UN ALGORITHME RAPIDE D'ANALYSE DES ARBRES DE DEFAILLANCE NON-COHERENTS

A FAST ALGORITHM FOR NON-COHERENTFAULT TREE ANALYSIS

R. ZIANI

Département Electronique, Université de Tizi-ouzou
Ziani_r@yahoo.fr

RESUME

Nous présentons dans cet article un algorithme rapide pour la recherche des implicants premiers des arbres de défaillance non cohérents. Ces derniers sont fréquemment rencontrés lors de l'analyse de sûreté des systèmes comprenant des boucles de régulation. Leur traitement présente beaucoup plus de difficultés que celui des arbres de défaillance cohérents à cause des implicants de consensus. La recherche des implicants premiers passe par deux étapes. Dans une première étape, nous déterminons tous les implicants. Dans une deuxième étape, nous éliminons les implicants redondants afin d'obtenir les implicants premiers. L'atout majeur de notre algorithme, est que lors de la recherche des implicants, un certain nombre d'entre eux ne sont pas calculés. Ainsi, la réduction portera sur un nombre de termes plus restreint. Nous montrerons à travers un exemple d'application que cette nouvelle approche permet un gain important en espace mémoire et en temps d'exécution.

1 INTRODUCTION

L'arbre de défaillance (AdD) est une technique d'ingénierie bien connue et largement utilisée dans les études de sûreté de fonctionnement des systèmes. Un AdD est défini comme un graphe orienté formé de niveaux successifs tel que chaque évènement est généré par des évènements d'ordre inférieur, agissant à travers des portes logiques. Son principe consiste à partir d'un événement «sommet» associé à la défaillance du système et à rechercher les causes puis toutes les combinaisons d'évènements qui conduisent à la réalisation de cet événement sommet. Son efficacité a été notamment révélée grâce au rapport «WASH 1400» de Rasmussen sur l'évaluation des risques d'accident dans les centrales nucléaires [1].

Depuis 1961 qui voit naître ce concept, un grand nombre d'algorithmes pour la recherche des ensembles minimaux ont été développés [2]. Ce fut d'abord des algorithmes de simulation et des algorithmes de manipulation algébriques, puis des algorithmes de décomposition dits matriciels [3,4] et par la suite des algorithmes de décision binaires [5,6,7].

Selon la nature du système, nous distinguons les arbres de défaillance cohérents (AdD-C) et les arbres de défaillance non- cohérents (AdD-NC). Les AdD-C sont caractérisés par les opérateurs logiques ET, OU et par des variables de base monoformes représentant la défaillance des éléments du système (évènements élémentaires de même catégorie x_i). Les AdD-NC sont caractérisés par les opérateurs logiques ET, OU, NAND, NOR, EOR, NON et par des variables de base biformes représentant la défaillance et la non-défaillance des éléments du système (évènements élémentaires de catégorie différente x_i et $\overline{x_i}$).

Si la maîtrise conceptuelle des AdD est désormais acquise, nous nous trouvons encore confrontés à des problèmes lors de leur exploitation informatique, notamment lorsque l'étude de sûreté est menée sur de petits systèmes informatiques. Ces difficultés apparaissent lors du traitement qualitatif de l'AdD qui consiste en la recherche des ensembles minimaux (ensemble d'éléments dont la défaillance conduit à la défaillance du système). Ces ensembles minimaux sont appelés coupes minimales dans le cas des AdD-C et implicants premiers dans le cas des AdD-NC. L'inconvénient majeur induit par ce genre de technique réside dans l'espace mémoire requis au stockage de tous ces ensembles et au temps d'exécution qui peut s'avérer alors prohibitif. Ceci se produit notamment, lors du traitement d'arbres de grande taille. Pour améliorer la performance de ces algorithmes, l'effort des chercheurs a surtout porté sur l'étape de réduction. En effet, cette dernière reste celle qui consomme le plus de temps à cause du nombre de comparaisons nécessaires pour l'obtention des ensembles minimaux.

Dans les sections qui suivent, nous présentons un nouvel algorithme pour la recherche des implicants premiers d'un AdD-NC. Son atout majeur réside dans le gain en espace mémoire et en temps d'exécution.

2 PRINCIPE DE L'ALGORITHME

L'algorithme que nous avons développé est basé sur le principe suivant : «une condition nécessaire et suffisante pour qu'une fonction Ψ (fonction de structure de l'AdD) admette un produit de littéraux P = L_1 L_2 …… L_n comme implicant est que tous les produits de son dual Ψ* s'annulent quand tous les littéraux du produit P sont portés à zéro [4]».

L'arbre de défaillance donné en figure 1-a sera utilisé pour illustrer notre l'algorithme. Ce dernier comprend trois étapes :

Etape1 : Cette étape consiste à construire l'arbre dual et à déterminer les expressions locales des différentes portes ainsi que leur domaine. Pour ce faire nous procédons de la façon suivante :

- Remplacer dans l'arbre de défaillance les portes NAND, NOR, EOR par leur équivalent en portes AND, OR, NON, donné en figure 2-a.
- Ramener l'opérateur NON au dernier niveau de l'AdD où il sera pris en compte de manière implicite en remplaçant la variable par sa variable complémentaire comme indiqué en figure 2-b.
- Construire l'arbre dual. Celui-ci est obtenu en remplaçant les portes AND par des portes OR et les portes OR par des portes AND. L'arbre de défaillance de la figure 1-b est obtenu.
- Obtenir les expressions locales des portes de l'arbre dual en décomposant chaque porte en ses entrées.
- Obtenir le domaine P [Gi] de chaque porte Gi en déterminant tous les littéraux qui s'y rapportent.

Etape 2 : Cette étape consiste à rechercher dans l'arbre dual les opérateurs L_k^{ik} ……. L_1^{i1} qui satisfont l'une des identités suivantes :

$$T_k\Psi^* = L_k^{ik} \ldots\ldots\ldots L_1^{i1}\ \Psi^* = 0 \quad (1)$$

où

$$T_k\Psi^* = L_k^{ik} \ldots\ldots\ldots L_1^{i1}\ \Psi^* = 1 \quad (2)$$

Avec :

Ψ* : la fonction booléenne de l'événement sommet de l'AdD dual

i_1, ……. i_k : des entiers prenant la valeur 0 ou 1, et

L_i : un littéral désignant un événement de base x_i ou son événement complémentaire $\overline{x_i}$

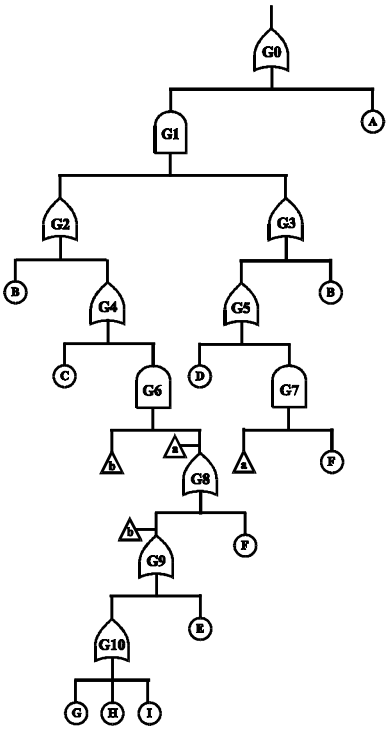

Figure 1-a : *AdD1*

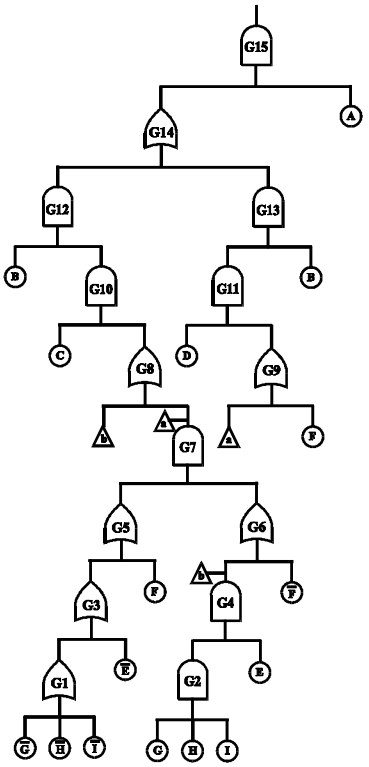

Figure 1-b : *arbre dual*

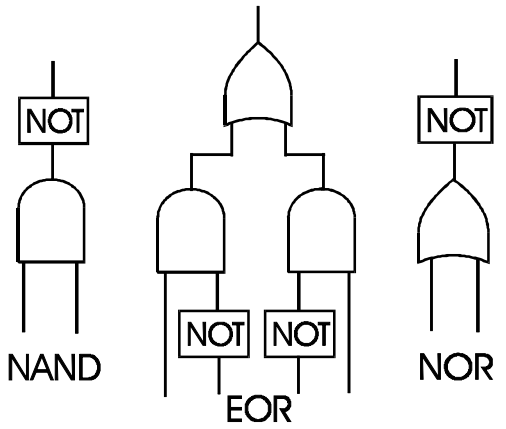

Figure 2-a : *Librairie des NAND, NOR et EOR*

Lorsqu'un littéral L apparaît dans l'expression booléenne de Ψ*, nous procédons à une classification dichotomique engendrant ainsi deux classes d'implicants L^1 et L^0.

La classe L^1 qui représente la classe d'implicants ne contenant pas L est obtenue en remplaçant dans Ψ* le littéral L par un «1» (L=1).

La classe L^0 qui représente la classe d'implicants contenant L est obtenue en remplaçant dans Ψ* le littéral L par un «0» (L=0) et le littéral $\overline{L}$ par un «1 » ($\overline{L}$ =1).

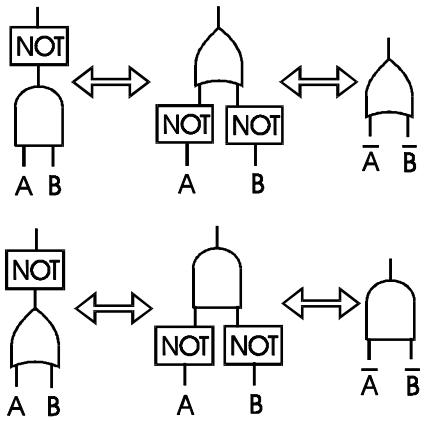

Figure 2-b : *Transformation équivalente*

Chaque nouvelle sous-classe subit à son tour une classification dichotomique et le traitement se poursuit jusqu'à arriver à un nœud terminal vérifiant l'identité (2.1) ou (2.2). On dit alors que la classe est irréductible. Cette procédure est décrite par l'arbre binaire de la figure 3 dans laquelle Le nœud terminal est représenté par un cercle noir. Les noeuds non terminaux sont classés en 2 catégories, à savoir les nœuds actifs contenant des littéraux et les nœuds non actifs ne contenant que des portes logiques.

La classe vérifiant l'identité (1) signifie que l'ensemble des littéraux de cette classe constitue un implicant pour la fonction Ψ et celui ci est un implicant premier dans sa classe.

La classe vérifiant l'identité (2) signifie que l'ensemble des littéraux de cette classe ne constitue pas un implicant pour la fonctionΨ.

Ainsi par exemple sur la figure 3, les nœuds terminaux I et II vérifient respectivement :

$A^0\Psi^* = 0$ et $F^1C^0D^1B^1A^1\Psi^* = 1$

Ce qui signifie que le littéral A est un implicant premier.

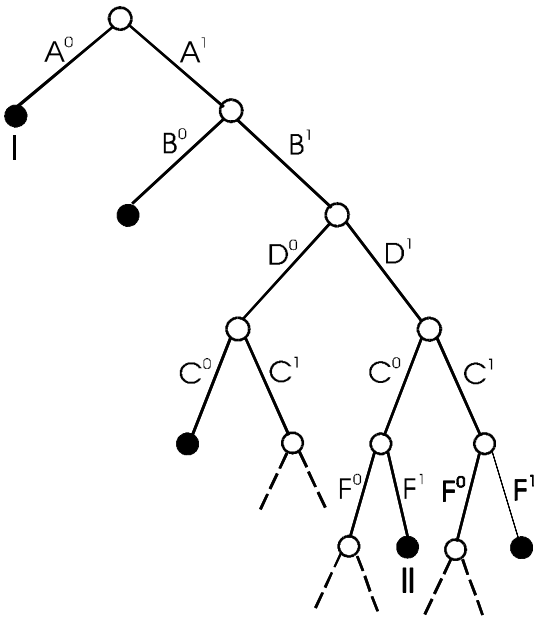

Figure 3 : *Arbre binaire*

L'étape 2 peut être détaillée comme suit :

Etape 2-1 : initialiser k à 0

Le nœud de départ a l'identité $T_0\Psi^* = T_0 G_M$.

(dans l'exemple de la figure 1.b nous aurons :

$T_0\Psi^* = T_0 G_{15} = AT_0 G_{14}$).

Dans les étapes 2.2 à 2.4, les nœuds à l'entrée k sont traités.

Etape 2-2 : tester la nature des nœuds. Si tous les nœuds sont terminaux, aller à l'étape 3, sinon aller à l'étape 2-3. Un nœud est terminal s'il vérifie l'identité $T_k\Psi^* = 0$ ou $T_k\Psi^* = 1$.

Etape 2-3 : transformer les nœuds non actifs en nœuds actifs ou terminaux. Aller à l'étape 3 quand l'entrée k ne contient que des nœuds terminaux, sinon aller à l'étape 2-4 pour traiter les nœuds actifs.

Etape 2-4 : à l'apparition d'un littéral L dans l'identité associée au nœud actif T_k, créer à l'étape k+1 deux nœuds : un nœud à gauche $L^0\ T_{k+1}\Psi^*$ et un nœud à droite $L^1\ T_{k+1}\Psi^*$ et revenir à l'étape 2-2.

Détermination des implicants

La performance de l'algorithme dépend principalement du choix du type de parcours de l'arbre binaire de la figure 3 .En effet pour déterminer les implicants nous pouvons utiliser un parcours en largeur (voir fig. 4.a) ou un parcours en profondeur que nous appellerons «parcours droite gauche» (voir fig. 4-b).

Dans le premier type de parcours nous traitons 2^k nœuds à l'étape k. De plus il est nécessaire d'avoir comme information les 2^{k-1} nœuds précédents. Pour des AdD dans lesquels les nœuds terminaux n'apparaissent qu'à une certaine profondeur de l'arbre, le nombre de nœuds à examiner et à conserver devient alors élevé et peut ainsi induire des problèmes de capacité mémoire.

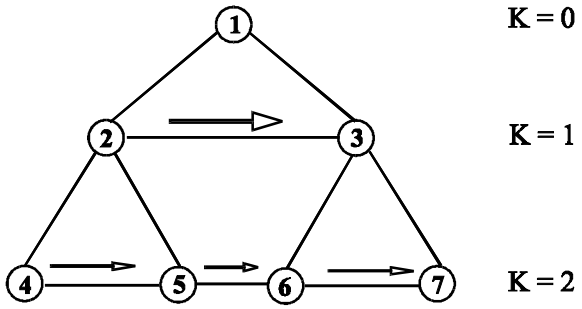

Figure 4-a : *Parcours en largeur*

Dans le second type de parcours, nous traitons le nœud racine puis tous les nœuds se trouvant dans la branche de droite jusqu'à aboutir à un nœud terminal. Nous remontons ensuite jusqu'au premier nœud ayant un nœud adjacent, et en partant de ce dernier nous réalisons la même opération. Nous recommençons cette procédure jusqu'à traitement de tous les nœuds.

L'intérêt principal de ce type de parcours est l'élimination de certains implicants redondants avant l'étape de réduction.

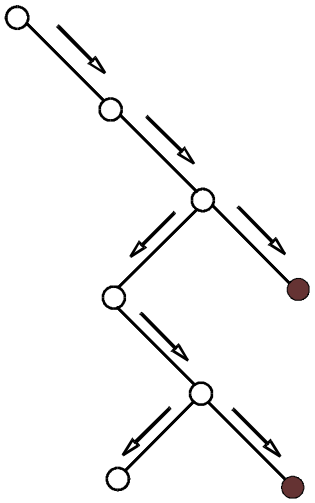

Figure 4-b : *Parcours droite gauche*

Ainsi par exemple, dans le cas de la figure 4.c dans laquelle la présence de littéraux est notée P et l'absence de littéraux A, pour tous les chemins du type AAPAPAAA ayant abouti à un nœud terminal M tel que $T_k\Psi^* = 0$, nous pouvons remonter dans l'arbre jusqu'au premier P rencontré : le nœud S. Tous les nœuds situés à la gauche de S sont : soit des nœuds aboutissant à la relation $T_k\Psi^* = 1$**,** donc sans intérêt, soit des nœuds aboutissant à la relation $T_k\Psi^* = 0$, ce qui donne des implicants redondants à ceux obtenus au nœud M. Il est donc inutile de traiter ces nœuds. Dans la figure 4.c ces nœuds se trouvent dans la partie hachurée.

Etape 3 : Détermination des implicants premiers

Pour ce faire, nous regroupons tous les littéraux L_j ayant l'exposant i_j=0 pour chaque opérateur qui satisfait l'identité (1). Chaque groupe de littéraux obtenu de cette manière est un implicant.

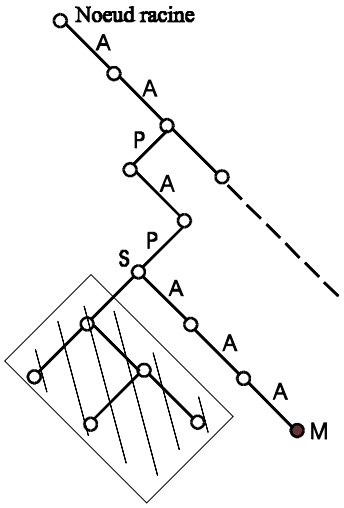

Figure 4-c : *Illustration du parcours droite gauche*

Nous procédons ensuite à une réduction afin d'éliminer les implicants redondants. Un implicant P_i est éliminé s'il existe un implicant $P_j \neq P_i$ tel que $P_j \subset P_i$. La liste des implicants restants constitue l'ensemble des implicants premiers de l'arbre de défaillance étudié.

3 APPLICATIONS

Notre algorithme a été appliqué à deux exemples d'arbres de défaillance.

1erexemple : AdD 1 de la figure 1-a.

La logique de cet AdD contient 11 portes et 9 évènements élémentaires. Son traitement a conduit aux résultats suivants :

- Nombre de nœuds étudiés = 77
- Nombre d'implicants obtenus = 8
- Nombre d'implicants premiers obtenus = 8
- Liste des implicants premiers:

{A}, {B}, {C,D}, {D,E,$\overline{F}$}, {D,$\overline{F}$,G}, {D,$\overline{F}$,H}, {D ,$\overline{F}$, I}, {C,$\overline{E}$, F,$\overline{G}$,$\overline{H}$,$\overline{I}$}

Notons que pour cet exemple tous les implicants obtenus sont premiers. L'étape de réduction n'est donc pas nécessaire.

2^{éme} exemple : AdD 2 de la figure 5.

La logique de cet AdD contient 16 portes et 19 évènements élémentaires. Son traitement a conduit aux résultats suivants :

- Nombre de nœuds étudiés = 2447
- Nombre d'implicants obtenus = 165
- Nombre d'implicants premiers obtenus = 69

Pour éliminer les implicants redondants, nous passons par une phase de réduction qui nécessite (165 *164)/2 = 13530 comparaisons.

Après réduction nous obtenons :

- 3 implicants premiers d'ordre 1 : {1}, {2}, {3}
- 64 implicants premiers d'ordre 2 :

{14,17}, {14,19}, {14,15}, {14,13}, {14,11}, {14,9}, {14,8}, {14,7}, {12,17}, {12,19}, {12,15}, {12,13}, {12,11}, {12,9}, {12,8}, {12,7}, {10,17}, {10,19}, {10,15}, {10,13}, {10,11}, {10,9}, {10,8}, {10,7}, {6,17}, {6,19}, {6,15}, { 6,13}, {6,11}, {6,9}, {6,8}, {6,7}, {5,17}, {5,19}, {5,15}, {5,13}, {5,11}, {5,9}, {5,8}, {5,7}, {4,17}, {4,19}, {4,15}, {4,13}, {4,11}, {4,9}, {4,8}, {4,7}, {16,17}, {16,19}, {16,15}, {16,13}, {16,11}, {16,9}, {16,8}, {16,7}, {18,17}, {18,19}, {18,15}, {18,13}, {18,11}, {18,9}, {18,8}, {18,7}

- 2 implicants premiers d'ordre 7 :

{16,$\overline{4}$,$\overline{5}$,$\overline{6}$,$\overline{14}$,$\overline{12}$,$\overline{10}$} , {18,$\overline{4}$,$\overline{5}$,$\overline{6}$,$\overline{14}$,$\overline{12}$,$\overline{10}$}

4 COMPARAISON AVEC L'ALGORITHME DE KUMAMOTO [4]

Afin d'évaluer les performances de notre algorithme, nous avons comparé nos résultats à ceux obtenus par l'algorithme de Kumamoto. Ce dernier qui constitue une référence dans le domaine des systèmes non cohérents a été appliqué aux deux arbres de défaillance AdD 1 et AdD 2. Nous obtenons les résultats suivants :

1er cas : AdD 1

- Nombre de nœuds étudiés = 77
- Nombre d'implicants obtenus = 8
- Nombre d'implicants premiers obtenus = 8

2^{éme} cas : AdD 2

- Nombre de nœuds étudiés = 5003
- Nombre d'implicants obtenus = 1557
- Nombre d'implicants premiers obtenus = 69

La phase de réduction nécessite dans ce cas (1557*1556)/2 = 1211346 comparaisons.

5 DISCUSSION DES RESULTATS

La comparaison des résultats obtenus par les deux algorithmes montre que dans l'exemple de l'AdD1 notre algorithme donne les mêmes résultats que celui de Kumamoto car il n'y a pas de phase de réduction. Néanmoins cet exemple montre que notre algorithme fonctionne correctement. Dans l'exemple de l'AdD2 nous constatons que notre algorithme donne de meilleurs résultats. En effet d'une part, le nombre de nœuds étudiés est réduit de moitié et le nombre d'implicants obtenus avant réduction est réduit d'un facteur 10. Ce qui permet un gain appréciable en espace mémoire. D'autre part, le temps d'exécution a ainsi pu être réduit de moitié.

6 CONCLUSION

L'algorithme que nous avons décrit dans les sections précédentes ainsi que celui de Kumamoto ont été implémentés en langage C sur un micro-ordinateur de type «pentium4». Comme nous l'avons montré au paragraphe 5, leur application à deux types d'arbres de défaillance montre que notre algorithme donne de meilleurs résultats notamment pour les architectures d'arbre de défaillance, qui présentent un grand nombre d'évènements répétés.

Notre algorithme pourrait être amélioré en l'associant à un autre algorithme que nous avons développé lors de travaux antérieurs [8]. Très sommairement, ce dernier consiste à diviser la classe des implicants en deux classes, à savoir une classe qui contient des évènements répétés et une autre qui n'en contient pas. La réduction ne portera alors que sur la première classe et le temps d'exécution sera ainsi diminué.

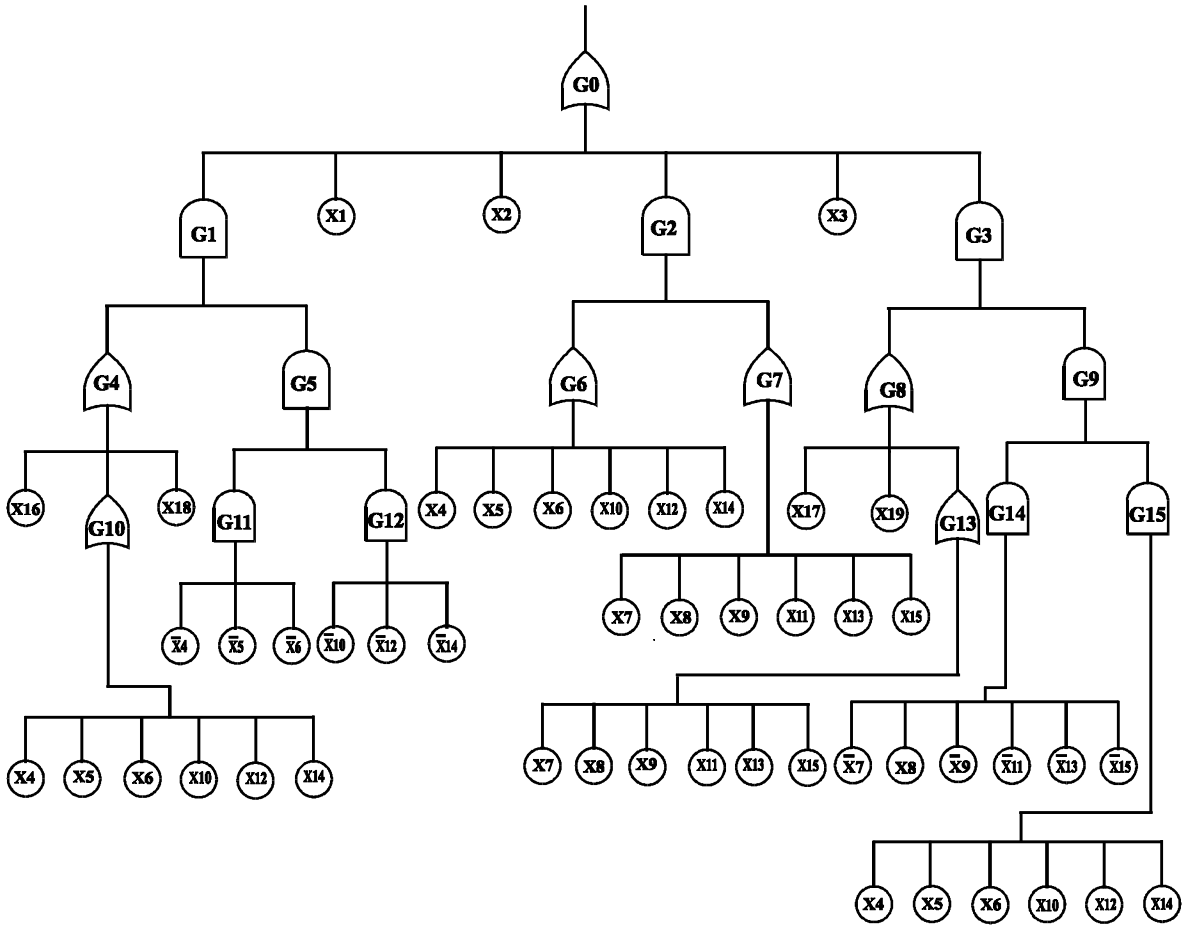

Figure 5 : *AdD 2*

REFERENCES BIBLIOGRAPHIQUES

[1] US Nuclear Regulatory Commission "Reactor Safety Study, An assessment of accident risk US commercial nuclear power plants" WASH-1400, NUREG-75/014, 1975.

[2] W.S. LEE, D.L.GROSH, F.A. TILLMAN, C.H. LIE "Fault tree analysis, methods and applications - a review", IEEE Trans. On Reliability, vol R-34, août 1985.

[3] J.B. FUSSELL, W.E. VESELY "A new methodology for obtaining cuts sets for fault trees", Trans. Am. Nucl. Soc.,vol 15, juin 1972.

[4] H. KUMAMOTO, E.J. HENLEY "Top-down Algorithm for Obtaining Prime Implicant Sets of Non-Coherent Fault Trees", IEEE Trans. On Reliability, vol R-27, octobre 1978.

[5] A.RAUZY "New algorithms for fault tree analysis", Reliability Engineering & System Safety, vol 40, 1993.

[6] O. COUDERT, J.C.MADRE "MetaPrime: An Interactive Fault-Tree Analyser", IEEE Trans. On Reliability, vol 43, mars 1994.

[7] J.A. CARRASCO, V. SUNE "An algorithm to find minimal cuts of coherent fault tree with even-classes, using a decision tree, IEEE Trans. On Reliability, vol 48, mars 1999.

[8] N. LIMNIOS, R. ZIANI, "An Algorithm for Reducing Cut Sets in Fault-Tree Analysis", IEEE Trans. On Reliability, vol R-27, décembre 1986.