**: se è invece g_4 ad essere guasto, allora e_1 e e_2 dovrebbero essere riparati dalla scatola connessa a g_4.

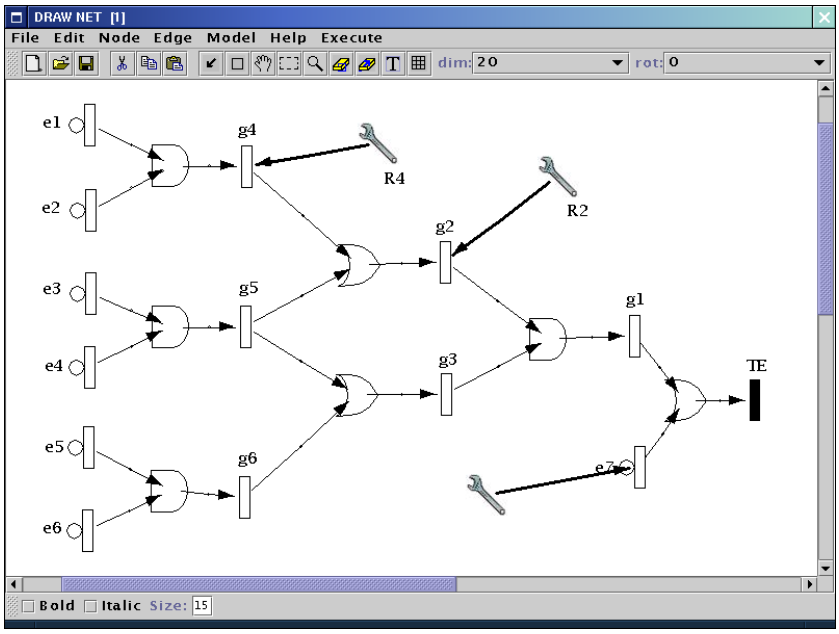


Figura 6.2: Albero di esempio con scatole di riparazione in cascata

6.4.3 Effetti della riparazione

Torniamo al primo caso, quello con g_5 guasto; in questa situazione il guasto non si propaga solo a g_2, ma anche a g_3 e di conseguenza al *Top Event*; quindi anche se le scatole di riparazione hanno riportato allo stato operativo alcuni componenti, il guasto si è comunque propagato a livello superiore.

Bisognerebbe allora, al termine di ogni riparazione, rideterminare lo stato dei nodi interni, a partire da quelli più vicini alla frontiera dell'albero; nel caso in questione g_2 sarà nuovamente funzionante e quindi anche g_1.

Un altro aspetto da definire è cosa succede durante la riparazione di un componente, agli altri dispositivi; se si stabilisce che durante la riparazione di un componente, gli altri si possano guastare, l'effetto della riparazione potrebbe essere nullo.

Supponiamo che la scatola di riparazione collegata a g_2 stia riparando il relativo sottoalbero; dopo aver riparato e_1, e_2, e_3, durante la riparazione di e_4, e_1 e e_2 si guastano entrambi nuovamente; il guasto si propagherà a g_4 e quindi a g_2 riportando quest'ultimo allo stato di guasto nonostante la riparazione appena avviata.

Sarebbe meglio che durante una riparazione, il funzionamento di tutti componenti del sottoalbero venisse bloccato; in questo modo si riparerebbero

quelli guasti e si impedirebbe agli altri di guastarsi, garantendo il buon fine della riparazione.

6.4.4 Conclusioni

Bisogna quindi stabilire una semantica ben precisa prima di applicare le scatole di riparazione, argomento del prossimo capitolo, dato che gli aspetti da valutare a tal proposito sono diversi.

L'analisi di alberi con eventi riparabili va fatta per moduli; più precisamente si devono staccare le parti dell'albero dove operano delle scatole di riparazione e analizzarle separatamente con tecniche appropriate.

L'ideale sarebbe che le scatole di riparazione fossero collegate ad eventi alla radice dei moduli e non ci fossero riparazioni in cascata; in questo modo non ci sarebbero conflitti di competenza tra riparatori e si semplificherebbe l'analisi.

6.5 Le riparazioni in Astra

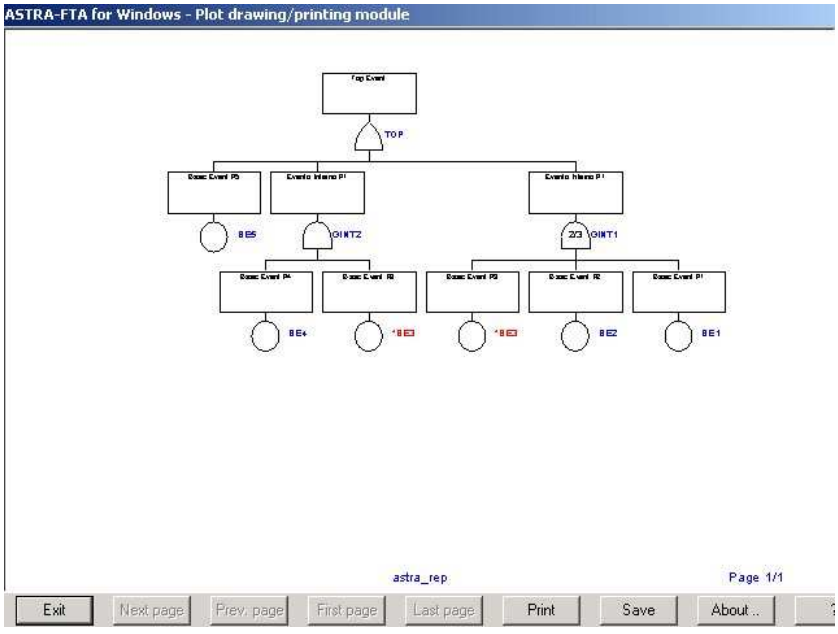


Figura 6.3: Albero di esempio

Astra permette la presenza di nodi dell'albero riparabili, ma questi possono essere solo componenti di base e non interni sottosistemi; consideriamo

come esempio l'albero di Fig. 6.3; al momento esso non ha componenti riparabili, infatti il relativo file con estensione **.tfx** ha questo contenuto:

```
astra_repair
[EVENTS]
BE1 0.00001 0 0 0 0 "Basic Event #1"
BE2 0.00002 0 0 0 0 "Basic Event #2"
BE3 0.00003 0 0 0 0 "Basic Event #3"
BE4 0.00004 0 0 0 0 "Basic Event #4"
BE5 0.00005 0 0 0 0 "Basic Event #5"
[GATES]
TOP OR GINT1 GINT2 BE5 "Top Event"
GINT1 2/3 BE1 BE2 BE3 "Evento Interno #1"
GINT2 AND BE3 BE4 "Evento Interno #2"
$END
```

Le probabilità di guasto del sistema al tempo di missione $T = 500000$ e quella relativa agli eventi di base sono indicate nelle Fig. 6.4 e 6.5; tutti gli eventi di base rispettano una distribuzione esponenziale; infatti l'unico parametro esplicitato per gli eventi di base è il primo; visto che gli eventi di base hanno tutti un tempo medio di vita (**MTTF**, cioè il valore inverso del tasso di guasto) piccolo rispetto a T le probabilità di guasto mostrate in Fig. 6.4 e in Fig. 6.5 sono prossime a 1.

Supponiamo ora che alcuni componenti del sistema, $BE1$, $BE3$ e $BE5$, siano riparabili; per indicare a Astra che un componente è riparabile secondo un determinato tasso di riparazione, si deve esplicitare nel file per Astra, il secondo parametro con tale valore; esso è l'inverso del tempo medio di riparazione (**MTTR**):

```
astra_repair
[EVENTS]
BE1 0.00001 0.5 0 0 0 "Basic Event #1"
BE2 0.00002 0 0 0 0 "Basic Event #2"
BE3 0.00003 0.025 0 0 0 "Basic Event #3"
BE4 0.00004 0 0 0 0 "Basic Event #4"
BE5 0.00005 0.0125 0 0 0 "Basic Event #5"
[GATES]
TOP OR GINT1 GINT2 BE5 "Top Event"
GINT1 2/3 BE1 BE2 BE3 "Evento Interno #1"
GINT2 AND BE3 BE4 "Evento Interno #2"
$END
```

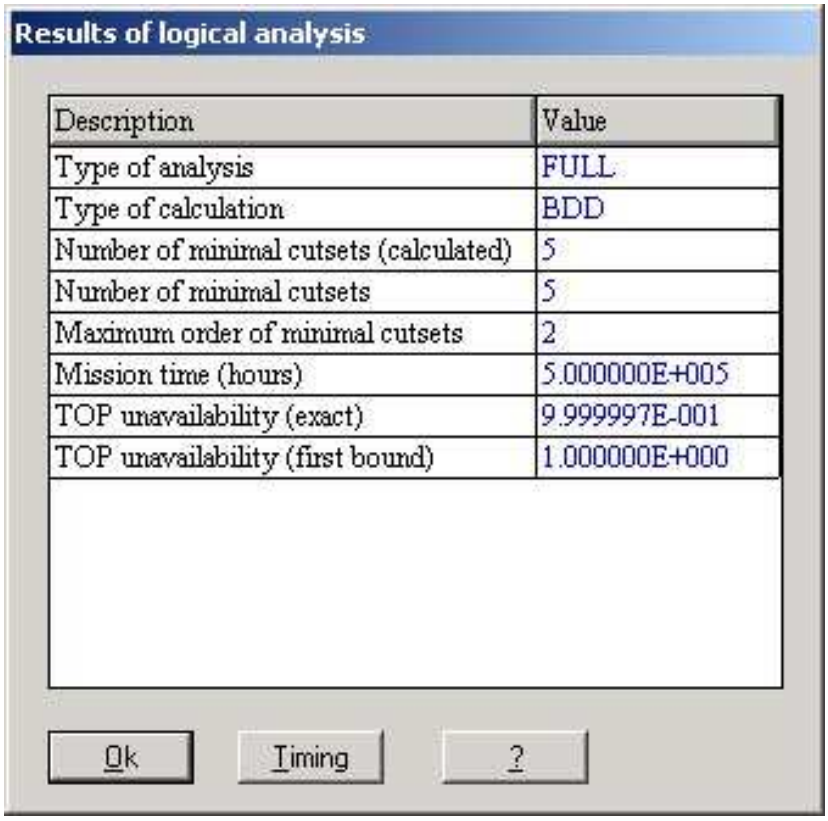


Figura 6.4: Indici di affidabilità del sistema

Figura 6.5: Le probabilità di guasto dei componenti al tempo di missione

Componente	λ	MTTF	μ	MTTR
B1	0.00001	100000	0.5	2
B2	0.00002	200000	-	-
B3	0.00003	300000	0.02	40
B4	0.00004	400000	-	-
B5	0.00005	500000	0.0125	80

Tabella 6.2: Tasso di guasto, tempo medio di vita, tasso di riparazione e durata media di riparazione per ogni componente

A $BE1$ corrisponde una riparazione veloce (Fig. 6.6) di tasso $\mu_1 = 0.5$, mentre $BE2$ e $BE3$ hanno dei riparatori più lenti, rispettivamente di tasso $\mu_2 = 0.02$ e $\mu_3 = 0.0125$ (Tab. 6.1); se non si esplicitano gli altri parametri (tempo del primo test e durata dell'intervallo tra due test), Astra intende che la riparazione ha inizio immediatamente dopo il guasto.

Astra calcola la probabilità di guasto al tempo T del componente E con tasso di guasto λ e tasso di riparazione μ T secondo la seguente formula:

$$P^E(T) = \frac{\lambda}{\lambda + \mu}(1 - e^{-(\lambda+\mu)T}).$$

La Fig. 6.7 mostra le probabilità di guasto del sistema al tempo $T = 500000$ dopo l'introduzione delle riparazioni, mentre la Fig. 6.8 indica quelle dei componenti di base.

Confrontando i valori di Fig. 6.5 con quelli di Fig. 6.8 si nota come la probabilità di guasto dei componenti riparabili si sia decisamente ridimensionata dato che per questi dispositivi si alternano periodi di funzionamento lunghi e di guasto brevi; infatti, se si considera ad esempio $B1$, il tempo medio di funzionamento (**MTTF**) prima di un guasto è pari a 100, mentre il tempo medio necessario per effettuare la riparazione (**MTTR**) è 2 (Tab. 6.1).

Ovviamente, la riduzione della probabilità di guasto di alcuni componenti ha un effetto positivo sull'affidabilità generale del sistema (Fig. 6.7).

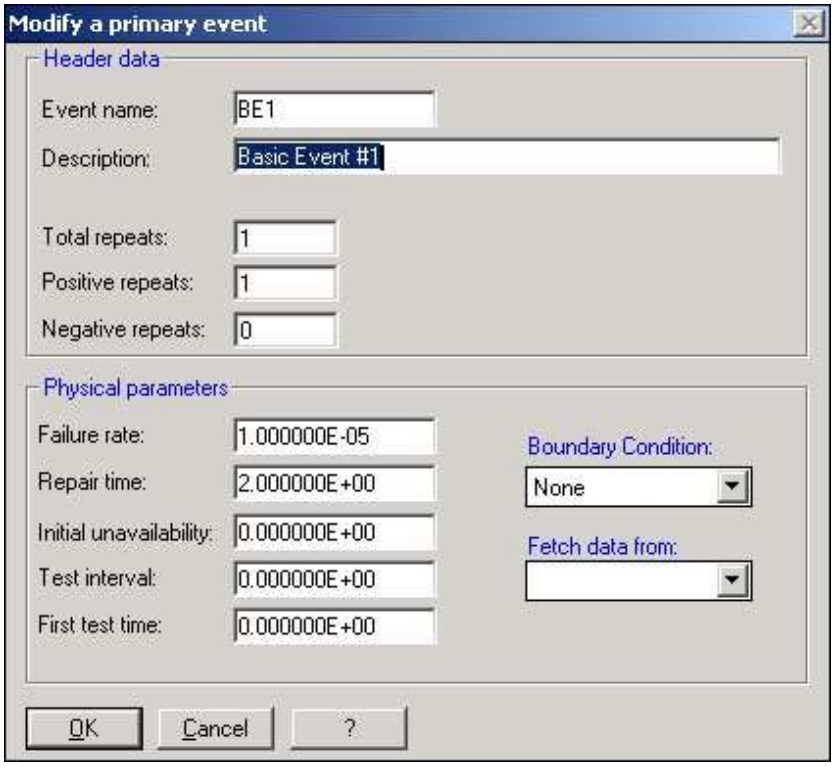


Figura 6.6: Inserimento del tempo di riparazione (MMTR) per $BE1$

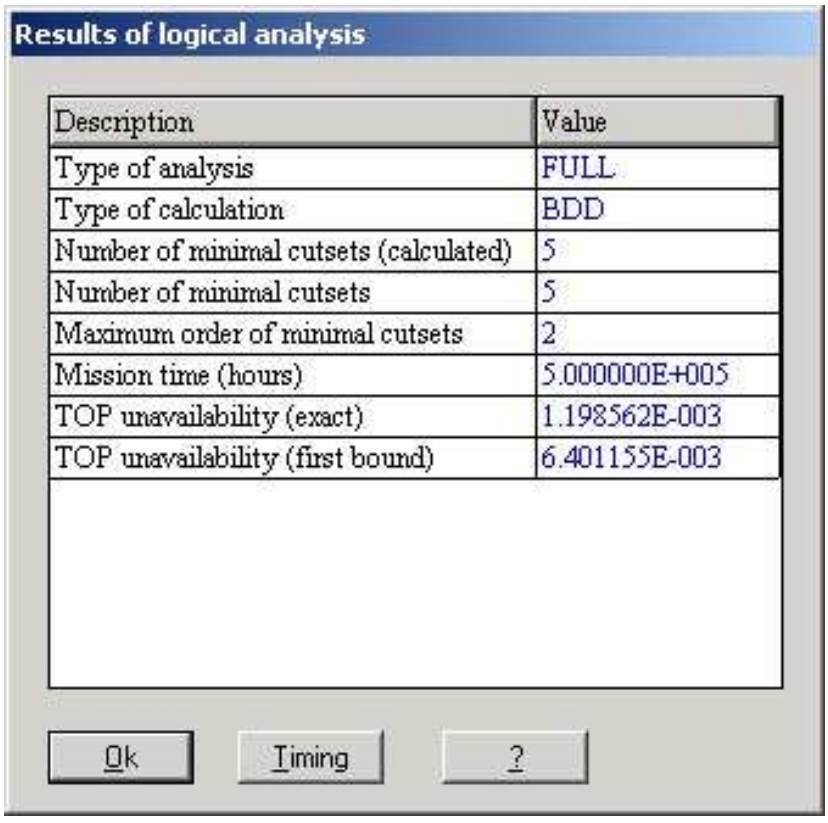


Figura 6.7: Indici di affidabilità del sistema dopo l'introduzione dei riparatori

Figura 6.8: Le probabilità di guasto dei componenti al tempo di missione dopo l'introduzione dei riparatatori

Capitolo 7

L'analisi per moduli

7.1 Introduzione

Come illustrato nei capitoli precedenti, la divisione in moduli di un albero dei guasti consente di semplificarne l'analisi considerando e valutando una parte per volta; l'analisi per moduli non è utile soltanto in questo senso, ma anche quando ci sono parti dell'albero che non possono essere analizzate con il metodo tipico degli alberi dei guasti cioè l'*analisi qualitativa e quantitativa*.

E' il caso degli alberi con eventi riparabili dove i sottoalberi il cui stato dipende da una scatola di riparazione, devono essere esaminati in modo diverso facendo ricorso alle Reti di Petri [14] [13] e/o alle Catene di Markov [10] (analisi nello spazio degli stati).

Le parti riparabili devono quindi essere staccate dall'albero, analizzate propriamente e sostituite da eventi di base che rispecchiano la probabilità di guasto in presenza di riparazione; fatto questo per ciascuna di esse ci si riconduce ad un albero senza riparatori da esaminare con il metodo tradizionale nella sua interezza o per moduli (non riparabili).

Quando si stacca una parte dell'albero, anche se riparabile, bisogna sempre controllare che questa non abbia eventi condivisi, che si tratti cioè di moduli.

7.2 Tipi di moduli

Supponendo di dover individuare i moduli di un albero anche con eventi riparabili si possono definire le seguenti categorie di moduli [8]:

- **MSC** *(**M**odulo a **S**oluzione **C**ombinatoria)*: si risolve in modo tradizionale e ha queste caratteristiche:

  - non contiene nessun evento condiviso con altri sottoalberi;
  - nessuno dei suoi eventi è collegato ad una scatola di riparazione;
  - nessuno dei suoi nodi antenati è connesso ad una scatola di riparazione.

  Si tratta dei moduli più primitivi, illustrati nel Cap. 5; la prima condizione verifica che si tratti di sottoalberi indipendenti, la seconda che non ci siano riparazioni, la terza che non ci siano dipendenze dovute a scatole di riparazione collegate a livello superiore.

- **MSC massimi** (**MSC_{max}**): moduli di tipo MSC che non sono contenuti all'interno di un altro MSC;

- **MSS** *(**M**odulo a **S**oluzione nello **S**pazio degli stati)*: viene risolto ricorrendo alle Reti di Petri e/o alle Catene di Markov e ha le seguenti proprietà:

  - non contiene nessun evento condiviso con altri sottoalberi;
  - contiene almeno un evento collegato ad una scatola di riparazione;
  - nessuno dei suoi nodi antenati è connesso ad una scatola di riparazione.

  Si tratta di moduli riparabili e in quanto tali devono essere indipendenti (prima condizione), devono contenere degli eventi riparabili (seconda condizione) e non devono avere dipendenze date da riparazioni a livello superiore.

  Non è detto che tutti gli eventi contenuti in un MSS siano riparabili, dipende a cosa sono collegate le scatole di riparazione: se una scatola è connessa all'evento alla radice del modulo, tutti gli eventi al di sotto si potranno riparare oppure ci potrebbe essere solo un sottoinsieme di questi che si può riparare nel caso in cui la scatola di riparazione sia collegata ad un evento diverso da quello alla radice.

- **MSS minimi** (**MSS_{min}**): MSS che non contengono al loro interno altri moduli di qualunque natura.

In questa classificazione non si tiene conto del fatto che gli eventi di base possono essere considerati per definizione moduli; questo perchè sarebbe inutile sostituire un evento di base con un altro identico e inoltre non ci sarebbero MSC_{max} al di fuori degli eventi di base.

Quindi in questa classificazione i moduli non possono corrispondere ad eventi di base, ma solo ad eventi interni.

7.3 L'algoritmo di analisi per moduli

Supponendo di essere in grado di rilevare i moduli e di definirne la natura secondo le categorie elencate sopra e indicando con S_{comb} e S_{state} rispettivamente l'insieme dei MSC_{max} e l'insieme dei MSS_{min}, l'algoritmo per l'analisi per moduli [8] di un albero dei guasti con scatole di riparazione consiste dei seguenti passi:

1. Determina gli insiemi S_{comb} e S_{state}
2. Se ($S_{state} \neq \emptyset$) allora risolvi ogni $MSS \in S_{state}$ e sostituiscilo con l'equivalente evento di base
3. Determina gli insiemi S_{comb} e S_{state}
4. Se ($S_{comb} \neq \emptyset$) allora risolvi ogni $MSC \in S_{comb}$ e sostituiscilo con l'equivalente evento di base
5. Se l'albero non è ridotto ad un unico evento di base allora vai a 1.
6. FINE.

L'algoritmo opera esclusivamente su MSC_{max} e MSS_{min}; questo perché la soluzione di tipo combinatoria è più semplice di quella nello spazio degli stati e quindi si cerca di fare analisi nello spazio degli stati sui moduli più piccoli che la richiedono (MSS minimi) e utilizzare l'analisi combinatoria (analisi qualitativa e quantitativa) sui moduli di tipo MSC più grandi (MSC massimi).

I primi moduli ad essere esaminati sono i MSS_{min} dato che devono essere risolti per forza a parte; una volta sostituiti da un evento di base con l'appropriata probabilità di guasto, l'individuazione dei moduli va rifatta dato che la struttura dell'albero è adesso cambiata.

Dopodiché i MSC massimi subiscono lo stesso trattamento; tutto questo viene ripetuto finché l'intero albero non è stato ridotto ad un unico evento di base, cioè è stato risolto. Questo può avvenire in due situaazioni:

- l'albero è stato ridotto ad un unico MSC_{max}, dato che tutti i moduli riparabili sono stati precedentemente sostituiti, quindi non ci sono altre parti riparabili da considerare; in questo caso il modulo rimanente viene risolto con la tecnica di tipo combinatorio.
- L'albero è un unico MSS_{min} che viene risolto nello spazio degli stati; questo accade ad esempio se il *Top Event* dell'albero di partenza è collegato ad una scatola di riparazione.

7.3.1 L'esecuzione dell'algoritmo

Prendiamo come esempio l'albero di Fig. 6.2; da quanto descritto nel capitolo precedente i moduli risultano g_6, g_1 e TE, gli ultimi due riparabili.

Stabiliamo con più precisione la natura di questi moduli:

- g_6 è indipendente, non discende da nodi collegati a scatole di riparazione e non ne contiene e viene quindi classificato come MSC; inoltre non è contenuto all'interno di un altro MSC, quindi si tratta di un MSC_{max}.
- g_1 non ha nodi terminali che appaiono in altre parti dell'albero e contiene al suo interno due eventi collegati a scatole di riparazione; è quindi un MSS, ma non minimo, in quanto al suo interno si trova il modulo g_6.
- TE è per definizione modulo, contiene eventi riparabili e altri moduli; si tratta di un MSS.

Applichiamo l'algoritmo: non ci sono al momento MSS_{min} quindi si stacca, si risolve e si sostituisce con un evento di base l'unico MSC_{max}, cioè g_6; l'effetto sull'albero è rappresentato in Fig. 7.1.

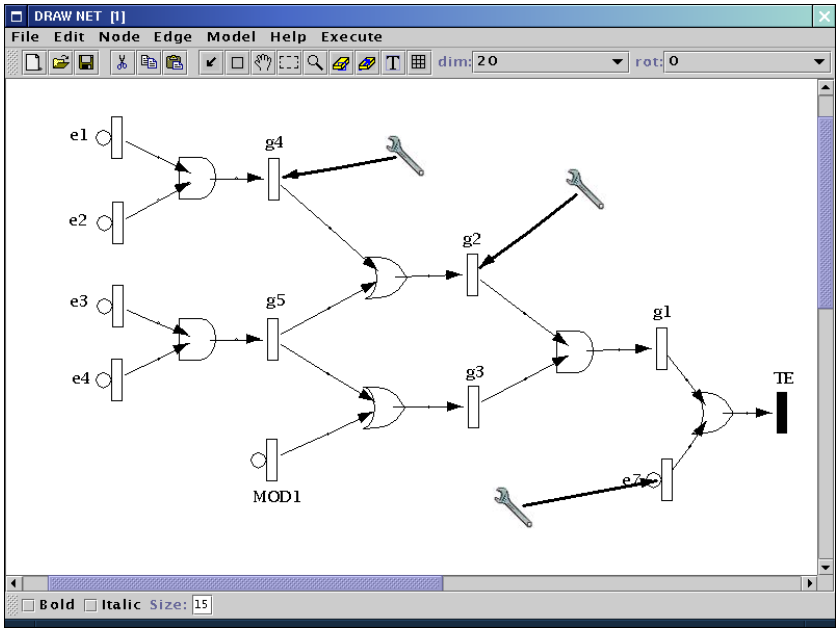


Figura 7.1: Sostituzione del modulo g_6 con MOD_1

Ora si determinano nuovamente i moduli; essi sono: g_1 e TE.

g_1 è un MSS_{min} visto che è indipendente, contiene eventi riparabili e non contiene altri moduli; per TE valgono le stesse considerazione fatte al passo precedente.

g_1 viene risolto a parte e sostituito dall'evento di base MOD_2 come in Fig. 7.2.

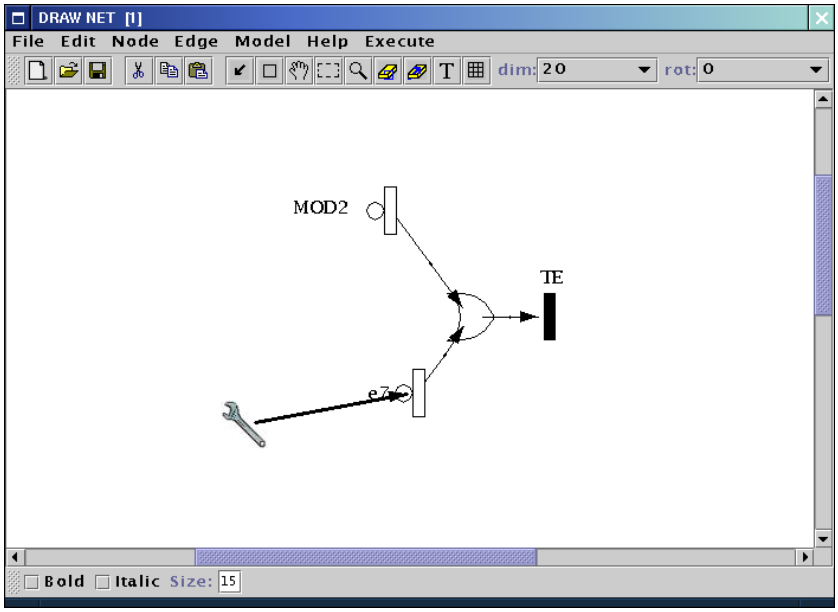


Figura 7.2: Sostituzione del modulo g_1 con MOD_2

L'albero è ora ridotto ad una porta logica a cui sono collegati due eventi di base, uno dei quali, e_7, è connesso ad una scatola di riparazione.

Tenendo conto che in questo algoritmo gli eventi di base non sono trattati come moduli, risulta come unico modulo TE, di tipo MSS_{min}.

L'albero si trova nello stato di massima semplificazione e quindi viene direttamente risolto, ottenendo la soluzione finale; se e_7 non fosse riparabile, TE sarebbe stato un MSC_{max} e risolto come tale.

7.4 Implementazione

Si è implementato il programma *RFTproc* che riceve come input un file XML generato da DrawNet che rappresenta l'albero con eventi riparabili, ne determina i moduli e li classifica, li analizza e risolve l'albero.

L'albero riparabile viene quindi disegnato con DrawNet, se ne genera il file XML con il comando **Solve** e si lancia *RFTproc* con indicazione del file

XML relativo all'albero da esaminare; lanciando *RFTproc* vengono eseguite le seguenti operazioni:

- parsing del file XML
- determinazione della struttura dell'albero (linking)
- eventuale unfolding
- individuazione dei moduli e della loro natura
- analisi per moduli

Durante l'esecuzione di *RFTproc* vengono richiamati altri strumenti:

- PFT2SWN (Traduttore di PFT in Reti di Petri colorate, realizzato da Claudio Bertoncello per il DISTA [15])
- l'analizzatore di Reti di Petri [18]
- il traduttore da DrawNet a Sharpe (illustrato nel Cap. 4)
- Sharpe [4].

Il programma restituisce come output la probabilità di guasto del sistema rappresentato dall'albero disegnato con DrawNet al tempo t; questo valore deve essere esplicitato dall'utente all'inizio dell'esecuzione, la quale avviene con il comando

```
RFTproc <nome file>
```

Il nome deve essere quello del file generato da DrawNet senza l'estensione **.xml**.

7.4.1 Parsing

Il parser di *RFTproc* è lo stesso del traduttore da XML a Sharpe predisposto anche per la lettura delle righe che descrivono le scatole di riparazione e gli appositi archi per il loro collegamento agli eventi, contraddistinte rispettivamente dai tag **Repair** e **Wrench**, secondo quanto descritto nel Cap. 6.

Scatole e relativi archi sono memorizzati in apposite strutture dati; per le scatole di riparazione i campi sono:

- **nome**: assegnato da DrawNet; identifica univocamente la scatola di riparazione;

- **rate**: tasso di riparazione; il valore è quello indicato dall'utente in DrawNet e corrisponde all'inverso del tempo medio per effettuare la riparazione;

- **oggetto**: puntatore all'evento a cui la scatola è collegata tramite l'apposito arco (oggetto della riparazione);

- **tipo**: tipo dell'evento oggetto della riparazione (di base o interno).

Gli archi specifici per le scatole di riparazione sono identificati dai campi **From** e **To** come gli archi che collegano le porte logiche agli eventi; **From** indica il nome della scatola, **To** quello dell'evento.

7.4.2 Linking

Anche la fase di *linking* è la stessa del traduttore da DrawNet a Sharpe con qualche aggiunta riguardante ovviamente le scatole di riparazione; infatti dopo aver ricreato in memoria la struttura dell'albero in base agli archi ordinari, vengono presi in considerazione quelli per le scatole di riparazione.

Per ognuno di essi si va a vedere nel campo **To** qual è il nome dell'evento oggetto della riparazione, quindi lo si ricerca nei vettori degli eventi (di base o interni) ed una volta trovato, l'evento viene qualificato come riparabile impostandone al valore 1 un apposito campo di nome *ripara*; successivamente vengono assegnati i campi **oggetto** e **tipo** relativi al riparatore indicato dal nome contenuto nel campo **From** dell'arco in esame. Infine si collega l'evento alla scatola con un puntatore all'elemento opportuno del vettore dei riparatori.

I campi **From** e **To** non possono contenere nomi di eventi o di scatole che non esistono, in quanto From e To vengono specificati direttamente da DrawNet quando si disegna un arco con i nomi esatti; se poi l'utente cancella uno degli oggetti alle estremità dell'arco, DrawNet eliminerà anche quest'ultimo.

Quindi quando si fa la ricerca di un evento con il nome indicato nel campo **To** nell'apposito vettore, l'evento viene sicuramente trovato.

Fatto ciò per tutti gli archi di riparazione, si possono distinguere gli eventi riparabili dal resto degli eventi.

7.4.3 Unfolding

La presenza anche qui della procedura di *unfolding* permette di avere dei PFT riparabili; in realtà l'analisi per moduli non avviene sull'albero parametrico, ma sulla sua corrispondente versione senza parametri dopo l'*unfolding*.

Se in questa fase vengono duplicati degli eventi riparabili, le relative copie saranno anch'esse riparabili secondo gli stessi tassi dell'originale parametrico.

7.4.4 Individuazione dei moduli

Per trovare i moduli si fa ricorso alla procedura descritta nei Cap. 5 e 6 che garantisce la condizione di indipendenza sia per gli eventi condivisi sia per la presenza di scatole di riparazione connesse a nodi a livello superiore; una volta individuati, i moduli devono anche essere classificati.

Per fare questo viene fatta una visita del modulo durante la quale vengono contate le scatole di riparazione collegate ad eventi e si determina il numero di moduli contenuti all'interno del modulo in questione.

Se il modulo contiene almeno un evento riparabile, esso è classificato come MSS; se poi non contiene al suo interno altri moduli, sara un MSS_{min}.

Se invece il modulo non contiene nessun evento riparabile esso sarà classificato come MSC_{max}; quindi si ricercano all'interno del modulo dei moduli precedentemente classificati come MCS_{max}; in caso ve ne siano presenti, essi vengono riclassificati come MCS (non massimi).

7.4.5 Analisi per moduli

Una volta individuati i moduli si deve procedere alla loro analisi specifica. RFTproc è al momento in grado di avviare l'analisi solo di MSC (tramite Sharpe) e di MSS_{min} con una sola scatola di riparazione e collegata alla radice del modulo; questo tipo di modulo sarà indicato con la sigla MSS_{top}.

Per questo anche l'algoritmo per la soluzione per moduli è stato implementato in una versione modificata:

1. Determina i moduli e la loro natura
2. Se non ci sono MSS di qualunque genere allora risolvi l'albero con Sharpe e vai a 6.
3. Se non ci sono MSS_{top} allora visualizza messaggio di errore e vai a 6.
4. Se c'è un unico MSS_{top} e questo è il *Top Event* analizza il modulo e vai a 6.

5. Se ci sono dei moduli di tipo MSS_{top} allora risolvili, sostituiscili con l'evento di base corrispondente e vai a 1.

6. FINE.

La Fig. 7.3 mostra il diagramma di flusso di tale algoritmo.

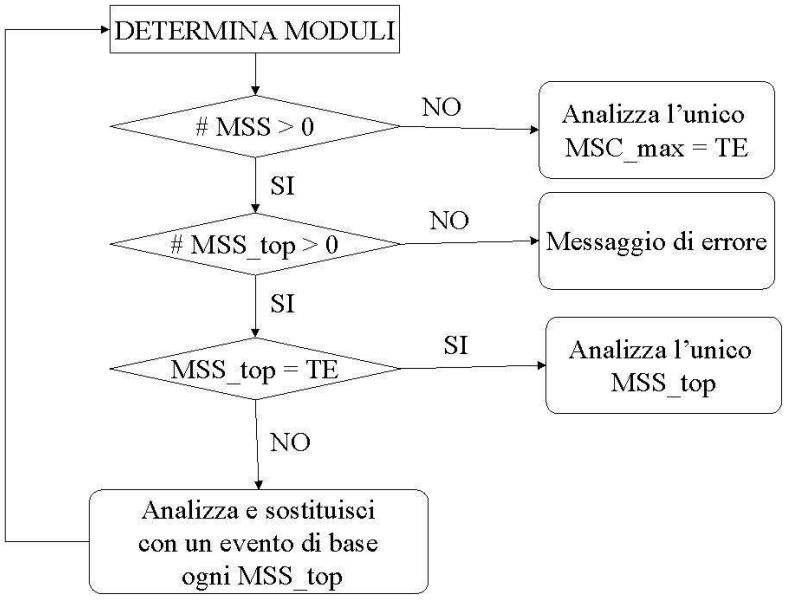


Figura 7.3: Schema generale dell'analisi per moduli implementata

Questa versione rispetto alla precedente, risolve e sostituisce i moduli MSS_{top} finché ce ne sono; supponendo che l'albero non contenga moduli riparabili non supportati, una volta che tutti i MSS_{top} sono stati sostituiti, l'albero si riduce per forza ad un unico modulo di tipo MSC_{max} che viene risolto con Sharpe come un normale albero dei guasti.

La soluzione dei moduli MSS_{top} avviene in questo modo (Fig. 7.4):

1. Viene generato il file XML che descrive il modulo come un albero indipendente in cui l'evento alla radice diventa il Top Event; la scatola di riparazione invece non viene indicata.

   La procedura per la generazione del codice XML parte dall'evento alla radice del modulo e genera la riga che descrive il *Top Event*; dopodiché, genera il codice relativo alla porta logica associata e all'arco di collegamento; quindi per ogni input della porta si genera il codice relativo ad

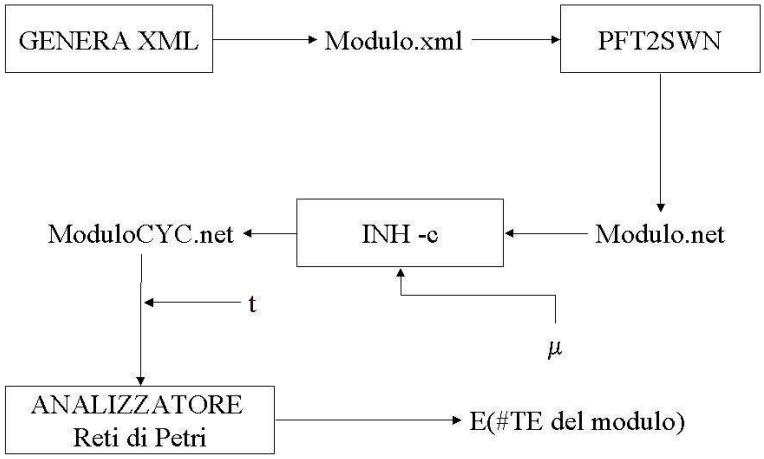


Figura 7.4: Passaggi nell'analisi dei moduli MSS_{top}

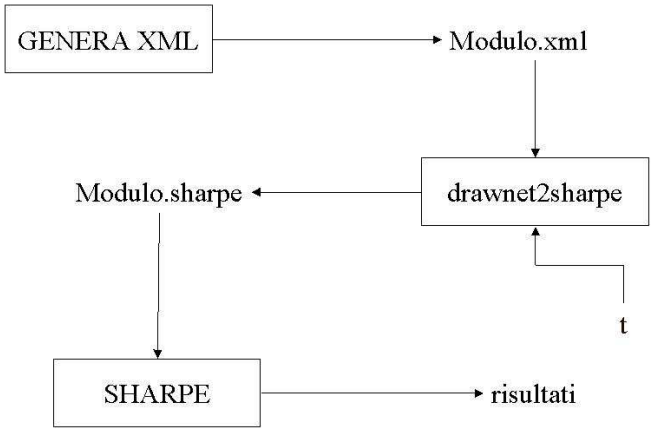


Figura 7.5: Passaggi nell'analisi dei moduli MSC_{max}

un evento interno o di base e all'arco che lo collega alla porta; a questo punto la procedura viene ripetuta ricorsivamente per ogni evento di input.

Il file XML generato rispecchia il formalismo di DrawNet prima dell'introduzione delle scatole di riparazione, assumendo che la riparazione riguardi solo il *Top Event* del modulo; tale formalismo deve essere rispettato anche per eseguire correttamente il prossimo strumento utilizzato.

2. Il file XML che contiene il modulo viene passato a PFT2SWN che genera la rete di Petri corrispondente e la salva in un apposito file secondo un determinato formalismo di rappresentazione (quello di GreatSPN); in realtà PFT2SWN è un traduttore di alberi dei guasti in forma parametrica (PFT) in Reti di Petri colorate (SWN). Nel modo in cui viene qui utitilizzato esso genererà una rete di Petri normale dato che l'albero non è in forma parametrica perché ha subito in precedenza l'operazione di *unfolding*.

3. La rete di Petri generata da PFT2SWN viene ora passata a INH [15] che inserisce un arco inibitore dal posto rappresentante il *Top Event* verso ogni transizione della rete e una transizione speciale che rende ciclica la rete; quindi aggiorna il file che contiene la rappresentazione della rete. In questo modo viene rappresentato sotto forma di rete di Petri il modulo e la sua riparazione.

   La transizione speciale ha l'effetto di riportare tutta la rete allo stato iniziale eliminando tutti i token presenti; viene inserita da INH ha come tasso di transizione (valore inverso del tempo medio necessario per scattare) il valore di default 1.0; si è reso quindi necessario andare a modificare il codice di INH in modo tale che il tasso di transizione della transizione speciale nella rete di Petri generata corrisponda al tasso di riparazione della relativa scatola di riparazione .

   Per fare questo, prima di lanciare PFT2SWN e INH, si risale dall'evento riparabile alla sua scatola di riparazione tramite il puntatatore di collegamento e si individua il tasso di riparazione corrispondente contenuto nel campo **rate**.

   INH è stato modificato in modo tale che all'avvio riceva come parametro il tasso di riparazione (oltre al nome del file che contiene la rete) e lo assegni al tasso della transizione che rende la rete ciclica.

4. Una volta eseguito questo programma la rete che rappresenta il modulo riparabile viene analizzata per avere una probabilità di guasto ad un

certo tempo t; a questo scopo, viene lanciato l'analizzatore di reti di Petri a cui viene indicato di calcolare il numero medio di token presenti nel posto al tempo che rappresenta il *Top Event* della rete al tempo t; questo indice viene considerato come la probabilità di guasto del modulo in quanto nel posto relativo al *Top Event* il numero di token presenti può essere 0 o 1 e quindi il numero medio di token presenti equivale alla probabilità che ce ne sia uno; dopo aver calcolato questo valore, l'analizzatore di reti di Petri lo salva in un file specifico.

5. Una volta calcolata la probabilità di guasto del modulo MSS_{top}, questo deve essere sostituito da un apposito evento di base; per fare ciò viene caricata dal file dei risultati la probabilità di guasto del modulo, il modulo viene completamente staccato dall'albero e al suo posto viene attaccato un evento di base creato con probabilità di guasto costante e pari a quella caricata.

   L'operazione di distacco-attacco viene fatta sostituendo nella struttura dati della porta logica al di sopra del modulo il puntatore all'evento alla radice del modulo con il puntatore all'evento di base che lo rimpiazza.

Questa procedura viene applicata a tutti i moduli di tipo MSS_{top}; dopodiché non ci sono più parti dell'albero da risolvere nello spazio degli stati e l'albero così semplificato è un modulo MSC_{max}.

Ora per l'albero semplificato viene generata la corrispondente rappresentazione XML sempre rispettando il solito formalismo e viene lanciato il traduttore da DrawNet a Sharpe (Cap. 4) che genera il file per Sharpe corrispondente all'albero semplificato con l'indicazione di calcolarne la probabilità di guasto ad un certo tempo t (Fig. 7.5) che è lo stesso per cui erano state valutate le Reti di Petri riguardanti i moduli MSS_{top}.

Il traduttore da DrawNet a Sharpe è stato adattato per ricevere tra i parametri anche il tempo a cui calcolare la probabilità di guasto, da indicare nel file per Sharpe; inoltre questa versione del traduttore non prevede l'indicazione del nome del file di input e del nome di quello di output, ma richiede solo il nome del file XML che desrive il modulo senza l'estensione **.xml** e genera il file per Sharpe con lo stesso nome del file di input anche se con l'estensione .sharpe.

Sharpe viene eseguito sul file appena generato e il risultato restituito non è solo la probabilità di guasto al tempo t dell'albero semplificato, ma anche quella dell'albero di partenza.

Un caso particolare si ha quando vi è una solo scatola di riparazione e questa è collegata al *Top Event* dell'albero di partenza; in questa situazione

l'intero albero è un MSS_{top} e viene completamente tradotto in rete di Petri e analizzato come tale.

Il risultato è la probabilità di guasto del sistema e la procedura di analisi per moduli termina così; dunque non viene utilizzato Sharpe in quanto l'albero non si semplifica a un MSC massimo.

7.4.6 Tipo di riparazione

L'utilizzo degli strumenti PFT2SWN e INH per i moduli riparabili permette soltanto di analizzare i MSS_{top} dato che è possibile generare una rete che contiene un solo ciclo (una sola riparazione) e che parte dal posto rappresentante il *Top Event* (riparazione del Top Event).

Il tipo di riparazione che viene così modellato prevede che quando il guasto si propaga alla radice del modulo, la riparazione si attivi riportando allo stato funzionante tutti i componenti del modulo.

La presenza degli archi inibitori dal posto rappresentante il *Top Event* del modulo a tutte le transizioni corrispondenti ai vari eventi, impedisce che questi si guastino dopo che il guasto si è già propagato al Top Event.

In questo caso è come se il funzionamento dei componenti del modulo ancora operativi venisse bloccato in attesa che la riparazione abbia effetto per poi rimettere in attività l'intero modulo. Quindi al guasto del Top Event e durante la riparazione non ci possono essere altri guasti di altri componenti o sottosistemi.

7.4.7 La traduzione in Reti di Petri

La Fig. 7.6 indica come viene tradotto in rete di Petri un evento di base, mentre la Fig. 7.6 mostra le reti equivalenti alle porte logiche [14] [13].

- L'evento di base viene tradotto da una transizione che rappresenta l'avvenimento del guasto ed un posto che indica lo stato di guasto del componente.

  La transizione è temporizzata in base al parametro λ della distribuzione dell'evento di base (Fig. 7.6); l'arco inibitore impedisce che la transizione avvenga quando c'è già un token nel posto, quindi impedisce che il componente si guasti quando è già in questo stato.

- La porta logica **or** (Fig. 7.7 (a)) diventa un insieme di transizioni immediate, una per ogni evento di input; ognuna mette un token in un posto che indica l'evento di guasto di output.

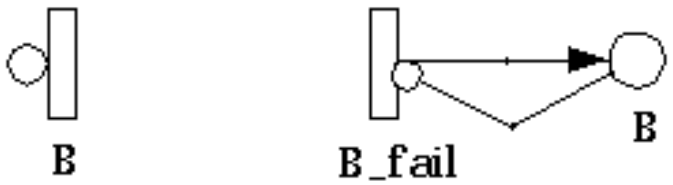


Figura 7.6: Un evento di base nella forma di rete di Petri

In questo modo il token può comparire nel posto di output tramite una qualunque delle transizioni, la prima che scatta, e basta che ne scatti una. A livello logico il guasto si propaga attraverso la porta **or** se almeno uno degli input è guasto.

Gli archi inibitori impediscono che un token venga trasferito nel posto di output quando questo ne contiene già uno.

- La porta logica **and** (Fig. 7.7 (b)) viene tradotta in rete di Petri come un'unica transizione che, quando c'è un token in ogni posto rappresentante il guasto di un componente fa apparire un token nel posto che indica il guasto generale.

  Perché la transizione scatti in tutti i posti degli eventi di input deve essere presente il token così come la porta logica **and** consente la propagazione del guasto se tutti gli eventi di input sono in questo stato.

  L'arco inibitore ha la stessa funzione che ha nella traduzione in rete di Petri della porta logica or, solo che riguarda la sola transizione.

- Per quanto riguarda la porta logica **K out of N**, questa viene rappresentata in rete di Petri come la combinazione delle reti relative alle porte **and** e **or**.

7.4.8 Caso di esempio

La Fig. 7.8 mostra l'albero su cui verrà eseguito *RFTproc* con $t = 5000$; il listato XML corrispondente è il seguente:

```
<?xml  version='1.0' encoding='UTF-8' standalone='no'?>
```

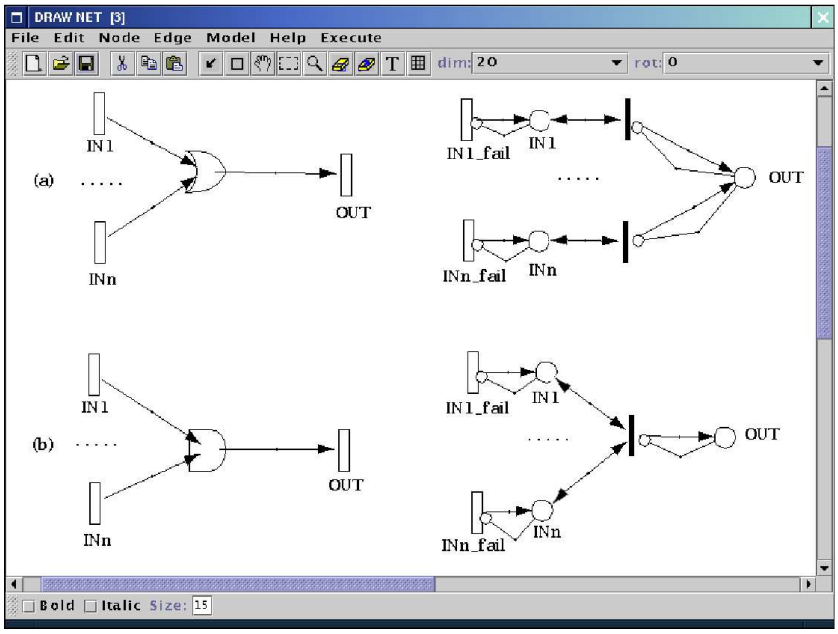


Figura 7.7: Traduzione delle porte logiche in Reti di Petri

```
<PFT name='esempio_moduli' visibility='false' Title=''>
   <TopEvent name='TopEvent0_1' visibility='false' Label='TE'/>
   <Event name='Event1_1' visibility='false'
    Label='g1' Parameters=''/>
   <BasicEvent name='BasicEvent2_1' visibility='false'
    Distribution='ALL EXP 0.00007' Label='e7' Parameters=''/>
   <And name='And3_1' visibility='false'/>
   <Event name='Event4_1' visibility='false' Label='g2'
    Parameters=''/>
   <Event name='Event5_1' visibility='false' Label='g3'
    Parameters=''/>
   <Or name='Or6_1' visibility='false'/>
   <Or name='Or7_1' visibility='false'/>
   <Event name='Event8_1' visibility='false' Label='g5'
    Parameters=''/>
   <Event name='Event9_1' visibility='false' Label='g4'
    Parameters=''/>
   <Event name='Event10_1' visibility='false' Label='g6'
    Parameters=''/>
   <And name='And11_1' visibility='false'/>
```

```
<And name='And12_1' visibility='false'/>
<And name='And13_1' visibility='false'/>
<BasicEvent name='BasicEvent14_1' visibility='false'
 Distribution='ALL EXP 0.00001' Label='e1' Parameters=''/>
<BasicEvent name='BasicEvent15_1' visibility='false'
 Distribution='ALL EXP 0.00002' Label='e2' Parameters=''/>
<BasicEvent name='BasicEvent16_1' visibility='false'
 Distribution='ALL EXP 0.00003' Label='e3' Parameters=''/>
<BasicEvent name='BasicEvent17_1' visibility='false'
 Distribution='ALL EXP 0.00004' Label='e4' Parameters=''/>
<BasicEvent name='BasicEvent18_1' visibility='false'
 Distribution='ALL EXP 0.00005' Label='e5' Parameters=''/>
<BasicEvent name='BasicEvent19_1' visibility='false'
 Distribution='ALL EXP 0.00006' Label='e_6' Parameters=''/>
<Or name='Or20_1' visibility='false'/>
<Arc name='Arc0_1' visibility='false' from='BasicEvent14_1'
 to='And11_1'/>
<Arc name='Arc1_1' visibility='false' from='BasicEvent15_1'
 to='And11_1'/>
<Arc name='Arc2_1' visibility='false' from='And11_1'
 to='Event9_1'/>
<Arc name='Arc3_1' visibility='false' from='BasicEvent16_1'
 to='And12_1'/>
<Arc name='Arc4_1' visibility='false' from='BasicEvent17_1'
 to='And12_1'/>
<Arc name='Arc5_1' visibility='false' from='And12_1'
 to='Event8_1'/>
<Arc name='Arc6_1' visibility='false' from='BasicEvent18_1'
 to='And13_1'/>
<Arc name='Arc7_1' visibility='false' from='BasicEvent19_1'
 to='And13_1'/>
<Arc name='Arc8_1' visibility='false' from='And13_1'
 to='Event10_1'/>
<Arc name='Arc9_1' visibility='false' from='Event9_1'
 to='Or6_1'/>
<Arc name='Arc11_1' visibility='false' from='Or6_1'
 to='Event4_1'/>
<Arc name='Arc12_1' visibility='false' from='Or7_1'
 to='Event5_1'/>
<Arc name='Arc13_1' visibility='false' from='Event8_1'
 to='Or7_1'/>
```

```
   <Arc name='Arc14_1' visibility='false' from='Event10_1'
    to='Or7_1'/>
   <Arc name='Arc15_1' visibility='false' from='Event4_1'
    to='And3_1'/>
   <Arc name='Arc16_1' visibility='false' from='Event5_1'
    to='And3_1'/>
   <Arc name='Arc17_1' visibility='false' from='And3_1'
    to='Event1_1'/>
   <Arc name='Arc18_1' visibility='false' from='Event1_1'
    to='Or20_1'/>
   <Arc name='Arc19_1' visibility='false' from='BasicEvent2_1'
    to='Or20_1'/>
   <Arc name='Arc21_1' visibility='false' from='Or20_1'
    to='TopEvent0_1'/>
   <Repair name='Repair7' visibility='false' Repair_Rate='0.025'/>
   <Wrench name='Wrench12' visibility='false' from='Repair7'
    to='Event9_1'/>
   <BasicEvent name='BasicEvent9' visibility='false'
    Distribution='ALL EXP 0.00008' Label='e8' Parameters=''/>
   <Arc name='Arc14' visibility='false' from='BasicEvent9' to='Or6_1'/>
   <Repair name='Repair10' visibility='false' Repair_Rate='0.0333'/>
   <Wrench name='Wrench15' visibility='false' from='Repair10'
    to='Event5_1'/>
</PFT>
```

La fase di *parsing* carica nelle apposite strutture dati eventi, porte logiche, archi e riparatori; segue il *linking* che ricostruisce la struttura dell'albero in memoria prima collegando le porte logiche agli eventi valutano gli archi di tipo **Arc** e poi collegando le scatole di riparazione ai relativi eventi dall'esame degli archi di tipo **Wrench**.

Ciò che risulta è un albero con eventi riparabili, in particolare i sottoalberi g_3 e g_4 a cui corrispondono rispettivamente i tassi di riparazione 0.0333 e 0.025.

Non si tratta di un PFT, quindi non è necessario l'*unfolding*; si passa allora all'individuazione dei moduli; l'algoritmo trova come MSS_{top} g_3 e g_4, mentre g_1, g_2 e TE sono MSS.

Infatti g_3 e g_4 sono indipendenti e hanno una scatola di riparazione collegata che genera una dipendenza sui relativi sottoalberi impedendo qui la presenza di moduli.

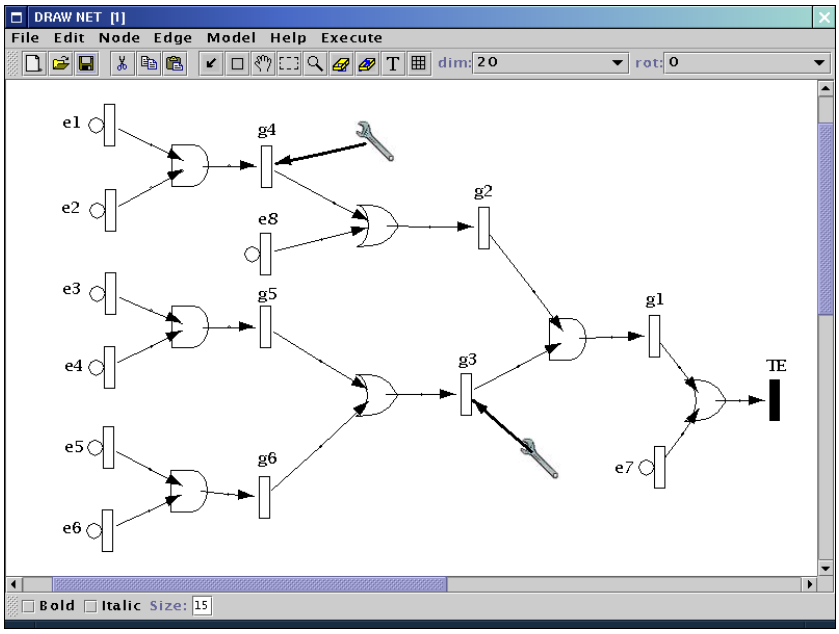


Figura 7.8: Albero sul quale eseguire RFTproc

Ora per ciascuno dei MSS_{top} viene generato il relativo file XML, questo viene passato a PFT2SWN che genera la rete di Petri corrispondente, INH inserisce la riparazione, la rete risultante viene analizzata e il risultato diviene la probabilità di guasto dell'evento di base che sostituirà il modulo.

Il listato XML per g_3 è il seguente:

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<PFT name='esempio_moduli_g3' visibility='false'
 Title='modulo_SSM_min_di_esempio_moduli'>
   <TopEvent name='TopEvent0' visibility='false' Label='TE'/>
   <Or name='Or1' visibility='false'/>
   <Arc name='Arc0' visibility='false' from='Or1' to='TopEvent0'/>
   <Event name='Event2' visibility='false' Label='g5' Parameters=''/>
   <And name='And3' visibility='false'/>
   <Arc name='Arc1' visibility='false' from='And3' to='Event2'/>
   <BasicEvent name='BasicEvent4' visibility='false'
    Distribution='ALL EXP 0.00003' Label='e3' Parameters=''/>
   <Arc name='Arc2' visibility='false' from='BasicEvent4' to='And3'/>
   <BasicEvent name='BasicEvent5' visibility='false'
    Distribution='ALL EXP 0.00004' Label='e4' Parameters=''/>
   <Arc name='Arc3' visibility='false' from='BasicEvent5' to='And3'/>
```

```
   <Arc name='Arc4' visibility='false' from='Event2' to='Or1'/>
   <Event name='Event6' visibility='false' Label='g6' Parameters=''/>
   <And name='And7' visibility='false'/>
   <Arc name='Arc5' visibility='false' from='And7' to='Event6'/>
   <BasicEvent name='BasicEvent8' visibility='false'
    Distribution='ALL EXP 0.00005' Label='e5' Parameters=''/>
   <Arc name='Arc6' visibility='false' from='BasicEvent8' to='And7'/>
   <BasicEvent name='BasicEvent9' visibility='false'
    Distribution='ALL EXP 0.00006' Label='e_6' Parameters=''/>
   <Arc name='Arc7' visibility='false' from='BasicEvent9' to='And7'/>
   <Arc name='Arc8' visibility='false' from='Event6' to='Or1'/>
</PFT>
```

Questo file viene preso come input da PFT2SWN che deriva la rete di Fig. 7.9; successivamente INH introduce la riparazione inserendo gli archi inibitori e rendendo la rete risultante ciclica come in Fig. 7.10 inserendo la transizione temporizzata secondo il tasso di riparazione del riparatore di g_3.

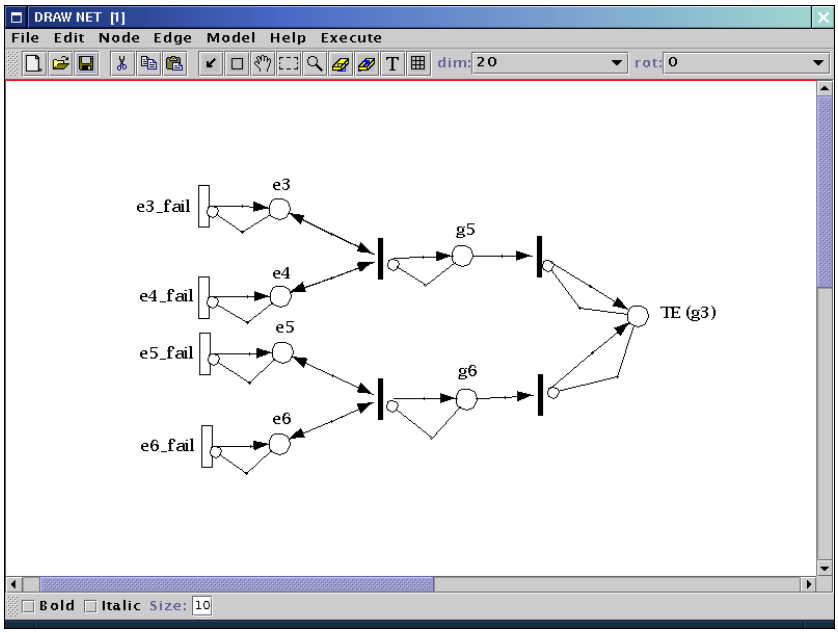


Figura 7.9: Traduzione del modulo g_3 in Reti di Petri

L'analizzatore di Reti di Petri calcola il numero medio di token nel posto relativo al *Top Event* per la rete di g_3 al tempo $t = 5000$, quindi rimpiazza il modulo con un evento di base che ha tale valore come probabilità di guasto (al tempo t), come in Fig. 7.11.

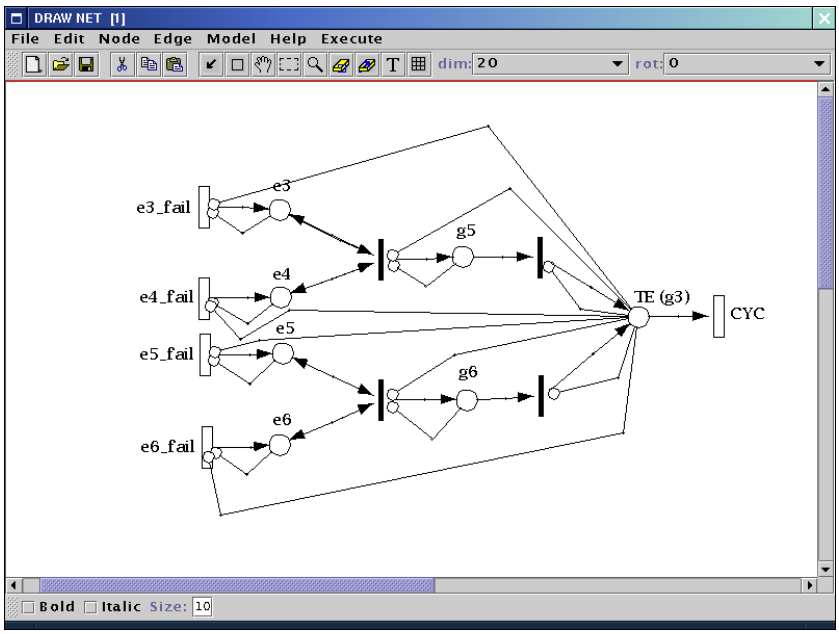


Figura 7.10: Inserimento della riparazione nella rete di g_3

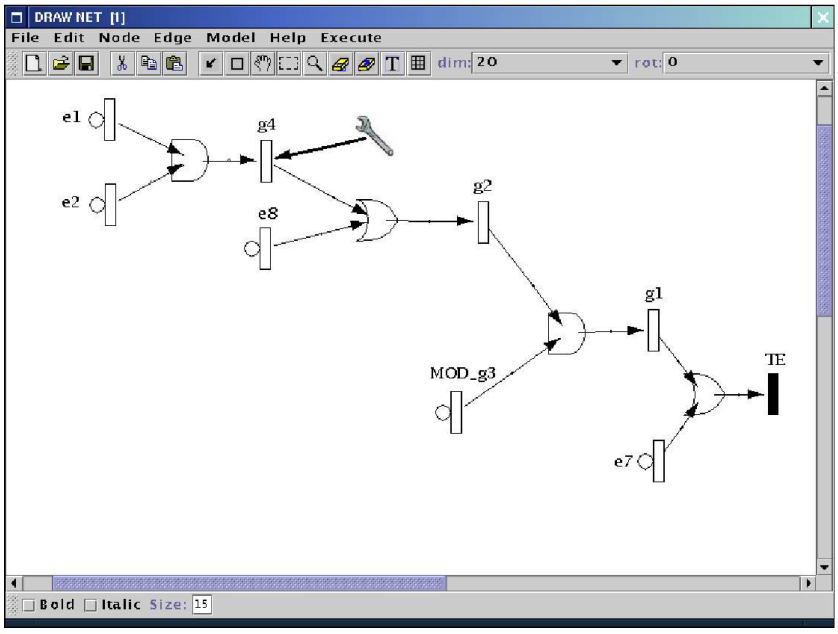


Figura 7.11: Sostituzione di g_3 con l'evento di base MOD_g_3

La stessa procedura viene ora applicata all'altro MSS_{top}, g_4, di cui segue il codice XML.

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<PFT name='esempio_moduli_g4' visibility='false'
 Title='modulo_SSM_min_di_esempio_moduli'>
   <TopEvent name='TopEvent0' visibility='false' Label='TE'/>
   <And name='And1' visibility='false'/>
   <Arc name='Arc0' visibility='false' from='And1' to='TopEvent0'/>
   <BasicEvent name='BasicEvent2' visibility='false'
    Distribution='ALL EXP 0.00001' Label='e1' Parameters=''/>
   <Arc name='Arc1' visibility='false' from='BasicEvent2' to='And1'/>
   <BasicEvent name='BasicEvent3' visibility='false'
    Distribution='ALL EXP 0.00002' Label='e2' Parameters=''/>
   <Arc name='Arc2' visibility='false' from='BasicEvent3' to='And1'/>
</PFT>
```

La rete di Petri corrispondente è quella di Fig.7.12 e in Fig. 7.13 g_4 viene sostituito dall'evento di base MOD_g_4 che ha probabilità ....

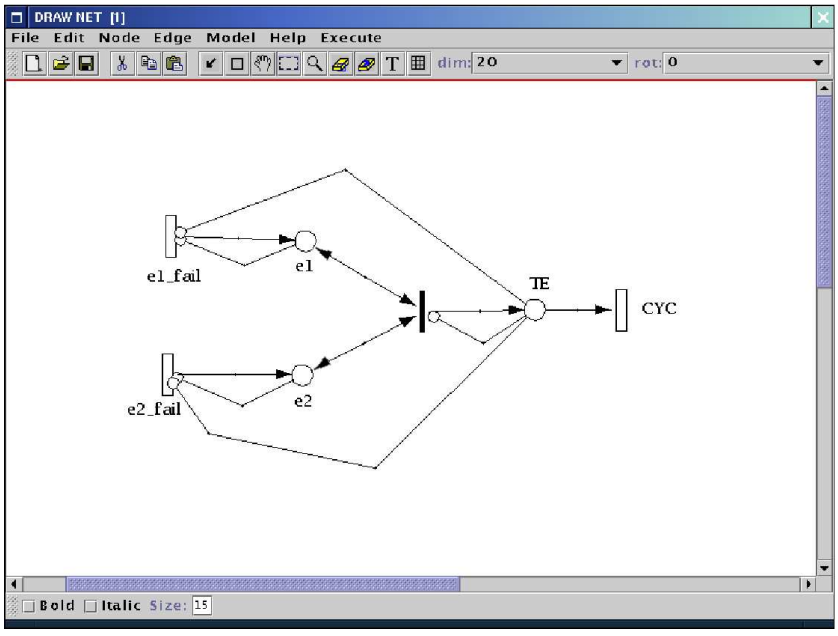


Figura 7.12: Rete di Petri ciclica relativa a g_4

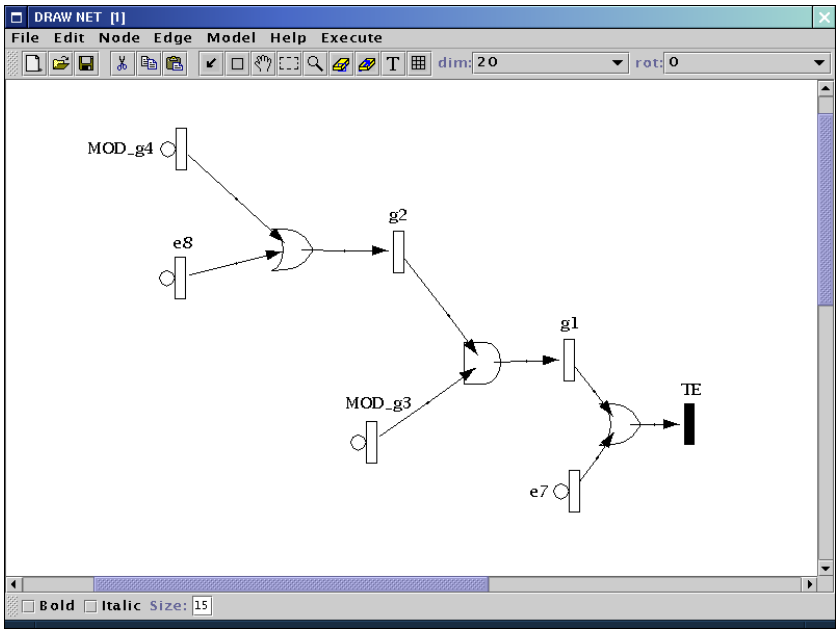


Figura 7.13: Sostituzione di g_4 con l'evento di base MOD_g_4

Sull'albero così semplificato vengono nuovamente individuati i moduli e non risultano MSS_{top}, ma un unico MSC_{max} corrispondente a TE; per analizzare questo tipo di modulo si fa ricorso a Sharpe.

Viene generato il file XML relativo a questo modulo, viene lanciato il traduttore da DrawNet a Sharpe che lo traduce nel formalismo di Sharpe, quindi viene eseguito quest'ultimo che restituisce la probabilità di guasto al tempo t ....

Il file XML che descrive l'albero semplificato ha questo contenuto:

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<PFT name='esempio_moduli' visibility='false'
 Title='modulo_CSM_MAX_di_esempio_moduli'>
   <TopEvent name='TopEvent0' visibility='false' Label='TE'/>
   <Or name='Or1' visibility='false'/>
   <Arc name='Arc0' visibility='false' from='Or1' to='TopEvent0'/>
   <Event name='Event2' visibility='false' Label='g1' Parameters=''/>
   <And name='And3' visibility='false'/>
   <Arc name='Arc1' visibility='false' from='And3' to='Event2'/>
   <Event name='Event4' visibility='false' Label='g2' Parameters=''/>
   <Or name='Or5' visibility='false'/>
   <Arc name='Arc2' visibility='false' from='Or5' to='Event4'/>
```

```
   <BasicEvent name='BasicEvent6' visibility='false'
    Distribution='ALL DET 0.000116' Label='MOD_g4' Parameters=''/>
   <Arc name='Arc3' visibility='false' from='BasicEvent6' to='Or5'/>
   <BasicEvent name='BasicEvent7' visibility='false'
    Distribution='ALL EXP 0.00008' Label='BasicEvent9' Parameters=''/>
   <Arc name='Arc4' visibility='false' from='BasicEvent7' to='Or5'/>
   <Arc name='Arc5' visibility='false' from='Event4' to='And3'/>
   <BasicEvent name='BasicEvent8' visibility='false'
    Distribution='ALL DET 0.002702' Label='MOD_g3' Parameters=''/>
   <Arc name='Arc6' visibility='false' from='BasicEvent8' to='And3'/>
   <Arc name='Arc7' visibility='false' from='Event2' to='Or1'/>
   <BasicEvent name='BasicEvent9' visibility='false'
    Distribution='ALL EXP 0.00007' Label='e7' Parameters=''/>
   <Arc name='Arc8' visibility='false' from='BasicEvent9' to='Or1'/>
</PFT>
```

Il traduttore verso Sharpe produce questo file:

```
* modulo_CSM_MAX_di_esempio_moduli

ftree esempio_moduli_TOP

repeat MOD_g4 prob(0.000116)
repeat e8 exp(v2)
repeat MOD_g3 gen prob(0.002702)
repeat e7 exp(v3)

or g2 MOD_g4 e8
and g1 g2 MOD_g3
or TOP g1 e7
end

bind
v2 0.00008
v3 0.00007
end

eval(esempio_moduli) 5000 5000 5000
end
```

In questo file, gli eventi di base corrispondenti ai moduli sostituiti sono dichiarati con probabilità costante e quindi indipendente dal tempo; questo

perchè tale valore è già stato calcolato rispetto a un tempo, in questo caso 5000, a differenza degli altri eventi che sono dichiarati di distribuzione esponenziale, la cui probabilità di guasto deve essere ancora valutata rispetto al tempo.

L'esecuzione di Sharpe su questo file porta al risultato:

```
system mod_TOP
    t          F(t)

5.0000 e+03  2.9594 e-01
```

Capitolo 8

Nuovi tipi di porte logiche

8.1 Introduzione

La costruzione di alberi dei guasti è avvenuta finora usando delle porte logiche elementari quali *and*, *or* e *k:n*; queste possono non essere sufficienti quando si vuole modellare delle situazioni particolari in cui la propagazione del guasto avviene non solo per una combinazione di eventi, ma anche attraverso altre condizioni quali

- l'ordine temporale degli eventi
- la presenza di componenti che cominciano ad operare dopo altri
- la presenza di componenti sostitutivi di altri
- la presenza di eventi di guasto che causano direttamente il guasto di altri componenti

Sono state ideate delle nuove porte logiche per modellare in modo diretto e graficamente semplice queste condizioni in un albero dei guasti; naturalmente la presenza di queste porte ha delle notevoli influenze sull'analisi dell'albero dato che non tutte possono essere valutate con le tecniche combinatorie; si rende quindi necessaria anche qui come per gli eventi riparabili un'analisi per moduli in cui si devono stabilire nuovi criteri per l'individuazione dei moduli e nuovi metodi di valutazione di questi; si deve poi tener conto dell'eventuale presenza di riparatori.

Le porte logiche si possono dividere in due categorie: **statiche** e **dinamiche**.

Quelle **statiche** sono risolvibili valutando le possibili combinazioni degli eventi di input; appartengono a questa classe le porte usate finora, la nuova porta $k : n_{cons}$ [16] (k su n consecutivi) e la porta **Xor** (or esclusivo).

Le porte dinamiche sono invece quelle che modellano l'ordine temporale degli eventi e particolari dipendenze tra questi; verranno esaminate alcune porte dinamiche, più precisamente le porte **P_And**, **Seq_And**, **Fdep**, **WSP**, **CSP** [10] [11]. ,

Un albero in cui sono presenti delle porte dinamiche viene detto *Albero dei guasti dinamico* (**DFT**), altrimenti si tratta di un albero statico.

8.2 La porta $k : n_{cons}$

8.2.1 Significato logico

Come già detto, il nome $k : n_{cons}$ [16] sta per k su n consecutivi e più precisamente questa porta logica propaga il guasto dagli eventi di input a quello di output se k eventi consecutivi tra gli n di input si guastano.

Per k eventi consecutivi si intende che questi devono essere adiacenti uno all'altro a livello grafico o in base ad un indice; ad esempio se gli eventi di input della porta $k : n_{cons}$ sono $X_1, X_2, \ldots, X_8$ e $k = 3$ il guasto si propagherà se si guastano X_1, X_2, X_3 oppure X_2, X_3, X_4 oppure X_3, X_4, X_5 ecc.

Più formalmente

$$Y = 3/8_{cons}(X_1 \ldots X_8) =$$
$$= (X_1 \wedge X_2 \wedge X_3) \vee (X_2 \wedge X_3 \wedge X_4) \vee (X_3 \wedge X_4 \wedge X_5) \vee$$
$$\vee (X_4 \wedge X_5 \wedge X_6) \vee (X_5 \wedge X_6 \wedge X_7) \vee (X_6 \wedge X_7 \wedge X_8)$$

Questa formula prende in esame tutte le possibili combinazioni di 3 eventi consecutivi tra 8; perché il guasto si propaghi se ne deve verificare almeno una, cioè tutti i tre elementi di almeno una delle congiunzioni si devono guastare.

Rispetto alla porta *k:n* generica, il numero delle combinazioni considerate è minore, dato che si devono valutare quelle di elementi consecutivi, anziché tutte le combinazioni di k elementi tra n.

Questa porta si usa di solito quando gli eventi di input riguardano componenti fisici uguali tra loro che per la loro natura si trovano uno adiacente all'altro; un esempio potrebbe essere una fila di lampioni che illumina una strada; il guasto di una lampadina qua e là non compromette gravemente l'illuminazione, mentre se un certo numero di lampadine consecutive si guastasse una parte della strada resterebbe al buio.

8.2.2 Rappresentazione in DrawNet

Per poter rappresentare questa porta logica in DrawNet si è dovuto modificare il formalismo per gli alberi dei guasti che DrawNet carica al suo avvio;

in particolare è stata aggiunta una riga per indicare la presenza della nuova porta logica:

```
<nodeType parent="KofN" name="K_out_N_cons"/>
```

I campi che descrivono la porta in DrawNet sono, oltre al nome identificativo assegnato da DrawNet, il valore k e il valore n che devono essere specificati dall'utente come nel caso della porta *k:n*

Per quanto riguarda il collegamento della porta con gli eventi, essa può avere eventi di input singoli o di tipo *Replicator*, purchè il loro numero complessivo rispetti il valore del parametro n.

Per rappresentare a livello grafico la porta $k : n_{cons}$ si è dovuta creare un'apposita icona da associare alla porta logica con *TableConfig* prima di avviare DrawNet con il nuovo formalismo.

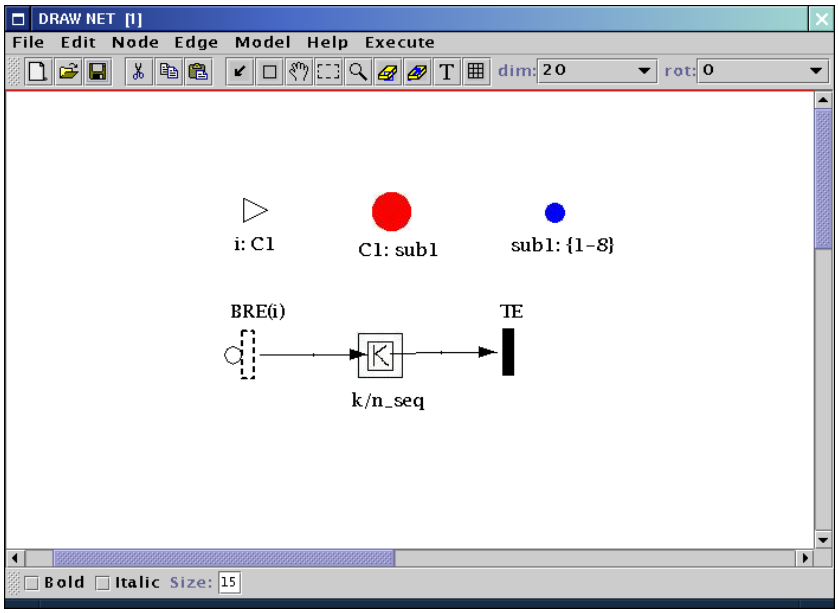


Figura 8.1: La porta $k : n_{cons}$

8.2.3 Implementazione

Sia il traduttore da DrawNet a Sharpe, sia il programma per l'analisi per moduli sono stati aggiornati in funzione della porta $k : n_{cons}$.

Per quanto riguarda la fase di *parsing* si è tenuto conto del tag *K_out_N* che indica la porta; essa viene memorizzata insieme alle altre porte caricando

anche i valori di k ed n; il *linking* semplicemente collega la porta agli eventi di input e a quello di output grazie all'esame degli appositi archi (generici).

Nella fase di *unfolding* viene duplicato l'evento replicato di input se questo è presente, secondo l'apposito parametro; quindi se si tratta di un evento di base, solo questo sarà duplicato, altrimenti tutto il sottoalbero.

A questo punto è stata introdotta una nuova procedura che modifica la struttura dell'albero trasformando la porta logica $k : n_{cons}$ in una combinazione logicamente equivalente di porte *and* e *or* secondo la formula precedente.

Ora che la porta $k : n_{cons}$ è stata sostituita, l'albero può essere trattato normalmente (tradotto per Sharpe, per Astra o analizzato per moduli).

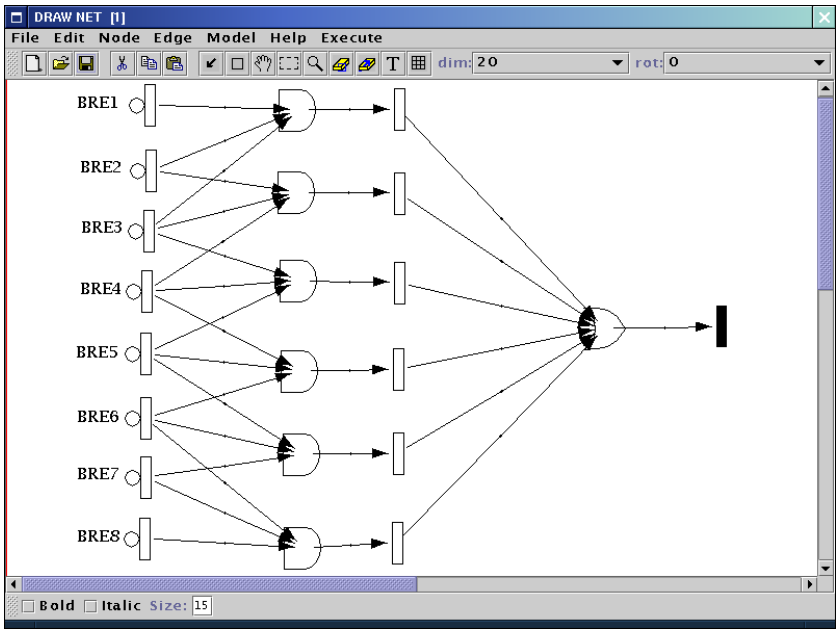


Figura 8.2: La porta di Fig. 8.1 espressa come combinazione di and e or

8.3 La porta Or Esclusivo

Or esclusivo (**Xor**) è una porta che, come *or*, ha un numero variabile di eventi di input; il guasto si propaga attraverso **Xor** se soltanto uno degli eventi di guasto si verifica; questo può essere qualsiasi, ma si deve verificare esclusivamente questo.

Sharpe non prevede questo tipo di porta, mentre lo fa Astra che durante l'analisi dell'albero la sostituisce con la corrispondente espressione logica:

$$Xor(A, B, C) = (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C)$$

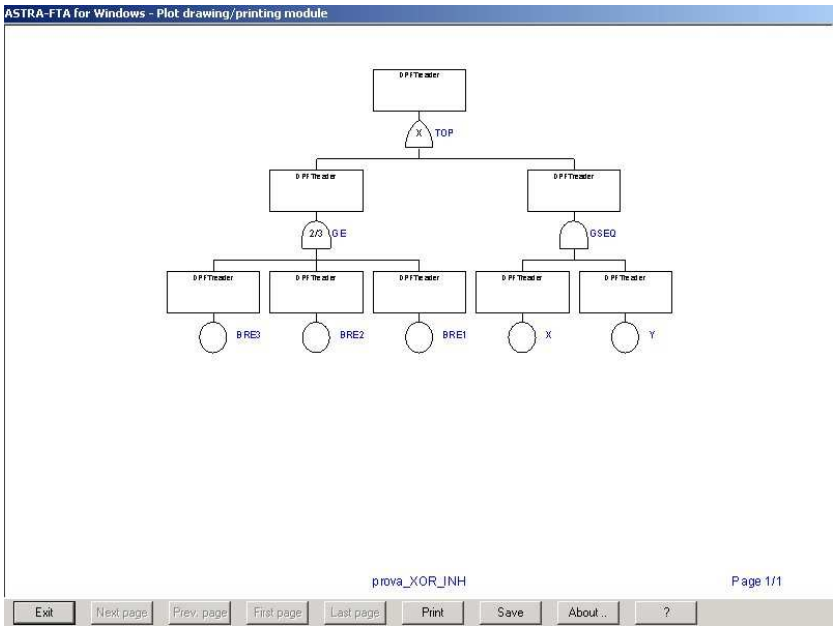


Figura 8.3: Esempio di applicazione di Xor in Astra

8.4 Porte logiche dinamiche: And prioritario

8.4.1 Significato logico

La porta logica *And prioritario* [10] [11] (indicata con **P_And**) ha due eventi di input i quali devono verificarsi entrambi perché il guasto si propaghi e in un certo ordine.

Se A e B sono gli input della porta, A e B si devono guastare e A deve farlo prima di B; entrambe le condizioni si devono verificare, altrimenti il guasto non si propaga.

8.4.2 Rappresentazione in DrawNet

Il formalismo per DrawNet è stato aggiornato aggiungendo la presenza di **P_And** tra le porte logiche e creando un nuovo tipo di arco di nome **Order** il

quale ha il compito di collegare alla porta i suoi eventi di input e di assegnare attraverso il campo **No.** un numero che indica l'ordine con il quale essi si devono verificare; questo numero deve essere assegnato dall'utente che disegna l'albero con DrawNet.

Le righe aggiunte al formalismo a tal proposito sono le seguenti:

```
<nodeType parent="" name="PAND"/>

<edgeType parent="" name="Order">
    <propertyType name="No." default="1"/>
    <constraint fromType="Event0" fromCardinality=""
     toType="PAND" toCardinality="2"/>
</edgeType>
```

Se A si deve verficare prima di B l'arco **Order** che collega A alla porta avrà $No. = 1$, mentre per B $No. = 2$; si è poi imposto il vincolo per cui la porta **P_And** deve avere due input; potrebbe anche avere un unico input se questo fosse un evento replicato con un parametro la cui cardinalità corrisponde a 2; l'evento di output corrispondente viene ovviamente collegato alla porta con un arco ordinario come per le altre porte logiche.

Anche per la porta **P_And** è stata creata un'apposita icona per la sua indicazione a livello grafico.

Eliminando il vincolo che impone alla porta due soli figli si potrebbe rappresentare la porta And sequenziale (Seq_And) che corrisponde ad un P_And, ma con più di 2 eventi di input, e che verrà trattata sucessivamente.

8.4.3 Traduzione nel Modello di Markov

Definendo la semantica degli alberi dei guasti dinamici nel *Modello di Markov* [10], gli stati rappresentano le combinazioni di eventi guasti o funzionanti, mentre le transizioni avvengono al guastarsi di uno dei componenti.

La traduzione nel Modello di Markov della porta logica **P_And** prevede uno stato di partenza indicato con 00 in cui sia A sia B sono funzionanti e quindi anche il sottosistema relativo alla porta.

Se B è il primo componente a guastarsi, si passa direttamente allo stato assorbente **Oper** che rappresenta l'operatività del sottosistema, dato che, perché ci sia il guasto generale, si deve per forza guastare prima A.

Se invece si guasta A, il modello passa allo stato 10 (A guasto, B non guasto), ma il sottosistema è ancora funzionante; se in seguito si guasta anche B allora vi è la transizione allo stato assorbente **Fail** che indica il guasto del sottosistema.

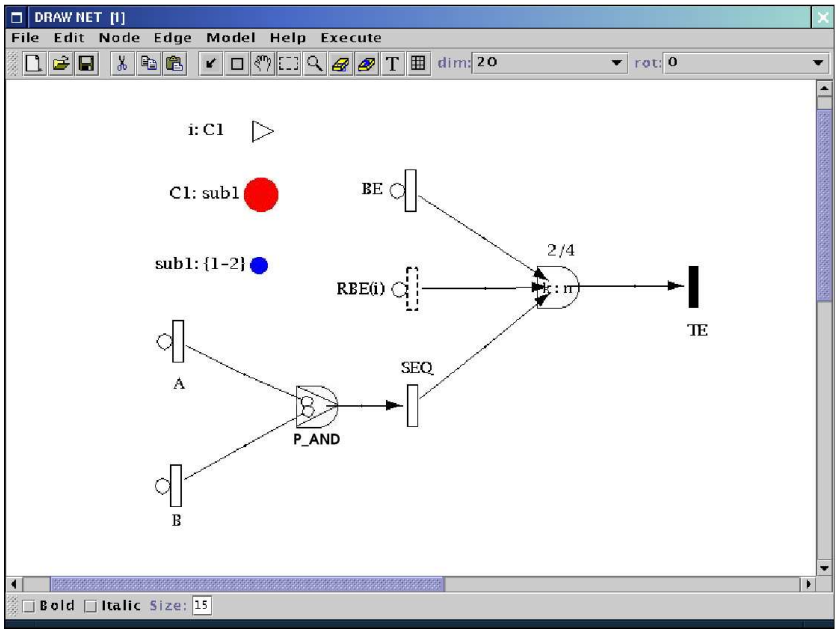


Figura 8.4: La porta P_And

Infatti in questo caso entrambi i componenti sono guasti e A ha smesso di funzionare prima di B, rispettando la logica della porta **P_And**.

8.4.4 Traduzione in Rete di Petri

La versione in *Rete di Petri* della porta **P_And** prevede una transizione, $T1$ che scatta quando A e B sono nello stato guasto causando il guasto generale; se però B si è guastato prima di A, B fa scattare un'altra transizione, $T2$, che porta allo stato di operatività bloccando $T1$.

Analogamente, se A si guasta per primo, blocca la transizione $T2$.

8.4.5 P_And in Astra

Astra prevede la presenza nell'albero della porta logica And prioritario che Astra però chiama *Inhibit* (**INH**); a livello grafico Astra distingue con due archi dall'aspetto diverso A e B che possono anche essere eventi interni.

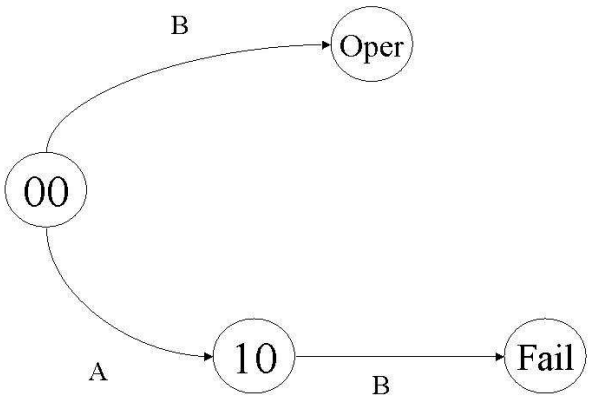


Figura 8.5: Modello di Markov per la porta P_And

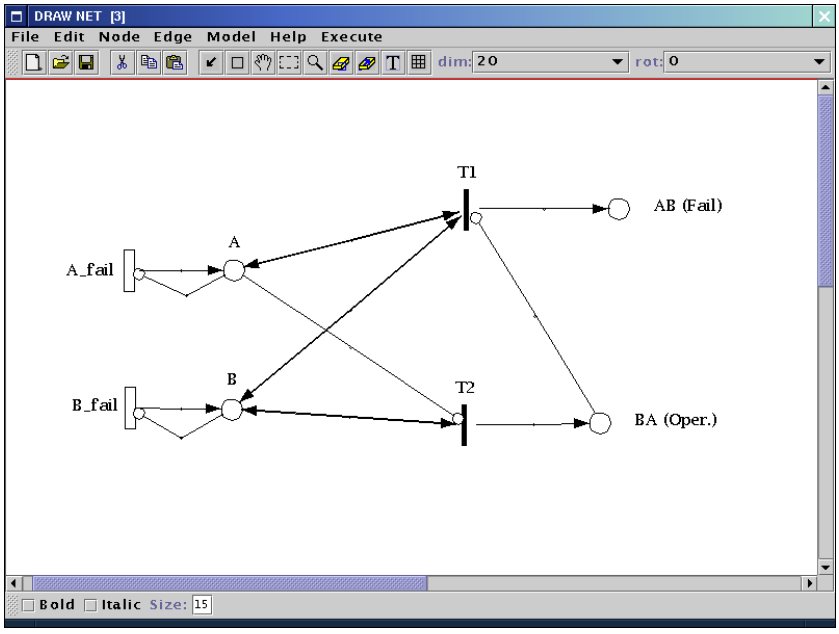


Figura 8.6: Rete di Petri per la porta P_And

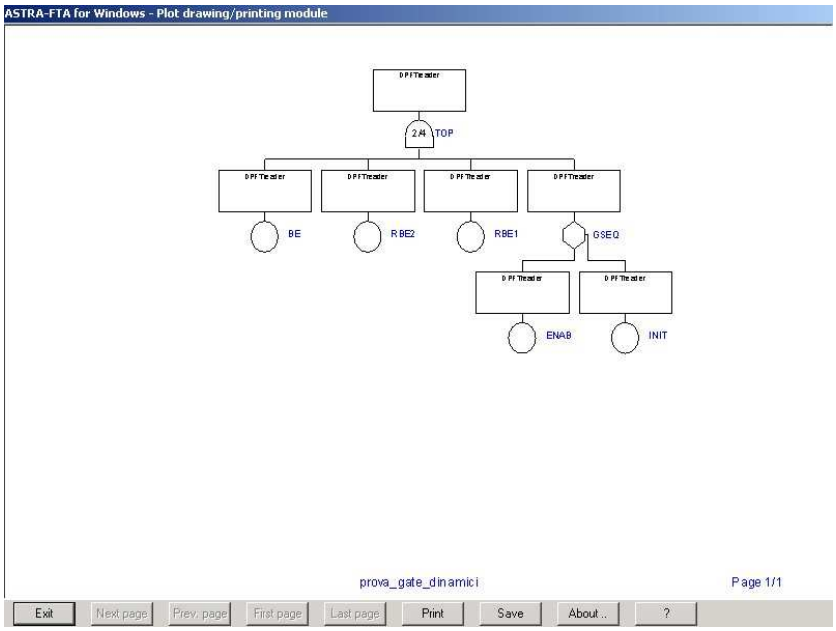


Figura 8.7: L'albero di Fig. 8.4 in Astra con la porta INH al posto di P_And

8.5 La porta And Sequenziale

8.5.1 Significato logico

La porta And Sequenziale (più brevemente **And_Seq**) [11] corrisponde a un And prioritario con un numero di eventi di input superiore a 2; questi si devono verificare tutti e in ordine stabilito che dovrebbe essere quello con cui compaiono al di sotto della porta; in questo caso il guasto si propaga all'evento di output.

Se l'ordine non è rispettato, il guasto dei componenti, anche di tutti, non determina il guasto generale.

8.5.2 Traduzione nel Modello di Markov

Nella rappresentazione markoviana della porta **And_Seq** c'è un solo percorso che porta dallo stato iniziale, quello con tutti i componenti funzionanti, allo stato di guasto generale; questo percorso rispetta esattamente l'ordine con cui si devono verificare i guasti; quindi se A, B, C e D sono gli eventi di input della porta, essi si devono guastare in tale ordine.

Se invece si guasta per primo B, ad esempio, non ci sarà mai il guasto generale e quindi si passa direttamente ad uno stato assorbente di funzionamento.

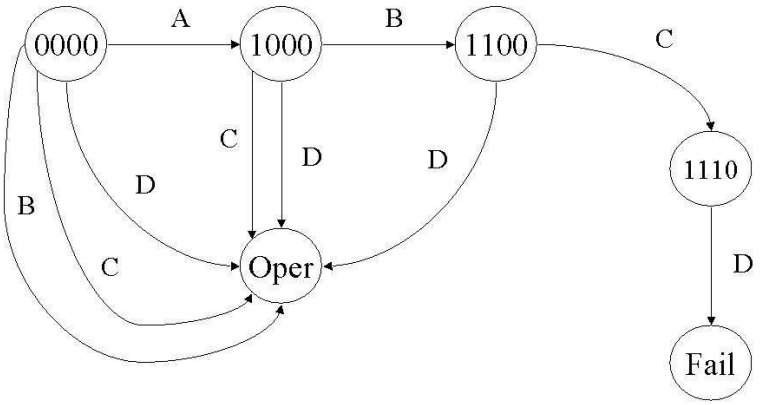


Figura 8.8: Modello di Markov per la porta And_Seq

8.5.3 Traduzione in Rete di Petri

Supponendo che gli eventi di input della porta siano 4, A, B, C, D, la rete contiene una serie di transizioni che portano allo stato di guasto generale se gli eventi accadono in tale ordine; agli eventi B, C, D corrisponde una specifica transizione che, nel caso questi eventi non rispettino l'ordine assegnato, porta nello stato di operatività bloccando la serie di transizioni che portano al guasto generale (Fig. 8.9).

8.6 La porta Fdep

8.6.1 Significato logico

La porta **Fdep** [10] [11] modella la dipendenza funzionale (**F**unctional **dep**endency): ha un numero variabile di eventi di input che si distinguono in:

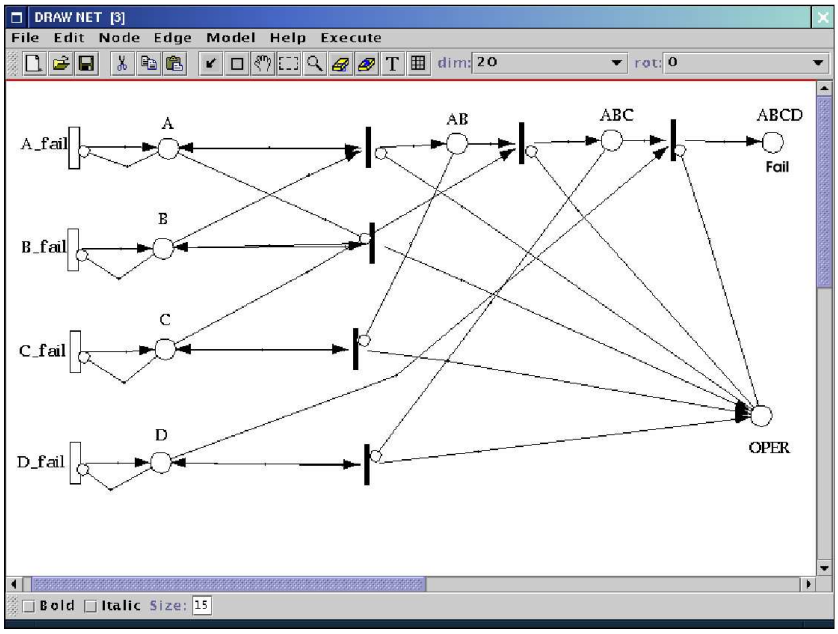


Figura 8.9: Rete di Petri per la porta And_Seq

- eventi dipendenti,

- evento *Trigger*.

Quando l'evento Trigger si guasta, anche gli altri eventi smettono di funzionare; gli eventi dipendenti si possono guastare autonomamente o per effetto del Trigger.

L'evento output della porta rispecchia lo stato del Trigger: il guasto non si propaga all'evento di output se si guastano degli input dipendenti, ma solo se si guasta il Trigger e di conseguenza gli altri eventi di input; la porta determina quindi una dipendenza degli eventi dipendenti nei confronti dell'evento Trigger.

Un esempio di applicazione della porta logica dinamica **Fdep** è quello in cui si hanno una serie di dispositivi elettrici collegati ad una fonte di corrente elettrica; se la corrente dovesse mancare allora i dispositivi smetterebbero di operare; il funzionamento dei dispositivi elettrici dipende quindi dall'erogazione di corrente elettrica.

8.6.2 Rappresentazione in DrawNet

La rappresentazione in DrawNet di questa porta prevede un apposito arco per collegare il Trigger alla porta e distinguerlo dagli altri eventi di input; si è poi imposto che un solo Trigger possa essere connesso alla porta rappresentata dalla specifica icona.

Seguono le righe aggiunte al formalismo di DrawNet per la porta **Fdep**.

```
<nodeType parent="Gate" name="FDEP"/>

<edgeType parent="" name="Trigger">
    <constraint fromType="Event" fromCardinality=""
     toType="FDEP" toCardinality="1"/>
    <constraint fromType="BasicEvent" fromCardinality=""
     toType="FDEP" toCardinality="1"/>
</edgeType>
```

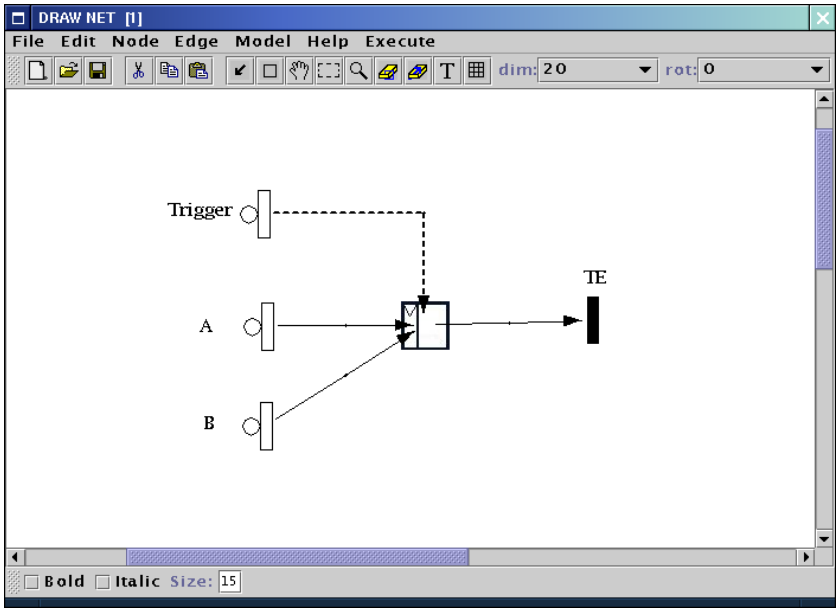


Figura 8.10: La porta Fdep

8.6.3 Traduzione nel Modello di Markov

Supponendo che la porta **Fdep** abbia due eventi dipendenti di input oltre al Trigger event, lo stato di partenza del modello di Markov corrispondente indica tutti i tre componenti operativi.

Dallo stato di partenza e da ogni stato intermedio, in caso di guasto del Trigger si passa direttamente allo stato assorbente che indica il guasto del sottosistema; gli stati intermedi rappresentano le varie configurazioni di guasto autonomo dei componenti indipendenti.

Nello stato assorbente si va comunque solo in caso di guasto del Trigger indipendentemente dallo stato degli altri componenti.

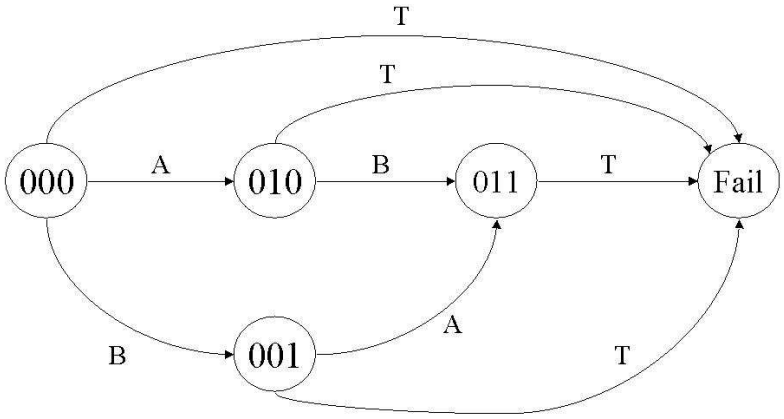


Figura 8.11: Modello di Markov per la porta Fdep

8.6.4 Traduzione in Rete di Petri

Nella rete di Petri corrispondente che prevede due eventi dipendenti, A e B, questi si possono guastare naturalmente, tramite la relativa transizione temporizzata, oppure possono smettere di funzionare al guasto del Trigger che fa scattare un gruppo di transizioni, T_A e T_B che determinano il guasto sia di A sia di B, se questi non sono già in tale stato. Un'altra transizione, T, sempre al momento del guasto del Trigger, determina il guasto generale (Fig. 8.12).

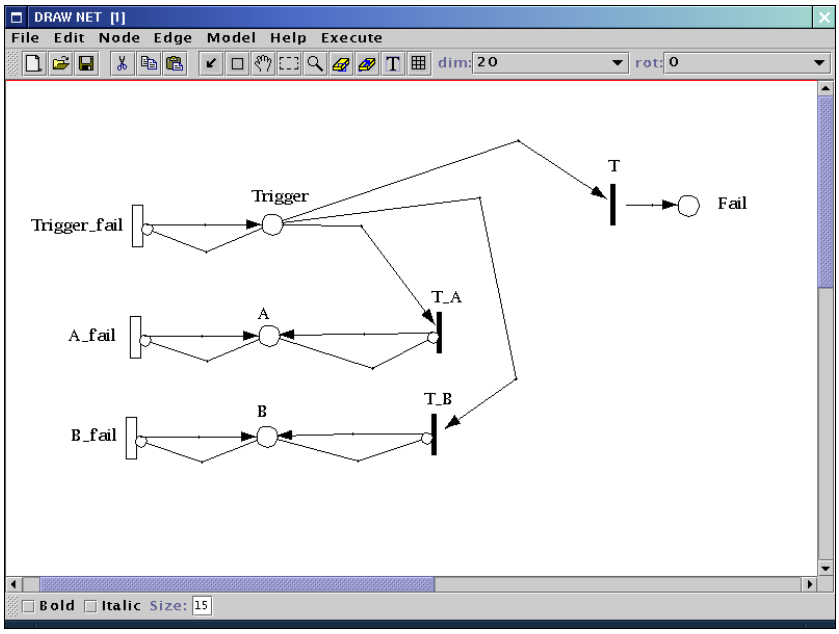


Figura 8.12: Rete di Petri per la porta Fdep

8.7 La porta WSP

8.7.1 Significato logico

La porta **WSP** (***W**arm **Sp**are Gate*) [10] consente di modellare la situazione in cui un componente del sistema dispone di una serie di dispositivi di riserva che lo sostituiscono in caso questo si guasti; la porta ha quindi un evento di input che rappresenta il guasto del componente principale e un insieme di eventi che rappresentano il guasto dei componenti di riserva.

Durante il funzionamento del componente principale, quelli di riserva non sono operanti, ma in uno stato di attesa (*stand-by*); quando il componente principale si guasta viene sostituito dalla prima riserva disponibile funzionante; infatti le riserve potrebbero a loro volta guastarsi, anche durante il periodo di funzionamento del componente principale.

I componenti di riserva però hanno due tassi di guasto, uno per lo stato di attesa e uno per quello di funzionamento; se il tasso di guasto del componente di riserva S_i nello stato di funzionamento è λ_i, il tasso di guasto per lo stato di attesa sarà $\alpha\lambda$, con α di valore compreso tra 0 e 1 e detto *fattore di attenuazione*.

In questo modo la probabilità che un componente di riserva si guasti durante l'attesa è minore rispetto a quella durante durante il funzionamento

del sistema. /indexcomponenti di riserva

Il guasto si propaga attraverso la porta **WSP** al relativo evento di output se, dopo il guasto del componente principale, tutte le riserve si sono a loro volta guastate.

I dispositivi di riserva si potrebbero guastare tutti prima del guasto del componente principale; in questo caso il guasto non si propaga attravero la porta finché il componente principale non si è ancora guastato.

8.7.2 Traduzione nel Modello di Markov

Il modello di Markov corrispondente alla porta WSP ha come negli altri casi lo stato iniziale che indica il funzionamento di tutti i componenti; gli stati intermedi tengono conto di tutti i possibili ordini di guasto supponendo ci siano due riserve, S_1 e S_2, per il componente principale P.

Nello stato assorbente che rappresenta il guasto generale si arriva comunque attraverso il guasto di tutti i componenti (principale e di riserva); fino a quando non si giunge in questo stato il sottosistema è da considerarsi operativo in quanto se non è il componente principale a svolgere una certa attività, è una delle riserve a farlo.

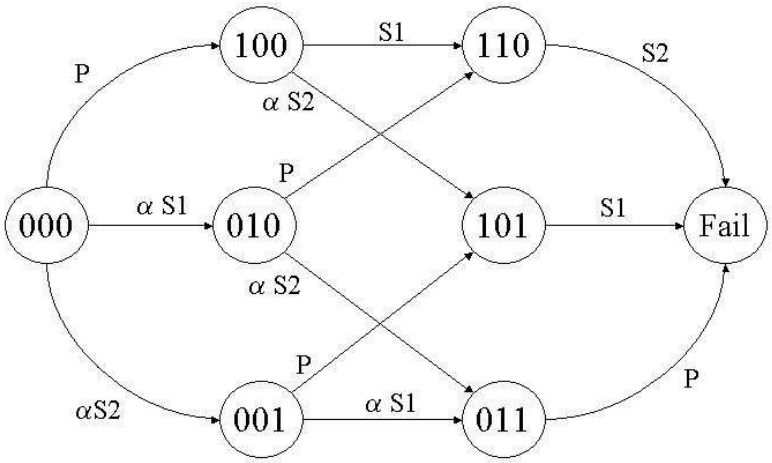


Figura 8.13: Modello di Markov per la porta WSP

8.7.3 Traduzione nella rete di Petri

Nella rete di Petri relativa alla porta **WSP** i posti P_S1 e P_S2 rappresentano rispettivamente lo stato in cui P è rimpiazzato da S_1 e quello in cui è rimpiazzato da S_2. Il token compare nel posto P_S1 se, nel momento del guasto di P, S_1 non è guasto attraverso la transizione $T2$; $T2$ viene bloccata se S_1 è guasto.

Nel posto P_S2 il token può essere portato da due transizioni: $T1$ e $T3$; $T1$ scatta se al momento in cui si guasta P, anche $S1$ è guasto; questa transizione è può essere bloccata dal guasto di S_2.

$T3$ scatta quando S_1, che sta sostituendo P, si guasta, portando il token in P_S2; anche questa transizione non può scattare se S_2 è guasto.

Il token nel posto Fail può arrivare attraverso le transizioni $T4$, $T5$ o $T6$; $T4$ scatta quando si guasta S_2 e questo stava sostituendo P; $T5$ scatta se quando P si guasta, S_1 e S_2 sono già in tale stato; $T6$ scatta se S_1, che stava rimpiazzando P, si guasta e in quel momento S_2 è già guasto (Fig. 8.14).

Le transizioni temporizzate $S1\_fail$ e $S2\_fail$ che determinano il guasto dei componenti di riserva hanno un tasso che varia a seconda della disposizione dei token nella rete; la transizione $S1\_fail$ scatta con tasso $\alpha\lambda_{S1}$ se nel posto P non c'è il token, altrimenti con tasso λ_{S1}; la transizione $S2_fail$ scatta con tasso $lambda_{S2}$ se c'è il token sia nel posto P sia nel posto S_1, altrimenti con tasso $\alpha\lambda_{S2}$.

8.8 La porta CSP

8.8.1 Significato logico

Il difetto della porta **WSP** consiste nel fatto che i componenti di riserva si possono guastare anche prima del componente principale venendo meno al loro specifico compito; nel caso peggiore tutti i componenti di riserva si potrebbero guastare prima di quello principale.

Per evitare questa situazione si dovrebbe fare in modo che le riserve cominciassero a funzionare solo al momento del guasto del componente principale o della riserva che lo sostituiva; la porta **CSP** (***C**old **Sp**are Gate*) [11] permette di modellare questa necessità.

Mentre il componente principale lavora, le riserve sono spente; non appena si guasta, la prima riserva nell'ordine con cui sono indicate graficamente, si accende e sostituisce il componente principale; se questa a sua volta si guasta toccherà alla riserva successiva adempiere le funzioni del componente principale e così via fino a quando tutte le riserve sono guaste.

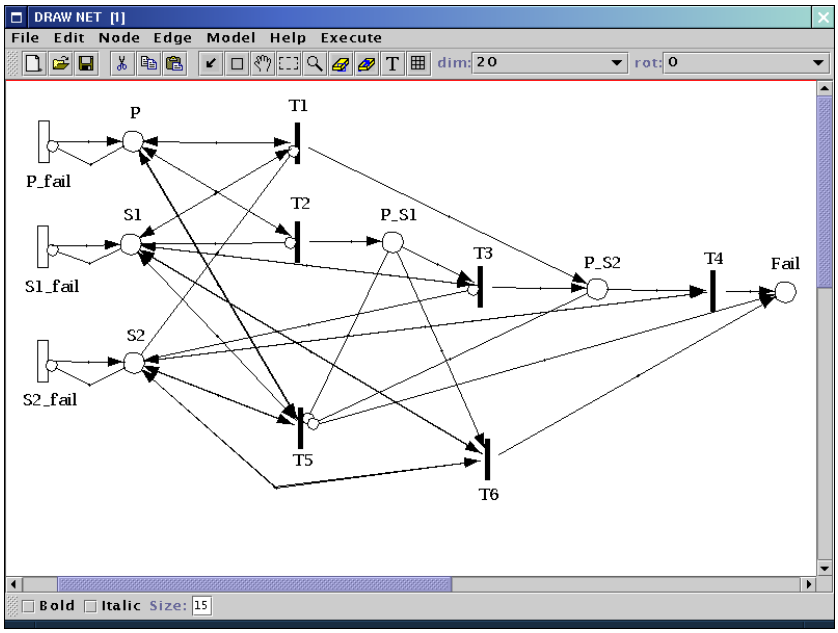


Figura 8.14: Rete di Petri per la porta WSP

A questo punto il guasto si propaga attraverso la porta **CSP** al relativo evento di output.

Quando si usa la porta **CSP** le riserve non sono in stato di attesa durante il funzionamento del componente principale, ma spente e quindi non si possono guastare; questa situazione equivale a quella che si avrebbe nel caso della porta **WSP** con il fattore di attenuazione $\alpha = 0$.

In sintesi, perché il guasto si propaghi si deve guastare prima il componente principale e poi tutti quelli di riserva nell'ordine con cui sono stati rappresentati nell'albero come eventi di input della porta.

Quando il componente principale o una riserva si guastano, non si deve individuare la prima riserva funzionante tra quelle a disposizione, come nel caso della porta **WSP**, ma si tratterà della riserva immediatamente successiva.

8.8.2 Traduzione nel Modello di Markov

Il modello di Markov corrispondente alla porta **CSP** è decisamente semplice: supponendo sempre che ci sia un componente principale P e due di riserva, S_1 e S_2, si parte dallo stato in cui P è funzionante (indicato dal valore 0) e S_1 e S_2 spenti (valore -1); si può guastare solo P e in caso ciò avvenga si passa allo stato in cui P è guasto (1), S_1 è acceso e funzionante (0), mentre S_2 è

ancora spento; se si guasta anche S_1 si attiva S_2 il cui eventuale successivo guasto determina il guasto generale.

Il percorso per passare dallo stato iniziale a quello finale assorbente è dunque unico dato che la sequenza degli eventi di guasto deve per forza rispettare un determinato ordine.

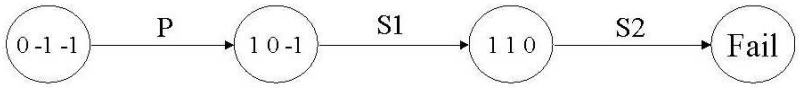


Figura 8.15: Modello di Markov per la porta CSP

8.8.3 Traduzione in Rete di Petri

In questa rete il guasto di S_1 è subordinato al guasto di P dato che la relativa transizione $S1\_fail$ può scattare solo se c'è il token nel posto P; per S_2 vale lo stesso discorso nei confronti del guasto di S_1.

Il guasto di S_2 determina il guasto generale portando un token nel posto Fail grazie alla transizione T (Fig. 8.16).

8.8.4 Rappresentazione in DrawNet

L'uso della porta **CSP** è più vantaggioso che ricorrere alla porta **WSP** perchè si evita il guasto dei componenti di riserva prima del tempo in cui si renderebbero utili e perchè l'analisi richiede uno spazio degli stati notevolmente ridimensionato.

Per questo si è scelto di adeguare DrawNet alla porta **CSP**; dato che è presumibile che il componente principale e quelli di riserva siano della stessa natura, si è pensato di rappresentare questa porta logica come un'icona distinta collegata ad un unico evento di input, ma di tipo *Replicator*, e naturalmente a un evento di output.

Si intende in questo modo che se $RE(i)$ è l'evento replicato, $RE(1)$ è il componente principale e i successivi quelli di riserva.

Ovviamente si è aggiornato il formalismo da passare a DrawNet inserendo queste righe:

```
<nodeType parent="" name="CSP"/>
```

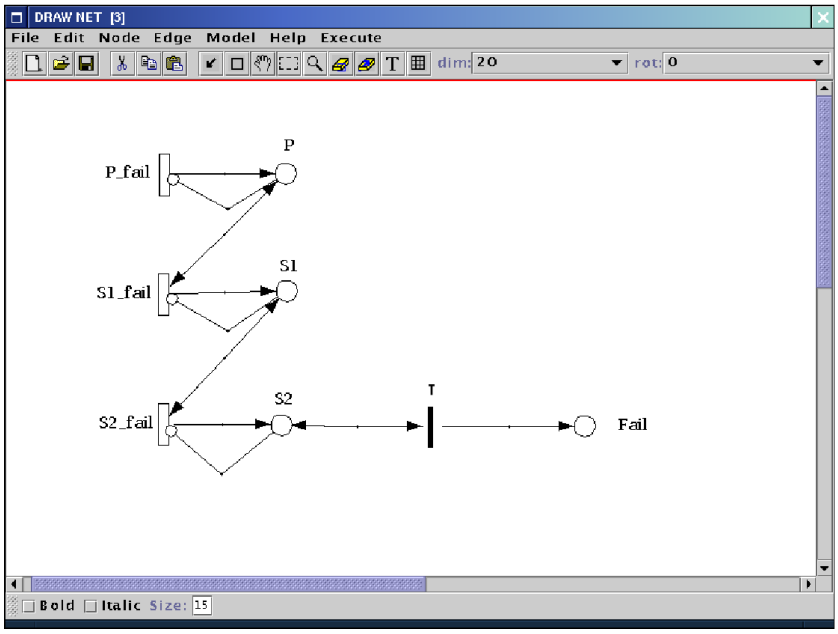


Figura 8.16: Rete di Petri per la porta CSP

```
<constraint fromType="ReplicatorEvent" fromCardinality="1"
 toType="CSP" toCardinality="1"/>
<constraint fromType="BasicReplicatorEvent" fromCardinality="1"
 toType="CSP" toCardinality="1"/>
```

La prima riga prevede la presenza della porta **CSP**, mentre le righe successive sono regole riguardanti gli archi generici e che impongono che ad una porta logica di tipo **CSP** si possa collegare solo un evento (di base o interno) di tipo *Replicator*.

Il formalismo finale per DrawNet considera quindi i PFT, le scatole di riparazione e le nuove porte logiche statiche e dinamiche; è il seguente:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE formalism SYSTEM "formalism.dtd">
<formalism parent="" name="PFT">

    <propertyType name="Title" default=""/>

        <nodeType parent="" name="Repair">
            <propertyType name="Repair_Rate" default="1.0"/>
```

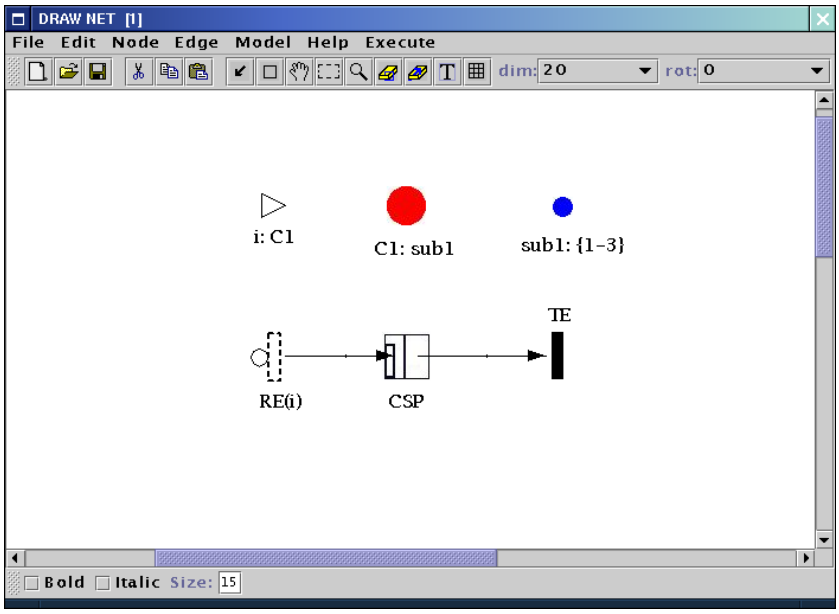


Figura 8.17: La porta CSP

```
</nodeType>

<nodeType parent="" name="Event0">
    <propertyType name="Label" default=""/>
    <propertyType name="Parameters" default=""/>
</nodeType>

<nodeType parent="Event0" name="Event"/>

<nodeType parent="Event0" name="BasicEvent">
      <propertyType name="Distribution" default="ALL EXP 1.0"/>
</nodeType>

<nodeType parent="Event" name="ReplicatorEvent">
    <propertyType name="PList" default=""/>
    <propertyType name="DeclaredParameters" default=""/>
    <propertyType name="PredSWN" default=""/>
</nodeType>

<nodeType parent="BasicEvent" name="BasicReplicatorEvent">
    <propertyType name="PList" default=""/>
```

```
    <propertyType name="DeclaredParameters" default=""/>
    <propertyType name="PredSWN" default=""/>
</nodeType>

<nodeType parent="" name="TopEvent">
    <propertyType name="Label" default=""/>
</nodeType>

<nodeType parent="" name="Gate"/>

  <nodeType parent="Gate" name="And"/>

<nodeType parent="Gate" name="Or"/>

<nodeType parent="" name="G2of3"/>

<nodeType parent="Gate" name="KofN">
    <propertyType name="K" default="2"/>
    <propertyType name="N" default="3"/>
</nodeType>

<nodeType parent="Gate" name="Xor"/>

<nodeType parent="KofN" name="K_out_N_cons"/>

<nodeType parent="" name="PAND"/>

<nodeType parent="Gate" name="FDEP"/>

<nodeType parent="" name="CSP"/>

<nodeType parent="" name="ColorClass">
    <propertyType name="SubClassList" default=""/>
    <propertyType name="Ordered" default="false"/>
</nodeType>

<nodeType parent="" name="ColorSubClass">
    <propertyType name="ElementList" default=""/>
</nodeType>

<nodeType parent="" name="ParameterType">
```

```
    <propertyType name="ParameterName" default=""/>
    <propertyType name="ParameterColor" default=""/>
</nodeType>

<edgeType parent="" name="Arc">
    <constraint fromType="Gate" fromCardinality="1"
     toType="Event" toCardinality="1"/>
    <constraint fromType="CSP" fromCardinality="1"
     toType="Event" toCardinality="1"/>
    <constraint fromType="G2of3" fromCardinality="1"
     toType="Event" toCardinality="1"/>
           <constraint fromType="PAND" fromCardinality="1"
     toType="Event" toCardinality="1"/>
    <constraint fromType="Gate" fromCardinality="1"
     toType="TopEvent" toCardinality="1"/>
    <constraint fromType="CSP" fromCardinality="1"
     toType="TopEvent" toCardinality="1"/>
    <constraint fromType="G2of3" fromCardinality="1"
     toType="TopEvent" toCardinality="1"/>
           <constraint fromType="PAND" fromCardinality="1"
     toType="TopEvent" toCardinality="1"/>
    <constraint fromType="Event0" fromCardinality=""
     toType="Gate" toCardinality=""/>
    <constraint fromType="Event0" fromCardinality=""
     toType="G2of3" toCardinality="3"/>
    <constraint fromType="ReplicatorEvent" fromCardinality="1"
     toType="CSP" toCardinality="1"/>
    <constraint fromType="BasicReplicatorEvent" fromCardinality="1"
     toType="CSP" toCardinality="1"/>
</edgeType>

<edgeType parent="" name="Order">
    <propertyType name="No." default="1"/>
    <constraint fromType="Event0" fromCardinality=""
     toType="PAND" toCardinality="2"/>
</edgeType>

    <edgeType parent="" name="Trigger">
    <constraint fromType="Event" fromCardinality=""
     toType="FDEP" toCardinality="1"/>
    <constraint fromType="BasicEvent" fromCardinality=""
```

```
        toType="FDEP" toCardinality="1"/>
    </edgeType>

    <edgeType parent="" name="Wrench">
        <constraint fromType="Repair" fromCardinality="1"
         toType="Event0" toCardinality="1"/>
        <constraint fromType="Repair" fromCardinality="1"
         toType="TopEvent" toCardinality="1"/>
    </edgeType>

</formalism>
```

La Fig. 8.18 indica le icone per le principali porte dinamiche introdotte, mentre la Fig. 8.19 mostra un albero PFT in cui si combinano porte logiche statiche, dinamiche e riparatori.

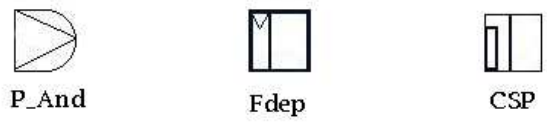


Figura 8.18: Le principali porte logiche dinamiche

8.9 Calcolo dei moduli di alberi dei guasti dinamici

Le porte logiche dinamiche determinano delle dipendenze tra i loro eventi di input; per questo le parti di un albero dinamico dove sono presenti tali porte non possono essere analizzate dal punto di vista combinatorio, ma nello spazio degli stati.

Un albero dinamico necessita perciò di un'analisi per moduli e potrebbe contenere anche eventi collegati a scatole di riparazione come nel caso dell'albero di Fig. 8.19.

Bisogna allora stabilire un algoritmo per l'individuazione dei moduli di un albero dinamico con eventi riparabili; per quanto riguarda la presenza di scatole di riparazione, anche per gli alberi dei guasti dinamici vale la regola

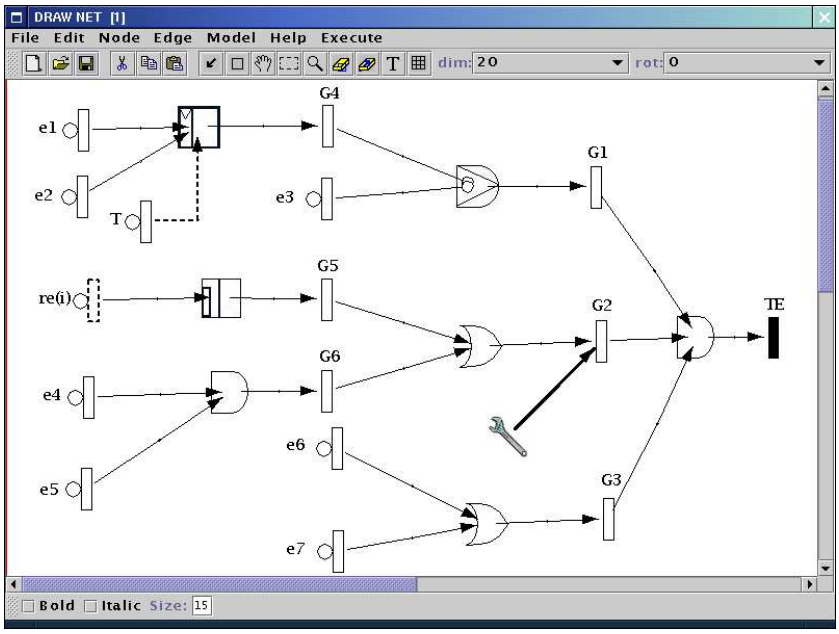


Figura 8.19: Albero con porte logiche dinamiche ed eventi riparabili

per cui al di sotto dell'evento a cui è collegata la scatola, non ci possono essere moduli, in quanto la riparazione determina una dipendenza su tutto il sottoalbero.

Analogamente le porte dinamiche impongono delle dipendenze sugli eventi di input che potrebbero essere anche eventi interni; per questo motivo anche al di sotto di una porta dinamica non ci possono essere moduli.

Un evento interno è la radice di un modulo se

- gli eventi sottostanti non sono condivisi con altre parti dell'albero;
- non discende da una porta dinamica;
- non discende da eventi collegati a scatole di riparazione.

8.9.1 L'algoritmo

Per stabilire se un sottoalbero non ha eventi condivisi si può sempre ricorrere all'esame dei tempi di visita dei nodi dell'albero come descritto nel Cap.5.

L'algoritmo per l'individuazione dei moduli del Cap. 6 prevedeva solo gli eventi riparabili; per le porte logiche dinamiche lo si può aggiornare nel seguente modo: si sostituisce la variabile *rep* con la variabiledip che segnala,

quando si visita un nodo, se questo è in una parte dell'albero sottoposta a dipendenza per riparazioni o per porte dinamiche.

```
VISITA2(nodo, dip)
{

  if ((nodo \‘e collegato a una scatola di riparazione) or
    (nodo \‘e output di una porta dinamica))
    dip2:=1;
  else
    dip2:=dip;

  for i:=1 to nodo.num_figli
  {
    ..........
  }

  if ((nodo.min_t1 > nodo.t1) and (nodo.max_last < nodo.t2) and (dip=0))
    nodo \‘e un modulo

  .........

}
```

8.9.2 La classificazione dei moduli

La distinzione tra MSC, MSC_{max}, MSS e MSS_{min} pu‘øessere ora fatta secondo nuovi criteri che tengono in considerazione le porte dinamiche:

- MSC
  - non contiene nessun evento condiviso con altri alberi;
  - nessuno dei suoi eventi è collegato ad una scatola di riparazione;
  - non contiene porte dinamiche;
  - nessuno dei suoni nodi antenati è connesso ad una scatola di riparazione;
  - non discende da una porta dinamica.
- MSC_{max}
  - è un modulo MSC;

  - non è contenuto all'interno di un altro MSC.

- MSS
  - non contiene nessun evento condiviso con altri sottoalberi;
  - contiene almeno un nodo collegato ad una scatola di riparazione *oppure* contiene almeno una porta dinamica;
  - nessuno dei suoni nodi antenati è connesso ad una scatola di riparazione;
  - non discende da una porta dinamica.

- MSS_{min}
  - è un modulo MSS;
  - non contiene al suo interno altri moduli di qualunque natura.

La procedura per l'analisi dei moduli di alberi dinamici [9] segue gli stessi passi descritti nel paragrafo 7.3.

8.9.3 Esempio

Consideriamo come esempio l'albero di Fig. 8.19 in cui si possono individuare tre moduli che sono: G_1, G_2 e G_3, oltre naturalmente al *Top Event*; G_1 è di tipo MSS_{min} in quanto è indipendente, contiene due porte dinamiche e nessun altro modulo al suo interno; anche G_2 è un MSS_{min} visto che è indipendente, contiene un evento collegato alla scatola di riparazione e non contiene moduli al suo interno a causa della riparazione; G_3 è un MSC_{max} dato che, oltre ad essere indipendente, non contiene né eventi riparabili né porte dinamiche e non si trova all'interno di altri moduli MSC. Il *Top Event* è da considerarsi un MSS dato che contiene riparazione e porte dinamiche e al suo interno si trovano altri moduli.

L'algoritmo analizza singolarmente prima i moduli MSS_{min}, in questo caso G_1 e G_2; quindi sostituisce ciascuno con un evento di base con probabilità di guasto corrispondente a quella del modulo; G_1 e G_2 sono analizzati nello spazio degli stati; le Fig. 8.20 e 8.21 mostrano rispettivamente la rete di Petri corrispondente a G_1 e quella corrispondente a G_2; dopo aver analizzato i moduli e averli sostituiti l'albero assume la forma mostrata in Fig. 8.22; l'ulteriore individuazione dei moduli sull'albero così modificato rileva il *Top Event* come MSC_{max} che come tale viene analizzato.

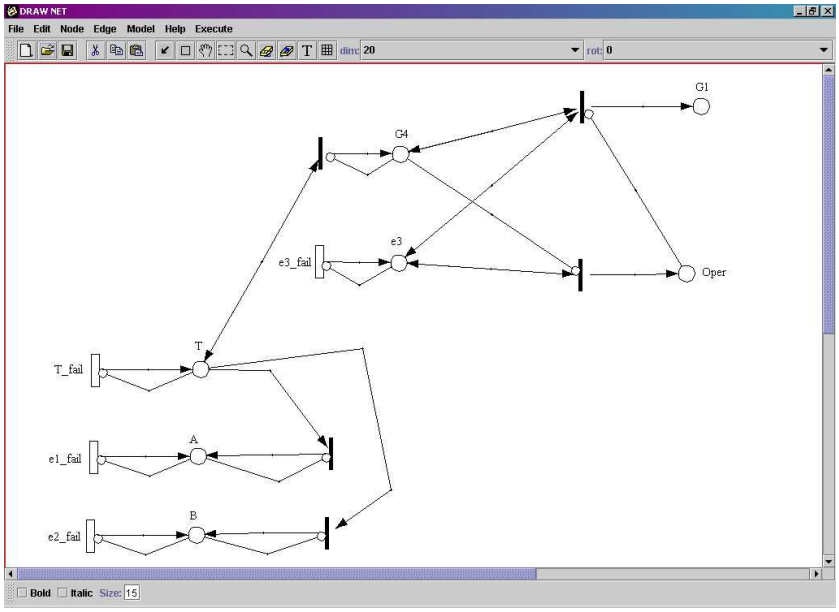


Figura 8.20: La rete di Petri per G_1

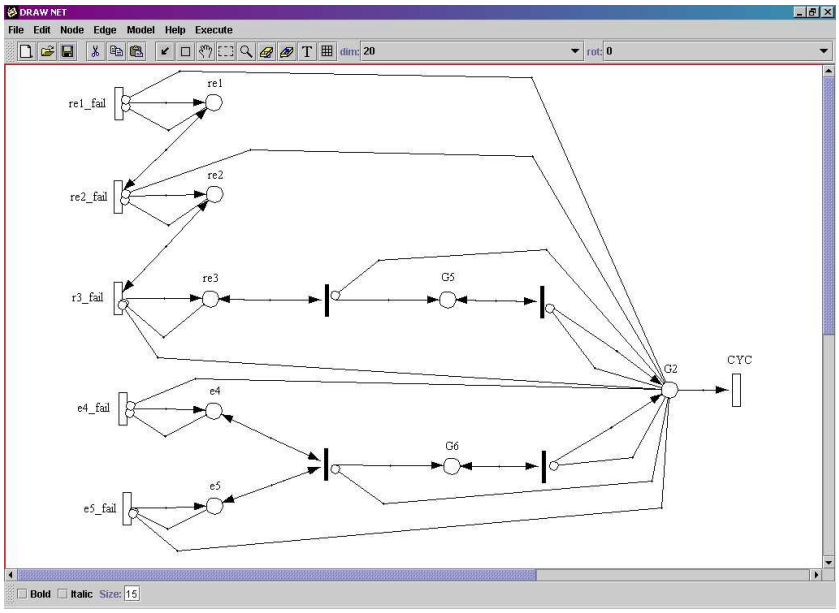


Figura 8.21: La rete di Petri per G_2

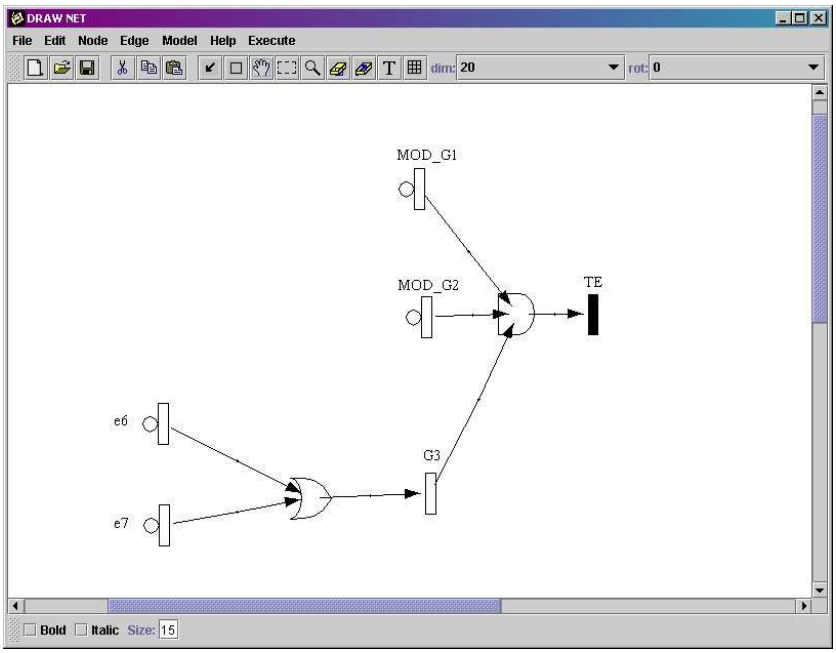


Figura 8.22: L'albero dopo le sostituzioni dei moduli

Conclusioni

Lo sviluppo della tesi è iniziato considerando alberi dei guasti dei più elementari per poi passare a quelli parametrici, a quelli con scatole di riparazione ed infine agli alberi dinamici.

Per quanto riguarda gli alberi statici senza riparatori, si è ricorso agli strumenti *Sharpe* e *Astra*, mentre per gli alberi riparabili si è sviluppato un sistema di analisi basato sulle Reti di Petri in grado però di analizzare solo determinati tipi di moduli.

Le porte logiche dinamiche, inoltre, sono state esaminate dal punto di vista logico, ma non è stato implementato un sistema per la loro analisi nello spazio degli stati.

Essendo comunque in grado di individuare e classificare correttamente i moduli di un albero dei guasti di qualunque genere e disponendo di un algoritmo che stabilisce quali moduli analizzare e in quale ordine, un ulteriore sviluppo dei temi trattati nella tesi potrebbe riguardare l'analisi di alberi con situazioni di riparazione più complicate e con porte logiche dinamiche.

Per quanto riguarda le riparazioni, gli aspetti ancora da affrontare sarebbero l'analisi di moduli con riparazioni in cascata e il caso in cui i componenti si possono guastare durante una riparazione.

Per quanto riguarda gli alberi dinamici, sarebbe da sviluppare la generazione automatica delle reti di Petri riguardanti i moduli con porte dinamiche, per effettuare poi l'analisi nello spazio degli stati.

Inoltre gli aspetti ancora da valutare circa le riparazioni potrebbero riguardare anche gli alberi dinamici, complicando ulteriormente la generazione delle relative reti di Petri.

Appendice A

Le Reti di Petri

Le *Reti di Petri* [14] [13] sono un formalismo grafico per rappresentare la struttura un sistema; rispetto agli altri modelli, le reti di Petri sono particolarmente adatte a descrivere le interazioni tra i componenti del sistema e le conseguenze che il funzionamento di uno di questi determina su di un altro.

Questo modello è stato ideato a **C. A. Petri** nel 1962 e nel corso degli anni è stato oggetto di numerosi studi, applicazioni ed evoluzioni quali le *reti di Petri stocastiche*.

A.1 Il modello

La rete di Petri è un modello che si base su tre elementi primitivi:

- **posti**: indicati in Fig. A.1 come cerchi;
- **transizioni**: indicati come rettangoli;
- **archi**.

Un arco può connettere un posto con una transizione o una transizione con un posto; un posto rappresenta una condizione, una transizione corrisponde ad un evento; la condizione corrente è quella data dal posto o dai posti dove è presente un *token* (gettone).

Una transizione ha un insieme di archi entranti e un insieme di archi uscenti; la transizione, quando "scatta" ha l'effetto di far passare il token dai posti relativi agli archi entranti (posti di ingresso) ai posti relativi agli archi uscenti (posti di uscita); la transizione può "scattare" se tutti i posti di ingresso sono marcati.

Un posto è marcato se in esso è presente almento un token; infatti in un posto potrebbero anche essere presenti un numero di token superiore a uno.

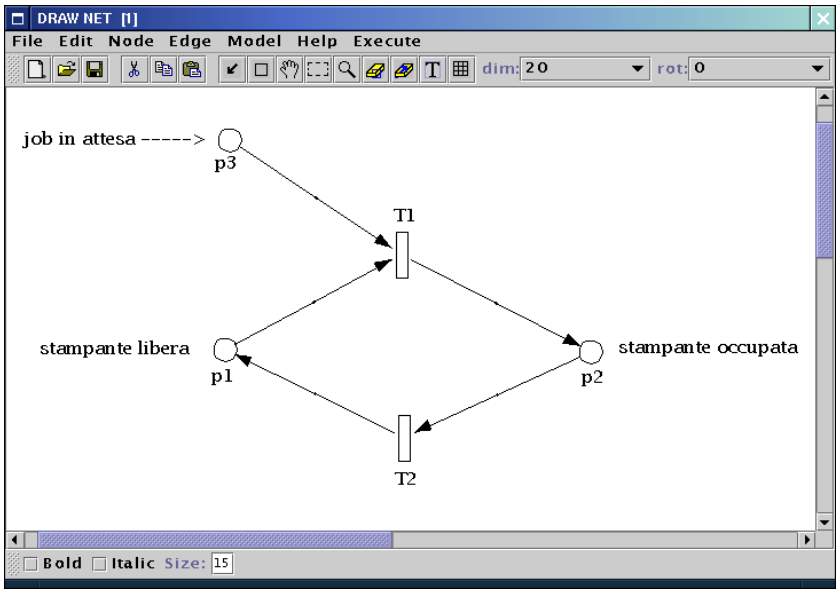


Figura A.1: La rete di Petri per una stampante

Prendiamo come esempio la rete di Petri di Fig. A.1; essa rappresenta il funzionamento di una stampante; la presenza di token nel posto $p3$ indica la condizione in cui ci sono dei documenti che attendono di essere stampati; in questo posto ci saranno tanti token quanti i documenti da stampare.

Un token nel posto $p2$ indica che la stampante è occupata nella stampa di qualche documento, mentre un token nel posto $p1$ rappresenta la condizione in cui la stampante non sta lavorando (stato *idle*).

La transizione $T1$ corrisponde all'evento di inizio della stampa di un nuovo documento; questa transizione può scattare se c'è almeno un token in $p3$, cioè c'è almeno un documento in attesa di essere stampato, e $p1$ è marcato, ossia la stampante è libera.

L'effetto della transizione $T1$ è quello di rimuovere un token da $p3$ (ora c'è un documento in meno in attesa) e il token di $p1$ (la stampante non è più libera) e di mettere un token nel posto di uscita, cioè $p2$ che, se marcato, indica che la stampante è occupata.

La transizione $T2$ ha invece l'effetto di rimuovere il token di $p2$ e di trasferirlo in $p1$ portando la stampante dallo stato di attività allo stato di inattività; l'evento che corrisponde a questa transizione è la fine della stampa del documento corrente.

In sintesi gli elementi che contraddistinguono una rete di Petri sono:

- $P = \{p_1, p_2, \ldots\}$ è l'insieme dei posti;

- $T = \{t_1, t_2, \ldots\}$ è l'insieme delle transizioni;
- I è la relazione di ingresso della transizione ed è rappresentata per mezzo degli archi diretti dai posti alle transizioni;
- O è relazione di uscita della transizione ed è rappresentata per mezzo degli archi diretti dalle transizioni ai posti;
- $M = \{m_1, m_2, \ldots\}$ è la marcatura, cioè il numero di token in ogni posto; il valore di m_i ìl numero di token presenti in p_i;
- M_0 è la marcatura iniziale, cioè l'insieme delle condizioni iniziali del modello.

A.2 Archi speciali

Solitamente viene prelevato da un posto un token ogni volta che scatta una transizione; se si vuole invece che una transizione rimuova da un posto più di un token, si può associare all'arco che va dal posto alla transizione una *molteplicità*.

In questo modo quando la transizione scatta rimuove più di un token dal posto; nella stessa maniera una transizione può depositare nel posto di uscita un numero di token superiore a uno.

Inoltre un arco può essere **bidirezionale**, nel senso che ha entrambi i versi, dal posto alla transizione e dalla transizione allo stesso posto; applicare questo tipo di arco equivale ad avere due archi, ognuno in un verso.

L'effetto dell'arco bidirezionale è quello di mantenere invariato il numero di dei token presenti in un posto nonostante la transizione sia scattata.

Un altro tipo di arco è l'arco **inibitore** che ha lo scopo di impedire la transizione a cui è collegato se nel posto all'altra estremità dell'arco è presente almeno un token.

La Fig. A.2 mostra una rete in cui appaiono un arco bidirezionale e un arco inibitore.

A.3 Tipi di transizione

Le transizioni, in base al tempo che richiedono per scattare una volta che i posti di ingresso sono tutti marcati, si dividono in due categorie:

- transizioni **immediate**: scattano subito, non appena i posti di ingresso sono marcati e il loro effetto è immediato; sono indicate graficamente da un rettangolo pieno.

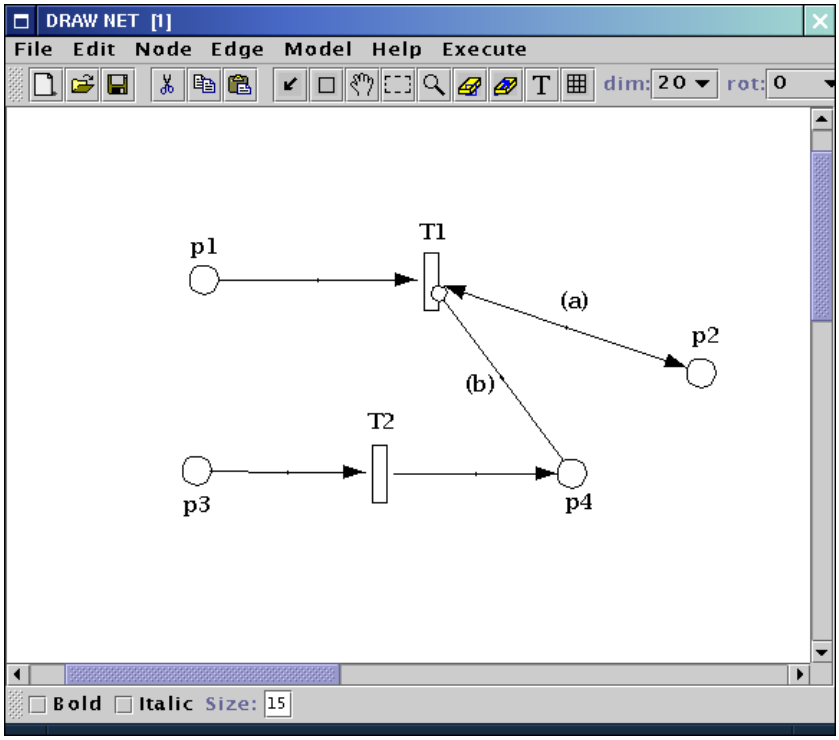


Figura A.2: Rete di Petri con un arco bidirezionale (a) e un arco inibitore (b)

- transizioni **temporizzate**: possono impiegare un certo tempo prima di scattare, nonostante tutti i posti di ingresso siano marcati; seguono una distribuzione esponenziale per cui, se una transizione ha parametro λ_i, il tempo medio necessario per scattare è $1/\lambda$.

  Una rete di Petri con transizioni temporizzate viene detta *Rete di Petri stocastica* (SPN); le transizione temporizzate sono indicate graficamente da un rettangolo vuoto.

La Fig. A.3 è un esempio di rete di Petri con i due tipi di transizione.

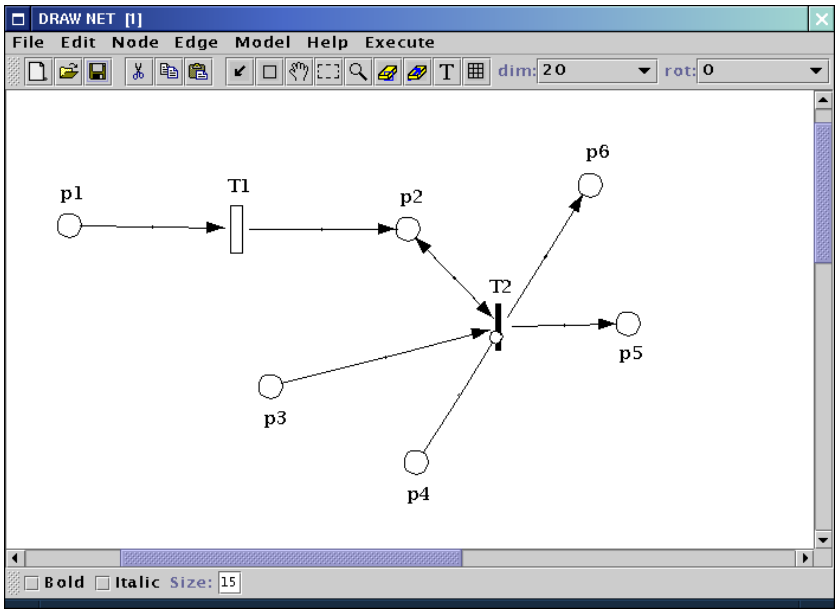


Figura A.3: Rete di Petri con transizioni immediate e temporizzate

Bibliografia

[1] W.G. Schneeweiss. *The Fault Tree Method*. LiLoLe Verlag, 1999.

[2] K.S.Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Wiley, 2002.

[3] S. Contini and A. Poucet. Advances on fault tree and event tree techniques. In A.G. Colombo and A. Saiz de Bustamante, editors, *System Reliability Assessment*, pages 77–102. Kluwer Academic P.G., 1990.

[4] R.A.Sahner K.S.Trivedi A.Puliafito. *Performance And Reliability Analysis Of Computer Systems*. Kluwer Academic Publishers, 1996.

[5] S.Contini. Astra - knowledge handbook - logical and probabilistic analysis method, 1998.

[6] A.Bobbio G.Franceschinis R.Gaeta L.Portinale. Parametric fault-tree for the dependability analysis of redundant systems and its high level petri nets semantics.

[7] http://www.di.unito.it/ greatspn/drawnet. DrawNet home page.

[8] G.Franceschinis M.Gribaudo M.Iacono N.Mazzocca V.Vittorini. Towards an object based multiformalism multi-solution modeling approach. *Proceedings del Second Workshop on Modeling of Objects, Components and Agents*, 2002.

[9] A.Anand A.K.Somani. Hierarchical analysis of fault trees with dependencies, using decomposition. *Proceeding Annual Reliability And Maintainability Symposium*, 1998.

[10] R.Manian D.W.Coppit K.J.Sullivan J.B.Dugan. Bridging the gap between systems and dynamic fault trees models. *Proceeding Annual Reliability And Maintainability Symposium*, 1999.

[11] J.B.Dugan S.J.Bavuso M.A.Boyd. Dynamic fault tree models for fault-tolerant computer systems. *IEEE Transactions On Reliability, vol. 41, no. 3*, 1992.

[12] Y.Dutuit A.Rauzy. A linear-time algorithm to find modules of fault trees. *IEEE Transactions On Reliability, vol. 45, no. 3*, 1996.

[13] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, 1995.

[14] A. Bobbio, G. Franceschinis, L. Portinale, and R. Gaeta. Exploiting Petri nets to support fault-tree based dependability analysis. In *8-th International Conference on Petri Nets and Performance Models - PNPM99*, pages 146–155. IEEE Computer Society, 1999.

[15] C.Bertoncello. ptf2swn: rapporto tecnico, 2001.

[16] W.G.Shneeweiss. Fault tree analysis of consecutive-k-out-of-n systems. *Esrel 2001: Safety And Reliability*, 2001.

[17] A. Bobbio, G. Franceschinis, L. Portinale, and R. Gaeta. Dependability assessment of an industrial programmable logic controller via parametric fault-tree and high level petri net. In *Proceedings 9th International Workshop on Petri Nets and Performance Models - PNPM01*, pages 29–38. IEEE Computer Society, 2001.

[18] http://www.di.unito.it/ greatspn. GreatSPN home page.

Indice analitico