Année 2001

THÈSE

présentée à

L'UNIVERSITÉ D'AIX-MARSEILLE I

au sein du Laboratoire d'Informatique de Marseille

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ D'AIX-MARSEILLE I

Spécialité : **Informatique**

par **Gilles AUDEMARD**

Résolution du problème SAT et génération de modèles finis en logique du premier ordre

Soutenue le 25 octobre 2001

Devant le jury composé de

Belaid BENHAMOU	Maître de Conférence à l'Université de Provence	(directeur de thèse)
Jean-Jacques CHABRIER	Professseur à l'Université de Bourgogne	(rapporteur)
Chu Min LI	Maître de Conférence à l'Université de Picardie	(examinateur)
Lakhdar SAIS	Professeur à l'Université Paul Sabatier	(examinateur)
Pierre SIEGEL	Professeur à l'Université de Provence	(examinateur)
Jian ZHANG	Research Professor - Chinese Academy of Sciences	(rapporteur)

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Remerciements

C'EST AVEC BEAUCOUP DE JOIE QUE J'ÉCRIS CETTE PAGE. En effet, elle signifie que j'en ai terminé avec ce mémoire et que je vais pouvoir remercier tous les gens qui, plus ou moins directement, m'ont aidé et soutenu durant ces trois années.

Tout d'abord je voudrais remercier Belaid BENHAMOU, mon directeur de thèse. Dès la maîtrise, il m'a mis en face de problèmes très intéressants. Il m'a appris, entre autres, à rédiger des articles, à exprimer correctement mes idées. Il a également réussi (ce qui ne doit pas être évident) à être présent juste ce qu'il faut.

Un grand merci à Pierre SIEGEL, Lakdhar SAIS (merci également pour m'avoir permis d'assister aux JNPC de Toulouse), Chu Min LI qui m'ont fait l'honneur d'être dans mon jury, et plus particulièrement à Jean–Jacques CHABRIER et à Jian ZHANG qui ont, en plus, accompli la contraignante tâche de rapporteur.

Plusieurs Laurent m'ont beaucoup apporté durant ces trois années. Je vais faire un tir groupé et les remercier en même temps. Laurent HENOCQUE qui est une personne très intéressante, autant d'un point de vue scientifique qu'humain. Laurent SIMON qui termine sa thèse à Paris. Nos discussions enflammées dans les bistrots en marge des JNPC et autres CADE me laisseront de grands souvenirs. Et enfin, Laurent IMBERT, nos parcours respectifs se ressemblent de plus en plus (pourvu que ça dure !).

Merci à Cédric (il a réussi à supporter tous les bruits que je faisais) , à Éric (et à son terrible cassoulet), à Fred (mais qui va s'occuper de son ordi ?) et à Cécile pour les pauses café, les pique–niques, les soirées...Merci à Philippe JEGOU (et au mouvementé voyage à Lyon). Merci également aux autres doctorants Éric, Lamia, Cyril, Vincent, Sarah et Julie. Il est agréable de savoir que l'on va travailler à côté de gens cools, gentils et de bonne humeur.

Et puis, comme (heureusement) il n'y a pas que le travail dans la vie, je tiens à remercier tous ceux qui m'ont permis de me changer les idées : les "pécheurs" (vu le nombre de poissons que nous attrapons, je me demande si ce terme est adéquat), les handballeurs, les "gens" avec qui on sort le vendredi soir, tout le monde en fait. Je ne mettrai pas de noms, trop peur d'oublier quelqu'un.

Merci à la petite famille pour m'avoir toujours soutenu. J'en termine, et par le plus important, un grand merci à Steph, Sally et Théo pour tout.

Introduction

En 1950, le mathématicien Alan Turing publiait un article [97] qui donnait naissance à ce que l'on allait appeler plus tard l'*intelligence artificielle*. Il posait la question «Les machines peuvent–elles penser ?» et proposait, pour tenter d'y répondre, un jeu nommé «le jeu de l'imitation». Ce jeu consiste à prendre un homme (A), une femme (B) et un troisième joueur (C). Les personnes A et B étant cachées, C doit deviner à l'aide de questions qui est la femme et qui est l'homme. La personne B, contrairement à A, doit faciliter la tâche de C. Alan Turing proposait de remplacer A ou B par un ordinateur et de regarder si le joueur C gagnait plus facilement ou non.

Son article est agrémenté d'argumentations concernant un ensemble de neufs objections (philosophique, mathématique . . .) que n'allaient pas manquer de donner les nombreux détracteurs (comme Hubert Dreyfus [39]) pensant que ce test ne serait jamais satisfait. Quant à Alan Turing, il imaginait que 50 ans plus tard, un ordinateur serait capable d'imiter suffisamment bien un des deux joueurs, pour que la personne C n'ait pas plus de 70% de chances de procéder à l'identification exacte de l'homme et de la femme au bout de 5 minutes. On arrive aujourd'hui aux termes de ces 50 ans et le test de Turing est loin d'avoir été satisfait. Les ordinateurs n'ont pas encore un comportement humain. Pour autant, de nombreuses recherches ont permis certaines avancées dans ce que l'on appelle l'*intelligence artificielle*.

Aujourd'hui, on peut définir l'*intelligence artificielle* comme l'ensemble des disciplines dont le but est d'imiter le comportement intelligent humain. Citons, par exemple, la traduction automatique, la démonstration automatique, la reconnaissance des formes, la compréhension de la parole, la modélisation de forme de raisonnement incertaine, temporelle. . .

Une des disciplines de recherche en intelligence artificielle concerne la représentation et le traitement des connaissances. Le langage choisi doit avoir une grande expressivité et doit, en même temps être efficace, c'est à dire que les méthodes de déduction associées doivent être rapides. Malheureusement, ces deux critères sont difficilement cumulables.

Une des logiques les plus utilisées jusqu'ici a été la logique propositionnelle. Ce langage possède un grand nombre de qualités. Il est très simple mais permet de représenter une grande variété de connaissances. De nombreuses autres logiques peuvent s'y ramener. De plus, cette logique est décidable. C'est à dire qu'il existe toujours un algorithme vérifiant en un temps fini qu'une formule de la

logique propositionnelle est satisfaisable ou pas.

Un autre atout de la logique propositionnelle est le problème SAT qui en est directement issu et auquel nous nous intéressons dans la première partie de cette thèse. Le problème SAT consiste à déterminer si une formule propositionnelle mise sous forme normale conjonctive est satisfaisable. Ce problème a été le premier à être démontré, par Stephen COOK [29], comme étant NP–complet. Beaucoup d'autres problèmes qui s'y ramènent naturellement ont pu être classifiés par la suite. De nombreuses études, théoriques et pratiques, ont porté sur le problème SAT. Elles ont permis de créer des algorithmes, souvent basés sur l'algorithme énumératif de DAVIS et PUTNAM [35, 34] de plus en plus efficaces.

Tous ces avantages ne doivent quand même pas faire oublier que la transformation d'un problème en logique propositionnelle peut poser des problèmes. Des instances de très grandes tailles peuvent être générées. De plus, on peut perdre la structure du problème initial qui est pourtant un élément très utile pour améliorer les performances des algorithmes qui en découlent. C'est pour ces diverses raisons que, dans la deuxième partie de cette thèse, nous nous sommes également intéressés à la logique du premier ordre.

La logique du premier ordre est plus expressive et plus concise que la logique propositionnelle. Elle a été initialement introduite par les mathématiciens qui l'utilisent toujours pour représenter l'ensemble des objets dont ils se servent. Elle a ensuite intéressé les informaticiens par sa capacité à représenter et à manipuler les connaissances.

La recherche de modèles pour des théories de la logique du premier ordre est un problème important. Les modèles peuvent servir pour trouver des contres–exemples de conjectures [48]. En effet, la manière la plus simple pour prouver qu'un théorème est faux est d'en exhiber un contre–exemple. Ce dernier peut alors être utilisé pour comprendre les raisons de l'inexactitude du théorème et permettre ainsi sa correction. De plus, le raisonnement humain ne se fait pas uniquement d'une manière syntaxique (la forme de la formule) mais utilise également la sémantique (le sens de la formule). On se sert beaucoup des modèles pour mettre en place des démonstrations. D'ailleurs, par le passé, certains générateurs de modèles ont servi de complément à des démonstrateurs de théorèmes, qui ne traitent eux que de la syntaxe des théories [89, 21].

La recherche de modèles finis s'avère également d'un grand intérêt. Contrairement à la logique du premier ordre qui est indécidable, la recherche de modèles finis est un problème décidable. En informatique une grande majorité des structures utilisées sont finies. Certains problèmes n'apparaissent d'ailleurs que sous une forme finie comme les graphes, la spécification de programmes...

Pour autant, les recherches concernant les générateurs de modèles finis ont été bien moindres que celles portant sur le problème SAT. Il apparaît aussi que contrairement au problème SAT où la méthode DP sert souvent de méthode de base, les générateurs de modèles finis sont souvent très différents, par

le formalisme adopté et par l'exploration de l'espace de recherche.

L'objet de ce travail est de proposer de nouveaux algorithmes pour améliorer, d'une part les méthodes énumératives de résolution du problème SAT, et de l'autre, les générateurs de modèles finis de théories de la logique du premier ordre.

La première partie de ce mémoire est consacrée à la logique propositionnelle. Nous commençons dans le chapitre 1 par définir les concepts nécessaires à la lecture de cette partie. Nous accompagnons cette étude d'un état de l'art succint autour du problème SAT.

Nous proposons dans le chapitre 2 une nouvelle méthode énumérative que nous appelons AVAL et qui utilise d'une façon optimale la propagation des mono–littéraux. Nous introduisons un algorithme de production de littéraux. Cet algorithme est appelé à chaque fois que la propriété classique des mono–littéraux utilisée dans la méthode de DAVIS et PUTNAM [34] ne s'applique pas. Le but de ce travail est double : d'une part, améliorer l'efficacité des méthodes d'énumération par l'utilisation efficace et optimale de la propagation des mono–littéraux pour élaguer au mieux l'arbre de recherche. De l'autre, présenter un algorithme robuste qui permet d'expliquer les limites des méthodes complètes sur la résolution des problèmes SAT difficiles.

Le chapitre 3 est consacré à une expérimentation de la méthode AVAL. Une étude détaillée basée sur les instances aléatoires y est menée. Elle a mis en évidence la structure de l'arbre de recherche lors de la résolution d'instances difficiles. Dépassé une certaine profondeur de l'arbre de recherche, une *avalanche* de propagations de mono–littéraux est observée. La dénomination AVAL de notre méthode vient de cette avalanche de propagations. La fin de ce chapitre est consacrée à la comparaison de AVAL avec diverses méthodes parmi les plus efficaces existant dans la littérature.

La deuxième partie de cette thèse porte sur la génération de modèles finis pour les théories multi–types de la logique du premier ordre. Dans son premier chapitre, nous donnons les définitions nécessaires à une bonne compréhension ainsi qu'une description des différents générateurs de modèles finis qui ont été proposés dans le passé.

Nous avons travaillé particulièrement sur les générateurs de modèles finis qui considèrent les théories multi-types comme des problèmes de satisfaction de contraintes particuliers. Nous nous sommes focalisés sur deux points capitaux dans la génération de modèles finis : la propagation des contraintes, et l'étude de la détection et de l'élimination des symétries du problème à résoudre.

Le chapitre 5 est dédié à la propagation des contraintes et à la proposition d'heuristiques pour les générateurs de modèles finis. La première approche est un pré-traitement qui modifie la syntaxe de la théorie d'entrée. Cette technique calcule une nouvelle théorie équivalente à celle de départ qui permet d'effectuer plus de propagations unitaires.

La deuxième approche se traduit par un algorithme de type «look-ahead» qui supprime pendant le processus de recherche du modèle fini les valeurs des domaines de différents termes qui sont incompatibles avec l'instantiation courante. Ceci permet de réduire l'espace de recherche du modèle en cours de construction. Cette méthode induit une heuristique de type UP (Unit Propagation) [43, 65, 78] que nous utilisons pour mieux choisir les prochains termes à instantier.

Dans la dernière approche, nous introduisons des nouvelles heuristiques pour les générateurs de modèles finis. En étudiant les dépendances qui existent entre les différents symboles fonctionnels, nous créons un graphe de dépendance. Il est exploité pour déterminer les symboles fonctionnels qu'il faut interpréter en premier lieu, donnant naissance à plusieurs heuristiques de type «échec d'abord».

Le deuxième point d'intérêt de cette partie est développé dans le chapitre 6 et consiste en l'étude des symétries pour la génération de modèles finis. Les théories multi-types, de par leur formulation, contiennent un grand nombre d'isomorphismes. Les générateurs de modèles finis n'exploitent pas de façon optimale les propriétés de symétrie et perdent tout avantage que peut ramener leur utilisation. Il faut donc supprimer une grande partie de ces isomorphismes pour avoir un générateur de modèles finis efficace.

Nous débutons le chapitre 6 en définissant les notions de symétries pour les théories multi–types de la logique du premier ordre et en rappelant les diverses techniques utilisées pour supprimer des symétries. Nous introduisons ensuite deux stratégies différentes mais complémentaires de traitement des isomorphismes.

Nous avons mis au point une heuristique, nommée XLNH («eXtended Least Number Heuristic»). C'est une extension de l'heuristique LNH qui supprime plus de symétries triviales. Elle est utilisable dès que la théorie possède une fonction unaire. Cette méthode commence par générer les interprétations canoniques (non symétriques) de cette fonction unaire. Nous détectons ensuite statiquement des interprétations isomorphes. L'espace de recherche est donc réduit sans sur–coût de temps comme le fait l'heuristique LNH utilisée par les générateurs de modèles finis SEM [106] et FMSET[15].

Bien qu'intéressantes, les symétries supprimées par les heuristiques LNH et XLNH disparaissent rapidement en avançant dans la solution. Ces deux stratégies laissent donc de nombreuses interprétations isomorphes. Nous avons montré de nouvelles propriétés concernant les symétries et avons étudié un algorithme qui permet de les détecter et de les exploiter dynamiquement. Nous montrons que les symétries supprimées par les heuristiques LNH et XLNH sont des cas particuliers de ce cadre d'étude et peuvent être combinées avec.

Le but du chapitre 7 est d'expérimenter les différentes méthodes et heuristiques décries dans les chapitres 5 et 6. Nous commençons par étudier leurs comportement. Dans un second temps nous faisons une comparaison de ces stratégies avec des générateurs existant dans la littérature. Les expérimentations sont faites sur une large variété de problèmes.

Bien entendu, nous terminons ce mémoire par une conclusion et évoquons différents axes de recherches futures.

Première partie

Résolution du problème SAT

Chapitre 1

Logique propositionnelle et problème SAT

LA LOGIQUE PROPOSITIONNELLE est la manière la plus simple pour représenter des connaissances. Elle a été introduite il y a près de 150 ans par George BOOLE [53]. Malgré son apparente simplicité, la logique propositionnelle permet de représenter une grande quantité de connaissances. Elle a pour objet l'étude des formes de raisonnement dont la validité est indépendante de la structure des propositions, et résulte uniquement de leurs propriétés d'être vraies ou fausses.

Le problème SAT est issu de la logique propositionnelle. Il consiste a déterminer si une formule propositionnelle mise sous forme clausale est satisfaisable ou pas. La plupart des démonstrateurs automatiques exploitent le théorème de déduction. Ils utilisent au moins un test de satisfaisabilité pour montrer qu'une information peut être déduite à partir d'un ensemble de connaissance (voir section 1.1.3). Le problème SAT s'est révélé important à ce titre.

Il occupe également une place particulière et centrale parmi les problèmes NP–complets. En effet, il a été le premier a être démontré comme tel par Stephen COOK [29]. De plus, de nombreux autres problèmes qui s'y ramènent naturellement ont pu être classifiés par la suite.

Toutes ces raisons ont fait que, au cours de ces 50 dernières années, les recherches portant sur le problème SAT ont été nombreuses et variées. En 1997, Jun GU, Paul W. PURDOM, John FRANCO et Benjamin W. WAH ont écrit un article extrêmement détaillé (134 pages et 561 références) passant en revue les diverses techniques algoritmiques mises au point pour résoudre le problème SAT [54]. Plus récemment, un «special issue» du "journal of Automated Reasonning" a publié une dizaine d'articles pour situer l'état des travaux sur le problème SAT en l'an 2000 [52].

Il semblerait illusoire dans ce chapitre de vouloir résumer l'ensemble de ces recherches. C'est pourquoi, après avoir défini les différents concepts nécessaires pour la suite de la lecture, nous nous limiterons aux quelques méthodes et heuristiques qui ont été récemment étudiées.

1.1 La logique propositionnelle

Nous définissons ici la logique propositionnelle en précisant tout d'abord la syntaxe puis la sémantique du langage.

1.1.1 Syntaxe

Définition 1.1 *Le langage de la logique propositionnelle est construit à partir de l'alphabet constitué de :*

- *Un ensemble $\mathcal{V}$ de variables propositionnelles*
- *Les parenthèses «(» et «)»*
- *Les opérateurs, appelés également connecteurs, non «$\neg$», ou «$\vee$», et «$\wedge$», implique «$\rightarrow$», équivaut «$\leftrightarrow$».*
- *les 2 constantes «Vrai» et «Faux», que l'on peut noter également par «True» et «False», «$\top$» et «$\bot$» ou encore «0» et «1».*

Définition 1.2 (Formule) *L'ensemble des formules du calcul propositionnel est le plus petit ensemble d'expressions tel que :*

- *Les variables propositionnelles et les constantes «$\top$» et «$\bot$» sont des formules.*
- *Si F est une formule alors $\neg F$ et (F) sont des formules.*
- *Si F et G sont des formules alors $F \vee G$, $F \wedge G$, $F \rightarrow G$, $F \leftrightarrow G$ sont des formules.*

Définition 1.3 *Soit $v \in \mathcal{V}$ une variable propositionnelle, on appelle* littéral *la variable v (littéral positif) ou sa négation $\neg v$ (littéral négatif).*

1.1.2 Sémantique

Définition 1.4 *Une interprétation I est une fonction $I : \mathcal{V} \mapsto \{Vrai, Faux\}$. On généralise une interprétation à l'ensemble des formules de la logique propositionnelle de manière inductive :*

- *$I(Vrai) = Vrai$ et $I(Faux) = Faux$*
- *$I(\neg F) = Vrai$ si et seulement si $I(F) = Faux$*
- *$I(F \vee G) = Vrai$ si et seulement si $I(F) = Vrai$ ou $I(G) = Vrai$*
- *$I(F \wedge G) = Vrai$ si et seulement si $I(F) = Vrai$ et $I(G) = Vrai$*

L'interprétation d'une formule est donc définie par l'interprétation des variables qui la composent. Elle peut également être définie comme l'ensemble de littéraux interprétés à vrai. Nous introduisons maintenant diverses notions nécessaires pour la suite de la lecture :

Définition 1.5

- *Une interprétation I est dite partielle, si il existe au moins une variable v pour laquelle $I(v)$ n'est pas défini. Dans le cas contraire, elle est dite complète.*
- *L'interprétation I satisfait la formule F (I est un modèle de F) si $I(F) = Vrai$. On note ceci $I \models F$.*
- *L'interprétation I falsifie la formule F (I est un contre modèle de F) si $I(F) = Faux$. On note ceci $I \not\models F$.*
- *Une formule F est consistante si elle admet au moins un modèle. Dans le cas contraire, elle est inconsistante.*
- *Une interprétation I satisfait un ensemble $\mathcal{F}$ de formules si elle satisfait toutes les formules de l'ensemble $\mathcal{F}$. On note ceci $I \models \mathcal{F}$*
- *Un ensemble d'interprétations satisfait une formule F si toutes les interprétations de l'ensemble satisfont la formule F.*

1.1.3 Conséquence logique

Définition 1.6 *Une formule F implique sémantiquement une formule G (noté $F \models G$) si tout modèle de F est un modèle de G. On dit aussi que G est une conséquence sémantique de F.*

Définition 1.7 *Un littéral l est impliqué par une formule F si et seulement si $F \models l$. Dans ce cas le littéral l est vrai dans les modèles de F.*

Exemple 1.1
Soit la formule $F = (\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) = \neg x_1 \wedge (x_2 \vee x_3)$. Le littéral $\neg x_1$ est un littéral impliqué par la formule F. ✍

Théorème 1.1 (déduction) *Soient deux formules F et G. $F \models G$ si et seulement si $F \wedge \neg G$ est une formule inconsistante.*

Le théorème 1.1 est très important pour la déduction automatique. Il montre que prouver l'implication sémantique revient à prouver l'inconsistance d'une formule. La grande majorité des algorithmes de démonstration automatique exploitent ce théorème.

1.1.4 Forme conjonctive normale CNF

Définition 1.8 *On appelle* clause propositionnelle *une disjonction de littéraux.*

On peut représenter les clauses (propositionnelles) comme un ensemble de littéraux. Les clauses sont notées $c_1, \ldots, c_n$. La définition suivante introduit quelques notions utiles concernant les clauses.

Définition 1.9

- *On appelle clause unaire ou mono–littéral une clause ne contenant qu'un seul littéral.*

- *On appelle clause binaire, une clause contenant 2 littéraux.*
- *On appelle clause n–aire une clause contenant exactement n littéraux.*
- *La clause vide, appelée aussi contradiction, est notée $\perp$ ou encore $\square$.*
- *On note $|c|$ le nombre de littéraux de la clause c.*
- *Une clause de Horn est une clause ne contenant qu'un seul littéral positif.*
- *Un littéral pur d'un ensemble de clauses $\mathcal{C}$ est un littéral qui n'apparaît que positivement ou négativement dans $\mathcal{C}$.*
- *Soit $\mathcal{C}$ un ensemble de clauses. On note $L_{\mathcal{C}}$ l'ensemble des littéraux apparaissant dans $\mathcal{C}$.*

Définition 1.10 *Une formule F est dite sous forme normale conjonctive (forme CNF) si et seulement si F est une conjonction de clauses, i.e., une conjonction de disjonctions de littéraux.*

On représente souvent une forme normale conjonctive par l'ensemble des clauses qu'elle contient.

Proposition 1.1 *Toute formule de la logique propositionnelle peut être réécrite sous une forme normale conjonctive.*

Cette transformation peut se faire en temps et en espace (nombre de variables propositionnelles) polynômial. Un algorithme est donné par Pierre SIEGEL dans [87].

Définition 1.11 *La clause c_1 sous–somme ou subsume la clause c_2 si tout littéral de c_1 est un littéral de c_2.*

Lorsque l'on teste la consistance d'une formule F mise sous forme CNF, les clauses sous–sommées peuvent être supprimées car elles sont des conséquences logiques des clauses qui les sous–somment.

Exemple 1.2
Soit les deux clauses $c_1 = (x_1 \vee \neg x_2)$ et $c_2 = (x_1 \vee \neg x_2 \vee x_3)$. La clause c_1 sous–somme la clause c_2. l'interprétation partielle $I = \{x_1\}$ est un modèle de c_1, c'est aussi un modèle de c_2. La réciproque est fausse : l'interprétation $I' = \{x_3\}$ est un modèle de c_2 mais pas de c_1. ✍

Définition 1.12 *Deux clauses c_1 et c_2 se résolvent si et seulement si il existe un littéral l tel que $l \in c_1$ et $\neg l \in c_2$. La clause $((c_1 - \{l\}) \cup (c_2 - \{\neg l\})$ est appelée résolvante en l de c_1 et c_2.*

Nous soulignons maintenant quelques aspects concernant l'aspect sémantique associé aux formules mises sous forme CNF. Une clause c est satisfaite par une interprétation I si au moins un des littéraux de c est vrai dans I. Une formule mise sous forme conjonctive normale est satisfaite si toutes les clauses qui la composent sont satisfaites. Le test de satisfaisabilité d'une formule sous forme CNF par une interprétation est donc simple. Il est à la base du problème SAT que nous introduisons maintenant.

1.1.5 Le problème SAT

Définition 1.13 (le problème SAT) *Soit $\mathcal{V}$ un ensemble de variables propositionnelles et F une formule mise sous forme* CNF*. Le problème* SAT *(forme abrégée du problème de satisfaction d'une formule de la logique propositionnelle mise sous forme normale conjonctive) s'énonce comme suit :*

«Existe–t–il une interprétation des variables de $\mathcal{V}$ qui satisfait la formule F, i.e., qui satisfait l'ensemble des clauses de F ?»

Le problème SAT est très important. De nombreux problèmes sont facilement représentables par la logique propositionnelle et de nombreux algorithmes ont été proposés pour résoudre le problème SAT. De plus, il occupe un rôle central en théorie de la complexité. Il a été le premier a être démontré comme étant un problème NP–complet par Stephen COOK [29]. Le livre de GAREY et JOHNSON [49] fournira au lecteur une étude détaillé de la théorie de la NP–complétude.

Définition 1.14 (k–SAT) *Soit un entier naturel k, k–*SAT *est un problème* SAT *où toutes les clauses contiennent exactement k littéraux.*

3–SAT est la forme la plus simple de SAT qui reste NP–complet. Hormis, 2–SAT, il existe diverses classes de SAT qui sont polynomiales. Citons par exemple , Horn–SAT (problème où toutes les clauses sont des clauses de Horn), r, r–SAT (problème ou les clauses possèdent r littéraux et où chaque variable apparaît au plus r fois).

Bien que le problème SAT appartienne à la classe des problèmes NP–complets, de nombreuses méthodes de résolution ont été proposées et fournissent des résultats satisfaisants dans la pratique. Elles peuvent se départager en deux grandes classes, les méthodes incomplètes et les méthodes complètes.

Méthodes incomplètes

Les méthodes incomplètes utilisent un parcours stochastique de l'ensemble des interprétations en réparant l'interprétation courante de manière à réduire le nombre de clauses non satisfaites. Citons par exemple GSAT [84] une des premières méthodes à base de réparations locales, TSAT [70] également basée sur la réparation locale et utilisant une liste *tabou* des variables afin d'éviter des réparations redondantes. Un état de l'art sur les méthodes de réparation locale est fait dans [54] et dans la thèse de Bertrand MAZURE [69]. Les méthodes incomplètes, bien que très efficaces, ne répondent que partiellement à la question de satisfaisabilité d'une instance SAT. Lorsqu'elles ne parviennent pas à prouver la consistance d'une instance, seules des conjectures concernant la satisfaisabilité peuvent être avancées, ce qui est insuffisant.

Méthodes complètes

Les méthodes complètes examinent la totalité de l'espace de recherche, elles répondent entièrement au problème de satisfaisabilité d'une instance. De nombreux algorithmes existent dans la littérature : programmation linéaire [60, 85], programmation parallèle [54, 94]...Mais la plupart des méthodes complètes sont des améliorations de la procédure de DAVIS et PUTNAM (DP) [35, 34], que nous nous attachons à définir dans la section suivante.

1.2 Algorithmes énumératifs pour SAT

Une des premières méthodes énumératives permettant de résoudre le problème SAT fut proposée par QUINE [79]. Cette méthode, décrite[1] dans l'algorithme 1.1 choisit une variable et l'élimine en effectuant toutes les résolutions possibles avec cette variable.

Algorithme 1.1 Procédure de Quine

Fonction Quine(C : Ensemble de clauses ; V ensemble de variable) : Booléen
Retourne : Vrai si C est satisfaisable, Faux sinon.
Début
Si C contient la clause vide **Alors** Retourner Faux
Si $V = \emptyset$ **Alors** retourner Vrai
Choisir $x \in V$
Retourner Quine($(C \wedge x) \vee (C \wedge \neg x)$,$V - \{x\}$)
Fin

En 1960, Martin DAVIS et Hillary PUTNAM proposaient une amélioration de l'algorithme de QUINE basée également sur le principe de résolution [35]. Étant très coûteuse en mémoire, cette méthode a été très peu étudiée par le passé. Récemment, du fait des capacités grandissantes des ordinateurs, des implantations efficaces ont été proposées. Citons par exemple la méthode ZRES de Philippe CHATALIC et Laurent SIMON [24] et la méthode DR proposée par Irina RISH et Rina DETCHER [81].

La plupart des méthodes basées sur l'algorithme de DAVIS et PUTNAM viennent en fait de l'amélioration de l'algorithme présenté par DAVIS, LOGEMANN et LOVELAND (DLL) en 1962 [34]. C'est donc abusivement que les méthodes basés sur l'algorithme DLL se trouvent dans la littérature nommées sous le nom de DP. C'est à cette version de l'algorithme de DAVIS et PUTNAM que nous nous intéressons.

Nous introduisons maintenant des notations nécessaires pour la suite de la lecture.

[1] Tout au long de ce mémoire nous décrivons les algorithmes sous forme de pseudo–pascal

Notation 1.1 *Soit C un ensemble de clauses et l un littéral. On note $C_{\{l\}}$ l'ensemble de clauses résultant de :*

- *La suppression de toutes les clauses où apparaît le littéral l, c'est–à–dire la suppression de toutes les clauses satisfaites par l.*
- *La suppression du littéral $\neg l$ dans toutes les clauses où il apparaît (clause raccourcie).*

On peut généraliser cette opération à un ensemble de littéraux $\mathcal{L}$, en appliquant ces deux règles à tous les littéraux de $\mathcal{L}$, l'ensemble résultant étant noté $C_{\mathcal{L}}$.

L'algorithme 1.2 décrit la procédure de DP. C'est une procédure de type «backtrack» où les solutions sont construites et évaluées incrémentalement. Elle utilise la propriété des mono–littéraux, caractérisée par la proposition 1.2, et la propriété des littéraux purs, caractérisée par la proposition 1.3. Ces deux propriétés permettent de simplifier l'ensemble de clauses courant.

Algorithme 1.2 Procédure DP

Fonction DP(C : Ensemble de clauses ; I Interprétation) : Booléen
Retourne : Vrai si C est satisfaisable par I, Faux sinon.
Début
Propagation_Unitaire(C, I) %% Algorithme 1.3
Si C contient la clause vide **Alors** retourner Faux
Si $C = \emptyset$ **Alors** retourner Vrai
v = Choisir_Variable() %%Algorithme 1.4
Si DP($C_{\{v\}}, I \cup \{v\}$) = Vrai **Alors** retourner Vrai
Sinon retourner DP($C_{\{\neg v\}}, I \cup \{\neg v\}$)
Fin

Proposition 1.2 *Soit C un ensemble de clauses et l une clause unitaire de C, alors C est satisfaisable si et seulement si $C \wedge \{l\}$ est satisfaisable.*

Proposition 1.3 *Soit C un ensemble de clauses et l un littéral pur de C, alors C est satisfaisable si et seulement si $C \wedge \{l\}$ est satisfaisable.*

Le résultat de la proposition 1.2 permet de faire la propagation unitaire : une clause ne contenant qu'un seul littéral l ne peut être vraie que si le littéral l est vrai. Cette opération donne lieu à la procédure décrite dans l'algorithme 1.3. Le résultat de la proposition 1.3 permet d'affecter à vrai tous les littéraux purs de l'ensemble de clauses. Cette opération réduit le nombre de clauses, mais pas la longueur de celles–ci. Contrairement à la propagation unitaire, la simplification par les littéraux purs n'est pas incluse dans tous les algorithmes basés sur DP. Nous ne donnons pas le schéma algorithmique propageant les littéraux purs, il est semblable à celui de l'algorithme 1.3.

La procédure DP est présentée selon un schéma récursif, les conditions d'arrêt sont soit l'obtention d'un modèle (l'ensemble de clauses devient vide), soit l'obtention de l'inconsistance (détection d'une clause vide dans l'ensemble de clauses). Cette procédure parcourt implicitement un arbre binaire de recherche dont :

- la racine est l'ensemble de clauses de départ
- les branches sont étiquetées par les littéraux affectés.
- Les nœuds sont les ensembles de clauses simplifiés par l'interprétation courante.
- Les feuilles sont étiquetées par la clause vide (échec) ou par l'ensemble vide de clauses (succès).

Algorithme 1.3 Propagation unitaire

Procédure Propagation_Unitaire(Var C : Ensemble de clauses ; Var I Interprétation)
Début
Si($\square \notin C$) et (il existe un mono–littéral l dans C)
Alors
Supprimer toutes les clauses de C ou apparaît l
Supprimer $\neg l$ de toutes les clauses ou il apparaît %%Correspond au calcul de $C_{\{l\}}$
Propagation_Unitaire(C,$I \cup \{l\}$)
Fin

Une des étapes importantes de la procédure de DP est le choix de la prochaine variable à instancier lorsque toutes les simplifications de l'ensemble courant des clauses ont été effectuées (propagation des mono–littéraux et des littéraux purs). Cette étape crée un point de choix dans l'arbre de recherche. Ce sont ces nœuds qui sont généralement comptabilisés lorsque l'on veut connaître la taille de l'arbre exploré par une méthode donnée. La section 1.3 présente quelques unes des heuristiques fréquemment utilisées.

1.3 Heuristiques

La sélection de la variable à interpréter lors d'un point de choix va être un des critères déterminants de l'efficacité d'une procédure de type DP. Le but d'une heuristique de choix est de proposer la meilleure variable possible par rapport à un critère donné. Une bonne heuristique permet de créer les sous-problèmes les plus simples possibles. L'algorithme 1.4 donne le schéma général d'une procédure de choix heuristique.

w_C est la fonction de poids. Elle est dépendante de l'ensemble courant des clauses. L'acuité de la variable sélectionnée dépend de la complexité plus ou moins importante de la fonction de poids. H est la fonction d'évaluation. Elle admet généralement en arguments le poids d'une variable et de

Algorithme 1.4 choisir_Variable

Fonction Choisir_Variable(C : ensemble de clause)
Retourne : Une variable non instanciée
Début
Pour Chaque Variable v de C **faire** Score(v) = $H(w_C(v), w_C(\neg v))$
Retourner la variable de score le plus élevé.
Fin

son opposé. Cette contrainte équilibre l'arbre de recherche. La procédure `Choisir_Variable` qui est proposée dans le schéma l'agorihmique 1.4 retourne une variable. Il est tout à fait possible qu'elle renvoie un littéral, afin de commencer par parcourir le «meilleur» des deux sous–arbres associé à la variable. Bien sûr, cette alternative n'est intéressante que pour pour des instances satisfaisables.

Nous décrivons d'abord les heuristiques de choix de type MOM'S, puis celles de type UP.

1.3.1 Heurisitique de type «Maximum Occurences in Minimun Size Clauses»

Beaucoup d'heuristiques utilisent la stratégie dite de l'«échec d'abord» («first fail» en anglais). Elle consiste à sélectionner la variable qui permet d'atteindre le plus rapidement possible l'inconsistance, ce qui a pour effet de réduire la profondeur de l'arbre de recherche. Le nombre d'occurences d'un littéral dans l'ensemble de clauses est déterminant pour les heuristiques de type «échec d'abord». Plus un littéral apparaît, plus il y a de clauses satisfaites lorsqu'il est affecté et plus il y a de clauses raccourcies lorsqu'il est falsifié (branche complémentaire). L'apparition dans les clauses binaires est également favorisé. Le raccourcissement de clauses binaires implique l'apparition de nouveaux mono–littéraux et donc de nouvelles propagations unitaires. Ce principe est connu sous le nom d'heuristique MOM'S «Maximum Occurences in Minimum Size clauses» (maximum d'occurences dans les clauses les plus petites) [41, 43, 57].

Nous décrivons maintenant la fonction de poids proposé par Robert JEROSLOW et Jinchang WANG [57] (noté JW). Elle calcule le poids d'un littéral l en fonction de la probabilité que l'interprétation falsifie les clauses où apparaît l.

Soient n le nombre de variables propopositionnelles d'un l'ensemble de clauses C et $c \in C$. Le nombre total d'interprétations est 2^n. Le nombre d'interprétations qui falsifient la clause c est $2^{n-|c|}$. La probabilité qu'une interprétation falsifie la clause c est donc $\frac{2^{n-|c|}}{2^n} = \frac{1}{2^{|c|}}$. On en déduit alors la fonction de poids du littéral l, et on a :

$$w_C(l) = \sum_{c \in C \mid l \in c} \frac{2^{n-|c|}}{2^n} = \sum_{c \in C \mid l \in c} \frac{1}{2^{|c|}}$$

L'heuristique proposée par JEROSLOW et WANG consiste simplement à affecter le littéral qui maximise cette fonction de poids. John HOOKER et B. VINAY ont proposé d'autres heuristiques

associées à cette fonction de poids [55]. Par exemple, l'heuristique «Two–Sided JEROSLOW WANG rule» qui choisit la variable v qui maximise $w_C(v) + w_C(\neg v)$.

Dans la section 1.5, nous introduisons d'autres heuristiques de type MOM'S. Nous introduisons maintenant les heuristiques de type UP qui ont été proposés plus récemment.

1.3.2 Heuristiques de type UP

Les heuristiques UP (pour «Unit Propagation») [43, 65, 78] utilisent le pouvoir de la propagation unitaire afin de déterminer plus efficacement la prochaine variable à instancier. Soient x une variable et C l'ensemble de clauses courant. On calcule `Propagation_Unitaire`($C_{\{x\}}$) (resp. `Propagation_Unitaire`($C_{\{\neg x\}}$)). Le poids du littéral x (respectivement $\neg x$) dépend alors de $C_{\{x\}}$ (respectivement $C_{\{\neg x\}}$). L'utilisation de la propagation unitaire dans le choix heuristique possède un double avantage :

- La meilleure variable n'est pas déterminée en fonction de l'ensemble courant de clauses, mais en fonction de ce qu'il deviendrait si celle–ci était affectée.
- Indépendamment, si l'une ou l'autre des propagations unitaires induisent la clause vide alors on considère l'autre littéral comme unitaire. Nous développons cette notion de littéral impliqué dans le chapitre 2.

Cette heuristique possède un coût calculatoire élevé, les méthodes qui l'exploitent réduisent son utilisation à un sous–ensemble des variables non instanciées ou à la partie haute de l'arbre de recherche.

La stratégie qui consiste à propager différentes contraintes en un nœud donné est connue sous le nom de «look–ahead» (que l'on peut traduire par «voir plus loin»). Elle est également utilisée pour mettre en valeur des contraintes cachées.

1.4 Instances difficiles

A l'opposé des instances SAT structurées issues de la planification, de la preuve de circuits, il existe des instances générées aléatoirement. Ces instances aléatoires sont intéressantes car les méthodes énumératives présentent des difficultés pour les résoudre.

Une instance k–SAT aléatoire est définie par son nombre de variables v et son nombre de clauses c. On détermine alors cette instance en générant aléatoirement les c clauses parmi les $2^k\binom{v}{k}$ possibles. De nombreuses études [32, 51] ont été menées sur ces instances et ont fait apparaître un phénomène de seuil. C'est sur certaines propriétés de graphes aléatoires, tel que le problème de la connexité, qu'ont été découverts les premiers phénomènes de seuil en informatique.

Un phénomène de seuil est observé quand la probabilité qu'une propriété soit satisfaite passe de presque 0 à presque 1 en fonction de l'évolution de certains paramètres. De plus le passage de 0 à 1

se fait brusquement. Le seuil est caractérisé par les valeurs des paramètres qui délimitent la zone de transition pour des instances de taille infinie.

Le seuil apparaît sur les problèmes k–SAT lorsque l'on regarde la probabilité qu'une instance soit satisfaite en fonction du rapport $\frac{c}{v}$. La figure 1.1 montre ce phénomène pour le problème 3–SAT. Nous observons que lorsque $\frac{c}{v} \leq 4$ les instances sont quasiment toutes satisfaisables et que pour $\frac{c}{v} \geq 4.7$ les instances sont presque toutes insatisfaisables. On observe également que la rapidité de la transition augmente avec l'augmentation du nombre de variables.

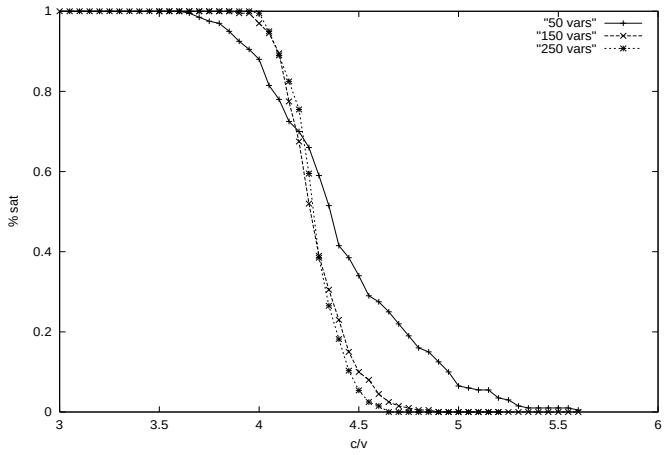


Figure 1.1. *Phénomène de seuil pour 3–*SAT *aléatoires.*
Moyenne sur 200 instances résolues par un algorithme DP *classique*

Le théorème 1.2 formalise la notion de seuil pour les problèmes k–SAT aléatoires. Il a été introduit par Ehud FRIEDGUT [46, 47] qui a donc prouvé que la fenêtre de la phase de transition tend vers 0 lorsque le nombre de variables tend vers l'infini. Par contre, il n'a pas prouvé que la valeur du seuil était fixe, celle–ci peut donc osciller en fonction du nombre de variables. Les meilleures bornes connues délimitant le seuil de 3–SAT sont 3.26 comme borne inférieure [1] et 4.506 comme borne supérieure [40]. Le théorème 1.3, prouvé par CHVÀTAL et REED [26], donne la valeur du seuil pour 2–SAT.

Théorème 1.2 *Soit $S_k(v, r)$ la probabilité qu'une instance k–*SAT *de v variables et de $r \times v$ clauses soit consistante. Pour $k \geq 2$, il existe une fonction $r_k(v)$ tel que quelque soit $\epsilon > 0$,*

$$\lim_{v \to \infty} S_k(v, r_k(v) - \epsilon) = 1 \quad \textit{et} \quad \lim_{v \to \infty} S_k(v, r_k(v) + \epsilon) = 0$$

Théorème 1.3 *Pour 2–*SAT*, la valeur du seuil est égale à 1 (autrement dit $r_k(v) = 1$).*

Les expérimentations ont également mis en avant que la difficulté de résolution des instances k–SAT aléatoires n'est pas uniforme [27]. Un pic de difficulté est situé au niveau du seuil, là où la

probabilité d'avoir une instance consistante est de $\frac{1}{2}$. La figure 1.2 illustre ce phénomène. La difficulté est mesurée en fonction du nombre d'appels à la procédure `Choisir_Variable` de la fonction DP (algorithme 1.2).

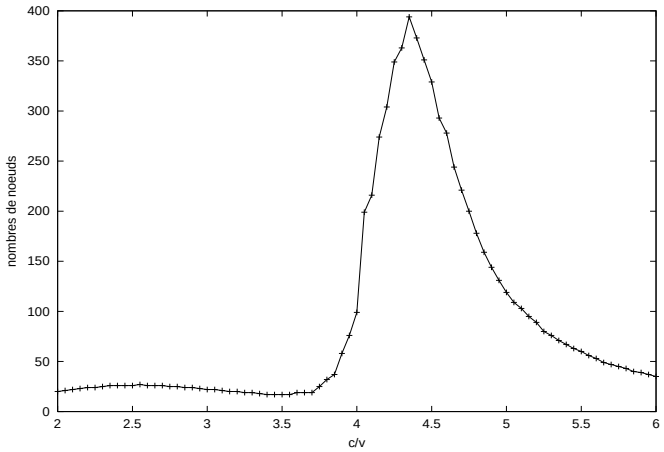


Figure 1.2. *Difficulté des instances 3–*SAT *aléatoires (200 variables) Moyenne sur 200 instances résolues par un algorithme* DP *classique*

L'intérêt de telles instances est la facilité avec laquelle on les génère. On peut les utiliser pour expérimenter de nouvelles méthodes et heuristiques. De plus, des méthodes efficaces sur ces instances se révèlent généralement efficaces sur des instances structurées.

En 1997, SELMAN et al. ont proposé une liste de 10 challenges pour la résolution de problèmes SAT [83]. Le premier s'énonce ainsi : «Prouver l'inconsistance d'une instance 3–SAT aléatoire de 700 variables situé dans la région du seuil (en temps raisonnable)». A ce jour, ce challenge n'a toujours pas été relevé ! Actuellement, les meilleures méthodes complètes pour 3–SAT sont capables de résoudre des instances de 500 variables en temps raisonnable (moins de deux heures)[2]. Nous décrivons d'ailleurs certaines de ces méthodes dans la prochaine section.

1.5 Quelques méthodes énumératives

Nous allons présenter quelques méthodes basées sur l'algorithme de DAVIS et PUTNAM, qui, dans la littérature, sont parmi les plus efficaces. Bien entendu, il est impossible de dresser une liste qui soit exhaustive. Citons, par exemple les méthodes GRASP [88], tableau [32], RELSAT [11], zChaff [75] qui donnent également de très bons résultats.

[2] A l'heure où ces lignes sont écrites, Olivier DUBOIS et Gilles DEQUEN proposent une nouvelle heuristique resolvant des instances difficiles de 700 variables en 26 jours [42]

1.5.1 Le système POSIT

La méthode POSIT est réalisée par FREEMAN [43, 44]. Elle inclut un pré–traitement, qui permet ensuite de garantir que chaque variable apparaît au moins 3 fois positivement et au moins 3 fois négativement. La procédure de choix de la variable utilise un mélange d'heuristique UP et une généralisation de l'heuristique de JEROSLOW et WANG. En effet, la fonction de poids d'un littéral l proposé par FREEMAN est :

$$w(l) = \sum_{c \in Cl \in c} \frac{1}{\gamma^{|c|}}$$

Où γ est une constante. Plus γ est grand plus l'apparition des littéraux dans des clauses de petites tailles prend de l'importance. FREEMAN a fixé γ à 5. Ainsi, l'apparition d'un littéral dans une clause de taille t équivaut à 5 apparitions dans des clauses de taille $t+1$.

L'heuristique de branchement comporte 4 étapes :

- Calculer le score de chaque variable. Créer le sous–ensemble S des variables de plus grand score. La fonction d'évaluation donnée par FREEMAN est : $H(x) = 1024 \times w(x) \times w(\neg x) + w(x) + w(\neg x)$.
- Utiliser l'heuristique UP.
- Supprimer de S les variables ayant eu un mauvais score avec la propagation unitaire.
- Choisir la variable de S maximisant l'heuristique de JW généralisée.

1.5.2 Le système SATZ

La méthode SATZ a été proposée par Chu Min LI et AMBULAGAN [65, 66]. A ce jour, SATZ est l'un des meilleurs algorithmes de résolution des instances aléatoires difficiles situé dans la région du seuil. Chu Min LI vient de proposer une nouvelle version, baptisée SATZ123 [63] qui améliore encore les performances sur ces instances.

La méthode SATZ inclut :

- Un pré–traitement qui crée des résolvantes de taille inférieure ou égale à 3. Chaque résolvante créée peut servir à nouveau pour créer d'autres résolvantes, le processus étant maintenu jusqu'à saturation. Deux conditions sont également imposées :
  - La résolvante de 2 clauses binaires n'est créée que si elle est unaire.
  - La résolvante d'une clause binaire et d'une clause ternaire n'est créée que si elle est binaire.
- Une heuristique de type UP

Les variables candidates à l'heuristique sont celles qui vérifient le prédicat $Prop_z$ que nous définissons ci–dessous :

Définition 1.15 *Soit x une variable et C l'ensemble de clauses courant. On note $Prop(x, i)$, le prédicat vrai lorsque x apparaît positivement et négativement dans les clauses binaires et apparaît au moins*

i fois dans ces clauses binaires. On note alors le prédicat $Prop_z(x)$ le premier de ces trois prédicats qui sélectionne au moins 10 variables.

1. $Prop(x, 4)$
2. $Prop(x, 3)$
3. $Vrai$

Le prédicat $Prop_z$ a été déterminé empiriquement. Il permet de satisfaire un critère important. En effet, le nombre de variables qui vérifient le prédicat $Prop_z$ est d'autant plus grand que l'on se situe en haut de l'arbre de recherche. C'est une caractéristique intéressante puisque c'est dans cette partie que le choix des variables est le plus important. La fonction poids d'un littéral l est donnée par le nombre de clauses binaires créees par la propagation unitaire de l dans C. La fonction d'évaluation H est celle proposée par FREEMAN [44].

La nouvelle version de SATZ [63] diffère de l'ancienne par la possibilité de pouvoir réaliser de la propagation unitaire sur deux niveaux (sur deux littéraux consécutifs). Si les poids $w(x)$ et $w(\neg x)$ sont d'ordre de grandeur différent, la fonction d'évaluation H ne sélectionne pas cette variable comme prochaine variable à instancier (symétrie de l'arbre de recherche). Cependant, si $w(x)$ est beaucoup plus grand que $w(\neg x)$ alors il est probable que $C_{\{x\}}$ soit inconsistant. La méthode SATZ123 tente de démontrer l'inconsistance de cet ensemble de clauses en propageant les variables (noté y) qui apparaissent positivement et négativement dans les nouvelles clauses binaires de $C_{\{x\}}$. En d'autres termes, on calcule $C_{\{x,y\}}$ et $C_{\{x,\neg y\}}$. Si ces deux ensembles de clauses sont inconsistants alors $C_{\{x\}}$ l'est aussi et on peut ainsi affecter $\neg x$ comme un littéral unitaire.

1.5.3 Le système SATO

SATO a été réalisé par Hantao ZHANG et Mark STICKEL [100, 101]. Les auteurs se sont focalisés sur la structure de données à utiliser pour coder efficacement les clauses. Elles sont codées sous forme d'arbres. C'est John DE KLEER qui a utilisé cette structure en premier pour le problème SAT [36].

Chaque nœud de l'arbre est soit :

– L'arbre vide, noté par *nil*
– La marque d'obtention de clause, notée $\Box$
– Un quadruplet $< v, p, n, r >$, où v représente l'index d'une variable, p un arbre représentant les clauses où v apparaît positivement, n un arbre représentant les clauses où v apparaît négativement et r un arbre représentant les clauses où n'apparaît pas v.

Les variables sont triées par ordre lexicographique de la racine vers les feuilles. L'exemple 1.3 montre une telle structure.

Exemple 1.3

Soit l'ensemble de clauses $\{(x_1 \vee x_2), (\neg x_1 \vee \neg x_2)\}$ L'arbre codant les clauses est

$$< x_1, < x_2, \square, nil, nil >, < x_2, nil, \square, nil >, nil >$$

et est explicité par la figure 1.3.

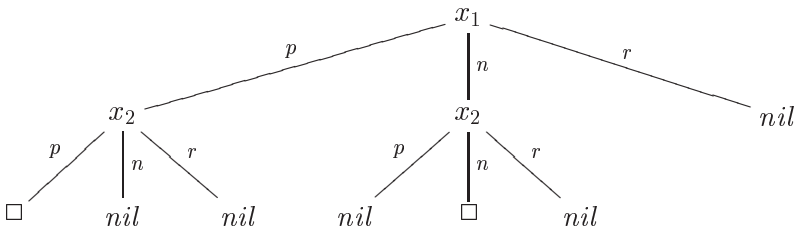


Figure 1.3. *Exemple d'arbre de clauses avec* SATO

Un algorithme qui fusionne deux arbres est donné. Son association avec cette structure de données procure les avantages suivants :

- Suppression des clauses dupliquées pendant la construction de l'arbre.
- Suppression des clauses sous-sommées pendant la construction de l'arbre.
- Calcul rapide de certaines résolvantes.
- Propagation unitaire efficace. Étant donné que les variables sont triées lexicographiquement, un seul parcours de l'arbre est nécessaire pour toute une phase de propagation. Pour supprimer les clauses satisfaites par un littéral l il suffit de mettre à *nil* les sous–arbres correspondant à l vrai. La recherche des nouveaux mono–littéraux se fait uniquement dans les sous–arbres correspondant au littéral $\neg l$.

En plus de cette structure de données, les auteurs proposent un système de saut arrière («backjumping» ou «look–back» en anglais) afin de mémoriser les anciens conflits et d'éviter des affectations connues pour être vouées à l'échec. Il est à noter que dans son article intitulé «Look-Ahead versus Look-Back for satisfiability problems», Chu Min LI [66] compare les techniques de «look–ahead» et de «look–back». Pour lui, la technique du saut arrière est inutile si les variables sont affectées dans le "bon ordre".

1.5.4 Le système CSAT

La méthode CSAT a été proposée par Yacine BOUFKHAD et Olivier DUBOIS, les détails peuvent être obtenus dans [41, 18]. Initialement, les auteurs ont proposé plusieurs méthodes, spécialisées soit dans la recherche d'une solution soit dans la preuve d'instances insatisfaisables. CSAT est plus spécialisée dans la preuve d'inconsistance mais obtient également de bons résultats pour les instances satisfaisables.

Tout d'abord CSAT utilise un pré–traitement qui intègre :

- La recherche de symétries de bases. Une étude des divers algorithmes de détection et d'utilisation des symétries est effectuée dans le chapitre 6.

– Le calcul de résolvantes. Seules les résolvantes de taille inférieure ou égale aux clauses qui se résolvent sont créees.

L'heuristique de branchement est calculée grâce à la fonction d'évaluation suivante : $H(x) = 1.5 \times min(F(x), F(\neg x)) + F(x) + F(\neg x)$, où $F(l)$ dépend du poids des opposés des littéraux inclus dans les clauses binaires où apparaît l. La fonction de poids est égale à :

$$w(l) = \sum_{l \in c} - \ln(1 - \frac{1}{(2^{|c|} - 1)^2})$$

Les auteurs proposent également une étape dite de «local processing» exécutée avant un point de choix. Elle consiste à propager un littéral, et, si la clause vide apparaît, à traiter son opposé comme un mono–littéral. Cette étape est limitée à un sous–ensemble de littéraux apparaissant dans les clauses binaires.

1.5.5 Vers des algorithmes ciblés

Récemment, diverses implantations ciblées de DAVIS et PUTNAM ont été réalisées dans le but de pouvoir résoudre des instances particulières du problème SAT.

Un des challenges proposés par SELMAN et al. dans [83] concernait la résolution des instances *par32* du challenge DIMACS [38] en un temps raisonnable. Ces instances comportent un grand nombre de clauses d'équivalences. Chu Min LI a réalisé EQSATZ [64] pour tenter de relever ce challenge. EQSATZ est une version de SATZ complétée par une recherche de clauses d'équivalence et par des règles de simplification sur ces clauses.

Définition 1.16 (clause d'équivalence) *Soit $l_1 \ldots, l_k$ k littéraux. Une clause d'équivalence de longueur k peut être écrite comme suit :*

$$l_1 \leftrightarrow l_2 \leftrightarrow \ldots \leftrightarrow l_k$$

La transformation sous forme CNF d'une clause d'équivalence de longueur k nécessite 2^{k-1} clauses CNF. Par exemple, $l_1 \leftrightarrow l_2$ est équivalente aux deux clauses $\{(l_1 \vee \neg l_2), \neg(l_1 \vee l_2)\}$. Ici, le codage sous forme CNF accroît exponentiellement le nombre de clauses nécessaires. De plus, les équivalences qui peuvent apparaître pendant la recherche entre les différents littéraux sont cachées par la forme CNF et ne sont donc pas exploitées. Chu Min LI souligne que dans le cas de clauses d'équivalence transformées en CNF très peu de mono–littéraux sont propagés, ce qui explique la difficulté des méthodes classiques pour résoudre les problèmes *par32*.

D'autres implantations de DAVIS et PUTNAM sur des formes normales généralisées ont été réalisées. Citons Peter BAUMGARTNER et Fabio MASSACCI [10] qui ont proposé une version de DAVIS et PUTNAM traitant des formules qui contiennent des ou exclusifs. Ces clauses apparaissent, entre

autres, dans des problèmes de vérification, de planification. Citons encore, la méthode BCSAT proposée par Tommi A. JUNTTILA et Ilkka NIEMELÄ [59] qui représente les formules par des circuits booléens. Un circuit booléen est un graphe orienté acyclique où les nœuds sont les portes et les feuilles les variables propositionnelles.

Ces méthodes ont vu le jour pour plusieurs raisons. Tout d'abord, il est difficile de concevoir une méthode qui soit efficace pour toutes les instances SAT. La section 3.5 met ce point en avant. De plus, la transformation sous forme SAT fait perdre les propriétés intrinsèques des problèmes de départ. On ne peux pas profiter de sa structure pour améliorer le choix heuristiques des variables à instancier. Nous rencontrerons à nouveau ces difficultés lorsque nous voudrons générer des modèles finis de théories du premier ordre (partie 2).

1.6 Conclusion

Ces dernières années, les problèmes aléatoires ont eu une grosse influence sur les recherches ayant trait au problème SAT. Il y a eu, d'un coté, des recherches théoriques cherchant à déterminer la valeur du seuil pour 3–SAT ; et de l'autre, des recherches pratiques avec la proposition d'algorithmes de résolution de plus en plus efficaces.

C'est ce coté pratique qui nous intéresse le plus et qui nous amène à nous poser certaines questions. Pour quelles raisons le premier challenge de Bart SELMAN [83] n'a–t–il toujours pas été relevé ? Comment réduire le nombre de nœuds nécessaires pour résoudre des instances aléatoires difficiles ? C'est pour répondre à ce type de questions que nous proposons dans le chapitre suivant une méthode énumérative. Son but est de propager le plus de mono–littéraux possible afin d'arriver le plus rapidement possible à l'inconsistance.

Chapitre 2

La méthode AVAL

NOUS INTRODUISONS DANS CE CHAPITRE une méthode énumérative basée sur l'algorithme de DAVIS et PUTNAM. Cette méthode améliore la procédure classique de DP en essayant de maximiser les propagations unitaires faites pendant la recherche. Pour ce faire, un algorithme de production de littéraux est implanté. Il est appelé dès que l'ensemble courant de clauses ne contient plus de mono–littéraux. La combinaison de cet algorithme de production de littéraux et de la méthode DP donne lieu à la méthode AVAL. Cette appellation est due à l'observation d'un phénomène d'avalanche de mono–littéraux produits lors de la résolution de problèmes aléatoires situé dans la région du seuil. En essayant de faire apparaître à faible coût des mono–littéraux cachés, notre méthode minimise le nombre de points de choix et donne lieu à un algorithme robuste moins sensible aux heuristiques. Les travaux de cette partie ont été publiés aux «journées nationales de résolution pratique des problèmes NP–complets» JNPC99 [5] et à «International conference on Computational Logic» CL00 [6].

2.1 Production des mono–littéraux

Après la propagation de toutes les clauses unitaires et de tous les littéraux purs, la procédure DP choisit et affecte une variable, créant un point de choix dans l'arbre de recherche. Seulement, il est probable que des littéraux puissent être impliqués (voir la définition 1.7) par l'ensemble de clauses courant. Nous pouvons considérer ces littéraux comme des mono–littéraux «cachés». Si nous arrivons à les détecter, nous évitons certains points de choix dans l'arbre de recherche. Pour essayer de produire un littéral, il suffit de tester l'inconsistance d'un ensemble de clauses (théorème de déduction 1.1). Cette opération nécessite une surcharge de travail, elle est donc restreinte : le test de consistance est réduit à la propagation unitaire et seuls les littéraux apparaissant dans les clauses binaires sont candidats à la production. Ces derniers ont le plus de chance d'être produits.

Soient I l'interprétation courante et C_I l'ensemble des clauses simplifiées par I (voir la notation 1.1). Pour montrer que l est impliqué par C_I ($C_I \models l$), nous devons prouver l'inconsistance de

$C_I \wedge \{\neg l\}$. Deux cas sont alors possibles :

- Soit $C_I \wedge \{\neg l\} \models \square$ et dans ce cas l est produit par C_I et peut être considéré comme un mono–littéral.
- Soit $C_I \wedge \{\neg l\} \not\models \square$, alors l ne peut être produit par C_I. Mais cet échec met en valeur des littéraux qui ne peuvent pas être impliqués par l'ensemble de clauses C_I. Il est donc inutile de les considérer pour la production.

L'efficacité de notre algorithme de recherche des littéraux impliqués est due en grande partie à l'élimination de ces variables inutiles. Formellement, nous avons :

Proposition 2.1 *Soit B_I l'ensemble des clauses binaires de C_I et L_{B_I} l'ensemble des littéraux de B_I. Soit $l \in L_{B_I}$, Si $C_I \cup \{\neg l\} \models a_{i\in\{1,\ldots,n\}}$ et $C_I \cup \{\neg l\} \not\models \square$ alors*

$$\forall i \in \{1,\ldots,n\} \textit{ tel que } a_i \in L_{B_I},\ C_I \cup \{a_i\} \not\models \square$$

Autrement dit, $\forall i \in \{1,\ldots,n\} \quad C_I \not\models \neg a_i$.

Preuve Soit $l \in L_{B_I}$, tel que $C_I \cup \{\neg l\} \models a_{i\in\{1,\ldots,n\}}$ et $C_I \cup \{\neg l\} \not\models \square$. Supposons qu'il existe $j \in \{1,\ldots,n\}$ tel que $C_I \cup \{a_j\} \models \square$ ($C_I \models a_j$). On a alors $C_I \cup \{\neg l\} \cup \{a_j\} \models \square$. Mais $C_I \cup \{\neg l\} \cup \{\neg a_j\} \models \square$, et donc $C_I \cup \{\neg l\} \models \square$. Ce qui est en contradiction avec l'hypothèse.
Par conséquent $\forall i \in \{1,\ldots,n\} \quad C_I \not\models \neg a_i$. ❑

Les littéraux $\neg a_1, \ldots, \neg a_n$, opposés des littéraux propagés durant l'échec de production de l, ne peuvent pas être des conséquences logiques de C_I. Il est donc hors de propos de les considérer comme candidats à la production. Le résultat de la proposition 2.1 est utilisé dans l'algorithme de recherche des littéraux impliqués, noté RLI, (algorithme 2.1). Le vecteur `Candidat[`l`]=Vrai` exprime le fait que le littéral l est candidat à la production. L'étape d'initialisation consiste à considérer comme candidats tous les littéraux apparaissant dans les clauses binaires. Ensuite, nous choisissons un candidat potentiel et propageons son opposé en mettant à jour la marque `candidat` des opposés de littéraux propagés. Et recommençons le processus si cette propagation n'a pas impliqué la clause vide. Nous montrons le fonctionnement de l'algorithme RLI dans l'exemple 2.1.

Exemple 2.1
Soit l'ensemble de clauses $C = \{(x_1 \vee \neg x_2 \vee \neg x_3), (x_1 \vee x_2), (\neg x_2 \vee x_3)\}$. L'ensemble des littéraux apparaissant dans les clauses binaires est $L_B = \{x_1, x_2, \neg x_2, x_3\}$. L'état du vecteur `candidat` est le suivant :

Littéral	x_1	x_2	$\neg x_2$	x_3
Candidat	Vrai	Vrai	Vrai	Vrai

Si nous essayons de produire x_3, nous obtenons $C \cup \{\neg x_3\} \models \neg x_2, x_1$ et $C \cup \{\neg x_3\} \not\models \square$. D'après la proposition 2.1, nous avons $C \cup \{\neg x_2\} \not\models \square$ et $C \cup \{x_1\} \not\models \square$, les littéraux x_3, x_2 et $\neg x_1$ ne peuvent être produits par C. Le vecteur `candidat` devient :

Algorithme 2.1 Recherche des littéraux impliqués (RLI)

Fonction RLI(C : ensemble de clauses ; I : interprétation)
Retourne : un littéral l si $C \models l$
0 si $\nexists l \in L_{B_I}$ tel que $C \models l$

Début
Pour tout $l \in L_{B_I}$ **Faire** Candidat[l] = Vrai
Pour tout $l \in L_{B_I}$ tel que Candidat[l] = Vrai
Faire %%Essayer d'impliquer l
Candidat[l] = Faux
$I' = I \cup \{\neg l\}$
Tant que ($C_{I'} \neq \emptyset$) et ($\square \notin C_{I'}$) et ($\exists x \in L_{C_{I'}}$ tel que x est un mono–littéral)
Faire %% Propagation unitaire
$I' = I' \cup \{x\}$
Candidat[$\neg x$]=Faux
Si ($\square \in C_{I'}$) **Alors** Retourner l %%l est implqiué par C
Retourner 0
Fin

Littéral	x_1	x_2	$\neg x_2$	x_3
Candidat	Vrai	Faux	Vrai	Faux

Nous essayons de produire x_1, on a alors $C \cup \{\neg x_1\} \models x_2, x_3, \square$. Donc x_1 est un littéral impliqué par C et la procédure RLI peut le retourner comme mono–littéral. ✍

L'algorithme RLI déduit un littéral parmi ceux apparaissant dans les clauses binaires. Sa terminaison, sa correction et sa complétude sont données par les propositions suivantes.

Proposition 2.2 *Soit C l'ensemble des clauses courant , et l un littéral apparaissant dans une clause binaire. Si* RLI *produit l (* RLI$(C) \models l$*) alors $C \equiv C \cup \{l\}$.*

Preuve Soit l un littéral produit par RLI. Alors, $(C \cup \{\neg l\}) \models \square$.
Mais $C \equiv (C \cup \{l\}) \vee (C \cup \{\neg l\})$. On a donc $C \equiv C \cup \{l\}$ et la correction de RLI est prouvée. ❑

Proposition 2.3 *Si C_I est l'ensemble courant de clauses et B_I le sous ensemble de clauses binaires, alors l'algorithme* RLI *termine et sa complexité dans le pire des cas est en*

$$O(|L_{B_I}| \times |L_{C_I}|)$$

Preuve Quand RLI essaie de produire un littéral l ($C_I \models l$?), elle propage dans le pire des cas L_{C_I} mono–littéraux. Si l n'est pas productible ($C_I \not\models l$), la procédure RLI essaie de produire un autre

littéral parmi ceux apparaissant dans L_{B_I}. Le pire des cas intervient alors quand tous les littéraux propagés par l'échec de production de l ($C_I \not\models l$) n'appartiennent pas à L_{B_I}.
Dans ce cas RLI essaie de produire tous les littéraux de L_{B_I}. Donc, la complexité dans le pire des cas est en $O(|L_{B_I}| \times |L_{C_I}|)$. ◻

La méthode CSAT [41] inclue une étape dite de «local processing». Durant cette étape, une recherche de littéraux impliqués par l'ensemble de clauses est faite de manière similaire. Pour plus d'efficacité les auteurs restreignent cette recherche à un sous–ensemble de littéraux apparaissant dans les clauses binaires (ceux qui ont le plus d'occurences). Ils ne prennent pas en compte que certains candidats (littéraux de la proposition 2.1) ne peuvent être impliqués, bien que Yacine BOUFKHAD ait remarqué la présence de ces littéraux inutiles dans sa thèse [18].

2.2 La méthode AVAL

La combinaison de l'algorithme de recherche des littéraux impliqués (RLI), décrit précédemment, et de la procédure de DAVIS et PUTNAM définit la méthode énumérative AVAL.

La différence entre AVAL et DP est due à la recherche de littéraux impliqués lorsque l'ensemble de clauses courant ne contient ni de littéraux purs ni de mono–littéraux explicites. En propageant les littéraux impliqués, la méthode AVAL évite certains points de choix que DP parcourt. De la même manière que l'algorithme DP est une amélioration de l'algorithme de QUINE par l'ajout des propriétés de littéraux purs et unaires, la méthode AVAL peut être considérée comme une amélioration de l'algorithme de DAVIS et PUTNAM par l'ajout de la propriété des littéraux impliqués.

Quand l'algorithme RLI réussit à produire un littéral, celui–ci est considéré comme un mono–littéral, son affectation supprime un nœud de l'arbre de recherche. En cas d'échec dans la production, nous choisissons, à l'aide d'une heuristique, la prochaine variable à instancier, ce qui crée un point de choix dans l'arbre de recherche. L'exploitation de littéraux produits par RLI tend à minimiser le nombre de ces points de choix pour une heuristique donnée. L'algorithme 2.2 schématise la méthode AVAL. De même que pour l'algorithme de DAVIS et PUTNAM, le premier appel à AVAL est initialisé avec $I = \emptyset$ et $C = C_\emptyset$. L'exemple 2.2 montre le fonctionnement de la méthode AVAL.

Exemple 2.2
Soit C l'ensemble de clauses suivant :

$$\begin{array}{llll}
\{(\neg x_2, x_3, x_4), & (\neg x_2, \neg x_3, \neg x_4), & (x_1, x_3, x_4), & (x_2, x_3, \neg x_4), \\
(\neg x_1, \neg x_2, x_3), & (\neg x_1, x_2, \neg x_3), & (x_1, x_2, \neg x_3), & (\neg x_1, \neg x_2, \neg x_3), \\
(\neg x_1, \neg x_3, x_4), & (\neg x_1, \neg x_3, \neg x_4), & (x_1, \neg x_3, x_4), & (\neg x_2, x_3, \neg x_4), \\
(x_1, x_2, \neg x_3), & (\neg x_1, \neg x_2, \neg x_3), & (\neg x_1, \neg x_2, \neg x_4)\} &
\end{array}$$

La figure 2.1 montre le comportement des méthodes DP et AVAL lors de la recherche de la satisfaisabilité de l'ensemble de clauses C. Le symbole «$\bot$» désigne l'inconsistance, «$\top$» désigne l'ob-

Algorithme 2.2 La méthode AVAL

Fonction AVAL(C : Ensemble de clauses ; I Interprétation) : Booléen
Retourne : Vrai si C est satisfaisable par I, Faux sinon.
Début
Propagation_Unitaire(C, I) %% Algorithme 1.3
Si C contient la clause vide **Alors** retourner Faux
Si $C = \emptyset$ **Alors** retourner Vrai
q=rli() %% Algorithme 2.1 ou 2.3
Si $q \neq 0$ **Alors** Retourner AVAL($C_{\{q\}}, I \cup \{q\}$) %%Littéral impliqué
v = Choisir_Variable() %% Algorithme 1.4
Si AVAL($C_{\{v\}}, I \cup \{v\}$) = Vrai **Alors** retourner Vrai
Sinon retourner AVAL($C_{\{\neg v\}}, I \cup \{\neg v\}$)
Fin

tention d'un modèle. L'algorithme RLI détecte deux littéraux impliqués (indiqué par (rli)) sur deux tentatives . L'arbre généré par AVAL est sensiblement plus petit et ne contient qu'un seul nœud alors que celui de DP en contient 3.

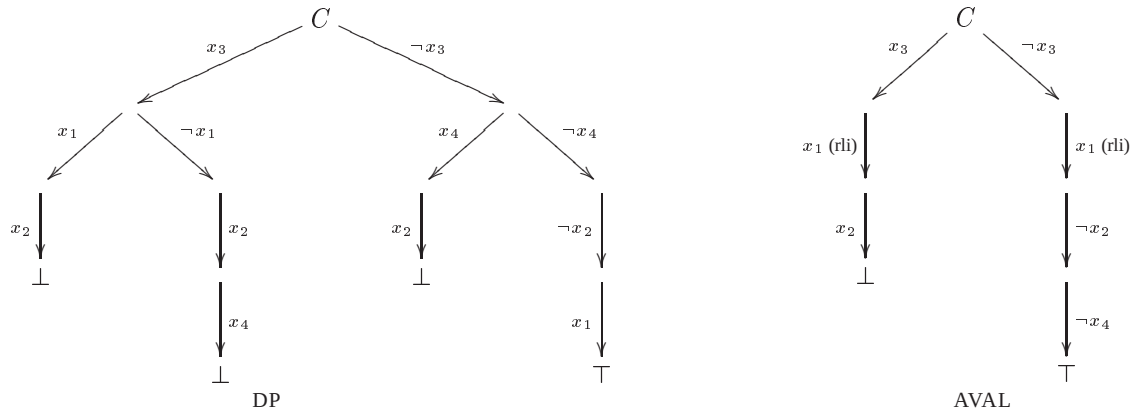


Figure 2.1. *Arbres de recherche générés par les méthodes* DP *et* AVAL

Dans la section suivante, nous donnons une nouvelle version de l'algorithme RLI qui prend en compte le fait que la méthode AVAL l'appelle à chaque nœud de l'arbre de recherche.

2.3 Une version dynamique de RLI

L'algorithme classique RLI essaie de produire un littéral à chaque nœud de l'arbre de recherche en utilisant la propagation unitaire. Il n'évite alors pas un certain travail inutile. Entre deux nœuds consé-

cutifs, la structure de l'ensemble des clauses ne change que relativement peu : peu de clauses binaires sont créees et peu sont supprimées. Dés lors, il est probable de recommencer les mêmes propagations vouées à l'échec qui vont supprimer les mêmes candidats inutiles. Un large travail redondant peut ainsi être effectué. L'exemple 2.3 confirme ces propos.

Exemple 2.3
Soit C l'ensemble de clauses suivant :

$$C = \{(x_1 \vee x_2 \vee x_3), (x_1 \vee x_2 \vee \neg x_3), (x_1 \vee \neg x_4), (x_4 \vee \neg x_5)\}$$

L'ensemble des littéraux apparaissant dans les clauses binaires est $L_B = \{x_1, \neg x_4, x_4, \neg x_5\}$.
Voici la liste des propagations faites durant le premier appel à RLI lors de l'essai de production de x_1, $\neg x_4$ et $\neg x_5$.

- $C \cup \{\neg x_1\} \models \neg x_4 \neg x_5$
- $C \cup \{x_4\} \models x_1$
- $C \cup \{x_5\} \models x_4, x_1$

Aucun littéral n'est produit. Supposons que la méthode AVAL choisisse et interpréte à vrai la variable x_3. Le système de clauses courant devient :

$$C_{\{x_3\}} = \{x_1 \vee x_2, x_1 \vee \neg x_4, x_4 \vee \neg x_5\}$$

L'ensemble des littéraux des clauses binaires devient : $\{x_1, x_2, \neg x_4, x_4, \neg x_5\}$.
Et les propagations faites par le nouvel appel à RLI sont :

- $C_{\{x_3\}} \cup \{\neg x_1\} \models \neg x_4, \neg x_5, x_2$
- $C_{\{x_3\}} \cup \{x_4\} \models x_1$
- $C_{\{x_3\}} \cup \{x_5\} \models x_4, x_1$
- $C_{\{x_3\}} \cup \{\neg x_2\} \models x_1$

L'examen de cette seconde liste amène des remarques :

1. Le test de production de x_1 ($C_{\{x_3\}} \cup \{\neg x_1\}$) duplique la première liste et la complète par le littéral x_2. Ceci est dû à la présence de la nouvelle clause binaire $x_1 \vee x_2$.
2. Les tests de production de $\neg x_4$ et $\neg x_5$ sont les mêmes que ceux de la première étape.
3. L'algorithme RLI effectue un nouveau test de production, car x_2 est un nouveau candidat à la production.

Pour éviter les redondances dues à la reproduction de littéraux déjà produits au nœud précédent, nous avons modifié l'algorithme RLI. Nous mémorisons, tout au long de la recherche, les propagations faites par chaque test de production, créant ainsi des listes que nous appelons listes de production. Nous essayons de produire uniquement :

– Les littéraux que l'on a essayé de produire par le passé et qui ont, dans leur liste de production, des littéraux ayant des nouvelles occurrences dans les clauses binaires (comme le littéral x_1 de l'exemple 2.3). Ceci est du au fait que nous limitons le test de consistance à la propagation unitaire.
– Les littéraux qui n'ont pas déjà essayé d'être produits et dont leur opposé n'apparaît dans aucune liste production (comme le littéral x_2 de l'exemple 2.3).

L'algorithme 2.3 décrit cette nouvelle procédure que nous avons appelé RLI_INC (version incrémentale de RLI).

Algorithme 2.3 Une version incrémentale de l'algorithme de recherche des littéraux impliqués

Fonction RLI_INC(C : ensemble de clauses ; I : interprétation)
Retourne : un littéral l si $C \models l$
0 si $\nexists l \in L_{B_I}$ tel que $C \models l$

Début
Pour Chaque nouvelle clause binaire $u \vee v$
Faire
Si (App[$\neg u$]>0) %% Continuer les listes ou apparaît $\neg u$
Alors
Pour Chaque liste_production[$\neg u$] commençant par $\neg l$
Faire
Si Continue_propagation($\neg l$,C,I)=Inconsistance **Alors** retourner l
Si(App[$\neg v$]>0) %% Continuer les listes ou apparaît $\neg v$
Alors
Pour Chaque liste_production[$\neg v$] commençant par $\neg l$
Faire
Si Continue_propagation($\neg l$,C,I)=Inconsistance **Alors** Retourner l
Pour Chaque $w \in L_{B_I}$ tel que App[$\neg w$]=0 %%Création des nouvelles listes de productions.
Faire
Si Propagation($\neg w$,C,I)=Inconsistance **Alors** Retourner w
Retourner 0
Fin

L'algorithme RLI_INC garde en mémoire différentes données :

– L'ensemble des listes de productions
– le nombre `App[`u`]` qui est égal au nombre d'apparitions du littéral u dans les listes de production. Ainsi,si `App[`u`]=`0 alors le littéral $\neg u$ est candidat à la production. Au début de l'arbre de recherche le vecteur `App` est initialisée à 0 pour tous les littéraux.

– `liste_production(`u`)` contient l'ensemble des listes de production dans lesquelles u apparaît.

Il utilise également différentes fonctions et procédures :

– La fonction `Continue_propagation(`l, C, I`)` continue la liste de production commençant par l et incrémente la marque `App` de tous les **nouveaux** littéraux propagés. Pareillement à `Propagation` elle retourne la clause vide en cas d'inconsistance.

– La fonction `Propagation(`$\neg w, C, I$`)` crée une nouvelle liste de production (commençant par $\neg w$) et incrémente la marque `App` de tous les littéraux propagés. Dans le cas où l'ensemble de clauses résultant est inconsistant alors la clause vide est retournée. Cette procédure possède un schéma algorithmique relativement semblable à la procédure `Propagation Unitaire` de l'algorithme 1.3.

Ainsi, l'algorithme RLI_INC scrute en premier lieu les clauses binaires nouvellement créées. Il continue alors les listes de productions qui peuvent être complétées et peuvent ainsi produire un mono–littéral caché. Si tel n'est pas le cas, il essaie de produire les littéraux ayant des occurences dans des clauses binaires et encore candidats à la production. Nous montrons son comportement dans l'exemple qui suit.

Exemple 2.4

Reprenons l'exemple 2.3 et utilisons maintenant l'algorithme 2.3 pour essayer de produire des mono–littéraux cachés. Nous avons

$$C = \{(x_1 \vee x_2 \vee x_3), (x_1 \vee x_2 \vee \neg x_3), (x_1 \vee \neg x_4), (x_4 \vee \neg x_5)\}$$

L'ensemble des littéraux apparaissant dans les clauses binaires est $\{x_1, \neg x_4, x_4, \neg x_5\}$.

Le premier appel reste bien sûr inchangé. Nous mémorisons les listes de propagations faites par la procédure `Propagation` lors de l'essai de production des littéraux x_1, $\neg x_4$ et $\neg x_5$.

– $C \cup \{\neg x_1\} \models \neg x_4 \neg x_5$

– $C \cup \{x_4\} \models x_1$

– $C \cup \{x_5\} \models x_4, x_1$

Aucun littéral n'est produit. Nous donnons l'état du vecteur `App`.

Littéral	x_1	x_2	x_3	x_4	x_5	$\neg x_1$	$\neg x_2$	$\neg x_3$	$\neg x_4$	$\neg x_5$
Apparitions	2	0	0	2	1	1	0	0	1	1

Nous interprétons x_3 à Vrai. Le système de clauses courant est :

$$C_{\{x_3\}} = \{x_1 \vee x_2, x_1 \vee \neg x_4, x_4 \vee \neg x_5\}$$

L'ensemble des littéraux appartenant à des clauses binaires devient : $\{x_1, x_2, \neg x_4, x_4, \neg x_5\}$. Et l'ensemble des nouvelles clauses binaires est : $\{(x_1 \vee x_2)\}$.

App[$\neg x_1$] est non nul, donc nous devons compléter les listes de production où il apparaît, ici seulement la première. En appelant la procédure Continue_propagation($\neg x_1, C_{\{x_3\}}, \{x_3\}$), on obtient :

– $C_{\{x_3\}} \cup \{\neg x_1\} \models \neg x_4, \neg x_5, x_2$

App[$\neg x_2$] est nul, donc nous essayons de produire x_2. L'appel de Propagation($\neg x_2, C_{\{x_3\}}, \{x_3\}$) donne :

– $C_{\{x_3\}} \cup \{\neg x_2\} \models x_1$

On retrouve bien les listes de productions que nous avons réalisées dans l'exemple 2.3 sachant que deux d'entre elles sont restées identiques et n'ont pas eu besoin d'être réactualisées. ✍

L'algorithme de recherche des littéraux impliqués nécessite une plus grande quantité de mémoire dans sa version incrémentale (algorithme 2.3) que dans sa version de base (algorithme 2.1). L'algorithme RLI utilise un tableau d'entiers, sa complexité en mémoire est de l'ordre de $O(n)$. La nouvelle version incrémentale RLI_INC nécessite toujours un tableau d'entiers, mais doit également garder en mémoire les listes de productions ainsi que les listes d'apparitions des littéraux dans les listes de productions. La complexité en mémoire est donc de l'ordre de $O(n^3)$.

2.4 Le choix des littéraux

De même que le choix heuristique d'un littéral est important dans une procédure énumérative de type DP (voir section 1.3), le choix du littéral que l'on essaie de produire est important pour l'efficacité des algorithmes 2.1 et 2.3 de recherche des littéraux impliqués. Une première approche envisageable consiste à sélectionner en premier lieu les littéraux ayant le plus d'occurences dans les clauses binaires. Cette stratégie propage le plus de littéraux possibles et peut permettre d'atteindre le plus rapidement possible l'inconsistance. Le problème de ce choix réside dans l'obligation d'ordonner, à chaque nœud, les littéraux en fonction de leurs nombre d'occurences dans les clauses binaires. Le gain potentiel induit par cette stratégie risque d'être sérieusement compromis par le temps nécessaire au tri des littéraux.

Nous avons donc utilisé une autre approche qui consiste à utiliser au mieux le «pouvoir» de la proposition 2.1. A chaque appel de la procédure RLI, nous essayons d'éliminer le plus de littéraux impossibles à produire. Nous savons qu'un littéral n'est plus candidat à l'implication lorsque son opposé a été propagé lors d'un échec de production. Supposons que nous limitions la propagation de la méthode RLI aux clauses binaires. Dans ce cas, un littéral est supprimé de la liste des candidats uniquement si son opposé est propagé, et donc si il appartient à une clause binaire. Bien que cette hypothèse sur la limitation de la propagation soit fausse, nous allons utiliser cette remarque afin d'optimiser l'impact de la proposition 2.1 sur l'algorithme RLI. Les littéraux binaires purs, ceux qui n'apparaissent que positivement ou négativement dans les clauses binaires, ont donc peu de chance d'être supprimés de l'ensemble des candidats, nous les choisissons donc en premier comme candidats

à la production. La section 3.4 montre l'impact d'une telle stratégie sur l'algorithme de recherche des littéraux impliqués.

2.5 Conclusion

Nous avons proposé deux versions de l'algorithme de recherche des littéraux impliqués RLI. L'une classique, l'autre prenant en compte les résultats acquis lors de la construction de l'interprétation. Ces procédures détectent des mono–littéraux «cachés» que ne peux pas propager la procédure DP. La combinaison de l'un de ces deux algorithmes avec la méthode DP donne naissance à la méthode AVAL. C'est lors des expérimentations, menées dans le chapitre suivant, que nous explicitons cette appellation.

Chapitre 3

Expérimentations

L'OBJECTIF DE CE CHAPITRE est de faire une étude expérimentale de l'algorithme de recherche des littéraux impliqués RLI et de la méthode AVAL. Nous commençons par comparer les procédures DAVIS et PUTNAM et AVAL, ensuite, nous observons le comportement de l'algorithme de recherche des littéraux impliqués lors de la résolution d'instances aléatoires difficiles situé dans la région du seuil. Cette partie met en évidence l'influence de la production de littéraux sur la méthode DP et montre le comportement des méthodes énumératives lors de la résolution de problèmes aléatoires difficiles. Enfin, nous comparons les méthodes AVAL, POSIT [43], SATO [101] et SATZ[66] sur les problèmes aléatoires situés dans la région du seuil et sur des problèmes structurés issus des challenges DIMACS [38] et Beijing [13].

Toutes les expérimentations faites dans ce mémoire sont réalisés sur un K6II cadencé à 400 MHz avec 128 Mo de RAM, fonctionnant sous un système Linux 2.4. Les temps sont exprimés en secondes.

3.1 Comparaison des méthodes DP et AVAL

Nous avons comparé les méthodes énumératives AVAL et DP sur des instances aléatoires situées dans la région du seuil ($\frac{c}{v} = 4.25$). Deux heuristiques ont été utilisées :

- L'heuristique de JEROSLOW et WANG généralisé par FREEMAN que nous avons défini dans la section 1.5.1. Nous la dénotons par les initiales JWF.
- L'heuristique basé sur la fonction de poids utilisée dans CSAT(voir section 1.5.4), dénoté par les initiales CS.

Les résultats des tableaux 3.1 et 3.2 ont été obtenus sur une moyenne de 100 instances. Ils montrent que pour les deux heuristiques utilisées, la méthode AVAL parcourt moins de nœuds (tableau 3.1) et est plus rapide (tableau 3.2) que la méthode DP. Cela confirme l'efficacité de l'algorithme de recherche des littéraux impliqués et l'intérêt de le combiner avec DP. De plus, le gain croît au fur et à mesure que le nombre de variables augmente, donnant ainsi espoir de résoudre des instances de

Nombre de variables	100	150	200	250
DP + JWF	165	1 297	9 332	65 208
AVAL + JWF	18	101	573	3 327
DP + CS	328	3 291	29 425	243 819
AVAL + CS	21	133	860	5 233

Tableau 3.1. *Comparaison entre* DP *et* AVAL *(Nœuds)*

Nombre de variables	100	150	200	250
DP + JWF	0.01	0.12	1.1	9.8
AVAL + JWF	0.01	0.1	0.8	6.5
DP + CS	0.04	0.56	6.3	64
AVAL + CS	0.01	0.14	1.2	10

Tableau 3.2. *Comparaison entre* DP *et* AVAL *(Temps)*

grandes tailles. L'utilisation de l'algorithme RLI réduit le nombre de points de choix d'une heuristique donnée. La méthode AVAL qui en découle peut donc remplacer efficacement DP comme algorithme de base des méthodes énumératives.

La déduction des littéraux semble prometteuse. Il est important d'étudier dans le détail son comportement lors de la résolution, notamment de regarder son taux de réussite.

3.2 Taux de réussite dans la production de littéraux

Le tableau 3.3 montre que 90 % des appels à l'algorithme de déduction RLI fournissent un littéral impliqué par l'ensemble de clauses. C'est pour cette raison que l'espace de recherche de la méthode AVAL est sensiblement plus petit que celui de la méthode DP. Il est intéressant de noter que plus le nombre de variables augmente, plus le pourcentage de littéraux produits augmente.

Nombre de variables	100	140	180	220	260
Nb d'appels à RLI	170	860	3 795	14 136	62 672
Nombre de succès de RLI	152	784	3 494	13 105	58 372
pourcentage de succès	89	91	92	92	93

Tableau 3.3. *Pourcentage de succès de l'algorithme* RLI

La figure 3.1 donne le pourcentage de réussite de l'algorithme RLI en fonction de la profondeur

de l'arbre de recherche sur des instances aléatoires de 250 variables. Les résultats obtenus sont très intéressants. La majorité des échecs de production sont situés en haut de l'arbre de recherche (sous la profondeur $\frac{v}{10} = 25$). Après avoir dépassé cette profondeur, tous les appels réussissent et la méthode AVAL consiste uniquement en la propagation des mono–littéraux. Une *avalanche* de mono–littéraux propagés est créee dès les premières réussites de l'algorithme RLI permettant de terminer la recherche en temps linéaire. La dénomination AVAL est tirée de cette avalanche de mono–littéraux observée lors de la résolution de problèmes aléatoires situés dans la région du seuil.

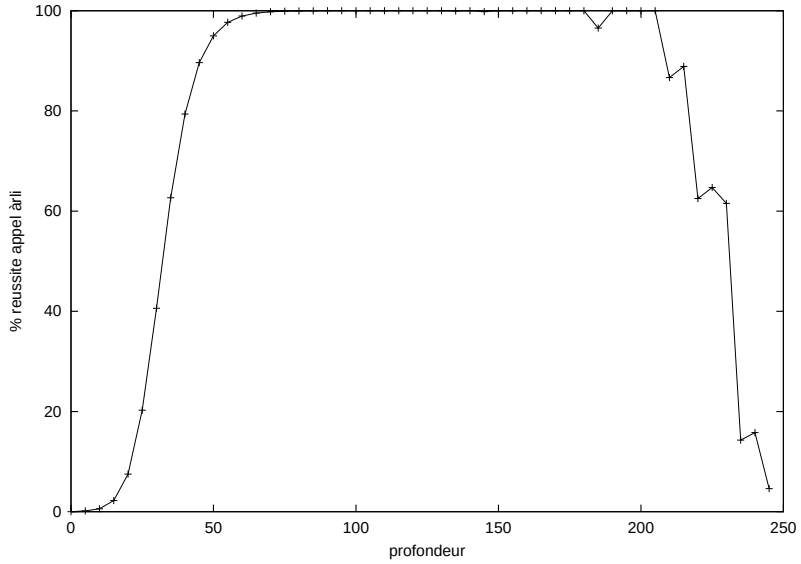


Figure 3.1. *Réussite des appels à* RLI *en fonction de la profondeur de l'arbre de recherche (250 variables)*

Ce phénomène d'avalanche de mono–littéraux est également observé dans DP, mais il est moins marqué et commence beaucoup plus tard (à un niveau plus profond) dans l'arbre de recherche. Ceci explique les différences d'efficacité entre DP et AVAL. Plus tôt arrive l'avalanche, plus efficace est l'algorithme. Ces résultats peuvent laisser penser qu'une borne minimale du nombre de nœuds qu'une procédure énumérative doit parcourir pour une heuristique donnée est atteinte grâce à la procédure AVAL. Bien que cela ne soit pas évident, cette remarque s'applique également à l'heuristique UP. En effet, lorsque Chu Min LI proposa la méthode SATZ, il proposa également diverses variantes dépendant du nombre de variables candidates à la propagation unitaire [65]. La procédure notée SATA examine toutes les variables. C'est la variante qui parcourt le moins de nœuds et qui peut être considérée comme la réunion de l'heuristique UP sans propagation des littéraux impliqués et de l'algorithme RLI.

Connaissant la zone des appels à RLI qui réussissent, il semble judicieux de déterminer où se trouve la majorité de ceux–ci. La figure 3.2 fournit ces renseignements. Elle donne le nombre d'appels

à la procédure RLI en fonction de la profondeur de l'arbre de recherche (instances de 250 variables). Les barres représentent le nombre d'appels à RLI, la ligne le nombre d'appels réussis. On remarque qu'en haut de l'arbre de recherche (sous une profondeur $\frac{v}{10} = 25$) il y a peu d'appels à cette procédure. C'est la partie la plus difficile de l'arbre de recherche, là où l'on effectue la plupart des points de choix. De plus, dans cette région, les appels ne fournissent que très rarement un littéral impliqué car l'ensemble des clauses contient très peu de clauses binaires. C'est entre les profondeurs 25 (soit $\frac{v}{10}$) et 100 (soit $\frac{v}{2.5}$) que se trouve la majorité des appels à RLI et dans la plupart des cas ceux–ci aboutissent et fournissent un littéral impliqué.

Dépassée la profondeur 100 (soit $\frac{v}{2.5}$) le nombre d'appels est négligeable. Il n'y a plus que des mono–littéraux à propager pour terminer la recherche. C'est pour cette raison que la figure 3.1 présente un aspect irrégulier et décroissant après la profondeur 200 (soit $\frac{v}{1.5}$) , le nombre d'appels RLI étant infime (par exemple, à la profondeur 200, on fait en moyenne 0.09 appels à RLI), un échec de production a beaucoup d'influence sur le pourcentage de réussite.

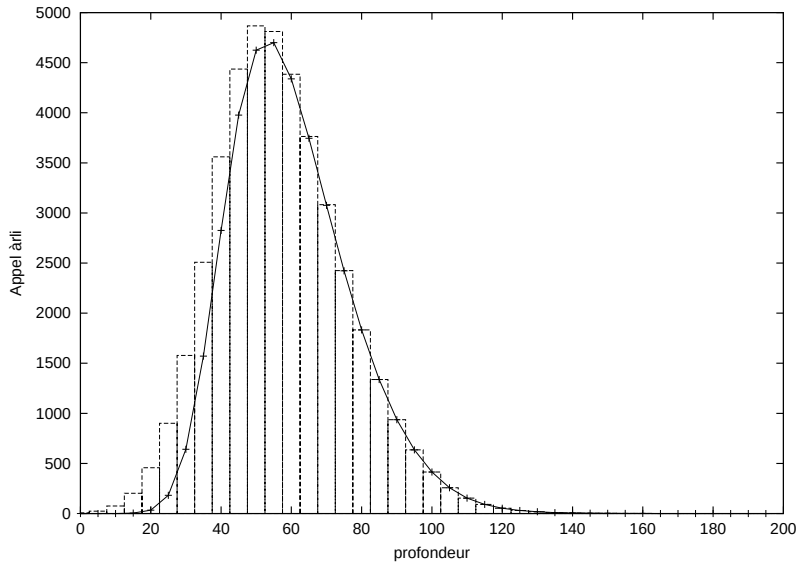


Figure 3.2. *Localisation des appels à* RLI *(250 variables)*

Il est clair que certains paramètres, notamment le nombre de clauses binaires, favorisent la production de littéraux. Nous étudions maintenant l'influence de ces paramètres sur l'implication des littéraux.

3.3 Seuil de production des mono–littéraux

Le nombre de clauses binaires présentes dans l'ensemble de clauses courant a un grand impact sur la production de littéraux. Plus il y a de clauses binaires, plus la probabilité de succès dans la

recherche de littéraux impliqués est grande. Comme les performances de la méthode AVAL dépendent de l'efficacité de l'algorithme RLI, il est important de connaître le nombre de clauses binaires nécessaires pour produire un littéral. En théorie, le seuil de satisfaisabilité pour les problèmes aléatoires de 2–SAT est égal à 1 (voir théorème 1.3). La procédure RLI doit donc pouvoir réussir à produire un littéral quand le nombre de clauses binaires est égal au nombre de variables non instanciées.

Mais en pratique, la production de littéraux est garantie avec un nombre plus petit de variables que de clauses. La figure 3.3 présente le pourcentage de succès de l'algorithme RLI en fonction du rapport entre le nombre de clauses binaires de l'ensemble de clauses courant et le nombre de variables non interprétées. Ces résultats ont été obtenus sur une moyenne de 100 instances sur des problèmes aléatoires de 250 variables situés dans la région du seuil. La courbe obtenue est similaire à celle de la figure 1.1 caractérisant le seuil de satisfaisabilité des problèmes k–SAT. Un phénomène de transition est donc observé. Lorsque peu de clauses binaires existent (pour un rapport $\frac{c}{v} \leq 0.6$), la production échoue car il y a trop peu de clauses binaires. Au–dessus de ce rapport, l'algorithme RLI produit toujours un littéral. De plus, la phase de transition est rapide. Ces résultats peuvent être exploités pour améliorer la procédure RLI afin d'évaluer la probabilité de produire un littéral en fonction du nombre de clauses binaires présent dans l'ensemble de clauses.

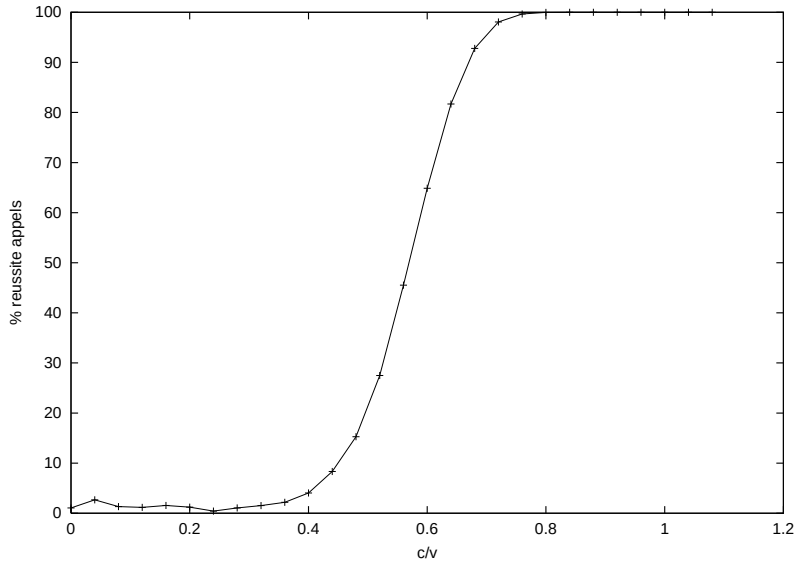


Figure 3.3. *Seuil de production (250 variables)*

Nous avons jusqu'ici proposé une étude sur la localisation des appels à l'algorithme RLI et sur le nombre de littéraux produits. Mais pour que cet algorithme soit utilisable d'un point de vue informatique il doit être efficace. Nous étudions, dans la prochaine section, sa complexité en moyenne.

3.4 Complexité de l'algorithme RLI

Dans cette section, nous regardons de plus près la complexité de la procédure RLI que nous déterminons en fonction du nombre de propagations unitaires. Plusieurs questions se posent :

- La complexité dans le pire des cas est en $O(|L_{B_I}| \times |L_{C_I}|)$, mais quelle est la complexité en moyenne sur les problèmes aléatoires ?
- Les performances de l'algorithme de déduction incrémental 2.3 sont–elles meilleures que celles de l'algorithme non incrémental ?
- Un choix stratégique des littéraux à produire influe–t–il sur la complexité en moyenne de l'algorithme RLI ?

Ces différentes questions trouvent leurs réponses avec la figure 3.4. Les résultats sont obtenus sur une moyenne de 200 instances situés dans la région difficile. En abscisses, nous avons le nombre de variables et en ordonnées le nombre de propagations moyennes faites par appel aux différentes procédures RLI. La courbe du haut correspond à la procédure RLI dans sa version non incrémentale (algorithme 2.1), celle du milieu correspond à la version de base où l'on essaie de produire en premier les littéraux binaires purs et la plus basse correspond à la version incrémentale RLI_INC. (algorithme 2.3).

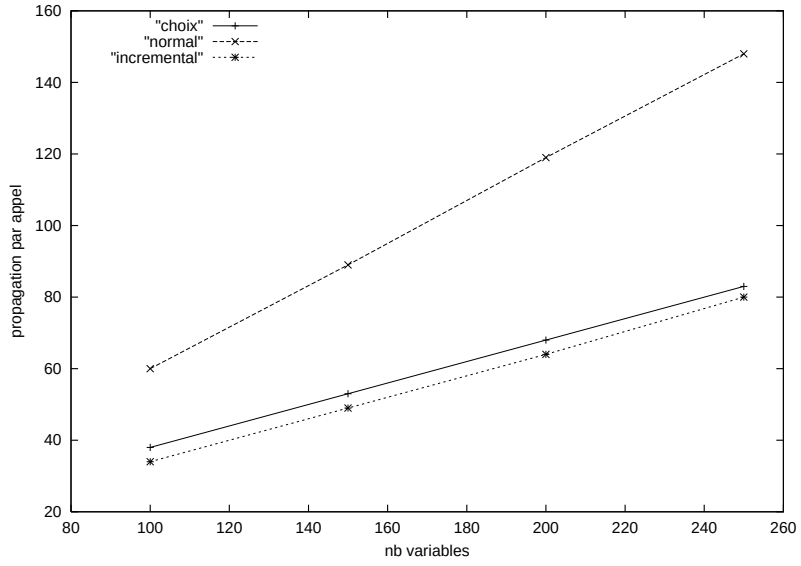


Figure 3.4. *Complexité des différents algorithmes* RLI

Lors de la résolution de problèmes aléatoires, ces trois procédures ont une complexité linéaire par rapport au nombre de variables. C'est cette complexité linéaire, bien meilleure que la complexité dans le pire des cas, qui permet d'utiliser l'algorithme RLI à tous les points de choix de l'arbre de recherche.

La version incrémentale possède une meilleure complexité que la version de base avec un gain qui croît avec le nombre de variables, 26 % pour 100 variables et 36 % pour 250 variables. L'augmentation de la différence vient du fait que plus le nombre de variables est important, moins la structure de l'ensemble de clauses diffère entre deux nœuds consécutifs.

Favoriser les littéraux purs comme candidats à la production donne une complexité en moyenne similaire à celle de la version incrémentale. Il met en évidence qu'un choix judicieux est tout aussi important qu'un nouvel algorithme, tout en étant beaucoup plus simple à mettre en oeuvre. Le problème qui survient alors est que les littéraux produits ont de grandes chances d'être des littéraux binaires purs qui ne vont pas propager d'autres mono–littéraux, mais seulement satisfaire un certain nombre de clauses binaires. Le nombre d'appels à la procédure RLI est donc plus important. Le choix des littéraux à la production nécessite donc un compromis entre un choix aléatoire et un choix des littéraux binaires purs.

Après avoir observé de près le comportement de l'algorithme RLI et de la méthode AVAL, il reste à savoir si cette dernière peut rivaliser avec d'autres méthodes qui ont déja montré leur efficacité.

3.5 Comparaison avec SATZ, SATO et POSIT

La comparaison est faite sur des problèmes aléatoires situés dans la région du seuil et sur les challenges DIMACS et Beijing. Nous pourrions baser notre comparaison sur une plus grande variété d'instances et de méthodes, mais le temps de calcul et la synthèse des résultats serait alors trop importants. Le site SATEX de Laurent SIMON [25] qui est consacrée à la comparaison de nombreuses méthodes, semble être une voie prometteuse pour répondre à ce types de problèmes.

Durant toute cette section, la méthode AVAL est agrémentée d'un pré–traitement comme celui de SATZ qui favorise la création de clauses binaires. Comme la production de littéraux échoue souvent dans la partie haute de l'arbre de recherche (profondeur inférieure à $\frac{v}{10}$ de la figure 3.2), nous utilisons dans cette région l'heuristique UP appliquée à tous les littéraux ayant des occurrences dans les clauses binaires. Après cette profondeur, nous utilisons l'heuristique MOM'S proposée par John FREEMAN dans [43]. De plus, La méthode AVAL est combinée avec l'algorithme 2.1 de recherche des littéraux impliqués. Même si la complexité en moyenne est meilleure avec l'algorithme incrémental, le coût de maintenance des listes de production est trop élevé et les temps de calculs générés par cette version sont, pour l'instant, trop importants.

3.5.1 Instances aléatoires

Nous avons généré des problèmes 3–SAT aléatoires situés au voisinage du seuil ($\frac{c}{v} = 4.25$). Le nombre de variables varie de 100 à 400 par pas de 50. Les résultats sont obtenus sur une moyenne de 200 instances jusqu'à 300 variables et de 100 instances au–delà. Le tableau 3.4 montre les résultats

obtenus par les quatre méthodes.

Nb de variables		100	150	200	250	300	350	400
AVAL	Temps	0.03	0.12	0.77	5.3	42	262	2 390
	Nœuds	14	71	378	2 178	13 769	69 709	502 499
SATZ	Temps	0.03	0.1	0.4	2.5	15.4	88	602
	Nœuds	18	111	590	3 089	17 942	87 060	521 399
POSIT	Temps	0.007	0.07	0.52	4.2	33	187	1 911
	Nœuds	39	261	1 489	10 118	66 789	327 709	2 904 226
SATO	Temps	0.06	2.5	100	2 680	> 7200	> 7200	> 7200
	Nœuds	212	1 488	9 537	57 110	-	-	-

Tableau 3.4. *Comparaison sur les Problèmes Aléatoires*

De toutes les méthodes, AVAL est celle qui parcourt le moins de nœuds. La méthode SATZ montre que l'utilisation de l'heuristique UP sur des variables judicieusement sélectionnées permet de ne parcourir qu'un nombre restreint de nœuds. Par contre les méthodes POSIT et surtout SATO ont un espace de recherche de très grande taille. Dans le cas de SATO, cela tend à montrer que l'utilisation du «backjumping» est une mauvaise stratégie sur les instances aléatoires.

La comparaison sur le temps de calcul montre une grande rapidité de la méthode SATZ qui, à notre connaissance, est la méthode la plus efficace pour la résolution d'instances aléatoires. La méthode POSIT est un peu plus rapide que AVAL. La méthode SATO est incapable de résoudre des instances de 300 variables en moins de deux heures. La maintenance de la structure d'arbre (introduite à la section 1.5.3) possède un coût calculatoire trop élevé.

Chu Min LI et Sylvain GÉRARD [67] montrent, par une étude expérimentale sur les problèmes aléatoires, que la taille de l'arbre de recherche d'une méthode énumérative qui choisirait à chaque nœud la meilleure variable serait de l'ordre de $2^{(v/22.18)-1.4}$ pour 300 variables. Les deux méthodes SATZ (taille de l'arbre de l'ordre de $2^{(v/21.22)}$) et AVAL (taille de l'arbre de l'ordre de $2^{(v/21.81)}$) ne sont pas trop éloignées de cette complexité minimale en utilisant des approches différentes. Par contre, ces résultats sont très éloignés de la taille théorique empirique, égale à $2^{(v/326.2539)}$, que proposent Paul BEAME et al. dans [12].

Grâce aux nombreux travaux entrepris ces dernières années, il nous semble que les méthodes énumératives de type DAVIS et PUTNAM ont atteint une complexité minimale. Le challenge de SELMAN semble impossible à résoudre avec de telles méthodes (bien entendu, sans l'aide de moyens calculatoires), cet avis est également partagé par Chu Min LI et Sylvain GÉRARD [67]. Peut être que l'élaboration de méthodes sensiblement différentes permettront, un jour, de relever ce challenge.

		AVAL		SATZ		POSIT		SATO	
Classe de problèmes	#M	#S	Temps	#S	Temps	#S	Temps	#S	Temps
aim-50	24	24	7.5	24	0.31	24	0.08	24	0.13
aim-100	24	24	1.8	24	159	24	6.6	24	0.21
aim-200	24	24	2.2	24	0.09	12	86 401	24	0.71
dubois	13	9	36 475	10	31 415	9	38 807	13	0.8
hole	5	5	84	5	101	5	176	5	292
ii8	14	14	4.8	14	2.6	14	0.35	14	0.98
ii16	10	10	101	10	56	9	7 230	10	4.5
ii32	17	17	383	16	7 445	17	117	17	8
jnh	50	50	6.4	50	6.2	50	0.15	50	1.7
par8	10	10	0.35	10	0.4	10	0.01	10	0.17
par16	10	10	92	10	1 443	10	17	10	302
ssa	8	7	9 000	8	421	8	15	8	2.6

Tableau 3.5. *Comparaison sur le challenge* DIMACS

3.5.2 Challenge DIMACS et Beijing

Les problèmes du challenge DIMACS[1] [38] sont des problèmes difficiles à résoudre par les méthodes énumératives. Certaines méthodes ont vu le jour pour résoudre des instances ouvertes de ce challenge (voir la description EQSATZ dans la section 1.5.5). Les instances sont regroupées par classe. Le temps de résolution est limité à deux heures (7200 secondes). Les résultats sont donnés par la table 3.5. La colonne *#M* indique le nombre de problèmes d'une classe. La colonne *#S* le nombre d'instances résolues par une méthode. Enfin, la colonne *Temps* donne le temps total de résolution d'une classe.

Les résultats obtenus par la méthode SATO sont meilleurs que ceux des trois autres méthodes. Exceptées les classes *Dubois* et *ssa*, la méthode AVAL résout toutes les instances des classes avec des temps satisfaisants. Les méthodes SATZ et POSIT donnent également des résultats satisfaisants. Ceci tend à montrer l'intérêt de mettre au point des méthodes efficaces sur les problèmes aléatoires qui s'avèrent également efficaces sur des instances structurées.

Les instances du challenge Beijing [13] sont listées individuellement. Les résultats, répertoriés dans le tableau 3.6, sont donc couplés au nombre de nœuds nécessaires pour la résolution. Le temps de résolution maximal est toujours limité à deux heures. La méthode AVAL résout quatorze problèmes sur seize, SATZ en résout treize, POSIT huit et SATO en résout 11. Les résultats obtenus sur ce challenge sont donc également satisfaisants. Notons que la méthode AVAL est la seule capable de résoudre le problème *2bit_add_10*.

[1] Nous avons omis certaines instances du challenge DIMACS comme *par32*

	AVAL		SATZ		POSIT		SATO	
Problème	Temps	Nœuds	Temps	Nœuds	Temps	Nœuds	Temps	Nœuds
2bitadd_10	1 400	1 909 173	-	-	-	-	-	-
2bitadd_11	2.6	4 814	60	60 498	0.29	970	-	-
2bitadd_12	15	28 814	0.01	57	0.O3	38	-	-
2bitcomp_5	0.01	15	0.01	6	0.01	35	0.02	65
2bitmax_6	0.02	13	0.02	7	0.01	14	0.02	93
3bitadd_31	-	-	-	-	-	-	-	-
3bitadd_32	-	-	1 670	297652	-	-	-	-
3blocks	1	16	0.92	12	0.8	608	0.26	141
4blocks	533	18 821	458	116 826	-	-	4 139	1 597 477
4blocksb	5.2	97	4	8	30	8 643	0.53	37
e0ddr2-10-by-5-1	31	60	59	27	-	-	5.1	530
e0ddr2-10-by-5-4	97	1 191	61	32	1 628	38 200	2.2	176
enddr2-10-by-5-1	28	50	-	-	-	-	2.9	213
enddr2-10-by-5-8	16	11	65	31	-	-	2.4	183
ewddr2-10-by-5-1	17	11	90	44	78	298	2.6	197
ewddr2-10-by-5-8	80	157	74	40	-	-	2.8	176

Tableau 3.6. *Comparaison sur le challenge Beijing*

3.6 Conclusion

Bien entendu, aucune des méthodes comparées n'est systématiquement meilleure que les autres. L'heuristique choisie, le pré–traitement ou les techniques de «back–jumping» et de «look–ahead» donnent de plus ou moins bons résultats suivant l'instance à résoudre. Il semble irréaliste de pouvoir créer un solveur de problèmes SAT qui soit efficace pour toutes les instances. C'est pour cette raison qu'apparaissent des algorithmes ciblés sur des instances particulières comme ceux que nous avons introduits dans la section 1.5.5.

Les résultats obtenus montrent également qu'il n'est pas irréaliste de vouloir combiner l'algorithme de recherche des littéraux impliqués RLI avec la méthode DP à tous les nœuds de l'arbre de recherche. De plus l'étude expérimentale que nous avons menée fournit un certain nombre de renseignements qui pourraient servir à optimiser la localisation des appels à cet algorithme. Cette étude a également mis en avant des arguments quant à la difficulté de relever le challenge de SELMAN et al. [84].

Deuxième partie

Génération de Modèles Finis

Chapitre 4

Génération de modèles finis en logique du premier ordre

LA LOGIQUE DU PREMIER ORDRE possède un pouvoir d'expression plus élevé que la logique propositionnelle. Elle a été introduite en premier lieu par les mathématiciens pour répondre à leurs besoins. Elle permet d'ailleurs de représenter tous les objets dont ils se servent. Elle a ensuite intéressé les informaticiens par son aptitude à représenter et à manipuler les connaissances.

Nous nous intéressons à la recherche de modèles finis pour des théories du premier ordre. Différents points de vues rendent les modèles intéressants. Tout d'abord, le raisonnement humain ne peut se faire sans l'aide de modèles. Ainsi, on s'aide toujours d'un exemple pour rechercher des démonstrations de théorèmes. De plus, les modèles peuvent expliquer la non–validité de conjectures, et servir ensuite à proposer des versions corrigées de ces conjectures. Enfin, la recherche de modèles finis est un problème décidable. Il suffit de tester toutes les interprétations possibles de la théorie pour en exhiber un modèle ou pour montrer son inconsistance. Mais cet atout ne nous dispense pas d'être en face d'un problème NP–complet et donc difficile.

En premier lieu, nous définissons dans ce chapitre la logique du premier ordre multi–types. Nous donnons ensuite une description de divers générateurs de modèles finis.

4.1 La logique des prédicats

Comme nous l'avons fait pour la logique propositionnelle, nous commençons par définir la syntaxe de la logique du premier ordre avant de passer à la sémantique. Nous introduisons dans ces définitions la présence de types, autrement nommés sortes.

4.1.1 Syntaxe

Définition 4.1 *Le langage de la logique des prédicats est construit à partir de l'alphabet contenant :*

- *Les parenthèses «(» et «)», Un ensemble de connecteurs : non «¬», ou «∨», et «∧», implique «→», équivalent «↔» et les quantificateurs universel «∀» et existentiel «∃»*
- *Un ensemble $\mathcal{S}$ de types (sortes) disjoints.*
- *Un ensemble $\mathcal{V}$ de variables. Chaque variable possède un type.*
- *Un ensemble $\mathcal{F}$ de symboles fonctionnels. Un symbole fonctionnel f d'arité k est spécifié par sa signature $f : s_1, s_2, \ldots, s_k \mapsto s$ où $s_1, s_2, \ldots, s_k, s$ sont des types de S. Les constantes sont des fonctions d'arité 0.*
- *Un ensemble $\mathcal{P}$ de symboles relationnels appelés aussi prédicats. Si p est un prédicat d'arité k alors p est spécifié par sa signature $p : s_1, s_2, \ldots, s_k$ où $s_1, s_2, \ldots, s_k$ sont des types de S.*

Définition 4.2 (Terme) *L'ensemble des* termes *de la logique du premier ordre est le plus petit ensemble d'expressions tel que :*

- *Les variables sont des termes*
- *Si $t_1, t_2, \ldots, t_k$ sont des termes et f un symbole fonctionnel d'arité k alors $f(t_1, t_2, \ldots, t_k)$ est un terme.*

Notation 4.1 *t étant un terme, on note $sort(t)$ le type unique du terme t.*

L'exemple suivant va servir de base pour expliciter les différentes notions introduites dans cette section.

Exemple 4.1

- Le type `personne`
- les prédicats
  - `est_femme : personne` qui représente l'affirmation qu'une personne est une femme.
  - `est_homme : personne` qui représente l'affirmation qu'une personne est un homme.
  - `est_grand_mère : personne` qui représente l'affirmation qu'une personne est une grand–mère.
- Les fonctions `mère : personne` $\mapsto$ `personne`, `père : personne` $\mapsto$ `personne` qui représentent la mère et le père d'une personne.

Dans ces conditions, le terme `mère(mère(x))` désigne donc la grand–mère maternelle de `x`. ✍

Définition 4.3 (Atome) *L'ensemble des* atomes *est le plus petit ensemble d'expressions tel que : Si $t_1, t_2, \ldots, t_k$ sont des termes et p un prédicat d'arité k alors $p(t_1, t_2, \ldots, t_k)$ est un atome.*

Exemple 4.2

Reprenons l'exemple 4.1. Voici deux atomes :

- `est_femmme(mère(x))`
- `est_homme(père(x))` ✍

Définition 4.4 (Formule) *L'ensemble des* formules *de la logique du premier ordre est le plus petit ensemble d'expressions tel que :*

- *$Vrai$ ($\top$) et $Faux$ ($\bot$) sont des formules.*
- *Les atomes sont des formules.*
- *Si A, B sont des formules alors $\neg A$, (A), $A \to B$, $A \vee B$, $A \leftrightarrow B$ et $A \wedge B$ sont des formules.*
- *Si A est une formule et x une variable alors $\forall x A$ et $\exists x A$ sont des formules.*

Les formules de la forme $P(t_1, t_2, \ldots, t_n)$ et $\neg P(t_1, t_2, \ldots, t_n)$ sont appelées des littéraux (positif et négatif).

Exemple 4.3

Reprenons une nouvelle fois l'exemple 4.1, voici trois formules différentes :

$\forall$ `x est_femmme(mère(x))`

$\forall$ `x est_homme(père(x))`

$\forall$ `x est_grand_mère(x)` $\leftrightarrow \exists$ `y,z (x = mère(y))` $\wedge$ `(y = mère(z)` $\vee$ `y = père(z))` ✍

Définition 4.5 (theorie) *Une théorie T de la logique du premier ordre est caractérisée par le quadruplet $(\mathcal{S}, \mathcal{F}, \mathcal{P}, A)$ dans lequel $\mathcal{S}$ est l'ensemble des types de la théorie, $\mathcal{F}$ celui des symboles fonctionnels et $\mathcal{P}$ celui des prédicats. Et enfin, A est une formule.*

Nous introduisons maintenant les notions de substitutions et d'unifications qui seront utiles lors de la définition de l'aspect sémantique des théories du premier ordre.

Définition 4.6 (Substitution) *On appelle substitution le remplacement d'une variable x par le terme t, on note une telle substitution par $(x/\ t)$. Si A est une formule on note alors $A(x/\ t)$ la formule obtenue en remplaçant dans A toutes les occurences de x par t.*

Si ϑ_1 et ϑ_2 sont deux substitutions disjointes, on note $\vartheta_1 \cup \vartheta_2$ leur composition.

Définition 4.7 (Unification) *Étant donné un ensemble de formules $\mathcal{A} = \{A_1, \ldots, A_n\}$ on appelle unificateur de $\mathcal{A}$ toute substitution ϑ telle que*

$$A_1(\vartheta) = A_2(\vartheta) = \cdots = A_n(\vartheta)$$

Exemple 4.4

Soit $A_1 = p(x, z)$, $A_2 = p(f(y), g(a))$ et $A_3 = p(f(w), z)$. on a alors $\vartheta = (x/\ f(u)) \cup (y/\ u) \cup (z/\ g(a))$ est un unificateur de A_1, A_2 et A_3.

En effet, on a bien : $A_1(\vartheta) = A_1(\vartheta) = A_3(\vartheta) = p(f(u), g(a))$ ✍

4.1.2 Sémantique

En logique propositionnelle, l'interprétation d'une formule est déterminée par l'interprétation des variables propositionnelles qui composent cette formule. En logique des prédicats, l'interprétation d'une formule sera déterminée par l'interprétation donnée aux symboles fonctionnels et aux prédicats. Nous ne considérons que des modèles finis, c'est à dire des modèles pour lesquels les domaines associés aux types sont finis.

Définition 4.8 *Soit $T = (\mathcal{S}, \mathcal{F}, \mathcal{P}, A)$ une théorie du premier ordre. On associe à chaque type s de $\mathcal{S}$ un domaine D_s de taille finie. L'interprétation I associe alors à chaque symbole fonctionnel $f : s_1, s_2, \ldots, s_k \mapsto s$, une fonction de $D_{s_1} \times \ldots \times D_{s_k}$ dans D_s et à chaque symbole relationnel $p : s_1, s_2, \ldots, s_k$ une fonction $D_{s_1} \times \ldots \times D_{s_k} \mapsto \{Vrai, Faux\}$.*

On généralise, par induction, une interprétation aux termes, aux atomes et aux formules du premier ordre de la façon suivante :

1. *Généralisation aux termes*

   Si $t_1, \ldots, t_k$ sont des termes alors $I[f(t_1, t_2, \ldots, t_k)] = I[f](I[t_1], \ldots, I[t_k])$

2. *Généralisation aux atomes*

   Si p est un prédicat d'arité k et $t_1, \ldots, t_k$ sont des termes alors

   $I[p(t_1, \ldots, t_k)] = I[p](I[t_1], \ldots, I[t_2])$

   Si t_1 et t_2 sont deux termes de même type alors $I[t_1 = t_2] = Vrai$ si et seulement si $I[t_1] = I[t_2]$.

3. *Généralisation aux formules*

   Si A et B sont deux formules alors
   - $I[Vrai] = Vrai$ et $I[Faux] = Faux$.
   - $I[\neg A] = Vrai$ si et seulement si $I[A] = Faux$
   - $I[A \vee B] = Vrai$ si et seulement si $I[A] = Vrai$ ou $I[B] = Vrai$.
   - $I[A \wedge B] = Vrai$ si et seulement si $I[A] = Vrai$ et $I[B] = Vrai$.
   - $I[A \rightarrow B] = Vrai$ si et seulement si $I[A] = Faux$ ou $I[B] = Vrai$
   - $I[A \leftrightarrow B] = Vrai$ si et seulement si $(I[A] = I[B])$
   - $I[\exists x A] = Vrai$ si il existe $e \in Sort(x)$ tel que $I[A(x/\ e)] = Vrai$.
   - $I[\forall x A] = Vrai$ si pour tout $e \in Sort(x)$ $I[A(x/\ e)] = Vrai$.

La définition suivante généralise des notions déja introduites pour la logique propositionnelle (définition 1.5).

Définition 4.9

- *Une interprétation est dite partielle, si il existe au moins un terme terminal qui n'est pas interprété, dans le cas contraire elle est complète.*
- *Une interprétation I satisfait la formule A (I est un modèle de A) si $I(A) = Vrai$. On le note $I \models A$.*
- *Une interprétation I falsifie la formule A (I est un contre modèle de A) si $I(A) = Faux$. On note $I \not\models A$.*
- *Une formule A est consistante (satisfaisable) si elle admet au moins un modèle. Dans le cas contraire, elle est inconsistante (insatisfaisable).*

Exemple 4.5

Soit la théorie suivante $T = (\{S\}, \{h\}, \emptyset, A)$ où $A = (\forall x, h(x,x) = x) \wedge (\forall x \forall y, h(h(x,y),x) = y)$. On associe à S le domaine $D_n = \{0, \ldots, n-1\}$ et Lorsque $n = 4$ cette théorie possède plusieurs modèles dont un est donné par l'interprétation I_h suivante :

I_h	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

Ceci correspond aux affectations : $h(0,0) = 0$, $h(0,1) = 2$, etc... ✍

Dans la section suivante nous introduisons des notations et des conventions qui facilitent la présentation et la lecture des algorithmes et des méthodes que nous définissons dans la suite de cette partie.

4.1.3 Conventions et notations

Tout d'abord, nous considérons les prédicats comme des fonctions dont le domaine d'arrivée est le type booléen qui est alors un nouveau type intégré à ceux de la théorie. Ainsi, on réduit une théorie $T = (\mathcal{S}, \mathcal{F}, \mathcal{P}, A)$ au triplet $T = (\mathcal{S}, \mathcal{F} \cup \mathcal{P}, A)$.

On considère un domaine de n individus comme l'ensemble des n premiers entiers. Les types sont disjoints et les signatures des fonctions sont connues, il n'y a donc pas de confusion entre les différents individus.

Définition 4.10 (Terme Terminal) *Soit $f : s_1, s_2, \ldots, s_k \mapsto s$ un symbole fonctionnel, on appelle terme terminal ou cellule («ground cell» en anglais) un terme de la forme $f(e_1, e_2, \ldots, e_k)$ où $\forall i \in \{1, \ldots, k\}$, $e_i \in D_{s_i}$.*

Étant donné un ensemble de domaines finis, la définition d'une interprétation consiste à donner une valeur à chaque terme terminal. Elle est facilement représentable par des *tables d'opérations* de chaque symbole fonctionnel et de chaque prédicat.

Dés lors qu'il n'y a pas d'ambiguité sur l'interprétation I en question nous notons $f(e_1, \ldots, e_k) = e$ en lieu et place de $I(f(e_1, \ldots, e_k)) = e$ (c'est ce que nous avons fait dans l'exemple 4.5).

Enfin, nous autorisons la présence d'éléments de domaines dans les termes. Ce qui nous permet de prendre en compte des formules telles que $\forall x, f(x, 0) = 0$.

4.1.4 Formes normales conjonctives

Définition 4.11 *Une formule F est dite sous forme normale conjonctive si elle est de la forme*

$$F = \bigwedge_{i=1}^{n} \forall x_i \bigvee_{j=1}^{n} L_{ij}$$

Où les L_{ij} sont des littéraux. Les clauses $(\forall x_i \bigvee_{j=1}^{n} L_{ij})$ sont également appelées axiomes.

On peut omettre les quantificateurs dans la représentation de formules CNF car toutes les variables sont quantifiées universellement. Toute formule sous forme CNF est indifférement notée comme une conjonction ou comme un ensemble de clauses, comme cela est le cas pour la forme CNF de la logique propositionnelle.

Toute formule possède une formule mise sous forme clausale qui lui est équivalente. Cette transformation nécessite, entre autre, une étape de *skolémisation*. Elle consiste à remplacer les quantificateurs existentiels «∃» par des nouveaux symboles fonctionnels que l'on nomme souvent fonctions de skolem. Pour plus de détails sur les formules de skolem le lecteur pourra se référer à [37].

La mise sous forme clausale d'une formule peut nécessiter une augmentation exponentielle de sa taille. Des algorithmes utilisant les techniques de renommage permettent d'obtenir des transformations polynomiales en taille. Par exemple, un algorithme optimal lorsque la formule ne possède pas le connecteur «↔» est donné par Thierry BOY DE LA TOUR [19].

Théories équationnelles

Les théories équationnelles sont un sous–ensemble de la logique du premier ordre. Le seul prédicat autorisé est celui de l'égalité. On a souligné que cela ne pose pas de problèmes car on peut coder les prédicats comme des fonctions particulières. De plus, elles ne possèdent qu'un seul type. Les formules sont sous forme normale conjonctive. Certains générateurs de modèles finis que nous introduisons dans la prochaine section utilisent les théories équationnelles comme théories d'entrée.

4.2 Quelques générateurs de modèles finis

Comme nous l'avons vu dans le chapitre 1, les méthodes de résolution du problème SAT sont souvent basées sur l'algorithme de DAVIS et PUTNAM. De nombreuses améliorations de cet algorithme ont été proposées : recherche d'heuristiques efficaces, technique de «look ahead», structure de

données, etc...A l'opposé, les recherches sur la génération de modèles finis ont donné lieu à de nombreuses méthodes différentes par leur formalisme et par leur exploration de l'espace de recherche. Nous nous proposons maintenant d'en détailler quelques–unes parmi les plus connues. Durant ces rappels, nous expliquons brièvement quelles stratégies sont utilisées par les générateurs de modèles finis pour supprimer des symétries. Nous détaillerons plus profondement ces stratégies dans le chapitre 6.

4.2.1 La méthode MACE

Le générateur de modèles finis MACE a été proposé par William MCCUNE dans [71]. La méthode MACE transforme, tout d'abord, une théorie de la logique du premier ordre en instance SAT de la logique propositionnelle. Les étapes de la transformation sont résumées dans la figure 4.1. Le système MACE est également constitué d'une procédure de résolution basé sur DAVIS et PUTNAM.

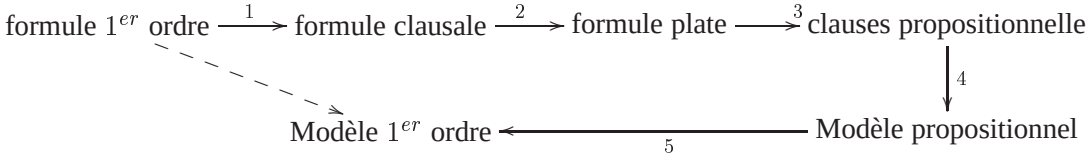


Figure 4.1. *Transformation d'une théorie dans* MACE

Les étapes de génération de modèles sont les suivantes :

1. La transformation en forme clausale du premier ordre. L'auteur utilise pour cela le démonstrateur de théorème OTTER [72].
2. L'applatissement des clauses et la suppression des symboles fonctionnels en introduisant des nouveaux prédicats. La suppression d'un symbole fonctionnel f d'arité k se fait en introduisant un nouveau prédicat P_f d'arité $k+1$:$f(x_1,\ldots,x_k) = y \Leftrightarrow P_f(x_1,\ldots,x_k,y)$. De nouvelles clauses sont introduites de façon à assurer la totalité et l'unicité des images. L'applatissement des clauses se fait en réécrivant les termes $P[t]$ [1] ne contenant pas l'égalité par $P[x] \vee t \neq x$. L'égalité $\alpha = \beta$ est transformée en deux clauses $(\alpha \neq x \vee \beta = x)$ et $(\alpha = x \vee \beta \neq x)$.
3. La génération de la forme clausale propositionnelle
4. Le calcul de modèles de la forme clausale propositionnelle en utilisant la méthode DP.
5. La transformation de tout modèle obtenu en un modèle du premier ordre.

Exemple 4.6
L'algorithme de transformation en formule SAT (étape 2) transforme l'axiome $f(g(x),x) = e$ en deux clauses :

[1] le terme t apparait comme argument du prédicat P

– $\neg P_g(x,y) \vee \neg P_e(z) \vee P_f(y,x,z)$
– $\neg P_g(x,y) \vee P_e(z) \vee \neg P_f(y,x,z)$ ✍

Le générateur MACE s'est révélé efficace dans la pratique. Nous l'utiliserons lors des expérimentations afin de voir l'impact au niveau du temps et de l'espace de recherche d'une méthode transformant en logique propositionelle des théories du premier ordre.

Une méthode similaire, nommée MODGEN, a été proposée par Sun KIM et Hantao ZHANG [61]. Les deux auteurs choisissent également de traiter les théories du premier ordre par des procédures de décision de la logique propositionnelle. Les raisons qu'ils invoquent sont que le problème SAT est le coeur des problèmes NP–complets, et beaucoup de problèmes peuvent facilement être convertis en instance du problème SAT. De plus, il existe une grande variété de procédures de décision disponibles pour SAT.

Ils utilisent également le démonstrateur OTTER pour transformer une formule en forme clausale. Lors de la suppression des symboles fonctionnels, un cas particulier est appliqué lorsqu'une fonction de skolem f n'apparaît que dans une seule clause $C(f(x))$: si le domaine de $f(x)$ est $\{a_1, \ldots, a_n\}$, on peut alors remplacer $C(f(x))$ par $C(a_1) \vee \ldots \vee C(a_n)$. Cette exception supprime un grand nombre de clauses lors de la transformation en logique propositionnelle.

4.2.2 La méthode FMSET

La méthode FMSET «Finite Model Search for Equational Theories», a été réalisée par Belaid BENHAMOU et Laurent HENOCQUE [15]. FMSET est limitée aux théories équationnelles. Cette méthode essaie de concilier le pouvoir d'expression de la logique du premier ordre avec l'efficacité de propagation de la logique propositionnelle.

La génération comporte un pré–traitement qui consiste à transformer la théorie équationnelle en un ensemble de clauses simples, c'est à dire en un ensemble de clauses où tous les littéraux ont un degré égal à 1. Quand elle est appliquée à des clauses de longueur 1, cette transformation permet de n'obtenir que des clauses de horn. Elle fonctionne de la manière suivante :

1. les littéraux de la forme $(t_1 = t_2)$ deviennent $((t_1 \neq var[t1]) \vee (t_2 = var[t_1])) \wedge ((t_2 \neq var[t_2]) \vee (t_1 = var[t_2]))$
2. les littéraux de la forme $-t_1 \neq t_2)$ deviennent $((t_1 \neq var[t_1]) \vee (t_2 \neq var[t_1])) \wedge ((t_2 \neq var[t_2]) \vee (t_1 \neq var[t_2]))$
3. $f(t_1, \ldots, t_k) = z$ devient $(t_1 \neq var[t_1]) \vee \ldots \vee (t_k \neq var[t_k]) \vee (f(var[t_1], \ldots, var[t_k]) = z)$.
4. $f(t_1, \ldots, t_k) \neq z$ devient $(t_1 \neq var[t_1]) \vee \ldots \vee (t_k \neq var[t_k]) \vee (f(var[t_1], \ldots, var[t_k]) \neq z)$.

$var[t]$ désigne une nouvelle variable remplaçant le terme si celui–ci contient un symbole fonctionnel ou désigne t si t est une variable. Nous montrons ci–dessous un exemple d'une telle transformation qui est sensiblement identique à celle qu'opère MACE.

Exemple 4.7
Soit l'axiome $f(f(x,y),z) = f(x,f(y,z))$ représentant l'associativité de la fonction f. La transformation en clauses simples de ce dernier axiome génère l'ensemble de clauses suivant :

- $r \neq f(x,y) \vee s \neq f(r,z) \vee f(x,t) = s \vee t \neq f(y,z)$
- $u \neq f(y,z) \vee v \neq f(x,u) \vee f(x,y) \neq w \vee v = f(w,z)$

Où r, s, t, u, v, w sont des nouvelles variables, rajoutées par les transformations. ✍

La transformation en calcul propositionnel requiert une grande consommation de mémoire à cause des nombreuses clauses produites. De plus certaines clauses, directement satisfaites, se révèlent inutiles. Pour tenter de résoudre ces problèmes, les auteurs ont proposé une représentation intermédiaire entre la logique propositionnelle et la logique du premier ordre. Seules les clauses propositionnelles nécessaires pour la résolution sont effectivement générées. Dans l'exemple 4.8 nous montrons brièvement le fonctionnement de cette stratégie. Ensuite la génération se fait par un algorithme de type backtrack où les noeuds sont étiquetés par les cellules ce et où chaque noeud possède deux branches, l'une satisfaisant $ce = e$ et l'autre satisfaisant $ce \neq e$. De même que SEM, FMSET utilise l'heuristique LNH afin de supprimer des interprétations isomorphes.

De cette manière, FMSET est une méthode qui allie les qualités de propagations de la logique propositionnelle tout en conservant un formalisme de la logique du premier ordre.

Exemple 4.8
Soit la clause $(f(z) \neq y) \vee (h(x) \neq y) \vee (g(y) = x)$. Si $f(0) = 1$ on l'unifie à $f(z) \neq y$ par la substitution $(y \to 1) \cup (z \to 0)$ et on obtient la nouvelle clause $h(x) \neq 1 \vee g(1) = x$.
Si plus tard on a $g(1) \neq 3$ on propagera alors $h(3) \neq 1 \ldots$ ✍

4.2.3 La méthode SEM

La méthode SEM (System for Enumerating Model) a été introduite par Hantao ZHANG et Jian ZHANG [106]. C'est un générateur de modèles finis pour les théories multi–types mises sous forme skolémisées. SEM combine les techniques de SATO [101] (voir section 1.5.3) pour les structures de données et celles de FALCON [103] qui est générateur de modèles finis pour des théories équationnelles.

La méthode SEM effectue une phase de pré–traitement. Celle–ci calcule l'ensemble des instances terminales résultant de la substitution des variables de l'ensemble des formules de départ par les individus des domaines. La propagation des contraintes tire bénéfice de cette transformation. Nous donnons un exemple simple d'une telle transformation.

Exemple 4.9
Soit une théorie $T = \{\{s\}, \{f,h\}, \{f(x,h(y)) = y\}\}$. On associe le domaine $D_3 = \{0,1,2\}$ au type s. La phase de pré–traitement crée alors $3^2 = 9$ instances terminales parmi lesquelles : $f(0,h(0)) = 0$, $f(1,h(0)) = 0$, $f(2,h(0)) = 0$, $f(0,h(1)) = 1, \ldots$ ✍

La méthode SEM considère la génération de modèles finis comme un problème CSP particulier où les variables sont les termes terminaux, les domaines sont les domaines associés aux types des termes terminaux et les contraintes sont les axiomes terminaux. La recherche parcourt un arbre n–aire «en profondeur d'abord» dans le style de celui de DAVIS et PUTNAM. Les noeuds de cet arbre sont étiquetés par les termes terminaux et les arêtes par les valeurs prises par ces cellules. La description de SEM est donnée par l'algorithme 4.1. Cette procédure utilise les paramètres suivants :

- I : ensemble des termes terminaux ce interprétés, $I = \{(ce = e) | e \in Sort(ce)\}$
- B : ensemble des termes terminaux non instanciés avec leurs valeurs possibles, $B = \{(ce, D)$ tel que $D \subseteq Sort(ce)\}$
- A : ensemble des clauses simplifié par l'interprétation I.

Algorithme 4.1 Schéma algorithmique de SEM

Fonction SEM(I, B, A)
Retourne : Vrai Si A est satisfaisable par I, Faux sinon
Début
Si $B = \emptyset$ **Alors** Retourner Vrai %%I est un modèle
Choisir $(ce_i, D_i) \in B$
$B = B - (ce_i, D_i)$
Si $D_i = \emptyset$ **Alors** Retourner Faux %%Pas de modèle
Pour Chaque $e \in D_i$ %%Point de choix
    Faire
    Si (Propa$(I \cup (ce_i = e), B, A) \neq$ Faux) **Alors** Retourner sem(I, B, A) %%algorithme 4.2
Retourner Faux %% Pas de modèle
Fin

Jian et Hantao ZHANG proposent dans [105] une suite de règles de propagations des contraintes pour la génération de modèles finis. Le système SEM utilise, quelquefois de manière restrictive, des règles proposées dans cet article. L'ensemble de ces règles de propagations conduit à l'algorithme 4.2.

L'algorithme `Propa` modifie le triplet (I, B, A) jusqu'à ce qu'un point fixe soit atteint. Les termes terminaux ce de I sont remplacés par les valeurs e qu'on leur affecte. On recherche ensuite des clauses unitaires pour affecter un nouveau terme terminal (clauses unitaires de la forme ce, $\neg ce$, $ce = e$) ou des clauses unitaires de la forme $ce \neq e$ pour supprimer des valeurs incompatibles. Puis on recherche des cellules n'ayant qu'une valeur possible.

Pour propager efficacement ces contraintes, une structure d'arbre est utilisée pour coder les termes. Chaque terme est un arbre, les feuilles sont étiquetées par des valeurs de domaines et les noeuds internes par des symboles fonctionnels. Les noeuds possèdent un compteur correspondant au nombre de leur fils qui sont des noeuds internes.

Algorithme 4.2 Propagation des contraintes sous SEM

Fonction Propa(Var I, Var B, Var A)
Retourne : Vrai si l'interprétation I est valide dans A, faux sinon.
Début
Répéter 1
  Pour chaque $(ce = e) \in I$ Remplacer ce par e dans A 2
  Si $\perp \in A$ **Alors** Retouner Faux %%Contradiction 3
  Pour Chaque clause unitaire $l \in A$ **Do** 4
    Si l est un terme terminal ce (resp. $\neg ce$) et $(ce, D) \in B$ **Alors** 5
      $B = B - \{(ce, D)\}$ 6
      $I = I \cup \{(ce = True)\}$ (resp. $\{(ce = False)\}$) %% Affectation 7
    Si $(l = EQ(ce, e))$ et $((ce, D) \in B)$ **Alors** 8
      $B = B - \{(ce, D)\}$ 9
      $I = I \cup \{(ce = e)\}$ %% Affectation 10
    Si $l = \neg EQ(ce, e)$ et $(ce, D) \in B$ **Alors** 11
      $D = D - \{e\}$ %%Élimination de Valeur 12
  Pour Chaque $(ce, D) \in B$ **Faire** 13
    Si $D = \emptyset$ **Alors Retourner** Faux %%Contradiction 14
    Si $D = \{e\}$ **Alors** %%1 seule valeur possible 15
      $B = B - \{(ce, D)\}$ 16
      $I = I \cup \{(ce, e)\}$ 17
Jusqu'à plus de changement dans A 18
Retourner Vrai 19
Fin

Afin de supprimer certaines symétries triviales, SEM utilise l'heuristique LNH «Least Number Heuristic». L'heuristique LNH généralise à tout l'arbre de recherche l'idée qu'au départ toutes les variables sont interchangeables et donc symétriques. De manière informelle, un ensemble de valeurs est interchangeable, si la permutation de ses éléments ne change pas la théorie. Nous expliquons le fonctionnement de cette heuristique dans la section 6.2.1.

Jian et Hanto ZHANG ont proposé un générateur de modèles finis combinant le générateur SEM avec un algorithme de recherche locale [107]. Cette méthode mixte a permis de résoudre quelques problèmes ouverts. C'est à notre connaissance le seul générateur de modèles finis existant dans la littérature qui soit incomplet.

Une méthode nommée STORK légèrement similaire à SEM a été proposée par Olga SHUMSKY et al. [86]. Ses auteurs utilisent également une structure d'arbre pour les axiomes. Pour propager au mieux les contraintes négatives, ils utilisent une heuristique qui rajoute, en cours de recherche, des

axiomes. Nous présentons au chapitre 5 d'autres façons de propager efficacement des contraintes négatives.

4.2.4 La méthode FINDER

Le système FINDER (FINite Domain EnumeratoR) est une réalisation de John SLANEY [90, 93, 91]. Il traite des théories multi–types. C'est un générateur fondé sur l'énumération et le retour arrière. FINDER ne traite pas des symétries. Par contre, des réfutations sont rajoutées à l'ensemble des contraintes. Cela permet de ne jamais échouer deux fois pour la même raison. L'exemple suivant explicite cette notion.

Exemple 4.10
On considère la théorie 4.5 contenant les axiomes suivant :$\forall x \forall y, h(h(x,y),z) = h(x,h(y,z))$ codant l'associativité de la loi h.
Supposons que : $h(0,1) = 0$, $h(0,3) = 1$, $h(1,3) = 1$, $h(3,3) = 1$.
On a alors, $h(h(0,3),3) \neq h(0,h(3,3))$. L'axiome d'associativité est donc contredit.
Afin d'éviter d'échouer pour la même raison, FINDER rajoute la contrainte $h(0,1) \neq 0 \vee h(0,3) \neq 1 \vee h(1,3) \neq 1 \vee h(3,3) \neq 1$ à l'ensemble des axiomes. ✍

Algorithme 4.3 Génération de modèle par FINDER

Fonction FINDER(A : Formule ; I : ensemble des termes terminaux)
Retourne : Vrai si A est satisfaisable par I, Faux sinon.
Début
Si (tous les termes terminaux sont interprétés)
  Alors Retourne Test(S)
  Sinon
  Choisir un terme terminal non instancié ce
  Pour toutes les valeurs possibles de ce
    Faire
    Supprimer les valeurs incompatibles dans S.
    Si (aucun élément de S non interprété n'a de domaine vide)
      Alors Retourner FINDER(S)
      Sinon Retourner Faux
Fin

La procédure de recherche de FINDER est décrite dans l'algorithme 4.3. La fonction `Test` renvoie Vrai si I est bien un modèle, sinon elle rajoute les réfutations à l'ensemble des axiomes. Cette base de données de réfutations peut énormément croître, mais la méthode FINDER dispose d'échappatoires pour limiter sa taille.

Une version combinant le prouveur de théorème de William MC CUNE [72], nommé OTTER, et de FINDER a été proposée sous le nom de SCOTT (Semantically COnstrained OTter) [89, 92]. Cela permet de rechercher des preuves de théorèmes et en même temps des contre–exemples de conjectures.

4.2.5 La méthode FMC

Nicolas PELTIER a proposé FMC (Finie Model Constructor) dans [76, 77]. Cette méthode est sensiblement différente des deux premières. Elle ne propage pas des contraintes mais teste la consistance d'une interprétation complète. Le système FMC peut donc s'appliquer facilement à une formule quelconque de la logique du premier ordre, voire à d'autres logiques.

La première étape consiste à créer un ordre sur les termes terminaux qui se généralise facilement à l'ensemble des interprétations complètes. L'ordre généré est alors utilisé par l'algorithme de recherche qui est décrit dans l'algorithme 4.4.

Algorithme 4.4 Algorithme de recherche de FMC

Fonction FMC(A : Formule) : Booléen
Retourne : Vrai si A est satisfaisable, Faux sinon.
Début
$I = I_0$ %% I_0 plus petite interprétation
Tant que Succ(I) existe et $I \not\models A$
Faire I = Succ(I) %% Successeur de I dans l'ordre
Si $I \models A$ **Alors** Retourner Vrai **Sinon** Retourner Faux
Fin

FMC teste les interprétations les unes après les autres par ordre croissant (fonction successeur `succ`) jusqu'à en trouver une qui soit un modèle de la formule A ou jusqu'à ce qu'il n'y ait plus d'interprétations possibles. L'algorithme de base qui parcourt successivement toutes les interprétations est bien entendu inexploitable dans la pratique. Le nombre total d'interprétations d'une fonction binaire sur un domaine de 4 éléments est égal à 4^{16} soit un total de 4 294 967 296. Dans FMC on utilise une fonction Φ à la place de la fonction `Succ` et qui vérifie les propriétés suivantes :

- Si $I \not\models A$, alors $\Phi(I) > I$ *Terminaison*
- Si $I \not\models A$ et si $\Phi(I)$ existe, alors $\forall J \ / \ I < J < \Phi(I)\ J \not\models A$ *Complétude*
- Si $I \not\models A$ et si $\Phi(I)$ n'existe pas, alors $\forall J \ / \ I < J\ J \not\models A$ *Complétude*

La fonction Φ saute des interprétations dont on sait qu'elles sont des contre modèles de la formule A. Elle utilise pour cela la notion de réfutation de modèles :

Définition 4.12 *Soit I une interprétation dans une théorie A telle que $I \not\models A$, une réfutation R de A*

par rapport à I est un sous–ensemble des termes terminaux vérifiant :

$$\forall J, I =_{|R} J \Rightarrow J \not\models A$$

Une réfutation dans une interprétation d'une théorie A est un ensemble de termes terminaux responsables de la falsification de A par l'interprétation. La fonction $\Phi_R(I) = J$ associée à une réfutation R de A par rapport à I, choisit J comme la plus petite interprétation plus grande que I qui vérifie $I \neq_{|R} J$.

La fonction de saut de FMC utilise une amélioration de la notion de réfutation en considérant l'ensemble des cellules responsables de l'échec depuis le début de la recherche, c'est à dire l'union de toutes les réfutations générées dans les interprétations passées. C'est la notion de *réfutation couvrante*.

L'ordre sur les termes terminaux va influer directement sur les performances de la méthode FMC. Or, dans un algorithme de type backtrack classique (DP pour SAT, foward checking pour CSP) il est difficile de choisir la bonne variable à interpréter. Les heuristiques de choix qui ont été proposées dépendent souvent de l'ensemble courant des clauses (voir section 1.3). C'est pour cela qu'il nous semble difficile de déterminer, avant de débuter la recherche, un ordre global des termes terminaux qui soit performant tout au long de la recherche.

La méthode FMC n'utilise pas l'heuristique LNH, mais teste avec certaines restrictions si l'interprétation courante est symétrique à des interprétations précédentes. En restreignant la détection à certains isomorphismes, il restera certainement des interprétations redondantes, mais la détection se fera rapidement (en un temps linéaire par rapport à la réfutation couvrante). Ainsi, la méthode FMC détecte et supprime des symétries dynamiquement (pendant la recherche).

4.2.6 La méthode MGTP

Le générateur de modèles MGTP (Model Generation Theorem Prover) a été proposé par Masayuki FUJITA et al. [48, 91]. Il est basé sur les principes de la méthode SATCHMO [68]. Il fonctionne sur une machine parallèle possédant jusqu'à 256 processeurs et est écrit dans le langage déclaratif de programmation parallèle logique KL1. La méthode MGTP fonctionne à partir de règles de résolution et de règles de branchement. Tous les axiomes du problème sont exprimés de la manière suivante :

$$A_{l+1} \wedge \ldots \wedge A_m \rightarrow A_1 \vee \ldots \vee A_l$$

L'algorithme 4.5 décrit brièvement le schéma de recherche de MGTP. Les deux règles suivantes sont appliquées jusqu'à ce que l'ensemble des candidats ne change plus.

- *Règle d'extension* : Si il existe un axiome $A \rightarrow C$ et une substitution σ telle que $A\sigma$ est satisfait dans $\mathcal{S}$ et que $C\sigma$ n'est pas satisfait dans $\mathcal{S}$ alors rajouter $C\sigma$ dans $\mathcal{S}$.
- *Règle de rejet* : Si il existe un axiome $A \rightarrow Faux$ dans Σ et une substitution σ telle que $A\sigma$ est satisfait dans $\mathcal{S}$ alors supprimer $A\sigma$ de $\mathcal{S}$.

Algorithme 4.5 Génération de modèles par MGTP

Fonction MGTP(Σ : Ensemble d'axiomes ;$\mathcal{S}$: Ensemble des candidats au modèle)
Retourne : Vrai si A est satisfaisable, Faux sinon.
Début
Répéter
    Appliquer la règle d'extension
    Appliquer la règle de rejet
Jusqu'à plus de changements dans $\mathcal{S}$
Si $\mathcal{S}$ est vide **alors** Retourner Faux **Sinon** Retourner Vrai
Fin

Après utilisation de cette procédure l'ensemble $\mathcal{S}$ sera vide si la théorie est inconsistante, dans le cas contraire il contiendra un modèle. Le mécanisme de branchement se fait donc en choisissant une clause, un choix heuristique choisit celle contenant le moins de littéraux possibles. La parallélisation s'effectue lors de la séparation en sous–branches de l'arbre de recherche. Chaque branche étant traitée indépendamment, aucune communication n'est nécessaire entre les processeurs (data–parallélisme). Une accélération linéaire, en fonction du nombre de processeurs, a été observée sur de nombreux problèmes.

Afin de supprimer les isomorphismes de certaines théories, les auteurs de MGTP proposent de poser des contraintes additionnelles exprimant les symétries dans la formalisation du problème [91].

4.3 Conclusion et discussion

La figure 4.2 donne une classification possible des différentes approches que nous avons vues tout au long de cette section. La classification porte sur le formalisme adopté par l'algorithme de recherche et sur l'approche utilisée pour supprimer des isomorphismes.

Lorsque nous nous sommes intéressé à la génération de modèles finis pour des théories de la logique du premier ordre, nous avons en premier lieu opté pour une transformation des fomules en logique propositionnelle. Les partisans de cette transformation présentent deux arguments principaux. D'une part, la propagation des contraintes est très efficace en logique propositionnele. De l'autre, il existe une grande variété d'algorithmes, complets ou incomplets, de résolution du problème SAT.

Pour autant, en utilisant ce choix, nous nous sommes trouvé confrontés à certains obstacles. Le premier d'entre eux est la taille des formules CNF propositionnelles qui traduisent le problème initial. En effet, cette transformation peut nécessiter quelques dizaines de milliers de clauses et de variables. Elle s'avère souvent inexploitable dans la pratique. Le second obstacle est la perte de la structure de la théorie de départ qui est pourtant très importante. Cette structure peut servir pour mettre au point

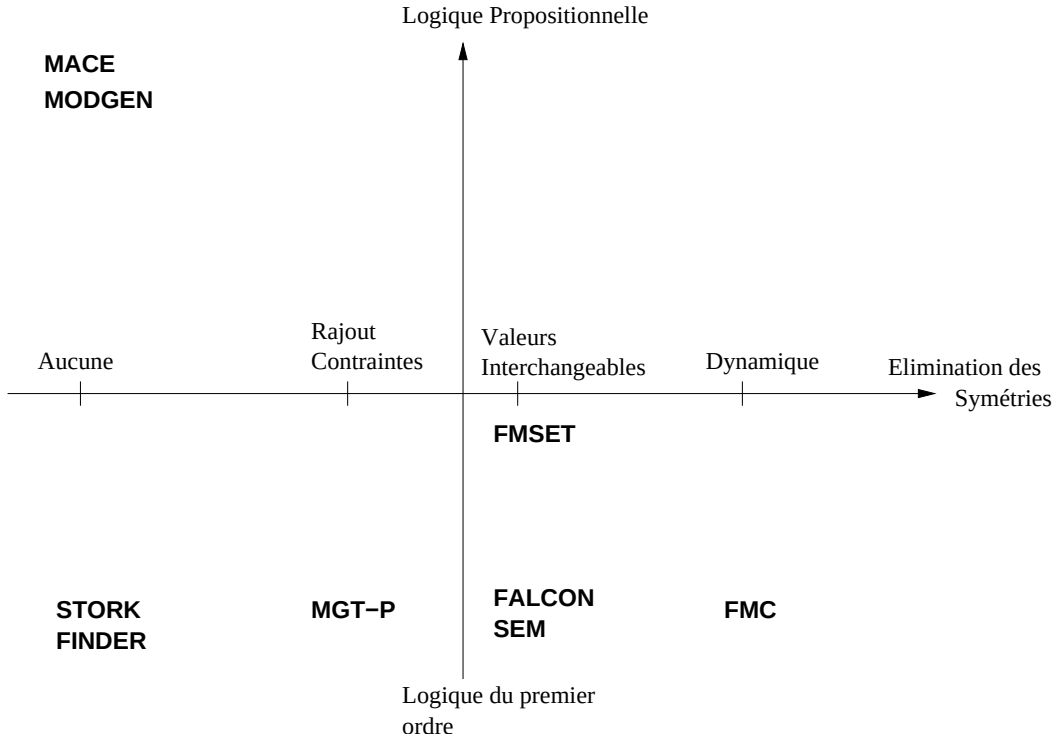


Figure 4.2. *Classification des générateurs de modèles finis*

des heuristiques spécialisées pour ces problèmes. De plus, les théories de la logique du premier ordre ont souvent un grand nombre d'interprétations symétriques. Et la génération de modèles finis ne peut pas être efficace sans un traitement adapté pour en réduire le nombre. Il est alors bien plus aisé de détecter ces symétries en utilisant la représentation de la logique du premier ordre.

Nous avons donc choisi de garder le formalisme de la logique du premier ordre. De plus, nous avons opter pour les générateurs qui propagent des contraintes (SEM, FALCON, FMSET...). Nous considérons également que nous ne travaillons qu'avec des théories dont les formules sont sous forme CNF. Néamoins, certaines stratégies que nous introduirons dans la suite de ce mémoire peuvent facilement être adaptées vers d'autres types de générateurs de modèles finis et de formules..

Le premier travail auquel nous nous sommes attachés (cf. chapitre 5) à consisté à améliorer la propagation des contraintes et à élaborer des heuristiques de choix de variables pour ces générateurs. Le deuxième travail concerne les symétries. Nous définissons plus précisement la notion de symétries dans le chapitre 6. Nous proposons ensuite deux approches complémentaires de détection et de suppression d'interprétations isomorphes. Le chapitre 7 contient une étude expérimentale de l'ensemble des stratégies proposées.

Chapitre 5

Propagations de Contraintes et Heuristiques

LA PROPAGATION DES CONTRAINTES a été beaucoup étudiée pour les problèmes de satisfaction de contraintes (consistance d'arcs, de chemins) et pour le problème SAT (technique de «look-ahead»). Dans la plupart des cas, c'est une étape difficile et coûteuse qui nécessite un compromis entre la réduction de l'espace de recherche et le temps consacré à cette réduction.

Nous introduisons dans ce chapitre, différentes voies possibles pour améliorer la propagation de contraintes des générateurs de modèles finis. Dans la section 5.1, nous proposons deux techniques qui améliorent la méthode SEM [106] qui est la méthode de base que nous avons choisie. Leur but est de propager le plus de contraintes négatives (information du type $f(1,2) \neq 1$ par exemple). Dans la section 5.2, nous introduisons diverses heuristiques qui optimisent la propagation d'affectations (du style $f(1,2) = 0$). Elles s'appuient sur la syntaxe de la théorie pour déterminer un ordre de préférence sur les symboles fonctionnels.

5.1 Filtrage des domaines

L'algorithme 4.2 (voir chapitre 4, page 57) de propagation de SEM ne propage des assignations négatives, appelées également contraintes négatives, que lorsque des mono–littéraux de la forme $ce \neq e$ existent dans l'ensemble courant de clauses (voir la ligne 12 de l'algorithme 4.2). Dans les autres cas, seules les affectations, appelées aussi contraintes positives, sont propagées. Ce qui tend à accroître inutilement le nombre de branches nécessaires que doit parcourir l'algorithme de recherche de SEM (algorithme 4.1).

Regardons de plus près la théorie de l'exemple 4.5. Elle contient les deux axiomes $h(x,x) = x$ et $h((x,y),x) = y$. Les affectations $h(0,0) = 0$, $h(1,1) = 1$ sont propagées grâce au premier axiome. Supposons qu'au point de choix suivant, nous interprétions $h(0,1)$ à 0. La clause terminale

$h(h(0,1),0) = 1$ est réduite à $h(0,0) = 1$, ce qui est incompatible avec les affectations précédentes. Si plusieurs points de choix héritent de $h(0,1)$ comme terme terminal à affecter, l'arbre de recherche contiendra plusieurs branches affectées par $h(0,1) = 0$ conduisant directement à l'inconsistance. Ces échecs peuvent être évités si l'interprétation de $h(0,0) = 0$ propage la contrainte $h(0,1) \neq 0$.

Nous allons voir comment des valeurs incompatibles avec l'interprétation courante peuvent être éliminées des domaines de leurs cellules correspondantes. La première approche «statique» est réalisée en pré–traitement et consiste en une réécriture de certaines clauses de l'ensemble de clauses décrivant la théorie. La seconde approche «dynamique» réalise un filtrage à chaque noeud de l'arbre de recherche. Elle consiste en une technique de type «look ahead» dans le même esprit que celles existant pour les méthodes énumératives du problème SAT et CSP (cf section 1.3.2). Cette stratégie dynamique peut également être utilisée pour élaborer des heuristiques. Les résultats de ces travaux ont été publiés à la conférence internationale «Conference on Automated Deduction» (CADE17) [4].

5.1.1 Transformation des clauses (Approche statique)

Comme nous venons de le faire remarquer, la méthode SEM propage des contraintes négatives uniquement lorsque des mono–littéraux négatifs de la forme $ce \neq e$ existent explicitement dans l'ensemble de clauses. Nous essayons alors de faire apparaître de tels littéraux en réécrivant certaines clauses. En utilisant une technique d'applatissement comme celle de la méthode FMSET [15] (section 4.2.2), nous pouvons transformer des clauses décrivant le problème en d'autres clauses logiquement équivalentes et contenant des littéraux négatifs. L'application systématique de cette technique à toutes les clauses propagerait autant d'informations qu'une approche basée sur la logique propositionelle telle que DP, mais elle incluerait également le même problème de consommation mémoire. Pour des raisons d'efficacité, nous limitons donc cette réécriture à certains littéraux. Comme nous le verrons au chapitre 7, cette simple transformation diminue drastiquement l'espace de recherche et le temps d'exécution du générateur de modèles finis SEM.

Définition 5.1 *Si f et g sont deux symboles fonctionnels et $x_1, , x_2, \ldots, x_l$ sont des variables alors on appelle littéral réductible tout littéral de la forme suivante :*

$$f(x_1, \ldots, x_m, g(x_{k+1}, \ldots, x_l), x_{m+1}, \ldots, x_k) = x_0$$

Définition 5.2 *Une clause contenant un littéral réductible est appelée clause réductible.*

En utilisant la transformation des clauses de la méthode FMSET [15] nous pouvons réécrire chaque littéral réductible $f(x_1, \ldots, x_m, g(x_{k+1}, \ldots, x_l), x_{m+1}, \ldots, x_k) = x_0$ en $v \neq g(x_{k+1}, \ldots, x_l) \vee f(x_1, \ldots, x_m, v, x_{m+1}, \ldots, x_k) = x_0$. Cette transformation préserve la sémantique de la clause et introduit un littéral négatif $v \neq g(x_{k+1}, \ldots, x_l)$ nécessaire à SEM pour éliminer des valeurs incom-

patibles avec l'interprétation courante. L'inconvénient majeur de cette technique est l'ajout d'une variable auxiliaire v qui entraine un besoin de mémoire supplémentaire. Formellement, nous avons :

Proposition 5.1 *Tout littéral de la forme* $f(x_1, \ldots, x_m, g(x_{k+1}, \ldots, x_l), x_{m+1}, \ldots, x_k) = x_0$ *est sémantiquement équivalent à la clause*

$$f(x_1, \ldots, x_m, v, x_{m+1}, \ldots, x_k) = x_0 \vee v \neq g(x_{k+1}, \ldots, x_l) \quad (1)$$

Preuve Soit I un modèle de $f(x_1, \ldots, x_m, g(x_{k+1}, \ldots, x_l), x_{m+1}, \ldots, x_k) = x_0$.
Nous devons prouver que toute interprétation J identique à I, sauf peut–être sur v, est un modèle de $f(x_1, \ldots, x_m, v, x_{m+1}, \ldots, x_k) = x_0 \vee v \neq g(x_{k+1}, \ldots, x_l)$. Il y a deux cas possibles :

- J est un modèle de $v \neq g(x_{k+1}, \ldots, x_l)$ et on peut conclure que J est un modèle de (1).
- J est un modèle de $v = g(x_{k+1}, \ldots, x_l)$ et alors, par hypothèse,
  J est un modèle de $f(x_1, \ldots, x_m, v, x_{m+1}, \ldots, x_k) = x_0$ donc J est un modèle de (1).

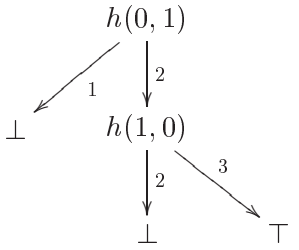
Réciproquement, Soit J un modèle de $f(x_1, \ldots, x_m, v, x_{m+1}, \ldots, x_k) = x_0 \vee v \neq g(x_{k+1}, \ldots, x_l)$. La variable v est universellement quantifiée. En particulier, l'interprétation I identique à J, excepté sur v où $I(v) = I(g(x_{k+1}, \ldots, x_l))$, est un modèle de $f(x_1, \ldots, x_m, v, x_{m+1}, \ldots, x_k) = x_0$ et est donc aussi un modèle de $f(x_1, \ldots, x_m, g(x_{k+1}, \ldots, x_l), x_{m+1}, \ldots, x_k) = x_0$. ❑

L'algorithme induit par cette proposition consiste en un pré–traitement. Avant le début de la recherche, les littéraux réductibles sont détectés et aplatis. Cette transformation est très simple et se fait en un temps linéaire par rapport au nombre de littéraux. Nous avons appelé CTSEM (Clause Transformation and SEM) la combinaison de la transformation de clauses et du générateur de modèles SEM. Voici un exemple de son fonctionnement.

Exemple 5.1
Reprenons la théorie de l'exemple 4.5 contenant les deux axiomes suivants :

- $\forall x, h(x, x) = x$
- $\forall x \forall y, h(h(x, y), x) = y$

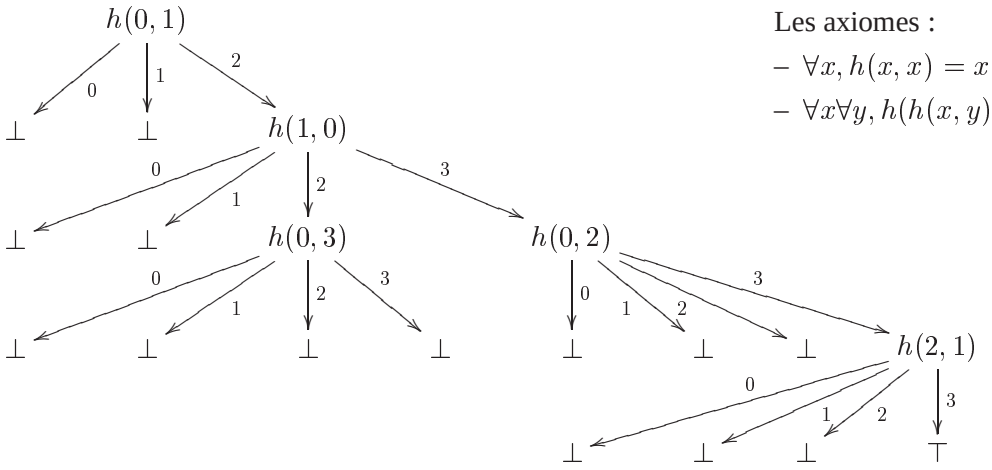
Nous recherchons des modèles sur un domaine $D_4 = \{0, 1, 2, 3\}$. La clause $h(h(x, y), x) = y$ est réductible. Elle est donc transformée en la clause équivalente $h(v, x) = y \vee v \neq h(x, y)$. Considérons maintenant la clause terminale $h(0, 0) = 1 \vee 0 \neq h(0, 1)$ issue de cette dernière.
L'affectation à 0 du terme terminal $h(0, 0)$ implique que le littéral $h(0, 0) = 1$ devient faux. Par conséquent le fait $h(0, 1) \neq 0$ est propagé et la valeur 0 est éliminée du domaine du terme terminal $h(0, 1)$. Les figures 5.1 et 5.2 montrent les arbres de recherche de SEM sur ce petit exemple, avec et sans la transformation des clauses. Le $\perp$ représente l'inconsistance et $\top$ représente l'obtention d'un modèle. Les branches non représentées ont été supprimées par des propagations de contraintes.
On observe bien sur la figure 5.2 que de nombreuses affectations conduisent directement à l'inconsistance. Ces branches ont été directement supprimées par l'élimination de valeurs incompatibles lorsque



Les axiomes :
- $\forall x, h(x, x) = x$
- $\forall x \forall y \forall v, h(v, x) = y \vee v \neq h(x, y)$

Figure 5.1. *Arbre de recherche avec la transformation de clauses réductibles*

l'on a transformé les clauses réductibles (figure 5.1). La propagation, si elle est faite rapidement, va également réduire le temps de génération de modèles finis.



Les axiomes :
- $\forall x, h(x, x) = x$
- $\forall x \forall y, h(h(x, y), x) = y$

Figure 5.2. *Arbre de recherche sans transformation des clauses réductibles*

5.1.2 Élimination dynamique des valeurs incompatibles

Lorsque l'on ne veut pas augmenter la taille de la théorie par l'introduction de variables auxiliaires, ou lorsque la théorie ne possède pas de clauses réductibles, une autre stratégie d'élimination de valeurs s'offre à nous. Avec la méthode AVAL (voir le chapitre 2), nous avons utilisé un algorithme de type «look–ahead» afin de détecter des mono–littéraux cachés. L'utilisation d'une technique similaire peut être appliquée aux générateurs de modèles finis afin d'éliminer certaines valeurs des domaines des termes terminaux qui ne participent pas aux solutions. De plus, elle induit une nouvelle heuristique dans le style des heuristiques UP pour choisir le prochain terme terminal à instancier.

Soit $B = \{(ce, D_{ce})\}$ l'ensemble des termes terminaux non instanciés et leurs valeurs possibles en un noeud donné de l'arbre de recherche. Soit C_I l'ensemble des axiomes simplifiés par l'ensemble des affectations I. Soit ce un terme terminal non instancié et $e \in D_{ce}$ une valeur possible du domaine

de ce. Si la propagation de l'instanciation $ce = e$ provoque une inconsistance, alors on peut supprimer la valeur e du domaine de ce. En d'autres termes, si la procédure `Propa`($I \cup \{(ce, e)\}, B, C$) retourne l'inconsistance, alors nous pouvons supprimer e de D_{ce}.

Afin de garder une méthode qui reste rapide malgré les appels supplémentaires à la procédure `Propa`, nous évitons de propager certaines valeurs qui s'avèrent compatibles avec l'affectation courante. Nous généralisons ainsi la proposition 2.1 (chapitre 2, page 26) que nous avons utilisée dans l'algorithme de recherche des littéraux impliqués RLI. Après l'appel à `Propa`($I \cup \{(ce, e)\}, B, C$), il y a deux cas possibles :

- $C_{I \cup \{(ce,e)\}}$ est inconsistant et alors $D_{ce} = D_{ce} - \{e\}$. C'est le cas de l'élimination de valeur.
- $C_{I \cup \{(ce,e)\}}$ n'est pas inconsistant. Les affectations $(ce_i = e_i)$ propagées durant le processus d'affectation sont toutes compatibles avec l'interprétation courante. La valeur e_i de ce_i ne sera jamais supprimée. Il est inutile de la prendre candidate à l'élimination.

Formellement, nous avons les propositions suivante :

Proposition 5.2 *Soient* $(ce, D_{ce}) \in B$ *et* $e \in D_{ce}$. *Si* $C_{I \cup \{(ce,e)\}} \models \bot$ *alors* C_I *est équivalent à* $C_I \wedge (ce \neq e)$.

Preuve C_I est équivalent à $(C_I \wedge (ce = e)) \vee (C_I \wedge (ce \neq e))$
Or, $C_I \wedge (ce = e)$ est inconsistant.
Donc, C_I est équivalent à $C_I \wedge (ce \neq e)$. ❑

Proposition 5.3 *Soient* $(ce, D_{ce}) \in B$ *et* $e \in D_{ce}$. *Si* $C_{I \cup \{(ce,e)\}} \models (ce_1, e_1), ...(ce_n, e_n)$ *et si* $C_{I \cup \{(ce,e)\}} \not\models \bot$ *alors* $\forall i \in \{1...n\} | ce_i \in B$, $C_{I \cup \{(ce_i, e_i)\}} \not\models \bot$

Preuve Supposons $C_{I \cup \{(ce,e)\}} \models (ce_1, e_1), ...(ce_n, e_n)$ et $C_{I \cup \{(ce,e)\}} \not\models \bot$.
Supposons de plus que $\exists (ce_k, e_k)$ avec $k \in \{1..n\}$ tel que $C_{I \cup \{(ce_k, e_k)\}} \models \bot$
Nous avons donc $C_I \wedge (ce_k = e_k) \wedge (ce = e) \models \bot$.
Mais, par hypothèse, on a $C_I \wedge (ce = e) \models (ce_k = e_k)$, ceci implique $C_I \wedge \neg(ce_k = e_k) \wedge (ce = e) \models \bot$.
Par conséquent, on a $C_I \wedge (ce = e) \models \bot$, ce qui est en contradiction avec les hypothèses. ❑

Remarque 5.1 L'expression $C_I \wedge (ce = e)$ est équivalente à $C_{I \cup \{(ce,e)\}}$

Dans le cas de la logique propositionnelle, le fait de ne pas propager les littéraux dont on sait qu'ils sont compatibles avec l'affectation courante suffit pour obtenir un algorithme efficace (voir section 2.1). Ceci n'est plus le cas lors de la génération de modèles finis. Les termes terminaux ont beaucoup de valeurs potentielles. C'est pour cela qu'il est nécessaire de restreindre le nombre de termes terminaux candidats à la réduction de leur domaine. Nous notons $T \subset B$ le sous–ensemble des termes terminaux candidats au filtrage. Il est clair que l'utilisation de l'heuristique LNH (nous

expliquons cette heuristique dans le chapitre 6) induit que seuls les termes terminaux dont les indices sont situés en dessous des mdn doivent être considérés (les termes terminaux candidats à l'heuristique LNH). Nous construisons ensuite l'ensemble T en suivant les règles :

- Si à un noeud donné la valeur mdn n'a pas augmenté alors T est l'ensemble des termes terminaux candidats à l'heuristique LNH qui ont le domaine le plus petit possible.
- Si la valeur mdn augmente, alors alors T est l'ensemble de tous les termes terminaux candidats à l'heuristique LNH.

Les résultats expérimentaux ont montré que, dans la majorité des cas, ce sous–ensemble de termes terminaux donne un bon compromis entre la réduction de l'arbre de recherche et celle du temps de calcul.

Heuristique UP

Les heuristiques de type «échec d'abord» font partie des meilleures stratégies des procédures énumératives. La méthode SEM, dans sa version originale, choisit comme prochain terme terminal celui de plus petit domaine et qui change le moins possible les valeurs mdn. La première condition réduit la largeur de l'arbre de recherche, la seconde préserve la symétrie triviale utilisée dans SEM. Nous notons H les deux conditions précédentes. A un noeud donné, il est possible que plusieurs termes terminaux vérifient la condition H. Mais certains s'avèrent meilleurs que d'autres en diminuant également la profondeur de l'arbre de recherche. Afin de prendre en compte ce dernier critère, nous utilisons les propagations faites pendant la phase d'élimination de valeurs. Nous comptabilisons le nombre d'affectations faites par chaque cellule avec toutes leurs valeurs possibles, nous notons ce nombre $w(ce)$ pour le terme terminal ce. Notre heuristique choisit finalement le terme terminal ce vérifiant la condition H et pour qui le nombre $w(ce)$ est le plus grand possible. Cette approche est similaire aux heuristiques UP [65, 43, 78] pour le problème SAT.

L'algorithme 5.1 décrit le processus d'élimination des valeurs ainsi que l'heuristique que nous venons de décrire. Il utilise les résultats des propositions 5.2 et 5.3. La marque `Candidat`$[ce, e]$ est vraie quand la valeur e du terme terminal ce est candidate à l'élimination. Le nombre Nb_p est égal au nombre de propagations faites durant l'appel à `Propa`. La combinaison de l'algorithme `Propa` et du générateur SEM crée une méthode que nous avons nommée VESEM (Value Elimination in SEM).

Les deux stratégies que nous venons de proposer essaient de réduire au plus la taille de domaines des termes terminaux à instancier. L'approche que nous introduisons maintenant consiste à propager le plus d'affectations possibles.

5.2 Dépendances des symboles fonctionnels

Nous nous proposons, dans cette section, de déterminer de nouvelles heuristiques prenant en compte la structure syntaxique de la théorie dont nous recherchons des modèles. Ce travail est en partie publié à IJCAR–2001 («International Join Conference on Automated Reasonning») [8].

Algorithme 5.1 Élimination de valeur et heuristique UP de choix des termes

Fonction Up_Heuristique(I, B, C) %%ensemble I,B,C de l'algorithme 4.1
Retourne : La prochaine cellule à instancier

Début
Pour Chaque $(ce, D) \in T$ **Faire**
    Pour Chaque $e \in D$ **Faire** Candidat[ce, e]=Vrai
Pour Chaque $(ce, D) \in T$ **Faire**
    $Nb_p = 0$
    Pour Chaque $e \in D$ tel que Candidat[ce, e]=Vrai **Faire**
        $I' = I$; $B' = B$; $C' = C$
        Ret=Propa$(I \cup (ce, e), B, C)$ %%Algorithme 4.2
        Pour Chaque (ce', e') propagé **Faire**
            Candidat[ce', e']=Faux
            $Nb_p = Nb_p + 1$
        Si Ret = Faux **Alors**
            $D = D - \{e\}$ %%Elimination de valeurs
            Si $|D| = 1$ **Alors** Retourner ce %% Propagation de ce
            Sinon
            $w(ce) = w(ce) + Nb_p$ %% Poids de ce
Retourner ce de plus petit domaine et maximisant w
Fin

Supposons l'existence d'un axiome $f(g(x), x) = x$ dans une théorie quelconque T. L'affectation d'un terme terminal associé à la fonction f propagera dans le meilleur des cas (voir la section précédente) des valeurs interdites sur des termes terminaux associés à la fonction g. D'un autre côté, l'affectation d'une cellule associée à la fonction g propagera une, voire plusieurs, affectations sur f. Il semble donc plus logique, afin d'arriver le plus rapidement possible à un modèle, de commencer par interpréter g avant f.

De même, il est évident que des fonctions n'apparaissant jamais dans des sous–termes sont uniquement déterminées quand les autres fonctions sont connues. Et l'affectation de termes terminaux issus de fonctions n'apparaissant jamais dans les plus hauts termes va propager énormément de «connaissance». Partant de ces quelques remarques, nous allons ordonner les fonctions. L'ordre est calculé avant le début de la phase de recherche de modèles à partir d'un graphe orienté et pondéré représentant les dépendances entre les fonctions. Cet ordre est ensuite utilisé pour mettre au point des heuristiques de choix des termes terminuax à instancier.

5.2.1 Graphe de dépendances

Intuitivement, l'axiome $f(g(x), x) = x$ génère une dépendance de la fonction g sur la fonction f. La connaissance de g implique une connaissance partielle sur f. Dans ce cas, le graphe de dépendances que nous construisons va contenir une arête $g \rightarrow f$. La définition 5.3 donne la notion de dépendance entre les symboles fonctionnels d'une théorie donnée.

Définition 5.3 *Soit $T = (\mathcal{S}, \mathcal{F}, A)$ une théorie. Le graphe de dépendances de la théorie T est le graphe orienté, pondéré $G = (\mathcal{X}, \mathcal{W})$ défini par :*

- *L'ensemble des sommets est l'ensemble des fonctions $\mathcal{F}$ hormis les constantes.*
- *L'arête $(f_i \rightarrow f_j)$ apparaît dans $\mathcal{W}$ si il existe un terme dans la formule A dans lequel f_i apparaît comme sous–terme de f_j.*
- *Le poids de l'arête $(f_i \rightarrow f_j)$ est égal au nombre de fois que f_i apparaît comme sous–terme de f_j dans la théorie.*

Définition 5.4 *On appelle fonction* strictement entrante*, une fonction dont le sommet associé dans le graphe de dépendances est de degré sortant nul. De même, on appelle fonction* strictement sortante*, une fonction dont le sommet associé dans le graphe de dépendances est de degré entrant nul.*

Le graphe de dépendances est crée sur le seul aspect syntaxique d'une théorie. Il donne l'ensemble des dépendances de la théorie et apporte des renseignements utiles sur les symboles fonctionnels. Nous connaissons alors les fonctions strictement sortantes qu'il faut générer prioritairement et les fonctions strictement entrantes que l'on doit le plus possible éviter d'instancier.

Il est évident qu'un graphe de dépendances n'apporte aucune information sur les théories ne comportant qu'un seul symbole fonctionnel. Dans tous les cas, et pour celui–ci en particulier, il serait intéressant d'essayer de créer un graphe sur l'aspect sémantique d'une théorie. Ce qui permettrait de connaitre le terme terminal le plus intéressant à affecter en premier.

Remarque 5.2 Les prédicats sont des fonctions strictement sortantes.

Exemple 5.2

Les axiomes suivant donnent la formulation d'un anneau unitaire dans lequel (g, a) est le groupe, m la loi multiplicative et ε l'élément neutre de m.

$a(\varepsilon, x) = x$	$a(x, \varepsilon) = x$
$a(g(x), x) = \varepsilon$	$a(x, g(x)) = \varepsilon$
$m(x, e) = x$	$m(e, x) = x$
$a(x, a(y, z)) = a(a(x, y), z)$	$m(x, m(y, z)) = m(m(x, y), z)$
$a(m(x, y), m(x, z)) = m(x, a(y, z))$	$a(m(x, z), m(y, z)) = m(a(x, y), z)$

Cet ensemble d'axiomes conduit au graphe de dépendances de la figure 5.3. La fonction g est une fonction strictement sortante et sera choisie en priorité pour définir les premiers termes terminaux à affecter.

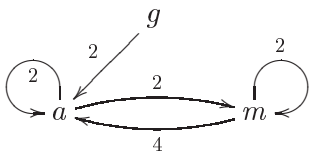


Figure 5.3. *Graphe de dépendances d'un anneau unitaire*

Remarque 5.3 Nous supposons, par la suite, que le graphe de dépendances ne contient qu'une seule composante fortement connexe. Dans le cas contraire, la théorie peut être transformée en plusieurs théories indépendantes, une par composante fortement connexe.

Nous introduisons maintenant une nouvelle définition de graphe de dépendances qui ne prend pas en compte les dépendances issues de fonctions strictement entrantes. Supposons une théorie contenant l'axiome $f(x) = g(h(x))$ où f est une fonction strictement entrante. Dans cet axiome, la connaissance de h ne propage pas de connaissance sur g. Nous supprimons donc cette dépendance du graphe introduit dans la définition 5.3. Après ce processus, une nouvelle fonction peut devenir strictement entrante, il faut donc le maintenir jusqu'à l'obtention d'un graphe stable. Le nouveau graphe obtenu est également nommé, lorsqu'il n'y a pas d'ambiguïté, graphe de dépendances.

5.2.2 Création d'un ordre

Nous allons maintenant établir un ordre de préférence entre les différents symboles fonctionnels d'une théorie en se servant du graphe de dépendances. Un ordre topologique sur les sommets d'un graphe quelconque n'est réalisable que sur des graphes ne contenant pas de circuits [30]. Comme les graphes de dépendances ne vérifient que très rarement cette propriété, nous déterminons donc un ordre «arbitraire». Il existe d'autres façons d'ordonner les symboles fonctionnels d'une théorie, mais celle que nous proposons a donné les meilleurs résultats expérimentaux.

Comme nous l'avons remarqué précédemment, les fonctions strictement sortantes sont choisies prioritairement et les fonctions strictement entrantes sont choisies en dernier. En fait, la génération des fonctions strictement entrantes est faite automatiquement par propagation de contraintes et ces fonctions très particulières ne sont presque jamais générées explicitement. Reste ensuite à déterminer un choix pour les constantes qui ont un rôle très particulier et pour les autres fonctions.

Les fonctions ayant des degrés entrants et des degrés sortants non nuls sont choisies en fonction du nombre de dépendances qu'elles ont envers les fonctions déjà ordonnées. L'arité des fonctions est

également un critère important, plus une fonction à une arité élevée, plus sa génération nécessite des affectations. C'est pour cela que lorsque deux fonctions différentes vérifient la même condition, nous choisissons prioritairement celle ayant l'arité la plus petite.

Les constantes ont un rôle totalement différent suivant le cas où elles apparaissent dans des équations ou dans des diséquations. Leur interprétation propage des affectations dans le premier cas, et supprime des valeurs de domaines dans le second. Les premières ont donc un rôle important pour la connaissance du modèle, les secondes sont d'un plus faible intérêt. De plus, l'affectation d'une constante supprime des valeurs interchangeables (au sens LNH). Nous avons donc choisi d'interpréter prioritairement les constantes apparaissant dans des équations. Celles qui interviennent dans des diséquations ont un ordre plus élevé que les fonctions dans lesquelles elles apparaissent.

L'algorithme 5.2 crée l'ordre de préférence pour une théorie donnée. Il est calculé une seule fois, avant le début de la recherche du modèle fini. Notons que certaines fonctions ne peuvent pas être départagées par l'algorithme `Ordre`. Les exemples 5.3 et 5.4 montrent l'ordre crée sur deux théories différentes.

Algorithme 5.2 Création d'un ordre sur les symboles fonctionnels

Procedure Ordre(G : Graphe de dépendances)
Début
Tant que il reste des fonctions non ordonnées
Faire
Sélectionner et ordonner une fonction f vérifiant la *première des conditions suivantes*
1 : f est une constante apparaissant dans une équation
2 : f est une fonction strictement sortante et d'arité minimale
3 : f est une fonction ayant le plus grand degré entrant venant de fonctions déjà ordonnées (et d'arité minimale)
4 : f est une constante apparaissant dans une inéquation
5 : f est une fonction strictement entrante
Fin

Exemple 5.3
Considérons la théorie RNG041–1 venant de la collection des problèmes TPTP [96]. Les axiomes et le graphe de dépendances de cette théorie sont donnés dans la figure 5.4.
En accord avec la stratégie de sélection défini dans l'algorithme 5.2, nous ordonnons d'abord les 2 constantes b et c qui apparaissent dans des équations. Ensuite, nous choisissons g et h qui sont deux fonctions unaires strictement sortantes. Le choix de a et de m est alors guidé par le poids de leurs arêtes venant de g et de h. L'ordre généré est donc : $b \leq c < g \leq h < m < a$. Les fonctions g et h ne peuvent être départagées, il en va de même pour les constantes b et c.

$a(x, a(y, z)) = a(a(x, y), z)$	$a(x, 0) = 0$
$m(x, m(y, z)) = m(m(x, y), z)$	$a(g(x), x) = x$
$m(x, a(y, z)) = a(m(x, y), m(x, z))$	$a(x, y) = a(y, x)$
$m(a(x, y), z) = a(m(x, z), m(y, z))$	$m(x, 1) = x$
$m(x, 0) = 0$	$m(1, x) = x$
$m(x, h(x)) = 1 \vee x = 0$	$m(0, x) = 0$
$m(h(x), x) = 1 \vee x = 0$	$b \neq 0$
$m(b, c) = 0$	$c \neq 0$

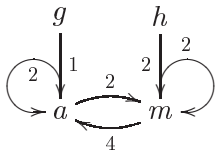
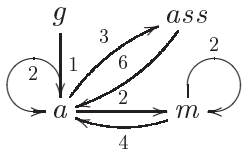


Figure 5.4. *Axiomes et graphe de dépendance du problème* RNG*041–1*

Exemple 5.4

Prenons maintenant le problème RNG025–8 de la collection TPTP [96]. Les axiomes et le graphe de dépendances sont donnés dans la figure 5.5 qui génère le graphe de dépendances ci-dessous :

$a(x, a(y, z)) = a(a(x, y), z)$	$a(x, 0) = 0$
$m(x, m(y, y)) = m(m(x, y), y)$	$a(g(x), x) = x$
$m(m(x, x), y) = m(x, m(x, y))$	$a(x, y) = a(y, x)$
$m(x, a(y, z)) = a(m(x, y), m(x, z))$	$m(x, 0) = 0$
$m(a(x, y), z) = a(m(x, z), m(y, z))$	$m(0, x) = 0$
$ass(w, x, a(y, z)) = a((ass(w, x, y), ass(w, x, z)$	
$ass(w, a(y, z), x) = a((ass(w, y, x), ass(w, z, x)$	
$ass(a(y, z), w, x) = a((ass(y, w, x), ass(z, w, x)$	
$com(x, y) = a(m(y, x), g(m(x, y)))$	
$a(ass(c, d, e), ass(c, e, d)) \neq 0$	



com

Figure 5.5. *Graphe de dépendances du problème* RNG*025–8*

La fonction g est strictement sortante, elle est choisie en premier. On choisit ensuite la fonction a. Puis, on a le choix entre les fonctions ass et m. On choisit ass qui a plus de dépendances venant de a (3 contre 2), puis m. On ordonne après les trois constantes c, d, e qui apparaîssent dans des inéquations, et on finit avec com qui est une fonction strictement entrante.

L'ordre généré est donc $g < a < ass < m < c \leq d \leq e < com$. ✍

Remarque 5.4 Le graphe de dépendances que nous construisons dépend uniquement de la syntaxe de la théorie et pas de la sémantique. Ainsi, une formulation différente d'une même théorie peut générer un graphe différent, et donc influer directement sur l'ordre crée.

5.2.3 Heuristiques

L'ordre ainsi établi sert ensuite à créer des heuristiques de choix utilisables dans n'importe quelle procédure énumérative générant des modèles finis, qu'elle utilise une représentation du premier ordre ou propositionnelle. Il peut également servir de base pour la création d'un nouvel ordre pour FMC [77, 76] qui jusqu'ici est déterminé par l'arité des fonctions. La méthode STORK [86] génère elle aussi les fonctions suivant un ordre, mais comme pour FMC, l'ordre proposé est uniquement dépendant de l'arité des fonctions.

Nous proposons deux heuristiques déterminées par plusieurs critères : l'ordre de préférence des symboles fonctionnels, la taille des domaines des termes terminaux et, si on l'utilise, l'optimisation de l'heuristique LNH.

L'heuristique GOT

La première heuristique que nous proposons consiste à générer entièrement une fonction avant de passer à la suivante, et de choisir les fonctions par ordre croissant. Le choix entre différents termes terminaux d'un même symbole fonctionnel est à définir. Une approche réaliste est déterminée par la taille des domaines des cellules, de façon à réduire la largeur de l'arbre de recherche, comme cela est fait dans SEM. Nous appelons cette heuristique GOT pour «Génération par Ordre Total».

Il est aisé de coupler cette heuristique avec l'heuristique LNH. Mais le nombre mdn va rapidement être maximal, et par conséquent les éléments vont rapidement devenir non interchangeables (après que la première fonction a été entièrement générée).

L'heuristique LNHO

C'est pour cela que nous proposons une deuxième stratégie qui prend plus en considération l'optimisation de l'heuristique LNH, laquelle choisit le terme terminal ayant les éléments les plus petits possibles. Le choix du terme terminal à instancier n'est plus déterminé uniquement par l'ordre du symbole fonctionnel. Nous essayons également d'incrémenter le moins rapidement les marques mdn. Cela donne naissance à une heuristique que nous nommons LNHO. Elle choisit comme prochain terme terminal $f(e_1, \ldots, e_k)$ celui qui vérifie les conditions suivantes listées par ordre décroissant de priorité.

1. $e_1, \ldots, e_k$ sont les plus petits possibles	Maximisation de LNH
2. f est d'ordre le plus petit possible	Maximisation de la stratégie «échec d'abord»
3. Le domaine de f est le plus petit possible	Minimisation de la largeur de l'arbre

5.3 Conclusion

Durant ce chapitre nous avons proposé diverses approches pour améliorer la propagation des contraintes des générateurs de modèles finis. Nous avons d'abord pris en considération les contraintes négatives. Les méthodes CTSEM et VESEM essaient ainsi de réduire le plus possible la taille des domaines des termes terminaux à instancier. Elles essaient donc de réduire la largeur de l'arbre de recherche.

Nous avons ensuite proposé les heuristiques GOT et LNHO. Elles essaient de propager le plus d'affectations pour réduire la hauteur de l'arbre de recherche. Par contre, et c'est l'objet du chapitre suivant, nous ne nous sommes pas intéressé aux traitements et aux suppressions des symétries qui. existent dans les théories multi–types de la logique du premier ordre.

Chapitre 6

Suppression de symétries

DE PAR LEUR STRUCTURE, les théories multi–types de la logique du premier ordre possèdent une grande quantité d'interprétations isomorphes. La recherche de modèles finis pour ces théories multi–types ne peut pas être efficace sans la considération et l'élimination de certaines interprétations symétriques. Ce chapitre y est entièrement consacré.

La première section introduit la notion de symétrie dans la logique du premier ordre. Nous rappelons ensuite diverses stratégies qui ont été utilisées dans le passé pour supprimer des isomorphismes. Nous présentons ensuite deux stratégies différentes mais complémentaires de traitement d'isomorphismes.

En premier lieu, nous proposons l'extension XLNH de l'heuristique LNH dans le sens où elle supprime plus de symétries triviales. Pour que l'heuristique XLNH soit utilisable, l'ensemble des symboles fonctionnels doit contenir une fonction unaire, bijective de préférence. Enfin, nous étudions un cadre plus général de recherche et d'utilisation des symétries. Cette approche dynamique peut être appliquée à n'importe quelle théorie mise sous forme clausale. Elle peut également être combinée aux heuristiques LNH et XLNH.

6.1 Symétries en logique du premier ordre

6.1.1 Notion de groupes et de groupes symétriques

Avant d'introduire la notion de symétries pour la logique du premier ordre, nous allons faire quelques rappels sur la théorie des groupes et des groupes symétriques.

Définition 6.1 *On appelle groupe tout couple $(G, +)$, où G est un ensemble et $+$ est une loi de composition interne, qui vérifie les conditions suivantes :*

- *La loi $+$ est associative*
- *La loi $+$ possède un élément neutre e unique.*

– *Tout élément a de G possède un inverse $\overline{a}$ pour la loi $+$: ($a + \overline{a} = e$).*

Un groupe est abélien si la loi $+$ est commutative. Si x est un élément de G et k un entier, alors x^k exprime k compositions sur x de la loi $+$, c'est à dire ($x + x + \ldots + x$). L'ordre d'un groupe est égal au cardinal de son ensemble. L'ordre d'un élément x est le plus petit entier n tel que $x^n = e$.

Définition 6.2 *On appelle sous–groupe de $(G, +)$ tout groupe de la forme $(H, +)$ où H est une partie stable*[1] *de G munie de la loi induite par celle de G.*

Proposition 6.1 *Soit E un ensemble. Si $Perm(E)$ est l'ensemble des permutations de E sur E alors $(Perm(E), \circ)$ est un groupe.*

L'élément neutre de ce groupe est la fonction identité définie de E dans E. Ainsi, la composition de deux permutations et l'inverse d'une permutation sont des permutations. Par abus de notation, on parlera souvent du groupe $Perm(E)$. Si E possède n éléments alors l'ordre de $Perm(E)$ est égal à $n!$.

Définition 6.3 (Cycle) *L'ensemble des remplacements $\sigma(a_1) = a_2$, $\sigma(a_2) = a_3 \ldots \sigma(a_n) = a_1$ forme un cycle de substitutions sur les individus $\{a_1, \ldots a_n\}$ de E. On note un tel cycle par le tuple $(a_1, a_2, \ldots, a_n)$, son ordre est n.*

Théorème 6.1 *Toute permutation d'un ensemble fini E se décompose en un produit de cycles dont les supports sont disjoints.*

Pour plus de résultats sur la théorie des groupes et des groupes symétriques, le lecteur peut se référer au livre de RAMIS et al. [80].

6.1.2 Les symétries

Intuitivement, une symétrie d'une théorie T de la logique du premier ordre est une permutation σ sur l'ensemble des types, des constantes ou des symboles fonctionnels, qui laisse inchangée la théorie. Ainsi si I est un modèle de T alors $\sigma(I)$ est également un modèle de T. Les symétries définissent des classes d'interprétations isomorphes. Il est inutile de considérer l'ensemble de toutes les interprétations d'une classe lors de la recherche d'un modèle fini, mais seulement un représentant. Nous nous limitons dans ce mémoire aux permutations opérant sur l'ensemble des types d'une théorie.

Définition 6.4 *Une permutation σ sur l'ensemble des types $S = \{s_1, \ldots, s_n\}$ est définie par le tuple de permutations $\sigma = (\sigma_1, \ldots, \sigma_n)$, où chaque $\sigma_i \in Perm(s_i)$.*

[1] $H \subset G$ est une partie stable de G pour la loi $+$, si pour tout $a, b \in H$ on a $a + b \in H$

Définition 6.5 *Si le symbole fonctionnel $f \in F$ est spécifié par $f : s_{i_1} \times \ldots \times s_{i_k} \mapsto s_j$ et que $f(e_{i_1}, \ldots, e_{i_k})$ est un terme terminal alors $\sigma(f(e_{i_1}, \ldots, e_{i_k})) = f(\sigma_{i_1}(e_{i_1}), \ldots \sigma_{i_k}(e_{i_k}))$.*

Une permutation σ peut donc se généraliser facilement aux termes, à la théorie et aux interprétations.

Remarque 6.1 Si σ est une permutation définie sur l'ensemble des types S de la théorie $T = (S, F, A)$, alors les ensembles S et F sont invariants par σ et $\sigma(T) = (S, F, \sigma(A))$.

Nous montrons maintenant un résultat important concernant les symétries en logique du premier ordre.

Théorème 6.2 *Soient $T = (\mathcal{S}, \mathcal{F}, A)$ une théorie et σ une permutation sur les domaines de $\mathcal{S}$. Si I est un modèle de T alors $\sigma(I)$ est un modèle de $\sigma(T)$.*

Preuve Soit σ une permutation de $Perm(\mathcal{S})$ et I un modèle de A. Nous montrons de manière inductive que $\sigma(I)$ est un modèle de $\sigma(A)$ (pour les connecteurs «$\vee$», «$\neg$» et le quantificateur «$\exists$»).

- Si A est de la forme $t = s$ alors $I(s) = I(t)$ donc $(\sigma(I))(s) = (\sigma(I))(t)$ et $\sigma(I)$ est un modèle de $\sigma(A)$.
- Si $A = \neg B$ on a alors $I(B) = Faux$. Supposons que $\sigma(I) \models \sigma(B)$ alors $\sigma^{-1}(\sigma(I)) \models \sigma^{-1}(\sigma(B))$ et donc $I \models B$ ce qui est contraire à l'hypothèse.
- Si $A = A_1 \vee A_2$. On a soit $I \models A_1$ soit $I \models A_2$. Par hypothèse on a donc, soit $\sigma(I) \models \sigma(A_1)$ soit $\sigma(I) \models \sigma(A_2)$ et donc $\sigma(I) \models \sigma(A_1 \vee A_2)$.
- Si $A = \exists x B$ alors A est équivalent à $\bigvee_{e \in Dom(x)} B(x \to e)$. or $I \models A$, d'après le cas démontré ci–dessus, on a donc $\sigma(I) \models \sigma(\bigvee_{e \in Dom(x)} B(x \to e))$ et donc $\sigma(I) \models \sigma(\exists x B)$. ↵

Définition 6.6 *Une permutation $\sigma \in Perm(\mathcal{S})$ d'une théorie T est une symétrie de T si $\sigma(T) = T$.*

Plus précisement, une symétrie d'une théorie $T = (S, F, A)$ est une permutation $\sigma \in Perm(S)$ telle que $\sigma(A) = A$.

Proposition 6.2 *L'ensemble des symétries de T, noté $Sym(T)$ forme un sous–groupe du groupe $(Perm(T), \circ)$.*

Preuve Nous devons montrer que :

- $Id \in Sym(T)$. Ceci est évident puisque $Id(T) = T$.
- $Sym(T)$ est stable par $\circ$.
  Soient σ_1 et σ_2 deux symétries de T. On a donc $\sigma_1(\sigma_2(T)) = \sigma_1(T) = T$. La seconde condition est donc vérifiée.

- si $\sigma \in Sym(T)$ alors $\sigma^{-1} \in Sym(T)$
  Soit $\sigma \in Sym(T)$. On a $\sigma^{-1}(\sigma(T)) = Id(T) = T$. De plus $\sigma^{-1}(\sigma(T)) = \sigma^{-1}(T)$, on a donc $\sigma^{-1}(T) = T$ et donc $\sigma^{-1} \in Sym(T)$. ⧉

Définition 6.7 *Soit T une théorie. Deux interprétations I et J sont symétriques si il existe une symétrie σ de T telle que $\sigma(I) = J$.*

Exemple 6.1
Reprenons la théorie de l'exemple 4.5.

- $\forall x, h(x, x) = x$
- $\forall x \forall y, h(h(x, y), x) = y$

Voici deux modèles I_h^1 et I_h^2 de cette théorie.

I_h^1	0	1	2	3
0	0	2	3	1
1	3	1	2	0
2	1	3	2	0
3	2	1	0	3

I_h^2	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

Ils sont liés par la symétrie $\sigma = (0) \circ (1) \circ (2, 3)$. En effet, on a $I_h^1 = \sigma(I_h^2)$. ✍

La section suivante rappelle différentes approches qui ont été utilisées, dans divers domaines, pour supprimer des symétries.

6.2 Différentes approches de suppression des symétries

Les symétries ont beaucoup été étudiées ces dernières années. Nous nous proposons de rappeler quelques stratégies que nous avons dissociées en différentes classes : la suppression de valeurs interchangeables, la suppression avec une recherche dynamique et enfin l'ajout de contraintes au problème principal.

6.2.1 Heuristique LNH et valeurs interchangeables

Lors de l'introduction de la méthode SEM à la section 4.2.3, nous avons brièvement parlé de la façon dont cette méthode supprime certaines symétries triviales. Nous nous y attardons un peu plus maintenant.

Les symétries triviales résultent du fait que toutes les variables des différents axiomes représentant le problème sont quantifiées universellement. Les individus de leurs domaines sont tous interchangeables et donc symétriques. De manière informelle, un ensemble de valeurs est interchangeable si la permutation de ces éléments ne change pas la théorie. De ce fait, si à un moment donné de la résolution les premiers individus $\{0 \dots mdn\}$ du domaine $D_n = \{0, 1, \dots, mdn, \dots, n-1\}$ ont été

utilisés dans la construction du modèle, alors les individus de la partie $\{mdn+1 \ldots n-1\}$ restent interchangeables. L'heuristique LNH exploite au maximum cette propriété en choisissant en priorité les individus du sous–ensemble $\{0, \ldots, mdn\}$ pour choisir et instancier le prochain terme terminal. Ceci permet de garder plus longtemps la symétrie entre les individus de la partie $\{mdn+1, \ldots, n-1\}$.

Le nombre mdn, «maximum designated number», est la frontière qui sépare les individus de la partie symétrique de ceux de la partie non symétrique du domaine D_n. Bien entendu, il existe un nombre mdn pour chaque type $s \in S$ de la théorie $T = (S, F, A)$. Chaque fois qu'un nouveau terme terminal $f(e_1, \ldots, e_k)$ est choisi, on met à jour les mdn des types Sort(e_1), Sort(e_2), etc...Les valeurs potentielles de la cellule $f(e_1, \ldots, e_k)$ seront alors comprises dans le sous–ensemble $\{0, \ldots, mdn+1\}$ des individus de $Sort(ce)$. En effet, l'individu $mdn+1$ représente toute la partie $\{mdn+1, \ldots, n-1\}$ d'individus symétriques.

D'autres articles ont traité des valeurs interchangeables. Dans [45], Eugene FREUDER élimine les valeurs interchangeables d'un problème de satisfaction de contraintes. Il définit plusieurs niveaux d'interchangeabilité.

Pedro MESEGUER et Carme TORRAS proposent également une heuristique de choix utilisant les symétries présentes dans un CSP [74]. Un pré–traitement est appliqué afin de connaître les isomorphismes du problème. Ensuite, l'heuristique choisit comme variable à instancier celle qui supprime le plus de symétries possibles. De cette manière, les symétries sont «cassées» le plus haut possible dans l'arbre de recherche ce qui conduit à la diminution de sa taille.

6.2.2 Recherche dynamique

Nous avons souligné, dans la section 4.2.5, que la méthode FMC supprimait dynamiquement certaines symétries pendant la recherche. Des approches dynamiques ont également été proposées dans d'autres domaines comme la logique propositionelle et les CSP. Dans [14, 16], Belaid BENHAMOU et Lakhdar SAIS détectent, pour les problèmes SAT, des permutations sur l'ensemble des littéraux qui laissent inchangé l'ensemble de clauses courant. Il mettent en pratique un principe de résolution augmenté par les symétries proposé par KRISHNAMURTHY [62]. Leur algorithme s'est avéré performant sur des problèmes difficiles à résoudre jusqu'alors par des méthodes énumératives, comme le problème des pigeons ou celui de RAMSEY [56].

Cynthia BROWN et al. proposent dans [20] un algorithme de type backtrack qui exploite dynamiquement des symétries et calcule à chaque étape le nouveau sous–groupe des permutations. Mais leur approche de détection est sensiblement différente de celle de Nicolas PELTIER ou de Belaid BENHAMOU car leur algorithme fonctionne uniquement à partir d'un groupe de permutations déterminé avant le début de la recherche. Ils ne traitent donc pas des symétries arbitraires. Une méthode similaire à celle de Cynthia BROWN a été proposée sous le nom de SCORE(FD/B) par Jacqueline CHABRIER et al. [23]. Elle détecte avant le début de la recherche des symétries puis les exploite

dynamiquement.

6.2.3 Rajout de contraintes

La stratégie utilisée par MGT-P consiste à rajouter des contraintes qui codent les symétries dans la formalisation du problème. Cette approche statique a également été utilisée pour la logique propositionnelle [33] et pour les problèmes de satisfaction de contraintes [58].

L'intégration de nouvelles contraintes va demander un pré–traitement qui consiste à rechercher les isomorphismes globaux du problème posé. Cette recherche est souvent faite de la manière suivante :

- Formulation du problème initial en un graphe
- Recherche des isomorphismes du graphe en utilisant par exemple le programme NAUTY de Brendan MC KAY[73].

Rolf BACKOFEN et Sebastian WILL proposent de supprimer dynamiquement les symétries en rajoutant, en cours de recherche, des contraintes [9]. Cette stratégie a été également utilisée dans [50] par Ian GENT et Barbara SMITH.

Les partisans de cette stratégie qui consiste à rajouter des contraintes s'appuient sur les arguments suivants :

- Portabilité : l'arrivée d'une nouvelle méthode de résolution ne demande pas une nouvelle implantation de l'algorithme.
- Possibilité d'utilisation avec des méthodes de recherche locale.

Le grossissement de la théorie de départ est l'inconvénient majeur de cette approche. De plus, seules les symétries «globales» peuvent être traitées.

6.2.4 Discussion

La classification de suppression des symétries que nous proposons n'est, bien–sûr, pas unique. Dans [22], Jacqueline CHABRIER et Eric NESPOULOS partitionnent différemment les approches des symétries. Ils utilisent les classes suivantes :

- Les symétries ne sont pas recherchées par la méthode de résolution mais sont données à priori, citons par exemple [91, 20]
- certaines symétries sont recherchées avant la résolution [23, 50].
- les symétries sont détéctées pendant la résolution [16, 77]

D'autres critères sont mis en valeur lors de cette classification. Les méthodes de la première classe, sont les plus faciles à utiliser au niveau calculatoire . Celles de la dernière sont les plus difficiles.

Les différentes stratégies de suppression des isomorphismes ont des qualités et des défauts. Le rajout de contraintes (utilisé par MGT-P) offre une utilisation avec tous les algorithmes existants et à

venir. Par contre, les clauses introduites grossissent la base de clauses et influent sur la consommation mémoire. De plus, il ne semble pas évident, excepté sur des problèmes simples comme le problème des reines ou celui des quasigroupes , de déterminer à l'avance les symétries. C'est pour celà que certaines méthodes (FMC) optent pour une recherche dynamique qui est plus générale. Par contre, cette stratégie implique un surcout en temps lors de la détection des isomorphismes. La détection de symétries «triviales» le long de la recherche comme le fait l'heuristique LNH semble être une alternative intéressante. Dans la section suivante, nous présentons une amélioration de cette heuristique qui supprime plus de symétries.

6.3 XLNH : Une extension de l'heuristique LNH

L'efficacité de LNH vient de la détection et de la suppression de certaines symétries sans surcoût de temps de calcul. Seul le maintien à jour des nombre mdn est nécessaire. L'idée principale de l'heuristique XLNH est d'exploiter l'existence d'une fonction unaire dans la théorie. Nous focalisons la recherche sur cette fonction en générant de façon constructive ses seules interprétations «canoniques» (non symétriques). Après avoir totalement interprété cette fonction, certains des nombreux isomorphismes encore présents sont récupérés de manière *statique*, comme l'heurstique LNH, grâce à la structure de cette fonction unaire. Le choix de la fonction unaire en question est important et sera guidé par l'ordre de préférence que nous avons défini dans la section 5.2. Nous avons appelé XLNH «eXtended LNH» cette méthode.

Ces travaux ont été présentés aux «Journées nationales de la résolution pratique des problèmes NP–complets» (JNPC2000) [7] et publiés à IJCAR01 «International Join Conference on Automated Reasoning» [8].

Dans un premier temps nous donnons quelques définitions et remarques concernant les fonctions unaires, puis nous expliquons cette méthode en nous restreignant à une fonction unaire bijective. Nous généraliserons ensuite cette approche à toutes les fonctions unaires. Sans perte de généralité, nous supposons que nous travaillons dans une théorie ne comportant qu'un seul type D_n.

6.3.1 Les fonctions unaires

Définition 6.8 *Soient un domaine $D_n = \{0, \ldots, n-1\}$ et une fonction $g : D_n \mapsto D_n$. Soit $D' \subseteq D_n$ un sous–ensemble de D_n. On définit $g(D')$ par l'ensemble $\{g(e)$ tel que $e \in D'\}$. De même, on définit $g(H)^{-1}$ par l'ensemble $\{e \in D_n$ tel que $g(e) \in H\}$*

Définition 6.9 (circuit) *Soient un domaine $D_n = \{0, \ldots, n-1\}$ et une fonction $g : D_n \mapsto D_n$. Un sous–ensemble C de D_n minimal pour l'inclusion tel que $g(C) = C$ est appelé un circuit de g. La taille d'un circuit est égale au cardinal de l'ensemble C (notation $|C|$).*

Notation 6.1 *Un élément e apparait dans au plus un circuit. On note ce circuit C_e.*

Définition 6.10 (restriction bijective) *Soient un domaine $D_n = \{0, \ldots, n-1\}$ et une fonction $g : D_n \mapsto D_n$. Le sous–ensemble $R_B \subseteq D_n$ qui est maximal pour l'inclusion et qui vérifie $g(R_B) = R_B = g(R_B)^{-1}$ est la restriction bijective de g.*

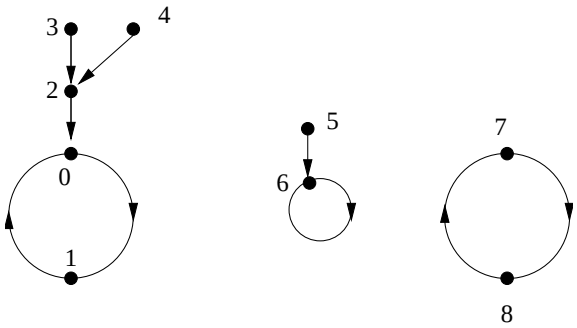
Les cycles tels que nous les avons définis dans la section 6.1.1 sont donc des circuits particuliers. La restriction bijective est égale à l'union de tous les cycles et non pas à l'union de tous les circuits. Dans le cas où g est une bijection, on a $R_B = D_n$. Ces différentes notions sont aisément visualisables sur le graphe orienté associé à la fonction. Les sommets d'un tel graphe sont étiquetés par les éléments de D_n. Si $g(i) = j$ alors l'arc $i \to j$ appartient à l'ensemble des arêtes.

Exemple 6.2

Soient le domaine D_9 et la fonction $g : D_9 \mapsto D_9$ suivante :

g	0	1	2	3	4	5	6	7	8
	1	0	0	2	2	6	6	8	7

Le graphe de g est



Dans cette interprétation nous avons 3 circuits. $(0, 1)$, (6) et $(7, 8)$. Le dernier est un cycle. La restriction bijective est $\{7, 8\}$.

6.3.2 Génération d'une fonction unaire bijective

Dans un premier temps, nous nous limitons à l'utilisation d'une fonction bijective. La première partie de la méthode consiste à interpréter entièrement une fonction bijective en considérant uniquement ses interprétations non isomorphes.

Le graphe associé à l'interprétation d'une fonction unaire bijective est composé uniquement de cycles. Afin de générer efficacement les interprétations non isomorphes nous allons ordonner les cycles de manière croissante suivant leur taille.

Proposition 6.3 *Soit g une bijection de $D_n \mapsto D_n$ et I_g une interprétation de g. Il existe une permutation $\sigma \in Perm(D_n)$ telle que $\sigma(I_g)$ vérifie :*

– *Pour chaque cycle* $c = (e_1, e_2, \ldots, e_k)$ *de* g*, on a* $\forall j < k$, $g(e_j) = e_{j+1}$ *et* $g(e_k) = e_1$.
– *Si* c_1 *et* c_2 *sont deux cycles consécutifs alors* $|c_1| \leq |c_2|$.

Preuve On démontre cette proposition en construisant la permutation σ. La fonction g étant une bijection, l'interprétation associée est une composition de cycles. On a donc $g = g_1 \circ g_2 \circ \ldots \circ g_k$. Les cycles sont disjoints, on peut supposer qu'ils sont ordonnés suivant leur taille croissante, on a donc $|g_1| \leq |g_2| \ldots |g_k|$.
On construit alors σ à partir de chaque cycle g_j.
On a $g_1 = (e_1, \ldots, e_l)$. On pose alors $\sigma(e_1) = 0$, ..., $\sigma(e_l) = l - 1$.
On a $g_2 = (e_n, \ldots, e_m)$. On pose alors $\sigma(e_n) = l$, ..., $\sigma(e_m) = l + m - n$.
En continuant de cette manière sur tous les cycles de g, on construit σ qui est bien une permutation puisque chaque individu e_i apparaît dans un et un seul cycle. De plus, il est évident que $\sigma(I_g)$ vérifie les conditions de la proposition. ▭

Définition 6.11 *Une interprétation* I_g *satisfaisant les conditions de la proposition 6.3 est dite canonique.*

Exemple 6.3
L'interprétation suivante de I_g est canonique. Elle possède 2 cycles de taille 1, un de taille 2 et un de taille 3. Elle est symétrique à l'interprétation non canonique de I'_g. Les deux interprétations sont liées par la permutation $\sigma = (0, 4) \circ (1, 2) \circ (3, 6, 5)$ et on a $I_g = \sigma(I'_g)$.

I_g	0	1	2	3	4	5	6
	0	1	3	2	5	6	4

I'_g	0	1	2	3	4	5	6
	6	5	2	0	4	1	3

La proposition 6.4 montre que le nombre d'interprétations canoniques existant sur un domaine de taille n est en relation étroite avec la notion de partition d'entiers [28] que nous définissons ci–dessous.

Définition 6.12 *Soit* n *un entier, on appelle partition de* n *une représentation de* n *en une somme d'entiers* ≥ 1*, l'ordre des sommants n'étant pas pris en considération.*

On note $p(n)$ le nombre de partitions de l'entier n. Par exemple l'entier 4 possède $p(4) = 5$ partitions d'entiers : $4 = 4 + 0 = 3 + 1 = 2 + 2 = 2 + 1 + 1 = 1 + 1 + 1 + 1$.

Proposition 6.4 *Soit* g *une fonction bijective sur un domaine* D_n*. Il existe exactement* $p(n)$ *interprétations canoniques de* g.

Preuve Soit une interprétation canonique de g. La fonction g est bijective, son interprétation est donc composée uniquement de cycles. Il existe $k \leq n$ tel que $g = g_1 \circ g_2 \circ \ldots \circ g_k$ avec $|g_1| \leq |g_2| \leq \cdots \leq |g_k|$. Comme tous les g_i sont disjoints, on a $|g_1| + |g_2| + \cdots + |g_k| = |g| = n$.
Le nombre d'interprétations canoniques est donc égal au nombre de partitions de n. Ce nombre vaut $p(n)$. ⧉

Le théorème 6.3 donne l'équation de récurrence que vérifient les nombres $p(n)$ [28]. La table 6.1 donne les valeurs de $p(n)$ pour les 11 premiers entiers.

Théorème 6.3 *Soit n un entier. Le nombre $p(n)$ vérifie la récurrence :*

$$\begin{aligned} p(n) &= \sum_{k>0} (-1)^{k-1} (p(n - k \times \frac{3k-1}{2}) + p(n - k \times \frac{3k+1}{2}) \\ &= p(n-1) + p(n-2) - (p(n-5) + p(n-7)) + p(n-12) + p(n-15) - \cdots \end{aligned}$$

n	0	1	2	3	4	5	6	7	8	9	10
$p(n)$	1	1	2	3	5	7	11	15	22	30	42

Tableau 6.1. *Nombre de partitions de n*

Le résultat de la proposition 6.3 est utilisé pour commencer par générer les seules interprétations canoniques d'une fonction bijective tout en conservant le caractère complet du générateur (aux isomorphismes près). Sachant que pour un domaine de taille n il existe $n!$ interprétations d'une fonction bijective et $p(n)$ interprétations canoniques, la réduction de l'espace de recherche est déjà conséquente.

Le résultat de la proposition 6.5 est utilisé en association d'une fonction unaire pour supprimer statiquement certaines interprétations symétriques encore présentes.

Proposition 6.5 *Soient T une théorie, g un symbole fonctionnel unaire bijectif de T et I un modèle de T. Soient h un autre symbole fonctionnel de T et $h(e_1, \ldots, e_k)$ un terme terminal interprété à v. Si v n'appartient à aucun cycle de g où apparaissent $e_1, \ldots, e_k$, alors pour tout w vérifiant :*

- *$w \neq v$*
- *w n'appartient à aucun cycle de g où apparaissent $e_1, \ldots, e_k$*
- *les tailles des cycles de g dans lesquels apparaissent v et w sont égales*

il existe une symétrie transformant I en un modèle I' de T dans lequel $h(e_1, \ldots, e_k) = w$.

Preuve Soient C_v et C_w les cycles de g, non vides, de v et w. Il y a deux cas possibles :

- $C_v \neq C_w$: Soit $\sigma : D_n \mapsto D_n$, une permutation égale à l'identité partout, excepté en C_v et C_w, et qui vérifie[2] $\sigma(g^i(v)) = g^i(w)$ et $\sigma(g^i(w)) = g^i(v)$ pour tout $i \in \{0, \ldots, |C_v| - 1\}$. On intervertit donc les éléments des différents cycles.
- $C_v = C_w$: Soit $\sigma : D_n \mapsto D_n$, une permutation égale a l'identité partout, sauf sur C_v, et qui vérifie $\sigma(g^i(v)) = g^i(w)$ pour tout $i \in \{0, \ldots, |C_v| - 1\}$. On décale donc les éléments du cycle.

la permutation ainsi définie vérifie $\sigma(h(e_1, \ldots, e_k) = v) = (h(\sigma(e_1), \ldots, \sigma(e_k)) = \sigma(v)) = (h(e_1, \ldots, e_k) = w)$. Elle laisse également inchangée l'interprétation de g. La théorie étant mise sous forme clausale, les axiomes sont universellement quantifiés. Comme aucun individu n'apparaît dans les clauses, $\sigma(I)$ est donc un modèle de T. ⧉

Exemple 6.4

Soit I_g l'interprétation de la fonction bijective g.

I_g	0	1	2	3	4
	0	1	2	4	3

Les interprétations $I_1 = I_g \cup \{h(1,2) = 3\}$ et $I_2 = I_g \cup \{h(1,2) = 4\}$ sont isomorphes. L'interprétation partielle I_1 constitue un début de modèle si et seulement si l'interprétation partielle I_2 constitue un début de modèle. ✍

6.3.3 La procédure d'énumération selon l'heuristique XLNH

Nous générons les modèles finis d'une théorie contenant une fonction unaire bijective g en utilisant les résultats des propositions 6.3 et 6.5. La génération se fait donc en deux étapes. Nous commençons par générer une interprétation canonique de g. Ensuite, nous utilisons une procédure énumérative pour continuer la recherche. Compte tenu des cycles de l'interprétation de la fonction g et de la proposition 6.5, nous mettons en avant de nouveaux individus interchangeables que nous supprimons pour élaguer l'arbre de recherche. Nous avons nommé «heuristique XLNH» l'ensemble de ce processus que nous explicitons maintenant.

Première étape

Nous commençons par générer une interprétation canonique de la fonction bijective. La construction se fait de la façon suivante (proposition 6.3) :

– Pour chaque élément e du domaine D_n, soit $g(e) = e + 1$, soit e est la fin d'un cycle et dans ce cas $g(e) = e - |C_e| - 1$
– Les cycles de g sont ordonnés suivant leur taille croissante.

[2] On note $g^i(v)$ l'application i fois de g sur v soit $g^i(v) = g(g(\ldots(g(v)\ldots))$

Nous n'avons pas décrit l'algorithme de construction d'une interprétation canonique d'une fonction bijective. Celui–ci est une version simplifiée de l'algorithme 6.2 qui génère les interprétations canoniques de fonctions unaires quelconques.

Deuxième étape

Pour chaque interprétation canonique I_g ainsi générée, l'algorithme complète l'interprétation par un processus classique d'énumération. L'algorithme 6.1 donne le schéma d'implantation et d'intégration de notre stratégie dans un générateur de modèles finis basé sur l'énumération. Cet algorithme nécessite l'introduction de la définition suivante :

Définition 6.13 *Un élément $e \in D_n$ a été* touché *si au moins l'une des deux conditions suivantes est vraie :*

- *e a été affecté comme valeur d'une cellule.*
- *il existe une cellule $h(e_1, \ldots, e, \ldots, e_k)$ instanciée ou en cours d'instanciation.*

On dira qu'un cycle a été touché si l'un de ses éléments l'a été.

Algorithme 6.1 La procédure d'énumération

Fonction Recherche_XLNH($I = I_g \cup I'$: Interprétation ; A : Axiomes) : Booléen
Retourne : Vrai si A est satisfait par I, Faux sinon.
Début
Pour tous les $a \in I$ non propagés **faire** Propager(a,A,I)
Si I contient des affectations incompatibles **Alors Retourner** Faux
Si A est vide **Alors retourner** Vrai
Choisir un terme terminal $t(e_1, \ldots, e_k)$ tel que max((e_j), $1 \leq j \leq k$) soit le plus petit possible
Pour chaque e_j **Faire** Mettre à jour $mdn_{|e_j|}$)
Pour chaque $e \in D_n$ tel que $e \leq mdn_{|C_e|} + 1$ **Faire**
    Si Recherche_XLNH($I \cup \{t(e_1, \ldots, e_k) = e\}$,$A$) **Alors Retourner** Vrai
Retourner Faux
Fin

L'algorithme `Recherche_XLNH` utilise une marque mdn_s pour chaque taille de cycle s, elle représente le maximum des index de fin des cycles de taille s touchés. Les cycles de taille égale sont consécutifs. Cet agencement permet de détecter les symétries de manière statique comme le fait l'heuristique LNH. Par la proposition 6.5 nous savons que lorsque on choisit une valeur e pour une cellule $t(e_1, \ldots, e_k)$, seules les valeurs plus petites que $mdn_{|C_e|} + 1$ ont besoin d'être testées.

L'heuristique LNH utilise une seule marque mdn. Elle sélectionne les termes terminaux de façon à incrémenter le moins rapidement possible cette valeur. L'heuristique XLNH que nous venons de

décrire sélectionne les termes terminaux de façon à incrémenter le moins rapidement possible les mdn correspondant aux différentes tailles de cycles.

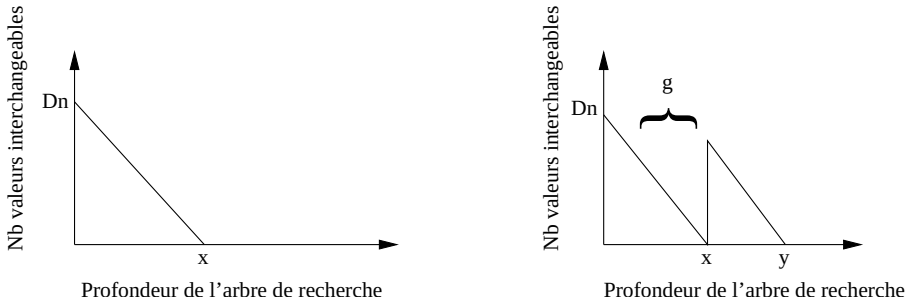


Figure 6.1. *Comportement de* LNH *et de* XLNH

La figure 6.1 illustre les différences entre les heuristiques LNH et XLNH. L'heuristique LNH exploite des individus interchangeables jusqu'à une certaine profondeur x de l'arbre de recherche (diagramme de gauche de la figure 6.1). A partir de cette limite, tous les individus ont été utilisés et l'heuristique LNH n'exploite plus aucune symétrie.

L'heuristique XLNH conserve beaucoup plus cette notion d'isomorphisme en repoussant le plus loin possible la profondeur où les individus ne sont plus interchangeables. Après la génération d'une interprétation canonique de g, l'heuristique XLNH exploite, à l'aide des cycles contenus dans g, de nouvelles valeurs interchangeables jusqu'à une profondeur y de l'arbre de recherche (diagramme de droite de la figure 6.1). La génération des interprétations canoniques conserve certains individus interchangeables, nous les exploitons pour couper l'arbre de recherche. Comme LNH, l'implantation de cette heuristique est entièrement statique et ne requiert donc aucun surcout de temps.

Remarque 6.2 Nous utilisons la méthode XLNH avec la stratégie de l'ordre des fonctions que nous avons détaillée dans la section 5.2. Lors de la création de cet ordre, une condition supplémentaire est rajoutée : nous préfèrons choisir une fonction unaire strictement sortante qui soit bijective à une strictement sortante qui ne le soit pas. L'exemple 5.3 voit donc une différence entre le choix de g (que l'on préférera) et le choix de h que l'algorithme 5.2 de création d'ordre sur les symboles fonctionnels ne permet pas de départager. Ensuite, la recherche commence en générant les interprétations canoniques de g.

Exemple 6.5

La figure 6.2 illustre la génération de tous les modèles non isomorphes au sens XLNH des groupes abéliens d'ordre 5. Une définition des groupes abéliens est donnée en 6.1.1. La génération est montrée dans la figure 6.2. Pour un domaine $D_5 = \{0, \ldots, 4\}$, seules trois interprétations canoniques de la fonction unaire g sont entièrement générées (I_g^1, I_g^2, I_g^3). Ces dernières sont représentées dans

les tableaux de la figure 6.2. Les arbres de la figure représentent l'extension de chacune des interprétations par l'algorithme `Recherche_XLNH` et les parenthèses désignent des valeurs supprimées par l'utilisation de l'heuristique XLNH. Par exemple, pour la deuxième interprétation canonique, les interprétations $h(1,2) = 3$ et $h(1,2) = 4$ sont symétriques (voir exemple 6.4). Les branches dues à la propagation des contraintes ne sont pas représentées. Le symbole $\bot$ représente l'inconsistance et le symbole $\top$ représente l'obtention d'un modèle.

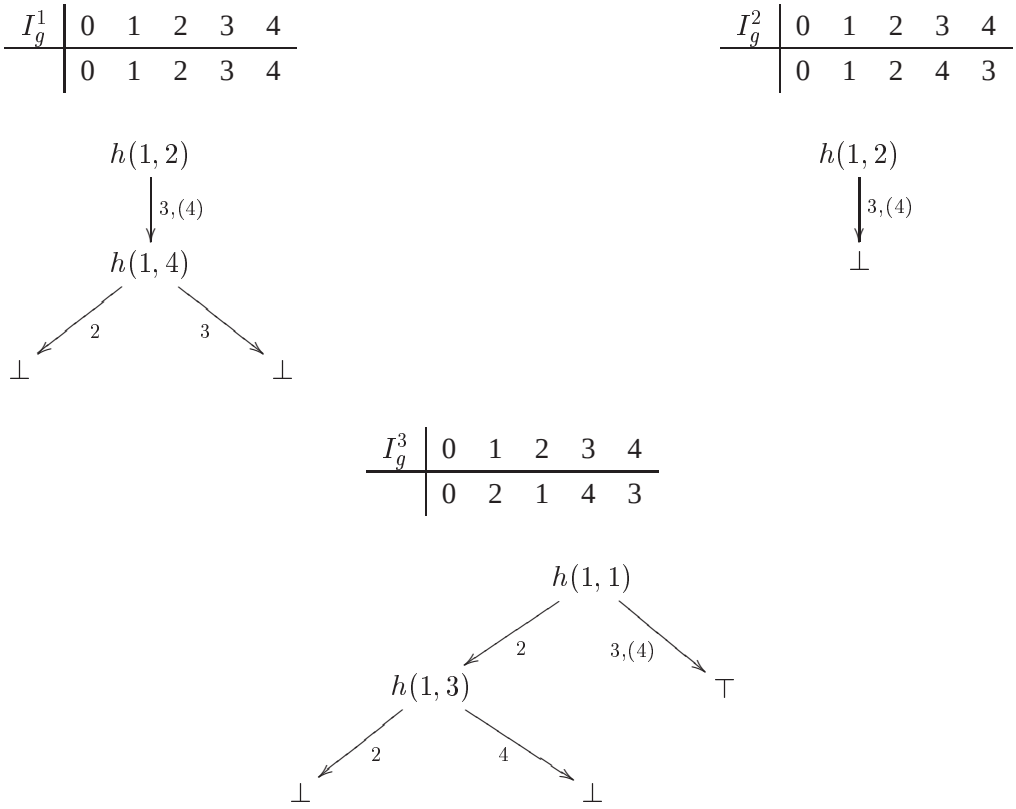


Figure 6.2. *L'arbre de recherche avec* XLNH *pour les groupes abéliens d'ordre 5*

On observe que la première interprétation g générée est l'identité. Tous les cycles ont donc la même taille égale à 1. Dans ce cas, on continue la recherche comme si on démarrait avec l'heuristique LNH et un mdn valant 0, alors qu'une certaine profondeur de l'arbre de recherche est déjà atteinte. Dans cet exemple, on génère un seul modèle de base sur les 6 existant. Les autres modèles sont symétriques à ce dernier. ✍

Dans la section suivante, nous généralisons la notion d'interprétations canoniques et l'heuristique XLNH à des fonctions unaires quelconques.

6.3.4 Extension à toutes les fonctions unaires

Même si des fonctions bijectives sont souvent présentes dans les théories (en particulier sur les structures mathématiques comme les groupes, les anneaux, etc...), il est intéressant d'utiliser un procédé similaire sur les fonctions unaires quelconques. Le premier intérêt réside dans le peu d'interprétations non isomorphes que contient une fonction unaire. Par exemple, il en existe 47 pour un domaine de taille 5. L'heuristique LNH génère 600 interprétations sur les 5^5 soit 3125 possibles. La génération d'interprétations canoniques pour une fonction unaire quelconque va éliminer, comme dans le cas de fonction bijective, un certain nombre d'interprétations isomorphes.

Nous généralisons la notion d'interprétations canoniques aux interprétations satisfaisant les critères de la proposition 6.6 qui concerne les fonctions unaires quelconques.

Proposition 6.6 *Soient g une fonction unaire de $D_n \mapsto D_n$ et I_g une interprétation de g. Il existe un entier $l < n$ et une permutation σ appartenant à $Perm(D_n)$ telle que $\sigma(I_g)$ vérifie :*

- *Tous les circuits sont dans le sous–ensemble $\{0, \ldots, l\}$ de D_n.*
- *Pour chaque circuit $c = (e_1, e_2, \ldots, e_k)$ de g, on a $\forall j < k$, $g(e_j) = e_{j+1}$ et $g(e_k) = e_1$.*
- *Si c_1 et c_2 sont deux circuits consécutifs alors $|c_1| \leq |c_2|$.*
- *Si $i > l$ alors $g(i) < i$ et $g(i) = j \Rightarrow g(i+1) \geq j$.*

Preuve Comme pour la proposition 6.3, la démonstration se fait en construisant la permutation σ.

Exemple 6.6
L'interprétation canonique I_g est symétrique à l'interprétation I'_g. Elle sont liées par la permutation $\sigma = (0,1,2,6) \circ (5) \circ (7,3) \circ (8,4)$. En effet, on a $I_g = \sigma(I'_g)$.

g	0	1	2	3	4	5	6	7	8
I_g	0	2	1	4	3	0	1	6	6

	0	1	2	3	4	5	6	7	8
I'_g	1	0	0	2	2	6	6	8	7

L'interprétation I_g contient 3 circuits, dans ce cas le nombre l est égal à 4, $mdn_1 = 0$, $mdn_2 = 2$. Les éléments 3 et 4 restent interchangeables. La figure 6.3 montre le graphe d'interprétation de cette fonction.

De même que nous n'avons généré que les interprétations canoniques d'une fonction bijective, nous pouvons donc ne générer que les interprétations canoniques d'une fonction unaire quelconque. La construction des interprétations canoniques d'une fonction unaire quelconque s'avère plus difficile que dans le cas d'une fonction bijective. L'algorithme 6.2 en décrit ce mécanisme. Il utilise les résultats des propositions 6.6 et 6.5.

L'algorithme interprète la fonction unaire en commençant par les circuits, tout en mémorisant les valeurs des mdn de chaque taille de cycle. Ensuite, à partir d'un certain rang (égal à l), les affectations

Algorithme 6.2 Construction d'une fonction unaire canonique

Fonction Interprétation_Can(I : interprétation ; A : axiomes ; $r, l, taille, debc$: entier)

Début

Si ($r = n - 1$) **Alors** Recherche_XLNH(I,A) %%Algorithme 6.1

Si ($r > l$) **Alors** %%Partie sans circuits

Pour i de $g(i-1)$ à $r-1$ tel que $i \leq mdn_{|C_i|}$

Faire

Mettre à jour $mdn_{|C_i|}$

$I(g(r)) = i$

Interprétation_Can($I, A, r+1, l, 0, -1$)

Si($r \leq l$) **Alors** %%Partie des circuits

Si $debc \neq -1$ **Alors**

Si ($r < n - 2$) **Alors** %%Continuer un circuit en cours

$I(g(r)) = r + 1$

Interprétation_Can($I, A, r+1, l+1, taille, debc$)

Si ($r - debc + 1 \geq taille$) %%Fermer un circuit

Alors

$I(g(r)) = debc$

Interprétation_Can($I, A, r+1, l+1, i - debc + 1, -1$)

Sinon %%Les circuits sont tous fermés

Si $taille = 1$ **Alors** %%circuit de taille 1

$I(g(r)) = r$

Interprétation_Can($I, A, r+1, l+1, 1, -1$)

Si ($n - r + 1 \geq taille$) **Alors** %%Créer un nouveau circuit

$I(g(r)) = r + 1$

Interprétation_Can($I, A, r+1, l+1, taille, i$)

Pour chaque $i \in D$ tel que $i \leq mdn_{|C_i|}$

Faire %%Commencer la partie sans circuits

Mettre à jour $mdn_{|C_i|}$

$I(g(r)) = i$

Interprétation_Can($I, A, r+1, l, 0, -1$)

Fin

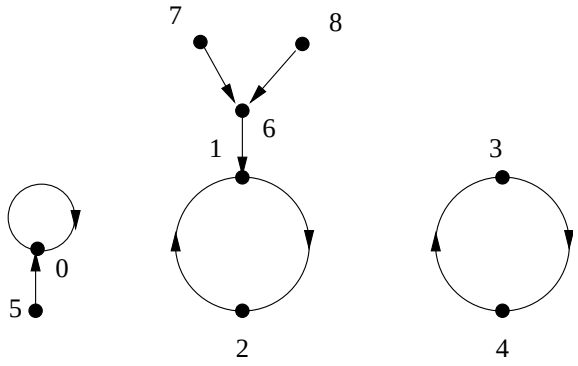


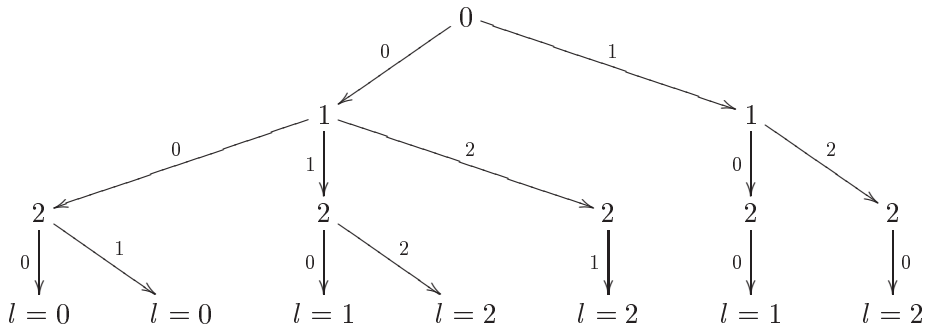
Figure 6.3. *Une interprétation unaire canonique*

ne sont plus que des ramifications sur les circuits en tenant compte de la dernière contrainte de la proposition 6.6 et de la valeur du mdn de chaque taille de cycle. La fonction `Interprétation_Can` est définie de manière récursive. Elle prend comme paramètres l'interprétation I, les axiomes A, l'argument r définissant la cellule de la fonction unaire à instancier, le rang l, la $taille$ des circuits la plus grande utilisée jusqu'alors et le début ($debc$) d'un circuit s'il y en a un en cours de construction (-1 sinon). Cette fonction est appelée initialement avec `Interprétation_Can(I,A,0,0,1,-1)`. Lorsque toute la fonction unaire a été interprétée, nous continuons la recherche avec l'algorithme `Recherche_xlnh`. Les valeurs v prises par les autres cellules lors de l'appel à `Recherche_xlnh` seront toutes les valeurs plus petites que $mdn_{|C_v|}$ ainsi que les valeurs supérieures à l.

Dans l'exemple qui suit, nous nous proposons de montrer le fonctionnement de l'algorithme `Interprétation_Can`.

Exemple 6.7

Nous montrons le fonctionnement de l'algorithme `Interprétation_Can` lorsque l'on génère toutes les interprétations canoniques d'une fonciton g dans un domaine $D_3 = \{0, 1, 2\}$. La construction est caractérisée par un arbre dans lequel les noeuds représentent les arguments de la fonction et les arêtes représentent les valeurs. Nous donnons la valeur de l sur les feuilles, dans le cas d'une interprétation bijective $l = 2$. L'algorithme `Interprétation_Can` construit donc 7 interprétations différentes dont 3 bijections.



Par exemple, l'interprétation canonique associée à la feuille la plus à gauche est $g(0) = g(1) = g(2) = 0$. Bien entendu, dans ce cas $l = 0$. ✍

Nous signalons que cet algorithme ne considère pas uniquement les interprétations non isomorphes d'une fonction unaire. En effet, les critères fournis par la proposition 6.6 laissent passer certaines interprétations symétriques. Le tableau 6.2 donne, pour diverses tailles de domaine, le nombre d'interprétations total, le nombre d'interprétations non symétriques, le nombre d'interprétations générées par l'algorithme `Interprétation_Can` et enfin le nombre d'interprétations générés par l'heuristique LNH. Jusqu'à l'ordre 3, l'algorithme `Interprétation_Can` ne génère que les interprétations non symétriques. Le travail inutile fourni après cet ordre reste relativement réduit. Ce n'est pas le cas de l'heuristique LNH qui génère beaucoup d'interprétations symétriques. Les résultats donnés par ce tableau illustrent bien la quantité de travail inutile fait lorsque l'on génère une fonction et que l'on ne prend pas en compte les isomorphismes.

Taille du domaine (n)	3	4	5	6	7
Interprétations totales (n^n)	27	256	3 125	46 656	823 543
Interprétations non symétriques	7	19	47	130	343
Interprétations générées par LNH	18	96	600	4 320	35 280
Interprétations générées par `Interprétation_Can`	7	20	59	190	631

Tableau 6.2. *Nombre d'interprétations générées par `Interprétation_Can`*

6.4 Recherche et exploitation dynamique des symétries

Nous étudions dans cette section un cadre plus général de la notion de symétrie pour la génération de modèles finis. Nous montrons que les symétries exploitées par les heuristiques LNH et XLNH sont des cas particuliers de ce cadre.

Bien qu'intéressantes, les symétries de départ exploitées par l'heuristique LNH (partie $\{mdn \ldots n-1\}$) disparaissent en avançant dans la recherche de la solution. Du fait des propagations, on arrive rapidement à utiliser tous les individus du domaine, et l'ensemble des individus symétriques au sens LNH se vide. Plus précisément, la partie symétrique $\{mdn+1 \ldots n-1\}$ diminue et devient vide quand la frontière mdn atteint la valeur $n-1$. Inversement la partie non symétrique $\{0 \ldots mdn\}$ augmente et finit par contenir tous les éléments de D_n. Ainsi les symétries LNH de la partie $\{mdn+1 \ldots n-1\}$ disparaissent au fur et à mesure, mais il reste beaucoup d'autres symétries non exploitées dans la partie $\{0 \ldots mdn\}$. On observe le même comportement lors de l'utilisation de l'heuristique XLNH (voir l'algorithme 6.1).

Nous nous sommes donc intéressés à la détection et à la suppression des symétries existant entre

les individus de la partie $\{0, \ldots, mdn\}$ non symétrique au sens LNH. Ces symétries sont détectées et éliminées dynamiquement pendant la recherche du modèle fini. Cette détection pourra aisément être combinée avec les heuristiques LNH et XLNH.

Ce travail a été publié aux «Journées Francophones de la Programmation en Logique et Programmation par Contraintes» (JFPLC'2001) [2] et au workshop «SymCon'01» [3].

6.4.1 Notion de symétries

A notre sens, une symétrie d'une théorie $T = (S, F, A)$ est une permutation des individus qui laisse invariante la structure du modèle partiel en cours de construction et l'ensemble des clauses courant. Formellement :

Notation 6.2 *Soient $T = (S, F, A)$ une théorie et I une interprétation partielle. On note A^t l'ensemble des instances terminales de A (voir section 4.2.3). On note également T_I = la théorie T simplifiée par I, A^t_I est alors l'ensemble des instances terminales simplifié par I et associé à la théorie T_I.*

Définition 6.14 *Soient $T = (S, F, A)$ une théorie, I l'interprétation courante et σ une permutation définie sur l'ensemble des types S. La permutation σ est une symétrie de T_I, si $\sigma(I) = I$ et $\sigma(A^t_I) = A^t_I$.*

La définition 6.14 donne de nouvelles conditions de la symétrie comparativement au cas propositionnel [16]. Pour vérifier les conditions de symétrie en un noeud donné de l'arbre de recherche, la permutation σ doit laisser invariant l'ensemble de clauses représentant l'interprétation partielle I et celles du système A^t définie en ce noeud. L'interprétation courante est en effet une structure mathématique qui regroupe des équations (clauses unitaires) de la forme : $f(e_1, \ldots, e_k) = e_{k+1}$ où $e_i \in s_i \in S$ qui sont en général sensibles à toute permutation des individus e_i. Ceci est différent du cas propositionnel où l'interprétation partielle n'est pas concernée par la permutation car elle ne contient pas des variables apparaissant dans l'ensemble courant des clauses.

Définition 6.15 *Soit $T = (S, F, A)$ une théorie. Deux individus e_1 et e_2 d'un type $s \in S$ sont symétriques s'il existe une symétrie σ de T telle que $\sigma(e_1) = e_2$.*

Remarque 6.3 Les éléments d'un cycle de symétrie sont symétriques deux à deux.

6.4.2 Détection des symétries

Les permutations vérifiant les conditions de la symétrie (définition 6.14) sont construites à partir de compositions de cycles de permutations. L'algorithme de détection des symétries comporte deux étapes distinctes : La première consiste à partitionner chaque type $s \in S$ de la théorie en classes

d'équivalence où les individus doivent vérifier certaines conditions nécessaires de la symétrie que nous discutons dans la proposition 6.7. Deux éléments peuvent être symétriques si et seulement si ils appartiennent à la même classe d'équivalence.

Les conditions nécessaires de la symétrie

Pour être symétriques, les individus des différents types doivent satisfaire les conditions nécessaires de la symétrie.

Définition 6.16 *Si I est une interprétation d'une théorie $T = (S, F, A)$ et $f \in I \cap F$ une fonction d'arité n, alors $\#_{I_{f_i}}(a)$ est le nombre d'occurences dans I de l'individu a comme i^{eme} argument de la fonction f. De même, $\#_{I_{f_{val}}}(a)$ est le nombre d'occurences dans I de l'élément a comme valeur de la fonction f.*

Exemple 6.8

Nous reprenons l'exemple 4.5. Les axiomes sont

- $\forall x, h(x, x) = x$
- $\forall x \forall y, h(h(x, y), x) = y$

Le domaine associé à cette théorie est D_5.

La table de gauche représente une interprétation partielle I.

h	0	1	2	3	4
0	0	2	1	4	-
1	2	1	0	-	-
2	1	0	2	-	-
3	-	-	-	3	0
4	3	-	-	-	4

i	$\#_{I_{h_1}}(i)$	$\#_{I_{h_2}}(i)$	$\#_{I_{h_{val}}}(i)$
0	4	4	4
1	3	3	3
2	3	3	3
3	2	2	2
4	2	2	2

Le tableau de droite représente alors les occurences des éléments de D_4 dans h. ✍

Proposition 6.7 (conditions nécessaires) *Soient une théorie $T = (S, F, A)$ ayant I comme interprétation partielle courante, a et b deux éléments d'un type $s \in S$. Pour que a et b soient symétriques dans T_I (appartiennent à un même cycle de symétrie), ils doivent satisfaire les conditions suivantes :*

- $\forall f \in I \cap F, \quad \forall i \in \{1, \ldots, \textit{arité}(f)\}, \quad \#_{I_{f_i}}(a) = \#_{I_{f_i}}(b)$
- $\forall f \in I \cap F, \quad \#_{I_{f_{val}}}(a) = \#_{I_{f_{val}}}(b)$.

Preuve Montrons la deuxième condition, la preuve est identique pour la première. Soient I l'interprétation courante de la théorie T, σ une symétrie de T_I, et deux individus a et b d'un type $s \in S$, tels que $\sigma(a) = b$. Supposons qu'il existe une fonction $f \in I \cap F$ telle que $\#_{I_{f_{val}}}(a) \neq \#_{I_{f_{val}}}(b)$. On aura donc, $\#_{\sigma(I)_{f_{val}}}(b) \neq \#_{I_{f_{val}}}(b)$, car le nombre d'occurences de b dans I comme valeur de f

est différent de celui de a. Ceci implique que $\sigma(I) \neq I$ d'où la contradiction car σ est une symétrie de T_I. ⏎

Ces conditions nécessaires forment une étape très importante de l'algorithme de détection des symétries. Elles permettent de réduire considérablement l'espace de recherche des permutations. Elles partitionnent les individus en classes d'équivalence formant ainsi les candidats potentiels sur lesquels s'effectue la recherche de la symétrie.

Calcul de la symétrie : vérification de la condition suffisante

La deuxième étape est celle qui construit la permutation (la symétrie) en utilisant les classes d'équivalence préalablement définies.

Une symétrie en calcul propositionnel [16] est une permutation qui doit laisser invariant l'ensemble de clauses courant. Cette condition rend un peu lourde la détection quand le cardinal de l'ensemble de clauses est important. Dans les modèles finis nous simplifions le calcul. En effet, si l'ensemble d'axiomes de départ ne contient aucune constante, il suffit de chercher une permutation laissant invariante l'interprétation partielle courante I. Nous montrons que dans ce cas l'invariance de l'ensemble de clauses terminales A_I^t correspondant est garanti. Ceci nous dispense donc d'une lourde tâche qui est la vérification de son invariance. Formellement, on a la proposition suivante :

Proposition 6.8 *Soient une théorie $T = (S, F, A)$, I l'interprétation partielle courante et σ une permutation de $Perm(S)$. Si A ne contient aucun individu e d'un type $s \in S$ et si $\sigma(I) = I$ alors $\sigma(A_I^t) = A_I^t$.*

Preuve Si A ne contient aucun élément $e \in s$, alors toute permutation $\sigma \in Perm(S)$ vérifie $\sigma(A^t) = A^t$. Donc $\sigma(A_I^t) = \sigma(A^t)_{\sigma(I)} = A^t_{\sigma(I)} = A_I^t$. ⏎

Ainsi, nous obtenons un résultat important pour la détection des symétries. La proposition 6.8 simplifie la condition suffisante de la symétrie de la définition 6.14. En conséquence l'efficacité de détection sera meilleure.

Remarque 6.4 Dans le cas où des individus apparaissent dans un sous-ensemble de clauses $A' \subset A$ de la théorie $T = (S, F, A)$, la permutation σ doit laisser invariant A'^t en plus de l'invariance de I, c'est à dire $\sigma(I) = I$ et $\sigma(A'^t) = A'^t$.

Nous montrons maintenant que les symétries traitées par les stratégies LNH et XLNH vérifient les conditions $\sigma(I) = I$ de la proposition 6.8.

Proposition 6.9 *Soient une théorie $T = (S, F, A)$, I l'interprétation partielle courante et σ une permutation au sens* LNH. *Si A ne contient aucune constante alors $\sigma(I) = I$.*

Preuve La permutation LNH σ forme un cycle complet des individus de la partie $\{mdn+1, \ldots, n-1\}$ du domaine. L'interprétation I est formée d'individus de la partie $\{0, \ldots, mdn\}$ et est donc indépendante de la permutation σ, et $\sigma(I) = I$. ❑

Proposition 6.10 *Soient une théorie $T = (S, F, A)$, I l'interprétation partielle courante et σ une permutation au sens* XLNH. *On a alors $\sigma(I) = I$.*

Preuve L'interprétation $I = I_g^b \cup I_g^s \cup I'$ où g est la fonction unaire et $I_g^b \cup I_g^s$ une interprétation canonique de g avec I_g^b la restriction bijective de I_g comprise entre 0 et l. σ est une permutation au sens XLNH, on a donc :

- $\sigma(i) = i$ pour tout $i \in \{l+1, \ldots, n-1\}$
- Pour chaque cycle $(i_1, \ldots, i_k)$ de la restriction bijective de g :
  - $\forall j \in \{1, \ldots, k\} \sigma(i_j) = i_j$ si $i_j \in I_g^s \cup I'$
  - σ contient le cycle $(i_1, \ldots, i_k)$ sinon

Par conséquent on a $\sigma(I) = \sigma(I_g^b \cup I_g^s \cup I') = \sigma(I_g^b) \cup \sigma(I_g^s) \cup \sigma(I') = I_g^b \cup I_g^s \cup I' = I$. ❑

Les symétries fournies par les heuristiques LNH et XLNH sont donc des cas particuliers de notre cadre d'étude.

L'algorithme 6.3 donne une description du schéma de la recherche des symétries dans le cas d'une implantation avec LNH. C'est une procédure de type backtrack qui dans le pire des cas peut avoir une complexité exponentielle. Mais dans la pratique, les symétries, quand elles existent, sont calculées avec un petit nombre de backtracks et n'affectent pas trop le temps de recherche des modèles. En théorie, la complexité du problème de recherche des symétries est équivalente à celle de recherche d'isomorphismes de graphe, une démonstration en est donnée par CRAWFORD dans [31]. Le problème d'isomorphisme de graphe est un problème intéressant de la théorie de la complexité car c'est un problème ouvert, il n'a pas encore été classifié dans P ou dans NP. Pour l'instant on ne connaît que des algorithmes exponentiels de détection d'isomorphismes de graphe. C'est pour cette raison que la plupart des méthodes recherchant dynamiquement des symétries sont incomplètes, elles stoppent au bout d'un certain nombre de backtracks ou au bout d'un certain temps. Du fait de la structure des problèmes rencontrés qui possèdent beaucoup d'isomorphismes , nous avons gardé le caractère complet dans notre algorithme DSYM.

Dans la figure 6.3, $Cl(e, s)$ désigne la classe d'équivalence de l'individu e du type $s \in S$ définie par les conditions nécessaires de la symétrie (proposition 6.7). La fonction `Partitionner`(s,I) crée les classes d'équivalence du type $s \in S$ correspondant aux conditions nécessaires de la symétrie. *Mdn(s)* est égal à la valeur du MDN associé au type $s \in S$. Cette valeur définit la frontière supérieure du sous-domaine $\{0 \ldots mdn\}$ de s dans lequel nous cherchons les symétries. Les deux fonctions précédentes sont très élémentaires, leurs codes respectifs ne sont pas donnés. L'exemple 6.9 montre le comportement de l'algorithme de recherche des symétries.

Algorithme 6.3 La procédure de Recherche de symétries (DSYM)

Fonction DSYM(I : Interprétation ; Var σ : permutation)
Sortie : Vrai si la permutation σ construite vérifie $\sigma(I) = I$, Faux sinon

Fonction Construction(Var σ : permutation)
Début
Si $\sigma(I) = I$ **Alors Retourner** Vrai
Choisir un type s et $e \in s$ tel que $0 \leq e \leq Mdn(s)$ et
Pour Chaque $a \in Cl(e, s)$
Faire
$\sigma(e) = a$
$Cl(e, s) = Cl(e, s) - a$
Si Construction(σ)=Vrai **alors Retourner** Vrai
Retourner Faux
Fin

Début
I=interprétation courante
Pour Chaque type s faire Partitionner(s, I).
Retourner Construction(σ)
Fin

Exemple 6.9

Reprenons une nouvelle fois les axiomes de l'exemple 4.5 :

- $\forall x, h(x, x) = x$
- $\forall x \forall y, h(h(x, y), x) = y$

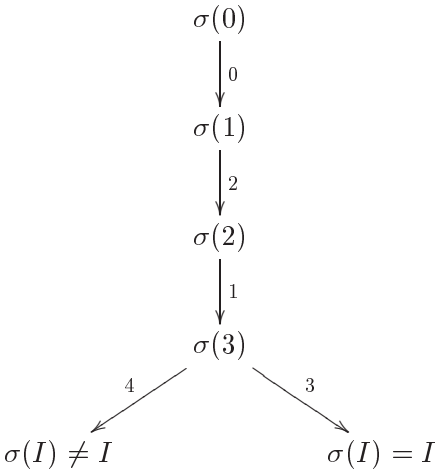
Nous utilisons un domaine de taille 5. Soit l'interprétation partielle courante I soit :

h	0	1	2	3	4
0	0	2	1	4	-
1	2	1	0	-	-
2	1	0	2	-	-
3	-	-	-	3	0
4	3	-	-	-	4

Les classes d'équivalences du domaine D_5 sont les suivantes :

- $Cl(0, D_5) = \{0\}$
- $Cl(1, D_5) = Cl(2, D_5) = \{1, 2\}$
- $Cl(3, D_5) = Cl(4, D_5) = \{3, 4\}$

L'arbre de recherche d'une symétrie laissant inchangée I est illustré ci–dessous. Les noeuds correspondent à des arguments de σ, les arêtes aux valeurs.



L'invariance de l'interprétation n'est pas satisfaite lorsque $\sigma(3) = 4$ car $h(0,3) = 4$ et $h(0,4)$ n'est pas déterminé. Par contre, la permutation $\sigma = (0) \circ (1,2) \circ (3) \circ (4)$ laisse invariante l'interprétation partielle I. C'est une symétrie. ✍

6.4.3 Exploitation des symétries

Dans cette section, nous donnons des propriétés permettant d'utiliser les symétries détectées afin d'améliorer l'efficacité des générateurs de modèles finis.

Proposition 6.11 *Soient $T = (S, F, A)$ une théorie, I l'interprétation courante invariante par $\sigma \in Perm(S)$ et t un terme terminal non instancié. Si l'affectation courante dans I est $t = e$, alors les deux extensions $I \cup \{t = e\}$ et $I \cup \{\sigma(t = e)\}$ de I sont symétriques.*

Preuve Comme I est invariante par σ, alors $\sigma(A_I^t) = A_I^t$ (proposition 6.8). Les interprétations $I \cup \{t = e\}$ et $\sigma(I \cup \{t = e\})$ sont symétriques (définition 6.7). Or $\sigma(I \cup \{t = e\}) = \sigma(I) \cup \{\sigma(t = e)\} = I \cup \{\sigma(t = e)\}$. Ce qui implique que les deux extensions $I \cup \{t = e\}$ et $I \cup \{\sigma(t = e)\}$ sont symétriques. ⧉

Ainsi, les instanciations $t = e$ et $\sigma(t = e)$ déterminent deux extensions symétriques de l'interprétation I. Il suffit d'en considérer une seule, si l'on ne veut calculer que les modèles «non isomorphes». Comme on s'intéresse beaucoup plus au problème de la consistance de la théorie T, il est important de savoir le lien de cohérence entre les deux extensions. C'est ce que nous exprimons par la proposition suivante :

Proposition 6.12 *Si l'interprétation partielle I est cohérente dans $T = (S, F, A)$, alors [l'extension $I \cup \{t = e\}$ est cohérente ssi $I \cup \{\sigma(t = e)\}$ est cohérente]*

Preuve C'est une conséquence de la proposition 6.11 qui garantit que les deux extensions sont symétriques et du fait que les interprétations symétriques sont équivalentes pour la cohérence (théorème 6.2 et définition 6.7). ⧉

La proposition 6.12 exprime un résultat important que nous exploitons pour élaguer l'arbre de recherche d'un modèle fini. Le gain se traduit par la coupure que nous réalisons quand l'instanciation $t = e$ mène à une incohérence. En effet, si l'instantiation $t = e$ est inconsistante dans l'interprétation partielle I, alors l'instantiation symétrique $\sigma(t = e)$ le sera aussi. Grâce à cette information, le parcours dans l'arbre de recherche de la branche inutile correspondant à $\sigma(t = e)$, où $\sigma(t) = \sigma(e)$, est évité. C'est la coupure des symétries.

Nous montrons dans ce qui va suivre qu'une symétrie σ d'ordre k peut réaliser k-1 coupures de branches dans l'arbre de recherche d'un modèle fini. Il est alors important de détecter des symétries d'un grand ordre pour assurer une meilleure utilisation.

Proposition 6.13 *Soient une théorie $T = (S, F, A)$, $\sigma \in Perm(S)$ et I l'interprétation courante. Si $\sigma(I) = I$, alors $\forall j \in \mathbb{Z}, \sigma^j(I) = I$.*

Preuve Par induction sur $j \in \mathbb{Z}$. La propriété est vraie pour $j = 0$, car $\sigma^0(I) = id(I) = I$. Supposons que la propriété est vraie à l'ordre j, montrons que cela reste vrai pour $j+1$. On a $\sigma^{j+1}(I) = \sigma(\sigma^j(I))$ et $\sigma^j(I) = \sigma(I)$ par l'hypothèse de récurrence. Donc $\sigma^{j+1}(I) = \sigma(I) = I$. ⧉

Remarque 6.5 Si dans une théorie $T = (S, F, A)$, l'interprétation I est invariante par σ, $\forall j \in \mathbb{Z}$ alors $\sigma^j(A^t_I) = A^t_I$, $\forall j \in \mathbb{Z}$.

Ainsi, nous généralisons le résultat de la proposition 6.12 à une puissance quelconque de σ. On a la proposition suivante :

Proposition 6.14 *Si l'interprétation partielle I est cohérente dans $T = (S, F, A)$ alors [l'extension $I \cup \{t = e\}$ est cohérente ssi $I \cup \{\sigma^j(t = e)\}$, $\forall j \in \mathbb{Z}$ est cohérente]*

Preuve C'est une conséquence des deux propositions 6.12 et 6.13. En effet, $\sigma^j(I) = I$, $\forall j \in \mathbb{Z}$ (proposition 6.13). Ce qui implique que les extensions $I \cup \{t = e\}$ et $I \cup \{\sigma^j(t = e)\}$, $\forall j \in \mathbb{Z}$ sont symétriques et donc équivalentes du point de vue cohérence (proposition 6.12). ⧉

La proposition 6.14 permet une meilleure utilisation de la symétrie σ. En effet, pour une symétrie d'ordre k, $k - 1$ coupures sont réalisées dans l'arbre de recherche quand $t = e$ est inconsistante. Ces coupures correspondent aux instanciations inconsistantes $\sigma^i(t = e)$ pour $i \in \{1, \ldots, k - 1\}$.

Exemple 6.10
Soit une théorie contenant une fonction binaire f. Soit $\sigma = (0) \circ (1, 2, 3, 4)$ une symétrie laissant inchangée l'interprétation courante. Si le prochain terme terminal à instancier est $f(1, 1) = 2$, les valeurs supprimées par σ seront alors $\sigma((f(1, 1) = 2)) = (f(2, 2) = 3)$, $f(3, 3) = 4$ et $f(4, 4) = 1$. ✍

6.4.4 Combinaison des symétries détectées avec celles de LNH

Nous pouvons combiner notre algorithme de recherche des symétries DSYM avec la suppression des valeurs interchangeables fournies par l'heuristique LNH. Nous détectons alors, moyennant la procédure DSYM, une symétrie σ sur les individus de la partie $\{0 \dots mdn\}$. La symétrie σ fournit en général $k-1$ extensions équivalentes de l'interprétation I si k est son ordre. Les deux symétries LNH et σ sont définies sur deux parties indépendantes du domaine, leur combinaison est faite de façon très naturelle. L'association de ces deux symétries permet de réaliser au total $n-mdn-1+k$ coupures de branches en chaque noeud de l'arbre de recherche. De la même manière, nous pouvons utiliser notre algorithme de recherche des symétries avec les symétries fournies par l'heuristique XLNH. Nous n'avons pas besoin de détecter des symétries durant la construction de la fonction unaire bijective étant donné que nous ne construisons que des interprétations non symétriques.

6.4.5 Symétries horizontales et verticales

L'algorithme 6.3 ne détecte qu'une seule symétrie de l'interprétation courante. Détecter toutes les symétries de l'interprétation courante serait à coup sur totalement inefficace. La perte de temps à rechercher toutes les symétries annulerait les gains obtenus par la suppression de branches dans l'arbre de recherche. Donc, L'algorithme de recherche de modèles que nous combinons avec la procédure DSYM va avoir un comportement variant avec la symétrie détectée.

C'est pour cette raison qu'il nous semble nécessaire de classifier les symétries en deux types, dépendant du terme terminal ce associé à l'affectation courante. Les deux classes proposées sont les suivantes :

- $\sigma(ce) = ce$, c'est le cas où les branches supprimées se trouvent au noeud courant. C'est ce que nous appelons symétries *horizontales*
- $\sigma(ce) \neq ce$, dans ce cas les branches supprimées seront situées plus bas dans l'arbre de recherche. C'est ce que nous appelons symétries *verticales*.

Les deux classes de symétries ont un impact très différents sur les branches supprimées. Dans le cas d'une symétrie horizontale, nous pouvons directement supprimer les branches isomorphes du terme terminal ce. Ceci est fait très simplement et assure une réduction de l'arbre de recherche. Dans le cas de symétries verticales, les branches coupées se trouvent plus bas dans l'arbre de recherche et il se peut que la coupure ne soit d'aucune utilité (si l'inconsistance est découverte avant). L'exemple 6.11 explicite ces notions.

Remarque 6.6 Les symétries détectées par les heuristiques LNH et XLNH sont toutes deux des symétries horizontales.

Exemple 6.11

Supposons que le terme terminal ce du noeud courant soit $f(0,1)$. La figure 6.4 montre l'effet d'une

symétrie horizontale $\sigma = (0) \circ (1) \circ (3, 4, 2)$ sur l'arbre de recherche. Les branches pointillées sont celles suprimmées. Lorsque l'on backtracke de la branche $f(0, 1) = 2$, on supprime les deux branches suivantes car les interprétations $f(0, 1) = 3$ et $f(0, 1) = 4$ sont symétriques.
Quand à la figure 6.5, elle montre l'effet de la symétrie verticale $\sigma = (0) \circ (1, 2, 3, 4)$. Dans ce cas, lorsque l'on backtracke de la branche $f(0, 1) = 0$ on supprime sur les autres branches l'interprétation $f(0, 2) = 0$. Lorsque l'on backtracke de la branche $f(0, 1) = 0$, on supprime sur les autres branches l'interprétation $f(0, 2) = 2$, et ainsi de suite.

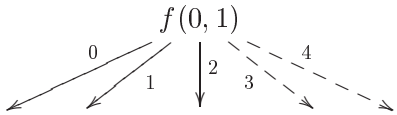


Figure 6.4. *Symétrie horizontale*

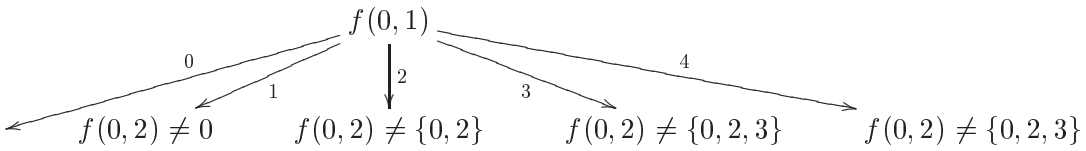


Figure 6.5. *Symétrie verticale*

Il est toujours préférable de rechercher des symétries horizontales pour élaguer les branches les plus hautes de l'arbre de recherche. De plus, imposer la détection de symétries horizontales ou verticales renforce les conditions nécessaires, ce qui réduit l'espace de recherche des symétries. Les expérimentations que nous avons conduites nous ont amené à combiner les impacts des deux types de symétries. A chaque noeud, une symétrie horizontale et une symétrie verticale sont recherchées et exploitées.

6.5 Conclusion

Nous avons proposé deux approches de détection et de suppression des isomorphismes pour les générateurs de modèles finis de logiques du premier ordre. Supprimant plus d'individus interchangeables, l'heuristique XLNH est une amélioration de l'heuristique LNH. Elle se focalise en premier lieu sur la génération des interprétations canoniques d'une fonction unaire. Elle utilise ensuite ces interprétations pour déterminier sans surcoût de temps des symétries triviales.

Nous avons, par la suite, proposé de nouvelles propriétés concernant les interprétations symétriques. Nous les avons utilisées pour détecter dynamiquement des symétries. Nous avons alors montré comment combiner les symétries détectées dynamiquemement avec les heuristiques LNH et XLNH.

Chapitre 7

Expérimentations

NOUS EXPÉRIMENTONS DANS CE CHAPITRE les différents algorithmes que nous avons proposés dans les chapitres 5 et 6. Nous étudions leur comportement puis nous faisons une étude comparative avec les méthodes SEM [106], FINDER [93] et MACE [71]. La comparaison est divisée en deux partie, l'une portant sur le problème de satisfaisabilité, l'autre sur la recherche de tous les modèles.

Nous avons réalisé toutes nos implantations sur le générateur de modèles SEM[1]. Plusieurs raisons ont dicté ce choix. Tout d'abord, SEM est l'une des méthodes les plus efficaces existant dans la littérature. Son code, écrit en C (environ 5800 lignes), est extrêmement clair. Enfin, pour pouvoir comparer honnêtement les différentes stratégies introduites tout au long de ce mémoire, un même générateur est indispensable.

L'étude expérimentale est faite sur quelques exemples de théories multi–types. La liste des exemples proposée est loin d'être exhaustive. Rien que la librairie TPTP, «Thousands of Problems for Theorem Provers» [96], qui porte bien son nom, propose 28 catégories de problèmes différents contenant chacun quelques dizaines d'instances. D'autres exemples de problèmes existent dans la littérature comme ceux fournis par Larry WOS dans [99]. Tous les problèmes cités ne comportent qu'un seul type.

7.1 Liste des problèmes expérimentés

Dans cette section, nous décrivons l'ensemble des théories sur lesquelles est basée l'étude expérimentale.

7.1.1 Groupes

Nous avons brièvement défini les groupes dans la section 6.1.1. Le tableau 7.1 donne les axiomes décrivant un groupe abélien (commutatif). La fonction $g(x)$ renvoie l'inverse de x, la loi a est la loi

[1] le code source de SEM est accessible à l'adresse Internet http://www.cs.uiowa.edu/~hzhang/sem.html

de composition interne.

Les problèmes issus de la théorie des groupes que nous testons sont :

– groupes abéliens notés AG. Ce problème est satisfaisable.
– groupes non abéliens satisfaisant donc l'axiome supplémentaire $a(c,b) \neq a(b,c)$, où b et c sont des constantes. Ce problème est satisfaisable.
– Le problème GRP100-1 de la librairie TPTP. C'est l'une des nombreuses conjectures qui consistent à trouver des axiomes uniques pour la théorie des groupes. Ce problème est insatisfaisable.

$a(x,\varepsilon) = x$	ε est l'élément neutre
$a(\varepsilon,x) = x$	
$a(x,g(x)) = \varepsilon$	g(x) est l'inverse de x
$a(g(x),x) = \varepsilon$	
$a(x,a(y,z)) = a(a(x,y),z)$	Associativité
$a(x,y) = a(y,x)$	Commutativité

Tableau 7.1. *Axiomes d'un groupe*

7.1.2 Anneaux

Un anneau est un groupe commutatif muni d'une loi multiplicative (m) associative et distributive par rapport à la loi du groupe (a). Les différents axiomes d'un anneau sont listés dans le tableau 7.2.

$a(\varepsilon,x) = x$	$a(x,\varepsilon) = x$	ε élément neutre pour a
$a(g(x),x) = \varepsilon$	$a(x,g(x)) = \varepsilon$	g inverse du groupe
$a(x,a(y,z)) = a(a(x,y),z)$		Associativité de la loi du groupe
$m(x,m(y,z)) = m(m(x,y),z)$		Associativité
$a(m(x,y),m(x,z)) = m(x,a(y,z))$		Distributivité
$a(m(x,z),m(y,z)) = m(a(x,y),z)$		Distributivité

Tableau 7.2. *Axiomes d'un anneau*

Les différents problèmes de la théorie des anneaux que nous avons utilisés sont les suivants :

– RU : anneau unitaire (existence d'un élément neutre pour la multiplication).
– RNA : anneau avec contre–exemple d'associativité de la loi m, ce problème a été proposé dans [103].
– Problème RNG025-8 de la librairie TPTP dont les axiomes sont listés dans la figure 5.5 (Chaque problème de cette librairie possède un rapport de difficulté compris entre 0 (facile) et 1. Celui–ci est égal à 0.67).

Ces problèmes sont tous satisfaisables.

7.1.3 Algèbre ternaire

$f(x, y, g(y)) = x$
$f(x, x, y) = x$
$f(f(v, w, x), y, f(v, w, z)) = f(v, w, f(x, y, z))$
$f(g(y), y, x) = x$
$f(x, y, y) = y$

Tableau 7.3. *Axiomes d'une algèbre ternaire*

Dans [102], Jian ZHANG propose une liste de problèmes mathématiques difficiles pour les générateurs de modèles finis. On y trouve, entre autres, le problème de l'algèbre ternaire, définie par les axiomes de la table 7.3. De nombreuses conjectures concernant les algèbres ternaires existent dans la littérature. Nous nous limitons au problème proposé par Jian ZHANG, correspondant à l'instance BOO019-1 de la librairie TPTP (rapport 0.67). Ce problème consiste à prouver que le dernier axiome est indépendant des 4 premiers. On essaie pour cela de trouver un modèle qui satisfait les 4 premiers axiomes et qui ne satisfait pas le cinquième, donc qui vérifie $f(a, b, b) \neq b$, où a et b sont 2 constantes. C'est un problème satisfaisable.

C'est l'un des premiers problèmes importants résolu par un ordinateur. Ouvert jusqu'en 1977, un modèle a été obtenu à l'aide d'un démonstrateur par résolution par Steven WINKER [98].

7.1.4 Quasigroupes

Un quasigroupe est un opérateur binaire «.» tel que les équations $a.x = b$ et $x.a = b$ possèdent une solution unique pour tout a et b. La table de cet opérateur est telle que chaque ligne et chaque colonne est une permutation de l'ensemble des éléments. Nous traitons uniquement des quasigroupes idempotents, c'est à dire satisfaisant la contrainte supplémentaire $(\forall x \quad x.x = x)$. Ensuite, l'ajout de différents axiomes conduit à la création de 7 problèmes différents. Nous décrivons les axiomes supplémentaires dans le tableau 7.4. Des descriptions complètes des quasigroupes sont données dans [71, 91, 95]. Dans cette étude, nous nous limitons aux problèmes QG1 et QG6.

7.1.5 Vérification de programmes

La vérification de programmes consiste à établir formellement qu'un programme calcule ce pour quoi il a été élaboré. Nous utilisons l'instance PRV004-1 de la collection TPTP (rapport 1.00 ; problème ouvert). Nous ne listons pas les nombreux axiomes de ce problème qui contient six constantes, 3 fonctions unaires et une fonction d'arité 2.

Nom	Contrainte
QG1	$xy = u \wedge zw = u \wedge vy = x \wedge vw = z \rightarrow x = z \vee y = w$
QG2	$xy = u \wedge zw = u \wedge vx = y \wedge vz = w \rightarrow x = z \vee y = w$
QG3	$(x.y).(y.x) = x$
QG4	$(x.y).(y.x) = y$
QG5	$((x.y).x).x = y \wedge x.((y.x).x) = y \wedge (x.(y.x)).x = y$
QG6	$(x.y).y = x.(x.y)$
QG7	$((x.y).x).y = x \wedge ((x.y).y).(x.y) = x$

Tableau 7.4. *Axiomes des quasigroupes*

7.1.6 Logique temporelle

Le dernier des problèmes testés est la formulation au premier ordre des «Linear-time Temporal Logic» (LTL). Les axiomes de ce problème sont listés dans la table 7.5. C'est Jian ZHANG qui a proposé ce problème [104]. Il possède exactement n modèles non isomorphes pour un domaine de taille n. Le challenge proposé est de générer le moins de modèles isomorphes possibles.

$R(x, x)$
$(R(x, y) \wedge R(y, z)) \rightarrow R(x, z)$
$R(x, y) \vee R(y, x)$
$(s(x) = y) \rightarrow R(x, y)$
$R(x, y) \rightarrow ((y = x) \vee (y = s(x)) \vee \ldots \vee (y = s^{n-1}(x)))$

Tableau 7.5. *Axiomes de* LTL *pour un domaine de taille* n

7.2 Comportement des différentes stratégies proposées

Nous étudions maintenant le comportement des différentes stratégies que nous avons proposées. Lorsque l'on veut identifier le comportement d'une méthode énumérative pour SAT on utilise souvent les instances aléatoires (voir chapitre 3). L'étude du comportement de générateurs de modèles finis nécessite un panel représentatif de problèmes.

7.2.1 La méthode VESEM

La méthode VESEM a été introduite à la section 5.1.2. Elle supprime dynamiquement des valeurs de domaines incompatibles avec l'interprétation courante et utilise une heuristique de type UP. Il est intéressant de connaître l'impact de l'utilisation de la proposition 5.3 qui détecte des valeurs

de domaines compatibles avec l'affectation courante, qui ne peuvent donc pas être supprimées. Le tableau 7.6 montre la résolution de trois problèmes différents lorsque VESEM utilise ou non le résultat de cette proposition. La colonne «Propagations» donne le nombre d'affectations effectuées dans la procédure `Up_Heuristique`, quant à la colonne «Temps UP» elle donne le temps requis par cette procédure.

		Sans Proposition 5.3			Avec Proposition 5.3		
Problème	Taille	Temps	Temps UP	Propagations	Temps	Temps UP	Propagations
RU	13	16.2	13.5	16 710	6.4	3.1	12 520
QG6	11	2.4	1.2	29 146	2.4	1.2	29 411
TBA	9	3.3	1.5	1 180	5.2	3.8	1 731

Tableau 7.6. *Impact de la proposition 5.3*

Dans tous les cas, la durée passée dans la procédure `UP_Heuristique` est loin d'être négligeable, au moins la moitié du temps CPU y est consacrée. C'est pour cette raison que des stratégies limitant son utilisation ont été nécessaires pour obtenir une méthode VESEM efficace. Pour les problèmes RU et TBA, l'utilisation de la procédure 5.3 s'avère nécessaire, par contre elle est inutile dans le cas du problème QG6. Il semble donc important de l'introduire dans la procédure `UP_Heuristique`, son implantation ne rajoutant aucun surcoût de calcul. De plus, le seul intérêt qu'il y a de propager, dans la procédure `UP_Heuristique`, les valeurs de domaines connues pour être compatibles avec l'interprétation courante concerne le choix heuristique de la cellule à instancier qui sera, dans ce cas, plus précis.

Il est d'ailleurs important de savoir s'il est judicieux de joindre l'heuristique UP à la suppression des valeurs de domaines. La table 7.7 donne l'impact de l'utilisation de cette heuristique sur deux problèmes différents.

		Avec heuristique		Sans heuristique	
Problème	Taille	Temps	Noeuds	Temps	Noeuds
RU	13	6.4	1 686	5.4	948
QG6	12	150	929 781	300	1 484 539

Tableau 7.7. *Impact de l'heuristique* UP

Contrairement à ce à quoi l'on pouvait s'attendre, l'heuristique UP ne donne pas toujours des résultats satisfaisants. Autant pour le problème QG6 le gain est important, autant pour le problème RU le temps de résolution et le nombre de points de choix de l'arbre de recherche sont augmentés. Il semble que le nombre de symboles fonctionnels de la théorie ait une influence sur l'efficacité de cette

heuristique. Plus il y a de fonctions, moins elle parait efficace.

Pour terminer, le tableau 7.8 donne le nombre de valeurs de domaines supprimées et le rapport entre ce nombre et le nombre de propagations de contraintes faites dans les appels à la procédure `UP_Heuristique`.

Problème	Taille	Valeurs Supprimées	Appels à `UP_Heuristique`	Rapport
GRP100-1	4	8 568	7 702	1.11
RNA	7	3 628	3 687	0.98
RU	13	10 044	1 686	5.95

Tableau 7.8. *Nombre de valeurs de domaines supprimées par la méthode* VESEM

Plus ce rapport est important, plus la méthode VESEM améliore SEM. L'efficacité de la méthode VESEM est donc liée à celle de l'heuristique UP et à la quantité de valeurs de domaines supprimées. Ces divers facteurs nous permettront de mieux appréhender son comportement lors des expérimentations faites dans la section 7.3.

7.2.2 Les heuristiques GOT et LNHO

Les heuristiques GOT et LNHO ont été décrites dans la section 5.2. Elles sont définies à partir de l'ordre généré par le graphe de dépendances. La table 7.9 montre les résultats obtenus lors de la résolution du problème RU de taille 7 avec différents ordres. Nous avons utilisé l'heuristique GOT qui génère fonction après fonction. L'ordre $g < a < m$ est créé par la fonction `Ordre` (algorithme 5.2). Il donne des résultats bien meilleurs que les 3 autres. L'arbre de recherche contient très peu de noeuds. Ceci est dû au fait que beaucoup de propagations de contraintes sont faites en commençant la recherche par la fonction g.

	$g < a < m$	$g < m < a$	$a < g < m$	$a < m < g$
Temps	0.06	2.9	12	600
Noeuds	268	16 973	488 480	18 561 266

Tableau 7.9. *Différents ordres pour la résolution de* RU

L'utilisation d'un ordre inverse à celui que génère la fonction `Ordre` s'avère totalement inefficace.

Les dépendances sont donc une notion importante qui accélèrent la recherche. Cette stratégie, qui consiste à réduire la hauteur de l'arbre de recherche, est sensiblement différente de celle utilisée dans les méthodes CTSEM et VESEM qui réduisent la taille des domaines et donc la largeur de l'arbre de recherche.

Nous nous attachons maintenant à comparer brièvement ces deux heuristiques. La table 7.10 donne les temps de calculs obtenus sur quatre problèmes : AG et RU qui ne contiennent pas de constantes apparaissant dans les inéquations, contrairement aux deux autres problèmes, GRP100-1 et RNA[2]

			GOT		LNHO	
Problème	Taille	Sat	Temps	Noeuds	Temps	Noeuds
AG	13	Y	785	1 390 920	0.82	5 869
RU	11	Y	31	26 761	0.8	2 698
GRP100-1	4	N	0.05	6 417	0.22	12 373
RNA	9	N	2.74	90 200	28.3	137 496

Tableau 7.10. *Comportement des heursitiques* GOT *et* LNHO

Le comportement de ces deux heuristiques est totalement dépendant de ces constantes :

– L'heuristique GOT, qui génère fonction après fonction, n'est efficace que lorsqu'elles sont présentes.

– L'heuristique LNHO est moins efficace lorsqu'elles sont présentes.

L'heuristique LNHO qui essaie de garder des valeurs interchangeables le plus longtemps possible, interprète très rapidement les constantes. Contrairement à l'heuristique GOT qui ne s'occupe que de l'ordre et les interprète beaucoup plus tard.

Une nouvelle heuristique

Compte tenu des résultats du tableau 7.10, nous avons amélioré l'heuristique LNHO. Il semble en effet très important de ne générer ces constantes qu'en dernier lieu. Par contre, il est également important d'optimiser au mieux l'heuristique LNH. Nous allons donc utiliser l'heuristique LNHO mais sans interpréter les constantes qui apparaissent dans les inéquations. Ce n'est qu'après l'obtention d'un modèle potentiel que nous les instancions. De cette manière, nous utilisons le mieux possible les qualités des deux heuristiques GOT et LNHO. Le tableau 7.11 donne les résultats obtenus par cette nouvelle heuristique de choix que nous avons appelée LNHO+.

Pour les problèmes AG et RU les résultats obtenus sont bien entendu identiques à l'heuristique LNHO. Par contre pour les deux autres problèmes qui contiennent ces constantes si particulières nous améliorons toujours les temps de calculs des deux heuristiques GOT et LNHO.

[2]Nous rappelons que les heuristiques GOT et LNHO+ interprètent en priorité les constantes apparaissant dans les équations, et plus tard celles qui n'apparaissent que dans les inéquations.

			LNHO+	
Problème	Taille	Sat	Temps	Noeuds
AG	13	Y	0.82	5 869
RU	11	Y	0.8	2 698
GRP100-1	4	N	0.03	3 133
RNA	9	N	0.78	18 756

Tableau 7.11. *Amélioration de l'heuristique* LNHO

7.2.3 La méthode XLNH

L'heuristique XLNH, basée sur la génération d'une fonction unaire, a été introduite dans la section 6.3. Une fois que la procédure `Recherche_XLNH` a généré une interprétation canonique d'une fonction unaire, elle doit choisir, au moyen d'une heuristique, la prochaine cellule à interpréter. La table 7.12 donne les résultats de cette méthode en utilisant d'un côté une heuristique similaire à LNH tenant compte uniquement des mdn de tailles de cycles, et de l'autre en utilisant l'heuristique GOT.

		XLNH avec Ordre		XLNH sans Ordre	
Problème	Taille	Temps	Noeuds	Temps	Noeuds
RU	13	0.56	291	4.3	1 719
GRP100-1	5	0.04	3 168	0.37	14 213

Tableau 7.12. *Importance de l'ordre des fonctions dans l'heuristique* XLNH.

Les résultats sont sans appel. Lorsque l'on n'associe pas l'heuristique GOT à la procédure de recherche, les mdn des cycles augmentent trop vite et finissent par ne plus fournir de valeurs interchangeables. Il semble d'ailleurs logique qu'une fois que l'on a commencé par générer fonction par fonction, avec dans ce cas précis des interprétations canoniques, il vaut mieux continuer la recherche de cette manière. D'une certaine façon, la génération des interprétations canoniques de la fonction unaire optimise au mieux l'heuristique GOT. Il est d'ailleurs intéressant de noter que les premiers résultats satisfaisants que nous avons obtenus avec XLNH portaient uniquement sur la génération de groupes abéliens [7]. Cette théorie ne comportant que deux symboles fonctionnels, l'heuristique GOT est naturellement établie.

Une fois les interprétations canoniques construites, la méthode XLNH les utilise pour mettre en avant de nouvelles valeurs interchangeables. Le tableau 7.13 souligne que la suppression de ces symétries est une étape importante de cette méthode.

		Avec Suppression		Sans Suppression	
Problème	Taille	Temps	Noeuds	Temps	Noeuds
RU	10	0.15	148	1.3	3 241
GRP100-1	6	0.09	15 625	0.20	31 220

Tableau 7.13. *Influence de la suppression des valeurs interchangeables dans* XLNH

7.2.4 Recherche dynamique de symétries

La procédure DSYM de recherche dynamique des symétries a été introduite dans la section 6.4. Le tableau 7.14 montre le comportement de la procédure DSYM sur 3 problèmes différents. Il est très important de noter que dans ce cas précis les théories proposées comportent des éléments dans l'ensemble de clauses et non pas uniquement des fonctions. Les fonctions constantes ont donc déjà été instanciées.

En ce qui concerne les groupes abéliens, on remarque que 82 % des appels à la recherche des symétries fournissent un cycle de symétrie (rapport des colonnes *trouvées* sur *Appels* du tableau 7.14). Ceci permet de supprimer un grand nombre d'interprétations isomorphes (rubrique *supprimées*). Dans les problèmes AG, la détection prend un peu plus de temps que pour les autres problèmes car les conditions nécessaires sont toujours vérifiées et ne réduisent pas l'espace de recherche de la symétrie.

Dans les problèmes de groupes non abéliens, l'ajout de la clause $a(1,2) \neq a(2,1)$ exprimant la non–commutativité, induit une contrainte sur les éléments 1 et 2. Ces derniers ne peuvent être symétriques qu'entre eux (cf remarque de la section 6.4). Ceci renforce le pouvoir des conditions nécessaires et permet de réduire le temps de recherche des symétries.

Pour le problème RNA, codant les anneaux non associatifs, on rajoute la clause $p(p(2,3),4) \neq p(2,p(3,4))$ qui exprime la propriété de non–associativité. Cette clause ajoute des contraintes de conditions nécessaires sur les éléments {0,1,2,3,4} qui ne peuvent plus être symétriques qu'avec eux–mêmes.

On peut remarquer que l'apport des symétries croît en rapport avec la croissance de la taille du domaine, donnant ainsi espoir de résoudre des instances de grandes tailles.

Seulement, pour qu'une théorie comporte des éléments dans l'ensemble des clauses, il est nécessaire d'interpréter correctement ces éléments avant le début de la recherche. Ceci n'est pas une chose facile. Hormis les axiomes de non–commutativité et ceux de non–associativité, l'ensemble des théories que nous avons proposées pour cette étude codent les constantes comme des fonctions d'arité 0.

Une fois interprétées, les constantes fournissent un élément de domaine qui ne peut être symétrique qu'avec lui–même. La probabilité de détecter une permutation laissant invariante l'interprétation courante va dépendre du nombre de constantes interprétées. Plus le nombre de constantes est

Pb	Taille	Temps	Temps Sym	Noeuds	Appels	Trouvés	Supprimés
AG	24	68	55	89 213	12 033	9803	118698
	25	79	59	95 635	13 009	10737	133088
NAG	20	171	36	338 669	29388	25921	321123
	21	206	37	357 095	32480	27177	344578
RNA	8	5	0.4	51 491	7872	0	0
	9	10	0.8	81928	11 243	0	0
	10	62	5.3	252720	41 163	18 902	93184

Tableau 7.14. *Comportement de l'algorithme de recherche des symétries.*

important, moins on a de chances de trouver des symétries. Il est donc nécessaire d'interpréter les constantes le plus tard possible pour espérer en détecter. C'est pour cette raison que nous allons combiner l'algorithme DSYM avec l'heuristique LNHO+ ou avec la méthode XLNH. Pour montrer l'intérêt de telles combinaisons, nous donnons dans la table 7.15 le nombre de valeurs supprimées par la recherche dynamique des symétries lorsque nous la combinons avec les heuristiques LNH, LNHO+ et XLNH. Il apparaît donc que la procédure DSYM ne détecte des symétries sur les problèmes RNA et GRP100-1 que lorsqu'elle est utilisée avec les heuristiques LNHO+ ou XLNH.

	Problème RNA (taille 8)			Problème GRP100-1 (taille 4)		
Combinaison avec	Appels	Trouvées	Supprimées	Appels	Trouvées	Supprimées
LNH	14 333	6571	0	129 521	0	0
LNHO+	943	810	316	29	9	15
XLNH	12	12	14	5	8	7

Tableau 7.15. *Influence de l'heuristique* LNHO+ *sur la recherche des symétries*

Après avoir observé le comportement des différentes stratégies proposées, nous allons faire une comparaison sur différentes instances.

7.3 Comparaison des différentes méthodes

Nous nous proposons maintenant de faire une comparaison des méthodes suivantes :

- La méthode SEM[106].
- La méthode FINDER[93].
- La méthode MACE [71].
- La méthode CTSEM qui transforme les clauses réductibles (voir section 5.1.1).

– La méthode VESEM qui supprime des valeurs de domaines incompatibles utilisées avec l'heuristique UP (voir section 5.1.2).
– La méthode SEM accompagnée de l'heuristique LNHO+ (voir section 5.2).
– La méthode SEM accompagnée de l'heuristique LNHO+ et de la procédure de recherche dynamique des symétries DSYM (voir section 6.4).
– La méthode XLNH (voir section 6.3).
– La méthode XLNH accompagnée de la procédure de recherche dynamique des symétries DSYM.

La première partie de cette étude comparative porte sur le problème de satisfaisabilité de certaines théories définies dans la section 7.1. Nous limitons à une heure le temps de résolution maximal et à 100 Mo l'occupation mémoire maximale. Pour chaque théorie, une méthode donnée possède donc une taille de domaine maximale qu'elle peut raisonnablement résoudre.

Remarque 7.1

– Les méthodes MACE et FINDER ont été utilisées sans aucune option susceptible d'améliorer leurs performances.
– Certaines combinaisons de stratégies (CTSEM et VESEM ou VESEM et LNHO+ par exemple) ne donnaient aucun résultat satisfaisant, nous ne les avons donc pas inclues dans cette comparaison.

7.3.1 Le problème NAG

Le tableau 7.16 regroupe les temps de calcul obtenus lors de la résolution du problème NAG. Chaque ligne correspond à une méthode donnée et contient la taille de domaine maximale que celle–ci résout suivant les contraintes de temps et d'espace données. Par exemple, La méthode SEM résout le problème NAG avec 28 éléments en 1969 secondes, et nécessite plus d'une heure pour le résoudre avec 29 éléments. Les tableaux 7.17 à 7.27 qui apparaissent dans la suite de ce chapitre sont construits d'une façon semblable.

La méthode FINDER ne résout ce problème que pour pour une taille de domaine égale à 6. Ensuite de trop grandes réfutations sont construites, ce qui arrête la recherche. La méthode MACE obtient des résultats sensiblement supérieurs aux autres méthodes proposées. De plus, dépassé l'ordre 9, la construction des instances nécessitent plus de 100 Mo de mémoire. Dès la première théorie proposée, la transformation en clauses de la logique propositionnelle montre des limites.

La méthode SEM est améliorée par toutes les stratégies proposées dans ce mémoire, excepté par la méthode XLNH. Cette dernière utilise trop rapidement les éléments interchangeables et génère donc un arbre de recherche beaucoup trop large. De plus, sa combinaison avec la recherche dynamique des symétries donne de très mauvais résultats. L'amélioration de performances enregistrée par l'heuristique LNHO+ est faible. Le problème ne contenant que deux fonctions, la probabilité d'interpréter la mauvaise est réduite. L'ajout de la procédure de recherche dynamique des symétries améliore les

Méthode	Taille des domaines							
	6	9	18	20	28	30	32	34
FINDER	9							
MACE	0.4	3400						
XLNH et DSYM	0.01	0.05	349					
XLNH	0.01	0.04	7	18				
SEM	0.01	0.08	4.2	2	1 969			
LNHO+	0.01	0.09	4.1	8.6	234	1 107		
LNHO+ et DSYM	0.01	0.12	3.8	8.4	183	617	1.6	
VESEM	0.01	0.06	2	0.23	192	30	318	
CTSEM	0.01	0.04	1.6	0.5	102	69	161	709

Tableau 7.16. *Temps de résolution du problème* NAG

temps de calculs de cette heuristique. Le gain obtenu est de plus en plus important et permet d'ailleurs de résoudre des problèmes de plus grande taille. Les méthodes VESEM et surtout CTSEM donnent également des très bons résultats sur ce problème. Cela montre le travail inutile fait lorsque l'on ne supprime pas les valeurs de domaines incompatibles avec l'interprétation courante.

Remarquons que les temps de calculs des méthodes ne sont pas uniformes. Par exemple, la recherche dynamique des symétries SYM met plus de temps que la méthode VESEM pour l'ordre 30 et va beaucoup plus vite pour l'ordre 32. Il est donc important d'éviter de limiter les comparaisons sur une ou deux tailles de domaines. Un large éventail est nécessaire. En partenariat avec la borne supérieure de résolution, cet éventail donne un bon aperçu de l'efficacité réelle d'une méthode.

7.3.2 Le problème GRP

Le tableau 7.17 donne les temps de résolution obtenus sur le problème GRP100-1 de la librairie TPTP. Celui–ci est composé de 6 constantes (dont 5 qui apparaissent dans des inéquations) et de 3 symboles fonctionnels. La méthode FINDER résout en moins de 1 heure uniquement le problème ayant 3 éléments. A cause d'un problème d'affectation dynamique de la mémoire, la méthode MACE ne répond pas au problème de satisfaisabilité de cette instance.

On peut ensuite classifier les autres méthodes en deux grandes catégories. Celles qui utilisent l'ordre généré par le graphe de dépendance (LNHO+, XLNH,...) et celles qui ne l'utilisent pas. Les premières sont incapables de terminer en moins de 1 heure la résolution pour des tailles de plus de 6 éléments.

L'heuristique LNHO+ et surtout la méthode XLNH ont de très bons résultats, les accélérations obtenues sont conséquentes. Dans ces deux cas, l'ajout de la recherche dynamique des symétries

Méthode	Taille des domaines							
	3	4	5	7	9	10	13	15
FINDER	0.35							
CTSEM	0.48	278						
SEM	0.01	10	2 008					
VESEM	0.09	1.1	446					
LNHO+	0.01	0.03	0.05	0.45	45			
LNHO+ et DSYM	0.04	0.06	0.56	0.47	36	2 284		
XLNH	0.01	0.02	0.05	0.32	6.9	7.2	24	689
XLNH et DSYM	0.01	0.02	0.05	0.36	5.2	7.3	25	441

Tableau 7.17. *Temps de résolution du problème* GRP*100-1*

améliore leurs performances.

7.3.3 Le problème RU

Le tableau 7.18 concentre les résultats qui concernent le problème RU. Ici encore, la méthode MACE consomme rapidement au moins 100 Mo de mémoire (à partir de l'ordre 8). La méthode FINDER qui ne supprime aucun isomorphisme est moins rapide que toutes les autres.

Toutes les stratégies que nous avons proposées dans ce mémoire améliorent la méthode SEM. Les meilleurs résultats sont obtenus pour la méthode XLNH, avec ou sans la combinaison de la procédure DSYM. Dans ces deux cas, la limitation n'est pas due au temps de calcul, mais à la quantité de mémoire utilisée. Dans les temps à venir, les efforts algorithmiques qui jusqu'ici étaient essentiellement consacrés à la rapidité des algorithmes, vont peut–être devoir se porter également sur la consommation de mémoire des logiciels.

Le tableau 7.18 est agrémenté du nombre de noeuds nécessaires lors de la génération de modèles d'ordre 7 (nombres entre parenthèses). La méthode MACE parcourt très peu de points de choix. La formulation en calcul propositionnel optimise la propagation des contraintes mais est limitée, nous l'avons remarqué, par le nombre de variables et de clauses qu'elle engendre. Les différentes stratégies que nous proposons réduisent assez fortement le nombre de points de choix de la méthode SEM.

7.3.4 Les problème RNA

La table 7.19 concentre les résultats du problème RNA. Les méthodes FINDER et MACE sont les plus lentes. Les méthodes CTSEM et VESEM qui réduisent les domaines des cellules non encore instanciées n'ont également pas de très bons résultats. L'ordre sur les fonctions permet de générer

Méthode	Taille des domaines										
	6	7		9	14	19	23	25	27	28	45
MACE	3	(4)	9.4								
FINDER	7.4	(400)	27	1 769							
SEM	0.05	(389)	0.08	0.44	41	1 320					
CTSEM	0.03	(114)	0.04	0.13	6.1	413	2 290				
VESEM	0.04	(61)	0.07	0.48	8	743	1 581	3 120			
LNHO+	0.02	(137)	0.03	0.28	5.6	112	654	1 373	1 766		
LNHO+ et DSYM	0.02	(125)	0.04	0.25	5.2	90	473	934	1 121	2 306	
XLNH	0.03	(72)	0.03	0.11	0.89	3.2	7.3	9.2	12	22	147
XLNH et DSYM	0.01	(32)	0.04	0.10	0.89	3	6.6	7.4	9.8	17	87

Tableau 7.18. *Temps de résolution du problème* RU

plus rapidement que la méthode SEM classique. L'ajout de la recherche dynamique des symétries n'apporte aucune accélération à l'heuristique LNHO+. Il n'en va pas de même lorsqu'on l'utilise avec XLNH. Elle permet dans ce cas d'aller jusqu'à deux fois plus vite, le gain augmentant au fur et à mesure. Là encore, la limitation est due à la quantité de mémoire nécessaire pour la résolution du problème RNA de taille 47.

Méthode	Taille des domaines						
	4	6	7	9	15	16	46
FINDER	5.1						
MACE	0.19	3.3	10				
VESEM	0.07	0.76	2.5	141			
CTSEM	0.05	0.14	0.44	5.4	1990		
SEM	0.04	0.86	2.3	24	1 047	200	
LNHO+	0.01	0.02	0.03	0.78	20	0.24	
LNHO+ et DSYM	0.01	0.02	0.03	0.82	18	0.22	
XLNH	0.01	0.03	0.03	0.15	0.97	0.22	362
XLNH et DSYM	0.01	0.03	0.03	0.13	0.85	0.25	188

Tableau 7.19. *Temps de résolution du problème* RNA

7.3.5 Le problème RNG025-8

Le problème RNG025-8 est le seul exemple de théories que nous proposons qui contienne une fonction (autre qu'un prédicat) qui soit strictement sortante. Les résultats obtenus par les différentes méthodes sont listés dans la table 7.20. La méthode FINDER qui est limitée aux fonctions d'arité

Méthode	Taille des domaines					
	4	5	6	7	10	15
LNHO+ et DSYM	0.02	1 037	396			
LNHO+	0.02	964	373			
CTSEM	0.59	1.2	7.44	1 088		
SEM	0.58	1.54	8	1 391		
VESEM	0.44	3.1	15	755		
XLNH et DSYM	0.03	0.09	0.14	0.27	1.5	10
XLNH	0.04	0.07	0.14	0.3	1.7	9.6

Tableau 7.20. *Temps de résolution du problème* RNG*025-8*

inférieure à 2 ne peut résoudre ce problème. Il est le premier que nous proposons qui mette en échec l'heuristique LNHO+. L'ajout de la recherche dynamique des symétries aggrave encore les résultats de cette heuristique. Les méthodes SEM, CTSEM et VESEM ont un comportement similaire, même si cette dernière est un peu plus rapide que les deux autres. La méthode XLNH résout sans difficulté les problèmes jusqu'à l'ordre 15.

7.3.6 Le problème PRV004-1

Les résultats du problème de vérification de circuits sont listés dans le tableau 7.21. Ici, la recherche dynamique des symétries ralentit l'utilisation de l'heuristique LNHO+ car aucune interprétation symétrique n'est supprimée. Les méthodes CTSEM, XLNH et surtout FINDER donnent de très bons temps de calculs. C'est d'ailleurs le premier problème qui montre une supériorité de la méthode FINDER.

Méthode	Taille des domaines					
	6	7	8	9	12	29
MACE	5.5	27				
LNHO+ et DSYM	0.11	1.8	39	1 688		
SEM	0.57	5.6	73	1 126		
LNHO+	0.09	0.88	17	714		
VESEM	0.29	2.6	34	610		
CTSEM	0.06	0.1	0.33	2.02	1 689	
XLNH et DSYM	0.03	0.03	0.04	0.09	0.24	22
XLNH	0.01	0.02	0.05	0.06	0.2	21
FINDER	0.02	0.02	0.02	0.5	1	1.47

Tableau 7.21. *Temps de résolution du problème* PRV

7.3.7 Les quasigroupes

Le deux derniers exemples sur lesquels nous nous penchons portent sur la résolution des quasigroupes QG1 et QG6. Les tableaux 7.22 et 7.23 donnent les résultats obtenus. Ces problèmes sont très particuliers : il ne contiennent qu'une seule fonction qui est binaire. L'heuristique XLNH et celles issus de la création d'un ordre ne s'appliquent pas. Les quasigroupes ne possèdent pas de clauses réductibles et donc la méthode CTSEM ne s'applique également pas.

Méthode	Taille des domaines					
	4	5	6	7	8	9
LNH et DSYM	0.05	0.26	1.14	14.1		
SEM	0.07	0.25	1.15	8.78		
VESEM	0.05	0.15	1	6.3		
FINDER	0.03	0.02	1.23	0.53		
MACE	0.02	0.04	0.09	0.27	59	440

Tableau 7.22. *Temps de résolution du problème* QG1

Jusqu'à présent nous avons remarqué que la méthode MACE était souvent limitée par sa consommation de mémoire. Il en va autrement sur le problème QG1 où c'est la méthode SEM qui consomme plus de 100 Mo pour générer des instances de taille 8 (contre 2 Mo pour MACE). Les clauses qui codent ce problème contiennent beaucoup de littéraux et de variables, par conséquent l'ensemble des clauses terminales est très volumineux. De plus, MACE obtient d'excellents résultats sur ce problème.

Méthode	Taille des domaines					
	6	9	10	11	12	13
FINDER	0.02	0.03				
LNH et DSYM	0.01	0.15	3.6	1 368		
SEM	0.01	0.03	1.48	42	2 683	
VESEM	0.01	0.03	0.28	2.4	150	
MACE	0.01	0.05	0.12	0.78	14.4	365

Tableau 7.23. *Temps de résolution du problème* QG6

La génération de modèles de QG6 est également plus efficace en utilisant la méthode MACE. La méthode VESEM donne également de très bons résultats. Par contre, la recherche dynamique des symétries s'avère inefficace. Lorsque l'on regarde de plus près, on s'aperçoit que son utilisation entraîne un nombre de points de choix plus important. Les valeurs supprimées par les isomorphismes conduisent à instancier ensuite un mauvais terme terminal. Comme nous l'avons dit dans la section

5.2.1, on comprend alors l'intérêt de déterminer un ordre non pas à partir de la syntaxe de la théorie mais à partir de la sémantique de l'ensemble courant des clauses. C'est ce que nous faisons ici de manière détournée avec l'heuristique UP de VESEM.

7.4 Recherche de tous les modèles

Dans la section précédente, nous nous sommes limité à répondre au problème de satisfaisabilité des théories proposées. Nous nous attachons maintenant à rechercher l'ensemble des modèles des problèmes AG, RU, TBA et LTL. Le problème de la recherche de tous les modèles est bien entendu bien plus difficile que celui de la recherche de la satisfaisabilité. C'est un problème qui est #P–complet dans la théorie de la complexité.

Les diverses méthodes pouvant générer des modèles isomorphes, la connaissance du nombre de modèles obtenus va donc donner une indication sur le nombre d'interprétations isomorphes évitées. Les tableaux suivants se voient donc agrémentés de cette donnée, indiquée entre parenthèses.

Remarque 7.2 La méthode FINDER [93] possède l'option `cut` sur les fonctions. Deux modèles ne peuvent être différents uniquement sur les fonctions `cut`. Cette option est très utile lorsque l'on va rechercher tous les modèles de théories ayant des constantes comme le problème TBA. Nous l'avons donc implantée pour améliorer l'efficacité de nos méthodes.

7.4.1 Recherche de tous les modèles de AG

Nous commençons par générer l'ensemble des modèles des groupes abéliens. Les temps de calculs sont donnés dans le tableau 7.24 que nous agrémentons du nombre de modèles non isomorphes (réuni avec la taille des domaines).

Les méthodes MACE et FINDER qui ne suppriment aucun isomorphisme génèrent un grand nombre de modèles. Les méthodes SEM, CTSEM, VESEM et l'utilisation de l'heuristique LNHO+ obtiennent le même nombre de modèles. Ces différentes stratégies ont des temps de calculs comparables.

Ici, la recherche dynamique des symétries accélère les temps de calculs. D'ailleurs, dans le cas de la recherche de tous les modèles, cette stratégie est pleinement exploitée : les valeurs supprimées ont un effet tout au long de la recherche. Lorsqu'on l'utilise avec XLNH, elle fournit presque toujours un cycle de symétrie. Cette remarque est également valable pour une majorité des théories expérimentées dans ce chapitre.

Il est connu que l'ordre de tout sous-groupe d'un groupe divise l'ordre du groupe de départ. C'est pour cela que les groupes dont la taille a beaucoup de facteurs premiers possèdent plus de modèles non isomoprhes. La méthode XLNH met plus de temps à générer les groupes de taille 32 que ceux de taille 37. Elle préserve la structure du problème, contrairement aux méthodes SEM, CTSEM et VESEM qui ont un temps de résolution croissant en fonction de la taille des domaines.

Méthode		Taille des domaines							
		(1) 6	(2) 9	(7) 32	(1) 35	(1) 37	(1) 38	(1) 47	(1) 57
MACE	Temps	1.6	2 532						
	Modèles	(60)	(7 560)						
FINDER	Temps	1.45	1 241						
	Modèles	(27)	(336)						
CTSEM	Temps	0.01	0.09	1 024	1 801				
	Modèles	(6)	(4)	(2 295)	(13)				
SEM	Temps	0.01	0.09	990	1 734				
	Modèles	(6)	(4)	(2 295)	(13)				
VESEM	Temps	0.01	0.11	1 031	1 646	3 384			
	Modèles	(6)	(4)	(2 295)	(13)	(1)			
LNHO+	Temps	0.01	0.07	1 113	2 060	3 425			
	Modèles	(6)	(4)	(2 295)	(13)	(1)			
LNHO+ et DSYM	Temps	0.01	0.08	1 429	1 945	3 018	3 330		
	Modèles	(6)	(4)	(1 037)	(5)	(1)	(14)		
XLNH	Temps	0.01	0.03	178	44	70	566	382	
	Modèles	(2)	(4)	(529)	(13)	(1)	(2)	(1)	
XLNH et DSYM	Temps	0.01	0.03	39	14	17	190	72	252
	Modèles	(2)	(2)	(93)	(3)	(1)	(2)	(1)	(1)

Tableau 7.24. *Temps de résolution lors de la recherche de tous les modèles de* AG

Hans BESCHE et Bettina EICK [17] ont utilisé le programme GAP [82] pour générer les modèles non isomorphes des groupes jusqu'à l'ordre 1000 (excepté 512 et 768). En générant facilement des modèles jusqu'à l'ordre 57, nos méthodes n'ont pas les mêmes capacités. Les raisons d'un si grand écart sont dues au fait que le programme GAP utilise les propriétés connues sur les groupes comme celle énoncée au paragraphe précédent. Lorsqu'il génère les modèles d'ordre n , il utilise entre autres les modèles déjà générés, dont la taille divise n.

7.4.2 Recherche de tous les modèles de RU

Dans la section 7.3.3, nous nous sommes limité au test de satisfaisabilité du problème RU. Ici, nous allons comparer les méthodes dans le cas où l'on recherche tous les modèles de ce problème. La table 7.25 recense les résultats obtenus.

Les temps de calculs obtenus restent dans le même ordre de grandeur que lors de la recherche de la satisfaisabilité de cette instance. La méthode XLNH trouve énormement de modèles à l'ordre 16. Ceci est du au grand nombre de groupes potentiels que l'on construit. Le même problème se répète a l'ordre 32, où l'on ne trouve pas tous les modèles en moins d'une heure.

Méthode	Taille des domaines							
	6	7	16	19	23	25	28	31
FINDER	(6) 18	(40) 145						
MACE	(24) 4	(120) 78						
SEM	(1) 0.06	(1) 0.1	(1 745) 167	(1) 1366				
CTSEM	(1) 0.04	(1) 0.07	(1 745) 70	(1) 435	(1) 2 279			
VESEM	(1) 0.03	(1) 0.08	(1 745) 100	(1) 751	(1) 1 672	(26) 3 340		
LNHO+	(1) 0.02	(1) 0.05	(1 745) 72	(1) 119	(1) 692	(26) 1 558	(29) 3 490	
LNHO+et DSYM	(1) 0.02	(1) 0.04	(1 139) 60	(1) 95	(1) 494	(26) 1 068	(29) 2 427	
XLNH	(1) 0.01	(1) 0.04	(5 353) 246	(1) 3.5	(1) 8	(26) 18	(13) 32	(1) 34
XLNH et DSYM	(1) 0.02	(1) 0.04	(893) 48	(1) 3	(1) 6	(16) 13	(7) 23	(1) 26

Tableau 7.25. *Temps de résolution lors de la recherche de tous les modèles de* RU

7.4.3 Recherche de tous les modèles de TBA

Nous n'avons pas inclus les résultats concernant la satisfaisabilité du problème TBA. Dans ce cas les différentes stratégies proposées ont un comportement sensiblement identique : un modèle est très rapidement obtenu. Par contre, la recherche de tous les modèles donne, listés dans le tableau 7.26, des temps de calculs très disparates.

Ici l'heuristique LNHO+ favorise la suppression d'isomorphismes. Se consacrant prioritairement à la fonction unaire, elle propage plus de contraintes positives et permet de gérer plus efficacement

Méthode	(nb modèles) Taille des domaines							
	3		4		5		6	
MACE	(6)	0.11	(260)	7.4				
VESEM	(6)	0.01	(130)	0.24	(5235)	30		
CTSEM	(6)	0.01	(130)	0.23	(5235)	25		
SEM	(6)	0.01	(130)	0.19	(5 235)	23		
LNHO+	(6)	0.01	(118)	0.22	(4 066)	21	(242 356)	3 144
LNHO+ et DSYM	(4)	0.01	(87)	0.17	(2 287)	14	(109 365)	1 671
XLNH	(3)	0.01	(41)	0.1	(612)	4.2	(26 170)	443
XLNH et DSYM	(3)	0.01	(41)	0.1	(612)	4.3	(20 280)	367

Tableau 7.26. *Temps de résolution lors de la recherche de tous les modèles de* TBA

le nombre mdn utilisé dans l'heuristique LNH. Elle permet d'ailleurs de résoudre jusqu'à l'ordre 6 l'ensemble des modèles. Ce n'est qu'à partir de l'ordre 6 que l'utilisation de la recherche dynamique des symétries commence à générer moins de modèles que XLNH. De manière générale, plus le nombre d'éléments augmente, plus cette stratégie apparait efficace.

7.4.4 Le challenge du problème LTL

Le challenge associé au problème LTL («logical time-linear logic») est de générer le moins possible de modèles isomorphes, sachant qu'il en existe n pour un domaine de taille n [104]. La figure 7.1 montre la forme de l'interprétation unaire des modèles de la théorie LTL. C'est une forme de «lasso». Il y a un circuit de taille k, puis on a $s(k+1) = 0$, $s(k+2) = k+1$,.... Nous ne considérons bien entendu que les interprétations non isomorphes. Pour n donné, il existe n lassos différents, correspondant aux circuits de taille 1, 2,..., n. Chaque modèle non isomorphe du problème LTL est donc associé à un lasso.

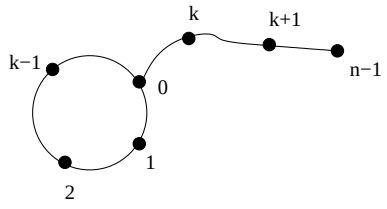


Figure 7.1. *Forme de la fonction unaire dans un modèle de* LTL

Le tableau 7.27 recense les résultats. Comme pour le problème TBA, l'heuristique LNHO+ améliore l'eficacité de la méthode LNH.

Taille	FINDER		MACE		SEM		LNHO+		LNHO+ et DSYM		XLNH	
4	(72)	0.04	(78)	0.01	(23)	0.01	(19)	0.01	(11)	0.01	(4)	0.01
5	(480)	0.6	(504)	0.7	(71)	0.01	(47)	0.01	(19)	0.01	(5)	0.01
6			(3 720)	132	(243)	0.11	(117)	0.05	(42)	0.03	(6)	0.01
7					(988)	1.3	(289)	0.34	(92)	0.15	(7)	0.02
8					(4 950)	23	(724)	5	(205)	2.5	(8)	0.02
9					(30 556)	629	(1 836)	98	(479)	61	(9)	0.05
10									(1 161)	1 955	(10)	0.1
29											(29)	15

Tableau 7.27. *Temps de résolution et nombre de modèles générés pour* LTL

Mais le résultat le plus intéressant est que l'heuristique XLNH produit uniquement les n modèles non isomorphes. Cette méthode génère uniquement les interprétations canoniques en forme de lasso. Ensuite elle ne fait aucune instantiation car le prédicat R, qui est une fonction strictement entrante, est entièrement interprété grâce à s.

Les méthodes MACE et FINDER qui n'utilisent aucune stratégie de suppression d'isomorphismes obtiennent, elles, une grande quantité de modèles isomorphes. Le temps qu'elles mettent à les générer est aussi extrêmement important.

7.5 Conclusion

Les expérimentations que nous avons menées tout au long de ce chapitre n'ont pas fait apparaître une approche qui soit toujours meilleure que les autres. Elles ont en revanche mis en évidence que les divers algorithmes que nous avons introduits dans ce mémoire surplantent assez régulièrement la méthode SEM qui est pourtant considérée comme étant l'une des plus performantes.

Diverses remarques surgissent également à l'issue de ces comparaisons. Tout d'abord, il semble obligatoire de supprimer une grande partie des interprétations isomorphes. Sans cela, le temps de calcul est rapidement rédhibitoire. Ensuite, la transformation de théories en instances SAT de la logique propositionnelle demande très rapidement une grande quantité de mémoire. Même si le nombre de points de choix est sensiblement réduit, le temps demandé par le chargement des instances et par le parcours des listes, donne vite des temps de calculs conséquents.

Nous pouvons également utiliser les résultats obtenus pour donner un cadre d'utilisation des différentes stratégies mises au point dans les chapitres 5 et 6.

L'heuristique LNHO+ Cette heuristique donne d'excellents résultats sans besoin de moyens calculatoires particuliers. Sur certains problèmes, les symboles fonctionnels ne sont pas tous départagés par la fonction `Ordre` qui peut surement être améliorée.

XLNH Cette méthode est utilisable dès qu'une fonction unaire existe dans la théorie. Elle est combinée avec l'heuristique GOT. Dans la grande majorité des cas, cette méthode est extrèmement rapide.

Symétries dynamiques Cette procédure peut être utilisée dans tous les cas. Il est toutefois préférable de la faire collaborer avec les heuristiques LNHO+ et XLNH. Elle permet dans ces deux cas une accéleration. Cette stratégie est pleinement exploitée lors de la recherche de tous les modèles d'une théorie. Plus la taille des domaines est importante, plus cette stratégie est performante, c'est là une caractéristique intéressante.

CTSEM Crée directement depuis SEM, cette méthode est utilisable dès que des clauses réductibles apparaissent dans l'ensemble des axiomes. Son comportement est difficile à cerner, il ne semble pas possible de déterminer pour quelles théories elle doit être mise en oeuvre.

VESEM Cette méthode est toujours utilisable, c'est l'un de ses avantages. Elle obtient souvent des performances proches de CTSEM, et peut certainement être optimisée.

Au vu de ces expérimentations, il semble tout de même que la combinaison de XLNH et de DSYM est la meilleure de toutes les méthodes. La suppression des isomorphismes est donc d'un très grand intérêt lorsque l'on recherche des modèles finis de théories du premier ordre.

Conclusion et perspectives

DANS LA PREMIÈRE PARTIE DE CETTE THÈSE, nous avons essentiellement travaillé sur la recherche d'algorithmes de résolution du problème SAT. Nous avons proposé des algorithmes qui recherchent intelligement les littéraux impliqués par l'ensemble de clauses courant. La combinaison de l'un de ces algorithmes avec la méthode DP donne alors naissance à la méthode AVAL. En propageant le plus de mono–littéraux possibles, cette méthode a montré des qualités de robustesse, qui la rendent peu sensible aux heuristiques. Nous avons mené des expérimentations détaillés lors de la résolution de problèmes aléatoires situés dans la région des problèmes difficiles. Ces expérimentations ont mis en avant deux phases distinctes lors de la résolution de ces instances. La partie haute de l'arbre de recherche contient peu de mono–littéraux «cachés». La majorité des points de choix se situent dans cette région, qui est la partie difficile de la recherche. Dépassée une certaine profondeur de l'arbre de recherche (de l'ordre de $\frac{v}{21}$), nous avons observé un phénomène d'*avalanche* de propagations donnant le nom à la méthode AVAL. C'est la région facile de la recherche qui ne consiste plus qu'en de la propagation unitaire. Ces diverses expérimentations permettent de mieux cerner la difficulté des méthodes énumératives à résoudre de grandes instances aléatoires de l'ordre de 700 variables. Enfin, Les comparaisons menées avec quelques–unes des meilleures méthodes ont donné des résultats satisfaisants.

Dans la deuxième partie de cette thèse, nous nous sommes intéressé à la génération de modèles finis de problèmes mathématiques comme les groupes, les anneaux. Nous avons commencé par essayer de résoudre ces problèmes en utilisant des méthodes de la logique propositionnelle. Nous avons alors rencontré des difficultés. La consommation de mémoire requise pour transformer ces problèmes en instances de la logique propositonnelle est extrèmement importante. Nous avons donc décidé de traiter les problèmes issus de la logique du premier ordre en gardant le formalisme de cette dernière. Le travail de documentation réalisé a alors donné une préférence sur un certain type de générateur qui propage des contraintes. Les contributions apportées sur la génération de modèles portent sur deux aspects différents.

Tout d'abord, nous nous sommes intéressé à la propagation des contraintes. Nous avons mis au point différentes voies pour l'améliorer. Les méthodes CTSEM et VESEM que nous avons proposées

sont des améliorations directes de la méthode SEM. Leur but est de propager le plus possible d'informations négatives (du style $f(1,2) \neq 0$), appelé également contraintes négatives. Pour ce faire, la méthode CTSEM transforme certaines clauses du problème initial et la méthode VESEM utilise un algorithme de filtrage dynamique. Nous avons également introduit des nouvelles heuristiques créées à partir d'un ordre de préférence d'interprétation des symboles fonctionnels. Ces heuristiques utilise la syntaxe de la théorie pour essayer de propager le plus d'informations positives (du style $f(1,2) = 0$).

Nous avons ensuite travaillé sur la détection et la suppression des symétries existant dans la logique du premier ordre. Ici aussi, les observations faites à partir de différentes stratégies de traitement d'isomorphismes nous ont amené vers deux idées différentes mais complémentaires. L'heuristique XLNH qui étend l'heuristique LNH, génére en premier lieu les seules interprétations canoniques d'une fonction unaire. Suite à l'interprétation de cette fonction des symétries sont déterminées sans aucun besoin de calculs.

Partant de la remarque que les symétries dues aux heuristiques LNH et XLNH disparaissent rapidement, la seconde idée consiste à introduire un nouveau cadre plus général d'étude des symétries. Cette étude à été ensuite utilisée pour mettre en place un algorithme (DSYM) de détection et de suppression d'interprétations isomorphes. Nous avons ensuite montré comment combiner les heuristiques LNH et XLNH avec l'algorithme dynamique DSYM.

L'expérimentation de toutes ces stratégies a montré des résultats très encourageants sur une grande variété de problèmes. La combinaison de certaines d'entre elles a donné lieu a des méthodes très efficaces. L'expérimentation a egalement mis en avant les difficultés rencontrées par les méthodes qui ne suppriment aucun isomorphisme (FINDER). De plus, la méthode MACE qui utilise le formalisme de la logique propositionnelle a exhibé les problèmes d'espaces mémoire rencontrés dans ce cas. En effet, bien que les problèmes NP–complets se réduisent polynomialement tous entre eux, Il semble nécessaire d'utiliser le formalisme de base. Non seulement, le passage d'un problème vers un autre peut engendrer des problèmes d'espaces mais, en plus, il fait perdre la structure de départ nécessaire pour mettre au point de nouvelles heuristiques ou algorithmes.

Perspectives

Une des premières perspectives de ce travail est l'amélioration des divers algorithmes proposés dans ce mémoire. Citons par exemple :

- Pour le problème SAT
  - La recherche de conditions afin d'optimiser la découverte de mono–littéraux cachés par l'algorithme RLI.
  - La recherche de nouvelles heuristiques afin de commencer plus tôt l'avalanche de mono–littéraux.

  - D'un point de vue expérimental, il serait également intéressant d'implanter l'algorithme RLI en utilisant la structure de données de SATO. En effet, cette dernière gère efficacement la propagation unitaire.
- Pour les générateurs de modèles finis en logique du premier ordre
  - Rechercher d'autres clauses réductibles pour améliorer CTSEM
  - Mettre au point des conditions pour que l'heuristique UP associée à la méthode VESEM soit plus efficace.
  - Essayer de choisir le terme terminal en fonction de la symétrie détectée par l'algorithme DSYM. Ce choix peut servir afin de supprimer le plus possible de branches isomorphes dans l'arbre de recherche. Ou, inversement, il peut être utilisé pour augmenter la probabilité de trouver une symétrie de l'interprétation courante.
  - Trouver de nouvelles propriétés afin de générer uniquement les interprétations non symétriques d'une fonction unaire quelconque.
  - Trouver de nouvelles conditions pour améliorer l'heuristique sur les ordres, notamment en utilisant la sémantique de la théorie.

Toutes ces améliorations s'inscrivent dans un proche avenir. D'autre objectifs, plus lointains, sont envisagés. Concernant le problème SAT, il serait souhaitable d'étudier plus précisement le phénomène de seuil de production que nous avons observé lors de la résolution d'intances aléatoires. Concernant la génération de modèles finis, il serait très intéressant de mettre au point une méthode que l'on pourrait qualifier d'incrémentale. En effet, lorsque l'on augmente la taille des domaines, on n'utilise jamais les résultats obtenus sur les domaines de taille inférieure et à chaque fois, l'algorithme de recherche a des problèmes «d'amnésie». Afin de résoudre ce problème, il faut mettre au point un algorithme qui, résolvant un problème de taille n, utilise les résultats obtenus avec des tailles plus petites que n. Nous avons remarqué que cela est utilisé par le programme GAP [82] lors de la résolution de structures de groupes. Une autre approche qui semble prometteuse est la mise en oeuvre d'un système propageant des contraintes et qui ne soit pas restreint aux formes clausales du premier ordre. En effet, les problèmes rencontrés lors de la transformation de formalisme apparaissent également lors de la transformation d'une formule du premier ordre en forme clausale. Les fonctions de skolem introduites peuvent grossir la taille de l'espace de recherche. De plus, au vu des expérimentations, il semble que de tels systèmes doivent tenir compte de l'espace mémoire utilisé.

Bibliographie

[1] Dimitris ACHLIOPTAS et Gregory B. SORKIN. «Optimal myopic algorithms for random 3–SAT». proceedings of the 41st Annual Symposium on Foundations of Computer Science, pp. 590–600. IEEE Computer Society Press, 2000.

[2] Gilles AUDEMARD et Belaid BENHAMOU. «Etude des symétries dans les modèles finis». Journées Francophones de la Programmation en Logique et Programmation par Contraintes (JFPLC'2001), pp. 109–122. Hermès Sciences, 2001.

[3] Gilles AUDEMARD et Belaid BENHAMOU. «Symmetry in finite model of first order logic». SymCon'01 – Symmetry in Constraints - CP01 Workshop, 2001. A paraitre.

[4] Gilles AUDEMARD, Belaid BENHAMOU et Laurent HENOCQUE. «Two techniques to improve Finite Model Search». David MCALLESTER, editeur, Proceedings of the 17th International Conference on Automated Deduction (CADE-17), volume 1831 de *LNCS*, pp. 302–308. Springer Verlag, 2000.

[5] Gilles AUDEMARD, Belaid BENHAMOU et Pierre SIEGEL. «La méthode d'avalanche AVAL : une méthode énumérative pour SAT.» Journées Nationales de la Résolution Pratique des problèmes NP-Complets (JNPC), pp. 17–25. 1999.

[6] Gilles AUDEMARD, Belaid BENHAMOU et Pierre SIEGEL. «AVAL : An enumerative method for SAT». Proceedings of first international conference on computational logic CL00, Londres, volume 1861 de *LNCS*, pp. 373–383. Springer Verlag, 2000.

[7] Gilles AUDEMARD et Laurent HENOCQUE. «Sur la génération des groupes finis non isomorphes». Journées Nationales de la Résolution Pratique des problèmes NP-Complets (JNPC) - Marseillle, pp. 57–66. 2000.

[8] Gilles AUDEMARD et Laurent HENOCQUE. «The eXtended Least Number Heuristic». Proceedings of the first International Join Conference in Automated Reasonning (IJCAR), pp. 427–442. Springer Verlag, 2001.

[9] Rolf BACKOFEN et Sebastian WILL. «Excluding Symmetries in Constraint Based Search». 5th Conf Principles and Practices conf constraint programming (CP99), pp. 73–97. 1999.

[10] Peter BAUMGARTNER et Fabio MASSACCI. «The Taming of the (X)OR». Proceedings of first international conference on computational logic CL00, Londres, volume 1861 de *LNCS*, pp. 508–522. Springer Verlag, 2000.

[11] Roberto J. BAYARDO, JR. et Robert C. SCHRAG. «Using CSP Look-Back Techniques to Solve Real-World SAT Instances». Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97), pp. 203–208. AAAI Press, Menlo Park, 1997.

[12] Paul BEAME, Richard KARP, Mike SAKS et Toni PITASSI. «On the Complexity of Unsatisfiability Proofs of Random k-CNF Formulas». Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98), pp. 561–571. ACM Press, New York, 1998.

[13] «International competition and symposium on satisfiability testing, Beijing, China», 1996. URL : http://www.cirl.uoregon.edu/crawford/beijing

[14] Belaid BENHAMOU. *Etude des Symétries et de la Cardinalité en Calcul Propositionnel : Application aux Algorithmes Syntaxiques*. Ph.D. thesis, Université de Provence, 1993.

[15] Belaid BENHAMOU et Laurent HENOCQUE. «A Hybrid Method for finite Model Search in Equational theories.» *Fundamenta Informaticae*, volume 39(1-2) pp. 21–38, 1999.

[16] Belaid BENHAMOU et Lakhdar SAIS. «Tractability through Symmetries in Propositional Calculus». *Journal of Automated Reasoning*, volume 12(1) pp. 89–102, 1994.

[17] Hans Ulrich BESCHE et Bettina EICK. «The groups of order at most 1000 except 512 and 768». *Journal of Symbolic Computation*, volume 27 pp. 405–413, 1999.

[18] Yacine BOUFKHAD. *Aspects probabilistes et algorithmiques du problème de Satisfaisabilité*. Ph.D. thesis, Université de Paris 6, Laboratoire LIP6, 1996.

[19] Thierry BOY DE LA TOUR. «An Optimality Result for Clause Form Translation». *Journal of Symbolic Computation*, volume 14 pp. 283–301, 1992.

[20] Cynthia A. BROWN, Larry FINKELSTEIN et Paul W. PURDOM, JR. «Backtrack Searching in the Presence of Symmetry». *Nordic Journal of Computing*, volume 3(3) pp. 203–219, 1996.

[21] Ricardo CAFERRA et Nicolas PELTIER. «Extending semantic resolution via automated model building :application», 1995.

[22] Jacqueline CHABRIER et Eric NESPOULOS. «Détection consructive de symétries pour les csp entiers». Journées Nationales de la Résolution Pratique des problèmes NP-Complets (JNPC) - Lyon, pp. 47–54. 1999.

[23] Jacqueline CHABRIER, Jean-Michel RICHER et Jean-Jacques CHABRIER. «Resolution of structured SAT problems with Score(FD/B)». Proceedings of the CP96 workshop on constraint programming applications, 1996.

[24] Philippe CHATALIC et Laurent SIMON. «Zres : The Old Davis and Putnam Procedure Meets ZBDD». D. MCALLESTER, editeur, Proceedings of the 17th International Conference on Automated Deduction (CADE-17), volume 1831 de *LNCS*, pp. 449–454. Springer Verlag, 2000.

[25] Philippe CHATALIC et Laurent SIMON. «SATEX : Un cadre de travail dynamique pour l'expérimentaion SAT». Journées Nationales de la Résolution Pratique des problèmes NP-Complets

(JNPC) - Toulouse, pp. 259–270. 2001.
URL : http://www.lri.fr/~simon/satex/satex.php3

[26] Vašek CHVÁTAL et Bruce REED. «Mick Gets Some (The Odds Are on His Side)». Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, pp. 620–627. IEEE Computer Society Press, Pittsburgh, PN, 1992.

[27] Vašek CHVÁTAL et Endre SZEMERÉDI. «Many Hard Examples for Resolution». *Journal of the ACM*, volume 35 pp. 759–768, 1988.

[28] Louis COMTET. *Analyse combinatoire*, volume 1, chapitre 2. Presses Universitaires de France, 1970.

[29] Stephen A. COOK. «The Complexity of Theorem Proving Procedures». Third Annual ACM Symposium On Theory of Computing, 1971.

[30] Thomas CORMEN, Charles LEISERSON et Ronald RIVEST. *Introduction a l'Algorithmique*, chapitre 23, pp. 476–478. Dunod, 1994.

[31] James CRAWFORD. «A theorical Analysis of Reasoning by Symmetry in First Order Logic». Proceedings of Workshop on Tractable Reasonning, AAI92, pp. 17–22. 1992.

[32] James CRAWFORD et Larry D. AUTON. «Experimental Results on the Crossover Point in Random 3-SAT». *AIJ : Artificial Intelligence*, volume 81 pp. 31–57, 1996.

[33] James CRAWFORD, Matthew L. GINSBERG, Eugene LUCK et Amitabha ROY. «Symmetry-Breaking Predicates for Search Problems». KR'96 : Principles of Knowledge Representation and Reasoning, pp. 148–159. Morgan Kaufmann, San Francisco, California, 1996.

[34] Martin DAVIS, George LOGEMANN et Donald LOVELAND. «A machine program for theorem-proving». *Communications of the ACM*, volume 5(7) pp. 394–397, 1962.

[35] Martin DAVIS et Hillary PUTNAM. «A Computing Procedure for quantification theory». *Journal of the ACM*, pp. 201–215, 1960.

[36] Johan DE KLEER. «An Improved Incremental Algorithm for Generating Prime Implicates». William SWARTOUT, editeur, Proceedings of the 10th National Conference on Artificial Intelligence, pp. 780–785. MIT Press, San Jose, CA, 1992.

[37] Martin DELAHAYE. *Outils Logiques pour l'Intelligence Artificielle*. Eyrolles, 1989.

[38] «Second Challenge on Satisfiability Testing organized by the center for Discrete Mathematics and Computer Science of Rutgers University», 1993.
URL : http://dimacs.rutgers.edu/Challenges

[39] H. DREYFUS. *What Computers Can't Do : The Limits on Artificial Intelligence*. Harper & Row, New York, 1979.

[40] Olivier DUBOIS. «Typical Random 3-SAT formulae and the satisfiability threshold». Third Workshop on the Satisfiability problem, SAT 2000, Renesse, Netherlands, 2000.

[41] Olivier DUBOIS, Pierre ANDRÉ, Yacine BOUFKHAD et Jacques CARLIER. «SAT versus UNSAT». *AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, 1996.

[42] Olivier DUBOIS et Gilles DEQUEN. «A backbone-search heuristic for efficient solving of hard 3–SAT formulae». Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI–01), 2001.

[43] John W. FREEMAN. *Improvements to Propositionnal Satisfiability Search Algorithms*. Ph.D. thesis, Univ. of Pennsylvania, Philadelphia, 1995.

[44] John W. FREEMAN. «Hard random 3-SAT problems and the Davis-Putnam procedure». *Artificial Intelligence*, volume 81(2) pp. 183–198, 1996.

[45] Eugene C. FREUDER. «Eliminating Interchangeable Values in Constraint Satisfaction Problems». proceedings of Application of Artificial Intelligence (AAAI-91), pp. 227–233. 1991.

[46] Ehud FRIEDGUT. *Neccessary and Sufficient Conditions for Sharp Threshold of Graphs Properties and kSAT Problems*. Ph.D. thesis, University of Jerusalem, 1991.

[47] Ehud FRIEDGUT et Jean BOURGAIN. «Sharp Thresholds of Graph Properties and the k-Sat Problem». *Journal of the American Mathematical Society*, volume 12 pp. 1017–1054, 1999.

[48] Masayuki FUJITA, John SLANEY et Franck BENNETT. «Automatic Generation of Some Results in Finite Algebra». Proceedings of International Join Conference on Artificial Intelligence, pp. 52–57. Morgan Kaufmann, 1993.

[49] Mickeal GAREY et David S. JOHNSON. *Computers and Intractability : A Guide to the Theory on NP-Completness*. A serie of books in the mathematical sciences, 1979.

[50] Ian P. GENT et Barbara SMITH. «Symmetry breaking in constraint programming». W. HORN, editeur, Proceedings of European Conference on Artificial Intelligence ECAI-2000, pp. 599–603. IOS Press, 2000.

[51] Ian P. GENT et Toby WALSH. «The SAT Phase Transition». Proceedings of the Eleventh European Conference on Artificial Intelligence, pp. 105–109. John Wiley and Sons, Chichester, 1994.

[52] Ian P. GENT et Toby WALSH. «Satisfiability in the Year 2000». *Journal of Automated Reasonning*, volume 24 pp. 1–3, 2000.

[53] Frédéric GILLOT. *Algèbre et Logique. D'après les textes originaux de G. Boole et W.S. Jevons*. Librairie Scientifique Blanchard, Paris, 1962.

[54] Jun GU, Paul W PURDOM, John FRANCO et Benjamin W WAH. «Algorithms for the Satisability : A Survey». *Series in Discrete Mathematics and Theorical Computer Science (DIMACS Workshop)*, volume 35 pp. 19–152, 1996.

[55] John N. HOOKER et V. VINAY. «Branching rules for satisfiability». *Journal of Automated Reasonning*, volume 15 pp. 359–383, 1995.

[56] Jihad JAAM. *Une étude sur les nombres de Ramsey classiques et multiples binaires et ternaires*. Ph.D. thesis, Université d'Aix–Marseille II, 1994.

[57] Robert G. JEROSLOW et Jinchang WANG. «Solving propositional Satisfiability Problems». *Annals of Mathematics and Artificial Intelligence*, volume 1 pp. 167–187, 1990.

[58] David JOSLIN et Amitabha ROY. «Exploiting Symmetry in Lifted CSPs». Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97), pp. 197–202. AAAI Press, 1997.

[59] Tommi A. JUNTTILA et Ilkka NIEMELA. «Towards an Efficient Tableau Method for Boolean Circuit Satisfiability Checking». in proceedings of first international conference on computational logic CL00, Londres, volume 1861 de *LNCS*, pp. 553–567. Springer Verlag, 2000.

[60] A. P. KAMATH, N. K. KARMARKAR, K. G. RAMAKRISHNAN et M. G. C. RESENDE. «Computational experience with an interior point algorithm on the satisfiability problem». *Annals of Operations Research*, volume 25 pp. 43–58, 1990.

[61] Sun KIM et Hantao ZHANG. «ModGen : Theorem Proving by Model Generation». Proceedings of the 12th National Conference on Artificial Intelligence. Volume 1, pp. 162–167. AAAI Press, 1994.

[62] Balakrishnan KRISHNAMURTHY. «Shorts Proofs for Tricky Formulas». *Acta Informatica*, volume 22 pp. 253–275, 1985.

[63] Chu Min LI. «A Constraint-Based Approach Search Trees for Satisfiability». *Information on Processing Letters*, volume 71(2) pp. 75–80, 1999.

[64] Chu Min LI. «Integrating Equivalency reasoning into Davis-Putnam procedure». proceedings of Conference on Artificial Intelligence AAAI, pp. 291–296. AAAI Press, Austin, USA, 2000.

[65] Chu Min LI et ANBULAGAN. «Heuristics Based on Unit Propagation for Satisfiability Problems». Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97), pp. 366–371. 1997.

[66] Chu Min LI et ANBULAGAN. «Look-Ahead Versus Look-Back for Satisfiability Problems». Principles and Practice of Constraint Programming (CP97), volume 1330 de *LNCS*, pp. 341–355. Springer Verlag, 1997.

[67] Chu Min LI et Sylvain GERARD. «On the Limit of Branching Rules for Hard Random Unsatisfiable 3-SAT». Proceedings of European Conference on Artificial Intelligence (ECAI-2000), pp. 98–102. IOS Press, 2000.

[68] Rainer MANTHEY et François BRY. «SATCHMO : A theorem prover implemented in Prolog». E. Lusk ; R. OVERBEEK, editeur, Proceedings on the 9th International Conference on Automated Deduction, volume 310 de *LNCS*, pp. 415–434. Springer Verlag, Berlin, 1988.

[69] Bertrand MAZURE. *De la Satisfaisabilité à la Compilation de Bases de Connaissances Propositionnelles*. Ph.D. thesis, Université d'Artois, 1999.

[70] Bertrand MAZURE, Lakhdar SAÏS et Éric GRÉGOIRE. «Tabu Search for SAT». Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97), pp. 281–285. AAAI Press, 1997.

[71] William MCCUNE. «A Davis-Putnam program and its application to finite first-order model search : quasigroup existence problems». Technical Memorandum ANL/MCS-TM-194, Argonne National Laboratories, IL/USA, 1994.

[72] William MCCUNE. *OTTER 3.0 Reference Manual and Guide*. Argonne National Laboratory/IL, USA, 1994.

[73] Brendan D. MCKAY. «nauty user's guide (version 1.5)». Rapport Technique TR-CS90 -02, Computer Science Department, Australian National University, 1990.

[74] Pedro MESEGUER et Carme TORRAS. «Solving Strategies for Highly Symmetric CSPs». Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), pp. 400–405. Morgan Kaufmann, 1999.

[75] Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG et Sharad MALIK. «Chaff : Engineering an efficient SAT solver». Proceedings of 38th Design Automation Conference (DAC01), 2001.

[76] Nicolas PELTIER. *Nouvelles Techniques pour la Construction de Modèles finis ou infinis en Déduction Automatique*. Ph.D. thesis, Institut National Polytechnique de Grenoble, 1997.

[77] Nicolas PELTIER. «A new method for automated finite model building exploiting failures and symmetries». *Journal of Logic and Computation*, volume 8(4) pp. 511–543, 1998.

[78] Daniele PRETOLANI. *Satisfiability and Hypergraphs*. Ph.D. thesis, Dipartimento di Informatica, Universita di Pisa, 1993.

[79] William V. QUINE. «Methods of logics». *Henry Holt, New York*, 1950.

[80] E. RAMIS, C. DESCHAMPS et J. ODOUX. *Mathématiques Spéciales*, volume 1, Algèbre. Masson, 1985.

[81] Irina RISH et Rina DETCHER. «Resolution versus Search : Two strategies for SAT». *Journal of Automated Reasonning*, volume 24 pp. 225–275, 2000.

[82] Martin SCHONERT. «GAP - Groups, Algorithms and Programming». A Reference Manual for GAP, Version 3.1. Lehrstuhl fur Mathematik, RWTH Aachen, Germany, 1994.

[83] Bart SELMAN, Henry KAUTZ et David MCALLESTER. «Ten Challenges in Propositional Reasoning and Search». Proceedings of the Fifteenth International Joint Conferences on Artificial Intelligence, pp. 50–54. Morgan Kaufmann, 1997.

[84] Bart SELMAN, Hector LEVESQUE et David MITCHELL. «A New Method for Solving Hard Satisfiability Problems». Proceedings of the 10th National Conference on Artificial Intelligence AAAI'94, pp. 440–446. 1994.

[85] C. SHI, A. VANNELLI et J. VLACH. «An improvement on Karmarkar's algorithm for integer programming», 1992.

[86] Olga SHUMSKY, Ralph W. WILKERSON, Fikret ERCAL et William MCCUNE. «Direct Finite First-Order Model Generation with Negative Constraint Propagation Heuristic». Proceedings of the ACM Symposium on Applied Computing, pp. 25–29. 1997.

[87] Pierre SIEGEL. «Représentation et Utilisation de la Connaissance en Calcul Propositionnel», 1987. Thèse d'État. Luminy, Marseille.

[88] Joao P. Marques SILVA et Karem A. SAKALLAH. «GRASP : A New Search Algorithm for Satisfiability». Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 220–227. IEEE Computer Society Press, Washington, 1996.

[89] John SLANEY. «SCOTT : A Model-Guided Theorem Prover». Proceedings of the International Join Conference on Artificial Intelligence (IJCAI 93), volume 1, pp. 109–114. Morgan Kaufmann, 1993.

[90] John SLANEY. «FINDER : Finite Domain Enumerator (System Description)». Alan BUNDY, editeur, 12th International Conference on Automated Deduction, LNAI 814, pp. 798–801. Springer Verlag, Nancy, France, 1994.

[91] John SLANEY, Masayuki FUJITA et Mark STICKEL. «Automated Reasoning and Exhaustive Search : Quasigroup Existence Problems». *Computers and Mathematics with Applications*, volume 29(2) pp. 115–132, 1993.

[92] John SLANEY, Ewing LUSK et William MCCUNE. «SCOTT : Semantically Constrained Otter (System Description)». Alan BUNDY, editeur, 12th International Conference on Automated Deduction, LNAI 814, pp. 764–768. Springer Verlag, Nancy, France, 1994.

[93] John SLANLEY. «FINDER : Finite Domain Enumerator. Version 3 Notes and Guides». Rapport technique, Austrian National University, 1993.

[94] Rok SOSIC, Jun GU et Robert R. JOHNSON. «The Unison algorithm : fast evaluation of Boolean expressions». *Design Automation of Electronic Systems*, volume 1(4) pp. 456–477, 1996.

[95] Mark E. STICKEL et Hantao ZHANG. «Studying Quasigroup Identities by Rewriting Techniques : Problems and First Results». Proceedings of Rewriting Techniques and Applications (RTA95), LNCS 914. 1995.

[96] Christian B. SUTTNER et Geoff SUTCLIFFE. «The TPTP Problem Library - v2.1.0». Rapport Technique JCU-CS-97/8, Department of Computer Science, James Cook University, 1997.
URL : http://www.cs.jcu.edu.au/ftp/pub/techreports/97-8.ps.gz

[97] Alan TURING. «Computing Machinery and Intelligence». *Computers and Thought, E.A*, volume 59 pp. 443–460, 1950.

[98] Steven K. WINKER. «Generation and verification of finite models and counter-examples using an automated theorem prover answering two open questions». *Journal of the ACM*, volume 29(2) pp. 273–284, 1982.

[99] Larry WOS. *Automated Reasonning : 33 basic search problems*. Prentice Hall, New Jersey, 1988.

[100] Hantao ZHANG. «SATO : An Efficient Propositional Prover». William MCCUNE, editeur, Proceedings of the 14th International Conference on Automated deduction, volume 1249 de *LNAI*, pp. 272–275. Springer Verlag, Berlin, 1997.

[101] Hantao ZHANG et Mark STICKEL. «Implemanting the Davis-Putnam method». *Journal of Automated Reasonning*, volume 24 pp. 277–296, 2000.

[102] Jian ZHANG. «Problems on the Generation of Finite Models». Alan BUNDY, editeur, 12th International Conference on Automated Deduction, LNAI 814, pp. 753–757. Springer Verlag, Nancy, France, 1994.

[103] Jian ZHANG. «Constructing Finite Algebras with FALCON». *Journal of Automated Reasoning*, volume 17(1) pp. 1–22, 1996.

[104] Jian ZHANG. «Test Problem and Perl Script for Finite Model Searching». Association of Automated Reasonning, Newsletter 47, 2000.
URL : http://www-unix.mcs.anl.gov/AAR/issueapril00/issueapril%00.html

[105] Jian ZHANG et Hantao ZHANG. «Constraint Propagation in Model Generation». Ugo MONTANARI, editeur, Proceedings of 1st International Conference on Principles and Practice of Constraint Programming (CP95) Marseille, 1995.

[106] Jian ZHANG et Hantao ZHANG. «SEM : a System for Enumerating Models». Chris S. MELLISH, editeur, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 298–303. 1995.

[107] Jian ZHANG et Hantao ZHANG. «Combining Local Search and Backtracking Techniques for Constraint Satisfaction». Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, pp. 369–374. AAAI Press / MIT Press, Menlo Park, 1996.

Résumé

Dans la première partie de cette thèse nous traitons du problème SAT. Nous proposons un algorithme permettant de détecter des mono–littéraux que la procédure de DAVIS et PUTNAM (DP) n'exploite pas. Cet algorithme est combiné avec la procédure DP créant ainsi une méthode que nous avons nommé AVAL. L'étude expérimentale que nous avons menée a mis en avant une avalanche de propagation de mono–littéraux donnant lieu à cette appellation.

Dans la deuxième partie, nous abordons la génération de modèles finis pour les théories multi–types de la logique du premier ordre.

Nous avons essayé d'améliorer la propagation des contraintes de diverses manières. La première utilise un pré–traitement qui modifie la syntaxe de la théorie. La deuxième est un algorithme de type «look-ahead» qui détecte les valeurs de domaines incompatibles et induit une heuristique de type UP. La dernière crée, à partir de la syntaxe de la théorie, un ordre de préférence sur les symboles fonctionnels que nous utilisons pour mettre au point des heuristiques optimisant la propagation.

Les problèmes exprimés sous forme de théories multi-types contiennent souvent un grand nombre de symétries. Il est nécessaire d'en supprimer une partie pour avoir un générateur de modèles finis efficace. Certains utilisent l'heuristique LNH pour supprimer les isomorphismes triviaux. Nous proposons une extension de cette dernière, nommée XLNH, qui ne génère que les interprétations non symétriques d'une fonction unaire. Elle détecte ensuite statiquement certains isomorphismes sans sur-coût de temps. Nous avons ensuite étudié un cadre plus général de la symétrie, nous montrons de nouvelles propriétés concernant les symétries et étudions un algorithme qui permet de les détecter et de les exploiter dynamiquement. Nous montrons, notamment, que les symétries supprimées par les heuristiques LNH et XLNH sont des cas particuliers de ce cadre d'étude.

Mots clés : problème SAT, logique du premier ordre, modèle finis, propagation, symétries.

Abstract

The first part of this thesis deals with the SAT problem expressed in propositional logic. An algorithm which detects mono–literals that the Davis and Putnam (DP) method does not exploit is proposed. The combination of this algorithm with the DP procedure results in the method AVAL. The name AVAL is due to a avalanche of unit clause propagation (UP) observed when experimenting random SAT instances.

The second part consists in finite model generation for many-sorted first order theories. We particularly works on finite model generators which consider such theories as specific constraint satisfaction problems.

First, we tried to improve constraint propagation by three different ways. The first way is a preprocessing that modifies the syntax of the theory. The second way is a look–ahead algorithm which detects incompatible domain values and which induces a UP heuristic. The last way uses the syntax of the theory for creating a preference order on functional symbols; thus, we suggest several heuristics which take into account this order.

Problems expressed in many sorted theories often have a lot of symmetries. Finite model generators must detect and suppress some of them in order to be efficient. Some generators use the LNH heuristic for eliminating trivial isomorphisms. We introduced an extension of this heuristic, named XLNH, which generates only non-isomorphic interpretations of a unary function. The extension XLNH statically detects some isomorphisms without any cost. At last, we sutied a more general notion of symmetry and proved new symmetry properties. We proposed an algorithm that allows to dynamically detect and suppress symmetries. We proved that isomorphisms which are deleted by either LNH or XLNH are particular cases of this symmetry notion.

keywords : SAT problem, finite models, First order logic, Propagation, Symmetry.