RAIRO. INFORMATIQUE THÉORIQUE ET APPLICATIONS

OLIVIER COUDERT
JEAN-CHRISTOPHE MADRE

Une approche intentionnelle du calcul des implicants premiers et essentiels des fonctions booléennes

RAIRO. Informatique théorique et applications, tome 28, n° 2 (1994), p. 125-149

<http://www.numdam.org/item?id=ITA_1994__28_2_125_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 28, n° 2, 1994, p. 125 à 149)

UNE APPROCHE INTENTIONNELLE DU CALCUL DES IMPLICANTS PREMIERS ET ESSENTIELS DES FONCTIONS BOOLÉENNES (*)

par Olivier COUDERT (1) et Jean-Christophe MADRE (1)

Communiqué par J.-E. PIN

Résumé. – *Toutes les techniques de calcul d'implicants premiers de fonctions booléennes qui ont été développées jusqu'à présent possèdent une complexité directement liée au nombre d'implicants premiers des fonctions. En pratique, ces techniques sont difficilement applicables à des fonctions booléennes ayant plus de 20 000 implicants premiers. Nous proposons ici un algorithme fondamentalement différent, qui calcule une représentation* intentionnelle *de l'ensemble des implicants premiers et essentiels. Cette représentation, que nous appelons* méta-produit, *permet de résoudre toute une gamme de problèmes complexes avec une complexité polynomiale par rapport à la taille des méta-produits. Comme il n'y a aucune relation entre la taille d'un méta-produit et le nombre d'implicants premiers qu'il dénote, l'algorithme que nous présentons ici possède la propriété remarquable d'avoir une complexité qui ne* dépend absolument pas *du nombre d'implicants premiers de la fonction traitée. Cette nouvelle approche est environ 1 000 fois plus rapide que les techniques actuellement utilisées, et permet de traiter des fonctions inabordables par les approches classiques, par exemple des fonctions ayant plus de* 10^{20} *implicants premiers.*

Abstract. – *All prime implicant computation procedures developed in the past have computational costs directly related to the number of prime implicants of the Boolean function under treatment. This makes these techniques difficult to apply to functions that have more than about 20 000 prime implicants. We propose here a new algorithm that has a completely different behavior. This algorithm computes an* implicit *representation of the set of prime implicants. This implicit representation of sets of products, called the* metaproduct *representation, allows us to solve a wide variety of problems involving sets of product with complexities that are polynomial with respect to the size of the metaproducts. Since there is no relation between the size of a metaproduct and the number of products of the set it represents, the prime implicant computation procedure presented in this paper has a computational cost that is no longer related to the number of prime implicants of the Boolean function under treatment. This new algorithm has been shown to be about 1 000 times faster than any other known procedure, and allows us to deal with Boolean functions that are out of reach of other techniques, for instance functions that have more than* 10^{20} *prime implicants.*

(*) Reçu le 14 décembre 1992.
(1) Centre de Recherche Bull, rue Jean Jaurès, 78340 Les Clayes-sous-bois, France.

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/94/02/$4.00/

INTRODUCTION

Le problème abordé ici, formalisé dans la Section 1.3, est le calcul des implicants premiers et des implicants premiers essentiels des fonctions booléennes. Ce problème est fondamental dans de nombreuses applications mathématiques et informatiques, par exemple la démonstration automatique, la synthèse et l'optimisation de circuits VLSI, l'étude de fiabilité et le diagnostic de systèmes.

Historique

En 1952, Quine note que le problème de trouver une procédure de réduction des formules propositionnelles en leurs plus simples formes équivalentes est extrêmement complexe, malgré la nature «simple» de la logique propositionnelle [30]. Quine s'intéresse alors au cas où les formules propositionnelles sont exprimées en forme normale disjonctive, aussi appellée somme de produits. Il cherche, à partir d'une formule f exprimée en forme normale disjonctive, à trouver une forme normale disjonctive la plus simple possible et équivalente à la formule f originale. La notion clé qu'il introduit est celle d'*implicant premier*, qu'il définit ainsi [32]:

«[A] prime implicant of a formula f [is] a fundamental formula that logically implies f but ceases to when deprived of any one literal»

Un implicant premier d'une formule f est donc une conjonction de littéraux qui implique f, mais qui cesse de l'impliquer dès qu'on lui retire un de ses littéraux. Quine montre d'abord qu'une formule est équivalente à la somme de tous ses implicants premiers, et que toute expression minimale d'une formule en forme normale disjonctive est faite d'implicants premiers [30, 31]. Cependant, le nombre d'implicants premiers d'une formule est exponentiel dans le pire cas. Par exemple, la formule $(x_1 \oplus \ldots \oplus x_n)$ à n variables possède 2^n implicants premiers. Quine démontre en 1959 que ce nombre peut même arbitrairement excéder le nombre d'interprétations pour lesquelles la formule s'évalue à 1 [32]. Au pire, une formule propositionnelle de n variables peut posséder jusqu'à $\Omega(3^n/n)$ implicants premiers.

Traditionnellement, et suivant en cela les travaux de Quine, les implicants premiers ont été utilisés pour minimiser des formules propositionnelles en vue de l'optimisation ou de la synthèse de circuits digitaux [24, 31, 4, 19, 20, 35, 5, 18, 39]. Ces travaux ont conduit à définir la notion d'*implicant premier essentiel*, aussi appelé *implicant essentiel*, c'est-à-dire un implicant premier qui apparaît *obligatoirement* dans toute somme de produits

minimale d'une formule, parce qu'il est le seul implicant premier à couvrir une des interprétations valuant la formule à 1.

Le calcul des implicants premiers est également au cœur des techniques d'analyse des *arbres de défaillance*, techniques introduites au début des années soixante pour l'étude de la fiabilité de systèmes complexes, tels que ceux conçus en avionique, dans le nucléaire, la chimie et la médecine. Un arbre de défaillance modélise logiquement les conditions de disfonctionnement du système considéré en fonction d'événements élémentaires [17]. L'arbre de défaillance décrit donc une fonction booléenne, et le calcul de ses implicants premiers, appelés dans cette communauté *coupures minimales*, permet l'évaluation de la fiabilité du système sous deux aspects [29, 44]. D'abord, l'analyse de la distribution des implicants premiers de l'arbre de défaillance permet d'évaluer qualitativement la fiabilité du système [44]. Ensuite, une étude quantitative de fiabilité est faite en calculant la probabilité de disfonctionnement du système à partir des probabilités d'occurrence des événements élémentaires. Il est aussi souhaitable de pouvoir calculer la contribution de la probabilité de certains ensembles d'implicants premiers à la probabilité globale de disfonctionnement du système [44].

Plus récemment, on s'est aperçu que le calcul des implicants premiers avait de plus larges applications, en particulier en intelligence artificielle et en démonstration automatique. L'utilisation des implicants premiers pour la preuve automatique de théorèmes est introduite en 1970 par J. R. Slage [40, 41]. Puis R. Reiter et J. de Kleer soulignent en 1987 leur importance dans les systèmes de maintien du raisonnement [34, 21, 26]. Ils apparaissent aussi comme une étape fondamentale dans les systèmes de diagnostic pour le traitement des fautes multiples [33, 22].

État de l'art

Depuis l'introduction du concept d'implicant premier, de nombreux travaux ont été menés pour développer des procédures efficaces d'obtention des implicants premiers d'une formule propositionnelle. On peut y distinguer : la méthode du consensus [2], qui découle d'une restriction du principe de résolution [15, 36] ; la technique des tables de Karnaugh [4], bien connue des concepteurs de circuits ; la méthode de Quine-McCluskey [24, 31, 4], où tous les implicants premiers d'une formule sont calculés pour en dériver une somme minimale ; la technique de résolution sémantique [41] introduite par J. R. Slage ; la méthode de Tison [43], postérieurement prouvée correcte dans [23]. Les techniques de calcul des implicants premiers essentiels utilisent

celles des implicants premiers. Elles sont encore plus complexes, et optimisées selon une classification des formules propositionnelles [37].

Malgré les efforts mis sur le calcul des implicants premiers, les algorithmes existant dérivent peu ou prou de l'algorithme original de Quine, et les améliorations, généralement du second ordre, sont obtenues au prix d'optimisations complexes et d'heuristiques adaptées à des applications particulières. De plus, tous ces algorithmes consistent à calculer les implicants premiers un par un, ce qui fait que la complexité de ces algorithmes est directement liée au nombre d'implicants premiers de la formule propositionnelle considérée. En pratique, les techniques actuelles sont limitées au traitement de fonctions ayant environ 20 000 implicants premiers.

Notre approche

Nous proposons ici un algorithme de calcul *en intention* de l'ensemble des implicants premiers et de l'ensemble des implicants premiers essentiels d'une fonction booléenne. Cet algorithme possède la propriété remarquable d'avoir une complexité qui ne *dépend absolument pas* du nombre d'implicants premiers de la fonction traitée. Cette nouvelle approche, testée sur des cas réels, est environ 1 000 fois plus rapide que les techniques actuellement utilisées, et permet de traiter des fonctions ayant plus de 10^{20} implicants premiers.

1. DÉFINITION FORMELLE DU PROBLÈME

Nous présentons ici la logique propositionnelle quantifiée, qui nous servira pour définir les problèmes traités et pour exprimer les solutions de ces problèmes. Puis nous explicitons la relation existant entre formules propositionnelles et fonctions booléennes. Enfin nous définissons formellement les notions d'implicant premier et d'implicant premier essentiel d'une fonction booléenne.

1.1. Formules propositionnelles quantifiées

On considère un ensemble $V_n = \{x_1, \ldots, x_n\}$ de variables propositionnelles, et l'ensemble F_n des formules propositionnelles quantifiées construites à partir de V_n et des symboles logiques classiques $\{\neg, \vee, \exists\}$:

$$\langle \textit{Formule} \rangle ::= 0 \mid 1 \mid \langle \textit{Variable} \rangle$$
$$::= \neg \langle \textit{Formule} \rangle$$

$$::= (\langle \textit{Formule} \rangle \vee \langle \textit{Formule} \rangle)$$
$$::= \exists \langle \textit{Variable} \rangle \langle \textit{Formule} \rangle$$
$$\langle \textit{Variable} \rangle ::= x_1 \mid x_2 \mid \ldots \mid x_n$$

Pour alléger l'écriture, on omettra les parenthèses lorsque celà ne crée pas d'ambiguïté : on supposera pour cela que la négation « $\neg$ » possède la précédence la plus forte, suivie de la disjonction « $\vee$ », puis de la quantification existentielle « $\exists$ ». On introduit les symboles $\{ \wedge, \Rightarrow, \Leftrightarrow, \oplus, \forall \}$, qui se réécrivent en fonction de ceux introduits précédemment par les règles suivantes, où f et g sont des formules propositionnelles quantifiées et x une variable :

$$(\forall x\, f) \rightarrow \neg (\exists x \neg f)$$
$$(f \wedge g) \rightarrow \neg (\neg f \vee \neg g)$$
$$(f \Rightarrow g) \rightarrow (\neg f \vee g)$$
$$(f \Leftrightarrow g) \rightarrow (f \Rightarrow g) \wedge (g \Rightarrow f)$$
$$(f \oplus g) \rightarrow \neg (f \Leftrightarrow g)$$

On suppose connue du lecteur la notion d'*occurrence*. L'occurrence d'une variable x dans une formule f est dite *libre* si et seulement si cette occurrence n'apparaît pas dans le champ d'un quantificateur portant sur x. L'occurrence non libre d'une variable est dite *liée*. Par exemple, dans la formule $(\forall y (x \wedge y \wedge z) \Rightarrow (\exists x (x \Leftrightarrow y)))$, z est libre, les occurrences de y sont liées, et x possède une occurrence libre (la première) et une occurrence liée (la seconde). On notera $f[x \leftarrow g]$ la formule obtenue en substituant toute occurrence libre de la variable x dans la formle f par la formule g.

1.2. Fonctions booléennes

Une *interprétation* i est une fonction de V_n dans $\{0, 1\}$. Une interprétation s'étend en une fonction i_* de F_n dans $\{0, 1\}$ par les égalités suivantes :

$$\begin{aligned}
i_*(1) &= 1 \\
i_*(0) &= 0 \\
i_*(x_k) &= i(x_k) \quad \text{ssi} \quad x_k \in V_n \\
i_*(\neg f) &= 1 \quad \text{ssi} \quad i_*(f) = 0 \\
i_*(f \vee g) &= 1 \quad \text{ssi} \quad i_*(f) = 1 \quad \text{ou} \quad i_*(g) = 1 \\
i_*(\exists x\, f) &= 1 \quad \text{ssi} \quad i_*(f[x \leftarrow 0]) = 1 \quad \text{ou} \quad i_*(f[x \leftarrow 1]) = 1
\end{aligned}$$

On confondra par la suite la fonction i et son extension i_*. Une formule propositionnelle f représente une unique fonction booléenne ff de $\{0, 1\}^n$ dans $\{0, 1\}$, définie par

$$ff(v_1, \ldots, v_n) = i_v(f),$$

où l'interprétation i_v est telle que $i_v(x_k) = v_k$. De par cette correspondance non ambigüe entre les formules propositionnelles de F_n et les fonctions booléennes de $(\{0, 1\}^n \to \{0, 1\})$, on confondra une formule avec la fonction qu'elle représente. Une formule désignant la fonction constante 1 sera appelée une tautologie. Deux formules désignant la même fonction seront dites équivalentes. Dans la suite, l'égalité sur les formules sera considérée modulo la relation d'équivalence.

1.3. Produit, implicant premier et essentiel

Un *littéral* est une variable propositionnelle x_k ou sa négation, qu'on notera aussi $\bar{x}_k$. L'ensemble P_n des *produits* construits à partir de l'ensemble des variables propositionnelles V_n est l'ensemble $\{x_1, \bar{x}_1, \varepsilon\} \times \cdots \times \{x_n, \bar{x}_n, \varepsilon\}$, où ε est la chaîne vide. Un *produit* est un élément de P_n, et sera interprété comme la conjonction logique des littéraux le composant. Par exemple, le produit $x_1 \bar{x}_2 x_4$ sera interprété comme la formule $(x_1 \wedge \neg x_2 \wedge x_4)$. Un produit p *contient* un produit p', ce qu'on notera $(p' \subseteqq p)$, si et seulement si la formule $(p' \Rightarrow p)$ est une tautologie, La relation « $\subseteqq$ » est un ordre partiel sur P_n.

Un produit p est un *implicant* d'une fonction f si et seulement si la formule $(p \Rightarrow f)$ est une tautologie. Un produit p est un *implicant premier* de f si et seulement si p est un implicant de f, et il n'existe pas d'autre implicant de f qui contienne p. Autrement dit, un produit p est un implicant premier de f si et seulement si p est un élément maximal de l'ensemble des implicants de f vis-à-vis de l'ordre partiel « $\subseteqq$ ». Un produit p est un *implicant premier essentiel* de f si et seulement si p est un implicant premier de f, et la fonction obtenue en sommant tous les autres implicants premiers de f est différente de f.

2. GRAPHES DE DÉCISION BINAIRES

Le calcul des implicants premiers que nous présenterons plus loin est basé sur la manipulation de fonctions booléennes. Il nous faut donc un outil de manipulation formelle des formules propositionnelles suffisamment puissant

pour mener ces calculs à bien. Nous utiliserons les graphes de décision binaires (BDDs), introduits en 1986 par R. E. Bryant [7].

2.1. Arbre de Shannon et BDD

L'expansion de Shannon d'une formule f par rapport à la variable propositionnelle x_k est le couple de formules $(f[x_k \leftarrow 0], f[x_k \leftarrow 1])$ [1]. L'expansion de Shannon possède la propriété suivante :

$$f = (\neg x_k \wedge f[x_k \leftarrow 0]) \vee (x_k \wedge f[x_k \leftarrow 1])$$

De plus l'expansion de Shannon est l'unique solution (modulo l'équivalence) de cette décomposition de f. On notera $\Delta(x_k, L, H)$ la fonction $(\neg x_k \wedge L) \vee (x_k \wedge H)$.

L'itération de la décomposition de Shannon d'une formule f sur les variables $x_1, \ldots, x_n$ produit un arbre unique modulo l'ordre dans lequel les variables sont choisies pour la décomposition. Cet arbre, appelé l'*arbre de Shannon* de f, possède $2^n - 1$ nœuds internes, et ses feuilles sont les constantes 0 et 1. La figure 1 montre l'arbre de Shannon de la formule $(x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \Leftrightarrow x_4))$ de F_4. L'évaluation de cette formule pour une interprétation de ses variables peut être faite en suivant le chemin, du sommet de l'arbre à une feuille, défini par cette interprétation. A chaque nœud, le chemin suit la branche gauche si la variable qui est associée au nœud est valuée à 0, et la branche droite si la variable est valuée à 1. Le chemin défini par l'interprétation $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, $x_4 = 0$ est donné en pointillé Figure 1.

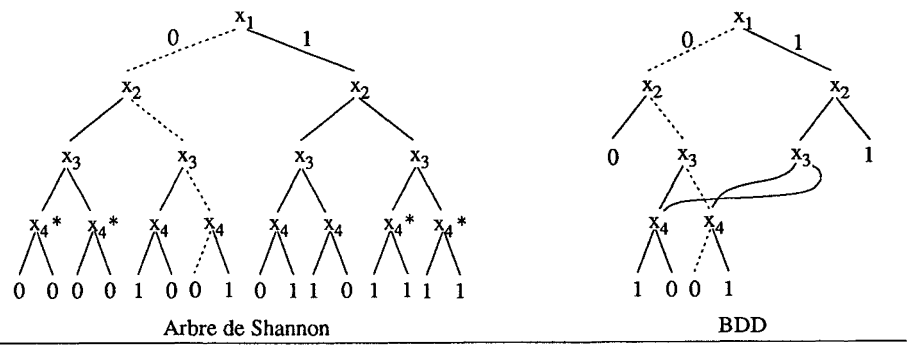

Figure 1. – **Arbre de Shannon et BDD de la formule** $(x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \Leftrightarrow x_4))$.

Le *graphe de décision binaire* (BDD pour *Binary Decision Diagram*) d'une formule f est une représentation compactée de son arbre de Shannon [7]. Le BDD est obtenu en utilisant deux règles de compaction. La première règle consiste à éliminer tout nœud ayant deux sous-arbres isomorphes en dérivant l'arc aboutissant à ce nœud vers l'un des sous-arbres, car un tel nœud n'apporte aucune information lors de l'évaluation de la formule. Ainsi les nœuds de la Figure 1 marqués par un « * » sont inutiles. La seconde règle consiste à identifier tous les sous-arbres isomorphes, ce qui transforme l'arbre en un graphe acyclique orienté, appelé le BDD de la formule f. Le BDD de la formule $(x_1 \wedge (x_3 \oplus x_4)) \vee (x_2 \wedge (x_3 \Leftrightarrow x_4))$ est donné Figure 1. Ces deux règles de compaction font que, dans de nombreux cas, la taille (*i.e.*, le nombre de nœuds) du BDD est considérablement inférieure à celle de l'arbre de Shannon qu'il représente.

2.2. Le problème de l'ordonnancement des variables

Les BDDs sont une représentation *canonique* modulo l'ordre des variables, et extrêmement compacte. Cependant, la structure et la taille d'un BDD dépendent étroitement de l'ordre choisi sur les variables lors de la décomposition. Il existe des formules admettant un BDD de taille polynomiale pour un certain ordre, et un BDD de taille exponentielle pour un autre ordre. Par exemple [11], la formule $((x_1 \Leftrightarrow x_{2n}) \wedge (x_2 \Leftrightarrow x_{2n-1}) \wedge \ldots \wedge (x_n \Leftrightarrow x_{n+1}))$, qui possède $2n$ variables, a un BDD de taille $3n$ avec l'ordre $x_1 < x_{2n} < x_2 < x_{2n-1} < \ldots < x_{n-1} < x_n$, et un BDD de taille $3(2^n - 1)$ avec l'ordre $x_1 < x_2 < \ldots < x_{2n}$. De plus, il existe des fonctions qui n'admettent pas de BDD de taille polynomiale quel que soit l'ordre de décomposition considéré [8].

Caculer un ordonnancement optimal des variables pour une formule f donnée, c'est-à-dire un ordonnancement pour lequel le BDD de f est de taille minimale, est un problème NP-difficile. Le meilleur algorithme connu qui calcule un ordre optimal [16] a une complexité en $O(n^2 3^n)$, ce qui interdit son utilisation pour $n > 10$. C'est pourquoi de nombreux travaux [27, 28, 29] ont porté sur le développement d'heuristiques exploitant la structure de la formule pour produire un bon ordonnancement des variables, c'est-à-dire un ordonnancement pour lequel le BDD a une taille raisonnable, s'il en existe un.

2.3. Manipulations des BDDs

Les opérateurs booléens classiques $\{ \vee, \wedge, \Rightarrow, \Leftrightarrow, \oplus \}$ peuvent être directement évalués avec une complexité quadratique sur les BDDs construits avec le même ordonnancement [7]. La négation d'un BDD se calcule linéairement [7], ou même en $O(1)$ avec les *graphes de décision typés* (TDG pour *Typed Decision Graphs*), [3, 25], qui sont une amélioration des BDDs. Tous ces algorithmes évaluant une connection logique sont décrits par les mêmes règles récursives, seules les récursions terminales étant différentes.

Cette remarquable uniformité des algorithmes de combinaison, tant pour leur définition que pour leur comportement, s'explique par la structure de graphe acyclique orienté des BDDs. Cette structure nous permet de définir des algorithmes extrêmement efficaces pour répondre à certains problèmes non triviaux, ce qui n'est pas toujours le cas avec d'autres représentations. Par exemple, compter le nombre d'interprétations valuant une fonction f à 1, ce qui correspond aussi à compter le nombre d'éléments de l'ensemble désigné par f, s'effectue avec une complexité linéaire par rapport à la taille du BDD de f, alors que ce même problème est NP-complet sur la forme normale disjonctive et sur la forme de Reed-Muller [11].

L'uniformité des algorithmes de manipulation ne se retrouve pas chez les autres représentations connues des fonctions booléennes. De plus, les algorithmes de combinaison pour ces représentations ont des complexités très souvent supérieures à celles offertes par les algorithmes travaillant sur les BDDs, voire exponentielles [11]. Le tableau 1 compare différentes manipulations sur les formes normales disjonctives (FND), la forme de Reed-Muller, et les BDDs [11]. Dans cette table, $|f|$ représente la taille de la représentation de la formule f, c'est-à-dire le nombre d'occurrences de littéraux pour la forme normale disjonctive et la forme de Reed-Muller, et le nombre de nœuds pour les BDDs. Les lignes « $\neg f$ » à « $(\forall x f)$ » correspondent au calcul de la représentation de l'expression donnée en supposant que les formules f et g sont déjà normalisées. Par exemple, la complexité donnée par la ligne « $(f \Rightarrow g)$ » et la colonne « **FND** » est celle du calcul de la forme normale disjonctive de la formule $(f \Rightarrow g)$ à partir des formes normales disjonctives de f et g. Les lignes $(\exists \vec{x} f)$ et $(\forall \vec{x} f)$ correspondent à l'élimination d'un nombre à priori quelconque de variables quantifiées existentiellement ou universellement. La ligne « taille max. » donne la taille maximum de la représentation d'une formule à n variables. La ligne « répartition » donne la taille majoritaire des représentations des 2^{2^n} fonctions à n variables. Ainsi, « $> 2^n/n$ p.p. » signifie « de taille supérieure à $2^n/n$ presque partout ».

TABLEAU 1

Comparaison des sommes de produits, forme de Reed-Muller, et TDG.

Opération	FND	Reed-Muller	TDG
Satisfiabilité	$O(1)$	$O(1)$	$O(1)$
Tautologie	CoNP-complet	$O(1)$	$O(1)$
$\neg f$	$O(\sqrt{\lvert f\rvert}^{\sqrt{\lvert f\rvert}+1})$	$O(1)$	$O(1)$
$(f \Rightarrow g)$	$O(\sqrt{\lvert f\rvert+\lvert g\rvert}^{\sqrt{\lvert f\rvert+\lvert g\rvert}+1})$	$O(\lvert f\rvert \times \lvert g\rvert \times \log(\lvert f\rvert \times \lvert g\rvert))$	$O(\lvert f\rvert \times \lvert g\rvert)$
$(f \Leftrightarrow g)$	$O(\sqrt{\lvert f\rvert+\lvert g\rvert}^{\sqrt{\lvert f\rvert+\lvert g\rvert}+1})$	$O((\lvert f\rvert+\lvert g\rvert)\log(\lvert f\rvert+\lvert g\rvert))$	$O(\lvert f\rvert \times \lvert g\rvert)$
$(f \oplus g)$	$O(\sqrt{\lvert f\rvert+\lvert g\rvert}^{\sqrt{\lvert f\rvert+\lvert g\rvert}+1})$	$O((\lvert f\rvert+\lvert g\rvert)\log(\lvert f\rvert+\lvert g\rvert))$	$O(\lvert f\rvert \times \lvert g\rvert)$
$(f \wedge g)$	$O(\lvert f\rvert \times \lvert g\rvert)$	$O(\lvert f\rvert \times \lvert g\rvert \times \log(\lvert f\rvert \times \lvert g\rvert))$	$O(\lvert f\rvert \times \lvert g\rvert)$
$(f \vee g)$	$O(\lvert f\rvert+\lvert g\rvert)$	$O(\lvert f\rvert \times \lvert g\rvert \times \log(\lvert f\rvert \times \lvert g\rvert))$	$O(\lvert f\rvert \times \lvert g\rvert)$
$(\exists x_k f)$	$O(\lvert f\rvert)$	$O(\lvert f\rvert^2 \times \log\lvert f\rvert)$	$O(\lvert f\rvert^2)$
$(\exists \vec{x} f)$	$O(\lvert f\rvert)$	$O(\lvert f\rvert \times 2^{\lvert f\rvert})$	$O(2^{\sqrt{\lvert f\rvert}})$
$(\forall x_k f)$	$O(\lvert f\rvert^2)$	$O(\lvert f\rvert^2 \times \log\lvert f\rvert)$	$O(\lvert f\rvert^2)$
$(\forall \vec{x} f)$	$O(2^{\lvert f\rvert})$	$O(\lvert f\rvert \times 2^{\lvert f\rvert})$	$O(2^{\sqrt{\lvert f\rvert}})$
taille max.	$n2^n$	$n\,2^{n-1}$	$O\left(\frac{2^n}{n}\right)$
répartition	$> \frac{2^n}{\log_2 n}$ p.p.	$> \frac{2^n}{\log_2 n}$ p.p.	$> \frac{2^n}{n}$ p.p.

Malgré le problème posé par l'ordonnancement des variables, les BDDs sont actuellement la représentation la plus intéressante des fonctions booléennes. En règle générale, c'est une représentation beaucoup plus dense [11] que les autres représentations des fonctions booléennes, par exemple la forme normale disjonctive, la forme de Reed-Muller [42, 6] (encore appelée somme exclusive de produits), ou la forme canonique de Blake [6].

Il n'y a aucune relation entre la taille d'un BDD et le nombre d'interprétations le valuant à 1, ce qui signifie que de très grands sous-ensembles de $\{0, 1\}^n$ peuvent être représentés par de petits BDDs (ceux-ci représentant leurs fonctions caractéristiques). Par exemple, l'ensemble $\{0, 1\}^n$, qui contient 2^n éléments, est représenté par le BDD « 1 », qui est de taille nulle. C'est pourquoi des opérations extrêmement complexes, exprimées en formules booléennes quantifiées, peuvent être effectuées sur des ensembles représentés par les BDDs de leurs fonctions caractéristiques, avec une complexité *indépendante* du nombre d'éléments des ensembles manipulés.

3. CALCUL SYMBOLIQUE DES IMPLICANTS PREMIERS ET ESSENTIELS

Nous présentons maintenant le nouveau procédé de calcul des implicants premiers et essentiels.

3.1. Ensembles et fonctions caractéristiques

Toute fonction booléenne f de $\{0, 1\}^n$ dans $\{0, 1\}$ représente un sous-ensemble unique de $\{0, 1\}^n$, noté S_f, et défini par

$$S_f = \{ x \in \{0, 1\}^n \mid f(x) = 1 \}.$$

Réciproquement tout sous-ensemble S de $\{0, 1\}^n$ peut être représenté par une unique fonction booléenne de $\{0, 1\}^n$ dans $\{0, 1\}$, notée χ_S, et définie par

$$\chi_S(x) = 1 \qquad \text{ssi} \quad x \in S.$$

La fonction χ_S associée à S est la *fonction caractéristique* de S. Travailler sur les fonctions caractéristiques au lieu des ensembles permet de plonger les manipulations ensemblistes dans le monde des manipulations formelles sur les formules propositionnelles. Ainsi la fonction caractéristique de l'union de deux ensembles est la disjonction de leurs fonctions caractéristiques, celle de leur intersection est la conjonction de leurs fonctions caractéristiques, etc. Si les fonctions caractéristiques sont représentées par leurs BDDs, alors la complexité des opérations ensemblistes n'est plus liée aux nombres d'éléments des ensembles sur lesquels on souhaite travailler, mais uniquement aux tailles des BDDs qui représentent ces ensembles. Ce principe simple, que nous avons introduit avec succès dans le domaine de la vérification formelle de systèmes séquentiels [10, 11], constitue l'une des bases de notre approche.

3.2. Métaproduit

L'idée première est de plonger de manière injective l'ensemble des produits dans un espace booléen, ce qui nous permettra d'exprimer des manipulations complexes sur les produits en utilisant la logique propositionnelle quantifiée. Il y a 3^n produits construits sur n variables propositionnelles. Un espace booléen de dimension $\lceil n \log_2(3) \rceil$ est donc suffisant pour représenter l'ensemble des produits de P_n. Mais si cette dimension est suffisante sur le plan théorique, elle n'est pas la plus intéressante pour l'expression des calculs qui doivent être effectués sur les produits. En effet, on souhaite disposer d'opérations ensemblistes sur les ensembles de produits qui puissent être évaluées à

faible coût sur leur plongement dans l'espace booléen. Nous proposons ici de plonger injectivement P_n dans l'espace $\{0, 1\}^n \times \{0, 1\}^n$ de dimension $2n$, et nous montrerons dans la suite les avantages de notre représentation. Nous définissons d'abord la fonction σ comme suit.

DÉFINITION 1 : *La fonction* σ *est une surjection de* $\{0, 1\}^n \times \{0, 1\}^n$ *dans* P_n *définie par*:

$$\sigma([o_1 \ldots o_n], [s_1 \ldots s_n]) = l_1 \ldots l_n,$$

avec

$$\begin{array}{llll} l_k = x_k & ssi\ o_k = 1 & et & s_k = 1, \\ l_k = \bar{x}_k & ssi\ o_k = 1 & et & s_k = 0, \\ l_k = \varepsilon & ssi\ o_k = 0. & & \end{array}$$

Par exemple, pour $n=4$, $\sigma[1101], [1001]) = x_1 \bar{x}_2 x_4$. Dans la suite, le vecteur de n variables $[o_1 \ldots o_n]$ (respectivement $[s_1 \ldots s_n]$) sera noté o (respectivement s). De même, un vecteur de n variables $[x_1 \ldots x_n]$ sera noté x.

Nous introduisons maintenant la notion de *métaproduit* d'un sous-ensemble de P_n, et donnons le théorème fondamental suivant [12].

DÉFINITION 2 : *Le métaproduit de l'ensemble de produits P inclus dans* P_n *est la fonction caractéristique de l'ensemble* $\sigma^{-1}(P)$, i.e., *de l'ensemble* $\{(o, s) \in \{0, 1\}^n \times \{0, 1\}^n \mid \sigma(o, s) \in P\}$.

THÉORÈME 1 : *Les métaproduits sont une représentation canonique des parties de* P_n.

Preuve: Comme la fonction $\sigma : \{0, 1\}^n \times \{0, 1\}^n \to P_n$ est surjective, sa fonction réciproque $\sigma^{-1} : 2^{P_n} \to 2^{\{0,1\}^n \times \{0,1\}^n}$, *i.e.*, la fonction qui associe à toute partie de P_n son métaproduit, est injective. □

Par exemple, le métaproduit du sous-ensemble $\{x_2 \bar{x}_4, x_1 x_3 x_4, x_1 \bar{x}_2 x_3 \bar{x}_4\}$ de P_4 est la fonction caractéristique de l'ensemble $\{(0101, 0100), (0101, 0110), (0101, 1100), (0101, 1110), (1011, 1011), (1011, 1111), (1111, 0101)\}$. Ceci est illustré par la Figure 2. Disposant des métaproduits, qui plongent injectivement 2^{P_n} dans $2^{\{0,1\}^n \times \{0,1\}^n}$, nous allons pouvoir traduire les manipulations d'ensembles de produits dans le monde de la logique propositionnelle quantifiée.

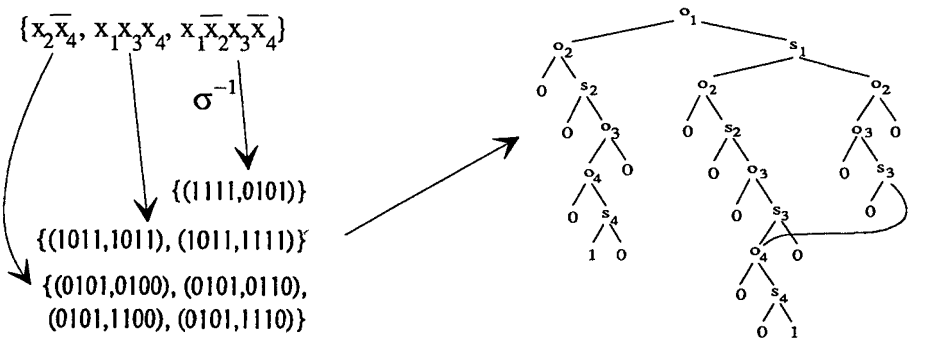

Figure 2. – D'un ensemble de produits à son métaproduit.

3.3. Propriétés des métaproduits

Comme l'ensemble des $\sigma^{-1}(p)$ pour $p \in P_n$ est une partition de $\{0, 1\}^n \times \{0, 1\}^n$, et du fait des propriétés des fonctions caractéristiques et de la définition des métaproduits, on obtient immédiatement les propriétés suivantes. Pour tous métaproduits $\mathscr{P}$ et $\mathscr{P}'$, la fonction $(\mathscr{P} \vee \mathscr{P}')$ est le métaproduit de l'union des ensembles de produits P et P' respectivement représentés par $\mathscr{P}$ et $\mathscr{P}'$; la fonction $(\mathscr{P} \wedge \mathscr{P}')$ est le métaproduit de l'intersection de ces ensembles de produits ; l'ensemble P est inclus dans P' si et seulement si la formule $(\mathscr{P} \Rightarrow \mathscr{P}')$ est une tautologie ; $(\neg\, \mathscr{P})$ est le métaproduit de l'ensemble de produits $(P_n \setminus P)$. De plus, le théorème suivant permet de lier un métaproduit et la fonction obtenue en sommant tous les produits de l'ensemble représenté par ce métaproduit.

THÉORÈME 2 : *Soit $\mathscr{P}$ le métaproduit d'une partie P de P_n, et f la somme des produits de P. On a l'égalité* :

$$f = \lambda s.(\exists\, o\, \mathscr{P}(o, s))$$

Preuve : Par définition, on a :

$$\begin{aligned} f(x) &= (\exists\, p \in P, x \in p) \\ &= (\exists\, o\, \exists\, s\ \ \sigma(o, s) \in P \wedge x \in \sigma(o, s)) \end{aligned}$$

De part la définition de σ, on déduit que $(x \in \sigma(o, s)) \Leftrightarrow (\sigma(o, s) = \sigma(o, x))$ est une tautologie, et donc la dernière égalité s'écrit aussi :

$$\begin{aligned} f(x) &= (\exists o \exists s \ \ \sigma(o, s) \in P \wedge (\sigma(o, s) = \sigma(o, x)) \\ &= (\exists o \ \ \sigma(o, x) \in P) \\ &= (\exists o \ \ \mathscr{P}(o, x)) \quad \square \end{aligned}$$

3.4. BDDs et métaproduits

Un métaproduit est une fonction booléenne, qui peut donc se représenter par un BDD. L'ordonnancement des $2n$ variables $\{o_1, \ldots, o_n, s_1, \ldots, s_n\}$ est alors primordial. Nous choisissons l'ordonnancement suivant :

$$o_{\pi(1)} < o'_{\pi(1)} < s_{\pi(1)} < s'_{\pi(1)} < x_{\pi(1)} <$$

$$\vdots$$

$$o_{\pi(n)} < o'_{\pi(n)} < s_{\pi(n)} < s'_{\pi(n)} < x_{\pi(n)}$$

où π est une permutation des entiers $\{1, \ldots, n\}$. Cet ordonnancement est adapté pour construire les BDDs des métaproduits. La figure 3 montre la forme générale de deux BDDs d'un même métaproduit, le BDD du bas étant obtenu avec le même ordonnancement des variables que celui utilisé pour le BDD du haut, à l'exception de l'échange des variables s_k et o_k. Cet échange permet de faire disparaître m nœuds du BDD, ce qui illustre le fait qu'un bon ordonnancement des variables doit satisfaire $o_k < s_k$.

De plus, cet ordonnancement fait que des opérations aussi diverses et utiles que compter le nombre de produits représentés par le BDD d'un métaproduit, construire le BDD du métaproduit représentant les produits d'un autre métaproduit qui satisfont une certaine propriété (taille du produit, occurrence ou absence d'un littéral, et plus généralement une propriété logique exprimée par un troisième BDD), peuvent être implantées avec une complexité polynomiale par rapport à la taille des BDDs des métaproduits. Par exemple, pour l'étude qualitative de fiabilité à partir d'un arbre de défaillance [44], il est intéressant de pouvoir calculer la distribution des implicants premiers par rapport à leur taille, ou de sélectionner les implicants premiers qui possèdent une certaine taille, ou qui (ne) contiennent (pas) certains littéraux. Avec l'ordonnancement des variables donné plus haut, toutes ces opérations peuvent être faites avec une complexité linéaire par rapport à la taille du BDD

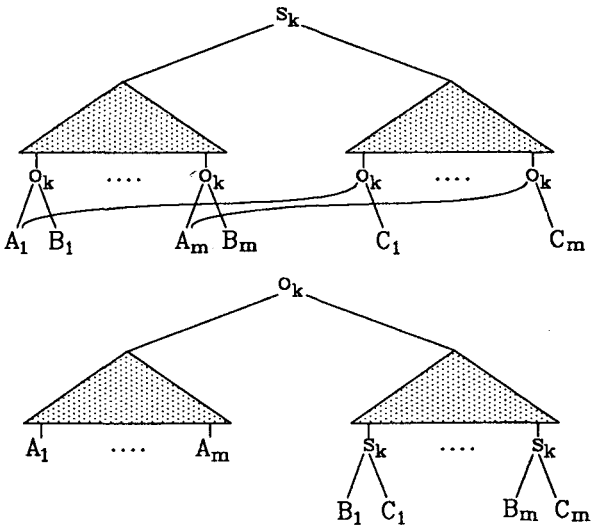

Figure 3. – Conséquence de l'échange des variables o_k et s_k dans un métaproduit.

du métaproduit de l'ensemble des implicants premiers, donc *indépendamment* du nombre d'implicants premiers.

4. CALCUL DU BDD DES MÉTAPRODUITS

Une première approche pour calculer le BDD du métaproduit des implicants premiers d'une fonction booléenne consiste à définir celui-ci en logique propositionnelle quantifiée, puis à directement réduire cette formule en utilisant le calcul sur les BDDs [12]. Cette approche a le mérite d'être immédiatement compréhensible, mais elle n'est pas la plus efficace en terme de coût de calcul. Par exemple, le calcul des implicants premiers basé sur cette approche [12] requiert l'introduction de $4n$ variables, puis des substitutions et des éliminations de variables quantifiées, opérations toutes les deux non polynomiales sur les BDDs.

4.1. Calcul des implicants premiers

Plutôt que d'utiliser la méthode décrite dans [12] qui réécrit des formules quantifiées, nous utilisons les propriétés des implicants premiers et essentiels [5] afin d'obtenir des équations récursives qui permettront d'obtenir

directement le métaproduit des implicants premiers essentiels sans substitution ou élimination de variables.

THÉORÈME 3 : *L'ensemble Prime (f) des implicants premiers de la fonction booléenne f définie de* $\{0, 1\}^n$ *dans* $\{0, 1\}$ *est récursivement décrit par*:

$$\begin{aligned}
&Prime(0) = \varnothing \\
&Prime(1) = \{\varepsilon\} \\
&Prime(f) = Prime(f_{\bar{x}_k} \wedge f_{x_k}) \cup \\
&\qquad \{\bar{x}_k\} \times (Prime(f_{\bar{x}_k}) \setminus Prime(f_{\bar{x}_k} \wedge f_{x_k})) \cup \\
&\qquad \{x_k\} \times (Prime(f_{x_k}) \setminus Prime(f_{\bar{x}_k} \wedge f_{x_k}))
\end{aligned}$$

Il suffit alors de remplacer les ensembles de produits par leur métaproduit, de remplacer les opérations ensemblistes par leurs correspondantes sur les métaproduits, ce qui produit un algorithme récursif qui construit directement le BDD du métaproduit des implicants premiers d'une fonction booléenne à partir de son BDD. Le schéma récursif de ce calcul est illustré Figure 4. Les

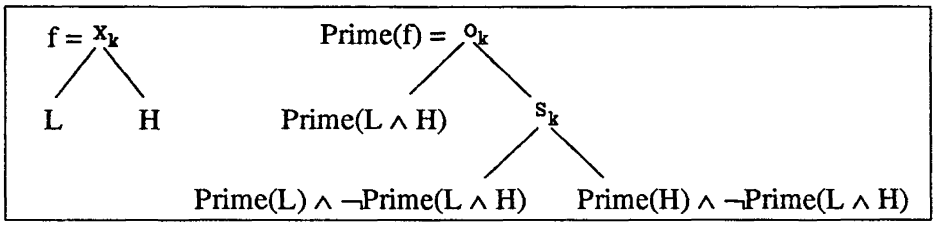

Figure 4. – **Évaluation de** *Prime*.

équations correspondantes s'écrivent:

$$\begin{aligned}
&Prime(0, k) = 0 \\
&Prime(1, k) = \neg \left(\bigvee_{j=1}^{n} o_j \right) \\
&Prime(f, k) = \Delta(o_k, Prime(f_{\bar{x}_k} \wedge f_{x_k}, k+1) \\
&\qquad \Delta(s_k, Prime(f_{\bar{x}_k}, k+1) \wedge \neg Prime(f_{\bar{x}_k} \wedge f_{x_k}, k+1), \\
&\qquad\quad Prime(f_{x_k}, k+1) \wedge \neg Prime(f_{\bar{x}_k} \wedge f_{x_k}, k+1)))
\end{aligned}$$

Il suffit d'évaluer $Prime(f, 1)$ sur le BDD de la fonction f définie sur les variables $\{x_1, \ldots, x_n\}$ pour obtenir le métaproduit de ses implicants premiers.

4.2. Calcul des implicants premiers essentiels

De la même façon que précédemment, le calcul du métaproduit des implicants premiers essentiels peut s'effectuer de deux manières. La première consiste à définir celui-ci par une formule propositionnelle quantifiée, et à la réduire en utilisant les BDDs [12]. Pour les mêmes raisons que celles citées plus haut, cette approche souffre du fait qu'on doit utiliser $4n$ variables et appliquer des substitutions et éliminations de variables qui rendent la réduction très coûteuse. On cherche alors à obtenir des équations récursives qui permettront de construire le BDD du métaproduit des implicants premiers essentiels sans faire appel à aucune substitution ni élimination. Nous introduisons les notations suivantes :

$$\begin{aligned} IntPro(P, S) &= \{ p \in P \mid p \cap S \neq \varnothing \} \\ Deg(P, 0) &= \{ x \in E \mid \forall p \in P,\ x \notin p \} \\ Deg(P, k+1) &= \{ x \in E \mid \exists p \in P,\ x \in p \wedge x \in Deg(P \setminus \{p\}, k) \} \end{aligned}$$

L'ensemble $IntPro(P, S)$ réunit les produits de P qui contiennent au moins un point de S. L'ensemble $Deg(P, k)$ est celui des points de $\{0, 1\}^n$ qui sont contenus par exactement k produits de P. Par définition, on a $Ess(f) = IntPro(Prime(f), Deg(Prime(f), 1))$. Il nous reste donc à exprimer les opérations *IntPro* et *Deg* avec des équations récursives. Les deux théorèmes suivant explicitent ces équations.

THÉORÈME 4 : *Soit P un ensemble de produits, et f une fonction booléenne. Les équations suivantes sont suffisantes* [14] *pour calculer l'ensemble $IntPro(P, f)$:*

$$\begin{aligned} IntPro(P, 0) &= \varnothing \\ IntPro(P, 1) &= P \\ IntPro(P, f) &= IntPro(P_{\varepsilon_k}, f_{\bar{x}_k} \vee f_{x_k}) \cup \\ &\qquad \{\bar{x}_k\} \times Int\,Pro(P_{\bar{x}_k}, f_{\bar{x}_k}) \cup \\ &\qquad \{x_k\} \times IntPro(P_{x_k}, f_{x_k}) \end{aligned}$$

THÉORÈME 5 : *Soit P un ensemble de produits. Les équations suivantes sont suffisantes pour calculer Deg(P, 0) et Deg(P, 1):*

$$
\begin{aligned}
&Deg(\varnothing, 0) = 1\\
&Deg(\varnothing, 1) = 0\\
&Deg(\{\varepsilon\}, 0) = 0\\
&Deg(\{\varepsilon\}, 1) = 1\\
&Deg(P, 0) = \Delta(x_k,\\
&\qquad Deg(P_{\varepsilon_k}, 0) \wedge Deg(P_{\bar{x}_k}, 0),\\
&\qquad Deg(P_{\varepsilon_k}, 0) \wedge Deg(P_{x_k}, 0))\\
&Deg(P, 1) = \Delta(x_k,\\
&\qquad (Deg(P_{\varepsilon_k}, 1) \wedge Deg(P_{\bar{x}_k}, 0)) \vee (Deg(P_{\varepsilon_k}, 0) \wedge Deg(P_{\bar{x}_k}, 1)),\\
&\qquad (Deg(P_{\varepsilon_k}, 1) \wedge Deg(P_{x_k}, 0)) \vee (Deg(P_{\varepsilon_k}, 0) \wedge Deg(P_{x_k}, 1))).
\end{aligned}
$$

La figure 5 illustre les règles récursives qui permettent de construire le métaproduit de $IntPro(\mathscr{P}, f)$ à partir du métaproduit $\mathscr{P}$ et du BDD f. La figure 6 illustre les règles récursives qui permettent de construire le BDD de $Deg(\mathscr{P}, 1)$ à partir du métaproduit $\mathscr{P}$.

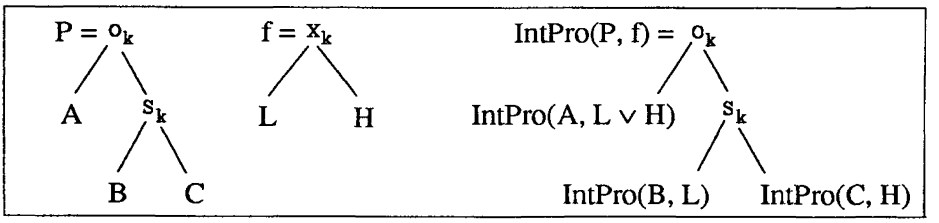

Figure 5. – **Évaluation de** *IntPro*.

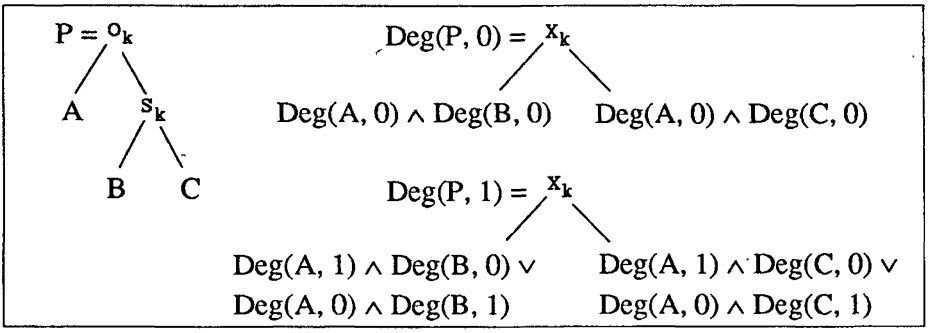

Figure 6. – **Évaluation de** *Deg*.

5. EXTENSIONS DE NOTRE APPROCHE

La notion d'implicant premier et d'implicant premier essentiel peut être étendue [5, 38, 39] à d'autres fonctions discrètes que les fonctions booléennes : aux fonctions booléennes partielles, c'est-à-dire non partout définies ; aux fonctions booléennes vectorielles partielles, c'est-à-dire aux fonctions de $\{0, 1\}^n$ dans $\{0, 1, \perp\}^m$; aux fonctions booléennes multivaluées partielles, c'est-à-dire aux fonctions de $D_1 \times \ldots \times D_n$ dans $\{0, 1, \perp\}$, où chaque domaine D_k est fini. L'extention aux fonctions booléennes partielles ne pose pas de problème majeur. Par contre, le calcul des implicants premiers des fonctions booléennes vectorielles ou multivaluées est considérablement plus complexe. Nous avons défini des procédures de calculs en intention des implicants premiers et essentiels sur ces types de fonctions discrètes [14], en utilisant les concepts introduits Section 3.

Nous avons élaboré les algorithmes présentés Section 3 au mois d'octobre 1991, et ils ont été présentés pour la première fois dans [12]. Depuis, nous avons introduit de nouveaux algorithmes de calcul symbolique des implicants premiers et essentiels [13, 14], plus puissants que ceux que nous avons décrits ici. Ces nouveaux algorithmes s'écartent complètement de la logique propositionnelle pour rentrer dans le cadre d'un *calcul formel* unifié sur les métaproduits. Cette approche est plus complexe, mais l'idée de base reste la même : disposer d'une représentation fonctionnelle booléenne des ensembles de produits qui peut être greffée sur une structure de donnée efficace, en l'occurrence encore plus compacte que les métaproduits. Nous avons choisi de ne présenter que le tout premier algorithme, car il demande beaucoup moins de place pour être exposé, et est beaucoup plus intuitif que notre toute dernière approche.

6. RÉSULTATS EXPÉRIMENTAUX

Le Tableau 2 compare les temps de calcul obtenus pour calculer les implicants premiers d'arbres de défaillance par une technique « classique », avec ceux obtenus par l'outil prototype METAPRIME développé chez Bull, qui utilise la méthode de calcul des implicants premiers décrite dans cet article. Ces arbres de défaillance sont des cas réels industriels et proviennent du domaine de l'avionique (exemple *b*4 à *bred*) et du nucléaire (exemples *ex* 1 à *edf* 2). La colonne **# var** donne le nombre d'événements élémentaires (*i. e.* le nombre n de variables propositionnelles) de l'arbre de défaillance, et la colonne **# premier** donne son nombre d'implicants premiers. La colonne

TABLEAU 2

Comparaison des temps de calcul des implicants premiers d'arbres de défaillance.

Nom	#var	#premier	$T_{classique}$ (s)	T_{MP} (s)
*b*4	8	4	3	0.1
*b*560	32	560	1	0.2
*b*2500	50	2376	7	0.8
*b*5000	110	5256	2640	2.3
*b*9000	50	9216	33	0.6
*b*26000	276	25988	84383	7.6
*b*8060	103	8060	10577	3.8
bind	109	82000000000	—	0.8
*b*14200	122	14217	2470	0.4
*b*16700	53	16707	130	0.3
*b*16200	51	16200	147	0.3
*b*17300	51	17280	74	0.3
*b*19500	121	19518	43279	0.6
*b*12100	170	12143	8702	15.7
bred	49	27778	—	0.8
*ex*1	60	46188	?	3.9
*ex*2	32	5630	?	1.8
*ex*3	92	24386	?	5.2
*edf*1	291	579720	—	3.2
*edf*2	382	20807446	—	369.0

$T_{\textbf{classique}}$ donne le temps de calcul obtenu avec un outil prototype développé par Dassault-Aviation utilisant une technique classique, et la colonne $T_{\textbf{MP}}$ donne le temps de calcul obtenu avec METAPRIME. Ces temps de calcul sont mesurés en secondes sur une machine SUN Sparc2, et comprennent aussi le calcul de la distribution des implicants premiers en fonction de leur taille. Un « − » indique que le calcul n'a pu être mené à son terme. Un «?» indique qu'à notre connaissance, il n'existe pas d'outil classique pouvant effectuer le traitement complet après plusieurs heures de calcul. Le gain de performance que l'on obtient sur ces exemples est de l'ordre de 1 000. De plus, les exemples qui ne peuvent être abordés par aucune technique classique à cause du trop grand nombre d'implicants premiers sont traités en quelques secondes avec notre technique.

Le tableau 3 présente la résolution, grâce à notre technique, de 25 problèmes ouverts. Les 20 exemples *ex* 1010 à *xparc* proviennent d'un ensemble de 145 descriptions de circuit qui constituent le MCNC benchmark [45]. Les 5 exemples *mul X* décrivent des multiplicateurs sur deux nombres entiers codés sur X bits, et les exemples *cbp X* des additionneurs sur deux nombres entiers codés sur X bits. Chacune des descriptions spécifie une fonction booléenne vectorielles de $\{0, 1\}^n$ dans $\{0, 1\}^m$, où n et m sont respectivement donnés par **#in** (le nombre d'entrées du circuit) et **#out** (le nombre de sorties du circuit). Un «oui» dans la colonne **undef** indique de plus que la fonction

TABLEAU 3

Résolution de 25 problèmes ouverts.

Nom	#in	#out	undef	#premier	T_{MP} (s)
*ex*1010	10	10	*oui*	25888	29
*test*2	11	35	*oui*	109099	998
*test*3	10	35	*oui*	41344	285
accpla	50	69	*non*	1758057	359
*ex*4	128	28	*non*	1.83487*e*+14	1.3
ibm	48	17	*non*	1047948800	3.5
jbp	36	57	*non*	2496809	184
mainpla	27	54	*non*	87692	398
misg	56	23	*non*	6499491840	0.4
mish	94	43	*non*	1.12437*e*+15	2.7
misj	35	14	*non*	139103	0.3
pdc	16	40	*oui*	23231	129
shift	19	16	*non*	165133	4.5
signet	39	8	*non*	78735	207
soar	83	94	*non*	3.30477*e*+14	8.0
ti	47	72	*non*	836287	52
*ts*10	22	16	*non*	524280	14.4
*x*2*dn*	82	56	*non*	1.14887*e*+16	9.7
*x*7*dn*	66	15	*non*	566698631	29
xparc	41	73	*non*	15039	34
*mul*06	12	12	*non*	26264	31
*mul*07	14	14	*non*	163361	320
*mul*08	16	16	*non*	984384	6872
*cbp*16	33	17	*non*	68751306483	0.6
*cbp*32	65	33	*non*	2.84699*e*+21	1.9

booléenne vectorielle n'est définie que partiellement. Parmi les 145 descriptions du MCNC benchmark, le nombre d'implicants premiers (généralisés aux fonctions booléennes vectorielles partielles) des exemples *ex* 1010 à *x parc* étaient jusqu'à présent inconnu. Notre technique a permis de résoudre ces problèmes ouverts. De même, le nombre d'implicants premiers (généralisés aux fonctions booléennes vectorielles partielles) des exemples *mul*06 à *cpb*32 était inconnu jusqu'alors, et ces 5 autres problèmes ouverts ont pu être résolus en utilisant notre technique. Les temps de calcul sont donnés par la colonne T_{MP} en secondes sur une machine SUN Sparc2.

La figure 7 montre un diagramme de performance du calcul d'implicants premiers par notre technique sur environ 60 exemples. Les cercles sont les exemples donnés par le Tableau 2, les ronds noirs sont des exemples tirés des 145 descriptions du MCNC benchmark. En abcisse sont portés les logarithmes en base 10 des nombres d'implicants premiers (généralisés aux fonctions booléennes vectorielles partielles), et en ordonnée les logarithmes en base 10 des temps de calcul en secondes nécessaires pour calculer les implicants premiers sur une machine SUN Sparc2. La droite inclinée indique

la meilleure performance qui pourra jamais être obtenue avec une technique énumérative, c'est-à-dire avec un temps de calcul linéaire par rapport au nombre d'implicants premiers. On voit aisément que la performance de notre algorithme sur ces exemples surpasse largement les meilleurs algorithmes actuellement connus. De plus, si l'on trace la droite de régression du nuage de points, on obtient une droite de pente quasi nulle : ces résultats expérimentaux corroborent donc le résultat théorique qui affirme que la complexité de notre algorithme *n'est pas liée* au nombre d'implicants premiers de la fonction traitée.

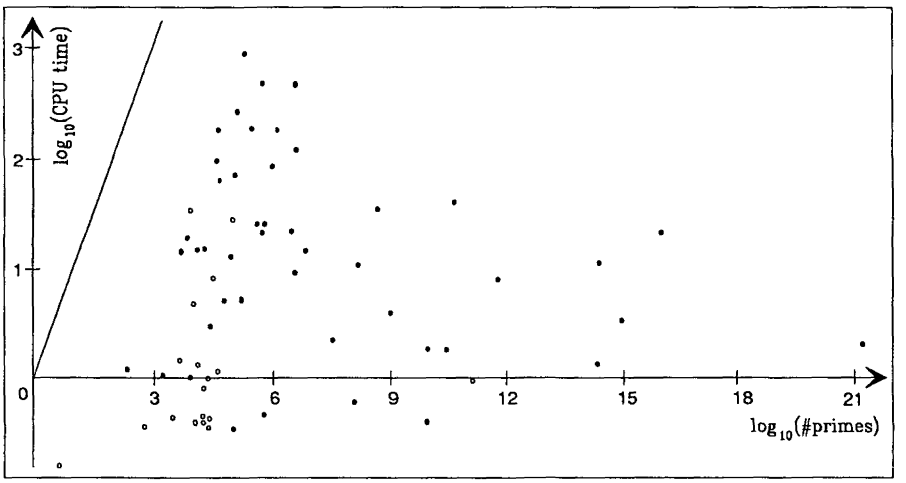

Figure 7. – Diagramme de performance.

7. CONCLUSION

Nous avons présenté ici un nouvel algorithme de calcul des implicants premiers et essentiels d'une fonction booléenne. Cet algorithme se démarque fondamentalement de ceux connus jusqu'alors, car le but n'est plus ici de calculer *en extension* les implicants premiers, *i. e.*, de les énumérer tous, comme le faisaient les techniques précédentes, mais de calculer une représentation *en intention* des implicants premiers, sur laquelle des problèmes complexes peuvent être résolus avec une complexité polynomiale par rapport à la taille de cette représentation.

Dans ce but, nous avons introduit la notion de *métaproduit*, qui permet de représenter en intention n'importe quel ensemble de produits. Un métaproduit

est une fonction booléenne, elle même représentée par son diagramme de décision binaire (BDD), qui est une représentation compacte et canonique des fonctions booléennes. Comme il n'existe aucune relation entre la taille du BDD d'un métaproduit et le nombre de produits que celui-ci représente, nous pouvons manipuler des ensembles de produits trop grands pour être représentés en extension, ce qui constituait la limite de toutes les approches précédentes. De plus, l'algorithme que nous avons proposé possède ainsi une complexité qui ne dépend *absolument pas* du nombre d'implicants premiers ou essentiels à calculer. C'est ce qui permet à cet algorithme de dépasser d'un facteur 1 000 toutes les méthodes existantes sur des exemples conséquents, et de pouvoir résoudre des problèmes inabordables avec ces techniques.

REMERCIEMENTS

Les auteurs tiennent à remercier Emmanuel Ledinot et Damien Alexandre de Dassault Aviation, Marc Bouissou de l'Électricité de France, et Damien Ehret d'Elf Aquitaine, pour nous avoir fourni des exemples industriels d'arbres de défaillance.

BIBLIOGRAPHIE

1. S. B. Akers, Binary Decision Diagrams, *IEEE Trans. on Computers*, 1978, vol. C-27.
2. T. C. Bartee, I. L. Lebow, I. S. Reed, *Theory and Design of Digital Machines*, McGraw-Hill, 1962.
3. J.-P. Billon, Perfect Normal Forms for Discrete Functions, *BULL Research Report n° 87019*, juin 1987.
4. N. N. Biswas, *Introduction to Logic and Switching Theory*, Gordon & Breach Science, 1975.
5. R. E. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
6. F. M. Brown, *Boolean Reasoning*, Kluwer Academic Publishers, 1990.
7. R. E. Bryant, Graph-Based Algorithms for Boolean Functions Manipulation, *IEEE Trans. on Computers*, 1986, vol. C35.
8. R. E. Bryant, *On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication*, Carnegie Mellon University Research Report, September 1988.
9. K. M. Butler, D. E. Ross, R. Kapur, M. R. Mercer, Heuristics to Compute Variable Orderings for Efficient Manipulations of Ordered Binary Decision Diagrams, in *Proc. of 28th Design Automation Conference*, San Francisco, California, June 1991, 417-420.
10. O. Coudert, J. C. Madre, A Unified Framework for the Formal Verification of Sequential Circuits, *Proc. of ICCAD'90*, novembre 1990, Santa Clara CA, U.S.A.

11. O. COUDERT, *S.I.A.M.: Une boîte à outils pour la preuve formelle de systèmes séquentiels*, Thèse de troisième cycle, École Nationale Supérieure des Télécommunications, Paris, France, octobre 1991.
12. O. COUDERT, J. C. MADRE, A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions, *Proc. of Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, mars 1992, Cambridge MA, U.S.A.
13. O. COUDERT, J. C. MADRE, Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions, *Proc. of 29th DAC*, juin 1992, Anaheim CA, U.S.A.
14. O. COUDERT, J. C. MADRE, A New Implicit DAG Based Prime and Essential Prime Computation Technique, *Proc. of International Symposium on Information Sciences*, juillet 1992, Fukuoka, Japon.
15. M. DAVIS, H. PUTNAM, A Computing Procedure for Quantification Theory, *Journal of the ACM*, 1960, vol. 7, 201-215.
16. S. J. FRIEDMAN, K. J. SUPOWIT, Finding the Optimal Variable Ordering for Binary Decision Diagrams, *IEEE Trans. on Computer*, 1990, vol. C-39, 710-713.
17. D. F. HASL, Advanced Concepts in Fault Tree Analysis, *Proc. of System Safety Symposium*, juin 1965, Seatle.
18. S. J. HONG, S. MUROGA, Absolute Minimization of Completely Specified Switching Functions, *IEEE Trans. on Computers*, 1991, vol. 40, 53-65.
19. H. R. HWA, A Method for Generating Prime Implicants of a Boolean Expression, *IEEE Trans. on Computers*, 1974, 637-641.
20. H. Y. HWANG, D. S. CHAO, M. E. VALDEZ, A New Technique for the Minimization of Switching Functions, *IEEE Southeastcon'85*, 1985, 299-304.
21. J. DE KLEER, An Assumption-Based TMS, *Artificial Intelligence*, 1986, vol. 28, 127-162.
22. J. DE KLEER, B. C. WILLIAMS, Diagnosing Multiple Faults, *Artificial Intelligence*, 1987, vol. 32, 97-130.
23. M. C. LOUI, G. BILARDI, *The Correctness of Tison's Method for Generating Prime Implicants*, Report R-952, UILU-ENG 82-2218, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1982.
24. E. L. Jr. MCCLUSKEY, Minimization of Boolean Functions, *Bell System Techniques*, 1959, vol. 35, 1417-1444.
25. J. C. MADRE, J. P. BILLON, Proving Circuit Correctness Using Formal Comparison Between Expected and Extracted Behaviour, *Proc. of the 25th DAC*, juillet 1988, Anaheim CA, U.S.A.
26. J. C. MADRE, O. COUDERT, A Logically Complete Reasoning Maintenance System Based on Logical Constrain Solver, *Proc. of IJCAI'91*, août 1991, Sydney, Australia.
27. S. MALIK, A. R. WANG, R. K. BRAYTON, A. SANGIOVANNI-VINCENTELLI, Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment, *Proc. of ICCAD'88, Santa Clara*, 1988, U.S.A.
28. S. MINATO, N. ISHIURA, S. YAJIMA, Shared Binary Decision Diagrams with Attributed Edges for Efficient Boolean Function Manipulation, *Proc. of the 27th Design Automation Conference*, June 1990, Las Vegas, Nevada, 52-57.
29. A. PAGÈS, M. GONDRAN, *Fiabilité des systèmes*, Eyrolles, 1980.
30. W. V. O. QUINE, The Problem of Simplifying Truth Functions, *American Mathematics Monthly*, 1952, vol. 59, 521-531.

31. W. V. O. QUINE, A Way to Simplify Truth Functions, *American Mathematics. Monthly*, 1955, vol. 62, 627-631.

32. W. V. O. QUINE, On Cores and Prime Implicants of Truth Functions, *American Mathematics Monthly*, 1959, vol. 66.

33. R. REITER, A Theory of Diagnosis From First Principles, *Artificial Intelligence*, 1987, vol. 32.

34. R. REITER, J. de KLEER, Foundation of Assumption-Based Truth Maintenance Systems: Preliminary Report, *Proc. of 6th AAAI*, 1987, 183-188.

35. V. T. RHYNE, P. S. NOE, M. H. MCKINNEY, U. W. POOCH, A New Technique for the Fast Minimization of Switching Functions, *IEEE Trans. on Computers*, 1977, vol. C-26/8, 757-764.

36. J. A. ROBINSON, A Machine-Oriented Logic Based on the Resolution Principle, *Journal of ACM*, 1965, vol. 12, 23-41.

37. J. P. ROTH, Algebraic Topological Methods for the Synthesis of Switching Systems, *Trans. of American Mathematical Society*, 1958, vol. 88/2, 301-326.

38. R. L. RUDELL, A. L. SANGIOVANNI-VINCENTELLI, Multiple Valued Minimization for PLA Optimization, *IEEE Trans. on CAD*, 1987, vol. 6, 727-750.

39. T. SASAO, An Application of Multiple-Valued Logic to a Design of Programmable Logic Arrays, *Proc. of 8th Int'l Symposium on Multiple Valued Logic*, 1978.

40. J. R. SLAGE, C. L. CHANG, R. C. T. LEE, Completeness Theorems for Semantics Resolution in Consequence Finding, *Proc. of IJCAI'69*, 1969, 281-285.

41. J. R. SLAGE, C. L. CHANG, R. C. T. LEE, A New Algorithm for Generating Prime Implicants, *IEEE Trans. on Computers*, 1970, vol. C-19 (4), 304-310.

42. M. STONE, The Theory of Representations for Boolean Algebra, *Trans. Amer. Math. Soc.*, 1936, vol. 40, 37-111.

43. P. TISON, Generalized Consensus Theory and Application to the Minimization of Boolean Functions, *IEEE Trans. on Electronic Computers*, 1967, vol. EC-16/4, 446-456.

44. A. VILLEMEUR, *Sûreté de fonctionnement des systèmes industriels*, Eyroles, 1988.

45. S. YANG, *Logic Synthesis and Optimization Benchmarks User Guide*, Microelectronics Center of North Carolina, January 1991.