Gestion de la Condition de Chemin dans la simulation symbolique.

Mireille Larnac

ACADEMIE DE MONTPELLIER

UNIVERSITE MONTPELLIER II

- SCIENCES ET TECHNIQUES DU LANGUEDOC -

THESE

présentée à l'Université des Sciences et Techniques du Languedoc

pour obtenir le diplôme de DOCTORAT

SPECIALITE : COMPOSANTS, SIGNAUX ET SYSTEMES

Formation Doctorale : Conception Assistée des Systèmes Informatiques, Automatiques et Microélectroniques

Gestion de la Condition de Chemin dans la Simulation Symbolique

par

Mireille LARNAC

Soutenue le 13 novembre 1992 devant le Jury composé de :

M.	BAYLE Pierre,	Dassault Electronique	Examinateur
M.	DURANTE Christian,	Professeur, Université de Montpellier II	Président
M.	FLOUTIER Denis,	Professeur, Université de Savoie	Rapporteur
M.	GIAMBIASI Norbert,	Professeur, Université d'Aix-Marseille III	Examinateur
Mme	MAGNIER Janine,	Professeur, Ecole des Mines d'Alès	Directeur de Thèse
Mlle	ROBACH Chantal,	Chercheur CNRS, INP Grenoble	Rapporteur

Je tiens à exprimer ma reconnaissance :

à Madame Janine MAGNIER, mon Directeur de thèse, et à Monsieur Norbert GIAMBIASI Directeur du LERI, qui ont tous deux contribué par leurs idées et leur assistance à la réalisation de ce travail, qu'ils soient ici remerciés pour leur humour, leur gentillesse, l'amitié et la confiance qu'ils m'ont accordées,

à Monsieur Pierre BAYLE, Responsable technique du département Outils de Conception chez DASSAULT Electronique, pour l'attention qu'il a portée à la lecture de ce manuscrit, ses remarques pertinentes et sa participation au jury,

à Monsieur Christian DURANTE, Professeur à l'Université de Montpellier II, qui me fait l'honneur de présider le jury,

à Mademoiselle Chantal ROBACH, Chargée de Recherches au CNRS, et Monsieur Denis FLOUTIER, Professeur à l'Université de Savoie qui, malgré leurs nombreuses charges, ont accepté d'être rapporteurs de ce travail,

à Rémy, compagnon de billard, de travail et de galère, avec qui mes rapports furent aussi divers qu'enrichissants,

à Philippe, François et Jin-Kao qui m'ont apporté leur aide quand cela s'est avéré nécessaire,

à Véronique et Pascale, secrétaires du LERI, et aux chercheurs qui m'ont offert leur soutien et leur amitié, en particulier Lucile, Marta, Claudia, Véronique, Gérard, Loïc, Gaby, Marc, Guilaine,

Merci à mes amis Poupoune, Véronique et Daniel, Annie et Pierre, Maguy, Franck, Françoise, Roro et Mamie pour leur soutien moral inestimable,

Merci également à La Vieille (alias Maman Chérie), mes frères Gilles et Vincent, Tiphaine et Serge,

Merci enfin à ma sœur Hélène, soutien médical et moral toujours présent, qui s'évertue à vouloir le renforcement de ma ceinture abdominale mais qui, pour se faire pardonner, a mis au monde les deux petites filles qui illuminent tout leur entourage ...

A Chloé et Margot

Introduction

Pendant de nombreuses années, la mise au point des circuits a consisté à concevoir des modèles, construire un prototype et vérifier ses fonctionnalités.
Ce processus par "essais-erreurs" est long et coûteux.
C'est pourquoi des méthodes de vérification de modèles, qui permettent la validation des circuits à chaque étape de leur conception, ont été développées.

Nous avons choisi de proposer un ensemble d'outils de vérification pour des circuits décrits au niveau comportemental. C'est tout naturellement que nous avons sélectionné le langage de description VHDL tant pour ses capacités à décrire des systèmes complexes, que pour sa large diffusion dans le monde des concepteurs.
Les travaux présentés dans ce mémoire s'inscrivent donc dans le cadre d'un projet général de vérification de descriptions comportementales de circuits digitaux, exprimées dans le langage VHDL.

La simulation symbolique est une des méthodes de vérification.
Nous nous intéressons ici à la résolution d'un problème spécifique à ce type de simulation : le traitement des branchements conditionnels présents dans la description.
En effet, en dépit du fait que la valeur de la condition est a priori symbolique, il

peut se produire que l'ensemble des choix effectués sur les branchements conditionnels précédemment rencontrés (**Condition de Chemin**) contienne la valeur de vérité de cette condition. Dans ce cas, le simulateur devra impérativement suivre le chemin d'exécution déterminé par la valeur de vérité de la condition.

Le premier chapitre est composé d'une présentation du système général de vérification de descriptions comportementales VHDL, puis d'un état de l'art sur la simulation symbolique et la Condition de Chemin.
Enfin, à partir de la forme des conditions qui peuvent être rencontrées dans le sous-ensemble "comportemental" de VHDL (dont les types de variables et signaux sont limités aux booléens, bits et entiers), quatre classes d'expressions apparaissent : expressions purement booléennes, expressions arithmétiques élémentaires, expressions arithmétiques complexes, et expressions mixtes.

Le chapitre 2 est consacré à la méthodologie générale de gestion de la Condition de Chemin, qui comprend :
- la définition de l'**opérateur de choix** dont la mise en œuvre permet de déterminer si la Condition de Chemin contient ou non la valeur de vérité de la condition à évaluer,
- la définition de la **variable** booléenne de **choix libre** (résultat de l'opération de choix),
- la présentation des propriétés de l'opérateur de choix.

La gestion de la Condition de Chemin pour les expressions booléennes fait l'objet du chapitre 3. Après une brève étude des méthodes de représentation et de manipulation de fonctions logiques, nous choisissons de gérer la Condition

de Chemin à l'aide des Diagrammes de Décision Binaire.

Le chapitre 4 est consacré à l'étude de la gestion de la Condition de Chemin pour un cas particulier d'expressions arithmétiques (expressions arithmétiques élémentaires) : la Condition de Chemin est un système d'inéquations linéaires, et la condition à évaluer n'est formée que d'une seule inéquation linéaire. La détermination de la variable de choix est basée sur l'exploitation des propriétés particulières aux expressions linéaires.

Nous étudions enfin dans le chapitre 5 comment gérer la Condition de Chemin dans le cadre d'expressions quelconques (arithmétiques quelconques ou mixtes) à partir :

- des méthodes de gestion de la Condition de Chemin pour les expressions particulières (présentées dans les chapitres 3 et 4),
- et des propriétés de l'opérateur de choix.

Finalement, nous concluons nos travaux par leur situation dans le projet général de vérification, et sur les perspectives d'utilisation des méthodes de gestion d'expressions symboliques définies dans ce mémoire.

Chapitre 1 - Présentation générale

Nous présentons dans ce chapitre le projet général de vérification qui fait l'objet de nos recherches, et décrivons plus précisément le cadre de notre étude. Cela nous conduit à évoquer la nature de la simulation symbolique, puis à préciser le rôle de la Condition de Chemin dans ce type de simulation.
Mais auparavant, nous définissons la terminologie afférente à la vérification, et effectuons un bref rappel sur VHDL.

1 - Terminologie

Dans la phase de conception d'un système complexe, nous pouvons distinguer deux points :
- les **spécifications** qui contiennent la description des fonctionnalités du système. Les spécifications sont **formelles** si elles sont exprimées dans un langage formel, elles sont **informelles** sinon.
- la **réalisation** qui est une mise en œuvre du système, et qui doit obéir aux spécifications.

La **vérification** est la tâche qui consiste à valider la réalisation par rapport aux

spécifications. Elle est **formelle** si elle est prise en charge par un outil formel de **démonstration** (ou **preuve**). Dans ce cas, les spécifications sont obligatoirement formelles. Par contre, que les spécifications soient formelles ou non, la vérification est **informelle** si elle est basée sur un traitement manuel par lequel l'utilisateur s'assure de la validité de la réalisation sans s'aider d'un outil formel.

La vérification de la réalisation est **partielle** si elle se limite à la vérification pour un sous-ensemble de l'ensemble des combinaisons d'entrée du système. A l'opposé, la vérification est **globale** si toutes les configurations d'exécution possibles du système sont vérifiées.

La vérification formelle partielle de la réalisation constitue donc la **preuve partielle** de la réalisation, et la vérification formelle globale de la réalisation est la **preuve globale**.

La **simulation symbolique** est une technique de simulation dont les entrées ne sont pas toutes instanciées par une valeur numérique, mais dont la valeur est une expression constituée de symboles. Une session de simulation symbolique correspond alors à l'exécution symbolique d'un sous-arbre de l'arbre d'exécution symbolique.
La simulation symbolique peut être considérée comme base d'un outil de vérification : la réalisation à vérifier est simulée symboliquement. Il reste alors à vérifier (formellement ou informellement) la validité des résultats de la simulation par rapport aux spécifications.

2 - Rappels sur VHDL

2.1 - Présentation générale

VHDL (VHSIC Hardware Description Language) est un langage de description de circuits digitaux issu d'un appel d'offre du Department of Defense (DoD) américain visant à standardiser les descriptions de circuits.
Lancé en 1981, ce programme a débouché sur la définition de la norme IEEE-Std 1076-1987 en Décembre 1987 [IEE87].
VHDL permet la description de systèmes complexes à tout niveau d'abstraction, et offre les concepts présents dans la plupart des langages de description de circuits (HDL) [IEE86, ARM89, LIP89, AIR90].

En particulier, ce langage permet :
- de définir et manipuler de façon distincte des variables abstraites et des signaux (données qui sont liées au matériel),
- de spécifier des données temporelles, et en particulier d'exprimer des retards sur des affectations de signaux,
- de décrire aussi bien des comportements séquentiels que concurrents,
- de synchroniser des processus par le biais d'instructions d'attente.

2.2 - VHDL comportemental

Les sous-ensembles de concepts VHDL qui définissent les différentes vues d'un modèle (comportement, structure, flot de données) ne sont pas inclus dans la norme. Cependant, le VDEG (VHDL Design Exchange Group), groupe qui

travaille sur la standardisation des modèles, a défini ces sous-ensembles selon plusieurs niveaux (S0 : Core VHDL, S1 : Design Exchange, S2 : Full VHDL). [BAR88, NUR89].

Le niveau S0 de VHDL comportemental contient les notions d'entités, d'architectures, de sous-programmes, de ports, de variables, d'instructions d'affectations, d'appels de procédure et de structures de contrôle (instructions IF, CASE, LOOP, ...). Le niveau S1 est obtenu en considérant les éléments du niveau S0 auquel sont principalement rajoutées les notions de packages, de déclarations de types, de déclarations de signaux, d'instruction d'attente (WAIT). Finalement, les concepts du niveau S1 et ceux d'alias, d'attributs, de déclaration de constantes et de sous-types, d'assertions concurrentes, d'appels de procédures concurrents, de fichiers et d'allocation dynamique composent le niveau Full VHDL.

2.3 - Vérification de descriptions VHDL

Le développement du langage a été accompagné de la diffusion d'outils (éditeurs, compilateurs, simulateurs, ...) [SHA86, HIN87, LOU88, MAR88, SAU88].

Les environnements VHDL disponibles sur le marché offrent seulement des outils de simulation par valeur. Il est toutefois possible d'effectuer des tâches de vérification grâce au développement du langage de spécifications, VAL (VHDL Annotation Language) [AUG87, AUG88]. Ce langage d'annotations, développé par une équipe de l'Université de Stanford, permet d'exprimer des conditions sur le type de résultat produit par le modèle, des conditions qualifiées temporellement sur les valeurs des signaux de la description VHDL, ainsi que

des assertions qui définissent un comportement de haut-niveau du modèle.
Les annotations VAL sont transformées en code VHDL par un pré-processeur avant le lancement de la simulation.

Les environnements VHDL existants proposent donc des outils de simulation par valeurs des descriptions VHDL, avec vérification éventuelle de spécifications exprimées en VAL [AUG91].

3 - Le projet de vérification

Le projet de vérification que nous présentons constitue la suite logique des travaux menés précédemment au sein de notre équipe de recherche.
En effet, un précédent projet de vérification par simulation symbolique a donné le jour à deux outils : SYMSIM et SYMSTRU, simulateurs symboliques de descriptions respectivement comportementales et structurelles de circuits digitaux décrits en DDL (niveau transfert de registres) [HAS87, GIA89].
Notons également le développement de l'outil SYSLOG de représentation et de vérification de machines séquentielles. Cet outil permet d'exprimer le comportement d'un circuit à l'aide d'une machine séquentielle, puis :
- de générer des séquences symboliques de vérification,
- de démontrer des propriétés d'une machine,
- de prouver l'équivalence de deux machines.

L'objectif du présent projet de vérification est d'offrir à l'utilisateur un ensemble d'outils lui permettant de vérifier une description comportementale exprimée en

VHDL.

Les différentes possibilités de vérification que nous souhaitons lui fournir doivent lui permettre d'effectuer aussi bien :

- une simulation par valeurs : un seul chemin de l'arbre d'exécution du programme est parcouru ; l'utilisateur vérifie alors manuellement ou automatiquement que les résultats obtenus le satisfont.
- une vérification par simulation symbolique : un chemin ou un sous-arbre de l'arbre d'exécution symbolique du programme est construit ; la vérification est alors manuelle ou formelle.
- ou une vérification globale par démonstration (preuve globale).

La structure du système de vérification est donnée dans la Figure 1.

Nos travaux, plus spécifiquement axés sur la vérification partielle de descriptions comportementales VHDL, concernent la mise en œuvre de la simulation symbolique.

Le noyau de simulation symbolique est explicité dans [ROG92a].

Le présent mémoire, quant à lui, traite d'un problème spécifique à la simulation symbolique : la Gestion de la Condition de Chemin.

Etudions rapidement les différentes composantes du système de vérification :

• Description comportementale VHDL :

Nous limitons le langage VHDL comportemental au niveau "S1 : Design Exchange" qui regroupe les concepts nécessaires à la description comportementale et à l'échange, entre équipes de conception, de modèles.

Ce sous-ensemble est détaillé dans [ROG92a] ; notons toutefois que nous avons limité les types de données manipulées aux types booléen, bit, et entier.

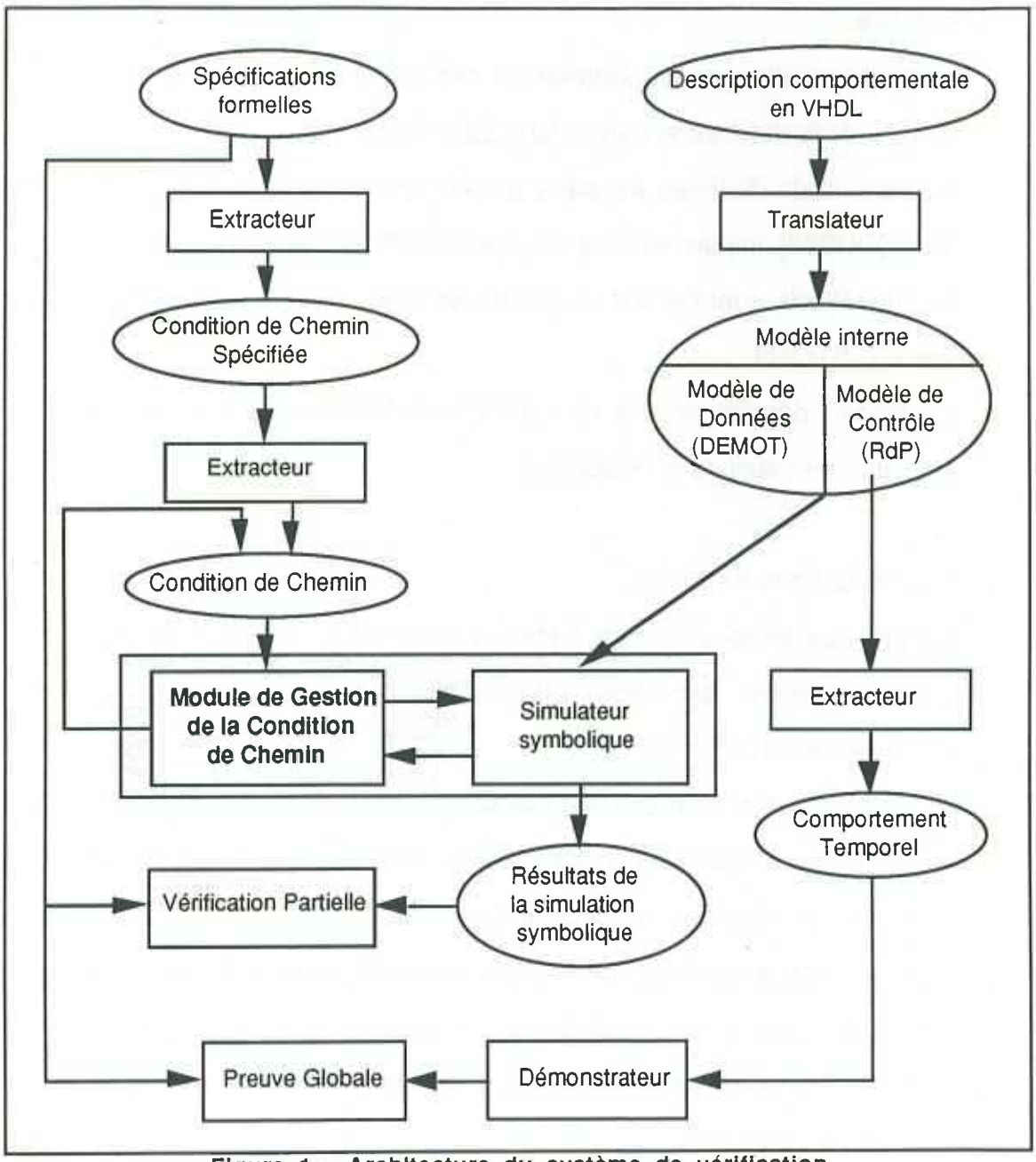


Figure 1 - Architecture du système de vérification

• Translateur - Modèle interne :

La description comportementale à vérifier est traduite dans un modèle interne, indépendant du langage de description, qui sépare explicitement le flot de données (données manipulées par le programme et actions sur ces données) du flot de contrôle (séquencement des actions). Le flot de données est alors

représenté par le modèle de données, le flot de contrôle par le modèle de contrôle.

Cette séparation entre données et contrôle est une approche connue, déjà utilisée dans d'autres systèmes [STO85, PEN86, DJA91].

Le modèle de données est basé sur les notions de graphes hiérarchisés/multi-vues [OUS88], implantées dans le système DEMOT.

Le modèle de contrôle est un réseau de Petri interprété et temporisé [PET81, BRA82, STO85].

Le modèle développé pour ce système est fortement inspiré de [DJA91] ; il est présenté en détail dans [ROG92a].

• Spécifications formelles

Un langage de spécifications formelles a été défini. Il permet d'exprimer :

- des conditions qualifiées temporellement sur les variables et signaux du programme VHDL à vérifier,
- des "activités" d'instructions ou de process VHDL [DEL91, BOU92a, BOU92b].

Finalement, la spécification s'interprète comme suit : si les conditions sur les variables et signaux de la description sont vraies, alors les instructions et process cités seront actifs pendant les intervalles de temps déterminés.

Pour des raisons de convivialité, un langage externe permet la saisie des spécifications ; la traduction dans le formalisme interne est effectuée par un compilateur [DEL91].

Remarque :

La première partie de la spécification (conditions sur les variables et signaux du programme VHDL) peut être assimilée aux stimuli donnés en entrée d'un simulateur par valeurs.

La vérification proprement dite utilise un démonstrateur et un simulateur symbolique.

• la preuve globale d'une description comportementale est très complexe. Cette tâche est scindée en deux : la preuve de la validité du comportement temporel de la description par rapport aux spécifications [GHA89] est tout d'abord effectuée ; la vérification de cohérence des données sera développée ultérieurement. Ces travaux [BOU92a, BOU92b] ne seront pas développés dans ce mémoire.

• les tâches de vérification partielle reposent sur l'utilisation de simulateurs (par valeurs ou symbolique) et de démonstrateurs.

Remarque :

Une fois la description comportementale validée, elle peut être considérée comme spécification formelle (description de référence) pour une description VHDL d'une autre vue (structurelle par exemple) du même système. Il est alors possible de vérifier la description structurelle à l'aide de l'outil SYSLOG [MAG90].

Rappelons que le problème que nous traitons ici est la gestion de la Condition de Chemin dans la simulation symbolique.

Afin de mieux situer et comprendre ce problème, il est nécessaire de revenir sur l'historique de la simulation symbolique, et d'étudier comment le problème des branchements conditionnels était traité jusqu'alors.

4 - La simulation symbolique

Historiquement, les premiers travaux de vérification basés sur l'exécution symbolique ont eu lieu dans les années 70 (IBM). L'objectif des systèmes développés était d'apporter la preuve de validité de programmes avec les premiers travaux de FLOYD [FLO67], puis de KING [KIN69, KIN71, KIN72a, KIN72b, KIN76, HAN76], HOWDEN [HOW76, HOW77] et DARRINGER [DAR78].
Ce dernier envisagea l'application des méthodes de preuve de programme à la vérification de matériel [DAR79].
La vérification est basée sur l'insertion d'assertions dans le programme. Lors de l'exécution symbolique, les assertions doivent être vérifiées.
A la même époque, CARTER, JOYNER et BRAND ont étudié le problème de la vérification appliqué aux descriptions de matériel [CAR79], ce qui donna lieu à de multiples travaux par la suite [WAG77, PIT82, HAS87, GIA89, BUR90].
Ces approches intègrent les techniques de preuve de programme, tout en tenant compte des spécificités du matériel.

Nous basons notre approche de la vérification de descriptions de circuits sur la constatation suivante : vérifier formellement une description (comportementale ou structurelle) de matériel nécessite de connaître les expressions symboliques qui représentent le comportement du modèle en fonction du temps.
Cette extraction nécessaire du comportement du modèle constitue la définition de la simulation symbolique. A partir de cette simulation, une vérification empirique, partielle ou globale peut être envisagée.

4.1 - Utilisation de la simulation symbolique pour la vérification

Effectuer une vérification globale - qui équivaut à la preuve de la description - nécessite la construction de l'arbre d'exécution symbolique dans son intégralité. C'est l'approche choisie par DARRINGER [DAR78, DAR79], pour prouver la validité d'un programme. Cette méthode, très coûteuse, n'est envisageable que sur des descriptions de petite taille, avec peu de branchements conditionnels. Elle est bien souvent inapplicable dans le cas de descriptions complexes.

Par contre, la vérification d'un chemin (ou d'un sous-arbre) de l'arbre d'exécution requiert simplement la sélection et la construction de ce chemin. Le système DISSECT d'évaluation symbolique et de test de programmes FORTRAN [HOW76, HOW77] en est un exemple ; ici, le chemin est défini a priori par la donnée de tous les paramètres nécessaires à la détermination de la valeur booléenne des tests.

C'est ce même type de démarche qui a été adopté pour le développement de SIMSYM et SIMSTRU, outils de simulation symbolique de descriptions respectivement comportementales et structurelles de circuits, exprimées en DDL, au niveau transfert de registres [HAS87, GIA89].

4.2 - Utilisation d'un simulateur symbolique

Le fonctionnement d'un simulateur symbolique diffère peu de celui d'un simulateur par valeurs dans l'exécution des opérations d'affectation de variables ou signaux. En effet, le simulateur symbolique doit effectuer les opérations contenues en partie droite de l'instruction d'affectation, et réaliser l'affectation effectivement à la date prévue. Il est tout de même nécessaire de redéfinir dans

ce cas les opérateurs qui s'appliquent sur les variables et signaux dont la valeur est une expression symbolique.

Par contre, le traitement de l'évaluation d'une condition symbolique (dans un branchement conditionnel) est un problème spécifique à la simulation symbolique. En effet, que faire à la rencontre d'un test ? Globalement, deux solutions peuvent être envisagées :

• suivre les deux chemins d'exécution possibles, tout en conservant les choix effectués depuis le début d'exécution. En fin d'exécution, il faut alors évaluer si la conjonction de toutes les conditions fixées au cours de la simulation est une expression valide (satisfiable) ou pas. Si cette expression est inconsistante, le chemin déterminé ne doit pas être retenu dans l'arbre d'exécution.

• choisir un chemin d'exécution (une valeur du test), et vérifier que le choix effectué n'est pas en désaccord avec ceux effectués auparavant.

Notons que la première approche conduit à la construction de l'arbre d'exécution symbolique. La deuxième se limite à la construction d'un chemin ou d'un sous-arbre de cet arbre.

Par contre, nous pouvons constater que pour chacune des deux méthodes d'exécution, il est nécessaire de déterminer la consistance d'un ensemble de conditions. En effet, il ne faut en aucun cas introduire une contradiction dans les valeurs des conditions : cela impliquerait une incohérence dans les valeurs des variables et signaux du programme.

C'est pourquoi il est nécessaire de mémoriser l'ensemble des choix qui déterminent un chemin d'exécution donné dans une expression appelée Condition de Chemin.

5 - La Condition de Chemin

Lors de la simulation symbolique, quelle que soit la stratégie choisie pour le traitement des branchements conditionnels, il est indispensable de conserver l'ensemble des choix qui ont été opérés jusqu'ici. Ces conditions déterminent comment parvenir jusqu'à un nœud de l'arbre donné ; leur conjonction est donc la condition propre au chemin qui mène à ce nœud. Cette expression s'appelle la **Condition de Chemin**.

Après avoir étudié comment les systèmes de vérification de logiciel ou de matériel (basés sur l'exécution symbolique) utilisent et gèrent la Condition de Chemin, nous présentons l'approche que nous avons retenue.

Les données manipulées par le système dédié à la preuve de logiciel de KING sont de type entier. Aucune spécification n'étant fournie, la Condition de Chemin est initialement vraie.
A la rencontre d'un test, le calcul d'implication d'une valeur de vérité de la condition par la Condition de Chemin est effectué. Si la condition (ou la négation de la condition) est impliquée par la Condition de Chemin, l'exécution symbolique se poursuit sur le chemin d'exécution déterminé. Par contre, si aucune valeur de vérité n'est impliquée, alors le système construit les deux chemins d'exécution, et associe sa Condition de Chemin à chacun d'eux. Cette approche conduit à la construction de l'arbre d'exécution complet.
Les expressions symboliques arithmétiques sont représentées sous une forme normalisée, et sont manipulées par un démonstrateur de théorèmes sur les entiers, présenté en détail dans [KIN69].

Quant aux travaux plus spécifiques à la vérification de matériel, notons que les systèmes définis dans [WAG77, CAR79, PIT82] construisent l'arbre d'exécution complet, dans le but d'effectuer la preuve globale de la description. Par contre, [HAS87] et [GIA89] prévoient la vérification partielle de la description, en ne construisant qu'un chemin d'exécution symbolique.
Tous ces systèmes s'adressent à des descriptions de circuits au niveau transfert de registres, et se ramènent au niveau du bit pour la représentation et la manipulation des données.

Comme nous l'avons précisé, nous choisissons de ne pas construire l'arbre d'exécution dans son intégralité lors de la simulation symbolique, mais de suivre un chemin d'exécution donné [GIA91, ROG92b].
La Condition de Chemin est alors utilisée à chaque rencontre d'un test, ceci afin de laisser le choix à l'utilisateur de la valeur de la condition à évaluer. Or, dans le cas où la Condition de Chemin courante fixe une valeur pour la condition, il n'est pas utile d'interroger l'utilisateur systématiquement [GIA91, LAR92].
Il suffit donc de calculer si la Condition de Chemin implique la valeur "vrai" ou la valeur "faux" pour la condition à évaluer, ou si au contraire choisir une valeur ("vrai" ou "faux") pour la condition n'introduit aucune contradiction.

En résumé, à la rencontre d'une condition à évaluer, nous effectuons dans l'ordre les opérations suivantes :
- déterminer si la condition peut être directement évaluée (les éléments qui la composent ont des valeurs numériques). Cette tâche est effectuée par le simulateur,
- dans la négative, calculer si la Condition de Chemin détermine une valeur de

vérité pour la condition. Pour cela, le simulateur fait appel au module de gestion de la Condition de Chemin. Si aucune valeur de vérité n'est déduite, alors le choix sur la valeur de vérité de la condition est libre,

- le module de gestion de la Condition de Chemin renvoie alors la valeur de vérité du test au simulateur ("vrai", "faux", "libre"). Dans les deux premiers cas, le simulateur doit emprunter le chemin d'exécution associé à cette valeur de vérité. Suivre l'autre chemin introduirait une incohérence. Par contre, si la réponse est "libre", l'utilisateur est interrogé sur la valeur de la condition. Le simulateur renvoie alors au module de gestion de la Condition de Chemin la valeur de la condition afin que la mise à jour de la Condition de Chemin soit effectuée.

6 - Conclusion

La preuve de validité d'un chemin ou d'un sous-arbre de l'arbre d'exécution symbolique d'un programme VHDL constitue une des techniques de vérification de descriptions de matériel.

Pour apporter cette preuve, il est nécessaire d'extraire le comportement de la description ; c'est le rôle du simulateur symbolique.

Celui-ci comporte d'une part un noyau de simulation, et d'autre part un système capable de gérer la Condition de Chemin.

Le noyau de simulation symbolique est présenté dans le mémoire de R. ROGACKI [ROG92a].

Quant à la présentation de la gestion de la Condition de Chemin, elle fait l'objet du présent rapport.

Chapitre 2 - Méthodologie générale

1 - Introduction

En cours de simulation symbolique, lorsque le simulateur n'est pas en mesure d'évaluer une condition, l'approche que nous avons choisie consiste à déterminer si la valeur de vérité de cette condition peut être déduite de la Condition de Chemin : c'est la gestion de la Condition de Chemin.

En fonction du résultat :

- le simulateur doit impérativement suivre un chemin d'exécution (qui correspond à la valeur de vérité déduite),
- l'utilisateur choisit la valeur de la condition ; le simulateur emprunte alors le chemin d'exécution associé, et la Condition de Chemin est enrichie de la contrainte définie par la condition, ou la négation de la condition, selon le choix de l'utilisateur.

Nous avons développé un système permettant de gérer la Condition de Chemin et son évolution au cours de la simulation symbolique.

Nous définissons dans ce chapitre les concepts nécessaires à cette gestion, à savoir la structure de toute expression conditionnelle, et le détail de la gestion de la Condition de Chemin :

- le mécanisme de construction de la Condition de Chemin,
- le mécanisme de détermination de la liberté du choix de la valeur de vérité

d'une condition (évaluation d'un test),

- le mécanisme de mise à jour de la Condition de Chemin.

2 - Nature des éléments

Au cours de la simulation symbolique, la valeur d'un signal ou d'une variable du programme VHDL est soit numérique, soit symbolique.

Dire que cette **valeur** est **symbolique** signifie que son expression est constituée de symboles et éventuellement de valeurs numériques.

Les symboles présents dans cette expression sont des **constantes symboliques**. Contrairement aux variables ou aux signaux, leur valeur ne change jamais.

Par exemple, considérons qu'à une date t_0, les variables et signaux du programme ont tous comme valeur la constante symbolique formée de leur nom précédé d'un $:

A la date t_0, les variables a,b,et c ont respectivement les valeurs symboliques $a, $b et $c, avec $a, $b et $c **constantes symboliques**.

Si à la date t_1, le simulateur exécute l'instruction : "c := a + 2*b - c + 6", alors nous aurons :

a = $a

b = $b

c = $a + 2*$b - $c + 6

Dans cet exemple, la variable c a changé de valeur, celle-ci reste symbolique, mais la constante symbolique $c est inchangée.

Les éléments de base manipulés par le module de gestion de la Condition de Chemin ne sont donc pas des variables, mais des constantes symboliques.

3 - Construction de la Condition de Chemin

• La **Condition de Chemin spécifiée** est composée de la première partie de la spécification ; c'est un ensemble de contraintes qualifiées temporellement sur les valeurs des signaux et variables du programme VHDL à vérifier (Cf. §3 du Chapitre 1).

• Pour une date t de simulation, les conditions de la Condition de Chemin Spécifiée qui sont vérifiées pour cette date (leur qualification temporelle contient t) sont extraites. Ces expressions ont la propriété d'être toutes vraies à cette date. Leur conjonction représente alors l'ensemble des conditions spécifiées, vraies à la date t ; cette expression est fonction des variables et signaux du programme VHDL, elle est atemporelle et sa syntaxe est en accord avec la grammaire du langage de description : c'est la **Condition de Chemin Extraite** pour la date t.

• La **Condition de Chemin Symbolique** est directement obtenue par substitution des variables et signaux présents dans la Condition de Chemin

Extraite, par leur valeur symbolique (expression de constantes symboliques).

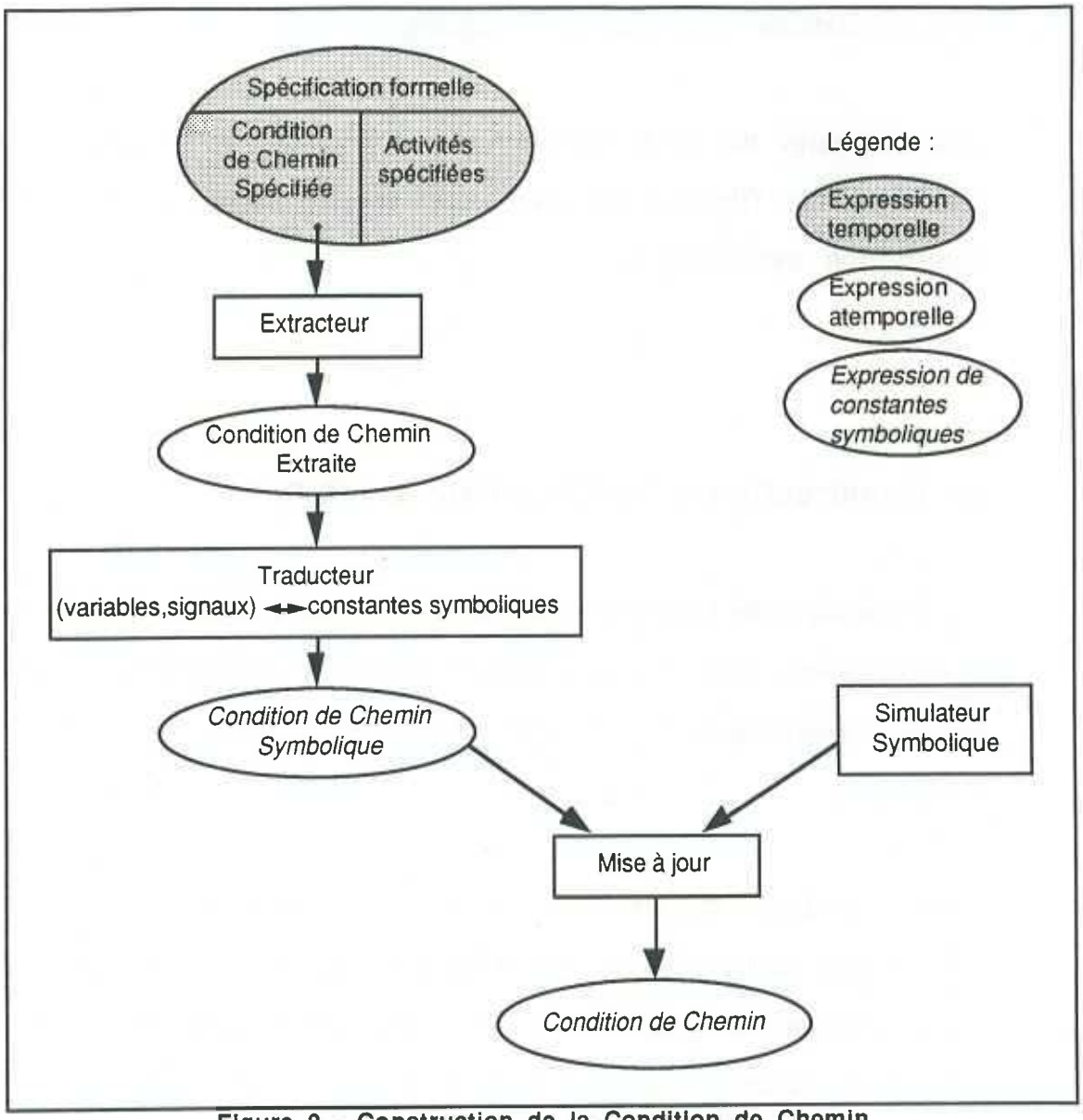


Figure 2 - Construction de la Condition de Chemin

• La **Condition de Chemin** est initialement vraie. Elle est ensuite mise à jour :

- par la Condition de Chemin Symbolique (obtenue à partir de la Condition de Chemin Spécifiée, via la Condition de Chemin Extraite) pour toute nouvelle date de simulation,

- par les conditions rencontrées par le simulateur symbolique, et dont la valeur

de vérité n'est pas contenue par la Condition de Chemin.

Ce processus de construction est illustré par la Figure 2.

Finalement, la Condition de Chemin est fonction de constantes symboliques ; elle est donc vraie pour toute date de simulation (antérieure ou postérieure à t).

Remarque fondamentale : consistance de la Condition de Chemin

Par essence, une spécification formelle est correcte. Cela signifie que les conditions mises en jeu ne se contredisent pas. La Condition de Chemin initiale, extraite de la spécification, est donc consistante.

Nous pouvons alors garantir que la Condition de Chemin reste consistante tout au long de la simulation. En effet, elle n'est mise à jour (enrichie d'une nouvelle contrainte), que si aucune valeur de vérité pour cette condition (vrai ou faux) n'en est déduite.

Nous verrons dans les chapitres suivants que cette propriété est fondamentale, car elle constitue la condition sine qua non à l'évaluation d'un test lors de la simulation symbolique.

4 - Exemple

Nous illustrons de manière informelle par un exemple simple :

- le déroulement de la simulation symbolique d'une description VHDL,
- la construction de la Condition de Chemin,
- l'utilisation du module de gestion de la Condition de Chemin.

La Figure 3 contient la description VHDL d'une entité E. Ses ports d'entrée sont les signaux :

- reset et cde de type booléen,

- a et b de type entier.

Le port de sortie de E est de type entier et se nomme resultat.

x est une variable entière interne au process P de l'architecture Behavior qui définit le comportement de l'entité E. Ce process est réactivé par les changements de valeur qui interviennent sur reset, a ou b.

```
Entity E is
port (a,b : in integer;
      cde,reset : in boolean;
      resultat : out integer);
end E;

Architecture Behavior of E is
begin
   P : process P(reset,a,b)
   variable x : integer;
   begin
      if reset
      then resultat <= 0;
      else
         if cde
         then x := a+b;
         else x := a-b;
         end if;
         if (x<0)
         then resultat <= -x;
         else resultat <= x;
         end if;
      end if;
   end process;
end Behavior;
```

Figure 3 - Description VHDL de l'entité E et de son architecture Behavior

Supposons que la partie Condition de Chemin Spécifiée de la spécification formelle fournie par l'utilisateur soit composée de l'unique condition :

(a + b > 5, [0..100]), qui signifie que la valeur de (a+b) est supérieure à 5 pour toute date de simulation comprise entre 0 et 100.

Initialement, les variables et signaux qui interviennent dans le process `P` ont une valeur symbolique :

`a = $a`

`b = $b`

`cde = $cde`

`reset = $reset`

`resultat = $resultat`

`x = $x`

La Condition de Chemin CC est vraie : CC = 1.

A la date de simulation 0, la Condition de Chemin est mise à jour :

- La Condition de Chemin Extraite est composée des conditions de la Condition de Chemin Spécifiée vraies pour la date 0 : `(a + b > 5)`
- La Condition de Chemin Symbolique est alors obtenue en remplaçant les signaux présents dans la Condition de Chemin Extraite par leur valeur symbolique : `($a + $b > 5)`
- Finalement, la Condition de Chemin est mise à jour :

CC := CC $\wedge$ `($a + $b > 5)`, d'où CC = `($a + $b >5)`.

Le simulateur exécutealors les instructions du programme VHDL.

La première instruction est conditionnelle ; elle est fonction de la valeur de vérité de l'expression `(reset)`. Le simulateur remplace ce signal par sa valeur. Il doit donc évaluer l'expression `($reset)`. La valeur de cette expression symbolique n'est pas connue, et ne peut pas être déduite de la Condition de Chemin CC. Dans ce cas, l'utilisateur intervient et fixe la valeur de vérité de ce test. Supposons qu'il réponde que la valeur de vérité est "faux". Alors avant de

poursuivre la simulation sur le chemin déterminé par cette valeur de vérité, la Condition de Chemin est mise à jour. On obtient : CC = ($a + $b > 5) $\wedge$ $\overline{\$reset}$.

Le simulateur exécute maintenant l'instruction conditionnelle suivante qui dépend de la valeur de vérité de l'expression (cde). De la même façon que précédemment, comme la valeur de vérité de ($cde) n'est pas connue, et n'est pas contenue dans la Condition de Chemin, l'utilisateur est interrogé. Supposons qu'il indique ici que la valeur de vérité de cette expression est "vrai". La Condition de Chemin est mise à jour et vaut maintenant :

CC = ($a + $b > 5) $\wedge$ $\overline{\$reset}$ $\wedge$ $cde.

Après exécution de l'instruction (x := a+b), la valeur de x est la suivante:

x = $a + $b

Le simulateur exécute ensuite l'instruction conditionnelle : (x<0). En substituant la variable x par sa valeur symbolique, il faut évaluer l'expression : ($a + $b <0). L'étude de cette condition dans le contexte de la Condition de Chemin permet de conclure qu'elle est fausse. En effet, la Condition de Chemin contient l'expression ($a + $b > 5).

De ce fait, **sans faire intervenir l'utilisateur**, le simulateur symbolique poursuit l'exécution du programme. Au moment où le process se trouve suspendu dans l'attente d'un événement sur un des signaux sensibles, l'état des données est le suivant :

a = $a

b = $b

cde = $cde

reset = $reset

resultat = $a + $b

x = $a + $b

et la Condition de Chemin CC vaut : CC = ($a + $b > 5) $\wedge$ $\overline{\$reset}$ $\wedge$ $cde.

5 - Structure générale d'une expression conditionnelle

Parmi les expressions manipulées par le simulateur symboliques, seules celles dont la valeur est booléenne peuvent intervenir dans la gestion de la Condition de Chemin. En effet, le simulateur fait appel au module de gestion de la Condition de Chemin quand il se trouve dans l'incapacité d'évaluer la valeur de vérité d'une condition.
Nous définissons ici la structure des conditions qui peuvent apparaître dans le module de gestion de la Condition de Chemin, compte tenu de la syntaxe de VHDL et des transformations nécessaires à l'expression des conditions dans la simulation symbolique.

Les éléments de base manipulés dans ces conditions sont des constantes symboliques.
Ces constantes symboliques sont associées aux variables et signaux de la description VHDL. Dans ce langage, les éléments sont typés ; les constantes symboliques le sont également (Cf. §3 du Chapitre 1).

Une condition (dans le module de gestion de la Condition de Chemin) est alors une expression booléenne dont les termes sont :
- des expressions formées de constantes symboliques booléennes et des opérateurs logiques classiques (OU, ET, négation). Nous pouvons également

envisager l'utilisation du OU-EXCLUSIF, du NOR, du NAND, ...

- des expressions dont les constantes symboliques sont arithmétiques (à valeurs entières) et dont le résultat est booléen. Il s'agit alors de deux expressions arithmétiques séparées par un opérateur relationnel ($=, \neq, <, \leq, >, \geq$).

Exemple :

Si a, b et c sont des constantes symboliques booléennes, et x, y et z sont des constantes symboliques arithmétiques, alors l'expression $a \vee (b \wedge (x+y \leq z-1) \wedge \overline{c})$ est une condition pour le module de gestion de la Condition de Chemin.

Remarque importante :

Il est à noter que VHDL ne prévoit pas de conversions de type implicites. Cela signifie qu'une variable déclarée comme entière ne sera manipulée que comme telle, et il ne sera pas possible de manipuler un des bits qui la compose.
Les constantes symboliques booléennes et arithmétiques sont donc totalement disjointes.

Les conditions manipulées dans le module de gestion de la Condition de Chemin sont donc des expressions logiques de constantes symboliques booléennes, et d'inéquations qui mettent en jeu des constantes symboliques arithmétiques.

Nous avons alors défini trois classes d'expressions :

- les expressions purement booléennes, qui ne mettent en jeu que des constantes symboliques booléennes,
- les expressions booléennes dont les éléments sont des constantes symboliques arithmétiques,

- les expressions mixtes (qui manipulent des constantes symboliques booléennes et des constantes symboliques arithmétiques).

Avant d'étudier dans le détail comment opérer sur chacune de ces classes d'expressions, nous allons établir les règles de manipulation communes à tout type de conditions.

6 - Evaluation d'un test au cours de la simulation symbolique

Connaissant la Condition de Chemin CC, nous voulons maintenant déterminer si la valeur de vérité d'une condition C est fixée par CC. Nous cherchons donc à déterminer si le choix de la valeur de vérité de C est libre ou non.

- Si ce choix n'est pas libre, le simulateur devra impérativement suivre le chemin correspondant à la valeur de la condition, et l'utilisateur ne devra pas intervenir (il pourrait effectuer "le mauvais choix" qui rendrait la Condition de Chemin inconsistante par l'opération de mise à jour ; le chemin suivi par le simulateur ne serait pas un chemin d'exécution valide).
- Si ce choix est libre, le simulateur devra soit suivre les deux chemins d'exécution symbolique possibles (en maintenant les Conditions de Chemin associées), soit choisir un des deux chemins d'exécution (au hasard, ou en demandant à l'utilisateur) et mettre à jour la Condition de Chemin.

Nous définissons une variable booléenne associée à la condition C, appelée **CHOIX-LIBRE** (notée CL), telle que :

- CL est vraie si la connaissance de CC (vraie) ne permet pas de déduire la valeur de vérité de C,

- CL est fausse si la valeur de vérité de CC (vraie) permet de déterminer la valeur de vérité de la condition C (vraie ou fausse).

Notation :
Le calcul de détermination de la valeur de C par CC sera noté (CC ➤ C). Nommons cette opération "opération de choix".
Le résultat de l'opération de choix ➤ est booléen. Il vaut 0 si CC permet de déterminer la valeur de vérité de C, 1 sinon.
La variable de choix libre est donc le résultat de cette opération. Nous pouvons alors écrire : CL = (CC ➤ C).

Sans préjuger de la forme des conditions CC et C, voyons dans quelles configurations la variable CL est vraie ou fausse.

6.1 - Préliminaires

Nous allons étudier les différentes relations qui peuvent lier les ensembles de définition et de solutions de deux fonctions booléennes. Dans chacun des cas, nous établirons ce que la connaissance de la première fonction apporte sur la seconde. Ces résultats seront alors directement applicables pour le calcul de la variable de choix associée à une condition C, en considérant que la première fonction booléenne est la Condition de Chemin, et que la deuxième représente la condition C.

Soit E et F deux ensembles respectivement de n et m variables, B = {0 , 1}.
On définit $\mathcal{E}$ et $\mathcal{F}$ deux ensembles respectivement de dimension n et m à valeurs

dans un espace vectoriel X.

On a : $E = \{e_1, \ldots, e_n\}$, $F = \{f_1, \ldots, f_m\}$,

$\mathcal{E} = \{(\alpha_1, \ldots, \alpha_n) \in X^n\}$, $\mathcal{F} = \{(\beta_1, \ldots, \beta_m) \in X^m\}$.

On définit la bijection b_E qui est telle que :

$\forall\ \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathcal{E}$,

$\forall\ i \in [1,n],\ \exists!\ j\ /\ b_E(\alpha_i) = e_j$

De même, on définit la bijection b_F qui est telle que :

$\forall\ \beta = (\beta_1, \ldots, \beta_m) \in \mathcal{F}$,

$\forall\ i \in [1,m],\ \exists!\ j\ /\ b_F(\beta_i) = f_j$

Les composantes éléments de $\mathcal{E}$ et $\mathcal{F}$ sont donc respectivement indicés par les éléments de E et F.

Soit deux fonctions $f : \mathcal{E} \to B$, et $g : \mathcal{F} \to B$.

On définit l'ensemble S_f de solutions de f : $S_f = \{x \in \mathcal{E}\ /\ f(x) = 1\}$.

L'ensemble $S_{\bar{f}} = \{x \in \mathcal{E}\ /\ f(x) = 0\}$ est tel que $S_f \cap S_{\bar{f}} = \varnothing$, et $S_f \cup S_{\bar{f}} = \mathcal{E}$.

De même, l'ensemble de solutions de g est : $S_g = \{x \in \mathcal{F}\ /\ g(x) = 1\}$,

et $S_{\bar{g}} = \{x \in \mathcal{F}\ /\ g(x) = 0\}$. On a : $S_g \cap S_{\bar{g}} = \varnothing$, et $S_g \cup S_{\bar{g}} = \mathcal{F}$.

On note I l'ensemble : $I = E \cap F$.

L'ensemble I est de cardinal p, avec $p \leq \min(n,m)$.

On définit alors l'ensemble $\mathcal{I}$ de dimension p à valeurs dans X.

La bijection b_I est définie de la façon suivante :

$b_I : [1,p] \to [1,n] \times [1,m]$

$b_I(\alpha) = (\beta,\gamma) \Leftrightarrow b_E(\beta) = b_F(\gamma)$

La projection p_1 est telle que si $b_I(\alpha) = (\beta,\gamma)$, alors $p_1(b_I(\alpha)) = \beta$.

De même, p_2 est telle que si $b_I(\alpha) = (\beta,\gamma)$, alors $p_2(b_I(\alpha)) = \gamma$.

On définit alors l'ensemble de solutions de f restreint à l'ensemble I de variables communes aux fonctions f et g de la façon suivante :

$S_{f/I} = \{x \in \mathfrak{I} / \exists\, y \in S_f / p_{p_1(b_I(1)), \ldots, p_1(b_I(p))}(y) = x\}$.

($p_{p_1(b_I(1)), \ldots, p_1(b_I(p))}(y)$ est le projeté de y suivant les variables communes à E et F)

De même, l'ensemble des solutions de g restreint à l'ensemble I est :

$S_{g/I} = \{x \in \mathfrak{I} / \exists\, y \in S_g / p_{p_2(b_I(1)), \ldots, p_2(b_I(p))}(y) = x\}$.

Exemple :

E = {a , b , c , d}, F = {a , x , b}. Si X = {0 , 1}, alors

$\mathfrak{E}$ = {(0,0,0,0) , (0,0,0,1) , ..., (1,1,1,1)} et $\mathfrak{F}$ = {(0,0,0) , (0,0,1), ..., (1,1,1)}.

Le rôle des fonctions b_E et b_F est d'associer un indice aux éléments de E et F.

Par exemple, on peut choisir :

$b_E(1) = a$, $b_E(2) = b$, $b_E(3) = c$, $b_E(4) = d$, et $b_F(1) = a$, $b_F(2) = x$, $b_F(3) = b$.

L'ensemble I contient les variables communes à E et F, soit : I = {a , b}.

Alors $b_I(1) = (1,1)$, et $b_I(2) = (2,3)$.

Supposons que $f(a,b,c,d) = a \wedge b$,et $g(a,x,b) = a \vee x$. Alors on a :

S_f = {(1,1,0,0) , (1,1,0,1) , (1,1,1,0) , (1,1,1,1)}

et S_g = {(1,0,0) , (1,0,1) , (0,1,0) , (0,1,1) , (1,1,0) , (1,1,1)}

La restriction des éléments de S_f et S_g par rapport à l'ensemble I est obtenue en éliminant de chacun de ces éléments les composantes qui correspondent à des variables qui ne figurent pas dans I. Dans le cas présent, il ne faut conserver des éléments de S_f que les composantes qui correspondent aux variables a et b (rangs 1 et 2). De même, il ne faut conserver des éléments de S_g que les composantes qui correspondent aux variables a et b (rangs 1 et 3). Finalement :

$S_{f/I}$ = {(1,1)} et $S_{g/I}$ = {(1,0) , (0,1) , (1,1)}.

Hypothèse : $S_{f/I} \neq \emptyset$

1er cas : $E \cap F = \emptyset$

L'ensemble de solutions d'une fonction étant un sous-ensemble de son ensemble de définition, le fait que l'intersection de E et F soit vide implique que l'ensemble I est vide. Donc $S_{f/I} \cap S_{g/I} = \emptyset$.

➡ La connaissance de la fonction f ne donne aucune indication sur la fonction g, et réciproquement.

2ème cas : $E \cap F \neq \emptyset$ et $S_{f/I} \subset S_{g/I}$

$\forall x \in S_f, \exists y \in S_{f/I} / y = p_{p_1(b_I(1)), ..., p_1(b_I(p))}(x)$ (y est le projeté de x suivant les variables communes)

Or, comme $S_{f/I} \subset S_{g/I}$, $y \in S_{g/I}$

Alors $\exists z \in S_g / p_{p_2(b_I(1)), ..., p_2(b_I(p))}(z) = y$ (par définition de $S_{g/I}$).

➡ si f est vraie, alors g est vraie

3ème cas : $E \cap F \neq \emptyset$ et $S_{g/I} \subset S_{f/I}$

$\exists x \in S_{f/I} \cap S_{g/I}$

Dans ce cas, si f est vraie, alors g est vraie (Cf. 1er cas).

ET

$\exists x' / x' \in S_{f/I}$ et $x' \notin S_{g/I}$

Alors par définition de $S_{g/I}$, $\exists y \in S_{\overline{g}} / p_{p_2(b_I(1)), ..., p_2(b_I(p))}(y) = x'$

Dans ce cas, si f est vraie, alors g est fausse.

➡ la valeur de f ne permet pas de conclure sur la valeur de g.

4ème cas : $\mathcal{G} \cap \mathcal{F} \neq \emptyset$ et $S_{f/I} \cap S_{g/I} \neq \emptyset$ et $S_{f/I} \cap S_{g/I} \neq S_{f/I}$ et $S_{f/I} \cap S_{g/I} \neq S_{g/I}$

$\exists x \in S_{f/I} \cap S_{g/I}$

Dans ce cas, si f est vraie, alors g est vraie (Cf. 1er cas).

ET

$\exists\ x' / x' \in S_{f/I}$ et $x' \notin S_{g/I}$

Alors par définition de $S_{g/I}$, $\exists\ y \in S_{\bar{g}} / p_{p_2(b_I(1)), \ldots, p_2(b_I(p))}(y) = x'$

Dans ce cas, si f est vraie, alors g est fausse.

ET

$\exists\ x'' / x'' \in S_{g/I}$ et $x'' \notin S_{f/I}$

Alors par définition de $S_{f/I}$, $\exists\ y' \in S_{\bar{f}} / p_{p_1(b_I(1)), \ldots, p_1(b_I(p))}(y') = x''$

Dans ce cas, si f est fausse, alors g est vraie.

➡ la valeur de f ne permet pas de conclure sur la valeur de g.

<u>5ème cas</u> : $E \cap F \neq \emptyset$ et $S_{f/I} \cap S_{g/I} = \emptyset$

$\forall\ x \in S_f, \exists\ y \in S_{f/I} / y = p_{p_1(b_I(1)), \ldots, p_1(b_I(p))}(x)$ (y est le projeté de x suivant les variables communes)

Or, $y \notin S_{g/I}$

Alors $\exists\ z \in S_{\bar{g}} / p_{p_2(b_I(1)), \ldots, p_2(b_I(p))}(z) = y$ (par définition de $S_{g/I}$).

➡ si f est vraie, alors g est fausse.

En conclusion, si les fonctions f et g sont indépendantes (fonction d'ensembles de variables disjoints), alors la donnée de la valeur de f ne fournit aucune indication sur la valeur de g. Par contre, si les ensembles de définition des deux fonctions f et g intersectent, alors il faut étudier l'intersection des ensembles de solutions des deux fonctions pour déterminer si la donnée de f permet de conclure sur la valeur de g.

6.2 - Détermination de la valeur de la variable de choix

En considérant que f est la fonction Condition de Chemin CC, et g la fonction qui représente la condition C, alors nous pouvons directement exploiter les résultats du paragraphe précédent et établir les résultats suivants :

• si la Condition de Chemin et la condition à évaluer sont fonction d'ensembles disjoints, alors le choix est libre (CL = 1),

• si l'ensemble de solutions de C (resp. $\overline{C}$) est inclus dans l'ensemble de solutions de CC, alors le choix est libre (CL = 1),

• si les ensembles de solutions de CC et C (resp. $\overline{C}$) intersectent sans se contenir, alors le choix est libre (CL = 1),

• si l'ensemble de solutions de CC est inclus dans l'ensemble de solutions de C (resp. $\overline{C}$), alors CC vraie implique C vraie (resp. fausse) et le choix n'est pas libre (CL = 0).

Dans le cas où le choix est libre (CL = (CC ➤ C) = 1), nous voyons apparaître deux cas distincts :

- CC et C sont fonction des mêmes variables, mais l'intersection des espaces de solutions ne permet pas de conclure,
- les ensembles de définitions de CC et C sont disjoints.

La détermination de la variable de choix s'effectue donc en deux étapes : calculer tout d'abord l'intersection entre les ensembles de définition des deux fonctions (CC et C). Si elle est vide, alors le choix est libre (CL = 1). Sinon, il faut étudier les positions relatives des ensembles de solutions des deux fonctions.

6.3 - Opérations sur les variables de choix

Nous serons amenés, dans certaines situations, à décomposer les expressions conditionnelles en plusieurs sous-expressions afin de parvenir à évaluer la variable de choix d'une condition. En effet, il est nécessaire de décomposer les conditions (que ce soit la Condition de Chemin ou la condition à évaluer) lorsqu'elles sont trop complexes (par exemple, elles font intervenir des constantes symboliques de types différents).

Une condition est exprimée à l'aide de l'ensemble fonctionnellement complet d'opérateurs (ET, OU, PAS). Aussi, il est nécessaire de définir le comportement de la variable de choix associée à une condition face à la négation de cette condition. De même, nous devons établir les règles liant les variables de choix de la conjonction ou la disjonction de deux conditions. Ce travail doit également être fait pour le cas où la Condition de Chemin est le OU ou le ET de plusieurs expressions conditionnelles.

6.3.1 - Négation

Soit CC la Condition de Chemin, C une condition.

CL = (CC ➤ C).

Comment déterminer CL' = (CC ➤ $\overline{C}$) ?

• Nous savons que CL = 0 si et seulement si $S_{CC} \subset S_C$ ou $S_{CC} \subset S_{\overline{C}}$.

Supposons que $S_{CC} \subset S_C$, alors la négation de $\overline{C}$ ($\overline{\overline{C}} = C$) est vraie si CC est vraie. Donc CL' = 0.

De même, si $S_{CC} \subset S_{\overline{C}}$, alors $\overline{C}$ est vraie si CC est vraie. Donc CL' = 0.

• Si CL = 1, alors quelle que soit la configuration des ensembles de définition et de solutions de CC et C, on vérifie aisément que CL' = 1.

Ce résultat (CL' = CL) était prévisible vu que le calcul de la variable de choix s'effectue sur la condition et sa négation.

Donc CL' = (CC ➤ $\overline{C}$) = (CC ➤ C) = CL

6.3.2 - Opérations logiques binaires

Sachant déterminer la valeur de la variable de choix pour une condition donnée, nous allons maintenant étudier comment se combinent deux variables de choix.

1er cas :

Soit CC la condition de chemin, C_1 et C_2 deux conditions,

CL_1 = (CC ➤ C_1),

CL_2 = (CC ➤ C_2).

* Comment déterminer CL = ((CC ➤ C_1) <op> (CC ➤ C_2)) où <op> est un opérateur logique binaire ?

Nous allons successivement étudier les cas du OU et du ET logiques :

• CL = ((CC ➤ C_1) ∨ (CC ➤ C_2))

Raisonnons sur les ensembles de solutions.

Le choix n'est pas libre si l'ensemble de solutions de CC est inclus dans l'ensemble de solutions de C_1 ou dans l'ensemble de solutions de C_2. Le choix du OU des deux expressions est donc libre si et seulement si les deux expressions sont à choix libre ; d'où la table de vérité :

CL_1	CL_2	$CL = ((CC \rightarrow C_1) \vee (CC \rightarrow C_2))$
0	0	0
0	1	0
1	0	0
1	1	1

Nous avons alors : $CL = ((CC \rightarrow C_1) \vee (CC \rightarrow C_2)) = CL_1 \wedge CL_2$

La variable de choix du OU de deux expressions est donc obtenue en effectuant le ET des variables de choix associées à chacune des expressions.

- $CL = ((CC \rightarrow C_1) \wedge (CC \rightarrow C_2))$

Le choix n'est pas libre si l'ensemble de solutions de CC est inclus dans l'ensemble de solutions de C_1 et dans l'ensemble de solutions de C_2. Le choix du ET des deux expressions est donc libre dès qu'une des deux expressions est à choix libre ; d'où la table de vérité :

CL_1	CL_2	$CL = ((CC \rightarrow C_1) \vee (CC \rightarrow C_2))$
0	0	0
0	1	1
1	0	1
1	1	1

Nous avons alors : $CL = ((CC \rightarrow C_1) \wedge (CC \rightarrow C_2)) = CL_1 \vee CL_2$

La variable de choix du ET de deux expressions est donc obtenue en effectuant le OU des variables de choix associées à chacune des expressions.

* Comment déterminer $CL = (CC \rightarrow (C_1 \text{ <op> } C_2))$ où <op> est un opérateur

logique binaire ?

Nous allons successivement étudier les cas du OU et du ET logiques :

- $CL = (CC \rightarrow (C_1 \vee C_2))$

Le choix n'est pas libre si l'ensemble de solutions de CC est inclus dans l'ensemble de solutions de $(C_1 \vee C_2)$. Or, $E_{C_1 \vee C_2} = E_{C_1} \cup E_{C_2}$. Le choix n'est donc pas libre si l'ensemble de solutions de CC est inclus dans l'ensemble de solutions de C_1 ou dans l'ensemble de solutions de C_2. Le choix du OU est donc libre si et seulement si les deux conditions sont à choix libre ; d'où la table de vérité :

CL_1	CL_2	$CL = (CC \rightarrow (C_1 \vee C_2))$
0	0	0
0	1	0
1	0	0
1	1	1

Nous avons alors : $CL = (CC \rightarrow (C_1 \vee C_2)) = CL_1 \wedge CL_2$

La variable de choix du OU de deux conditions est donc obtenue en effectuant le ET des variables de choix associées à chacune des conditions.

- $CL = (CC \rightarrow (C_1 \wedge C_2))$

Le choix n'est pas libre si l'ensemble de solutions de CC est inclus dans l'ensemble de solutions de $(C_1 \wedge C_2)$. Or, $E_{C_1 \wedge C_2} = E_{C_1} \cap E_{C_2}$. Le choix n'est donc pas libre si l'ensemble de solutions de CC est inclus dans l'ensemble de solutions de C_1 et dans l'ensemble de solutions de C_2. Le choix du ET est donc libre dès qu'une des deux conditions est à choix libre ; d'où la table de vérité :

CL_1	CL_2	$CL = (CC \blacktriangleright (C_1 \wedge C_2))$
0	0	0
0	1	1
1	0	1
1	1	1

Nous avons alors : $CL = (CC \blacktriangleright (C_1 \wedge C_2)) = CL_1 \vee CL_2$

La variable de choix du ET de deux conditions est donc obtenue en effectuant le OU des variables de choix associées à chacune des conditions.

2ème cas :

Soit CC_1 et CC_2 deux Conditions de Chemin, C une condition.

$CL_1 = (CC_1 \blacktriangleright C)$,

$CL_2 = (CC_2 \blacktriangleright C)$.

* Comment déterminer $CL = ((CC_1 \blacktriangleright C)$ <op> $(CC_2 \blacktriangleright C))$ où <op> est un opérateur logique binaire ?

Nous allons successivement étudier les cas du OU et du ET logiques :

• $CL = ((CC_1 \blacktriangleright C) \vee (CC_2 \blacktriangleright C))$

Le choix n'est pas libre si l'ensemble de solutions de CC_1 est inclus dans l'ensemble de solutions de C, ou si l'ensemble de solutions de CC_2 est inclus dans l'ensemble de solutions de C. Le choix du OU des deux expressions est donc libre si et seulement si les deux expressions sont à choix libre ; d'où la table de vérité :

CL_1	CL_2	$CL = ((CC_1 \rightarrow C) \vee (CC_2 \rightarrow C))$
0	0	0
0	1	0
1	0	0
1	1	1

Nous avons alors : $CL = ((CC_1 \rightarrow C) \vee (CC_2 \rightarrow C)) = CL_1 \wedge CL_2$

La variable de choix du OU de deux expressions est donc obtenue en effectuant le ET des variables de choix associées à chacune des expressions.

- $CL = ((CC_1 \rightarrow C) \wedge (CC_2 \rightarrow C))$

Le choix n'est pas libre si l'ensemble de solutions de CC_1 est inclus dans l'ensemble de solutions de C, et si l'ensemble de solutions de CC_2 est inclus dans l'ensemble de solutions de C. Le choix du ET des deux expressions est donc libre dès qu'une des deux expressions est à choix libre ; d'où la table de vérité :

CL_1	CL_2	$CL = ((CC_1 \rightarrow C) \wedge (CC_2 \rightarrow C))$
0	0	0
0	1	1
1	0	1
1	1	1

Nous avons alors : $CL = ((CC_1 \rightarrow C) \wedge (CC_2 \rightarrow C)) = CL_1 \vee CL_2$

La variable de choix du ET de deux expressions est donc obtenue en effectuant le OU des variables de choix associées à chacune des expressions.

* Comment déterminer CL = ((CC_1 <op> CC_2 ➤ C) où <op> est un opérateur logique binaire ?

Nous allons successivement étudier les cas du OU et du ET logiques :

• CL = (($CC_1 \vee CC_2$) ➤ C)

Le choix du OU des deux expressions qui forment la Condition de Chemin est libre dès qu'une des deux expressions est à choix libre ; d'où la table de vérité :

CL_1	CL_2	CL = (($CC_1 \vee CC_2$) ➤ C)
0	0	0
0	1	1
1	0	1
1	1	1

Nous avons alors : CL = (($CC_1 \vee CC_2$) ➤ C) = $CL_1 \vee CL_2$

La variable de choix du OU de deux expressions qui forment la Condition de Chemin est donc obtenue en effectuant le OU des variables de choix associées à chacune des expressions.

• CL = (($CC_1 \wedge CC_2$) ➤ C)

Le choix du ET des deux expressions qui forment la Condition de Chemin est libre si et seulement si les deux expressions sont à choix libre ; d'où la table de vérité :

CL_1	CL_2	$CL = ((CC_1 \wedge CC_2) \blacktriangleright C)$
0	0	0
0	1	0
1	0	0
1	1	1

Nous avons alors : $CL = ((CC_1 \wedge CC_2) \blacktriangleright C) = CL_1 \wedge CL_2$

La variable de choix du ET de deux expressions qui forment la Condition de Chemin est donc obtenue en effectuant le ET des variables de choix associées à chacune des expressions.

Nous avons déterminé les modes de combinaison de variables de choix associées à différentes conditions. Nous utiliserons ces propriétés lorsque nous serons confrontés à des expressions complexes, que nous décomposerons en termes de complexité moindre.

Retenons simplement que :

- les variables de choix associées à une condition ou à la négation de cette condition respectivement ont la même valeur.
- si une condition à évaluer est une conjonction (resp. disjonction) de termes, alors la valeur de la variable de choix associée à la condition est égale au résultat du OU (resp. ET) des valeurs des variables de choix associées à chacun des termes.
- si la Condition de Chemin est une conjonction (resp. disjonction) de termes, alors la valeur de la variable de choix associée à la condition à évaluer est égale au résultat du ET (resp. OU) des valeurs des variables de choix associées à la condition compte tenu du terme de la Condition de Chemin.

- la valeur de la variable de choix du OU (resp. ET) d'opérations de choix est égale au résultat du ET (resp. OU) des variables de choix associées à chacune des opérations.

7 - Mise à jour de la Condition de Chemin

Le simulateur demande au module de gestion de la Condition de Chemin de déterminer la valeur de la variable de choix-libre associée à une condition quand il n'est pas en mesure de l'évaluer directement, et qu'il a besoin de connaître la valeur de vérité de cette condition pour choisir le chemin d'exécution sur lequel il va continuer à évoluer.

Deux cas peuvent alors se présenter :

- la variable de choix-libre est fausse :

Dans ce cas, le module de gestion de la Condition de Chemin transmet la valeur de vérité de la condition au simulateur, qui emprunte alors le chemin déterminé par la Condition de Chemin.

- la variable de choix-libre est vraie :

Nous avons choisi de ne pas construire les deux chemins d'exécution possibles, mais de demander à l'utilisateur d'effectuer un choix concernant la valeur de vérité de la condition.

La Condition de Chemin doit alors tenir compte de ce choix ; elle est enrichie de la condition (ou de sa négation, suivant le choix de l'utilisateur).

Pratiquement, cette mise à jour consiste à effectuer le ET logique de la Condition de Chemin courante et de la condition (ou sa négation). Le résultat est la nouvelle Condition de Chemin, mise à jour.

8 - Améliorations algorithmiques pour le calcul de la variable de choix

8.1 - Introduction

Les descriptions de circuits manipulent généralement plusieurs dizaines de variables ou signaux. Or, nous pouvons remarquer que les conditions exprimées dans ces programmes ne lient pas toutes les variables entre elles ; cela s'explique par le fait que les entrées, sorties et états internes d'un circuit ne sont pas tous forcément liés.

Nous avons déterminé comment calculer la valeur de variable de choix d'une condition, connaissant la Condition de Chemin. Nous avons également défini les opérations sur les variables de choix pour le calcul de la variable de choix libre associée à une expression complexe.
Nous allons maintenant définir de façon plus précise dans quel cas le calcul de la variable de choix libre doit être effectué en intégralité, et dans quel cas nous pouvons conclure rapidement sur la valeur de la variable de choix.
Pour cela, nous allons étudier les différents cas pouvant se produire quant à la répartition des variables dans la Condition de Chemin et dans la condition à évaluer ; nous définirons alors quand et comment calculer la valeur de la variable de choix libre.

8.2 - Exemples

Nous allons montrer sur des exemples que le calcul de la valeur de la variable

de choix peut être fortement simplifié dans certains cas. Il s'agit dans tous les cas d'étudier l'intersection des ensembles de variables présentes dans les différents termes de la Condition de Chemin et de la condition, et de tirer les conclusions qui s'imposent.

• Supposons que la Condition de Chemin CC soit la conjonction de deux expressions CC_1 et CC_2.

CC_1 est fonction des variables x, y et z, CC_2 est fonction de t et u.

La condition à évaluer est fonction de la variable y.

Notons $CL_1 = (CC_1 \blacktriangleright C)$, et $CL_2 = (CC_2 \blacktriangleright C)$.

Nous avons vu dans le paragraphe précédent que $CL = ((CC_1 \wedge CC_2) \blacktriangleright C)$ est égale à $CL_1 \wedge CL_2$.

Etant donné que les ensembles de définition de CC_2 et C sont disjoints, nous savons que $CL_2 = (CC_2 \blacktriangleright C) = 1$.

Finalement, $CL = CL_1 \wedge CL_2 = CL_1 \wedge 1 = CL_1$.

La détermination de la valeur de la variable de choix se résume donc au calcul de $(CC_1 \blacktriangleright C)$.

• Supposons que la Condition de Chemin CC soit la disjonction de deux expressions CC_1 et CC_2.

CC_1 est fonction des variables x, y et z, CC_2 est fonction de t et u.

La condition à évaluer est fonction de la variable y.

Notons $CL_1 = (CC_1 \blacktriangleright C)$, et $CL_2 = (CC_2 \blacktriangleright C)$.

Nous avons vu dans le paragraphe précédent que $CL = ((CC_1 \vee CC_2) \blacktriangleright C)$ est égale à $CL_1 \vee CL_2$.

Etant donné que les ensembles de définition de CC_2 et C sont disjoints, nous avons : $CL_2 = (CC_2 \blacktriangleright C) = 1$.

Finalement, $CL = CL_1 \vee CL_2 = CL_1 \vee 1 = 1$.

Il est donc inutile de déterminer la valeur de CL_1, CL est toujours vraie.

• Supposons que la condition à évaluer C soit la conjonction de deux expressions C_1 et C_2.

C_1 est fonction de la variable y, C_2 est fonction de t et u.

La Condition de Chemin est fonction des variable x, y et z.

Notons $CL_1 = (CC \blacktriangleright C_1)$, et $CL_2 = (CC \blacktriangleright C_2)$.

Nous avons vu dans le paragraphe précédent que $CL = (CC \blacktriangleright (C_1 \wedge C_2))$ est égale à $CL_1 \vee CL_2$.

Etant donné que les ensembles de définition de CC et C_2 sont disjoints, nous avons : $CL_2 = (CC \blacktriangleright C_2) = 1$.

Finalement, $CL = CL_1 \vee CL_2 = CL_1 \vee 1 = 1$.

Il est donc inutile de déterminer la valeur de CL_1, CL est toujours vraie.

• Supposons que la condition à évaluer C soit la disjonction de deux expressions C_1 et C_2.

C_1 est fonction de la variable y, C_2 est fonction de t et u.

La Condition de Chemin est fonction des variable x, y et z.

Notons $CL_1 = (CC \blacktriangleright C_1)$, et $CL_2 = (CC \blacktriangleright C_2)$.

Nous avons vu dans le paragraphe précédent que $CL = (CC \blacktriangleright (C_1 \vee C_2))$ est égale à $CL_1 \wedge CL_2$.

Etant donné que les ensembles de définition de CC et C_2 sont disjoints, nous avons : $CL_2 = (CC \blacktriangleright C_2) = 1$.

Finalement, $CL = CL_1 \wedge CL_2 = CL_1 \wedge 1 = CL_1$.

La détermination de la valeur de la variable de choix se résume donc au calcul de $(CC_1 \blacktriangleright C)$.

Conclusion :

Il apparaît donc que :

- il n'est pas toujours nécessaire de mettre en œuvre une procédure de calcul de la variable de choix pour conclure que la Condition de Chemin n'implique pas de valeur de vérité pour la condition (i.e. le choix est libre),
- quand la valeur de la variable de choix libre n'est pas directement déductible de la répartition des variables dans les termes de la Condition de Chemin et de la condition à évaluer, le calcul peut se limiter à certains de ces termes.

Nous allons donc définir comment une condition peut être décomposée en expressions indépendantes, et déterminer quand et comment calculer la variable de choix.

8.3 - Décomposition d'une condition en termes

L'objet de paragraphe est de formaliser la méthode d'évaluation de la variable de choix, en tenant compte des propriétés qui permettent de simplifier ce calcul. Nous allons dans un premier temps poser les bases théoriques de la décomposition d'une expression booléenne en termes. Nous expliciterons alors comment nous définissons la notion de dépendance entre les variables présentes dans une expression.

8.3.1 - Définitions

On définit le graphe G = [X , U] tel que :

$X = \{x_1, \ldots, x_n\}$ est un ensemble. Un élément de X s'appelle un sommet.

U est un ensemble de couples non orientés de sommets. Un élément de U s'appelle une arête. Les sommets reliés par l'arête sont également appelés les extrémités de l'arête.

Une chaîne de longueur q est une séquence de q arêtes :

$$L = \{u_1, \ldots, u_q\}$$

telle que chaque arête u_r ($2 \leq r \leq q-1$) ait une extrémité commune avec u_{r-1} ($u_{r-1} \neq u_r$) et l'autre extrémité commune avec u_{r+1} ($u_{r+1} \neq u_r$).

L'extrémité i de u_1 non adjacente à u_2, et l'extrémité j de u_q non adjacente à u_{q-1} sont appelées les extrémités de la chaîne. On dit aussi que la chaîne L joint les sommets i et j.

La relation $\mathcal{R}$ telle que :

$$i \mathcal{R} j \Leftrightarrow \begin{cases} \text{soit } i=j \\ \text{soit il existe une chaîne joignant i et j} \end{cases}$$

est une relation d'équivalence (réflexive, symétrique, transitive).

Les classes d'équivalence induites sur X par cette relation forment une partition de X en $X_1, X_2, \ldots, X_p$.

Le nombre p de classes d'équivalences distinctes est appelé nombre de connexité du graphe.

Les graphes $G_1, G_2, \ldots, G_p$ engendrés par les sous-ensembles $X_1, X_2, \ldots, X_p$ sont appelés les composantes connexes du graphe G.

Si M est le nombre d'éléments de U, alors il existe un algorithme en O(M) qui détermine toutes les composantes connexes d'un graphe [GON85].

8.3.2 - Décomposition

La première étape est de choisir l'opérateur par rapport auquel nous définissons

la notion de terme.

En effet, nous pouvons indifféremment établir que les termes sont des conjonctions (de termes ou de variables) séparées par des opérateurs OU, ou des disjonctions (de termes ou de variables) séparées par des opérateurs ET. La construction des composantes connexes est identique quel que soit le choix effectué ; seul le mode d'utilisation des composantes connexes est différent (Cf. exemples §5.2).

Les cas les plus favorables de détermination de la variable de choix sont ceux pour lesquels nous savons a priori que CL égale 1. Cette situation correspond à deux cas :

- $(CC_1 \vee CC_2) \rightarrow C$ et $(CC_1 \rightarrow C = 1$ ou $CC_2 \rightarrow C = 1)$
- $CC \rightarrow (C_1 \wedge C_2)$ et $CC \rightarrow C_1 = 1$ ou $CC \rightarrow C_2 = 1$

La Condition de Chemin étant rarement présentée sous forme d'une disjonction d'expressions (Cf. procédures de construction et de mise à jour), nous choisissons de décomposer les expressions par rapport à l'opérateur ET.

Exemple :

Soit E l'expression booléenne :

$E = (x \vee y) \wedge (x \wedge z \vee \bar{t}) \wedge (u \vee v)$

E se décompose en trois termes T_1, T_2 et T_3 avec :

$T_1 = x \vee y$

$T_2 = x \wedge z \vee \bar{t}$

$T_3 = u \vee v$

8.3.3 - Graphe de dépendance d'une condition

Le graphe Gd de dépendance d'une condition est défini comme suit :

- l'ensemble des sommets est l'ensemble des variables de la condition

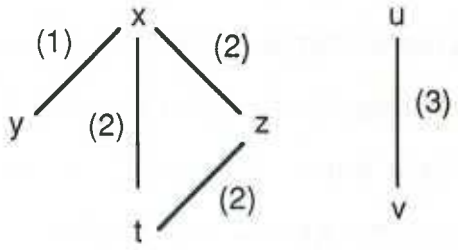
- il existe une arête entre toutes les paires de variables qui figurent dans un terme.

Exemple :

Le graphe de dépendance de la condition $E = (x \vee y) \wedge (x \wedge z \vee \bar{t}) \wedge (u \vee v)$, conjonction des trois termes tels que :

- le premier est fonction de x et y,
- le second est fonction de x, z et t,
- le troisième est fonction de u et v,

est le suivant :



Ce graphe possède deux composantes connexes.

Interprétation :

L'existence de plusieurs composantes connexes signifie que la condition est composée de plusieurs sous-ensembles de termes indépendants. De ce fait, l'introduction d'une contrainte sur des variables d'une composante connexe n'influe pas sur la valeur des éléments des autres composantes connexes du graphe de dépendance.

8.4 - Calcul de la variable de choix sur une composante connexe

Une fois les variables réparties dans les différents sous-ensembles, les sous-

ensembles de termes indépendants de la condition sont déterminés de la façon suivante :

Deux termes sont dépendants si leurs variables appartiennent à une même composante connexe, ils sont indépendants sinon.

Un sous-ensemble regroupe donc les termes dont les variables sont sommets d'une composante connexe.

Le nombre de sous-ensembles indépendants est alors celui de composantes connexes.

Remarque :

Pratiquement, pour déterminer si deux termes sont dépendants, il suffit de choisir au hasard une variable de chaque terme et de vérifier l'appartenance de ces deux variables à une même composante connexe. En effet, si une variable particulière d'un terme appartient à une composante connexe, alors toutes les variables de ce terme sont des sommets de cette composante connexe.

La mise en œuvre proprement dite du calcul de la valeur de la variable de choix s'opère de la façon suivante : il est nécessaire dans un premier temps de déterminer à quelle(s) composante(s) connexe(s) appartient la condition à évaluer, et quelle(s) composante(s) connexe(s) de la Condition de Chemin cette condition fait intervenir.

Si l'intersection entre ces deux ensembles de composantes connexes est vide, alors CL = 1. De même, si une au moins des composantes connexes de la condition à évaluer est totalement disjointe de l'ensemble des composantes connexes de la Condition de Chemin, alors CL = 1. Dans les autres cas, le calcul s'opère sur les composantes connexes de la Condition de Chemin concernées par la condition à évaluer.

8.5 - Optimisation

Nous avons vu que l'algorithme de recherche de composantes connexes était de complexité O(M), M étant le nombre d'arêtes entre les sommets du graphe.
Nous pouvons envisager une nouvelle forme du graphe de dépendance dont les sommets ne sont plus les variables mais les termes eux-mêmes. Le nombre de sommets est alors réduit.
La relation entre les sommets est inchangée : une arête relie deux sommets si et seulement si les deux termes ont au moins une variable en commun.

8.6 - Conclusion

Etudier la dépendance des termes d'une condition est une tâche de moindre coût, et qui casse la complexité de l'évaluation de la variable de choix.
La détermination de la variable de choix passe donc par deux étapes distinctes : calcul des composantes connexes de la Condition de Chemin, de la condition à évaluer (en effectuant une décomposition par rapport à l'opérateur logique ET), détermination des composantes connexes de la Condition de Chemin et de la condition à évaluer concernées par les calcul de choix, et enfin calcul sur ces ensembles de composantes connexes exclusivement.

9 - Conclusion

Nous avons défini l'opérateur de choix qui permet d'évaluer la liberté de choix

de la valeur d'une condition dans le contexte d'une Condition de Chemin donnée. Nous avons alors étudié comment évaluer le résultat de cette opération, puis les propriétés de distribution de l'opérateur de choix. Finalement, nous avons déterminé comment décomposer une expression complexe en expressions élémentaires, ce qui nous a permis de mettre en évidence des simplifications de la procédure d'évaluation de la variable de choix-libre (résultat de l'opération de choix).

Les expressions conditionnelles pourront maintenant être traitées :

- par une méthode d'évaluation de la variable de choix si les expressions mises en jeu sont élémentaires,
- grâce aux propriétés de distribution de l'opérateur de choix, et aux variables de choix associées aux expressions élémentaires qui composent les expressions complexes.

Il reste donc à détailler le traitement propre à chacune des catégories de conditions élémentaires.

Chapitre 3 - Gestion de conditions booléennes

Nous considérons ici le problème de la représentation, manipulation et calcul de variables de choix dans le cadre d'expressions logiques qui ne manipulent que des variables booléennes.

Après avoir précisé la forme des expressions manipulées, nous étudions un ensemble de méthodes de manipulations d'expressions logiques. Nous présentons alors l'approche que nous avons retenue, et terminons par la présentation des travaux réalisés.

1 - Forme des expressions

Nous nous intéressons ici au cas d'expressions logiques dont les éléments sont purement booléens.

Nous manipulons donc des conditions booléennes constituées de **constantes symboliques booléennes** et des **opérateurs logiques** classiques.

Toute expression conforme à la grammaire B.N.F. suivante est une condition valide pour la Gestion de la Condition de Chemin Booléenne (les éléments terminaux sont écrits en caractères gras) :

<condition-booléenne> ::= [<condition-booléenne> **OU**] <terme>

<terme> ::= [<terme> **ET**] [**NOT**] <facteur>

<facteur> ::= **constante-symbolique-booléenne**

| (<condition-booléenne>)

| **0**

| **1**

Notation :

Par la suite, l'opérateur de négation sera noté par une barre horizontale au dessus du terme complémenté, les opérateurs ET et OU logiques seront respectivement notés $\wedge$ et $\vee$.

2 - Calcul de la variable de choix sur le domaine booléen

Nous renvoyons à [PER67, LAG87, AUM88] pour les définitions de l'algèbre booléenne.

Nous étudions dans ce paragraphe comment les résultats du premier chapitre concernant le calcul de la variable de choix s'appliquent sur le domaine particulier de l'algèbre de Boole.

2.1 - Rappel : Détermination de la valeur de vérité d'une fonction booléenne à partir de la Condition de Chemin

Notons CC la fonction booléenne associée à la Condition de Chemin, et C la

fonction booléenne associée à la condition à évaluer.

Nous avons établi dans le Chapitre 2 que la variable de choix libre associée à la condition C est fausse si et seulement si ($S_{CC} \subset S_C$) ou ($S_{CC} \subset S_{\overline{C}}$), où S_{CC} est l'ensemble des solutions de CC, S_C l'ensemble des solutions de C, et $S_{\overline{C}}$ l'ensemble des solutions de $\overline{C}$.

Les constantes symboliques manipulées dans ces expressions étant de type booléen, comment utiliser cette propriété pour calculer la variable de choix ?

2.2 - Lien entre inclusion et implication

Intuitivement, l'opération logique à effectuer pour déterminer si le choix est libre est un calcul d'implication logique.

Etudions alors plus précisément le rapport que peut avoir l'inclusion des ensembles de solutions de CC et C, et le calcul d'implication logique entre CC et C :

Si E_{CC} est inclus dans E_C, que pouvons-nous déduire sur ($CC \Rightarrow C$) ?

L'ensemble $S_{\overline{CC} \vee C}$ de solutions de ($\overline{CC} \vee C$) est la réunion des ensembles de solutions de $\overline{CC}$ et de C, soit : $S_{\overline{CC} \vee C} = S_{\overline{CC}} \cup S_C$

Or, $S_{CC} \subset S_C$ et $S_{\overline{C}} \subset S_{\overline{CC}}$. On a alors :

$(S_{\overline{CC}} \cup S_C) \supset (S_{\overline{C}} \cup S_C) = B^n$

B^n est contenu par ($S_{\overline{CC}} \cup S_C$). On en déduit donc que $S_{\overline{CC} \vee C} = B^n$, ce qui signifie que S_{CC} inclus dans S_C implique ($CC \Rightarrow C$).

En conclusion, déterminer la valeur de vérité pour C connaissant CC revient à évaluer la validité de la formule ($CC \Rightarrow C$) (calcul d'implication logique).

Remarque :

Etant donné que notre problème est de déterminer la valeur de vérité de C, **sachant que CC est vraie**, il est plus correct d'utiliser la notation CC |= C. Plus précisément, évaluer CC |= C équivaut à vérifier que (CC ⇒ C) = 1.

2.3 - Détermination de la variable de choix

Dans le cas de fonctions booléennes, la détermination de la valeur de vérité de C, sachant que CC est vraie, passe par un calcul d'implication logique.

Le calcul de (CC ⇒ C) revient à l'évaluation de $\overline{CC} \vee C$.

En théorie, le résultat est une expression logique qui, une fois simplifiée, est égale soit à 1, soit à 0, ou reste une expression de variables booléennes.

Dans le détail, les cas qui peuvent se présenter sont les suivants :

- $\overline{CC} \vee C = 1$.

Alors (CC ⇒ C) est vrai.

- $\overline{CC} \vee C = 0$.

En théorie, cela signifie que : $\overline{CC} = 0$ et $C = 0$, soit $\overline{CC} = 0$ et $\overline{C} = 1$.

On obtient alors : $\overline{CC} \vee \overline{C} = 1$ d'où (CC ⇒ $\overline{C}$) est vrai.

En pratique, en appliquant simplement le calcul d'implication logique, nous n'obtiendrons ce résultat ($\overline{CC} \vee C = 0$) que dans le cas particulier où la Condition de Chemin CC se réduit à la constante booléenne 1 et où la condition à évaluer se réduit à la constante booléenne 0. Dans les autres cas, CC et/ou C étant des expressions de constantes symboliques, l'expression de $\overline{CC} \vee C$ ne peut pas être égale à 0 (OU logique de deux expressions ($\overline{CC}$, C) dont une au moins n'est pas la constante booléenne 0).

Conclure que (CC ⇒ $\overline{C}$) à partir de ce calcul est possible en interprétant le

résultat de $\overline{CC} \vee C$ **en tenant compte du fait que CC est vraie.**

- $\overline{CC} \vee C$ est une expression.

Si cette expression ne peut pas être simplifiée, alors CC ne détermine aucune valeur de vérité pour C. Dans ce cas, la variable de choix libre CL vaut 1.

La détermination de la valeur de la variable de choix consiste à calculer $\overline{CC} \vee C$. Si le résultat vaut 1, alors le choix n'est pas libre, et la condition est impliquée en vrai par la Condition de Chemin. Si ce n'est pas le cas, alors nous avons vu que le résultat de $\overline{CC} \vee C$ est une expression de constantes symboliques booléennes.

A ce stade, deux possibilités s'offrent alors pour déterminer si $CC \Rightarrow \overline{C}$ ou non.

La première solution consiste à interpréter l'expression en intégrant le fait que CC est vraie. Il faut alors effectuer le ET logique entre cette expression et CC, et comparer le résultat à la constante booléenne 0. Si $(\overline{CC} \vee C) \wedge CC$ vaut 0, alors $(CC \Rightarrow \overline{C})$ est vrai.

La deuxième solution consiste à effectuer directement le calcul de $(CC \Rightarrow \overline{C})$.

Dans les deux cas, si le résultat simplifié est une expression de constantes symboliques booléennes, alors la variable de choix libre est vraie.

Nous choisirons l'une ou l'autre des solutions en fonction de la complexité des opérations logiques (négation, ET, OU).

2.4 - Conclusion

Dans le cas d'expressions de constantes symboliques booléennes, la détermination de la variable de choix est mise en œuvre par le calcul

d'implication logique de la condition à évaluer par la Condition de Chemin. Ce résultat établi, nous pouvons maintenant décrire le processus complet de Gestion d'une Condition de Chemin booléenne.

3 - Gestion d'une Condition de Chemin booléenne

Nous rappelons que la gestion de la Condition de Chemin consiste, dans un premier temps, à construire la Condition de Chemin à partir d'une spécification, puis à calculer l'implication, et enfin à prévoir les mécanismes de mise à jour.

3.1 - Rappel : Extraction de la Condition de Chemin d'une spécification

Nous avons défini dans le Chapitre 1 (§3) comment la Condition de Chemin est extraite d'une éventuelle spécification : après extraction des conditions spécifiées valides pour la date de simulation traitée, il est nécessaire de remplacer dans chaque condition tous les signaux et variables par leur expression numérique ou symbolique ; les expressions obtenues viennent enrichir la Condition de Chemin existante.

La première étape consiste à extraire de la Condition de Chemin Spécifiée toutes les conditions dont la qualification temporelle commence avant ou à la date de simulation. Les conditions étant exprimées en fonction de variables ou signaux, l'expression de ces éléments en terme de constantes symboliques est effectuée. La Condition de Chemin est alors la conjonction de toutes ces

conditions et de la Condition de Chemin courante.

La construction de la Condition de Chemin nécessite donc de savoir :
- construire l'expression en fonction des constantes symboliques d'une expression fonction de variables et signaux du programme VHDL,
- représenter des fonctions logiques,
- effectuer le ET logique entre deux fonctions.

3.2 - Calcul de la variable de choix libre

Lorsque la Condition de Chemin est construite, et que la condition à évaluer est exprimée en fonction des constantes symboliques, il faut alors évaluer la variable de choix.
Dans le cas d'expressions booléennes, le calcul de la valeur de la variable de choix libre se ramène au calcul d'implication logique, à savoir l'évaluation de l'expression $(\overline{CC} \vee C)$. En réalité, conformément aux résultats présentés dans le Chapitre 2, il suffit d'évaluer cette implication sur la conjonction des composantes connexes de la Condition de Chemin concernées par la condition C.

Le calcul d'implication logique entre deux fonctions nécessite alors de savoir :
- représenter des fonctions logiques,
- effectuer la négation d'une fonction,
- effectuer le OU logique entre deux fonctions,
- interpréter le résultat.

3.3 - Mise à jour

Si la variable de choix est vraie, alors la procédure de mise à jour de la Condition de Chemin s'effectue en interrogeant l'utilisateur sur la valeur de vérité ("vrai" ou "faux") du test à évaluer. Quand l'utilisateur a fait son choix, la nouvelle Condition de Chemin est la conjonction de la Condition de Chemin courante et de l'expression de la condition (si la réponse de l'utilisateur est "vrai") ou de la négation de l'expression de la condition (si la réponse de l'utilisateur est "faux").
La mise à jour nécessite donc de savoir :
- représenter des fonctions logiques,
- effectuer la négation de fonctions logiques,
- effectuer le ET logique entre deux fonctions,
- recalculer le graphe de dépendance de la Condition de Chemin.

En résumé, le processus de Gestion de la Condition de Chemin dans le cadre d'expressions booléennes requiert de savoir :
- traduire une expression de variables et signaux dans sa forme fonction de constantes symboliques,
- représenter une expression logique,
- effectuer les opérations logiques de base sur cette représentation.

L'extraction de l'expression en fonction des constantes symboliques est une opération de manipulation d'expressions symboliques sur le modèle interne qui représente la description, tâche qui intervient très fréquemment dans l'utilisation du simulateur symbolique [ROG92a]. Il nous reste à choisir un mode de représentation des fonctions logiques et à décrire la mise en œuvre des

opérations de base.

Nous allons donc détailler dans ce qui suit les différents modes de représentation de fonctions booléennes, ainsi que les méthodes de manipulation associées.

3.4 - Rappels : Méthodes de représentation et de minimisation

Nous pouvons distinguer deux classes de méthodes de représentation de fonctions logiques : celles qui reposent sur la donnée de la valeur de toutes les combinaisons possibles de la fonction, et celles qui manipulent une forme qui permet de déduire la valeur de vérité pour une combinaison donnée. Les deux méthodes classiques représentatives de ces deux classes sont la table de vérité et la représentation algébrique.

La représentation d'une fonction logique sous forme de table de vérité est l'expression, dans une table, de la valeur de la fonction pour toutes les combinaisons possibles des variables de la fonction [PER67, LAG76, AUM88]. De ce fait, pour la représentation d'une fonction à n variables, la table contient 2^n lignes et (n+1) colonnes (une pour chaque variable et une pour la fonction). L'expression des valeurs de la fonction étant exhaustive, l'application d'une opération logique sur une ligne de la table est élémentaire. Il faut par contre effectuer l'opération sur chacune des lignes. L'examen des 2^n éléments est également nécessaire pour déterminer si la fonction vaut la fonction constante 1 (ou 0).

En opposition à l'expression exhaustive de tous les points d'une fonction logique par table de vérité, la forme algébrique est une expression constituée de OU et de ET logiques de variables de commutation, complémentées ou non, ou des éléments neutres des deux lois [LAG76].
Parmi ces formes algébriques, on distingue les deux formes canoniques (uniques) qui représentent tous les points vrais de la fonction, les bases premières, la base première complète et les bases minimales.
Alors que les bases canoniques sont une simple retranscription algébrique de la table de vérité, les autres bases constituent une expression compacte de la fonction, et qui doit être interprétée pour retrouver la table de vérité.
L'intérêt de ces formes algébriques réside dans leur compacité et leur lisibilité. Il est alors très souvent utile de les simplifier.
Une phase de minimisation est également indispensable si le mode de calcul d'implication retenu consiste à vérifier que $(\overline{CC} \vee C)$ vaut 1.

Nous étudions dans ce qui suit les différentes méthodes classiques de simplification de fonctions logiques.

La première classe de méthodes de simplification d'expressions logiques s'applique directement sur la forme algébrique et utilise principalement les propriétés d'absorption, de simplification, et les théorèmes du consensus [PER67, LAG76, AUM88].
De manière générale, on peut considérer que l'obtention d'un résultat satisfaisant en utilisant ces méthodes dépend beaucoup du savoir-faire de l'utilisateur ; de ce fait, ces raisonnements sont difficilement automatisables.

Toutefois, Tison [TIS67] propose un algorithme d'obtention de la base première

complète basé sur l'utilisation itérative (une itération par variable) et exclusive du théorème du consensus et de la propriété d'absorption. Cette méthode s'applique sur une somme de produits. La forme obtenue est la base première complète de la fonction. Cette expression est unique, mais n'est pas minimale. Cette méthode permet toutefois d'obtenir la valeur "1" si la fonction est toujours vraie.

Cet algorithme est de complexité exponentielle par rapport au nombre de variables.

Les autres méthodes classiques de simplification de fonctions booléennes sont tabulaires et sont principalement basées sur l'utilisation de la propriété d'absorption ; leur mise en œuvre est théoriquement aisée puisque suivant une démarche systématique.

Les plus célèbres de ces méthodes sont celles de Karnaugh et de Quine-McCluskey.

La méthode de Karnaugh est une méthode tabulaire manuelle de simplification de fonctions logiques (pour 6 variables au plus) [PER67, LAG87, AUM88].

Une case de la table représente un terme canonique de la fonction. Deux cases adjacentes diffèrent par une seule variable.

Si n est le nombre de variables, le principe de simplification est de combiner 2^p cases adjacentes ($1 \leq p \leq n$). Cette méthode donne une base minimale.

La méthode de Quine-McCluskey est une méthode algorithmique de simplification qui s'applique à la première forme canonique de la fonction. Son principe est le suivant [McC56, PER67] :

- former l'ensemble des implicants premiers de la fonction,
- utiliser une table de choix pour obtenir une forme minimale.

Pratiquement, il est tout d'abord nécessaire de constituer des classes. La classe C_n regroupe les implicants de la fonctions pour lesquels n variables sont égales à 1.

On compare alors entre eux tous les termes des classes C_i et C_{i+1} en appliquant la relation "$x \wedge y \vee x \wedge \bar{y} = x$." Ce processus est itéré sur les nouvelles classes ainsi créées. Les termes qui à un moment donné ne peuvent plus être regroupés avec les autres sont les implicants premiers de la fonctions.

Une table de choix est alors utilisée pour déterminer les implicants premiers essentiels de la fonction ; les sommets de la fonction non couverts par les implicants premiers essentiels sont couverts par des implicants judicieusement sélectionnés dans la table. Une base minimale est finalement obtenue.

Cette méthode ne s'applique que sur la première forme canonique de la fonction à minimiser. Son principe est de générer dans un premier temps tous les implicants premiers, et de sélectionner ensuite une couverture minimale. Pour une fonction de n variables, le nombre d'implicants premiers est dans le pire des cas de l'ordre de $\frac{3^n}{n}$ et la recherche d'une couverture minimale appartient à l'ensemble des problèmes NP-complets [BRA84].

Il n'est donc pas envisageable d'utiliser cet algorithme pour minimiser une fonction de plus d'une dizaine de variables. De plus, la donnée en entrée de la première forme canonique est une contrainte très lourde qui peut supposer un traitement important de l'expression de la fonction (passage par la table de vérité par exemple) avant de pouvoir exécuter l'algorithme.

En conclusion, les méthodes classiques de minimisation de fonctions logiques sont contraignantes pour la mise en œuvre (forme de la fonction en entrée), et

de complexité telle qu'elles ne peuvent pas être utilisées pour des fonctions de plus de dix variables.

4 - Méthodes de minimisation pour la synthèse de VLSI

Nous avons vu que les méthodes classiques de minimisation de fonctions logiques se limitaient à une dizaine de variables au maximum.
Des tâches telles que la synthèse de VLSI ne peuvent plus se contenter de ces outils, inopérants pour les quantités de données manipulées (plusieurs dizaines de variables d'entrée). Depuis les années 70, de nouvelles techniques de minimisation sont développées. Leur objectif est de pouvoir traiter un nombre beaucoup plus important de variables. Ces méthodes peuvent être distinguées selon deux grandes classes : d'une part les algorithmes dont l'objectif est de trouver la base minimale la meilleure selon les critères de nombre d'implicants et de littéraux mis en jeu ; d'autre part, les méthodes de recherche d'une couverture proche de l'optimale (mais pas forcément la meilleure) basées sur des heuristiques de choix des éléments de cette couverture.

Remarque :
Notons également de gros efforts dans la minimisation de fonctions logiques à sorties multiples. Le principe est ici de simplifier les différentes fonctions au maximum, tout en préservant les parties "communes" aux fonctions. Ce type de minimisation permet le traitement de fonctions multiples manipulant une centaine d'entrées et autant de sorties ; ces résultats ne présentent toutefois aucun intérêt pour notre application.

4.1 - Algorithmes de minimisation de fonctions booléennes avec génération de tous les implicants premiers

Une première classe de travaux pour l'obtention de la base minimale optimale (en nombre d'implicants et de littéraux) ont été menés en s'inspirant de la méthode de Quine-McCluskey [McC56].
En effet, la première étape de tous ces algorithmes est la génération de tous les implicants premiers de la fonction à minimiser.
La sélection de la couverture minimale à partir de la base première complète ainsi obtenue fait toujours l'objet de nombreux travaux. On peut cependant déceler deux tendances : les méthodes de réduction de cubes [BRA84, DAG86, RUD86] et les méthodes d'inclusion de fonctions [TIS67, CUT87, HON91].
Les derniers résultats disponibles [HON91] indiquent que par le biais des plus évolués de ces systèmes, des fonctions d'une trentaine de variables peuvent être minimisées dans des temps d'exécution raisonnables.

4.2 - Algorithmes de minimisation de fonctions booléennes sans génération de tous les implicants premiers

Une autre tendance pour la minimisation dite exacte des fonctions booléennes est basée sur la recherche de la couverture optimale sans génération de l'ensemble des implicants premiers. Le principe est la recherche, dans la forme initiale (donnée en entrée) de la fonction, des implicants premiers essentiels. La plupart des systèmes imposent la donnée de la fonction sous forme de somme de produits. Le plus célèbre de ces algorithmes est CAMP [BIS84, BIS86], suivi de CAMPII [BIS90]. Ces systèmes peuvent gérer des fonctions d'une trentaine

de variables.

4.3 - Algorithmes de minimisation de fonctions booléennes basés sur l'utilisation d'heuristiques

La deuxième classe d'algorithmes de minimisation de fonctions logiques recherche une couverture "minimisée", même si ce n'est pas la meilleure. Ces systèmes utilisent des heuristiques de choix des implicants premiers qui forment la base. La lignée la plus célèbre est issue de chez IBM avec MINI (1974) [HON74, BRA84], suivi de PRESTO (1981) [BRA84], puis d'ESPRESSO [BRAY84]. S'inspirant de tous ses prédécesseurs, ESPRESSO-II a vu le jour dans les années 1982-84 [BRA84].
Les performances de ces systèmes sont comparables à celles des algorithmes de minimisation exacte en terme de nombre de variables manipulées ; la détermination de la solution est globalement plus rapide.

4.4 - Synthèse

Les systèmes de minimisation de fonctions logiques développés pour répondre à la demande des applications telles que la synthèse de VLSI sont de deux types. Une première classe génère une base minimale en nombre de termes et de littéraux manipulés ; la deuxième catégorie recherche une couverture optimisée, mais pas forcément minimale.
Ces méthodes permettent de traiter des fonctions d'une trentaine de variables au maximum.

Nous pouvons envisager d'utiliser les algorithmes de minimisation "exacte" pour le calcul d'implication. En effet, ils permettent de décider si la fonction logique $(\overline{CC} \vee C)$ vaut 1 ou non. Par contre, les méthodes basées sur l'utilisation d'heuristiques ne peuvent pas s'appliquer car elles ne donnent pas forcément une forme minimale.

5 - Gestion de la Condition de Chemin par les Diagrammes de Décision Binaire (DDB)

Les diagrammes de décision binaire constituent une méthode de représentation et de manipulation de fonctions logiques. Cette approche, basée sur l'application de l'expansion de Shannon, a été définie par LEE [LEE59], précisée par AKERS [AKE77, AKE78a], puis reprise par BRYANT [BRY86].
Après une brève présentation de la méthode et de son utilisation, nous étudions l'apport des diagrammes de décision binaire pour la gestion de la Condition de Chemin.

5.1 - Rappel : Représentation d'une fonction logique par un arbre de décision binaire

5.1.1 - Définitions

Soit $f(x_1, \ldots, x_n)$ une fonction booléenne des variables $x_1, \ldots, x_n$.
On appelle **restriction** de la fonction f en x_i la fonction :

$$f|_{xi=k}(x_1,x_2, \ldots,x_n) = f(x_1, \ldots,x_{i-1},k,x_{i+1}, \ldots,x_n)$$

La **formule d'expansion de Shannon** appliquée à la fonction f pour la variable x_i est :

$$f(x_1,x_2, \ldots,x_n) = x_i \wedge f|_{xi=1}(x_1,x_2, \ldots,x_n) \vee \overline{x_i} \wedge f|_{xi=0}(x_1,x_2, \ldots,x_n)$$

Remarque :

La formule d'expansion ci-dessus est exprimée en fonction des opérateurs logiques (ET,OU). Elle peut également s'écrire en fonction des opérateurs (OU,ET), des opérateurs (ET,OU-EXCLUSIF), ... :

• Formule d'expansion en (OU,ET) :

$$f(x_1,x_2, \ldots,x_n) = (\overline{x_i} \vee f|_{xi=1}(x_1,x_2, \ldots,x_n)) \wedge (x_i \vee f|_{xi=0}(x_1,x_2, \ldots,x_n))$$

• Formule d'expansion en (ET,OU-EXCLUSIF) :

$$f(x_1,x_2, \ldots,x_n) = x_i \wedge f|_{xi=1}(x_1,x_2, \ldots,x_n) \oplus \overline{x_i} \wedge f|_{xi=0}(x_1,x_2, \ldots,x_n)$$

Si la formule d'expansion de Shannon est appliquée à f pour x_1, puis aux deux fonctions obtenues pour x_2, et ainsi de suite jusqu'à x_n, on obtient un arbre appelé **arbre de décision binaire**.

Exemple :

$f(a,b) = a \vee b$

$f(a,b) = a \wedge F0 \vee \overline{a} \wedge F1$

avec F0 = 1 F1 = b

$F0 = b \wedge F2 \vee \overline{b} \wedge F3$

avec F2 = 1 F3 = 1

$F1 = b \wedge F4 \vee \overline{b} \wedge F5$

avec F4 = 1 F5 = 0

La représentation arborescente est la suivante : le sous-arbre gauche représente la valeur de la fonction si la variable portée par le nœud est fausse, le sous-arbre droit représente la valeur de la fonction si la variable est vraie.
L'arbre associé à f est représenté dans la Figure 4(a).

5.1.2 - Simplification d'un arbre de décision binaire

Si les sous-arbres gauches et droits d'un nœud sont identiques, alors la valeur de la fonction est indépendante de la variable représentée par ce nœud. Ce nœud peut donc être retiré de l'arbre, et substitué par son sous-arbre (droit ou gauche).
L'arbre réduit de la fonction f(a,b) = a ∨ b est représenté dans la Figure 4(b).

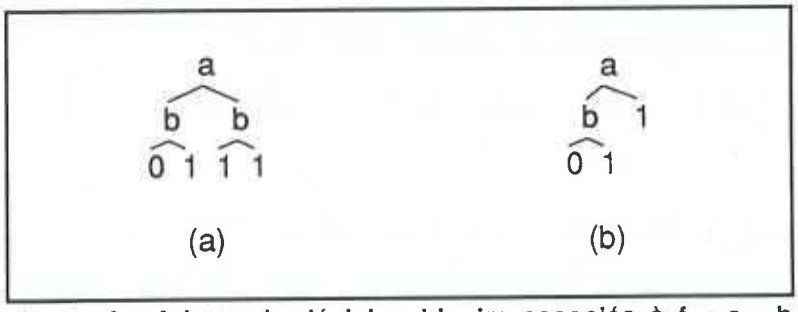


Figure 4 - Arbres de décision binaire associés à f = a ∨ b

Exemple :

f(a,b,c) = $a \wedge b \vee \bar{b} \wedge c \vee \bar{a} \wedge b$

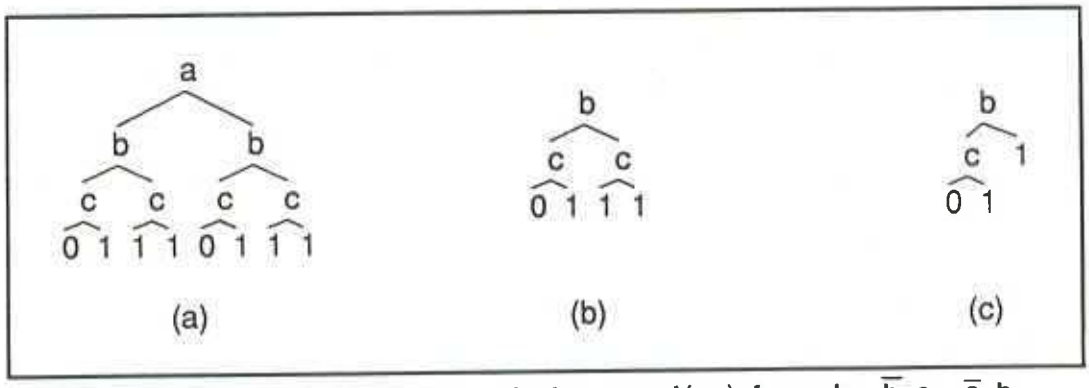


Figure 5 - Arbres de décision binaire associés à f = $a \wedge b \vee \bar{b} \wedge c \vee \bar{a} \wedge b$

L'arbre de décision binaire associé à f représenté par la Figure 5(a).
Nous constatons dans un premier temps que les deux sous-arbres de la racine sont identiques. L'arbre réduit est représenté par la Figure 5(b).
Finalement, en réduisant la branche droite de b, nous obtenons l'arbre réduit de la Figure 5(c).

5.2 - Manipulation des arbres de décision binaire

Nous avons vu comment appliquer la formule d'expansion de Shannon à une fonction logique pour obtenir l'arbre de décision binaire réduit associé.
Or, cette opération d'expansion nécessite à chaque étape des évaluations et manipulations symboliques difficiles à réaliser.
Par contre, nous pouvons retenir une méthode de représentation des fonctions booléennes - les arbres de décision binaire - possédant une puissance d'expression intéressante, et d'interprétation aisée.
Nous avons choisi de représenter les fonctions booléennes par des arbres de décision binaire. Pour cela, il a été nécessaire d'utiliser une méthode de construction qui n'est pas basée sur l'expansion de Shannon, mais sur la mise en œuvre des opérations logiques de base sur des arbres élémentaires.
C'est pourquoi nous détaillons le comportement des opérations logiques sur cette méthode de représentation, ainsi que la construction d'un arbre élémentaire associé à une simple variable ou constante symbolique. A partir de ces règles, nous pouvons construire l'arbre de décision binaire associé à une fonction logique, et ce sans passer par le calcul de l'expansion.

Détaillons tout d'abord le comportement des opérations logiques sur les arbres

de décision binaire.

5.2.1 - Négation d'une fonction logique

L'arbre négation d'une fonction logique est obtenu en complémentant les feuilles de l'arbre qui représente cette fonction.

Exemple :

L'arbre de décision binaire négation de la fonction f(a,b) = a ∨ b est représenté dans la Figure 6.

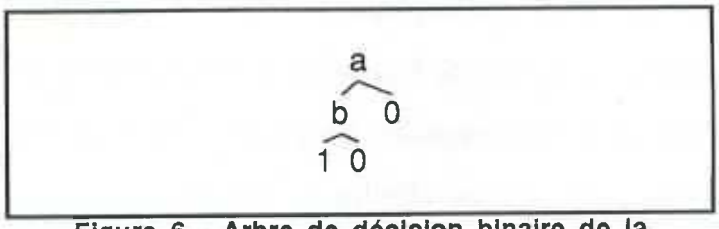


Figure 6 - Arbre de décision binaire de la fonction négation de f = a ∨ b

Cet arbre est obtenu en complémentant les feuilles de l'arbre qui représente f(a,b) = a ∨ b, et on peut vérifier qu'il représente bien la fonction :

$\bar{f}(a,b) = \overline{a \vee b} = \bar{a} \wedge \bar{b}$

5.2.2 - Opérations logiques binaires

Les opérations logiques entre deux arbres de décision binaire s'appliquent sur des arbres représentant des fonctions exprimées suivant le même ordonnancement des variables. L'arbre résultat de l'opération est systématiquement simplifié.

* OU logique entre deux arbres de décision binaire

- L'opération OU entre une feuille de valeur 1 et un arbre quelconque vaut 1.

- L'opération OU entre une feuille de valeur 0 et un arbre A est l'arbre A.
- L'opération OU entre deux arbres dont la racine porte une même variable x est l'arbre tel que :

la racine porte la variable x,

le sous-arbre gauche résultat est le OU des deux sous-arbres gauches des deux arbres,

le sous-arbre droit résultat est le OU des deux sous-arbres droits des deux arbres.

- L'opération OU entre deux arbres dont les racines ne portent pas les mêmes variables (x et y) est l'arbre tel que :

la racine porte la variable la plus "petite" (x) selon l'ordonnancement des variables,

le sous-arbre gauche est l'arbre résultat du OU entre le sous-arbre gauche de l'arbre de racine x et de l'arbre de racine y,

le sous-arbre droit est l'arbre résultat du OU entre le sous-arbre droit de l'arbre de racine x et de l'arbre de racine y.

L'algorithme est donné dans l'Annexe 1.

Exemple :

$f_1(a,b,c) = a \wedge b \vee \overline{b} \wedge c \vee \overline{a} \wedge b$

$f_2(a,b,c) = a \wedge c \vee b \wedge \overline{c}$

L'arbre de décision binaire de f_1 est représenté par la Figure 7(a), sa forme réduite par la Figure 7(b). De même, f_2 est représentée par l'arbre de la Figure 7(c), sa forme réduite par la Figure 7(d). Finalement, l'arbre de décision binaire résultat du OU logique entre f_1 et f_2 est représenté par la Figure 7(e) ; il est sous sa forme simplifiée dans la Figure 7(f).

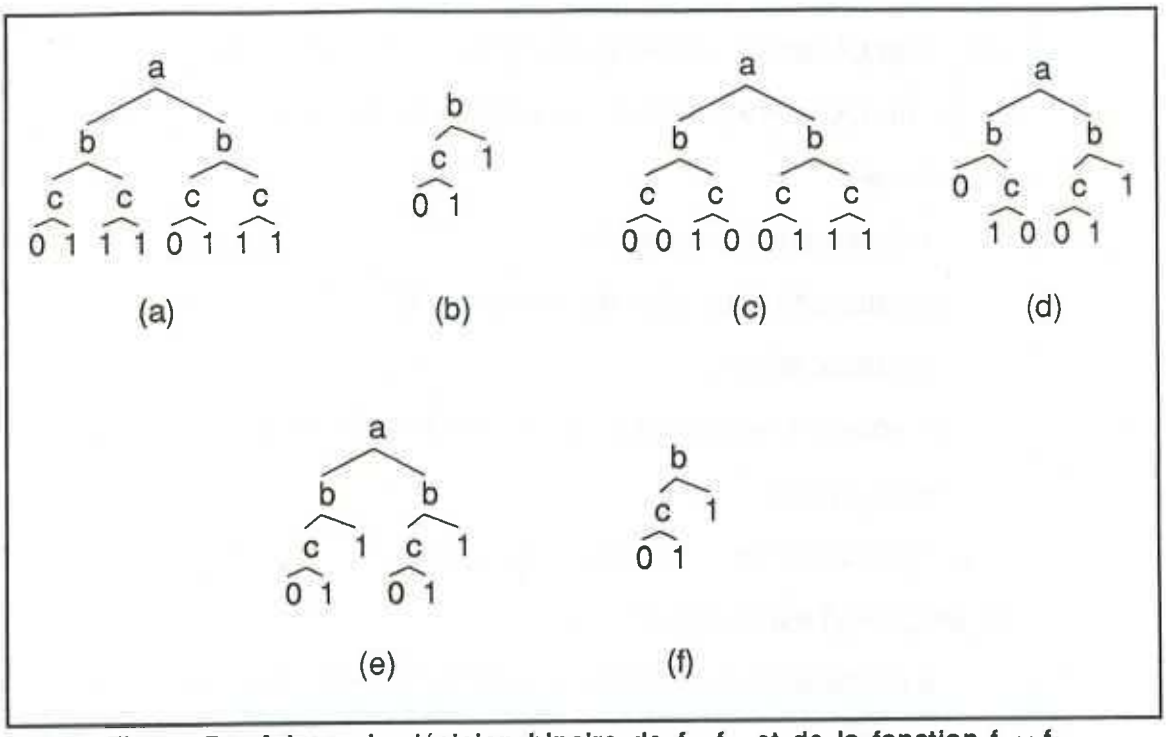


Figure 7 - Arbres de décision binaire de f_1, f_2, et de la fonction $f_1 \vee f_2$

* ET logique entre deux arbres de décision binaire

- L'opération ET entre une feuille de valeur 0 et un arbre quelconque vaut 0.

- L'opération ET entre une feuille de valeur 1 et un arbre A est l'arbre A.

- L'opération ET entre deux arbres dont la racine porte une même variable x est l'arbre tel que :

la racine est la variable x,

le sous-arbre gauche résultat est le ET des deux sous-arbres gauches des deux arbres,

le sous-arbre droit résultat est le ET des deux sous-arbres droits des deux arbres.

- L'opération ET entre deux arbres dont les racines ne portent pas les mêmes variables (x et y) est l'arbre tel que :

la racine porte la variable la plus "petite" (x) selon l'ordonnancement des variables,

le sous-arbre gauche est l'arbre résultat du ET entre le sous-arbre gauche de l'arbre de racine x et de l'arbre de racine y,

le sous-arbre droit est l'arbre résultat du ET entre le sous-arbre droit de l'arbre de racine x et de l'arbre de racine y.

L'algorithme est donné dans l'Annexe 1.

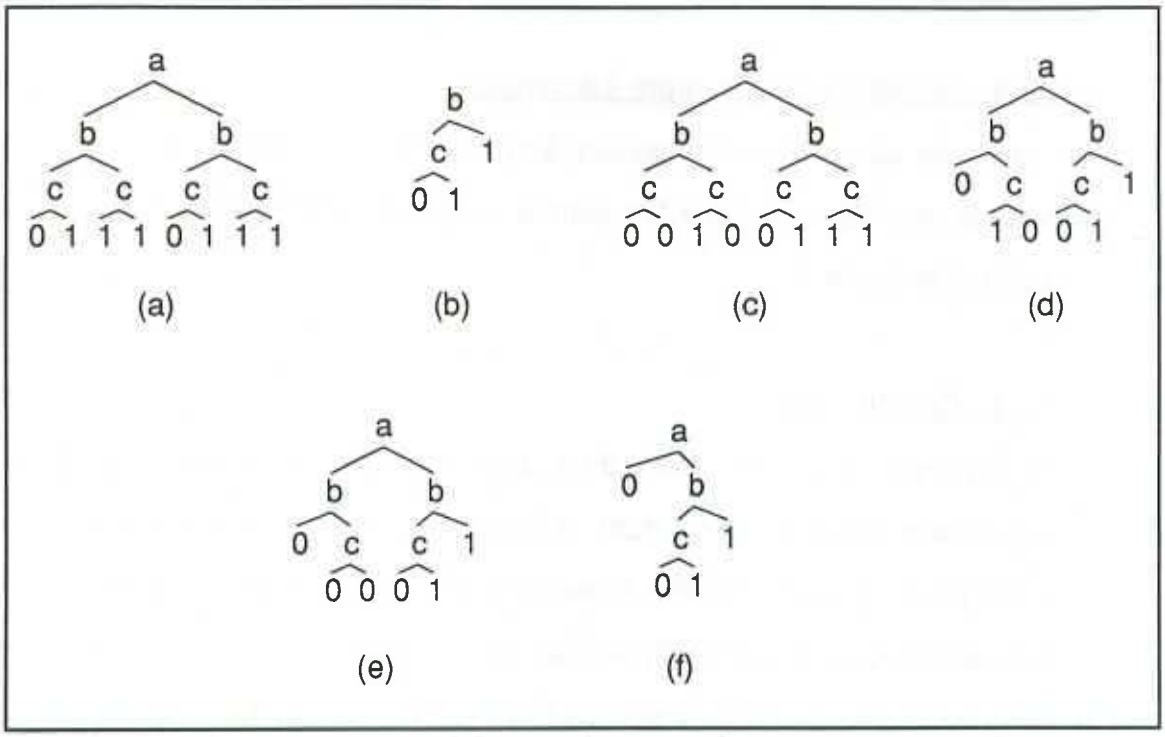


Figure 8 - Arbres de décision binaire de f_1, f_2, et de la fonction $f_1 \wedge f_2$

Exemple :

$f_1(a,b,c) = a \wedge b \vee \bar{b} \wedge c \vee \bar{a} \wedge b$

$f_2(a,b,c) = a \wedge c \vee b \wedge \bar{c}$

L'arbre de décision binaire de f_1 est représenté par la Figure 8(a), sa forme réduite par la Figure 8(b). De même, f_2 est représenté par l'arbre de la Figure 8(c), sa forme réduite par la Figure 8(d). Finalement, l'arbre de décision binaire résultat du OU logique entre f_1 et f_2 est représenté par la Figure 8(e) ; il est simplifié dans la Figure 8(f).

Les opérations logiques sur les arbres de décision binaire étant définies, nous pouvons maintenant définir comment construire l'arbre associé à une fonction logique à partir de son expression textuelle.

Pour cela, nous avons besoin de définir la construction d'un arbre élémentaire associé à une simple variable.

5.2.3 - Construction d'un arbre élémentaire

L'arbre de décision binaire associé à la fonction f(x) = x est l'arbre dont la racine porte le nom de la variable "x", dont le fils gauche est la feuille "0", et dont le fils droit est la feuille "1".

5.2.4 - Construction

Le principe de construction d'un arbre de décision binaire à partir d'une expression textuelle réside dans la reconnaissance d'une expression conforme à la syntaxe d'une condition booléenne (Cf §1), où une constante symbolique booléenne est une chaîne de caractères.

Quand une constante symbolique booléenne est identifiée, l'arbre élémentaire associé est construit.

Les opérations OU, ET, NEGATION sont effectuées quand les éléments sur lesquels elles s'appliquent sont construits.

Exemple :

$f(a,b,c) = a \vee \overline{b} \wedge c$

L'analyse de cette expression fait apparaître deux termes séparés par l'opérateur OU. Le premier terme étant une simple constante symbolique, l'arbre (Figure 9(a)) de décision binaire qui lui est associé est directement construit.

Le second terme ($\bar{b} \wedge c$) est composé de deux facteurs séparés par l'opérateur ET. Le premier facteur est la négation de la constante symbolique b, le second est constitué de la constante symbolique c. L'arbre qui représente la constante symbolique b est représenté par la Figure 9(b), l'arbre qui représente le premier facteur (négation de b) est l'arbre de la Figure 9(c). L'arbre qui représente le second facteur (c) est l'arbre de la Figure 9(d). L'arbre qui représente le second terme est le résultat du ET des Figures 9(c) et 9(d). C'est l'arbre de la Figure 7(e). La fonction f est le résultat du OU des deux termes (Figure 9(f)).

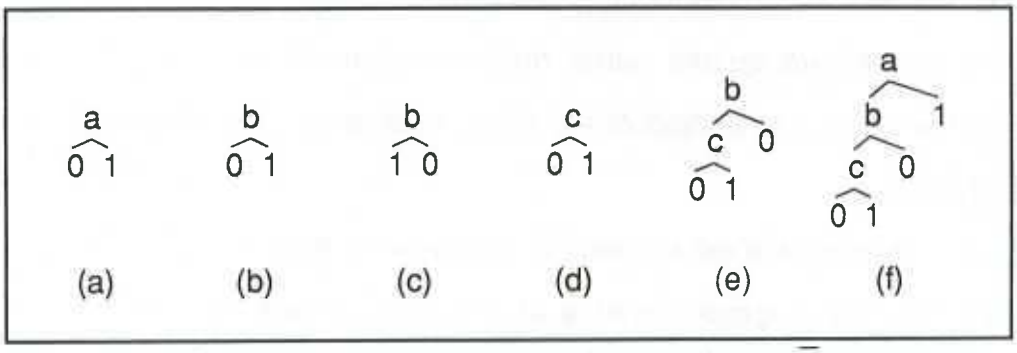


Figure 9 - Construction de la fonction f = a ∨ $\bar{b}$∧c

Nous pouvons remarquer que les arbres manipulés sont toujours sous forme simplifiée. Ce mode de construction évite donc la production de la représentation "exhaustive" de l'arbre, où toutes les combinaisons possibles des variables sont représentées par les 2^n feuilles de la fonction de n variables.

Les arbres de décision binaire constituent une méthode de représentation des fonctions logiques. Nous avons décrit une méthode de construction basée exclusivement sur l'application des opérations logiques sur les arbres élémentaires qui représentent les constantes symboliques. Par ce biais, nous avons donc éliminé la difficulté liée à la construction par application de l'expansion de Shannon.

Nous présentons maintenant une première amélioration de cette représentation, les diagrammes de décision binaire, et étudions ensuite la taille de ce mode de représentation, ainsi que la complexité des opérations.

5.3 - Diagrammes de décision binaire

Nous remarquons qu'un arbre de décision binaire peut posséder des feuilles, voire des sous-arbres identiques. La représentation peut alors être simplifiée, en ne conservant qu'une seule représentation de la feuille ou du sous-arbre concerné, et en remplaçant les autres feuilles ou sous-arbres par un arc vers le premier.

La représentation est strictement équivalente, mais moins encombrante. L'arbre est devenu un graphe orienté et acyclique, appelé **Diagramme de Décision Binaire** (DDB).

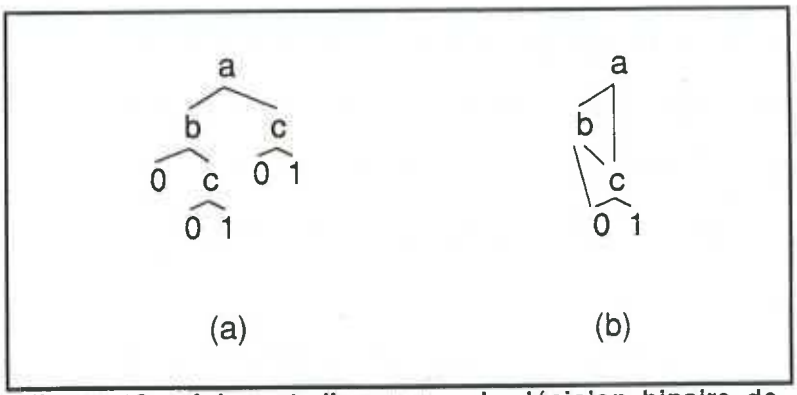


Figure 10 - Arbre et diagramme de décision binaire de la fonction f = a∧c ∨ $\bar{a}$∧b∧c

Exemple :

f(a,b,c) = a∧c ∨ $\bar{a}$∧b∧c

L'arbre de décision binaire associé à cette fonction est représenté par la Figure

10(a). Le diagramme de décision binaire correspondant est représenté par la Figure 10(b).

En cherchant et remplaçant tous les sous-arbres équivalents d'un diagramme de décision binaire, on obtient le diagramme de décision binaire réduit, minimal pour l'ordonnancement $x_1, \ldots, x_n$ des variables.

Remarque :

Pour deux ordonnancements différents des variables $x_1, \ldots, x_n$, les diagrammes de décision binaire réduits peuvent être de tailles très différentes [BRY86, MOR82].

Les opérations définies sur les arbres de décision binaire restent valides sur les diagrammes de décision binaire. Par exemple, sur un diagramme donné, on peut appliquer l'opération de négation. De même, sur deux diagrammes de décision binaire, on peut effectuer les opérations logiques (ET, OU, OU-EXCLUSIF, ...) et obtenir le diagramme résultat.

Remarque importante :

La représentation de fonctions logiques sous forme d'arbres de décision binaire, ou de diagrammes de décision binaire est dite "canonique". Il ne faut pas comprendre dans ce terme que cette représentation est une expression de la base canonique de la fonction, mais que, pour un ordonnancement de variables donné, les arbres (diagrammes) associés à deux fonctions booléennes équivalentes sont strictement identiques. **Pour un ordonnancement de variables donné, la représentation** sous forme d'arbre de décision binaire (diagramme de décision binaire) **d'une fonction, est unique.**

Par contre, l'écriture sous forme algébrique des termes contenus par un DDB associé à une fonction, donne une base qui n'est ni la base première complète, et qui n'est pas non plus minimale.

5.4 - Classes d'applications

Les diagrammes de décision binaire sont utilisés dans diverses applications.
La motivation d'AKERS [AKE77, AKE78a] pour le développement du concept de diagrammes de décision binaire était leur facilité d'utilisation pour le test fonctionnel [AKE78b, CHO89].
Parmi les applications implantées par la suite, nous pouvons noter plusieurs systèmes de vérification [ODA86, MAD88, COU90, COU91] ; leur principe est de construire le DDB correspondant à une description structurelle de circuit donnée et de comparer son adéquation avec le DDB associé à une "spécification" du circuit (description comportementale).
Signalons également l'utilisation des diagrammes de décision binaire (pour leur aptitude à simplifier des fonctions logiques de plusieurs dizaines de variables) dans des applications pour la synthèse de VLSI [MAL88, MAT89].

5.5 - Gestion de la Condition de Chemin Booléenne par les DDB

Gérer la Condition de Chemin Booléenne signifie construire la Condition de Chemin, calculer la variable de choix libre associée à une condition, et enfin mettre la Condition de Chemin à jour.

La construction de la Condition de Chemin consiste à construire dans un premier temps les diagrammes de décision binaire associés à chacune des conditions extraites de la spécification formelle, puis à effectuer le ET logique entre les diagrammes appartenant à une même composante connexe. La Condition de Chemin est alors un ensemble de diagrammes de décision binaire indépendants.

Cette construction itérative de la Condition de Chemin à travers le temps se justifie par le fait que les éléments qui la constituent ne sont pas des variables, mais des constantes symboliques. Une condition (portant sur des constantes symboliques) vraie à une date t, sera vérifiée pour toute date ultérieure à t.

A la rencontre d'une condition C à évaluer, le Module de Gestion de la Condition de Chemin Booléenne effectue les opérations suivantes :
- Construction du DDB conjonction de tous les DDBs associés aux composantes connexes de la Condition de Chemin. Notons le CC.
- Construction du DDB représentant $(\overline{CC} \vee C)$ et vérifier s'il se réduit à "1". Si c'est le cas, alors CC implique C, et la variable CL de choix libre vaut 0.
- Construction du DDB représentant $(\overline{CC} \vee \overline{C})$ et vérifier s'il se réduit à "1". Si c'est le cas, alors CC implique $\overline{C}$, et la variable CL de choix libre vaut 0.
- Si aucune des deux expressions ne se réduit à "1", alors le choix sur la valeur de vérité de la condition C est libre.

Le module de gestion de la Condition de Chemin renvoie la valeur de la variable de choix libre CL au simulateur. Si la valeur est 1, l'utilisateur est interrogé sur la valeur de la condition et la mise à jour de la Condition de

Chemin doit être effectuée.

La mise à jour de la Condition de Chemin avec la condition C (ou sa négation $\overline{C}$) consiste simplement à rajouter le DDB qui représente C (resp. $\overline{C}$) à la Condition de Chemin et à recalculer le graphe de dépendance. Notons que les DDB de C (resp. $\overline{C}$) ont déjà été construits pour le calcul d'implication.

5.6 - Complexité

Etudions maintenant la complexité de cette méthode de représentation. Les mesures possibles s'entendent en termes de taille des diagrammes, et en complexité des opérations.

5.6.1 - Taille du diagramme

Nous avons déjà signalé que l'ordonnancement des variables choisi pour la construction du diagramme de décision binaire associé à une fonction logique influait de façon très significative sur la taille du diagramme [MOR81, MOR82]. Le choix du meilleur ordonnancement des variables peut être effectué selon différents critères : la taille du diagramme, le coût du test d'une valeur de la fonction, ... [MOR81, MOR82].

La détermination du meilleur choix peut s'effectuer par comparaison de tous les diagrammes construits selon tous les ordonnancements possibles des variables (au nombre de $N_T(n) = n.(N_T(n-1))^2$ pour n variables, avec $N_T(1) = 1$, fonction qui croît plus vite que 2^{2^n}) ; il est évident que cette stratégie est inenvisageable pour des ensembles de plus de trois ou quatre variables. En réalité, la détermination de l'arbre optimal (en nombre de tests pour la détermination de la valeur de la

fonction pour une séquence particulière) est un problème NP-complet [HYA76] ; les méthodes efficaces de choix d'ordonnancements sont donc heuristiques. De nombreux travaux ont été développés sur ce thème [MOR81, MOR82, FUJ88, BUT91, ROS91].

Il est toutefois à noter que ce problème d'optimisation de taille de diagrammes ne se pose pas dans notre cas. Effectivement, la mise en œuvre de la construction de la Condition de Chemin à partir de la spécification, du calcul d'implication d'une valeur de vérité pour une condition, et de la mise à jour de la Condition de Chemin impliquent la manipulation de plusieurs diagrammes, et l'application de fonctions logiques binaires. Or, ces opérations ne sont efficaces et simples à implanter que sur des arbres (ou diagrammes) de décision binaire construits selon le même ordonnancement de variables.

Nous avons alors choisi de classer les variables selon l'ordre lexicographique.

La seule information que nous pouvons donc fournir sur la taille des données manipulées est que dans le pire des cas, pour une fonction de n variables, un arbre de décision binaire comporte de l'ordre de 2^n nœuds, le diagramme associé est de taille $O(\frac{2^n}{n})$ [BRY86, COU91].

En pratique, la plupart des diagrammes de décision binaire sont de taille linéaire par rapport au nombre de variables [MAD88].

Par exemple, le DDB qui représente la fonction qui réalise le OU ou le ET logiques de n variables contient (n + 2) nœuds.

De même, le DDB du OU-EXCLUSIF de n variables requiert (2n + 1) nœuds.

Ce mode de représentation est donc moins encombrant qu'une table de vérité (2^n valeurs), et beaucoup plus avantageux qu'une représentation sous forme de somme de produits (taille en $n.2^n$) [COU91].

5.6.2 - Négation d'une fonction

La négation d'une fonction est la tâche la plus coûteuse et la plus difficile à effectuer dans la plupart des systèmes de représentation de fonctions booléennes.

Cette opération requiert en effet 2^n opérations sur une table de vérité, et s'avère être très complexe sur des représentations algébriques de type somme de produits.

Par contre, un diagramme ne possédant qu'au plus deux feuilles (0 et 1), complémenter une fonction sur un DDB est une opération de coût linéaire (O(n)) [COU91]. Elle peut même être ramenée à un coût constant par un simple repérage direct des deux feuilles "0" et "1" ; cela évite alors le parcours obligatoire du diagramme à partir de la racine (en O(n)) pour atteindre ces feuilles.

5.6.3 - Opérations entre diagrammes

Les opérations binaires (et, ou, ou-exclusif, équivalence) sont de complexité O(|A|.|B|), |A| et |B| étant la taille (en nombre de nœuds) des deux diagrammes opérandes [BRY86].

Sachant que le nombre de nœuds est le plus souvent linéaire par rapport au nombre de variables, ce pire des cas est tout à fait satisfaisant par rapport à la complexité rencontrée dans les autres modes de représentation.

L'application d'une opération binaire entre deux diagrammes de décision binaire se termine toujours par une phase de simplification du diagramme résultat.

Les simplifications s'effectuent à deux niveaux : la première étape consiste à éliminer les nœuds redondants (dont les fils gauches et droits sont identiques)

de l'arbre de décision binaire construit ; la deuxième étape est la transformation de l'arbre de décision binaire réduit en diagramme de décision binaire par repérage des sous-arbres équivalents et remplacement de ces sous-arbres (sauf un) par un lien vers celui qui a été conservé.

Bien que théoriquement exponentielle (puisque basée sur un parcours d'arbre), cette procédure est indispensable puisqu'elle maintient les représentations des fonctions manipulées à une taille minimale pour l'ordonnancement des variables choisi. Cela permet alors de diminuer très significativement la taille des DDB, et donc la complexité des opérations logiques sur les diagrammes de décision binaire. De plus, le fait d'appliquer systématiquement ces procédures de minimisation réduit le DDB d'une fonction toujours vraie (resp. fausse) à la feuille "1" (resp. "0"). Ce résultat est primordial dans la phase de calcul d'implication.

Rappelons enfin que les méthodes de minimisation de fonctions logiques présentées auparavant sont soit inopérantes pour un nombre important de variables, soit plus complexes que la minimisation des diagrammes de décision binaire.

5.6.4 - Calcul d'implication

Tester si ($CC \Rightarrow C$) est vrai consiste simplement à vérifier que la racine du diagramme résultat de l'opération $\overline{CC} \vee C$ est la feuille "1". Cette opération est donc de coût constant (O(1)).

5.7 - Optimisations

Des diagrammes (représentant chacun une fonction logique) ayant des parties

identiques peuvent être partagés [MIN90]. Le diagramme partagé possède alors autant d'entrées que de fonctions manipulées. Cela permet une représentation certes moins lisible et plus délicate à manipuler, mais plus compacte. Ce type de représentations peut s'avérer intéressant pour des évaluations de fonctions qui varient peu.

Une autre amélioration possible consiste à typer les arcs vers les sous-graphes ; de ce fait, il est possible de ne représenter qu'une fois deux sous-fonctions complémentaires [MAD88, COU91].

Il est évidemment possible de combiner toutes les améliorations afin d'optimiser la représentation [MIN90].

Dans le cadre du problème qui nous préoccupe (gestion de la Condition de Chemin et calcul d'implication), une seule fonction booléenne est calculée. Les mécanismes de représentation et de manipulations de diagrammes multi-fonctions ne présentent donc pas d'intérêt.

5.8 - Conclusion

Nous avons établi que, globalement, les Diagrammes de Décision Binaire sont une méthode de représentation utilisable pour des fonctions logiques de plusieurs dizaines de variables. Les opérations logiques sont de complexité moindre que dans les autres modes de représentation ; notons également qu'une manipulation correcte de ces diagrammes implique qu'ils se trouvent toujours sous leur forme minimale pour l'ordonnancement de variables choisi.

Ces propriétés nous ont conduits à choisir les DDB pour la Gestion de la Condition de Chemin Booléenne, tâche qui requiert un grand nombre d'opérations logiques sur les fonctions logiques (construction de la Condition de Chemin et de la condition à évaluer, calcul de la variable de choix libre, mise à jour de la Condition de Chemin) dont les résultats doivent être sous leur forme minimale (interprétation du résultat du calcul d'implication pour la détermination de la variable de choix, complexité des opérations).

6 - Réalisations

Nous avons réalisé un logiciel permettant de gérer la Condition de Chemin Booléenne à l'aide des Diagrammes de Décision Binaire.
Nous pouvons distinguer trois parties :
- Construction de la Condition de Chemin à partir d'une spécification,
- Calcul de la variable de choix libre associé à une condition,
- Mise à jour de la Condition de Chemin.

Ce programme permet la saisie d'une spécification, puis propose un menu à l'utilisateur, qui lui offre le choix entre :
- construire la Condition de Chemin pour une date de simulation donnée,
- calculer la variable de choix pour une condition (entrée au clavier par l'utilisateur),
- afficher la Condition de Chemin courante.

Nous allons détailler les données, puis chacun de ces modules.

• La spécification

Initialement, l'utilisateur fournit :

- un ensemble de couples (condition booléenne , date). La date peut être numérique ou symbolique.
- un ensemble de conditions entre les dates.

• Construction de la Condition de Chemin à partir de la spécification

Le Diagramme de Décision Binaire correspondant à chacune des conditions booléennes de la spécification est construit (suivant l'ordre lexicographique des constantes symboliques).

• Construction de la Condition de Chemin pour une date de simulation

L'utilisateur indique une date de simulation.

La Condition de Chemin courante est un ensemble de DDB indépendants (un graphe par composante connexe). Le graphe de dépendance de la Condition de Chemin courante et des conditions valides pour cette date de simulation et toutes les dates antérieures est calculé. Les DDB de chaque nouvelle composante connexe sont alors construits. L'ensemble de DDB résultat de cette opération représente la nouvelle Condition de Chemin.

• Calcul de la variable de choix libre associée à une condition

L'utilisateur entre une condition booléenne au clavier. Le DDB correspondant à cette condition est construit.

L'ensemble de composantes connexes de la Condition de Chemin concerné par cette condition est déterminé. Le DDB résultat du ET logique de ces composantes connexes est construit ; notons le CC. C est le DDB associé à la

condition à évaluer.

Le DDB résultat du OU logique du complément de CC et de C est construit. Si ce DDB résultat se réduit à la feuille "1", la variable de choix-libre CL est fausse, et la Condition de Chemin implique la condition en vrai. Dans le cas contraire, le DDB résultat du OU logique du complément de CC et du complément de C est construit. Si ce DDB se réduit à la feuille "1", la variable de choix-libre CL est fausse, et la Condition de Chemin implique la condition en faux. Si aucun des deux DDB construits ne se ramène à la feuille "1", alors la variable de choix-libre CL est vraie.

• Mise à jour de la Condition de Chemin

Dans le cas où la variable CL de choix-libre est vraie, l'utilisateur est interrogé sur la valeur de vérité de la condition à évaluer. S'il répond que cette condition est vraie, la Condition de Chemin devient le résultat du ET entre CC et C. S'il répond que la condition est fausse, la Condition de Chemin est mise à jour avec la négation de C (ET logique de CC et du complément de C).

Le graphe de dépendance est recalculé, et les DDB des différentes composantes connexes mises à jour.

• Affichage de la Condition de Chemin

Cette fonction procède à l'affichage des DDB associés aux composantes connexes de la Condition de Chemin, en effectuant un parcours en profondeur de chacun des graphes.

Ce logiciel a été développé en langage C sur station de travail Apollo.

Il permet de représenter et manipuler des Diagrammes de Décision Binaire

fonction de plusieurs dizaines de variables dans des temps tout à fait raisonnables.

Par exemple, la construction d'un DDB associé à une condition fonction de 40 variables et le calcul d'implication pour une condition dont la valeur de vérité est contenue par la Condition de Chemin sont effectués en moins de 2,5 secondes. La construction du même DDB, le calcul d'implication pour une condition dont la valeur de vérité est libre, et la mise à jour de la Condition de Chemin occupent moins de 6 secondes CPU.

7 - Conclusion

Après avoir étudié différentes méthodes de représentation et de manipulation de fonctions booléennes, nous avons choisi les Diagrammes de Décision Binaire pour leur capacité à représenter des expressions logiques (fonction éventuellement de nombreuses variables), et pour la facilité de mise en œuvre des opérations logiques pour déterminer la variable de choix. Les opérations de construction et d'évaluation de la variable de choix sont basées sur l'utilisation exclusive des opérations de base sur les DDB.

Chapitre 4 - Gestion de conditions arithmétiques

Nous considérons ici le problème de gestion de la Condition de Chemin dans le cadre d'expressions logiques qui manipulent des variables arithmétiques. Face à la complexité de ce problème, nous nous limitons aux expressions booléennes d'inéquations linéaires. De plus, rappelons que le seul type arithmétique présent dans le sous-ensemble de VHDL retenu est le type entier. De ce fait, nous nous intéressons à des expressions booléennes d'inéquations linéaires à coefficients entiers.

Nous étudions le cas particulier où la Condition de Chemin est une conjonction d'inéquations et où la condition à évaluer est formée d'une seule inéquation. Après une présentation des méthodes qui peuvent s'appliquer pour le calcul de la variable de choix, nous étudions les limitations de ces systèmes et proposons une approche qui s'applique quand les solutions présentées auparavant ne peuvent pas être utilisées. Nous terminons par les travaux que nous avons réalisés.

1 - Calcul de la variable de choix

Nous avons vu dans le Chapitre 2 comment déterminer la valeur de la variable

de choix en fonction des ensembles de définition et des ensembles de solutions de la Condition de Chemin et de la condition à évaluer, et ce sans tenir compte de la forme des expressions booléennes manipulées.
Etant donné que nous manipulons ici des inéquations linéaires, nous établissons une méthodologie d'évaluation de la variable de choix spécifique à ces expressions.

Après avoir précisé la forme d'une inéquation, puis la structure de la Condition de Chemin et de la condition à évaluer, nous développons la méthode de calcul de la variable de choix appliquée au contexte d'expressions linéaires entières.

1.1 - Rappel : forme des expressions

Une inéquation C_j est de la forme :

$$C_j = (\sum_{i \in I} \lambda_{ij}\, x_i)\ @_j\ k_j$$

avec :

$\lambda_{ij}, x_i, k_j \in \mathbb{Z}$

$@_j \in \{<\ ;\ \leq\ ;\ >\ ;\ \geq\}$

$I = \{1, \ldots, n\}$

Notation :
Un système $(C_j)_{j \in J}$ d'inéquations représente la conjonction des expressions booléennes C_j :

$$(C_j)_{j \in J} = \bigwedge_{j \in J} C_j$$

1.2 - Principe du calcul de la variable de choix

Soit $CC = (C_j)_{j \in J}$ ($J = \{1, \ldots, m\}$) un système d'inéquations, et C une inéquation.

Notons S_{CC} l'espace de solutions de CC, S_C l'espace de solutions de C, $S_{\overline{C}}$ l'espace de solutions de $\overline{C}$.

S_{CC} est un ensemble de n-uplets $(c_1, \ldots, c_n) \in \mathbb{Z}^n$ tels que :

$$\forall j \in J, \models (\sum_{i \in I} \lambda_{ij} c_i) @_j k_j$$

De même, si $C = (\sum_{i \in I} \lambda_i x_i) @_C k_C$

S_C est un ensemble de n-uplets $(c'_1, \ldots, c'_n) \in \mathbb{Z}^n$ tels que :

$$\models (\sum_{i \in I} \lambda_i c'_i) @_C k_C$$

La variable de choix libre associée à C vaut 0 si l'espace de solutions de CC est inclus dans l'espace de solutions de C ou de $\overline{C}$ (Cf chapitre 1, §6)

Considérons le cas où l'espace de solutions de CC est inclus dans celui de C ($S_{CC} \cap S_C = S_{CC}$). Alors en notant (CC,C) le système d'inéquations formé du ET de CC et de C, nous avons immédiatement $S_{(CC,C)} = S_{CC}$. De plus, $S_{CC} \cap S_{\overline{C}} = \emptyset$ et $S_{(CC,\overline{C})} = \emptyset$.

De même, si l'espace de solutions de CC est inclus dans celui de $\overline{C}$, alors nous vérifions que $S_{(CC,\overline{C})} = S_{CC}$, et $S_{(CC,C)} = \emptyset$.

En résumé :

$CC \models C$ si $S_{CC} = S_{(CC,C)}$

$CC \models C$ si $S_{(CC,\overline{C})} = \emptyset$. Dans ce cas, le système $(CC,\overline{C})$ est inconsistant

$CC \models \overline{C}$ si $S_{CC} = S_{(CC,\overline{C})}$

$CC \models \overline{C}$ si $S_{(CC,C)} = \emptyset$. Dans ce cas, le système (CC,C) est inconsistant

Dans les autres cas ($S_{CC} \neq S_{(CC,C)}$ et $S_{(CC,\overline{C})} \neq \emptyset$ et $S_{CC} \neq S_{(CC,\overline{C})}$ et $S_{(CC,C)} \neq \emptyset$), le choix est libre (la variable de choix libre associée à C vaut 1).

Il est donc possible de déterminer la valeur de la variable de choix libre associée à C, soit en calculant les espaces de solutions des systèmes CC et (CC,C) et en les comparant, soit en évaluant la consistance du système $(CC,\overline{C})$.

Cette approche du problème de calcul de la variable de choix libre est basé sur le calcul des espaces de solutions des systèmes d'inéquations formés de la Condition de Chemin et de la condition à évaluer (ou sa négation). Les solutions mathématiques classiques pour résoudre ce problème sont empruntées à la Programmation Linéaire. Les méthodes de Programmation par Contraintes permettent également d'évaluer la variable de choix libre, ainsi que les méthodes basées sur la démonstration de théorèmes. Nous étudions donc l'adéquation de ces différents outils pour la gestion de la Condition de Chemin, et présentons l'approche que nous avons retenue.

2 - Résolution de systèmes d'inéquations

Nous pouvons envisager deux façons d'évaluer la variable de choix-libre

associée à la condition C connaissant la Condition de Chemin CC :
- déterminer si les espaces de solutions des systèmes CC et (CC,C) sont égaux,
- étudier la consistance du système (CC,$\overline{C}$).
La première méthode implique le calcul des solutions des deux systèmes d'inéquations, alors que la seconde nécessite une évaluation de la consistance.

Parmi l'ensemble des outils mathématiques, deux classes de méthodes sont applicables à ce problème :
- l'analyse numérique matricielle (recherche des solutions d'un système d'équations linéaires),
- la programmation linéaire (optimisation d'une fonction sous un ensemble de contraintes).

2.1 - Rappel : Résolution de systèmes d'équations linéaires par calcul matriciel

L'analyse numérique matricielle fournit des méthodes de détermination des solutions d'un système d'équations linéaires sur le corps ($\mathbb{R}$,+,.) des réels.
Les différentes méthodes de résolution de systèmes d'équations linéaires requièrent la donnée de n équations indépendantes à n inconnues. Ce système s'écrit alors directement sous forme matricielle (A.u = b), la matrice A des coefficients des variables (qui figurent dans le vecteur u) étant carrée et inversible (régulière) ; b est le vecteur des parties constantes.

Les différentes méthodes de résolution du système sont les suivantes:
- méthode de Cramer : résoudre le système revient à trouver les valeurs des

variables de u. Pratiquement, on calcule la matrice inverse A^{-1} de A. On a alors : $A^{-1}.A.u = A^{-1}.b$ soit $u = A^{-1}.b$.

Cette méthode est basée sur l'évaluation de (n+1) déterminants de matrices carrées de rang n. Le calcul d'un déterminant nécessite ((n!)-1) additions et (n-1)n! multiplications. Au total, la résolution d'un système d'équations par cette technique requiert de l'ordre de (n+1)! additions, (n+2)! multiplications et n divisions.

- méthode de Gauss : le principe est de déterminer la matrice M telle que M.A soit triangulaire supérieure. Le vecteur M.b est également calculé et il reste à résoudre le système M.A.u = M.b. M.A étant triangulaire supérieure, la résolution peut être effectuée par la méthode de remontée : sur la $n^{ième}$ ligne, seul le coefficient de la $n^{ième}$ colonne est non-nul. La valeur de u_n ($n^{ième}$ composante du vecteur u) est donc immédiatement obtenue en divisant la $n^{ième}$ composante de M.b par le coefficient de la $n^{ième}$ ligne, $n^{ième}$ colonne de M.A.

Connaissant u_n, l'étape suivante consiste à calculer la valeur de u_{n-1} de la même façon ; et ainsi de suite jusqu'à u_1.

En pratique, la matrice M n'est pas explicitement calculée ; la méthode recherche la matrice triangulaire supérieure (produit de M et A) par le biais d'une méthode de pivot. Le calcul de M.A (respectivement M.b) nécessite $\frac{n^3-n}{3}$ (respectivement $\frac{n(n-1)}{2}$) additions et $\frac{n^3-n}{3}$ (respectivement $\frac{n(n-1)}{2}$) multiplications. La résolution par remontée effectue $\frac{n(n-1)}{2}$ additions, $\frac{n(n-1)}{2}$ multiplications et n divisions.

Globalement, la méthode de Gauss pour la résolution d'un système d'équations linéaires requiert de l'ordre de $\frac{n^3}{3}$ additions, $\frac{n^3}{3}$ multiplications et $\frac{n^2}{2}$ divisions.

- techniques spécifiques de résolution pour des matrices ayant des propriétés particulières : méthode de Cholesky si A est symétrique définie positive,

factorisation LU de matrice tridiagonale, etc [CIA88].

Ces méthodes concernent la résolutions de systèmes de n équations à n inconnues. Or, la donnée du problème de calcul de la variable de choix est un système de m inéquations à n inconnues.

Afin d'utiliser les techniques de calcul des solutions présentées ci-dessus, il est nécessaire de ramener ce système d'inéquations à un système d'équations :

1 - Exprimer toutes les inéquations avec le même opérateur relationnel (≤).

2 - Transformer les inéquations en équations en introduisant des variables d'écart, puis rajouter au système les contraintes sur les variables d'écart (positives).

Le système obtenu contient alors m équations à (n+m) inconnues, sachant que les variables d'écart sont positives. n et m étant strictement positifs, le système comporte plus d'inconnues que d'équations.

Exemple :

$$\begin{cases} x + y < 4 \\ x \geq -2 \\ x - y \leq 0 \end{cases} \rightarrow \begin{cases} x + y \leq 3 \\ -x \leq 2 \\ x - y \leq 0 \end{cases} \rightarrow \begin{cases} x + y + e_1 = 3 \\ -x + e_2 = 2 \\ x - y + e_3 = 0 \\ (e_1, e_2, e_3) \in \mathbb{R}^+ \end{cases}$$

La résolution s'effectuera en considérant ((n+m)-m=n) inconnues comme étant des paramètres. Il ne faudra conserver que les solutions entières pour lesquelles les variables d'écart sont positives.

Remarque :

L'algorithme de Gauss (et ceux plus spécifiques) de résolution de systèmes d'équations linéaires sont de complexité polynômiale dans le cadre du calcul numérique.

Dans le cas présent, la détermination des solutions (par remontée à partir de la matrice triangulaire supérieure pour la méthode de Gauss) nécessite la mise en œuvre des méthodes de calcul symbolique (expression des opérandes sous forme normalisée, exécution des opérations, simplification et expression du résultat sous forme normalisée).

Conclusion :

La mise en œuvre de ces méthodes de résolution nécessite :

1 - la traduction du système d'inéquations sous forme d'un système d'équations avec introduction de variables d'écart.

2 - la recherche d'une matrice carrée inversible dans le système.

3 - la résolution du système en considérant certaines variables d'écart comme des paramètres.

4 - la restriction des domaines des variables aux solutions pour lesquelles les variables d'écart considérées comme paramètres sont positives.

5 - la restriction des domaines des variables d'écart considérées comme variables du système à leur partie positive.

6 - la détermination de l'existence de solutions entières, si l'espace de solutions est non-vide.

Du fait que les coefficients manipulés dans les expressions sont entiers, la première étape ne présente pas de difficulté. Effectivement, si E est une expression linéaire et k un entier relatif, alors les transformations à opérer sont les suivantes :

$$E \leq k \quad \Rightarrow \quad E \leq k$$

$$E < k \quad \Rightarrow \quad E \leq k-1$$

$$E \geq k \quad \Rightarrow \quad -(E) \leq -k$$

$$E > k \quad \Rightarrow \quad -(E) \leq -(k+1)$$

$$E = k \quad \Rightarrow \quad \begin{cases} E \leq k \\ E \geq k \end{cases}$$

On obtient alors une inéquation de la forme ($E' \leq k'$). L'équation associée est obtenue par introduction d'une variable d'écart e positive :

$$\begin{cases} E' + e = k' \\ e \geq 0 \end{cases}$$

L'introduction de m variables d'écart rend les m équations linéairement indépendantes. Il est alors aisé d'extraire un sous-système dont la matrice associée est régulière. Les paramètres sont choisis autant que possible parmi les variables d'écart.
La résolution proprement dite du système est alors confiée aux systèmes classiques de résolution d'équations linéaires évoqués auparavant.

La cinquième étape détermine si les domaines des variables d'écart prises comme variables du système sont vides. Si un domaine au moins est vide, alors le système est inconsistant.

La dernière étape consiste à extraire des solutions du système celles qui sont entières.
Or, les méthodes de résolution opèrent sur le corps des réels et manipulent les inverses (symétriques pour la multiplication) des éléments de la matrice (on rappelle que l'inverse d'un entier n'appartient pas à $\mathbb{Z}$). La procédure de calcul effectue alors des arrondis sur les nombres décimaux qu'elle manipule. Il est donc théoriquement impossible de déterminer les solutions entières. En pratique, il faut "reconnaître" des solutions entières (en fixant une valeur pour les

erreurs), et vérifier qu'elles sont effectivement solutions du système. Cependant, rien n'assure que toutes les solutions entières sont trouvées par cette méthode.

Pour tester l'égalité d'espaces de solutions de deux systèmes, ou déterminer l'inconsistance d'un système d'inéquations, il est nécessaire de connaître les solutions. En résumé, le calcul de la variable de choix basé sur les méthodes de calcul matriciel est envisageable sur les réels mais pose de graves problèmes lors du passage aux entiers.

2.2 - Méthodes de programmation linéaire

Nous étudions dans ce paragraphe comment les méthodes de programmation linéaire peuvent être utilisées pour le calcul de variables de choix-libre.
Le propos de la programmation linéaire est l'optimisation d'une fonctionnelle sous certaines contraintes (système d'inéquations).
Si l'ensemble de contraintes est consistant, il existe un polyèdre convexe solution du système d'inéquations. Ce polyèdre possède un nombre fini non nul de sommets. Si l'optimisation de l'expression est possible, la solution est un sommet de ce polyèdre [CIA88].

Une première méthode consiste à déterminer les coordonnées de tous les sommets et à choisir celui pour lequel la valeur de la fonction à optimiser est la plus satisfaisante.
Mais si le système est composé de m contraintes à n inconnues, (n+m) hyperplans sont définis dans l'espace, et les intersections n à n sont au nombre de :

$$C_{n+m}^{n} = \frac{(m+n)!}{m!\ n!}$$

Le nombre de points à considérer est vite inacceptable pour des valeurs raisonnables de n et m (plus de 3 millions de points si n=10 et m=15). De plus, la grande majorité de ces points ne présentent pas d'intérêt puisque à l'extérieur du polyèdre. Il est plus judicieux de travailler directement sur les sommets du polyèdre sans tous les calculer de façon explicite.

La méthode du simplexe [DAN63] repose sur l'évaluation de la fonctionnelle à optimiser en certains sommets du polyèdre construits par récurrence : partant d'un sommet u_0, il est possible de construire une suite $u_0, u_1, \ldots, u_k$ de sommets correspondant à des valeurs croissantes (resp. décroissantes) de la fonctionnelle à maximiser (resp. minimiser). Le nombre de sommets du polyèdre étant fini, ce procédé conduit à une solution en un nombre fini d'itérations [FAU79].

La complexité de cet algorithme est exponentielle dans le pire des cas. En pratique, l'exécution est en temps polynômial.

Bien que l'objectif poursuivi ne soit pas l'optimisation d'une fonction, il est possible d'utiliser cette méthode pour le calcul de la variable de choix. En effet, CC est le système de contraintes initial, et C est une nouvelle contrainte.

Plus précisément, si l'expression de C est :

$$(\sum_{i \in I} \lambda_i x_i)\ @_C\ k_C$$

alors minimiser, puis maximiser la partie linéaire de C donnent k_1 et k_2 réels tels que :

$$\text{Inf}\ (\sum_{i \in I} \lambda_i x_i) = k_1$$

$$\text{Sup}\ (\sum_{i \in I} \lambda_i x_i) = k_2$$

Il suffit alors de comparer k_C par rapport à k_1 et k_2 et de déterminer la valeur de la variable de choix-libre CL. Plus précisément, l'interprétation du résultat est la suivante :

si $(k_1 < k_C < k_2)$ ou $(k_2 < k_C < k_1)$
alors CC n'implique pas de valeur de vérité pour C, et CL = 1.

si $(k_C @_C k_1)$ et $(k_C @_C k_2)$
alors CC $\models$ C, et CL = 0.

si $(k_1 @_C k_C)$ et $(k_2 @_C k_C)$
alors CC $\models \overline{C}$, et CL = 0.

Remarque :
Le simplexe ne s'applique que si le système de contraintes est consistant. Cette méthode n'est donc envisagée que lorsque l'existence de solutions pour le système d'inéquations qui compose CC est prouvée.

Discussion :
Cette méthode de calcul de la variable de choix-libre d'une condition C sous la contrainte CC est applicable sur le domaine des réels. La démarche adoptée dans l'algorithme du simplexe assure en effet la consistance de CC ; par contre, aucune information relative à l'existence de solutions entières dans le polyèdre déterminé par CC n'est fournie. Il est donc indispensable de mettre en œuvre un mécanisme de calcul de solutions entières.
Il existe néanmoins des cas particuliers pour lesquels les solutions (sommets du polyèdre) sont entières : la matrice associée au système est :

- équilibrée (la matrice ne contient aucune sous-matrice carrée d'ordre impair dont les sommes de chaque ligne et de chaque colonne sont égales à 2),
- totalement unimodulaire (le déterminant de toute sous-matrice carrée est égal à 0, -1 ou +1. Cela implique en particulier que les coefficients de la matrice sont égaux à 0, -1 ou +1).

Dans le cas où les solutions obtenues ne sont pas entières, la méthode des troncatures [GOM58] s'applique sur la solution optimale déterminée par le simplexe. Elle est basée sur la propriété suivante : si deux variables sont entières, alors la différence entre ces variables (ou les expressions qui les représentent) est également entière (divisible par 1). L'application de cette propriété aux expressions du système définit de nouvelles contraintes ; l'algorithme du simplexe est alors réitéré sur ce nouveau système et la solution finale, si elle existe, est composée de valeurs entières. Des variantes de cette méthode sont proposées dans [GON73] et [GON85]. Ces procédures sont de complexité exponentielle ; elles convergent en un nombre fini de pas, mais ne sont réellement efficaces (peu d'itérations) que dans le cas particulier où les coefficients des variables appartiennent à {-1 ; 0 ; +1}.

2.3 - Synthèse

Les méthodes mathématiques classiques utilisables pour le calcul de variables de choix sont basées sur des algorithmes complexes qui calculent toutes les solutions (calcul matriciel) ou des solutions particulières (simplexe).

Elles fournissent des résultats directement utilisables dans le domaine des réels. Par contre, l'existence de solutions entières n'est pas prouvée.

Des méthodes qui utilisent les résultats du simplexe existent, mais la complexité

introduite n'est acceptable que dans le cadre de cas particuliers.

3 - Programmation par Contraintes

La Programmation Logique avec Contraintes (CLP) intègre les principes de programmation logique (inférence logique) et les mécanismes de résolution de contraintes sur des domaines spécifiques (réels, booléens, entiers, ...) [JAF87a, JAF87b]. L'unification syntaxique de l'Univers de Herbrand est remplacée dans le processus de résolution par la satisfaction de contraintes dans le domaine d'applications. Cette prise en compte des contraintes permet de pallier de façon très significative le principal inconvénient des systèmes de programmation logique (recherche des solutions de façon aveugle) qui est l'inefficacité.
Nous étudions comment les principaux systèmes de programmation avec contraintes que sont PROLOG III, CHIP, CLP($\mathcal{R}$) et CONSLOG peuvent traiter le problème de la gestion d'une Condition de Chemin arithmétique, et en particulier le calcul de la variable de choix-libre.

3.1 - CHIP

Dans le cadre de l'ECRC (European Computer-Industry Research Center), l'équipe dirigée par M. DINCBAS a développé CHIP (Constraint Handling In Prolog) qui s'attaque à des problèmes combinatoires discrets [DIN88].
Son originalité réside dans son exploitation active des contraintes avant toute génération de solution.

Les domaines d'application de CHIP sont les booléens, les rationnels et les domaines finis (entiers). Ce langage est donc directement utilisable pour le calcul de la variable de choix sur les inéquations linéaires à valeurs entières que nous avons définies.
Notons que CHARME [OPL89] est une version industrielle du langage.

3.2 - CLP($\mathcal{R}$)

CLP($\mathcal{R}$) a été développé chez I.B.M. [JAF88]. Son domaine d'application est l'ensemble des réels.
Le principe de résolution est de construire des séquences de dérivation qui produisent soit une résolvante vide (succès), soit une contrainte résiduelle, expression symbolique qui exprime les conditions nécessaires à la satisfaction du but.
L'obtention d'une contrainte résiduelle (resp. d'un échec) sur les réels avec CLP($\mathcal{R}$) signifie que le choix est libre (resp. que le système est inconsistant). Par contre, la réciproque n'est pas vérifiée : un succès (effacement d'un but) sur les réels n'implique pas forcément un succès sur les entiers.

3.3 - CONSLOG

Ce système [HAO91] travaille sur les domaines finis (entiers positifs).
Contrairement aux systèmes présentés précédemment, qui intègrent fortement la gestion des contraintes et les mécanismes d'inférence logique, CONSLOG s'applique à séparer le plus possible ces deux tâches (intégration faible).

L'intérêt de cette démarche réside dans la modularité qui permet le choix des outils (algorithmes de résolution de contraintes, systèmes d'inférence).
Après une phase d'évaluation partielle qui produit un premier ensemble de contraintes, le système exécute une boucle de résolution de contraintes ; cette phase se décompose en deux parties : raffinement des intervalles des contraintes, puis génération des valeurs pour les variables. Des résultats de consistance partielle sont disponibles après chaque phase de raffinement, mais la consistance globale n'est établie qu'à la fin de l'exécution.
Le domaine d'application de ce langage est l'ensemble des entiers ; il peut donc être utilisé pour la résolution de notre problème.

3.4 - Conclusion

Quelle que soit la méthode adoptée pour le calcul de la variable de choix (égalité des espaces de solutions de CC et (CC,C) ou inconsistance de $(CC,\overline{C})$), nous remarquons que :
- CHIP et CONSLOG travaillant sur le domaine des entiers, ces systèmes sont directement utilisables pour le calcul d'implication.
- la mise en œuvre de CLP($\mathfrak{R}$) pour l'évaluation de la variable de choix nécessite des fonctionnalités supplémentaires pour l'extraction des solutions entières de l'ensemble des solutions réelles.

Dans le cas du calcul de la variable de choix basé sur l'évaluation de la consistance de $(CC,\overline{C})$, aucun des systèmes de programmation par contraintes n'effectue de calcul de consistance globale avant de rechercher les solutions (qui ne présentent pas d'intérêt pour notre application). Au contraire, le calcul de consistance est étroitement lié à la détermination des solutions ; une méthode

générale d'évaluation de la consistance sans calcul des solutions ne peut donc pas être déduite des mécanismes de résolution de ces différents systèmes.

4 - Démonstration de théorèmes

Nous présentons brièvement les travaux effectués dans le domaine de la vérification de programmes par KING et FLOYD [KIN69, KIN72b].

Les éléments de base manipulés par les programmes à vérifier étant les entiers, cette méthode cherche à prouver la validité d'expressions arithmétiques (équations ou inéquations) et utilise pour cela un démonstrateur de théorèmes.

Pour prouver la validité d'une formule, le démonstrateur cherche à prouver l'inconsistance de sa négation. Pour cela, le système suit les deux étapes suivantes :

1 - Prise en compte du problème global :

Les formules à prouver sont données sous forme normale disjonctive. Leur négation (forme normale conjonctive) est construite. A ce stade, après toute manipulation des formules qui se produira par la suite, toute variable définie par une égalité est substituée par sa valeur dans les formules. Les fonctions spéciales (valeur absolue, modulo,...) sont éliminées.

2 - Décomposition du problème global en sous-problèmes :

Le problème global est décomposé pour obtenir autant que possible des expressions linéaires. Le système fait alors appel à un solveur linéaire qui calcule les éventuelles solutions entières du système d'équations ou inéquations linéaires extrait.

Le solveur linéaire de ce démonstrateur de théorèmes effectue une recherche

des solutions entières en deux étapes : il recherche tout d'abord l'ensemble des solutions réelles, et tente d'en extraire les solutions entières.

Conclusion :

Le problème de l'extraction des éventuelles solutions entières de l'ensemble des solutions réelles n'est pas résolu dans le solveur linéaire. En effet, si l'existence de solutions réelles est prouvé, les résultats quant à la non-existence de solutions entières ne sont pas garantis.

Notons également la grande complexité de ce démonstrateur de théorèmes, qui ne peut aborder que des exemples de taille modeste. Il nécessite en effet de l'ordre de 10 secondes pour démontrer un seul théorème ; le problème est plus aigu dans le cas où la formule à démontrer n'est pas valide.

La méthode de manipulation d'expressions arithmétiques pour le calcul de la variable de choix par démonstration de théorèmes rencontre donc les mêmes limitations que les procédures mathématiques ou que la programmation par contraintes en ce qui concerne la fiabilité des résultats sur les entiers, ainsi que les performances en termes de temps de calcul.

5 - Gestion d'une Condition de Chemin arithmétique

La gestion de la Condition de Chemin nécessite des mécanismes d'extraction de la Condition de Chemin à partir d'une spécification, d'évaluation de la variable de choix et de mise à jour de la Condition de Chemin.

• Tout comme dans le cas d'une Condition de Chemin purement booléenne, l'extraction des conditions (en fonction des constantes symboliques) de la

spécification est confiée aux outils développés dans le simulateur symbolique [ROG92a].

La construction de la Condition de Chemin à partir de la spécification consiste à ajouter à la Condition de Chemin courante les expressions fonction des constantes symboliques extraites de la spécification. Plus précisément, il faut effectuer le ET logique entre la Condition de Chemin et chacune des conditions.

Si toutes les conditions sont composées d'une seule inéquation ou de la conjonction de plusieurs inéquations, alors le résultat du ET logique sera une conjonction d'inéquations. Par contre, chaque disjonction introduit un nouveau système.

Exemple :

$C_1 : (x+y<6 \wedge x-z>2)$

$C_2 : (y<0 \vee x+z<4)$

La conjonction de C_1 et C_2 définit la disjonction des deux systèmes suivants :

$$\begin{cases} x+y<6 \\ x-z>2 \\ y<0 \end{cases} \vee \begin{cases} x+y<6 \\ x-z>2 \\ x+z<4 \end{cases}$$

Il faut ensuite construire le graphe de dépendance de la Condition de Chemin et calculer les composantes connexes.

• En ce qui concerne le calcul de la valeur de la variable de choix libre d'une condition C, nous avons étudié les différentes méthodes classiques qui peuvent être utilisées.

Il apparaît que toutes ces techniques mettent en œuvre des mécanismes de résolution qui recherchent des résultats inutiles pour notre application. Ces outils sont donc d'une complexité trop importante, et ne peuvent pas être utilisés dans la plupart des cas.

Nous avons donc défini une méthode d'évaluation de la variable de choix qui prend en compte la spécificité de notre problème.

• Mettre à jour la Condition de Chemin CC consiste à l'enrichir de l'inéquation qui correspond à la condition ou la négation de la condition selon le choix de l'utilisateur. Il s'agit donc simplement d'effectuer le ET logique de CC avec C ou $\overline{C}$, et de recalculer le graphe de dépendance.

5.1 - Forme des expressions

Nous nous limitons à des inéquations linéaires à coefficients entiers dont la valeur absolue est soit 0, soit 1. Les variables sont à valeurs dans l'ensemble des entiers relatifs. La constante de l'expression appartient à l'ensemble des entiers relatifs.

En résumé, une expression est de la forme :

$$(\sum_{i \in I} \lambda_{ij}\, x_i)\ @_j\ k_j$$

avec :

$\lambda_{ij} \in \{-1; 0; 1\}$

$@_j \in \{<; \leq; >; \geq\}$

$x_i, k_j \in \mathbb{Z}$

$I = \{1, \ldots, n\}$

Remarque :

Dans le cas d'une expression à coefficients entiers quelconques, nous procédons à un changement de variables qui permet de traduire une expression à coefficients entiers quelconques dans le formalisme défini ci-dessus.

Compte tenu de la forme particulière des inéquations manipulées, nous

pouvons étudier une autre méthode de calcul de la variable de choix basée sur le fait qu'une expression est impliquée par un ensemble d'expressions S si elle est combinaison linéaire des éléments de S.

5.2 - Méthode de calcul de la variable de choix

5.2.1 - Définitions

Soit $X = \{x_1, \ldots, x_n\}$ un ensemble de variables à valeurs dans $\mathbb{Z}$.

Soit C une inéquation.

$$C = (\sum_{i \in I} \lambda_i x_i) \; @ \; k$$

avec : $\lambda_i \in \{-1; 0; 1\}$

$@ \in \{<; \leq; >; \geq\}$

$x_i, k \in \mathbb{Z}$

$I = \{1, \ldots, n\}$

• La forme normalisée de C, notée C_N, est définie comme suit :

$$C_N = (\sum_{i \in I} \lambda_{i_N} x_i) \; @_N \; k_N$$

avec : $C = C_N$

$\lambda_{i_N} \in \{-1; 0; 1\}$

$@_N \in \{<; >\}$

$k_N \in \mathbb{Z}$

$[(\lambda_j \neq 0) \wedge (\forall k < j, \lambda_k = 0)] \Rightarrow (\lambda_{j_N} = 1)$

• Deux inéquations sont comparables si leurs formes normalisées possèdent le même opérateur relationnel.

• La signature d'une inéquation est la partie linéaire de cette expression suivant l'ordonnancement $(x_1, \ldots, x_n)$ des variables.

On a alors :

$$sig(C) = \sum_{i=1}^{n} \lambda_i x_i$$

On note également : $sig(C) = (\lambda_1, \ldots, \lambda_n)$.

• Soit S un système d'inéquations linéaires $(C_1, \ldots, C_m)$.

C est une combinaison linéaire possible si et seulement si :

$$\exists\, (\alpha_1, \ldots, \alpha_m) \in \{ -1; 0 ; +1 \}^m / sig(C) = \lambda . \sum_{i=1}^{m} \alpha_i . sig(C_i) , \lambda \in \mathbb{Z}^*$$

et

$$\forall\, (j, k) \in \{ 1, \ldots, m \}^2 , j \neq k, \alpha_j . C_j \text{ et } \alpha_k . C_k \text{ sont comparables}$$

• Soit S un système d'inéquations linéaires.

$$S = (C_i)_{i \in \{1,\ldots,n\}}$$

On note $S_0 = S$.

Le système S_{k+1} est construit de la façon suivante :

$$S_{k+1} = S_k \cup S'$$

avec : $S' = (C'_j)$

C'_j : combinaison linéaire possible obtenue à partir de S_k.

Si $S_{k+1} = S_k$ alors on note $S_k = S_F$.

Le système S_F d'inéquations linéaires est appelé système complet associé à S.

5.2.2 - Théorème 1

> Soit S un système d'inéquations linéaires.
>
> Soit C une inéquation linéaire.
>
> $C \in S_F$
>
> $\Leftrightarrow$
>
> C est une combinaison linéaire des inéquations de S.

Démonstration :

* $C \in S_F \Rightarrow$ C est une combinaison linéaire des inéquations de S

Immédiat.

* C est une combinaison linéaire des inéquations de S $\Rightarrow C \in S_F$

C combinaison linéaire des inéquations de S.

Alors il existe $(\alpha_1,\ldots,\alpha_n) \in \mathbb{Z}^n / \sum_{i=1}^{n} \alpha_i.C_i = \lambda.C, \lambda \in \mathbb{Z}^*$

Pour tout $j \in \{1,\ldots,n\}$,

si $\alpha_j \notin \{-1 ; 0 ; 1\}$ alors

si $(\alpha_j > 1)$ alors $\quad \lambda.C = \alpha_1.C_1 + \ldots + (\alpha_j - 1).C_j + C_j + \ldots + \alpha_n.C_n$

$$\text{d'où : } \lambda.C = (\alpha_j - 1).C_j + \sum_{i=1}^{n} \alpha'_i.C_i$$

avec : $\alpha'_i = \alpha_i$ si $i \neq j$

$\alpha'_j = 1$

si $(\alpha_j < -1)$ alors $\quad \lambda.C = \alpha_1.C_1 + \ldots + (\alpha_j + 1).C_j - C_j + \ldots + \alpha_n.C_n$

$$\text{d'où : } \lambda.C = (\alpha_j + 1).C_j + \sum_{i=1}^{n} \alpha'_i.C_i$$

avec : $\alpha'_i = \alpha_i$ si $i \neq j$

$\alpha'_j = -1$

5.2.3 - Théorème 2

> S_F est obtenu après un nombre fini d'itérations.

Démonstration :

Les coefficients α_i étant des entiers finis, la démonstration est immédiate d'après celle du théorème 1.

5.2.4 - Théorème 3

> Soit S un système d'inéquations linéaires.
>
> Soit C une inéquation linéaire.
>
> Une condition nécessaire pour que S |= C est la suivante :
>
> $\exists\, C_i \in S_F$ / sig(C) = sig(C_i)

5.2.5 - Théorème 4

> Soit S un système d'inéquations linéaires.
>
> Soit C une inéquation linéaire :
>
> $$C = (\sum_{i \in I} \lambda_i x_i)\ @_C\ k_C$$
>
> S |= C si et seulement si :
>
> - $\exists\, C_j \in S_F$ / sig(C) = sig(C_j)
>
> $$C_j = (\sum_{i \in I} \lambda_{ij} x_i)\ @_j\ k_j$$
>
> - $@_C$ et $@_j$ sont identiques
> - $|= (k_j\ @_C\ k_C)$.

5.2.6 - Interprétation géométrique

Dans l'espace E de dimension n, le système S_F associé à S, s'il a des solutions, délimite un polyèdre appelé E_S. Une inéquation C définit un hyperplan H dans E. Cet hyperplan délimite deux sous-espaces de E, E_C et $E_{\overline{C}}$, avec $E_C \cup E_{\overline{C}} = E$. Le calcul de la variable de choix consiste à déterminer si E_S est "coupé" par H ou non. Si H ne "coupe" pas E_S, alors C est impliquée en vrai ou en faux selon que E_S se trouve dans E_C ou dans $E_{\overline{C}}$. Si H et E_S intersectent, alors on ne peut pas conclure sur l'implication de C par le système S, et le choix est libre.

5.2.7 - Méthode de résolution

Construction

• Si le système S contient une seule inéquation C, le système complet S_F contient C et $\overline{C}$.

• Dans le cas où le système S contient plusieurs inéquations, si S_F est le système complet correspondant, alors rajouter l'inéquation C à ce système revient à effectivement rajouter C à S_F, ainsi que toutes les combinaisons linéaires possibles de C et des inéquations de S_F. Ce processus doit être itéré jusqu'à obtention du système complet.

Calcul de la variable de choix

Une fois le système complet S_F construit, déterminer la variable de choix d'une inéquation C revient à rechercher dans S_F l'ensemble E des inéquations normalisées qui ont la même signature que C (vérification de la condition nécessaire). Si E est vide, alors le choix est libre, sinon, s'il existe une inéquation C_j de même opérateur relationnel que C ($@_C$), alors la variable de choix est fausse si la relation (k_j $@_C$ k_C) est vraie (et C est vraie), ou si la relation ((k_C+1) $@_C$ k_j) est vraie (et C est fausse) (k_j et k_C sont les constantes de C_j et C

respectivement).

Mise à jour

Si la variable de choix-libre associée à C est vraie, alors l'utilisateur intervient et fixe la valeur de vérité de C. La mise à jour du système S consiste à rajouter C (resp. $\overline{C}$) au système complet S_F. Cette opération s'effectue par le biais de la construction d'un système résultat du ET logique entre deux systèmes (tel que défini ci-dessus).

5.3 - Structure de données

Afin de maintenir une certaine cohérence avec la représentation choisie pour les expressions booléennes, nous avons opté pour une structure arborescente. Un arbre représente une expression ou un système d'expressions.

Un nœud non-terminal de l'arbre représente une variable.

Trois arcs partent de ce nœud : un premier vers le sous-arbre qui représente les expressions si le coefficient associé à la variable est égal à -1, le second pour le coefficient 0, et enfin le dernier pour le coefficient égal à +1.

Un nœud terminal représente l'ensemble des qualifications de l'expression représentée par le chemin depuis la racine. Il peut porter une qualification ($< k_1$) qui indique que l'expression est inférieure à l'entier relatif k_1 ; de la même façon, le nœud peut porter l'information ($> k_2$), ou les deux informations.

Un nœud terminal qui représente une expression qui n'est pas qualifiée porte l'information "indéfini", notée ϕ.

Soit A un arbre.

• Un nœud de l'arbre qui représente la variable x_i est repéré par $A(x_i)$.

• Les fils d'un nœud $A(x_i)$ définis pour les coefficients -1, 0 et +1 respectivement, sont appelés $A(x_i).M$, $A(x_i).Z$ et $A(x_i).P$ respectivement.

• Si A(y) est un nœud terminal, alors :

* $A(y) = \phi$ si aucune qualification n'existe pour l'expression.

* $A(y).I = k_1$ si l'expression est inférieure à k_1

  $A(y).S = k_2$ si l'expression est supérieure à k_2.

5.4 - Construction

Etudions dans un premier temps la construction de l'arbre qui représente un simple inéquation, puis les règles de construction pour un système d'inéquations.

5.4.1 - Construction de l'arbre représentant une inéquation

• La première étape consiste à exprimer l'inéquation sous forme normalisée. Pour cela, il faut opérer les transformations éventuelles des opérateurs relationnels ≤ et ≥ en < et >.

Soit C une inéquation.

$$C = (\sum_{i \in I} \lambda_i x_i) \; @ \; k_C$$

Alors $C = (\sum_{i \in I} \lambda_i x_i) \; @' \; k'_C$ avec :

si (@ = "<" ou @ = "≤")
alors @' ← "<"
sinon @' ← ">"

si (@ = "≤")
alors $k'_C \leftarrow k_C + 1$
sinon si (@ = "≥")
alors $k'_C \leftarrow k_C - 1$
sinon $k'_C \leftarrow k_C$

• Il faut maintenant exprimer C sous forme normalisée. Ceci s'obtient en :

1 - ordonnant l'expression suivant $(x_1, ..., x_n)$ en faisant apparaître toutes les variables.

2 - cherchant le premier coefficient non nul dans cette expression ordonnée. Si ce coefficient est égal à +1, alors C_N est l'expression de C telle quelle.

Par contre, si ce coefficient est négatif, alors C_N est obtenu en multipliant tous les coefficients et la constante par (-1), et en changeant le sens de tous les opérateurs relationnels.

C_N, forme normalisée de C, est alors obtenue. On a :

$$C_N = (\sum_{i=1}^{n} \lambda_{i_N} x_i) \ @_N \ k_N$$

• La troisième étape est la construction de l'arbre proprement dite.

Exemple :

Construction de l'arbre associé à l'inéquation fonction des 3 variables (x_1,x_2,x_3) :

$C = -x_2 + x_3 \leq 2$

La transformation sous forme normalisée est alors :

1) $C = -x_2 + x_3 < 3$

2) $C_N = 0.x_1 + 1.x_2 + (-1).x_3 > -3$

La fonction de remplissage de l'arbre (fonction Remplir donnée dans l'Annexe 2) qui construit le chemin déterminé par les coefficients des variables de l'expression, et qui positionne la qualification de l'expression sur la feuille en fin de chemin donne alors l'arbre de la Figure 11(a).

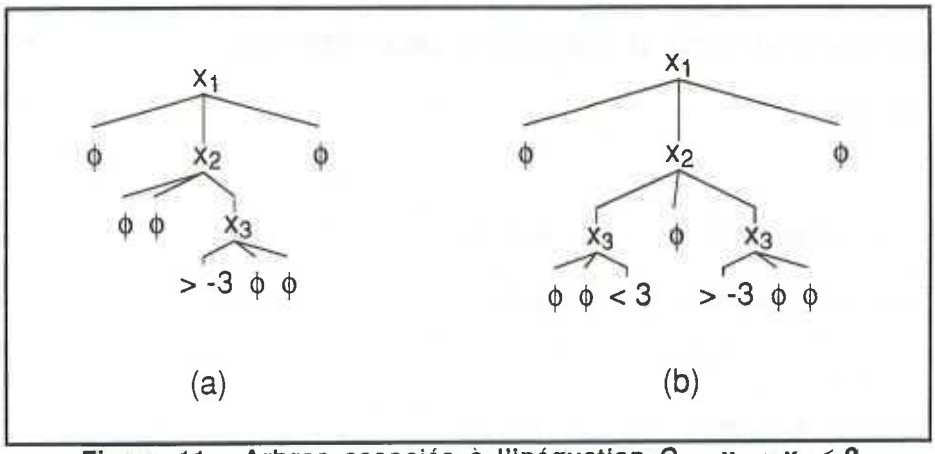


Figure 11 - Arbres associés à l'inéquation $C = x_2 + x_3 \leq 2$

La seule combinaison linéaire possible d'une seule inéquation est cette inéquation affectée d'un coefficient (-1). Le système complet est donc obtenu en rajoutant à l'arbre déjà construit, le chemin symétrique (cette opération est effectuée par la fonction Compléter, donnée dans l'Annexe 2).

Si nous reprenons le même exemple, nous obtenons l'arbre de la Figure 11(b), qui représente le système :

$$\begin{cases} 0.x_1 + 1.x_2 + (-1).x_3 > -3 \\ 0.x_1 + (-1).x_2 + 1.x_3 < 3 \end{cases}$$

5.4.2 - Combinaison d'arbres pour la représentation d'un système d'inéquations

Nous disposons de deux arbres représentant chacun une inéquation ou un système d'inéquations, en fonction du même ensemble de variables, suivant le même ordonnancement de ces variables.

La construction de l'arbre qui représente le système composé par les deux systèmes d'inéquations (représentés par les deux arbres) est fonction, localement à un nœud, des coefficients associés à la variable traitée et de la signature de l'expression construite jusqu'alors. Le principe est basé sur le fait que le coefficient associé à une variable est obtenu par addition des coefficients de cette variable dans les deux expressions considérées. Par exemple, le coefficient +1 pour la variable x_i peut être obtenu avec les combinaisons suivantes :

(1ère expression) ... + $0.x_i$ + ...
(2ème expression) ... + $1.x_i$ + ...

(1ère expression) ... + $1.x_i$ + ...
(2ème expression) ... + $0.x_i$ + ...

(1ère expression) ... + $1.x_i$ + ...
(2ème expression) ... + $1.x_i$ + ...

La qualification la plus contraignante sur l'expression obtenue est conservée.
La fonction de combinaison de deux systèmes (fonction Construct donnée dans l'Annexe 2) construit le système qui contient les inéquations des deux systèmes initiaux, ainsi que toutes les combinaisons possibles.

Exemple :
Construction d'un système d'inéquations fonction des 2 variables : (x_1,x_2)
• Première inéquation du système : $x_1 - x_2 < 6$
L'arbre associé à cette inéquation est représenté par la Figure 12(a) ; il

représente le système suivant :

$$\begin{cases} -x_1 + x_2 > -6 \\ x_1 - x_2 < 6 \end{cases}$$

• Deuxième inéquation du système : $x_1 - x_2 < 6$

L'arbre associé à cette inéquation est représenté par la Figure 12(b) ; il représente le système suivant :

$$\begin{cases} -x_1 - x_2 < -2 \\ x_1 + x_2 > 2 \end{cases}$$

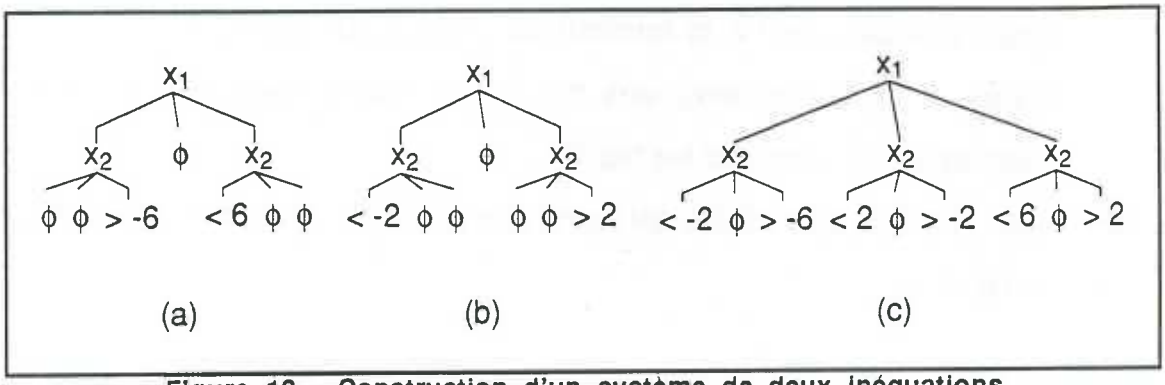


Figure 12 - Construction d'un système de deux inéquations

Le système complet est représenté par la Figure 12(c). Il représente le système suivant :

$$\begin{cases} -x_1 - x_2 < -2 \\ -x_1 + x_2 > -6 \\ -x_2 < 2 \\ x_2 > -2 \\ x_1 - x_2 < 6 \\ x_1 + x_2 > 2 \end{cases}$$

Ce système contient non seulement les deux inéquations de départ, mais également les combinaisons linéaires possibles (parmi lesquelles des contraintes sur x_2).

5.5 - Evaluation de la consistance

L'arbre résultat de l'opération de combinaison de deux systèmes représente un système d'inéquations dont la consistance n'est pas déterminée.

Le système est inconsistant si il existe une feuille y de l'arbre A correspondant, telle que A(y).I < A(y).S.

En effet, dans ce cas, l'intersection des espaces de solution des deux inéquations de signature A(y) est vide.

Après chaque appel à la fonction de construction du ET logique entre deux arbres, il est alors nécessaire de valider l'arbre résultat en comparant les qualifications portées par les feuilles.

Si le système est inconsistant, alors l'arbre est simplifié et se résume au nœud "indéfini" ϕ.

5.6 - Calcul de la variable de choix libre sur une composante connexe

Le calcul de la valeur de vérité de la variable de choix de la condition ne s'exécute que sur le système associé aux composantes connexes de la Condition de Chemin concernées par la condition.

Une fois les variables réparties dans les différents sous-ensembles, on détermine les sous-systèmes indépendants du système d'inéquations de la façon suivante : deux inéquations sont dépendantes si leurs variables appartiennent à une même composante connexe, elles sont indépendantes sinon.

Un sous-système regroupe donc les inéquations dont les variables sont

sommets d'une composante connexe. Le nombre de sous-systèmes indépendants est alors celui de composantes connexes.

Une fois le système complet qui représente la Condition de Chemin construit, le calcul de la variable de choix se résume à rechercher dans ce système l'ensemble des expressions de même signature que la condition dont nous recherchons la valeur de vérité.

Trois cas peuvent se présenter :

1er cas :

Une au moins des variables qui figurent dans la Condition C n'appartient pas à l'ensemble des variables de la Condition de Chemin CC.

Alors il n'existe pas de combinaison linéaire possible du système CC de signature égale à celle de C.

➡ Le choix de la valeur de C est libre (CL = 1).

2ème cas :

Toutes les variables qui figurent dans la Condition C appartiennent à une même composante connexe G_i.

Soit CC_i le sous-système de CC associé a G_i.

➡ Alors il suffit de calculer (CC_i ➤ C), sur la composante connexe G_i.

3ème cas :

Les variables qui figurent dans la Condition C appartiennent à plusieurs composantes connexes $(G_i)_{i=\{1, ..., m\}}$.

Soit CC_i le sous-système de CC associé a G_i.

➡ Alors il suffit de calculer $(\wedge_i CC_i) \blacktriangleright C$

En pratique, cela revient à parcourir l'arbre suivant la signature de l'expression dont on cherche la valeur de vérité. Si la recherche n'échoue pas sur une feuille "indéfinie", alors :

Si @ est l'opérateur relationnel de l'inéquation dont on recherche la valeur de vérité, k la constante de cette expression, et si il existe un chemin dans l'arbre correspondant à la signature de l'inéquation avec m la constante associée à l'opérateur @ sur la feuille de l'arbre et m' la constante associée à l'opérateur @' (@' = "<" si @ = ">" et vice-versa), alors :

- si l'expression (m @ k-1) est vraie, la variable de choix est fausse et l'inéquation est vraie.
- si l'expression (m' @ k) est vraie, la variable de choix est fausse et l'inéquation est fausse.
- la variable de choix est vraie dans les autres cas.

Notons que cette procédure ne nécessite pas la construction de l'arbre représentant la condition.

Cette procédure de calcul de la variable de choix est de complexité linéaire en fonction du nombre de variables du système construit (parcours d'un chemin de l'arbre).

Exemple :

La Condition de Chemin CC est composée de deux inéquations :

$x_1 - x_2 < 6$ et $x_1 + x_2 > 2$

Nous avons vu qu'après construction des deux arbres correspondant à chacune des inéquations, et combinaison des deux systèmes, nous obtenions l'arbre de la Figure 12(c).

• Soit C_1 la condition : (x_2 > -5)

Que pouvons-nous conclure sur la valeur de vérité de C_1 ?

Le chemin (de l'arbre qui représente CC) associé à la signature de C_1 apparaît en grisé sur la Figure 13(a).

Nous vérifions que (-2 > -5-1). Le choix n'est donc pas libre (CL = 0) et la condition C_1 est vérifiée.

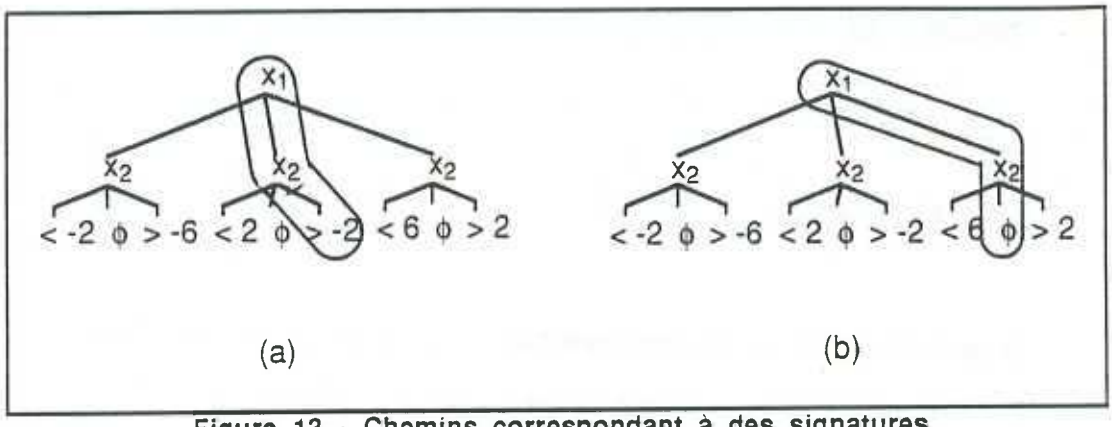


Figure 13 - Chemins correspondant à des signatures

• Soit C_2 la condition : (x_2 < -5)

Que pouvons-nous conclure sur la valeur de vérité de C_2 ?

La feuille de l'arbre qui représente l'expression de signature (0,+1) ne porte pas de qualification "<" (Figure 13(a)). Donc CC n'implique pas la valeur "vrai" pour C_2. Par contre, nous vérifions que (-2 > -5) est vrai. Donc le choix n'est pas libre (CL = 0) et la condition C_2 est impliquée en faux par CC.

• Soit C_3 la condition : (x_2 > 0)

Que pouvons-nous conclure sur la valeur de vérité de C_3 ?

Nous vérifions que (-2 > 0-1) est faux. Nous vérifions également que la feuille de l'arbre qui représente l'expression de signature (0,+1) ne porte pas de qualification "<" (Figure 13(a)). Donc le choix de la valeur de vérité de la condition C_3 est libre (CL = 1).

• Soit C_4 la condition : $(x_1 < 0)$

Que pouvons-nous conclure sur la valeur de vérité de C_4 ?

Le chemin (de l'arbre qui représente CC) associé à la signature de C_4 apparaît en grisé sur la Figure 13(b).

La feuille de l'arbre qui représente l'expression de signature (+1,0) est indéfinie.

Donc la variable de choix-libre associée à la condition C_4 est vraie (CL = 1).

• Soit C_5 la condition : $(x_3 < 2)$

CC et C_5 ne sont pas fonction des mêmes variables. D'après les résultats du chapitre 1, CL = 1.

5.7 - Mise à jour

Si la variable de choix libre associée à la condition est vraie, alors l'utilisateur est interrogé et fixe la valeur de vérité de la condition. La Condition de Chemin doit alors être mise à jour, ce qui signifie que la ou les composante(s) connexe(s) de la Condition de Chemin doivent être enrichies de la nouvelle condition et de toutes les combinaisons linéaires possibles afin d'obtenir le système complet. Il faut donc dans un premier temps construire l'arbre représentant la condition, puis effectuer le ET logique de cet arbre et de la Condition de Chemin.

Exemple :

Si nous reprenons l'exemple précédent, nous avons :

La Condition de Chemin CC est représentée par l'arbre de la Figure 12(c).

• Le choix de la condition C_3 : $(x_2 > 0)$ est libre.

Supposons que l'utilisateur indique que la condition est vraie.

Alors la Condition de Chemin est mise à jour avec l'inéquation : ($x_2 > 0$).

Nous obtenons alors la nouvelle Condition de Chemin de la Figure 14(a).

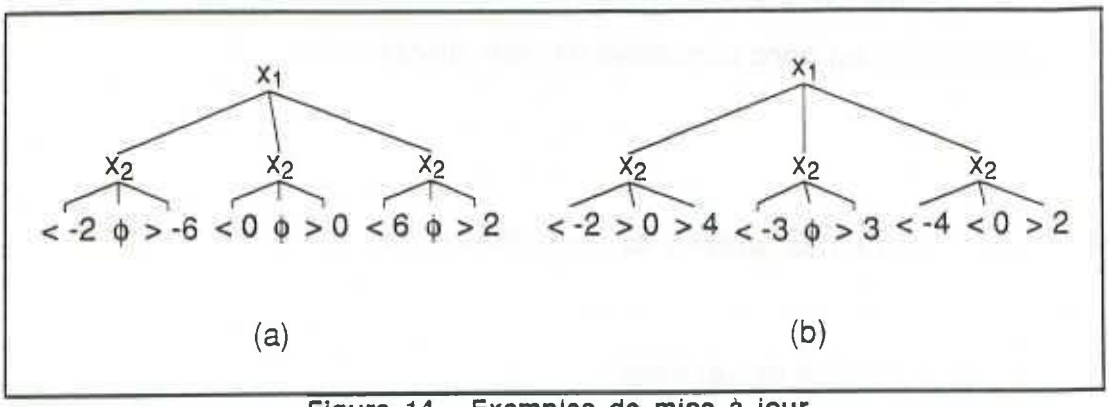


Figure 14 - Exemples de mise à jour

• Le choix de la condition C_4 : ($x_1 < 0$) est libre.

Supposons que l'utilisateur indique que la condition est vraie.

Alors la Condition de Chemin est mise à jour avec l'inéquation : ($x_1 < 0$).

Nous obtenons alors la nouvelle Condition de Chemin de la Figure 14(b).

Nous constatons sur cet exemple que le fait d'avoir précisé x_1 contraint plus fortement les autres expressions.

• Le choix de la condition C_5 : ($x_3 < 2$) est libre.

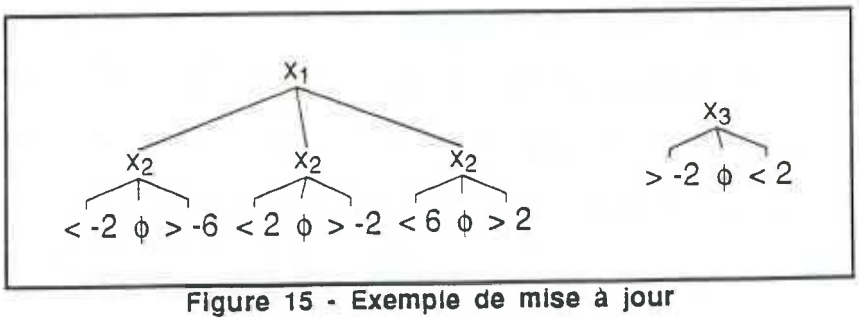


Figure 15 - Exemple de mise à jour

Supposons que l'utilisateur indique que la condition est vraie.

Alors la Condition de Chemin est mise à jour avec l'inéquation : ($x_3 < 2$).

La mise à jour de la Condition de Chemin implique l'insertion de la nouvelle variable x_3 dans le graphe de dépendance. Ce graphe contient maintenant deux composantes connexes ; une contient x_1 et x_2, et l'autre contient x_3. La Condition de Chemin est donc composée de deux arbres indépendants (Figure 15).

5.8 - Taille des arbres et complexité des algorithmes

Soit n le nombre de variables.

• La forme non réduite de l'arbre contient dans le pire des cas 3^n feuilles. L'arbre comporte donc $(3^{n+1}-1)/2$ nœuds au maximum. Remarquons toutefois qu'un arbre représentant une le système complet pour une seule inéquation au départ contient (4n - 1) feuilles, 2 seulement sont qualifiées (les autres portent l'information "indéfini"), et que l'arbre associé ne possède au plus que (6n - 2) nœuds.

• L'algorithme de construction d'un arbre à partir d'une inéquation est de complexité linéaire en fonction du nombre de variables du système.

• La construction de l'arbre résultat du ET logique de deux arbres (fonction d'un même ensemble de variables, suivant le même ordonnancement des variables) nécessite au plus $(3^n - 2^n)$ appels à la fonction de comparaison de deux feuilles (2 opérations élémentaires : comparaison des attributs "inférieur" et des attributs "supérieur"). Le nombre d'opérations élémentaires est donc de $2*(3^n - 2^n)$; l'algorithme est donc en $O(3^n)$ dans le pire des cas. Ce pire des cas correspond à la situation où les deux arbres sont "pleins", c'est à dire où toutes les combinaisons possibles des n variables sont qualifiées.

Or, les opérations de construction que nous effectuons sont toujours entre un arbre plus ou moins plein (la Condition de Chemin) et un arbre représentant une

seule inéquation, dont seulement 2 feuilles sont définies (portent une qualification). Alors en considérant que l'arbre qui représente la Condition de Chemin est plein (toutes les expressions possibles sont qualifiées), le ET logique entre la Condition de Chemin et la condition C nécessite $(3*(2^n - 1) -1)$ appels à la fonction élémentaire de comparaison de deux feuilles. Notons que ce nombre de comparaisons est constant quels que soient les coefficients de l'inéquation qui représente la condition à évaluer.
Le pire des cas théorique se ramène donc à une complexité en 2^n.

Cas particuliers :
La combinaison de deux inéquations requiert 4 comparaisons au maximum, quel que soit n.

- Le calcul de la variable de choix nécessite le parcours d'un chemin de l'arbre, et un test sur la feuille de l'arbre. La complexité est donc bien évidemment linéaire.
- La mise à jour nécessite l'appel à la fonction de construction de l'arbre résultat du ET logique de deux arbres.

5.9 - Réduction de la taille des arbres

Un arbre représentant un système complet d'inéquations est symétrique. En effet, si C est une inéquation du système de signature $(\alpha_1, ..., \alpha_n)$, alors l'inéquation $\overline{C}$, négation de C, de signature $(-\alpha_1, ..., -\alpha_n)$ appartient au système.
Il est donc possible de conserver une représentation réduite de l'arbre en ne conservant que les sous-arbres correspondant aux coefficients 0 et +1 des variables jusqu'à la rencontre d'un coefficient non nul. Cela revient à ne

conserver que la forme normalisée des inéquations.

Par exemple, l'arbre de la Figure 12(c) se réduit en l'arbre de la Figure 16.

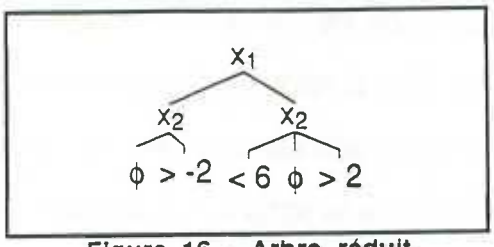


Figure 16 - Arbre réduit

De même, l'arbre de la Figure 17(a) se réduit en l'arbre de la Figure 17(b).

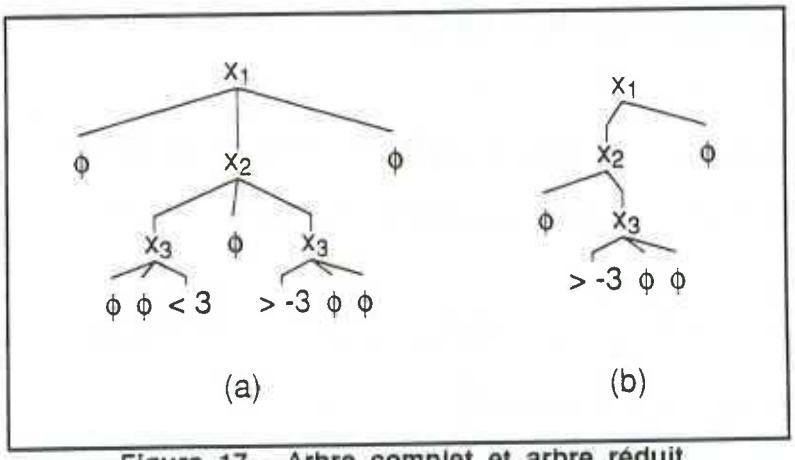


Figure 17 - Arbre complet et arbre réduit

La forme simplifiée contient au plus $2*3^{n-1}$ feuilles, soit 3^n nœuds.

Le gain de place est donc de l'ordre d'un tiers.

5.10 - Diagrammes

Une autre optimisation possible est de remplacer deux sous-arbres identiques par un seul sous-arbre et un lien vers ce sous-arbre.

Cette technique permet en particulier de ne conserver qu'un seul nœud "indéfini" ϕ. De même, si une signature est qualifiée pour l'opérateur relationnel < (resp. >), mais pas pour l'opérateur relationnel > (resp. <), alors la feuille correspondante porte la qualification "> MINVAL" (resp. "< MAXVAL"), MINVAL et MAXVAL correspondant à $-\infty$ et $+\infty$ respectivement. Il est alors possible de ne conserver qu'une seule représentation des valeurs de MINVAL et MAXVAL.
Le gain de place est alors appréciable dans le cas des arbres "creux", c'est à dire représentant peu d'inéquations (qui comportent donc un grand nombre de feuilles "indéfini", et de qualifications par rapport à MINVAL et MAXVAL).
Par exemple, dans le cas d'une seule inéquation en fonction de n variables, l'arbre est formé de (n+2) nœuds : n pour les variables, une feuille pour la qualification de l'expression et une feuille "indéfini" ϕ.

5.11 - Conclusion

Nous avons défini un mode de représentation et d'évaluation de la variable de choix sur des systèmes d'inéquations linéaires à coefficients appartenant à {-1 ; 0 ; +1} et à variables entières.
Bien que de taille exponentielle en fonction du nombre de variables, la représentation sous forme d'arbre est bien adaptée. En effet, grâce aux optimisations telles que "l'élagage de la branche gauche de l'arbre" puis la transformation sous forme de diagramme, un tiers de la place occupée est gagnée dans le pire des cas.
Notons également que la construction d'un arbre combinaison d'inéquations est coûteuse (exponentielle), mais que le calcul de la variable de choix associée à une condition se résume à suivre un chemin dans l'arbre, opération de coût

linéaire.

6 - Réalisations

Nous avons réalisé un logiciel permettant de gérer la Condition de Chemin Arithmétique à l'aide d'une structure arborescente.

Nous pouvons distinguer trois parties :

- Construction de la Condition de Chemin à partir d'une spécification,
- Calcul de la variable de choix libre associé à une condition,
- Mise à jour de la Condition de Chemin.

Dans sa première version, ce programme permet la saisie d'une spécification, puis propose un menu à l'utilisateur, qui lui offre le choix entre :

- construire la Condition de Chemin pour une date de simulation donnée,
- calculer la variable de choix pour une condition (entrée au clavier par l'utilisateur),
- afficher la Condition de Chemin courante.

Nous allons détailler les données, puis chacun de ces modules.

• La spécification

Initialement, l'utilisateur fournit :

- un ensemble de couples (inéquation linéaire , date). La date peut être numérique ou symbolique.
- un ensemble de conditions entre les dates.

• Construction de la Condition de Chemin à partir de la spécification

L'arbre correspondant à chacune des inéquations linéaires de la spécification est construit (suivant l'ordre lexicographique des constantes symboliques).

• Construction de la Condition de Chemin pour une date de simulation

L'utilisateur indique une date de simulation.

La Condition de Chemin courante est un ensemble d'arbres indépendants (un arbre par composante connexe). Le graphe de dépendance de la Condition de Chemin courante et des conditions (inéquations linéaires) valides pour cette date de simulation et toutes les dates antérieures est calculé. Les arbres de chaque nouvelle composante connexe sontt alors construits. L'ensemble d'arbres résultat de cette opération représente la nouvelle Condition de Chemin.

• Calcul de la variable de choix libre associée à une condition

L'utilisateur entre une condition (inéquation linéaire) au clavier. L'arbre correspondant à cette condition est construit.

L'ensemble de composantes connexes de la Condition de Chemin concerné par cette condition est déterminé. L'arbre résultat du ET logique de ces composantes connexes est construit ; notons le CC. C est l'arbre associé à la condition à évaluer.

Le programme parcourt le chemin de l'arbre CC qui correspond à la condition.

Si ce parcours se termine sur une feuille "indéfinie", la variable de choix-libre est vraie.

Si ce parcours se termine sur une feuille pour laquelle des contraintes sont définies, la valeur de la variable de choix-libre est définie selon la procédure explicitée dans le §4.6.

• Mise à jour de la Condition de Chemin

Dans le cas où la variable CL de choix-libre est vraie, l'utilisateur est interrogé sur la valeur de vérité de la condition à évaluer. S'il répond que cette condition est vraie, alors la Condition de Chemin devient le résultat du ET entre CC et C. S'il répond que la condition est fausse, alors la Condition de Chemin est mise à jour avec la négation de la condition C (ET logique de CC et du complément de C).

• Affichage de la Condition de Chemin

Cette fonction procède à l'affichage des arbres associés aux composantes connexes de la Condition de Chemin, en effectuant un parcours en profondeur de chacun des arbres.

Ce logiciel a été développé en langage C sur station de travail Apollo.

7 - Conclusion

Nous avons étudié dans ce chapitre les différentes méthodes qui peuvent être utilisées la gestion d'expressions mettant en jeu des inéquations linéaires à coefficients entiers.

Nous avons dans un premier temps établi que les techniques mathématiques étaient mal adaptées au problème : le domaine d'application du calcul matriciel (pour la résolution de systèmes d'équations linéaires) est le corps des réels, et l'extraction d'éventuelles solutions entières des espaces de solutions s'avère

impossible dans la plupart des cas. C'est également le cas pour la démonstration de théorèmes. Les méthodes de programmation linéaire sont également confrontées au problème de détermination de solutions entières.

Nous avons alors étudié les langages de programmation logique par contraintes. Certains de ces systèmes considèrent les entiers comme domaine d'application, et peuvent donc répondre au problème de calcul de variables de choix. Nous pouvons toutefois regretter que ces systèmes recherchent plus de résultats que nécessaire ; la conséquence est que ces outils sont trop lourds pour l'usage que nous en avons.

Nous avons donc défini une méthodologie appliquée à une forme particulière des expressions, basée sur la condition nécessaire à l'implication d'une valeur de vérité : la Condition ne peut être impliquée que si sa partie linéaire (signature) est combinaison linéaire des signatures des expressions qui composent la Condition de Chemin ; l'évaluation de la variable de choix proprement dite se limite alors à une comparaison de constantes.

Afin de pouvoir traiter des problèmes réels (plusieurs dizaines de variables), nous appliquons notre stratégie de mise en œuvre de la méthode de calcul de la variable de choix sur les seuls sous-ensembles d'inéquations (composantes connexes) de la Condition de Chemin susceptibles de déterminer la valeur de vérité de la condition à évaluer, et non pas sur la Condition de Chemin dans son intégralité.

Chapitre 5 - Gestion d'expressions complexes

Les deux chapitres précédents ont été consacrés à la gestion de la Condition de Chemin dans les cas suivants :

- la Condition de Chemin et la condition à évaluer sont des fonctions booléennes,
- la Condition de Chemin CC est un système d'inéquations linéaires à coefficients entiers et variables à valeurs entières ; la condition C à évaluer est une inéquation linéaire à coefficients entiers et variables à valeurs entières.

Nous étudions dans ce chapitre les cas restant à traiter, qui peuvent être scindés en deux classes :

- les expressions arithmétiques dans lesquelles CC et C présentent une autre configuration que celle prévue dans le chapitre 4,
- les expressions qui mettent en jeu à la fois des constantes symboliques booléennes et des constantes symboliques arithmétiques.

1 - Préliminaires

Nous avons étudié les cas simples de distribution des opérateurs logiques par

rapport à l'opérateur de choix (Cf chapitre 2 §6.3). Nous nous intéressons maintenant à des expressions plus complexes afin de parvenir à traiter le cas général.

Après un rappel des propriétés de l'opérateur de choix libre, nous revenons sur la combinaison des variables de choix libre dans le cas d'expressions complexes.

1.1 - Distribution de l'opérateur de choix

Nous rappelons que les propriétés de distribution de l'opérateur de choix libre par rapport aux OU et ET logiques, établies dans le Chapitre 2 (§6.3), sont les suivantes :

CC_1 et CC_2 sont des expressions logiques définissant des Conditions de Chemins, C est une condition à évaluer, $CL_1 = (CC_1 \blacktriangleright C)$ et $CL_2 = (CC_2 \blacktriangleright C)$. Nous avons alors :

$(CC_1 \wedge CC_2 \blacktriangleright C) \Leftrightarrow (CL_1 \wedge CL_2)$

$(CC_1 \vee CC_2 \blacktriangleright C) \Leftrightarrow (CL_1 \vee CL_2)$

CC est une expression logique définissant une Condition de Chemin, C_1 et C_2 sont des conditions à évaluer, $CL_1 = (CC \blacktriangleright C_1)$ et $CL_2 = (CC \blacktriangleright C_2)$. Nous avons alors :

$(CC \blacktriangleright C_1 \wedge C_2) \Leftrightarrow (CL_1 \vee CL_2)$

$(CC \blacktriangleright C_1 \vee C_2) \Leftrightarrow (CL_1 \wedge CL_2)$

Partant toujours de l'hypothèse selon laquelle nous connaissons la variable de choix de $(CC_i \blacktriangleright C_j)$, nous abordons le problème de la distribution de l'opérateur de choix quand les opérandes sont toutes deux des expressions complexes

(conjonction ou disjonction d'expressions).

Nous étudions les quatre cas de base pour lesquels les opérandes sont la conjonction ou la disjonction de deux termes.

Notation : $CL_{i,j} = (CC_i \rightarrow C_j)$.

• Calcul de $(CC_1 \wedge CC_2) \rightarrow (C_1 \wedge C_2)$.

Le choix n'est pas libre si une des situations suivantes est vérifiée :

• CC_1 permet de déterminer la valeur de C_1 et la valeur de C_2 : $CL_{1,1}$ et $CL_{1,2}$ sont fausses,

• CC_1 permet de déterminer la valeur de C_1 et CC_2 la valeur de C_2 : $CL_{1,1}$ et $CL_{2,2}$ sont fausses,

• CC_1 permet de déterminer la valeur de C_2 et CC_2 la valeur de C_1 : $CL_{1,2}$ et $CL_{2,1}$ sont fausses,

• CC_2 permet de déterminer la valeur de C_1 et la valeur de C_2 : $CL_{2,1}$ et $CL_{2,2}$ sont fausses.

Le choix est libre dans les autres cas ; nous pouvons alors écrire l'équation du complément de CL :

$$\overline{CL} = (\overline{CL_{1,1}} \wedge \overline{CL_{1,2}}) \vee (\overline{CL_{1,1}} \wedge \overline{CL_{2,2}}) \vee (\overline{CL_{2,1}} \wedge \overline{CL_{1,2}}) \vee (\overline{CL_{2,1}} \wedge \overline{CL_{2,2}})$$

d'où $CL = (CC_1 \wedge CC_2) \rightarrow (C_1 \wedge C_2) =$

$$\overline{(\overline{CL_{1,1}} \wedge \overline{CL_{1,2}}) \vee (\overline{CL_{1,1}} \wedge \overline{CL_{2,2}}) \vee (\overline{CL_{2,1}} \wedge \overline{CL_{1,2}}) \vee (\overline{CL_{2,1}} \wedge \overline{CL_{2,2}})} =$$

$$(CL_{1,1} \vee CL_{1,2}) \wedge (CL_{1,1} \vee CL_{2,2}) \wedge (CL_{2,1} \vee CL_{1,2}) \wedge (CL_{2,1} \vee CL_{2,2}) =$$

$$(CL_{1,1} \wedge CL_{2,1}) \vee (CL_{1,2} \wedge CL_{2,2})$$

• Calcul de $(CC_1 \wedge CC_2) \rightarrow (C_1 \vee C_2)$

Dans ce cas, le choix n'est pas libre si CC_1 permet de déterminer la valeur de C_1 ($CL_{1,1}$ est fausse), ou si CC_2 permet de déterminer la valeur de C_1 ($CL_{2,1}$ est

fausse), ou si CC_1 permet de déterminer la valeur de C_2 ($CL_{1,2}$ est fausse) ou finalement si CC_2 permet de déterminer la valeur de C_2 ($CL_{2,2}$ est fausse).

Nous avons donc :

$\overline{CL} = (\overline{CL_{1,1}} \vee \overline{CL_{2,1}} \vee \overline{CL_{1,2}} \vee \overline{CL_{2,2}})$

d'où $CL = (CC_1 \wedge CC_2) \rightarrow (C_1 \vee C_2) = \overline{(\overline{CL_{1,1}} \vee \overline{CL_{2,1}} \vee \overline{CL_{1,2}} \vee \overline{CL_{2,2}})} =$

$(CL_{1,1} \wedge CL_{1,2} \wedge CL_{2,1} \wedge CL_{2,2})$

• Calcul de $(CC_1 \vee CC_2) \rightarrow (C_1 \wedge C_2)$

Dans ce cas, le choix n'est pas libre si CC_1 et CC_2 permettent de déterminer la valeur de C_1 et de C_2 ($CL_{1,1}$ et $CL_{1,2}$ et $CL_{2,1}$ et $CL_{2,2}$ sont fausses).

Nous avons donc :

$\overline{CL} = (\overline{CL_{1,1}} \wedge \overline{CL_{1,2}} \wedge \overline{CL_{2,1}} \wedge \overline{CL_{2,2}})$

d'où $CL = (CC_1 \vee CC_2) \rightarrow (C_1 \wedge C_2) = \overline{(\overline{CL_{1,1}} \wedge \overline{CL_{1,2}} \wedge \overline{CL_{2,1}} \wedge \overline{CL_{2,2}})} =$

$(CL_{1,1} \vee CL_{1,2} \vee CL_{2,1} \vee CL_{2,2})$

• Calcul de $(CC_1 \vee CC_2) \rightarrow (C_1 \vee C_2)$.

Le choix n'est pas libre si une des situations suivantes est vérifiée :

• CC_1 permet de déterminer la valeur de C_1 et CC_2 la valeur de C_1 : $CL_{1,1}$ et $CL_{2,1}$ sont fausses,

• CC_1 permet de déterminer la valeur de C_1 et CC_2 la valeur de C_2 : $CL_{1,1}$ et $CL_{2,2}$ sont fausses,

• CC_1 permet de déterminer la valeur de C_2 et CC_2 la valeur de C_1 : $CL_{1,2}$ et $CL_{2,1}$ sont fausses,

• CC_1 permet de déterminer la valeur de C_2 et CC_2 la valeur de C_2 : $CL_{1,2}$ et $CL_{2,2}$ sont fausses.

Le choix est libre dans les autres cas ; nous pouvons alors écrire l'équation du complément de CL :

$\overline{CL} = (\overline{CL_{1,1}} \wedge \overline{CL_{2,1}}) \vee (\overline{CL_{1,1}} \wedge \overline{CL_{2,2}}) \vee (\overline{CL_{2,1}} \wedge \overline{CL_{1,2}}) \vee (\overline{CL_{1,2}} \ \overline{CL_{2,2}})$

d'où $CL = (CC_1 \vee CC_2) \rightarrow (C_1 \vee C_2) =$

$\overline{(\overline{CL_{1,1}} \wedge \overline{CL_{2,1}}) \vee (\overline{CL_{1,1}} \wedge \overline{CL_{2,2}}) \vee (\overline{CL_{2,1}} \wedge \overline{CL_{1,2}}) \vee (\overline{CL_{1,2}} \ \overline{CL_{2,2}})} =$

$(CL_{1,1} \vee CL_{2,1}) \wedge (CL_{1,1} \vee CL_{2,2}) \wedge (CL_{2,1} \vee CL_{1,2}) \wedge (CL_{1,2} \vee CL_{2,2}) =$

$(CL_{1,1} \wedge CL_{1,2}) \vee (CL_{2,1} \wedge CL_{2,2})$

1.2 - Calcul de la variable de choix libre sur deux expressions complexes

Nous étudions dans ce paragraphe une méthode de détermination de la variable de choix-libre dans le cas d'expressions complexes composées de conjonctions et disjonctions de termes (qui eux-mêmes peuvent être des expressions complexes).

Soit $(CC_i)_{i \in I}$ et $(C_j)_{j \in J}$ deux ensembles d'expressions booléennes.

Soit $I' \subset I$ et $J' \subset J$.

Hypothèse :

$\forall (i,j) \in I \times J$, nous supposons que la valeur de la variable de choix-libre $CL_{i,j}$ (avec $CL_{i,j} = (CC_i \rightarrow C_j)$) est connue (peut être calculée).

En généralisant les résultats du paragraphe précédent, nous déterminons comment la variable de choix d'une expression complexe est obtenue en fonction des variables de choix $CL_{i,j}$ "élémentaires". Nous pouvons donc établir :

1)

$$\bigwedge_{i \in I'} CC_i \rightarrow \bigwedge_{j \in J'} C_j \quad \Leftrightarrow \quad \bigvee_{j \in J'} \bigwedge_{i \in I'} CL_{i,j}$$

2)

$$\bigwedge_{i \in I'} CC_i \rightarrow \bigvee_{j \in J'} C_j \quad \Leftrightarrow \quad \bigwedge_{\substack{i \in I' \\ j \in J'}} CL_{i,j}$$

3)

$$\bigvee_{i \in I'} CC_i \rightarrow \bigwedge_{j \in J'} C_j \quad \Leftrightarrow \quad \bigvee_{\substack{i \in I' \\ j \in J'}} CL_{i,j}$$

4)

$$\bigvee_{i \in I'} CC_i \rightarrow \bigvee_{j \in J'} C_j \quad \Leftrightarrow \quad \bigvee_{i \in I'} \bigwedge_{j \in J'} CL_{i,j}$$

Ces manipulations permettent de déterminer la valeur de la variable de choix d'expressions complexes, connaissant la valeur de la variable de choix de deux expressions booléennes données.

Nous devons maintenant appliquer ces résultats pour la gestion d'expressions arithmétiques complexes et dans le cas d'expressions mixtes.

2 - Calcul de la variable de choix pour des expressions arithmétiques

2.1 - Remarque

La méthode (proposée dans le chapitre 4) pour l'évaluation de la variable de

choix-libre s'applique sur une Condition de Chemin CC ayant la forme d'une conjonction d'inéquations linéaires.

Pour simplifier l'écriture du système, nous codons chacune de ces inéquations linéaires par une variable booléenne. Nous pouvons alors écrire :

$$CC = \bigwedge_{k \in K} IL_k$$

A partir de cette conjonction d'inéquations linéaires, le système de résolution construit un ensemble de combinaisons linéaires, **incluses dans CC.**

L'expression du **système complet** est donc :

$$CC = (\bigwedge_{k \in K} IL_k) \wedge (\bigwedge_{l \in L} LC_l)$$

Supposons par exemple que CC soit la conjonction de deux conditions.

Nous avons alors :

$$CC = CC_1 \wedge CC_2$$

avec :

$$CC_1 = (\bigwedge_{k_1 \in K_1} IL_{k_1}) \wedge (\bigwedge_{l_1 \in L_1} LC_{l_1})$$

et

$$CC_2 = (\bigwedge_{k_2 \in K_2} IL_{k_2}) \wedge (\bigwedge_{l_2 \in L_2} LC_{l_2})$$

Alors l'expression du système complet qui représente CC est la suivante :

$$CC = (\bigwedge_{k_1 \in K_1} IL_{k_1}) \wedge (\bigwedge_{l_1 \in L_1} LC_{l_1}) \wedge (\bigwedge_{k_2 \in K_2} IL_{k_2}) \wedge (\bigwedge_{l_2 \in L_2} LC_{l_2}) \wedge (\bigwedge_{l \in L} \mathbf{LC_l})$$

Les $(LC_l)_{l \in L}$ n'appartiennent ni à CC_1, ni à CC_2. Effectuer la conjonction de CC_1 et CC_2 produit donc de nouvelles contraintes.

Dans le cas d'expressions arithmétiques, l'ensemble des inéquations de deux systèmes est strictement inclus dans l'ensemble des inéquations du système résultat du ET logique de ces deux systèmes. Cela modifie quelque peu les propriétés de distribution de l'opérateur de choix-libre dans le cadre

d'expressions complexes.

2.2 - Calcul de la variable de choix libre

Etant donné le codage adopté, qui transforme une expression logique d'inéquations linéaires en expression purement booléenne, nous pouvons envisager d'utiliser les résultats du §1.

- CC_i est une conjonction de variables booléennes (qui représentent chacune une inéquation)
- C_j est une variable booléenne (qui représente une inéquation).
- Le résultat de $(CC_i \rightarrow C_j)$ est connu.

Compte tenu de la remarque précédente (§2.1), certaines équivalences du §1 ne sont plus vérifiées ; nous pouvons tout de même établir des implications. En résumé, nous avons alors :

1)

$$\bigwedge_{i \in I'} CC_i \rightarrow \bigwedge_{j \in J'} C_j \quad \Leftarrow \quad \bigvee_{j \in J'} \bigwedge_{i \in I'} CL_{i,j}$$

2)

$$\bigwedge_{i \in I'} CC_i \rightarrow \bigvee_{j \in J'} C_j \quad \Leftarrow \quad \bigwedge_{\substack{i \in I' \\ j \in J'}} CL_{i,j}$$

3)

$$\bigvee_{i \in I'} CC_i \rightarrow \bigwedge_{j \in J'} C_j \quad \Leftrightarrow \quad \bigvee_{\substack{i \in I' \\ j \in J'}} CL_{i,j}$$

4)

$$\bigvee_{i \in I'} CC_i \rightarrow \bigvee_{j \in J'} C_j \quad \Leftrightarrow \quad \bigvee_{i \in I'} \bigwedge_{j \in J'} CL_{i,j}$$

Nous constatons que les cas 1) et 2) ne permettent pas toujours de conclure.

Le problème vient du fait que la Condition de Chemin est une conjonction de différentes Conditions de Chemins.

En effet, le ET logique de systèmes complets définit le système composé de tous les systèmes complets de départ, enrichi de nouvelles contraintes combinaisons linéaires d'inéquations issues des différents systèmes (Cf §2.1).

L'expression finale obtenue par conjonction de différentes Conditions de Chemin (elles-mêmes conjonctions de variables booléennes) se ramène alors à une conjonction de variables logiques (associativité du ET). Si nous notons CC cette expression, et CL_j la variable de choix associée à $(CC \rightarrow C_j)$, alors nous pouvons établir les équivalences :

1')

$$CC \rightarrow \bigwedge_{j \in J'} C_j \quad \Leftrightarrow \quad \bigvee_{j \in J'} CL_j$$

2')

$$CC \rightarrow \bigvee_{j \in J'} C_j \quad \Leftrightarrow \quad \bigwedge_{j \in J'} CL_j$$

Remarque :

En pratique, le calcul de la variable de choix sera tout d'abord effectué selon 1) (resp. 2)). Seulement dans le cas où aucune conclusion n'aura pû être tirée (la

variable de choix libre vaut 1), le calcul selon 1') (resp. 2')), qui nécessite l'évaluation de tous les ($CC \rightarrow C_j$), sera effectué.

2.3 - Calcul de la variable de choix libre pour des expressions arithmétiques quelconques

Nous avons défini comment évaluer la variable de choix libre pour des cas particuliers de Conditions de Chemin et de Conditions à évaluer.
Dans le cas d'expressions quelconques, cette méthode de calcul peut être appliquée de façon itérative : nous allons le voir sur un exemple.

Exemple :

Nous recherchons la valeur de $CL = CC \rightarrow C$, avec :

$CC = (x+y < 3) \wedge [(y > 2) \vee (z < 0)]$

$C = (x < 4) \vee (z < 3)$

• Codage des expressions :

$(x+y < 3) \leftarrow a$

$(y > 2) \leftarrow b$

$(z < 0) \leftarrow c$

$(x < 4) \leftarrow d$

$(z < 3) \leftarrow e$

d'où

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

$C = d \vee e$

• Calcul de la variable de choix CL :

$CL = a \wedge (b \vee c) \blacktriangleright d \vee e$

Notons : $CL_{x,y}$ la variable de choix associée à $(x \blacktriangleright y)$

Nous avons alors :

$a \wedge (b \vee c) \blacktriangleright d \vee e \quad \Leftarrow \quad (CL_{a,d} \wedge CL_{bvc,d} \wedge CL_{a,e} \wedge CL_{bvc,e})$

avec :

$CL_{bvc,d} = CL_{b,d} \vee CL_{c,d}$

$CL_{bvc,e} = CL_{b,e} \vee CL_{c,e}$

d'où :

$a \wedge (b \vee c) \blacktriangleright d \vee e \Leftarrow (CL_{a,d} \wedge (CL_{b,d} \vee CL_{c,d}) \wedge CL_{a,e} \wedge (CL_{b,e} \vee CL_{c,e}))$

• Calcul de la variable de choix-libre :

Les variables de choix libre élémentaires sont évaluées suivant la méthode présentée dans le chapitre 4. Nous avons alors :

- $CL_{a,d} = 1$
- $CL_{b,d} = 1$
- $CL_{c,d} = 1$
- $CL_{a,e} = 1$
- $CL_{b,e} = 1$
- $CL_{c,e} = 0$

Alors $CL = (CL_{a,d} \wedge (CL_{b,d} \vee CL_{c,d}) \wedge CL_{a,e} \wedge (CL_{b,e} \vee CL_{c,e})) =$

$1 \wedge [1 \vee 1] \wedge 1 \wedge [1 \vee 0] = 1$

Le choix est libre. Or, les variables manipulées représentent des expressions arithmétiques. Ne pas distribuer les conjonctions par rapport aux disjonctions entraîne une perte d'informations (Cf. §2.2).

En effet,

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

Puisque nous n'avons pas pu conclure précédemment, nous évaluons maintenant la variable de choix en développant la Condition de Chemin. Cela donne :

$a \wedge b \vee a \wedge c \rightarrow d \vee e$

$\Leftrightarrow \quad (CL_{a \wedge b, d \vee e} \vee CL_{a \wedge c, d \vee e})$

$\Leftrightarrow \quad (CL_{a \wedge b, d} \wedge CL_{a \wedge b, e} \vee CL_{a \wedge c, d} \wedge CL_{a \wedge c, e})$

$\Leftarrow \quad (CL_{a,d} \wedge CL_{b,d}) \wedge (CL_{a,e} \wedge CL_{b,e}) \vee (CL_{a,d} \wedge CL_{c,d}) \wedge (CL_{a,e} \wedge CL_{c,e})$

Les variables de choix libre élémentaires sont évaluées suivant la méthode présentée dans le chapitre 4. Nous avons alors :

- $CL_{a,d} = 1$
- $CL_{b,d} = 1$
- $CL_{a,e} = 1$
- $CL_{b,e} = 1$
- $CL_{c,d} = 1$
- $CL_{c,e} = 0$

Alors $CL = (CL_{a,d} \wedge CL_{b,d}) \wedge (CL_{a,e} \wedge CL_{b,e}) \vee (CL_{a,d} \wedge CL_{c,d}) \wedge (CL_{a,e} \wedge CL_{c,e}) =$

$(1 \wedge 1) \wedge (1 \wedge 1) \vee (1 \wedge 1) \wedge (1 \wedge 0) = 1 \vee 0 = 1$

La variable de choix libre calculée par cette méthode est vraie. Il est donc nécessaire d'opérer les calculs en prenant en compte la conjonction des éléments qui composent la Condition de Chemin (Cf. §2.2) ; la variable de choix vaut alors :

$a \wedge b \vee a \wedge c \rightarrow d \vee e \quad \Leftrightarrow \quad (CL_{a \wedge b, d} \wedge CL_{a \wedge b, e} \vee CL_{a \wedge c, d} \wedge CL_{a \wedge c, e})$

En évaluant les variables de choix par la méthode présentée dans le chapitre 4,

nous obtenons :

- $CL_{a \wedge b,d} = 0$
- $CL_{a \wedge b,e} = 1$
- $CL_{a \wedge c,d} = 1$
- $CL_{a \wedge c,e} = 0$

Finalement :

$CL = (CL_{a \wedge b,d} \wedge CL_{a \wedge b,e} \vee CL_{a \wedge c,d} \wedge CL_{a \wedge c,e}) = 0 \wedge 1 \vee 1 \wedge 0 = 0$

Le choix de la valeur de vérité de la condition C n'est pas libre.

2.4 - Conclusion

Compte tenu de la particularité des variables booléennes manipulées - elles représentent des inéquations linéaires -, nous avons défini une méthode de calcul de la variable de choix libre pour différentes configurations pour la Condition de Chemin et pour la Condition à évaluer.

En raison de la complexité exponentielle (en nombre de variables) du calcul de la variable de choix présenté dans le chapitre 4, celui-ci est effectué dans la mesure du possible sur des termes les plus petits possibles.

Dans le cas d'expressions quelconques, cette méthode s'applique itérativement à partir d'une forme "somme de produits".

3 - Calcul de la variable de choix pour des expressions mixtes

Les méthodes de calcul de variables de choix pour des expressions purement booléennes, et pour des conditions ne faisant intervenir que des constantes symboliques arithmétiques sont définies. Nous étudions maintenant le cas des expressions mixtes.

3.1 - Définition

Une expression conditionnelle mixte est une condition qui met en jeu à la fois des constantes symboliques booléennes directement issues du programme VHDL (qui forment les expressions qui représentent dans le programme la valeur de variables ou signaux de type BIT ou BOOLEAN), et de variables booléennes qui codent des expressions arithmétiques.

3.2 - Préliminaires

Soit CC_1 et CC_2 deux conditions booléennes dont les ensembles de définition sont disjoints.
Considérons une condition booléenne C_1 (resp. C_2) définie sur le même ensemble que CC_1 (resp. CC_2).

Les ensembles de définition de CC_1 et CC_2 étant disjoints, nous avons alors :

$CC_1 \blacktriangleright C_2 = CC_1 \blacktriangleright C_2 = 1$

3.3 - Calcul de la variable de choix libre

Nous appliquons les propriétés de distribution de l'opérateur de choix (§2) pour calculer la variable de choix d'une expression mixte, en tenant compte des particularités de ces expressions.

Considérons que CC_1 et C_1 sont des expressions booléennes dont les variables représentent des expressions arithmétiques (inéquations linéaires), et que CC_2 et C_2 ne manipulent que des constantes symboliques booléennes.

Les domaines de définition de (CC_1 et C_1) et de (CC_2 et C_2) sont bien évidemment disjoints. Les résultats du §2 peuvent donc s'appliquer de la façon suivante :

1)

$$
\begin{aligned}
(CC_1 \wedge CC_2) \blacktriangleright (C_1 \wedge C_2) &\Leftrightarrow (CL_{1,1} \vee CL_{1,2}) \wedge (CL_{1,1} \vee CL_{2,2}) \wedge \\
&\quad (CL_{2,1} \vee CL_{1,2}) \wedge (CL_{2,1} \vee CL_{2,2}) \\
&\Leftrightarrow (CL_{1,1} \vee 1) \wedge (CL_{1,1} \vee CL_{2,2}) \wedge \\
&\quad (1 \vee 1) \wedge (1 \vee CL_{2,2}) \\
&\Leftrightarrow 1 \wedge (CL_{1,1} \vee CL_{2,2}) \wedge 1 \wedge 1 \\
&\Leftrightarrow (CL_{1,1} \vee CL_{2,2})
\end{aligned}
$$

2)

$$
\begin{aligned}
(CC_1 \wedge CC_2) \blacktriangleright (C_1 \vee C_2) &\Leftrightarrow (CL_{1,1} \wedge CL_{1,2} \wedge CL_{2,1} \wedge CL_{2,2}) \\
&\Leftrightarrow (CL_{1,1} \wedge 1 \wedge 1 \wedge CL_{2,2}) \\
&\Leftrightarrow (CL_{1,1} \wedge CL_{2,2})
\end{aligned}
$$

3)

$$
\begin{aligned}
(CC_1 \vee CC_2) \blacktriangleright (C_1 \wedge C_2) &\Leftrightarrow (CL_{1,1} \vee CL_{1,2} \vee CL_{2,1} \vee CL_{2,2}) \\
&\Leftrightarrow (CL_{1,1} \vee 1 \vee 1 \vee CL_{2,2})
\end{aligned}
$$

$\Leftrightarrow$ 1

4)

$$(CC_1 \vee CC_2) \blacktriangleright (C_1 \vee C_2) \Leftrightarrow (CL_{1,1} \vee CL_{2,1}) \wedge (CL_{1,1} \vee CL_{2,2}) \wedge (CL_{2,1} \vee CL_{1,2}) \wedge (CL_{1,2} \vee CL_{2,2})$$

$$\Leftrightarrow (CL_{1,1} \vee 1) \wedge (CL_{1,1} \vee CL_{2,2}) \wedge (1 \vee 1) \wedge (1 \vee CL_{2,2})$$

$$\Leftrightarrow 1 \wedge (CL_{1,1} \vee CL_{2,2}) \wedge 1 \wedge 1$$

$$\Leftrightarrow (CL_{1,1} \vee CL_{2,2})$$

Nous pouvons alors retenir que :

- si la Condition de Chemin (resp. la condition à évaluer) est formée d'une disjonction (resp. conjonction) d'une expression de constantes symboliques booléennes et d'une expression de variables booléennes qui représentent des inéquations linéaires, alors le choix est toujours libre.
- si la Condition de Chemin et la condition à évaluer sont toutes deux formées d'une disjonction (resp. conjonction) d'une expression de constantes symboliques booléennes et d'une expression de variables booléennes qui représentent des inéquations linéaires, alors la variable de choix-libre prend la valeur du OU des deux variables de choix $CL_{1,1}$ et $CL_{1,2}$.
- si la Condition de Chemin (resp. la condition à évaluer) est formée d'une conjonction (resp. disjonction) d'une expression de constantes symboliques booléennes et d'une expression de variables booléennes qui représentent des inéquations linéaires, alors la variable de choix-libre prend la valeur du ET des deux variables de choix $CL_{1,1}$ et $CL_{1,2}$.

Nous allons illustrer sur un exemple la mise en œuvre du calcul de la variable de choix-libre sur des expressions mixtes. Par souci de lisibilité, les inéquations

linéaires sont codées par des variables booléennes écrites en majuscules ; rappelons que les éléments booléens écrits en minuscule dans les expressions de départ sont des constantes symboliques booléennes.

Exemple :

$CC = [(x+y < 3) \vee (z > 0)] \wedge [(a \vee b) \wedge c]$

$C = (x > 0) \vee c$

• Codage des expressions :

$(x+y < 3) \leftarrow A$

$(z > 0) \leftarrow B$

$(x > 0) \leftarrow D$

• Expression booléenne de CC et C :

$CC = (A \vee B) \wedge [(a \vee b) \wedge c]$

$C = D \vee c$

Remarque : A, B et D sont des variables booléennes qui représentent des expressions arithmétiques ; a, b et c sont des constantes symboliques booléennes associées aux variables booléennes du programme VHDL.

• Calcul de $(CC \rightarrow C)$:

$(A \vee B) \wedge [(a \vee b) \wedge c] \rightarrow D \vee c$

$\Leftrightarrow \quad CL_{A \vee B,D} \wedge CL_{A \vee B,c} \wedge CL_{(a \vee b) \wedge c,D} \wedge CL_{(a \vee b) \wedge c,c}$

$\Leftrightarrow \quad CL_{A \vee B,D} \wedge 1 \wedge 1 \wedge CL_{(a \vee b) \wedge c,c}$

$\Leftrightarrow \quad CL_{A \vee B,D} \wedge CL_{(a \vee b) \wedge c,c}$

La variable de choix $CL_{A \vee B,D}$ est évaluée par la méthode présentée dans le

chapitre 4.

Nous obtenons : $CL_{A \vee B,D} = 1$.

De même, la variable de choix $CL_{(a \vee b) \wedge c,c}$ est calculée selon la méthode relative aux expressions booléennes présentée dans le chapitre 3. Finalement, $CL_{(a \vee b) \wedge c,c} = 0$.

Donc CC ➤ C = $CL_{A \vee B,D} \wedge CL_{(a \vee b) \wedge c,c} = 1 \wedge 0 = 0$.

En conclusion, le choix de la valeur de vérité de la condition C n'est pas libre.

3.4 - Calcul de la variable de choix pour des expressions mixtes quelconques

La méthode de calcul de la variable de choix pour des expressions mixtes est définie pour des cas particuliers de la Condition de Chemin et de la Condition à évaluer.

Comme pour les expressions arithmétiques, dans le cas d'expressions mixtes complexes (qui ne peuvent pas être exprimées sous la forme d'une disjonction ou conjonction de deux conditions distinctes - booléenne et arithmétique -), il est possible de procéder de façon itérative.

Exemple :

CC = $(x+y < 3) \wedge [(y > 2) \vee a]$

C = $(x < 4) \vee a$

Quelle est la valeur de CL = CC ➤ C ?

• Codage des expressions :

$(x+y < 3) \leftarrow A$

$(y > 2) \leftarrow B$

$(x < 4) \leftarrow D$

• Expression booléenne de CC et C :

$CC = A \wedge (B \vee a) = A \wedge B \vee A \wedge a$

$C = D \vee a$

• Préliminaires :

$A \wedge B \rightarrow a = 1$

$$A \wedge a \rightarrow D \Leftrightarrow CL_{A,D} \wedge CL_{a,D}$$
$$\Leftrightarrow CL_{A,D} \wedge 1$$
$$\Leftrightarrow CL_{A,D}$$

$$A \wedge a \rightarrow a \Leftrightarrow CL_{A,a} \wedge CL_{a,a}$$
$$\Leftrightarrow 1 \wedge CL_{a,a}$$
$$\Leftrightarrow CL_{a,a}$$

• Calcul de $(CC \rightarrow C)$:

$$A \wedge B \vee A \wedge a \rightarrow D \vee a \Leftrightarrow (CL_{A \wedge B,D} \vee CL_{A \wedge a,D}) \wedge (CL_{A \wedge B,D} \vee CL_{A \wedge a,a}) \wedge (CL_{A \wedge a,D} \vee CL_{A \wedge B,a}) \wedge (CL_{A \wedge B,a} \vee CL_{A \wedge a,a})$$
$$\Leftrightarrow (CL_{A \wedge B,D} \vee CL_{A,D}) \wedge (CL_{A \wedge B,D} \vee CL_{a,a}) \wedge (CL_{A,D} \vee 1) \wedge (1 \vee CL_{a,a})$$
$$\Leftrightarrow (CL_{A \wedge B,D} \vee CL_{A,D}) \wedge (CL_{A \wedge B,D} \vee CL_{a,a})$$

Les variables de choix $CL_{A \wedge B,D}$ et $CL_{A,D}$ sont évaluées selon la méthode du chapitre 4. Elles sont respectivement égales à 0 et à 1.

De même, la variable de choix associée à $CL_{a,a}$, évaluée selon la méthode du chapitre 3 vaut 0.

Finalement :

$(CL_{A\wedge B,D} \vee CL_{A,D}) \wedge (CL_{A\wedge B,D} \vee CL_{a,a}) = (0 \vee 1) \wedge (0 \vee 0) = 0$

Le choix de la valeur de vérité de la condition C n'est donc pas libre.

3.5 - Conclusion

Nous avons défini une méthode d'évaluation de la variable de choix libre pour des expressions mixtes particulières ; nous considérons en effet seulement les expressions mixtes qu'il est possible d'exprimer comme conjonction ou disjonction d'une expression "arithmétique", et d'une expression purement booléenne.

Cette méthode peut aisément s'appliquer sur des expressions plus complexes de façon itérative. Il suffit d'exprimer les conditions sous forme de "somme de produits" dans les cas où des variables booléennes qui représentent des conditions arithmétiques interviennent (Cf. §2.3).

Finalement, toute expression conditionnelle mixte peut être traitée.

Les résultats du calcul de la variable de choix sont fonction des variables de choix pour la partie arithmétique et pour la partie booléenne.

4 - Conclusion

Les méthodologies de gestion de la Condition de Chemin, et en particulier de calcul de la variable de choix-libre associée à une condition, sont définies. Dans

ce chapitre, nous avons étudié comment utiliser ces méthodes d'évaluation de la variable de choix pour des expressions complexes exprimées sous des formes différentes que celles prévues par ces méthodes.

La valeur de la variable de choix est alors une fonction logique de variables de choix calculées dans le cadre de la gestion d'expressions booléennes et de la gestion d'expressions arithmétiques telles que définies dans le chapitre 4.

Conclusion générale

Nous avons défini et développé une méthode de gestion de la Condition de Chemin dans le cadre d'expressions logiques dont les éléments de base sont des constantes symboliques booléennes et des inéquations linéaires de constantes symboliques arithmétiques entières.

Pour cela, nous avons défini le concept de variable de choix-libre associée à une condition. La valeur de cette variable de choix permet de différencier deux cas : soit la Condition de Chemin impose une valeur de vérité pour la condition, (et donc un chemin d'exécution pour le simulateur), soit, au contraire, le choix de la valeur de vérité de cette condition est laissé au simulateur (qui peut alors choisir de faire intervenir l'utilisateur).

Nous avons développé des méthodes de calcul de variables de choix "élémentaires", qui sont basées :

- sur l'utilisation des Diagrammes de Décision Binaire pour les expressions booléennes,
- sur l'exploitation des propriétés des systèmes d'inéquations linéaires dans le cas d'expressions arithmétiques.

Nous avons également déterminé le comportement de la variable de choix libre pour des expressions complexes représentant des conditions de forme quelconque :

- en décomposant les expressions complexes en expressions élémentaires,
- en évaluant les variables de choix élémentaires associées,
- en évaluant la variable de choix grâce aux propriétés de distribution de l'opération de choix.

Finalement, la gestion de la Condition de Chemin permet de prendre en compte la majorité des expressions rencontrées par un simulateur symbolique de descriptions comportementales en VHDL.

Nous avons donc réalisé un simulateur symbolique de descriptions comportementales VHDL ; cet outil de simulation est utilisé dans le projet de vérification de descriptions VHDL, et peut être intégré dans d'autres environnements (test fonctionnel de circuits par exemple).

La vérification d'une description par simulation symbolique consiste alors à contrôler la conformité entre les résultats de la simulation symbolique et les spécifications. Or, ces résultats sont des expressions symboliques. La confrontation de ces éléments à la spécification formelle ne présente pas la complexité inhérente aux démonstrateurs classiques.

Ces travaux constituent donc une base nous permettant d'aborder l'utilisation de la simulation symbolique comme outil de preuve.

De plus, les méthodes de manipulation d'expressions symboliques développées dans le module de gestion de Condition de Chemin peuvent être utilisées, indépendamment de la simulation symbolique, dans des outils de preuve, ou de génération de séquence de test.

Références bibliographiques

[AKE77] Sheldon B. AKERS - On the Specification and Analysis of Large Digital Functions, FTCS 1977, pp. 88-93

[AKE78a] Sheldon B. AKERS - Binary Decision Diagrams, IEEE Transactions on Computers, vol. C-27, No. 6, June 1978, pp. 509-516

[AKE78b] Sheldon B. AKERS - Functional Testing with Binary Decision Diagrams, Proceedings 8th Annual IEEE on Fault Tolerant Computing, 1978, pp. 75-82

[AIR90] R. AIRIAU, J.M. BERGE, V. OLIVE, J. ROUILLARD - VHDL : du langage à la modélisation, Presses polytechniques et universitaires romandes, 1990

[ALA84] Riad ALALI - SACILAM : Une méthodologie interactive de conception hiérarchisée des systèmes logiques complexes, Thèse d'Etat, Université Montpellier II, 1984

[ARM89] James R. ARMSTRONG - Chip-Level Modeling with VHDL, Prentice Hall, Englewood Cliffs, 1989

[AUG87] Larry M. AUGUSTIN, Benoit A. GENNART, Youm HUH, David C. LUCKHAM, Alec G. STANCULESCU - VAL : An Annotation Language for VHDL, International Conference on Computer-Aided Design (ICCAD'87), Santa-Clara, California, November 1987, pp. 418-421

[AUG88] Larry M. AUGUSTIN, Benoit A. GENNART, Youm HUH, David C. LUCKHAM, Alec G. STANCULESCU - Verification of VHDL designs using VAL, 25th Design Automation Conference, Anaheim, California, June 1988, pp. 48-53

[AUG91] Larry M. AUGUSTIN, David C. LUCKHAM, Benoit A. GENNART, Youm HUH, Alec G. STANCULESCU - Hardware Design and Simulation in VAL/VHDL, Kluwer Academic Publishers, 1991

[AUM88] M. AUMIAUX - Logique binaire, MASSON, Paris, 1988

[BAR88] David L. BARTON - Behavioral Descriptions in VHDL, VLSI System Design, June 1988, pp. 28-33

[BIS84] Nripendra N. BISWAS - Computer Aided Minimization Procedure for Boolean Functions, 21st Design Automation Conference, 1984

[BIS86] Nripendra N. BISWAS - Computer Aided Minimization Procedure for Boolean Functions, IEEE Transactions on Computer-Aided Design, vol. CAD-5, No. 2, April 1986

[BIS90] Nripendra N. BISWAS - On Covering Distant Minterms by the CAMP Algorithm, IEEE Transactions on Computer-Aided Design, vol. 9, No. 7, July 1990

[BOU92a] Djamel BOUSSEBHA, Norbert GIAMBIASI, Janine MAGNIER - Temporal Specification and Verification of VHDL Descriptions, VHDL International Users' Forum, Spring 1992 Conference, Scottsdale, Arizona, 3-6 May 1992

[BOU92b] Djamel BOUSSEBHA, Norbert GIAMBIASI, Janine MAGNIER, Nathalie GIRARD - Temporal Verification of VHDL Descriptions, EURO-DAC 92, Hambourg, Germany, 7-10 September 1992

[BRA82] G.W. BRAMS - Réseaux de Petri : Théorie et Pratique, MASSON, Paris, 1982

[BRA84] Robert K. BRAYTON, Gary D. HACHTEL, Curtis T. McMULLEN, Alberto L. SANGIOVANNI-VINCENTELLI - Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, 1984

[BRY86] Randal E. BRYANT - Graph-Based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers, vol. C-35, No. 8, August 1986

[BUR90] J. R. BURCH, E. M. CLARKE, K. L. McMILLAN, David L. DILL - Sequential Circuit Verification using Symbolic Model Checking, 27th Design Automation Conference, Orlando, Florida, June 1990, pp. 46-51

[BUT91] Kenneth M. BUTLER, Don E. ROSS, Rohit KAPUR, M. RAY MERCER - Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams, 28th Design Automation Conference, San Francisco, California, June 1991, pp. 417-420

[CAR79] W. C. CARTER, W. H. JOYNER, D. BRAND - Symbolic Simulation for Correct Machine Design, 16th Design Automation Conference, San Diego, California, June 1979, pp. 280-286

[CHE79] Thomas E. CHEATHAM, Glenn H. HOLLOWAY, Judy A. TOWNLEY - Symbolic Evaluation and the Analysis of Programs, IEEE Transactions on Software Engineering, July 1979, pp. 402-417

[CHO89] Kyeongsoon CHO, Randal E. BRYANT - Test PAttern Generation for Sequential MOS Circuits by Symbolic Fault Simulation, 26th Design Automation Conference, Las Vegas, Nevada, June 1989, pp. 418-423

[CIA88] P.G. CIARLET - Introduction à l'analyse numérique matricielle et à l'optimisation, MASSON, Paris, 1988

[COU90] Olivier COUDERT, Christian BERTHET, Jean-Christophe MADRE - Formal Boolean Manipulations for the Verification of Sequential Machines, EDAC 90, 1990, pp. 57-61

[COU91] Olivier COUDERT - SIAM : Une Boite à Outils pour la Preuve Formelle de Systèmes Séquentiels, Thèse de l'E.N.S.T, Octobre 1991

[CUT87] R.B. CUTLER, S. MUROGA - Derivation of Minimal Sums for Completely Specified Functions, IEEE Transactions on Computers, vol. C-36, March 1987, pp 277-292

[DAG86] Michel R. DAGENAIS, Vinod K. AGARWAL, Nicholas C. RUMIN - McBOOLE : A New Procedure for Exact Logic Minimization, IEEE Transactions on Computer-Aided Design, vol. CAD-5, No. 1, January 1986

[DAN63] G.B. DANTZIG - Linear Programming and Extensions, Princeton University Press, Princeton, 1963

[DAR78] John A. DARRINGER, James C. KING - Applications of Symbolic Execution to Program Testing, Computer, April 1978, pp. 51-60

[DAR79] John A. DARRINGER - The Application of Program Verification Techniques to Hardware Verification, 16th Design Automation Conference, San Diego, California, June 1979, pp. 375-381

[DEL91] Thierry DELEAU - Langage de spécification du comportement temporel de circuits, Rapport de DEA, Université Montpellier II, Septembre 1991

[DIN88] M. DINCBAS, P. Van HENTENRYCK, H. SIMONIS, A. AGGOUN, T. GRAF, F.BERTHIER - The constraint logic programming language CHIP, Procs of Int Conference on Fifth Generation Computer Systems 88, ICOT, Tokyo, Japan, 1988

[DJA91] Kamel DJAFARI - , Modelisation en vue du test des flux de données des descriptions comportementales de circuits digitaux, Thèse de Doctorat, Université Montpellier II, Novembre 1991

[FAU79] Robert FAURE - Précis de recherche opérationnelle, DUNOD, Paris, 1979

[FLO67] Robert W. FLOYD - Assigning meanings to programs, Symposium on Applied Mathematics, Amer. Math. Soc., vol. 19, 1967, pp. 19-32

[FUJ88] Masahiro FUJITA, Hisanori FUJISAWA, Nobuaki KAWATO - Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams, International Conference on Computer-Aided Design (ICCAD'88), 1988, pp. 2-5

[GHA89] Malik GHALLAB, Amine MOUNIR-ALAOUI - Relations temporelles symboliques : Représentations et algorithmes, Rapport de Recherche LAAS 89.105, Avril 1989

[GIA89] Norbert GIAMBIASI, Rémy ROGACKI, Sylvie SAUMONT - Structural Symbolic Simulator of Digital Circuits, European Simulation Multiconference (ESM'89), Rome, June 1989, pp.387-392

[GIA91] Norbert GIAMBIASI, Janine MAGNIER, Mireille LARNAC, Rémy ROGACKI - Symbolic Simulation of Behavioral Descriptions in VHDL, VHDL International Users' Forum, Fall 1991 Meeting, Newport, California, 27-30 October 1991

[GOM58] R.E. GOMORY - An algorithm for integer solutions to linear programs, Princeton IBM Math. Report, 1958

[GON73] Michel GONDRAN - An efficient cutting-plane algorithm by the method of decreasing congruences, 8ème symposium de programmation mathématique, Stanford, 1973

[GON85] Michel GONDRAN, Michel MINOUX - Graphes et algorithmes, EYROLLES, Paris, 1985

[HAN76] Sidney L. HANTLER, James C. KING - An Introduction to Proving the Correctness of Programs, ACM Computing Surveys, September 1976, pp. 76-98

[HAO91] Jin-Kao HAO - Combiner la programmation en logique et la résolution des contraintes par évaluation partielle : Conslog, Thèse de Doctorat, Université de Bourgogne, Février 1991

[HAS87] Larbi HASSOUNI - Etude et développement d'un simulateur symbolique comportemental de circuits digitaux, Thèse de Doctorat, Université Aix-Marseille III, 1987

[HIN87] John HINES - Where VHDL fits within the CAD environment, 24th Design Automation Conference, Miami Beach, Florida, June 1987, pp. 491-494

[HON74] S.J. HONG, R.G. CAIN, D.L. OSTAPPKO - MIMI : A Heuristic Approach for Logic Minimization, IBM Journal of Research and Development, vol. 18, September 1974

[HON91]. Sung Je HONG, Saburo MUROGA - Absolute Minimization of Completely Specified Switching Functions, IEEE Transactions on Computers, Vol. 40, No. 1, January 1991

[HOW76] WIlliam E. HOWDEN - Reliability of the Path Analysis Testing Strategy, IEEE Transactions on Software Engineering, vol. SE-2, No. 3, September 1976, pp. 208-215

[HOW77] William E. HOWDEN - Symbolic Testing and the DISSECT Symbolic Evaluation System, IEEE Transactions on Software Engineering, July 1977, pp. 266-278, également dans Software

Testing and Validation Techniques : Tutorial, 2nd edition, IEEE Computer Society Press, pp. 194, 206

[HYA76] Laurent HYAFIL, Ronald L. RIVEST - Constructing Optimal Decision Trees is NP-Complete, Information Processing Letters, volume 5, number 1, May 1976, pp. 15-17

[IEE86] Special Issue on VHDL, IEEE Design and Test of Computers, April 1986

[IEE87] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1987, December 1987

[JAF87a] Joxan JAFFAR, Jean-Louis LASSEZ - Constraint Logic Programming, Procs of the 14th ACM POPL Conference, Munich, Germany, January 1987

[JAF87b] Joxan JAFFAR, Spiro MICHAYLOV - Methodology and Implementation of a CLP System, Procs of the 4th Int Conference on Logic Programming, Melbourne, Australia, May 1987

[JAF88] J. JAFFAR, S. MICHAYLOV, P.J. STUCKEY, H.C. YAP - The CLP($\mathcal{R}$) language and system, IBM/CMU Research Report, April 1988

[KIN69] James C. KING - A Program Verifier, Ph. D. Dissertation, Carnegie Mellon University, Pittsburg, Pa, September 1969

[KIN71] James C. KING - Proving Programs to be Correct, IEEE Transactions on Computers, vol. C-20, No. 11, November 1971, pp. 1331-1336

[KIN72a] James C. KING - A Program Verifier, Information Processing 71, North Holland Publishing Company, 1972, pp. 234-249

[KIN72b] James C. KING, Robert W. FLOYD - An Interpretation-Oriented Theorem Prover over Integers, Journal of Computer and System Sciences , 6, 1972, pp. 305-323

[KIN76] James C. KING - Symbolic Execution and Program Testing, Communications of the ACM, vol. 19, No. 7, July 1976, pp. 385-394

[LAG76] J. LAGASSE, M. COURVOISIER, J.P. RICHARD, Logique combinatoire, DUNOD, Paris, 1976

[LAR92] Mireille LARNAC, Norbert GIAMBIASI, Janine MAGNIER, Rémy ROGACKI - Handling the Path Condition in Symbolic Simulation, Symbolic Methods and Applications to Circuit Design (SMACD'92), Firenze, Italy, 8-9 Octobre 1992

[LEE59] C. Y. LEE - Representation of Switching Circuits by Binary-Decision Programs, The BELL System Technical Journal, vol. 38, July 1959, pp. 985-999

[LIP89] Roger LIPSETT, Carl SCHAEFER, Cary USSERY - VHDL : Hardware Description and Design, Kluwer Academic Publishers, 1989

[LOU88] M. LOUGHZAIL, M. COTE, M. ABOULHAMID, E. CERNY - Experience with the VHDL Environment, 25th Design Automation Conference, Anaheim, California, June 1988, pp. 28-33

[MAD88] Jean-Christophe MADRE, Jean-Paul BILLON - Proving Circuit Correctness using Formal Comparison between Expected and Extracted Behaviour, 25th Design Automation Conference, Anaheim, California, June 1988, pp. 205-210

[MAG90] Janine MAGNIER - Représentation symbolique et vérification formelle de machines séquentielles, Thèse d'Etat, Université Montpellier II, Juillet 1990

[MAL88] Sharad MALIK, Albert R. WANG, Robert K. BRAYTON, Alberto SANGIOVANNI-VINCENTELLI - Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment, International Conference on Computer-Aided Design (ICCAD'88), 1988, pp. 6-9

[MAR88] Erich MARSCHNER - A VHDL Design Environment, VLSI Systems Design, September 1988, pp. 40-49

[MAT89] Yusuke MATSUNAGA, Masahiro FUJITA - Multi-Level Logic Optimization Using Binary Decision Diagrams, International Conference on Computer-Aided Design (ICCAD'89), November 1989, pp. 556-559

[McC56] E.J. McCLUSKEY Jr. - Minimization of Boolean Functions, The BELL System Technical Journal, November 1956, pp 1417-1444

[MEY89] Ernest MEYER - VHDL opens the road to top-down design, Computer Design, February 1, 1989, pp. 57-62

[MIN90] Shin-ichi MINATO, Nagisa ISHIURA, Shuzo YAJIMA - Shared Binary Decision Diagrams with Attributed Edges for Efficient Boolean Function Manipulation, 27th Design Automation Conference, Orlando, Florida, June 1990, pp. 52-57

[MOR81] Bernard M. E. MORET, Michael G. THOMASON, Rafael C. GONZALEZ - Optimization Criteria for Decision Trees, Technical Report CS81-6, Computer Science Department, University of New Mexico, Albuquerque, New Mexico 87131, 1981

[MOR82] Bernard M. E. MORET - Decision Trees and Diagrams, ACM Computing Surveys, vol. 14, No. 4, December 1982, pp. 593-623

[NUR89] G.M. NURIE, P.J. MENCHINI - VHDL Model Portability, High Performance Systems, July 1989, pp. 76-85

[ODA86] Gotaro ODAWARA, Masahiro TOMITA, Osamu OKUZAWA, Tomomichi OHTA, Zhen-quan ZHUANG - A Logic Verifier Based on Boolean Comparison, 23rd Design Automation Conference, 1986, pp. 208-214

[OPL89] Alexis OPLOBEDU, Jacques MARCOVICH, Yves TOURBIER - CHARME : un langage industriel de programmation par contraintes, 9th Int. Workshop on Expert Systems and Applications, AVIGNON'89, May 1989

[OUS88] M.C. OUSSALAH - Modèles hiérarchisés/multi-vues pour le support de raisonnement dans les domaines techniques, Thèse de Doctorat, Université Aix-Marseille III, Juin 1988

[PEN86] Zebo PENG - Synthesis of VLSI Systems with the CAMAD Design Aid, 23rd Design Automation Conference, 1986, pp. 278-284

[PER67] J.P. PERRIN, M. DENOUETTE, E. DACLIN - Systèmes logiques, Tome 1, DUNOD, Paris, 1967

[PET81] J.L. PETERSON - Petri Net Theory and the Modeling of Systems, Prentice Hall, 1981

[PIT82] Vijay PITCHUMANI, Edward P. STABLER - A Formal Method for Computer Design Verification, 19th Design Automation Conference, 1982, pp. 809-814

[ROG92a] Rémy ROGACKI - Simulation symbolique de descriptions comportementales de circuits digitaux, Thèse de Doctorat, Université Montpellier II, 1992

[ROG92b] Rémy ROGACKI, Norbert GIAMBIASI, Mireille LARNAC, Janine MAGNIER - Symbolic Simulator of Behavioral Descriptions in VHDL, Symbolic Methods and Applications to Circuit Design (SMACD'92), Firenze, Italy, 8-9 Octobre 1992

[ROS91] Don E. ROSS, Kenneth E. BUTLER, M. RAY MERCER - Exact Ordered Binary Decision Diagram Size When Representing Classes of Symmetric Functions, Journal of Electronic Testing : Theory and Applications, 2, Kluwer Academic Publishers, Boston, 1991, pp. 243-259

[RUD86] Richard RUDELL, Alberto L. SANGIOVANNI-VINCENTELLI - Exact Minimization of Multiple-Valued Functions for PLA Optimization, International Conference on Computer-Aided Design (ICCAD'86), November 1986, pp 352-355

[SAU87] Larry F. SAUNDERS - The IBM VHDL Design System, 24th Design Automation Conference, Miami Beach, Florida, June 1987, pp. 484-490

[SHA86] Moe SHAHDAD - An overview of VHDL Language and Technology, 23rd Design Automation Conference, 1986, pp. 320-326

[STO85] P.D. STOTTS, T.W. PRATT - Hierarchical Modeling of Software Systems with Timed Petri Nets, International Workshop on Timed Petri Nets, Torino, Italy, July 1985, pp. 32-39

[TIS67] Pierre TISON - Generalization of Consensus Theory and Application to the Minimization of Boolean Functions, IEEE Transactions on Electronic Computers, vol. EC-16, No. 4, August 1967, pp 446-456

[WAG77] Todd J. WAGNER - Verification of Hardware Design thru Symbolic Manipulation, Procs. Design Automation and Microprocessors, Palo Alto, California, February 24-25, 1977, pp. 50-53

Table des Matières

Liste des Figures

Annexe 1

• Algorithme de principe du OU logique de deux arbres (ou diagrammes) de décision binaire :

fonction OU(x,y)

```
début
    si      nom(x) > nom(y)
    alors   inverser(x,y)
    si      x non terminal
    alors   si      nom(x) ≠ nom(y)
            alors   fils-gauche(z) ← OU(fils-gauche(x),y)
                    fils-droit(z) ← OU(fils-droit(x),y)
            sinon   fils-gauche(z) ← OU(fils-gauche(x),fils-gauche(y))
                    fils-droit(z) ← OU(fils-droit(x),fils-droit(y))
    sinon   si      valeur(x)=0 ou valeur(y)=1
            alors   z ← y
    retourner(simplifier(z))
fin
```

• Algorithme de principe du ET logique de deux arbres (ou diagrammes) de décision binaire :

fonction ET(x,y)

```
début
        si      nom(x) > nom(y)
        alors   inverser(x,y)
        si      x non terminal
        alors   si      nom(x) ≠ nom(y)
                alors   fils-gauche(z) ← ET(fils-gauche(x),y)
                        fils-droit(z) ← ET(fils-droit(x),y)
                sinon   fils-gauche(z) ← ET(fils-gauche(x),fils-gauche(y))
                        fils-droit(z) ← ET(fils-droit(x),fils-droit(y))
        sinon   si      valeur(x)=1
                alors   z ← y
        retourner(simplifier(z))
fin
```

Annexe 2

• Algorithme de principe de construction de l'arbre A associé à une inéquation de forme normalisée :

$$C_N = (\sum_{i=1}^{n} \lambda_{iN}\, x_i)\ @_N\ k_N$$

fonction Remplir($A(x_i)$)

```
début
    si      i ≤ n
    alors   Allouer(A(xi).M)
            Allouer(A(xi).Z)
            Allouer(A(xi).P)
            si      λiN = -1
            alors   Remplir(A(xi).M)
                    A(xi).Z ← ϕ
                    A(xi).P ← ϕ
            sinon si      λiN = 0
                  alors   Remplir(A(xi).Z)
                          A(xi).M ← ϕ
                          A(xi).P ← ϕ
                  sinon           Remplir(A(xi).P)
                          A(xi).M ← ϕ
                          A(xi).Z ← ϕ
    sinon si      sN = '<'
          alors   A(xi).I ← kN
                  A(xi).S ← kN
fin
```

• L'arbre A résultat de la fonction Remplir est complété par la fonction suivante ; il représente le système complet associé à l'inéquation C_N.

Fonction Compléter($A(x_i)$,$B(x_i)$)

```
début
    si      A(xi) ≠ ϕ
    alors   si      (A(xi).M ≠ ϕ ∨ A(xi).Z≠ϕ ∨ A(xi).P≠ϕ)
            alors   si      A(xi).M ≠ ϕ
                    alors   Compléter(A(xi).M)
                            Allouer(A(xi).P)
                    si      A(xi).Z ≠ ϕ
                    alors   Compléter(A(xi).Z)
                    si      A(xi).P ≠ ϕ
                    alors   Compléter(A(xi).P)
                            Allouer(A(xi).M)
            sinon   si      A(xi).I ≠ ϕ
                    alors   B(xi).S ← sup(B(xi).S, – A(xi).I)
                    si      A(xi).S ≠ ϕ
                    alors   B(xi).I ← inf(B(xi).I, – A(xi).S)
fin
```

• Algorithme de combinaison de deux arbres, A et B.

Le premier appel à cette fonction est : Construct($A(x_1)$,$B(x_1)$,0).

Fonction Construct($A(x_i)$,$B(x_i)$,n)

```
début
    si      n ≠ 0
    alors   j = | n |
    sinon   j = sig_{i-1}(A)
    si      A(x_i) = ϕ
    alors   si      B(x_i) ≠ ϕ
            alors   si      sig_{i-1}(B) = sig_{i-1}(C)
                    alors   C(x_i) ← B(x_i)
                    sinon   C(x_i) ← ϕ
            sinon   C(x_i) ← ϕ
    sinon   si      B(x_i) = ϕ
            alors   si      sig_{i-1}(A) = sig_{i-1}(C)
                    alors   C(x_i) ← A(x_i)
            sinon   C(x_i) ← ϕ
    sinon   si      (j ≠ 0 ∧ i = n)
            alors   si      (A(x_i).I ≠ ϕ ∨ A(x_i).S ≠ ϕ ∨ B(x_i).I ≠ ϕ ∨ B(x_i).S ≠ ϕ)
                    alors   Allouer(C(x_i))
                            si      sig_{i-1}(A) = k . sig_{i-1}(B)      k ∈ ℤ*
                            alors   si      j ≠ 0
                                    alors   si      A(x_i).I ≠ ϕ
                                            alors   C(x_i).I ← inf(C(x_i).I,inf(k.A(x_i).I,B(x_i).I))
                                            sinon   C(x_i).I ← inf(C(x_i).I,B(x_i).I)
                                            si      A(x_i).S ≠ ϕ
                                            alors   C(x_i).S ← sup ( C(x_i).S,
                                                                sup( k.A(x_i).S, B(x_i).S))
                                            sinon   C(x_i).S ← sup ( C(x_i).S, B(x_i).S)
                                    sinon   C(x_i) ← ϕ
                            sinon   si      j ≠ 0
                                    alors   si      j = 1
                                            alors   si      (A(x_i).I ≠ ϕ ∧ B(x_i).I ≠ ϕ)
                                                    alors   C(x_i).I ← inf (C(x_i).I,
                                                                A(x_i).I + B(x_i).I - 1)
```

```
                                    si      (A(xi).S ≠ ϕ ∧ B(xi).S ≠ ϕ)
                                    alors   C(xi).S ← sup (C(xi).S,
                                                    A(xi).S+B(xi).S+1)
                                sinon si    (A(xi).I ≠ ϕ ∧ B(xi).I ≠ ϕ)
                                    alors   C(xi).I ← inf (C(xi).I,
                                                    (A(xi).I + B(xi).I)/2)
                                    si      (A(xi).S ≠ ϕ ∧ B(xi).S ≠ ϕ)
                                    alors   C(xi).S ← sup(C(xi).S,
                                                    (A(xi).S+B(xi).S)/2)
                            sinon   C(xi) ← ϕ
                    sinon   C(xi) ← ϕ
            sinon si        (j≠0 ∨ A(xi).M≠ϕ ∨ A(xi).Z≠ϕ ∨ A(xi).P≠ϕ ∨ B(xi).M≠ϕ ∨
                             B(xi).Z≠ϕ ∨ B(xi).P≠ϕ)
                    alors   Allouer(C(xi))
                            sig(i)(A) ← -1; sig(i)(B) ← +1; sig(i)(C) ← 0;
                            C(xi).Z ← Construct (A(xi).M, B(xi).P, 0)
                            sig(i)(A) ← 0; sig(i)(B) ← 0; sig(i)(C) ← 0;
                            C(xi).Z ← Construct (A(xi).Z, B(xi).Z, 0)
                            sig(i)(A) ← +1; sig(i)(B) ← -1; sig(i)(C) ← 0;
                            C(xi).Z ← Construct (A(xi).P, B(xi).M, 0)
                            si      j ≠ 2
                            alors   sig(i)(A) ← -1; sig(i)(B) ← 0; sig(i)(C) ← -1;
                                    C(xi).M ← Construct (A(xi).M, B(xi).Z, -1)
                                    sig(i)(A) ← 0; sig(i)(B) ← -1; sig(i)(C) ← -1;
                                    C(xi).M ← Construct (A(xi).Z, B(xi).M, -1)
                                    sig(i)(A) ← +1; sig(i)(B) ← 0; sig(i)(C) ← +1;
                                    C(xi).P ← Construct (A(xi).M, B(xi).P, +1)
                                    sig(i)(A) ← 0; sig(i)(B) ← +1; sig(i)(C) ← +1;
                                    C(xi).P ← Construct (A(xi).M, B(xi).P, +1)
                            si      j ≠ 1
                            alors   sig(i)(A) ← -1; sig(i)(B) ← -1; sig(i)(C) ← -2;
                                    C(xi).M ← Construct (A(xi).M, B(xi).M, -2)
                                    sig(i)(A) ← +1; sig(i)(B) ← +1; sig(i)(C) ← +2;
                                    C(xi).P ← Construct (A(xi).P, B(xi).P, +2)

fin
```

Résumé :

Le traitement des branchements conditionnels est un problème spécifique à la simulation symbolique.
La gestion de la Condition de Chemin consiste à déterminer si la valeur de vérité d'une condition est contenue dans tous les choix qui ont été opérés sur les tests précédemment rencontrés, ou si, au contraire, le choix de cette valeur est laissé à la libre appréciation de l'utilisateur : c'est le rôle de l'opérateur de choix.
Cet opérateur et les propriétés qui le caractérisent sont définis. Son application sur des expressions particulières, puis dans le cas général est ensuite étudiée.

Mots-clés :

Vérification de circuits digitaux - Simulation symbolique - Condition de Chemin - Opérateur de choix - Variable de choix-libre