Numéro d'ordre 92

UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE

RAPPORT SCIENTIFIQUE
présenté par

Antoine RAUZY

pour obtenir
L'HABILITATION À DIRIGER DES RECHERCHES
SPÉCIALITÉ INFORMATIQUE

Démonstration automatique expérimentale

Soutenance le 7 juin 1996
Composition du Jury :

Président	Robert Cori	Professeur de l'Université Bordeaux I
Rapporteurs	Nicolas Halbwachs	Directeur de Recherche au CNRS
	Gérard Plateau	Professeur de l'Université Paris-Nord
	Mike Robson	Professeur de l'Université Bordeaux I
Examinateurs	André Arnold	Professeur de l'Université Bordeaux I
	Gérard Berry	Directeur de Recherche de l'École des Mines de Paris

gens una sumus

Table des matières

Remerciements

Merci à André Arnold pour m'avoir accueilli dans son équipe et m'avoir fait découvrir la vérification de modèles. Ses convictions sur les tâches scientifiques de l'heure ont grandement contribué à forger et renforcer les miennes. Je lui dois beaucoup.

Merci à Robert Cori d'avoir accepté de présider ce jury et de m'avoir aidé en diverses (et ô combien importantes) occasions. Je sais ce que je lui dois et je lui en suis reconnaissant.

Merci à Nicolas Halbwachs, à Gérard Plateau et à Mike Robson d'avoir accepté d'être rapporteurs de ce mémoire.

Merci à Gérard Berry d'avoir accepté de participer à ce jury.

Merci à Pierre Siegel pour avoir encadré ma thèse de doctorat, pour m'avoir proposé d'animer ensemble le projet inter-PRC "Classes Polynomiales", pour m'avoir fait une petite place au LIM depuis bientôt trois ans et pour mille autres choses encore. Au delà de Pierre, merci à toute l'équipe logique du LIM et aux participants aux groupes de travail du PRC-IA pour ce que nous avons vécu ensemble. Parmi eux, je voudrais tout spécialement saluer Laurent Oxusoff et Richard Génisson.

Merci aux membres de la petite équipe programmation logique du LaBRI, Michel Billaud, Marc-Michel Corsini et Kaninda Musumbu. Ils sont au début de cette histoire. Marc-Michel surtout, avec qui j'ai longuement travaillé.

Merci aux membres de l'équipe MTVSI du LaBRI et en particulier Didier Bégay et Alain Griffault. Nous avons, avec Didier, partagé beaucoup de choses, entre autres un bureau, un appartement, une passion commune pour cette idée qu'il faut appliquer les théories – c'est à dire leur faire subir le feu de l'expérience industrielle – et de nombreuses discussions où nous parlions du monde comme il va. Merci pour tout cela.

Merci à Yves Dutuit pour m'avoir fait découvrir la sûreté de fonctionnement et pour notre collaboration dans le cadre du projet Aralia.

Merci à Jean-Marc Boï, Odile Papini, Baudoin le Charlier et Srecko Brlek.

Merci enfin à tous ceux qui ont créé et font vivre le LaBRI. Ce laboratoire est vraiment un endroit où il fait bon travailler et où j'ai beaucoup appris et beaucoup reçu.

En Tête

Le présent document se veut une synthèse des activités de recherche que j'ai menées depuis janvier 1988 jusqu'à cette fin d'année 1995.

Traditionnellement, avant de commencer un rapport sur un travail de quelque importance, on se doit de préciser la démarche suivie, de donner au lecteur quelques garanties méthodologiques qui le rassure sur le bien-fondé scientifique et l'unité de ce qui suit. Cependant, ces huit années, passées au Groupe d'Intelligence Artificielle de Marseille puis au Laboratoire Bordelais de Recherches en Informatique, sur des postes d'enseignant-chercheur puis sur un poste de chercheur à temps plein, ont été aussi, pour moi, celles de la formation méthodologique. Plutôt donc que le récit linéaire d'une démarche fixée *a priori*, ce rapport est un instantané de la vision rétrospective que j'ai aujourd'hui de mes travaux. Comme tout instantané, il renvoie les certitudes et les doutes du moment.

S'il m'est impossible d'ordonner mes recherches autour d'un axe technique unique, je peux en revanche en préciser sans difficulté le paradigme, au sens donné par T. Kuhn à ce mot. Il se résume ainsi :

≪ Explorer les formalismes logiques et les algorithmes de résolution associés permettant d'exprimer des problèmes de nature symbolique et de les résoudre automatiquement en des temps raisonnables ≫.

Rentrent dans ce schéma mes travaux autour du problème SAT, ceux sur l'extraction et la révision de connaissances, ceux sur la réalisation de résolveurs pour les langages de programmation logique avec contraintes, ceux sur les divers applications des diagrammes binaires de décision, ceux, enfin, sur l'analyse de programme logiques et de systèmes de processus communicants. Tous comprennent une phase importante d'expérimentation, c'est à dire de programmation. Par goût, si ce n'est par raison, j'ai fait mien l'aphorisme prêté au grand Lénine : ≪ la théorie n'est pas un dogme mais un guide pour l'action ≫.

La démonstration automatique expérimentale, si l'on peut regrouper les thèmes évoqués ci-dessus sous cette appellation, est une discipline récente. En effet, les chercheurs ne disposent finalement que depuis peu de temps de machines assez puissantes pour traiter des problèmes de tailles intéressantes et en traiter suffisamment pour avoir des bases statistiques sérieuses. En témoigne, par exemple, la mise en évidence de phénomènes de seuils dans le problème SAT. Que le lecteur veuille donc bien me pardonner l'aspect parfois un peu prosélyte de ce qui suit : j'aimerais que ce document soit une défense et illustration de cette démarche mi-théorique mi-expérimentale et qui fait une place importante à la qualité de la programmation. Mais bien sûr, comme le dit Qohèlèt :

Habèl ha balîm hakol habèl.

Marseille, le 25 octobre 1995.

Synopsis

Le corps de ce document est divisé en trois chapitres et une annexe composée de plusieurs articles les illustrant. Ce découpage répond à mes préoccupations actuelles : un chapitre pour le problème SAT, un chapitre pour le langage Toupie et finalement un chapitre pour les applications des techniques de manipulation de fonctions booléennes et notamment leur mise en œuvre dans le cadre de l'analyse de fiabilité de systèmes industriels. L'avantage de cette organisation est de permettre d'exposer mes travaux récents. Son inconvénient est de ne pas laisser de place à un certain nombre d'autres résultats, principalement liés à la représentation de connaissances [408, 407, 380, 379, 381, 382, 383]. À vrai dire, ces derniers occupent une faible place dans mon activité et, pour des raisons d'équilibre, j'ai préféré ne pas les présenter ici.

L'objectif de ce document est double : d'une part exposer les résultats que j'ai obtenu, d'autre part retracer mon itinéraire scientifique. Chacun des trois chapitres contient donc des indications biographiques. Pour autant que je puisse en juger, il y a deux types de mémoires d'habilitation à diriger des recherches : ceux qui sont des sommes scientifiques et ceux qui exposent une démarche. J'ai opté pour la deuxième approche qui correspond mieux, à mon sens, à l'esprit de l'habilitation. D'où les notes biographiques.

Les trois chapitres sont de tailles inégales. Pour plusieurs raisons. D'une part, je n'ai pas consacré la même énergie à chacun de ces trois thèmes. D'autre part, les articles [133, 141] joints en annexe donnent une bonne vision du projet Toupie et j'ai rédigé par ailleurs un document technique assez complet sur l'utilisation des diagrammes binaires de décision dans le cadre de la sûreté de fonctionnement [423]. Les chapitres correspondants sont donc volontairement réduits, afin d'éviter trop de redondance. En revanche, je n'ai publié à ce jour que peu d'articles sur le problème SAT alors qu'il a occupé, et occupe toujours, une part majeure de mon activité de recherche. Une remarque s'impose ici : le problème SAT est un des sujets les plus difficiles – et sans aucun doute l'un des plus intéressants, parce que l'un des plus fondamentaux – que j'ai rencontrés en informatique. Le formalisme utilisé est simplissime, les connaissances nécessaires pour l'aborder sont relativement réduites, mais tout progrès est difficile, notamment parce que l'intuition est très mauvaise conseillère et que toute idée doit être soigneusement vérifiée. De plus, il n'existait pas, au moins en France, de communauté active sur ce thème, ni de corpus bibliographique constitué. Il me semble qu'une des tâches de l'heure est de faire le point sur le problème SAT au travers d'un document de synthèse. Les articles joints en annexe ne donnent cependant qu'une vision très incomplète de mes travaux (et bien entendu une vision totalement parcellaire de ce qui se fait en général sur le sujet). J'ai donc essayé d'écrire un premier chapitre aussi complet que possible, en particulier sur les référence bibliographiques, sans toutefois trop détailler l'exposé, ce qui aurait dépassé le cadre du présent exercice.

Le chapitre sur le langage de contraintes Toupie expose la philosophie de l'outil plus qu'il ne présente l'outil lui-même. Toupie se situe à la confluence de deux courants de pensée : la modélisation/vérification de systèmes informatisés d'une part, la programmation (logique) avec contraintes d'autre part. Il m'a semblé plus intéressant de justifier les raisons de ce mariage que d'entrer dans les détails techniques propres à Toupie (qui se trouvent de toutes

façons dans les articles que j'ai publié sur la question).

Le troisième chapitre contient une première partie sur la mise en œuvre de résolveurs de contraintes booléennes dans les langages de programmation logiques avec contraintes. Cette partie aurait tout aussi bien pu être intégrée dans le chapitre sur le problème SAT ou même dans celui sur Toupie, qui est aussi un langage de programmation par contraintes. J'ai finalement choisi de le placer avec la présentation de mes travaux sur la sûreté de fonctionnement parce qu'il me semble que l'adéquation d'un résolveur se juge à l'aune de la plus ou moins grande facilité avec laquelle il permet d'implémenter les algorithmes spécifiques à la résolution de tel ou tel problème pratique particulier.

La seconde section du troisième chapitre est donc consacrée au projet Aralia, c'est à dire à l'utilisation des diagrammes binaires de décision dans le cadre de la sûreté de fonctionnement. Comme la précédente, cette section traite de problèmes dans lesquels l'ingénierie joue un rôle fondamental. Mon intérêt pour la sûreté de fonctionnement est relativement récent. Cependant, il me semble que c'est un thème fort pour les années à venir. D'où la place accordée à ces questions dans ce mémoire.

Les trois chapitres sont largement indépendants les uns des autres et peuvent donc être lus dans n'importe quel ordre.

Voici quelques indications sur les articles joints en annexe :

- L'article [373] donné annexe B.1 a été publié récemment mais est en fait issu de la troisième partie de ma thèse. Il illustre assez bien mes premiers travaux sur le problème SAT. C'est pourquoi je l'ai intégré à ce document.
- L'article [421] donné annexe B.2 est représentatif de mes recherches dans le cadre du projet inter-PRC «Classes Polynomiales». En particulier, il décrit en détails la procédure de Davis et Putnam (qui est au cœur de tous mes travaux sur SAT), ce qui peut être utile au lecteur non familier avec le domaine.
- Comme je l'ai dit plus haut, les articles [133] (annexe C.1) et [141] (annexe C.2) présentent mes travaux sur Toupie. Le second, contenant une description assez complète de l'outil, peut être lu comme une introduction au sujet.
- L'article [414] (annexe D.1) témoigne des propositions que j'ai faites sur l'implémentation de résolveurs de contraintes booléennes pour les langages de programmation logiques avec contraintes.
- Enfin, l'article [410] (annexe D.2) est le premier article que j'ai publié sur l'utilisation des diagrammes binaires de décision dans le cadre de la sûreté de fonctionnement. Il introduit cette technique et peut donc lui aussi être utile au lecteur non familier avec le sujet.

Pour finir, j'aimerais revenir sur la méthode adoptée pour écrire ce mémoire. Le lecteur pourra être surpris, peut-être même choqué, d'y trouver un certain nombre d'affirmations non formellement démontrées, à proprement parler, de prises de position. Disons le tout net, je ne crois pas beaucoup à la fameuse méthode hypothético-déductive, au moins pour ce qui concerne la production et la reproduction au quotidien de la science. Je ne suis jamais parti d'hypothèses que j'aurai cherché à valider ou à infirmer par le raisonnement ou l'expérimentation. Je crois au contraire le paradigme «démonstration automatique expérimentale» fait de thèses plus ou moins solides et qui entretiennent un rapport dia-

lectique avec l'expérimentation[1]. Par conséquent, ce mémoire aurait perdu tout intérêt scientifique et toute saveur à ne contenir que des faits avérés. Autrement dit, j'ai préféré, en y décrivant les moteurs de mon activité de recherche, prendre le risque de me tromper et d'être en désaccord avec le lecteur, que me résoudre à un discours purement technique.

On écrit beaucoup pour soi et un peu pour les autres. Mon seul souhait est que ce document soit utile, de quelque façon que ce soit, au lecteur.

[1] J'entends par là que seul ce rapport existe : thèses et expérimentations ne sont pas séparables.

Chapitre 1

Le problème SAT

Ce chapitre présente les travaux que j'ai effectués autour du problème SAT. Il est organisé comme suit. La section 1.1 contient une brève introduction au problème et rappelle quelques résultats importants. Elle précise l'environnement dans lequel les recherches présentées dans les deux sections suivantes ont été effectuées. La section 1.2 regroupe un certain nombre de travaux, pour la plupart assez anciens, que j'ai réalisé dans le cadre des groupes ≪DétAut≫[1] [180] et ≪Bahia≫[2] [21, 22] du PRC-IA[3]. Finalement, la section 1.3 fait état de résultats relativement récents que j'ai obtenu sur les classes polynomiales du problème SAT et des problèmes de satisfaction de contraintes (dans le cadre du projet inter-PRC ≪Classes Polynomiales≫[4] [390]).

1.1 Une brève introduction au problème SAT

1.1.1 Le problème

Soit $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ une expression booléenne sous forme normale conjonctive, c'est à dire où chaque *clause* C_i est une disjonction de *littéraux* et chaque littéral est soit une variable x_i soit sa négation $\neg x_i$ $(1 \leq i \leq n)$[5]. Le problème SAT consiste à déterminer si une telle expression ϕ est vraie pour une certaine affectation de valeurs booléennes aux variables $x_1, \ldots, x_n$. ϕ est dite *satisfiable*[6] si une telle affectation existe et *insatisfiable* sinon.

[1] Déduction Automatique.

[2] Booléens : Algorithmes et Heuristiques pour l'Intelligence Artificielle.

[3] Programme de Recherches Coordonnées ≪Intelligence Artificielle≫.

[4] J'étais co-responsable de ce projet avec P. Siegel. Je suis actuellement co-responsable avec M.-C. Vilarem du groupe du PRC-IA ≪Ressac≫qui fait suite aux groupes Bahia et CSP-Flex et au projet Classes Polynomiales.

[5] Dans la suite de ce chapitre, je garderai ces notations : n représentera le nombre de variables, m le nombre de clauses et $|\phi|$ le nombre de littéraux apparaissant dans les instances SAT considérées.

[6] Une discussion fréquente entre chercheurs francophones consiste à juger des mérites respectifs de *satisfiable* et *satisfaisable*. Aucun des deux mots ne figurant dans les dictionnaires courants, j'ai choisi l'orthographe anglaise. *If you can't beat them, join them.*

SAT est le problème NP-complet de référence [223, 378]. Pour plusieurs raisons. D'abord, c'est le premier problème a avoir été démontré complet pour la classe NP, par S.A. Cook en 1971 [126]. Ensuite, le calcul propositionnel permet d'exprimer relativement facilement d'autres problèmes de décision. C'est bien entendu ce que capture la notion de réduction entre problèmes qui est au cœur de la définition de complétude pour les classes de complexité [293]. Mais au delà, on retrouve le problème SAT en tant que sous-problème ou cas particulier dans de nombreux domaines, de la démonstration automatique à la preuve de circuits en passant par l'optimisation combinatoire et la planification. Cela étant, la majeure partie des travaux sur SAT (y compris les miens) étudient le problème pour lui-même et non en vue d'une application immédiate à la résolution de tel ou tel problème ≪pratique≫ particulier. En un mot, SAT est avant tout un artefact. Schématiquement, les principaux axes de recherches sont les suivants :

- L'analyse de complexité (au pire, en moyenne) des algorithmes de résolution.
- La mise au point de démonstrateurs efficaces (dans le cas général, pour certaines classes de formules) et l'amélioration pratique des démonstrateurs existants.
- La recherche de restrictions polynomiales, c'est à dire de classes d'instances pour lesquelles on a des algorithmes de reconnaissance (c.à.d. de test d'appartenance à la classe) et de décision (c.à.d. de test de la satisfiabilité) polynomiaux.
- La compréhension de ce qui fait la difficulté du problème, ou, autrement dit, la recherche de la frontière entre instances faciles et instances difficiles.

J'ai principalement travaillé sur les aspects algorithmiques de la résolution de SAT. La présentation qui suit est donc essentiellement orientée dans cette direction. En particulier, il sera souvent fait état de la procédure de Davis et Putnam [172][7]. Cette procédure est l'archétype des méthodes énumératives, c'est à dire des méthodes dont le principe est de séparer le traitement d'une instance ϕ en celui de $\phi_{x\leftarrow 0}$ et $\phi_{x\leftarrow 1}$[8], pour une certaine variable x, ces affectations s'accompagnant en général de simplifications de complexité polynomiale[9] (voir l'article [421] donné annexe B.2 pour une présentation plus détaillée de cet algorithme). Comme nous le verrons, munie de bonnes heuristiques de branchement, la

[7] Cette référence fait souvent problème puisque H. Putnam, philosophe de sciences de son état, n'en est pas cosignataire. En fait, une première version de l'algorithme a bien été donnée par M. Davis et H. Putnam dans [173]. Cette première version n'est toutefois pas l'algorithme énumératif désigné sous le nom de ≪procédure de Davis et Putnam≫qui a été proposé lui dans [172] (voir aussi la présentation qu'en donne D. Loveland dans son livre [326]). Pour couronner cette sombre histoire, V. Chvátal and E. Szemerédi, dans [107], signalent que cette fameuse procédure a été proposée, au début du siécle, par L. Löwenheim dans une série d'articles [327, 328, 329].

[8] $\phi_{x\leftarrow v}$ dénote l'instance ϕ dans laquelle la variable x a été substituée par la valeur booléenne v. Par abus, on note souvent ϕ_p, où p est un littéral, l'affectation satisfaisant p dans ϕ. ϕ_p est équivalent à l'ensemble de clauses ϕ dans lequel on a supprimé les clauses contenant p et supprimé des autres clauses les occurrences de l'opposé de p.

[9] La procédure de Davis et Putnam procède à deux simplications : propagation des *littéraux purs*, c'est à dire des littéraux dont l'opposé n'apparaît pas dans l'instance, et propagation des *littéraux unitaires*, c'est à dire apparaissant dans une clause de longueur de 1. Si p est un litéral pur ou unitaire dans ϕ, alors ϕ et ϕ_p sont équivalents du point de vue de la satisfiabilité. On peut donc répéter l'opération consistant à satisfaire ces littéraux jusqu'à ce qu'il n'y en ait plus ou qu'une clause vide apparaisse. D'où le terme de propagation.

procédure de Davis et Putnam est l'une des méthodes les plus efficaces en pratique.

1.1.2 Digression

Dans la section précédente, j'ai présenté le problème SAT comme le problème NP-complet de référence et j'ai esquissé les raisons de s'y intéresser *per se*. De nombreux manuels l'introduisent autrement, à savoir sous l'angle «démonstration automatique», au prétexte que prouver qu'une théorie Θ implique une formule ϕ revient à montrer que $\Theta \wedge \neg\phi$ est insatisfiable. Le calcul propositionnel en général y est vu comme une version dégénérée du calcul des prédicats, dont l'intérêt principal, pour ne pas dire unique, est d'être décidable. Cette approche me semble discutable. Pour plusieurs raisons. D'abord, elle ignore les applications pratiques du calcul propositionnel dont j'ai essayé de montrer quelques aspects dans ce mémoire. Ensuite, elle laisse par trop de coté la question de l'effectivité des démonstrateurs (un ensemble d'axiomes et de règles d'inférence ne constitue pas un algorithme, entre autres parce que ni les heuristiques, ni les structures de données ne sont précisées)[10]. Enfin, elle rend complexes des choses simples, obligeant en particulier à des preuves par induction délicates (comme celles de complétude des méthodes utilisant le principe de résolution).

La procédure de Davis et Putnam et les diagrammes binaires de décision, dont il sera beaucoup question dans ce mémoire, rendent justice de cette complication. Ils prennent les variables booléennes pour ce qu'elles sont : soit vraies, soit fausses. Et du coup, ils sont beaucoup plus efficaces pour résoudre en pratique les problèmes que tous les systèmes d'axiomes et de règles d'inférences possibles et imaginables.

Pour qui programme, simple et beau sont synonymes. C'est donc armé du rasoir d'Ockham que j'étudie le calcul propositionnel : comment obtenir de bons résultats tant pratiques que théoriques avec des moyens simples. Moyens simples, esprit simples, le royaume de dieu leur est paraît-il ouvert.

Fin de la digression et retour au problème SAT.

1.1.3 Résultats de complexité

Pour autant donc que P soit différent de NP, on ne connaît pas d'algorithme efficace (polynomial) pour résoudre SAT. Quelques restrictions polynomiales de SAT ont été exhibées. Les plus connues sont Horn-SAT (ensembles de clauses ayant au plus un littéral positif par clause) et 2-SAT (k-SAT désigne les instances composées de clauses ayant toutes k littéraux). Ces restrictions font l'objet d'une discussion assez détaillée section 1.3. S.A. Cook, dans [126], a montré que 3-SAT est NP-complet. En fait, toute instance SAT se ramène facilement à une instance 3-SAT équivalente du point de vue de la satisfiabilité. De même, toute formule propositionnelle peut être transformée en un ensemble de clauses équivalent et dont la taille est linéaire en la taille de la formule initiale [388, 509]. Ces deux réductions nécessitent toutefois l'ajout de nouvelles variables.

[10] La démonstration automatique et le centralisme démocratique ont à mon sens ceci en commun qu'il faut prendre garde à ce que le substantif ne l'emporte pas sur l'adjectif.

Comme il y a peu d'espoir de trouver un algorithme polynomial pour résoudre SAT en général, on a cherché à déterminer la complexité au pire et en moyenne des principaux algorithmes connus :

Résultats obtenus à l'aide d'instances structurées

Les principaux résultats de complexité au pire ont été obtenus pour les algorithmes utilisant le principe de résolution de Robinson [427][11]. Un arbre de résolution est un arbre binaire dont la racine est la clause vide, dont tout nœud intérieur est étiqueté par une clause obtenue par résolution des clauses étiquetant ses fils et dont les feuilles sont étiquetées par des clauses de l'instance considérée. La taille d'un arbre de résolution (d'une preuve) est par définition le nombre de clauses différentes étiquetant ses nœuds. G.S. Tseitin, dans [480] a introduit la résolution normale[12] et a prouvé qu'il existe des familles d'instances pour lesquelles toute preuve par résolution normale contient au moins $\mathcal{O}(2^{\sqrt{n/32}})$ clauses. Comme l'ont remarqué S.A. Cook et R.A. Reckhow [128], les procédures énumératives peuvent être vues comme des versions déterministes de la résolution normale et donc ce qui vaut pour cette dernière vaut aussi par exemple pour la procédure de Davis et Putnam sous ses deux présentations [173] et [172]. Z. Galil, dans [218] a «amélioré» ce résultat en donnant une borne inférieure de la taille des preuves de 2^{cn}, pour une certaine constante $c > 1$. En 1985, A. Haken [245] a généralisé cette borne inférieure à la résolution sans restriction, en utilisant une formule codant le problème des tiroirs et des chaussettes[13]. Plusieurs auteurs ont ensuite amélioré ces résultats (voir par exemple [483, 87, 125]).

Tous ces travaux s'appuient des instances SAT ayant des structures bien particulières (c'est le cas aussi d'autres articles comme [306]). Or, ces instances peuvent être traitées relativement facilement en ajoutant un peu de puissance aux algorithmes. C'est en particulier le cas pour le problème des tiroirs et des chaussettes qui peut clairement être résolu efficacement en utilisant des systèmes de Frege étendus(qui permettent de d'utiliser des lemmes) ou en tirant parti des symétries (voir par exemple [127, 305, 44, 29]).

Résultats obtenus sur les instances générées aléatoirement

Une autre façon d'aborder le problème est de considérer des instances générées aléatoirement et qui ont donc peu de chances de posséder des régularités exploitables. Deux modèles de génération aléatoire ont été principalement étudiés :

Le modèle dit «à densité fixée» **:** Les clauses sont générées indépendamment. Pour chaque clause C et chaque variable x, le littéral x est présent dans C avec une probabilité p et le littéral $\neg x$ est présent avec une probabilité q (on suppose souvent sans perte

[11] Ce principe consiste à produire à partir de deux clauses $p \vee C$ et $\neg p \vee D$ de l'instance considérée ϕ la clause impliquée $C \vee D$.

[12] Regular Resolution. Un arbre de résolution est dit normal si dans aucun des chemins de la racine aux feuilles on effectue deux résolutions sur la même variable.

[13] En anglais, Pigeon-Hole. Ce problème consiste à essayer de placer N paires de chaussettes dans M tiroirs de façon à ce qu'il y ait au plus une paire de chaussette par tiroir.

de généralité que $p = q$). Et donc x est absente de la clause avec une probabilité $1 - p - q$.

Le modèle dit «à longueur constante» : Les clauses sont générées indépendamment. Pour chaque clause C, on tire k variables parmi les n considérées (qui sont équiprobables), puis pour chaque variable on tire une parité (les deux parités étant équiprobables).

Ce type d'instances a deux intérêts : d'une part il fournit un banc d'essais inépuisable et dimensionnable à volonté, d'autre part le modèle de longueur constante capture la NP-complétude de SAT. Ce résultat, qui est à mon sens le plus important sur SAT, est dû à V. Chvátal et E. Szemerédi [107]. Dans cet article ils ont montré que :

1. Pour tout $k \geq 3$, au delà d'un certain rapport m/n la probabilité de tirer une instance k-SAT satisfiable tend vers 0 quand n tend vers l'infini[14].
2. Pour un rapport m/n donné, le nombre moyen de d'étapes nécessaires pour prouver par résolution qu'une instance est insatisfiable croît exponentiellement avec n.

Dont acte.

Pour les instances 3-SAT, E. Humbert a donné récemment (dans sa thèse [275]) pour les algorithmes énumératifs, de type procédure de Davis et Putnam, une borne inférieure de la complexité au pire de $1,562^n$ qui améliore les résultats précédents [365, 438].

Le modèle à densité fixée a fait l'objet d'un certain nombre de travaux (voir par exemple [241, 242, 398, 400, 83]) d'où il ressort que les instances générées sont en général faciles sauf si la probabilité p est très petite par rapport à n (ce qui les rapproche, au moins en pratique, de celles générées avec le modèle à longueur constante).

Phénomènes de seuils

Les instances k-SAT générées avec le modèle à longueur constante ont attiré récemment une attention considérable pour une autre raison : on a découvert qu'elles semblent obéir à une loi 0/1[15] : pour chaque valeur de k, il existe une valeur du rapport n/m en deçà de laquelle la probabilité de tirer une instance satisfiable tend vers 1 quand n augmente et au delà de laquelle cette probabilité tend vers 0. La région de transition de phase se réduit rapidement avec n. Ce phénomène remarquable a été observé par plusieurs auteurs indépendamment [104, 192, 361, 312, 162]. Pour $k = 2$, la valeur 1 du seuil a été établie analytiquement, là encore par plusieurs auteurs indépendamment [106, 239]. Pour $k = 3$ et $k = 4$ des valeurs ont été observée expérimentalement (respectivement 4.25 pour $k = 3$ et 9.8 pour $k = 4$). Des bornes inférieures et supérieures ont été trouvées (voir [99, 106, 74, 211] pour les premières et [107, 292] pour les secondes). Des phénomènes de seuils apparaissent aussi pour des instances ayant des clauses de différentes longueurs (en proportions fixées) [229].

[14] En fait, ce résultat est facile à montrer : la probabilité qu'une affectation satisfasse une clause est $1 - 2^{-k}$. Donc la probabilité qu'elle satisfasse les m clauses est $(1 - 2^{-k})^m$. Par conséquent la probabilité qu'au moins une des 2^n affectations satisfasse l'instance est majorée par $(2(1 - 2^{-k})^\rho)^n$, où $\rho = m/n$. Il est facile de voir qu'il existe toujours un ρ tel que $(1 - 2^{-k})^\rho < 1/2$. D'où le résultat.

[15] Voir [464] pour un exposé sur ces lois.

En outre, on observe, pour toutes les méthodes de résolution testées, un pic de difficulté dans la région de transition de phase. Autour du seuil, il est très clair que les temps de calcul, par exemple de la procédure de Davis et Putnam, croissent exponentiellement avec n (voir entre autres [361, 191, 390]). À gauche du seuil, les instances sont faciles (voir en particulier [74]). À droite du seuil, leur difficulté est nettement moindre qu'au seuil, même si elle reste relativement importante.

De nombreux travaux portent actuellement sur la mise en évidence de phénomènes de seuils, de pics de difficulté dans la résolution dans différents problèmes NP-complets et sur les liens de ces phénomènes avec ceux observés en physique statistique – par exemple, les phénomènes de percolation [467], ou d'orientation des verres de spins [352] – [229, 445, 163, 443]. Ces questions ont reçues un large écho, y compris dans la presse ≪grand public≫ comme en témoigne des articles récents dans Science [298] et Pour la science [181].

1.1.4 Les algorithmes de résolution

Vers une taxinomie des méthodes

De nombreux algorithmes ont été proposés pour résoudre SAT. Parmi les grands anciens, on peut citer les règles de récriture de G. Boole (en 1854 !) [58] et de C.E. Shannon [451], la procédure de W.V. Quine [403], les procédures de M. Davis et H. Putnam [173, 172], la résolution de J.A. Robinson [427], et les tableaux analytiques de R.M. Smullyan [463]. Il serait présomptueux, vue l'étendue de la littérature, de prétendre à un catalogue complet des divers algorithmes ou principes d'algorithmes proposés à ce jour. On peut toutefois se risquer à une taxinomie de ces méthodes et quelques remarques issues de l'expérience[16]. Je serais donc tenté de répartir les méthodes pour résoudre SAT comme suit :

- Les méthodes de récriture qui cherchent à mettre la formule considérée sous une forme normale dans laquelle sa satisfiabilité peut être directement vérifiée. Historiquement, ce sont les premières à avoir été proposées [58, 451]. Dans cette catégorie, on trouve les tableaux analytiques de Smullyan déjà cités [463, 497], les algorithmes de mises sous forme de Reed-Muller et d'unification booléenne [434, 505, 351, 273, 82, 64, 342, 343] et certains algorithmes mis en œuvre dans le cadre de la preuve de circuits logiques (voir par exemple [501]).
- Les méthodes utilisant le principe de résolution [427] et ses diverses extensions (le livre de D. Loveland [326] décline de nombreux algorithmes utilisant ce principe, voir aussi [507, 62, 178]). La plus remarquable de ces méthodes est peut-être la SL-Resolution de R. Kowalski and D. Kuehner [302, 455, 168] dont a été dérivée la SLD-Resolution [8] qui n'est rien d'autre que le moteur d'inférence de Prolog [122].
- Les méthodes énumératives, au premier rang desquelles se trouve la procédure de Davis et Putnam [172, 326], et les nombreuses variations/améliorations et heuristiques associées [49, 45, 29, 162, 163, 91, 190, 191, 220, 222, 257, 269, 271, 287, 289, 311, 331, 365, 372, 401, 399, 512]. Dans cette catégorie, on peut aussi ranger les matrices

[16] Un des mes travaux dans le cadre du groupe Bahia du PRC-IA a justement été de suggérer une première ébauche de classement [21].

d'interconnection de Bibel [43] et les extensions de la procédure de Davis et Putnam aux formules non clausales [364, 33, 38, 114, 169, 487, 344].
- Les méthodes de programmation linéaire en 0/1, sur lesquelles il existe une littérature abondante (citons le travail pionnier de C.E. Blair, R.G. Jeroslow et J.K. Lowe [49], mais aussi les travaux de J.N. Hooker [267, 266, 270, 268] et de G. Plateau et H. Bennaceur [35, 36]).
- Les méthodes de réparation locale parmi lesquelles on trouve les algorithmes de type recuit simulé [297, 37], tabou [236, 237], les algorithmes génétiques [176, 254], les réseaux de neurones [291] et autres algorithmes essayant d'améliorer une solution courante [243, 450, 447, 448] ...Ces méthodes sont par nature incomplètes. Cependant, des travaux récents (entre autres de B. Selman [450]) ont montré qu'elles permettent de résoudre des instances intraitables pour toutes les autres méthodes connues (cette question est discutée plus en détails section 1.1.4).
- Les diagrammes binaires de décision (BDD). Introduits par B. Akers [5], ils ont été développés par R. Bryant [77, 70, 78]. De fait, on pourrait les ranger parmi les méthodes de récriture puisque leur principe consiste à mettre la formule considérée sous forme normale (de Shannon [451]). Cependant, ils méritent d'être traités à part, vu les succès qu'ils ont remportés ces dernières années (dans d'autres cadres que SAT).

Il est difficile de mettre de l'ordre dans cet inventaire à la Prévert (on a même essayé de résoudre SAT à l'aide de solutions d'ADN [323] !). D'autant que certaines des méthodes évoquées ci-dessus ne sont pas à proprement parler des algorithmes mais plutôt des règles d'inférences (c'est en particulier le cas pour les méthodes utilisant des généralisations du principe de résolution) et que les résultats fournis sont très différents d'une méthode à l'autre (une réfutation, une solution, une forme normale, ...).

Comparaisons d'efficacité

On a finalement deux façons de comparer l'efficacité des méthodes :
- La simulation polynomiale [126]. Intuitivement, un système de preuve A simule polynomialement un système de preuve B si pour toute instance de problème P, la taille de la preuve de P dans A est bornée par un polynôme de la taille de la preuve de P dans B. Pour les systèmes utilisant le principe de résolution, on dispose d'un certain nombre de résultats (voir par exemple [129, 238, 240, 496, 373] et [267] pour les liens entre la recherche de coupes dans des systèmes d'inéquations linéaires et la résolution). Ces résultats sont importants mais limités : ils ne peuvent concerner que les méthodes pour lesquelles il y a une notion de preuve. De plus, les méthodes comparées dans les articles cités sont souvent un peu artificielles et très peu efficaces en pratique.
- Le chronomètre, arbitre suprême. On a là quelques résultats. Les méthodes utilisant le principe de résolution et ses extensions sont manifestement les plus mauvaises. De plusieurs ordres de magnitude. Intuitivement, en plus de l'explosion en temps de calcul, inhérente à toutes les méthodes, elles ont l'inconvénient majeur d'être aussi

	nature	taille des instances
Mise sous forme normale, BDD	complètes	30/40
Résolution	complètes	20/30
Méthodes énumératives	complètes	250/300
Programmation linéaire en 0/1	complètes	100/150
Réparation locale	incomplètes	1000

TAB. 1.1 – Une comparaison des méthodes pour résoudre SAT

très consommatrices d'espace mémoire. Les méthodes de mise sous forme normale, y compris les diagrammes binaires de décision, sont efficaces quand on doit coder et manipuler toutes les solutions d'une formule, mais pas quand on cherche simplement à savoir s'il y a en une (voir par exemple [482] pour une comparaison de l'efficacité de la procédure de Davis et Putnam et des BDD[17]). Les méthodes de programmation linéaire en 0/1, si elles sont meilleures que les précédentes, ne semblent pas aussi efficaces que les méthodes énumératives et sont surtout assez lourdes à mettre en œuvre (voir les temps de calculs donnés dans [270, 362, 256, 35]). Il reste finalement les méthodes de réparation locale et les méthodes énumératives qui semblent respectivement les plus efficaces pour trouver une solution s'il y en a une (cf section 1.1.4) et prouver qu'il n'y en a pas dans le cas contraire.

La table 1.1 résume la comparaison ébauchée ci-dessus en donnant un ordre de grandeur du nombre de variables des instances SAT difficiles que l'on peut traiter en des temps raisonnables avec les meilleures méthodes de chaque catégorie. Les instances dont il est question ici sont des instances 3-SAT difficiles générées aléatoirement (cf section 1.1.3). Par temps de calcul raisonnable, j'entends quelques minutes sur une station de travail. J'ai programmé et comparé un certain nombre des méthodes évoquées ci-dessus (cf section 1.2.2).

Efficacité des méthodes de réparation locale

Une des grandes nouveautés de ces dernières années est venue des méthodes de réparation locale. Ces méthodes consistent à partir d'une affectation complète des variables et à «réparer» cette affectation en changeant la valeur d'une variable tant qu'une solution n'a pas été trouvée ou qu'un certain nombre prédéfini de changements n'a pas été effectué. Le plus connu des algorithmes de ce type est sans doute le fameux recuit simulé [297] (on peut aussi citer le tabou [236, 237]). On savait ces méthodes très efficaces pour résoudre des problèmes d'optimisation combinatoire [297]. La surprise est venue de ce qu'elles sont aussi très efficaces pour traiter SAT, où une solution *exacte* est demandée. Ainsi, à la suite de [359], B. Selman, H. Levesque et D. Mitchell, dans [450] ont proposé un algorithme très simple (GSAT) qui permet de résoudre des instances SAT aléatoires difficiles qu'aucune

[17] J'ai fait un bon nombre d'expériences comparatives avec Aralia [425] et Pardi [416] qui confirment les résultats de [482].

méthode complète connue n'est capable de traiter. Depuis cet article, un grand nombre de travaux ont été publiés (voir par exemple [227, 228, 368, 447, 448, 449, 37, 230, 445]) relatant divers résultats expérimentaux, une étude analytique étant bien entendu beaucoup plus difficile (voir tout de même [377, 301]).

1.2 Divers travaux autour du problème SAT

1.2.1 Premiers pas

Ma première expérience en recherche a été d'essayer de comprendre les mécanismes sous-jacents de la SL-Resolution de R. Kowalski et D. Kuehner [302]. La SL-Resolution est un algorithme de test de satisfiabilité d'ensembles de clauses. Définie pour les clauses du premier ordre, elle n'est réellement applicable qu'en calcul propositionnel. Ceci est dû à l'introduction de la notion d'ancêtres qui rompt avec le caractère local du principe de résolution de J.A. Robinson [427].

En 1988, La SL-Resolution était «à la mode» au Groupe d'Intelligence Artificielle de Marseille-Luminy : P. Siegel en avait dérivé un algorithme permettant d'extraire des informations pertinentes d'un ensemble de clauses modélisant une connaissance (qui constitue le corps de sa thèse d'état [455]). Cet algorithme était au cœur de l'implémentation du module de résolution de contraintes booléennes du tout nouveau PrologIII [120, 31].

La preuve de la complétude de la SL-Resolution nécessite, même en calcul propositionnel, une induction non triviale sur la longueur des preuves et il était assez naturel d'essayer de la reformuler en termes simples d'affectations et de modèles. Ce faisant, L. Oxusoff et moi avons découvert une méthode de retour arrière intelligent pour la procédure de Davis et Putnam [172] telle qu'elle est mise en œuvre dans le livre de D. Loveland [326], sans d'ailleurs connaître, ni la procédure de Davis et Putnam et les nombreux travaux sur le problème SAT, ni la littérature sur le retour arrière intelligent (par exemple [466, 255, 76]). Cette méthode, appelée «principe de partition du modèle», est décrite dans l'article donné annexe B.2. Elle généralise une idée de S. Even, A. Itai et A. Shamir [203] ainsi de la notion d'«autark» proposée par B. Monien et E. Speckenmeyer dans [365]. En fait, son apport principal consiste en un lemme de codage qui permet de détecter en temps constant si le principe de retour arrière intelligent s'applique.

Notre version de la procédure de Davis et Putnam munie d'heuristiques convenables (et rebaptisée évaluation sémantique) s'est montré incomparablement plus efficace que la SL-Résolution tant pour tester la satisfiablité que pour produire les sous clauses impliquées par la formule initiale, sur un banc d'essai composé de cryptogrammes, de puzzles logiques et de petits problèmes combinatoires [287, 372, 53]. Il faut bien dire que cette efficacité ne devait quasiment rien à notre propriété de retour arrière intelligent. Le banc d'essai que nous avons proposé a servi à plusieurs autres chercheurs pour tester leurs outils [362, 29, 459, 461, 92].

Nous avons pu, grâce au principe de partition du modèle, reformuler partiellement la SL-Resolution et montrer que l'arbre de preuve qu'elle construit est toujours plus grand que celui construit par une variante de la méthode dite des matrices d'interconnection de

Bibel [43] munie précisément de cette propriété [372, 373]. L'intérêt de cette comparaison (par rapport aux autres contributions du même type [129, 238, 240, 496]) est que les deux algorithmes sont *a priori* assez différents et qu'il n'y a donc pas de correspondance immédiatement visible. De plus, elle a permis de comprendre pourquoi la SL-Resolution, pourtant considérée comme l'une des meilleures méthodes utilisant le principe de résolution, est toujours plus mauvaise en pratique que la procédure de Davis et Putnam (la méthode des matrices d'interconnection étant manifestement une mauvaise implémentation de cette dernière).

1.2.2 La plateforme logicielle Pardi

J'ai arrété de travailler sur le problème SAT à mon arrivée au LaBRI (en 1989). Je m'y suis à nouveau intéressé dans le cadre du projet Bahia du PRC-IA [21, 22] (à partir de 1992). Mon principal travail dans le cadre de ce projet a été la réalisation de Pardi. Pardi est une plateforme logicielle pour l'expérimentation d'algorithmes énumératifs de résolution du problème SAT [416]. Plus précisément, Pardi code un ensemble de clauses sous forme d'une matrice creuse et fournit toutes les primitives nécessaires pour mettre à jour cette structure de données lors de l'affectation des variables et pour afficher son état courant. L'objectif de cette plateforme était de mettre à disposition des membres du groupe un certain nombre d'algorithmes et un jeu de tests significatif pour pouvoir comparer l'efficacité pratique des différentes méthodes proposées. Dans sa dernière version, Pardi comprend :

- Une quinzaine d'algorithmes complets (notamment la procédure de Davis et Putnam [172]) et incomplets (notamment GSAT [450]).
- Une dizaine d'heuristiques (firstfail, ...) pour ces différentes procédures (certaines étant originales, d'autres ayant été proposées dans la littérature, par exemple dans [289, 191, 271]).
- Cinq générateurs aléatoires d'ensembles de clauses. Chaque générateur correspondant à un modèle statistique différent.
- À coté de la plateforme proprement dite, j'ai constitué un banc d'essais. Il comprend à ce jour environ de nombreux problèmes différents (puzzles logiques, problèmes combinatoires, ...) [372].

En fait, un des intérêts de Pardi est qu'il est relativement facile d'ajouter (et de supprimer) un algorithme, une heuristique, ou générateur d'instances et de mener des études statistiques comparatives entre les performances de différentes méthodes en termes de temps de calculs, de nombre d'appels ou toute autre mesure. La version distribuée de Pardi est de ce point de vue minimale mais représente tout de même quelques dizaines de milliers de lignes de code C. Pardi a été utilisé par l'équipe logique du LIM[18] et par moi-même et nous a permis d'expérimenter un grand nombre de méthodes. *A posteriori*, le projet Pardi est à la fois un triple échec et un succès. Échec d'abord du point de vue de l'objectif initial : nous n'avons pas réussi à améliorer significativement les algorithmes existants. Échec ensuite du point de vue de la rentabilité en termes du nombre de publications immédiatement

[18] Laboratoire d'Informatique de Marseille.

générées. Échec enfin et surtout du point de vue génie logiciel : faire cohabiter de nombreux algorithmes dans une même structure de données n'a été possible qu'au prix d'une complexité importante du logiciel. À la reflexion, il aurait mieux valut faire une boite à outil composée de petites commandes Unix (traducteurs, algorithmes, générateurs, ...) et les utiliser au travers de scripts du langage de commandes. Une récriture dans cette direction a commencé. Mais ce constat pessimiste ne doit pas cacher ce qu'a apporté la plateforme Pardi :

- Elle a permis de comparer un grand nombre d'algorithmes et d'heuristiques. Cette comparaison ne pouvait être qu'expérimentale. Et pour être équitable, elle devait être menée dans un cadre logiciel unique. En effet, aucun outil théorique ne permet de prévoir les performances pratiques des algorithmes énumératifs (la complexité théorique n'étant qu'asymptotique) et encore moins de prévoir la taille des instances traitables en des temps raisonnables. Il faut insister ici sur le fait que dans ce domaine, l'intuition est très mauvaise conseillère : il est toujours facile de justifier un résultat *a posteriori* et plus encore d'avoir de fausses bonnes idées.
- Elle a permis de refaire un certain nombre d'expériences évoquées dans la littérature récente, et ainsi d'une part de bien comprendre les méthodes proposées, d'autre part de vérifier (et parfois d'infirmer) les résultats annoncés (sur l'efficacité des algorithmes, l'existence de phénomès de seuils, les méthodes de réparation locale, ...).
- À défaut de preuve formelle, elle a permis de se convaincre qu'aucune amélioration sensible ne peut être apportée à la procédure de Davis et Putnam, ou, autrement dit, que les limites évoquées table 1.1 sont en quelque sorte intrinsèques.

Il n'est pas possible d'avancer de conclusions définitives sur la ≪bonne implantation≫ d'une méthode énumérative pour résoudre SAT. Voici néanmoins quelques remarques issues de l'expérience accumulée avec Pardi :

- L'efficacité, au chronomètre, d'un algorithme dépend pour beaucoup du choix à très bas niveau des structures de données. Par exemple, csat, le programme de O. Dubois et col. qui a gagné le challenge DIMACS sur SAT [191], recopie des environnements plutôt que de mettre en œuvre un retour arrière classique. Cette solution est très couteuse en place mémoire et n'est possible que sur une classe limitée d'instances. Pour l'instant aucune étude comparative sérieuse des différentes façons d'implanter l'affectation des variables et le retour arrière n'a été faite. Pire, les auteurs ne parlent quasiment jamais de ces ≪détails≫ qui sont pourtant décisifs[19].
- De nombreuses heuristiques de branchement ont été proposées (voir par exemple [45, 191, 162, 163, 271, 289, 331, 365, 512]). Elles sont plus ou moins compliquées. En fait, il semble que la seule chose déterminante soit de choisir la prochaine variable de façon à ce que, sur les deux branches correspondantes de l'arbre de recherche, le plus grand nombre de variables possible aient leur valeur fixée. Il est par ailleurs important d'équilibrer les arbres (et donc de maximiser le minimum sur les deux branches du nombre de variables éliminées). Le nombre d'occurrences dans les clauses binaires

[19]Bien sûr asymptotiquement tout cela n'est que billevesées, mais encore une fois il n'est pas question ici de complexité théorique mais de savoir ce que l'on peut faire ≪à notre age et à l'heure qu'il est≫.

semblent donc être le critère de choix important. Comme le remarque J.N. Hooker et V. Vinay dans [271], même sur les problèmes satisfiables, la taille des arbres de démonstration est loin d'être linéaire par rapport au nombre de variables. Par conséquent, sur la plupart des branches, on cherche à prouver l'insatisfiabilité de l'instance correspondante. Les heuristiques allant dans ce sens sont donc toujours préférables.

– La procédure de Davis et Putnam propose deux filtrages à chaque nœud de l'arbre de démonstration : la propagation des littéraux unitaires et celles de littéraux purs[20]. La première est fondamentale, la seconde ne semble pas très utile (toutefois, suivant les structures de données choisies elle peut être effectuée à très faible coût). Il semble que l'on puisse intégrer des filtrages plus complexes, en particulier en travaillant sur les clauses binaires (voir [45, 311, 191, 222]).

En résumé, il reste beaucoup de travail avant d'avoir une vision claire de ce qu'il est possible de faire et comment le faire.

J'ai insisté sur Pardi dans le cadre de ce mémoire parce que ce projet illustre assez bien un des problèmes de la recherche expérimentale en informatique : elle nécessite la réalisation d'outils d'(auto)formation et d'expérimentation, lourds, peu ≪rentables≫ et pourtant indispensables.

1.2.3 Sur la génération aléatoire d'instances difficiles

Afin d'illustrer le type d'études menées avec Pardi, cette section décrit celle que j'ai effectuée sur la génération aléatoire d'instances SAT satisfiables difficiles. Dès lors que l'intérêt de la génération aléatoire d'instances SAT est avéré, d'une part par le résultat de Chvàtal et Szemerédi [107] montrant qu'elles sont dures pour la résolution et donc pour toutes les méthodes énumératives du type procédure de Davis et Putnam (ce qui est confirmé en pratique), d'autre part par la mise en évidence de phénomènes de seuils [104, 162, 163, 229, 230, 312, 361], deux questions se posent :

1. Les instances générées avec le modèle standard sont elles les plus dures possibles ?
2. Est-il possible de générer aléatoirement des instances satisfiable difficiles ?

Cette deuxième question est particulièrement importante car, dans l'affirmative, on aurait un moyen simple et efficace de tester le taux de succès des algorithmes incomplets du type recuit simulé, tabou, GSAT ...De plus, le résultat de V. Chvátal et E. Szemerédi ne vaut que pour les instances insatisfiables. La situation ne serait donc pas trop défavorable si l'on était presque sûr de trouver une solution quand il y en a une (rapprochant de ce fait et au moins en pratique les classes NP et RP – voir [369] pour une superbe introduction aux algorithmes randomisés).

J'ai donc procédé avec R. Génisson et L. Sais à une large étude expérimentale tant pour les méthodes complètes que pour les méthodes incomplètes et dont l'idée était de voir ≪ce qui se passait≫ lorsqu'on modifiait légèrement le modèle de génération [226, 420].

[20] Un littéral est unitaire s'il apparaît dans une clause de longueur 1, il est pur si son opposé n'apparaît pas dans ϕ.

On peut par exemple remarquer qu'une instance est satisfiable si et seulement si elle ne contient pas de clause positive à un renommage de certaines variables près (un renommage substitue pour certaines variables x, les occurrences de x par $\neg x$ et réciproquement). On peut donc générer des instances satisfiables en interdisant le tirage de telles clauses. Afin de maintenir l'équiprobabilité des littéraux positifs et négatifs — sans laquelle les instances générées sont faciles pour toutes les méthodes — on pose un certain nombre de contraintes linéaires sur la probabilités de chaque structure de clause. La structure étant la séquence des signes des littéraux de la clause. On montre que le système de contraintes ainsi obtenu est entièrement déterminé par la valeur donnée à la probabilité de tirer une clause ne contenant que des littéraux négatifs.

Pour les deux types de méthodes (complètes et incomplètes), on peut répondre aux deux questions ci-dessus de la façon suivante [420] :

1. Les instances générées sont d'autant plus dures que la valeur donnée à la probabilité de tirer une clause négative est importante. Toutefois, même quand cette probabilité est maximum, les instances générées ne sont pas beaucoup plus difficiles que celles obtenues selon le modèle standard (elles rendent toutefois beaucoup plus délicat le réglage des paramètres des procédures incomplètes).
2. On peut générer des instances satisfiables au moins aussi dures que celles obtenues avec le modèle standard, ce qui ouvre de nombreuses perspectives pour tester les méthodes incomplètes.

Je continue à travailler sur ces questions avec avec R. Génisson du LIM et dans le cadre du groupe ≪Ressac≫ du PRC-IA.

1.2.4 Concurrence et procédure de Davis et Putnam

Le principal problème de la mise en œuvre de la procédure de Davis et Putnam est le choix d'une bonne heuristique. Pour les instances insatisfiables, seul le choix de la prochaine variable à affecter importe. Pour les instances satisfiables, le choix de la première valeur à donner à cette variable peut se réveler aussi important. Or, il arrive fréquement que les heuristiques choisies se ≪trompent≫ de valeur. Dans [28], R. Génisson, B. Benhamou et moi avons étudié dans quelle mesure le maintien ≪en parallèle≫ des deux affectations possibles pouvait améliorer ou dégrader les performances de la procédure de Davis et Putnam. L'algorithme que nous avons proposé consiste à faire en quelque sorte un parcours en largeur d'abord de l'arbre de résolution, en paramétrant le nombre de processus disponibles ainsi que la taille en deça de laquelle on revient à un parcours en profondeur d'abord. La large étude expérimentale que nous avons effectuée sur des instances aléatoires satisfiables (et générées suivant [420]) montre que pour chaque valeur du nombre de variables et du nombre de clauses, il existe effectivement un nombre de processeurs optimum (qui est en général supérieur à un). Autrement dit, une implémentation séquentielle d'un algorithme parallèle peut être plus efficace que la version purement séquentielle du même algorithme, parce que les stratégies d'exploration de l'espace de recherche ne sont pas les mêmes.

1.3 Classes polynomiales

Cette section présente mes travaux sur les restrictions polynomiales du problème SAT. La première partie fait un bref tour d'horizon sur le sujet. Suit la présentation de mes travaux proprement dits. En fait, la rédaction de la première partie peut aussi être considéré comme un travail de recherche dans la mesure où il n'existe pas, du moins à ma connaissance, d'article faisant un tel état de l'art. Pas plus d'ailleurs qu'une communauté constituée travaillant sur cette question. Rassembler une bibliographie éparse a donc demandé un effort substantiel.

Il faut insister sur le fait que définir une classe polynomiale demande deux algorithmes : un algorithme de reconnaissance, qui décide si une instance SAT donnée appartient ou non à la classe, et un algorithme de décision pour les instances appartenant à la classe. Ces deux algorithmes peuvent être éventuellement confondus en un seul, mais ils doivent être impérativement tous les deux polynomiaux pour que la classe le soit. Cette remarque a motivé mes recherches sur ce thème. En effet, si chaque classe polynomiale connue demandait un algorithme de reconnaissance spécifique, programmer un démonstrateur général qui ≪récupère≫ chacune de ces classes serait non seulement fastidieux mais encore fort peu efficace car il faudrait, à chaque étape de la démonstration, appliquer un à un ces algorithmes de reconnaissance. J'ai donc étudié le comportement de la procédure de Davis et Putnam sur les principales restrictions polynomiales de SAT. Plus généralement, je me suis intéressé à ce que l'on peut faire sur ces classes avec des outils algorithmiques simples, au premier rang desquels se trouve le mécanisme ≪affectation plus propagation des littéraux purs et unitaires≫. Cette approche ≪pratique≫ de la question à l'avantage de montrer le caractère souvent artificiel des restrictions et des algorithmes proposés. Et comme dit la chanson : ≪travail facile ou besogne très dure n'ont de valeur qu'en leur utilité≫[21].

1.3.1 Principales classes polynomiales

Clauses binaires

La première classe polynomiale à avoir été exhibée (par S.A. Cook [126]) est 2-SAT, i.e. les instances formées de clauses binaires. Le nombre de clauses binaires sur un vocabulaire de n variables est $2^2 \binom{n}{2}$, c'est à dire quadratique. La saturation pour le principe de résolution[22] est donc polynomiale sur 2-SAT (la résolvante de deux clauses binaires étant au pire de longueur 2). Deux algorithmes linéaires ont été proposés pour résoudre 2-SAT : le premier par S. Even, A. Itai et A. Shamir [203] qui est un algorithme énumératif utilisant un principe de retour arrière intelligent (la notion d'≪autark≫ [365] ou de partition du modèle [287]), le second par B. Aspvall, M. Plass et R. Tarjan [16] qui utilise un codage des

[21] Le triomphe de l'anarchie, par C. d'Arvray.

[22] La saturation est l'algorithme consistant à ajouter à l'ensemble de clauses considéré toutes les résolvantes possibles entre deux clauses de l'ensemble et à recommencer cette opération avec jusqu'à obtention de la clause vide ou à ce qu'il n'y ait plus de nouvelles clauses produites. On montre facilement que cet algorithme est complet pour SAT.

clauses en termes de graphes et l'algorithme de R. Tarjan pour calculer les composantes fortement connexes d'un graphe orienté [472]. Comme on le verra, ces deux algorithmes jouent un rôle fondamental dans l'étude des classes polynomiales du problème SAT.

L'algorithme proposé dans [16] résout en fait le problème des formules booléennes quantifiées restreint aux clauses binaires, dont le cas général[23] est le problème P-espace-complet de référence [290].

D'autres problèmes liés aux clauses binaires peuvent être résolus efficacement. Ainsi, on peut :

- Vérifier qu'une instance 2-SAT admet une unique solution en temps linéaire [252, 260].
- Produire toutes les variables impliquées et les classes de variables équivalentes en temps linéaire [253][24].
- Déterminer la satisfiabilité, les variables impliquées, les classes de littéraux équivalents ≪on-line≫ en $\mathcal{O}(mn)$[25] [286].

Pour finir signalons que les clauses binaires ont une certaine utilité pratique dans le cadre de la preuve de circuits logiques [405, 311].

Clauses de Horn

L'autre restriction polynomiale bien connue du problème SAT est Horn-SAT (toutes les clauses comportent au plus un littéral positif). Cette classe de formules a été exhibée, comme son nom l'indique, par A. Horn dans un article de 1951 [272] faisant suite à un travail de J.C.C McKinsey [348]. Les clauses de Horn jouent un rôle important en démonstration automatique et elles ont donc été étudiées depuis longtemps (voir par exemple [263, 446]). Comme pour les clauses binaires, il existe un algorithme trivial pour tester la satisfiabilité d'un ensemble de clauses de Horn : il suffit de saturer l'ensemble pour la propagation sur les clauses unitaires positives, ce qui peut être fait en temps linéaire [357, 169, 421]. Plusieurs autres algorithmes linéaires ont été proposés qui utilisent des techniques de graphes [187, 444, 233].

Au contraire de ce qui se passe pour les clauses binaires, le probème de la reconnaissance des clauses de Horn n'est pas trivial. Il n'est en effet pas immédiat de déterminer si une instance SAT est de Horn à un renommage près des variables[26]. On sait toutefois faire cela en temps linéaire [15, 95, 258] en ramenant le problème à celui de la satisfiabilité d'un ensemble de clauses binaires (cf section 1.3.4 pour une discussion détaillée).

Un certain nombre d'autres problèmes liés aux clauses de Horn peuvent être résolus efficacement. Ainsi, on peut :

- Déterminer si un ensemble de clauses de Horn admet une solution unique en temps

[23] Ce problème est appelé QBF, pour ≪Quantified Boolean Formulae≫, dans [223].

[24] En fait, il ne s'agit pas vraiment de temps linéaire mais plutôt de temps linéaire en moyenne sur des instances tirées aléatoirement avec le modèle de longueur constante (cf section 1.1.3).

[25] Tout cela est facilement obtenu grâce à l'algorithme de G.F. Italiano [280] pour calculer ≪on-line≫ la clôture transitive d'un graphe et qui est en $\mathcal{O}(mn)$.

[26] Renommer les variables $x_1,\ldots,x_k$ d'une formule sous forme normale conjonctive ϕ consiste à substituer dans ϕ les littéraux construits sur les x_i par leurs opposés.

linéaire [358, 391].
- Déterminer si un ensemble de clauses admet un Horn renommage unique en temps linéaire [260].
- Maintenir «on-line» la satisfiabilité d'un ensemble de clauses de Horn en temps linéaire amorti [19, 488].

En revanche, Horn-QBF ne semble pas dans P, mais je ne connais pas de preuve ou de réfutation de cette conjecture. D'autre part, trouver un sous-ensemble Horn maximum est NP-complet [97].

Généralisation de Horn-SAT et 2-SAT

Il était assez naturel de chercher à généraliser Horn-SAT et 2-SAT. Plusieurs auteurs l'ont fait, de différentes façons.

q-Horn Clauses : La classe des q-Horn clauses a été introduite dans [59]. Elle généralise à la fois les clauses de Horn et les clauses binaires.

Un ensemble de clauses est dit *q-Horn* si il existe une partition des variables en deux sous-ensembles disjoints H et Q tels que :
- Aucune clause ne contient plus de deux variables de Q.
- Aucune clause ne contient plus d'un littéral positif de H.
- Aucune clause contenant un littéral positif de H ne contient un littéral de Q.

On considère bien sûr les instances q-Horn modulo un renommage. Une fois reconnu les ensembles H et Q, il est facile de décider de la satisfiabilité d'un ensemble q-Horn. On affecte la valeur 0 à toutes les variables de H et on résout le problème restant, qui ne contient plus que des clauses binaires. Dans [61], un algorithme linéaire est proposé pour reconnaître Q et H et dans [60] un indice de complexité est proposé qui utilise cet algorithme. Notons aussi que J.-J. Hébrard, dans [259], a proposé une généralisation des q-Horn clauses dans laquelle on impose aucune condition sur Q (il reste toujours que l'ensemble de départ est satisfiable si et seulement si le sous-ensemble des clauses construites sur Q l'est). On peut calculer en temps linéaire le plus grand ensemble H possible qui est unique : c'est ce que J.-J. Hébrard appelle la base de Horn. Pour démontrer l'unicité, on utilise une version du théorème de partition du modèle similaire à celle que proposé dans [373] pour comparer la SL-Resolution avec la procédure de Davis et Putnam (voir aussi section 1.3.4).

Hiérarchie de Dalal et Etherington S. Yamasaki et S. Doshita, dans [510] ont proposé une autre généralisation des clauses de Horn dans laquelle les clauses peuvent contenir plusieurs littéraux positifs à condition que ces derniers soient en quelque sorte imbriqués. Ils ont proposé un algorithme, utilisant le principe de résolution et en $\mathcal{O}(|\phi|^3)$, pour décider les instances de cette classe. Un autre algorithme, énumératif et en $\mathcal{O}(|\phi|^2)$, a été proposé par V. Arvind et S. Biswas dans [14].

G. Gallo et M.G. Scutellà, dans [221] ont généralisé la classe de [510] et l'algorithme [14] en définissant une hiérarchie de Γ_0, Γ_1, ...d'instances SAT, où Γ_0 est Horn-SAT et Γ_k est défini à partir de Γ_{k-1} en ajoutant des littéraux positifs de la façon suivante : $S \in \Gamma_k$ si

et seulement si il existe un littéral p tel que $S_p \in \Gamma_{k-1}$ et $S_{\overline{p}} \in \Gamma_k$. Γ_1 est la classe de [510]. Ils ont proposé deux algorithmes pour reconnaître et résoudre les instances de la classe Γ_k : le premier est $\mathcal{O}(|S|)$ en espace et en $\mathcal{O}(|\phi|.n^{2k})$ en temps ; Le second, qui utilise un mécanisme de cache, est en $\mathcal{O}(|\phi|.n^k)$ en temps et en espace.

La hiérarchie de G. Gallo et M.G. Scutellà a elle même été généralisée par M. Dalal et D.W. Etherington dans [171]. Cet article introduit en fait deux hiérarchies entrelacées, (Δ_i) et (Ω_i). Soit ϕ une instance SAT, $\phi^\star$ dénote ϕ simplifiée pour la propagation sur les littéraux unitaires et purs. ϕ est dite *de base* si $\phi^\star$ contient la clause vide ou si elle ne contient pas de clause positive (ne contenant que des littéraux positifs) ou encore si elle ne contient pas de clause négative. La classe Δ_0 contient les instances de base. On remarque ensuite que pour toute affectation partielle σ, si $\phi^\star_\sigma \subset \phi$ alors ϕ est satisfiable si et seulement si $\phi^\star_\sigma \subset \phi$ l'est (principe de partition du modèle [287]). Maintenant,

1. Pour tout k, $S \in \Omega_k$, si et seulement si ou bien $S \in \Delta_k$ ou bien pour tout littéral p p, ou bien $S^\star_p \in \Delta_k$ et $S^\star_{\overline{p}} \in \Omega_k$, ou bien $S^\star_p \subset S$ et $S^\star_p \in \Omega_k$.
2. Pour tout $k > 0$, $S \in \Delta_k$, si et seulement si un littéral p tel que, ou bien $S^\star_p \in \Omega_{k-1}$ et $S^\star_{\overline{p}} \in \Delta_k$, ou bien $S^\star_p \subset S$ et $S^\star_p \in \Delta_k$.

On peut remarquer que Ω_0 contient 2-SAT (à la différence de la hiérarchie de G. Gallo et M.G. Scutellà). Il est facile de dériver un algorithme pour reconnaître et résoudre ces instances. Sa complexité en temps est en $\mathcal{O}(|\phi|.n^{3k+1})$ et peut être réduite comme dans [221] à $\mathcal{O}(|\phi|.n^k)$ en utilisant un mécanisme de cache.

Extension liées à la programmation linéaire en 0/1 : V. Chandru et J.N. Hooker ont défini, dans [96], une autre classe de formule généralisant les clauses de Horn (voir aussi [470]) et qui peut être décidée par résolution unitaire. Cette généralisation s'appuie des résultats de programmation linéaire en 0/1. Toutefois, il est très difficile d'avoir la moindre intuition sur ce que sont les instances de cette classe. Dans [439], J.S. Schlipf et col. ont démontré que l'algorithme consistant à choisir une variable, regarder le résultat de la propagation unitaire pour ses deux littéraux, garder un des deux littéraux ne menant pas à une contradiction et recommencer tant qu'on aboutit pas à une double contradiction ou à l'ensemble vide de clauses, est complet pour une classe de formules incluant la classe de [96] et celle des formules équilibrées définie dans [123] (qui découle aussi de résultats de programmation linéaire en 0/1). L'inconvénient de ces classes est qu'on ne dispose pas pour elles d'algorithme de reconnaissance polynomial.

Restrictions sur les relations

Un des premiers articles parus sur les restrictions polynomiales de SAT est celui T.J. Schaeffer [437]. Il considère le problème suivant : supposons que l'on travaille sur des contraintes propositionnelles plutôt que sur des clauses. Une contrainte propositionnelle d'arité n est un sous-ensemble de $\{0,1\}^n$, ou autrement dit une relation de la forme $\lambda x_1, \ldots, x_n r(x_1, \ldots, x_n)$, les x_i étant des paramètres propositionnels. Pour décrire un problème, on se donne un ensemble $\mathcal{R}$ de relations que l'on suppose clos pour la quantification existentielle, la conjonc-

tion, et la substitution de variables par des variables ou des constantes. On appelle donc SAT($\mathcal{R}$) le plus petit ensemble de formules tel que :

- Si r est une relation de $\mathcal{R}$ d'arité k et x_1, ..., x_k sont des variables, alors $r(x_1, \ldots, x_k)$ appartient à SAT($\mathcal{R}$).
- Si f et g appartiennent à $\mathcal{R}$ et x et y sont des variables, alors $\exists x f$, $f \wedge g$, $f_{x \leftarrow y}$, $f_{x \leftarrow 0}$ et $f_{x \leftarrow 1}$ appartiennent à SAT($\mathcal{R}$).

La question est : est-il alors possible de classer la difficulté de SAT($\mathcal{R}$) en fonction de $\mathcal{R}$?

Le théorème s'énonce comme suit. Si $\mathcal{R}$ satisfait une des conditions (a)-(f) suivantes, alors SAT($\mathcal{R}$) est P-complet, sinon il est NP-complet.

(a) Toute relation de $\mathcal{R}$ contient $(0, \ldots, 0)$.

(b) Toute relation de $\mathcal{R}$ contient $(1, \ldots, 1)$.

(c) Toute relation de $\mathcal{R}$ peut être écrite comme un ensemble de clauses contenant au plus un littéral positif par clause.

(d) Toute relation de $\mathcal{R}$ peut être écrite comme un ensemble de clauses contenant au plus un littéral négatif par clause.

(e) Toute relation de $\mathcal{R}$ peut être écrite comme un ensemble de clauses binaires.

(f) Toute relation de $\mathcal{R}$ est affine, c'est à dire peut être écrite comme un ensemble d'équations de la forme $x_1 \oplus \ldots \oplus x_k = 0$ ou $x_1 \oplus \ldots \oplus x_k = 1$.

Cette très jolie caractérisation peut être étendue aux formules booléennes quantifiées : $\mathcal{R}$-QBF est P-complet pour les formules respectant au moins une des conditions (c)-(f) et P-espace-complet sinon.

Curieusement, le travail de T.J. Schaeffer est resté longtemps inexploité, et ce n'est que récemment que ses résultats ont été étendus aux problèmes de satisfaction de contraintes [486, 131, 244, 288, 179]. D'autres théorèmes de dichotomie ont été obtenus par N. Creignou qui étendent celui de Schaefer. Avec M. More, elle a ainsi caractérisé les relations traitables (pour la satisfiabilité) en exprimant la structure de leurs solutions par des langages rationnels [165]. Avec M. Hermann, elle a montré que les conditions que le problème de compter les solutions d'une instance $SAT(\mathcal{R})$ est dans FP si et seulement si toutes les relations de $\mathcal{R}$ sont affines (autrement $\#SAT(\mathcal{R})$ est #P-complet) [166]. Avec J.-J. Hébrard, elle a étudié les restrictions de SAT dont on peut énumérer les solutions de telle façon qu'il sécoule un temps polynomial entre deux solutions [164]. Notons pour finir que la classe des relations affines résiste très sérieusement au traitement par une méthode énumérative et qu'elle est une des rares dans ce cas (cette question sera discutée plus en détails section 1.3.3).

Restrictions sur le nombre d'occurrences par variable

Pour prouver que le problème du voyageur de commerce est NP-complet, C. Papadimitriou a réduit 3-SAT à 3,5-SAT, la classe des instances 3-SAT dans lesquelles chaque variable apparaît au plus 5 fois [376] (plus généralement, r,s-SAT désigne la classe des instances SAT dans lesquelles chaque clause contient exactement r variables et chaque variable apparaît au plus s fois). Reprenant cette idée (de limiter le nombre d'occurrences par variable), C.A. Tovey, dans [478], a montré les résultats suivants :

- SAT $\propto \star$,3-SAT ($\star$ voulant dire pour tout r).
- 3,4-SAT est NP-complet.
- $\star$,2-SAT $\in$ P.
- Toute instance r,r-SAT est satisfiable (démontré grâce à un théorème de P. Hall [248])

Il a de plus émis la conjecture que si $s \leq 2^{r-1} - 1$, toute instance r,s-SAT est satisfiable. Cette conjecture a été réfutée par O. Dubois [189] qui a exhibé une instance 4,6-SAT insatisfiable. Dans ce papier, il est établi que si toute instance r_0,s_0-SAT est satisfiable, alors pour tout entier positif λ, toute instance $r_0 + \lambda$,$s_0 + \lambda[s_0/r_0]$-SAT est satisfiable ($[x]$ désignant la partie entière de x). Ces résultats ont été améliorés par J. Kratochvil, P. Savicky et Z. Tuza [304] qui ont prouvé qu'il existe un entier $f(r)$ tel que r,s-SAT est dans P pour $s \leq f(r)$ et est NP-complet pour $s > f(r)$. De plus, $f(r)$ croît exponentiellement avec r de telle façon que $\lfloor 2^r/er \rfloor \leq f(r) \leq 2^{r-1} - 2^{r-4} - 1$. Pour finir, signalons que E. Humbert, dans sa thèse [275], a donné un certain nombre d'autres restrictions NP-complètes (portant principalement sur le nombre de clauses pouvant partager les mêmes variables).

Formules bien imbriquées

D. Lichenstein, dans [319], s'est intéressé aux formules dont le graphe sous-jacent[27] est planaire. Il a montré que même restreint aux formules planaires 3-SAT est NP-complet et 3-QBF est P-espace-complet. D.E. Knuth, dans [299], a repris cette idée et a défini la classe des formules bien imbriquées : soit $<$ un ordre total sur les variables étendu en un préordre sur les littéraux ($l \equiv \neg l$). Une clause C *chevauche* une clause D si il existe des littéraux $p, q \in C$ et $r \in D$ tels que $p < r < q$. Un ensemble de clauses est *bien imbriqué* s'il ne contient pas deux clauses se chevauchant. D.E. Knuth, a proposé un algorithme de décision linéaire pour cette classe de formules, mais cet algorithme suppose que les variables sont ordonnées suivant $<$ et que les clauses sont ordonnées suivant l'ordre induit. De plus, D.E. Knuth, n'a pas traité le problème de la reconnaissance. P. Rossa, dans [430], a donné un algorithme qui détermine en temps linéaire si une formule est bien imbriquée pour un ordre donné des variables.

À ce jour, je ne connais pas d'algorithme de reconnaissance complet (c'est à dire pour tout ordre sur les variables) qui soit polynomial. Notons aussi que les formules bien imbriquées semblent poser des problèmes aux méthodes énumératives (cf section 1.3.3).

1.3.2 Puissance de la procédure de Davis et Putnam

Comme on peut s'en convaincre à la lecture de la section précédente, les algorithmes de reconnaissance et de décision pour les classes polynomiales connues du problème SAT sont très différents les uns des autres et très différents des démonstrateurs généraux. Or, il est intéressant de garantir que ces derniers ont un comportement raisonnable sur les

[27] C'est à dire le graphe non orienté dont les sommets sont les variables et les clauses et qui est construit en reliant chaque clause aux variables qu'elle contient.

formules que l'on sait traiter efficacement. On pourrait bien sûr y incorporer des routines spécialisées dans le traitement de telle ou telle classe de formules. Cette façon de procéder n'est pas toutefois satisfaisante car elle est lourde à mettre en œuvre et le surcoût engendré par la reconnaissance des cas particuliers risque fort d'être important. Je me suis donc demandé quelles étaient les performances de la procédure de Davis et Putnam sur les principales classes polynomiales de SAT. Les résultats de ces investigations sont consignés dans [415] et [421]. L'article [421] est donné annexe B.2, aussi je n'en présente ici qu'un résumé succinct.

J'ai commencé par étudier la complexité exacte de la mise en œuvre de la procédure de Davis et Putnam. M. Minoux a montré que la résolution unitaire est linéaire [357]. J'ai montré que la propagation des littéraux purs et la mise à jour des compteurs servant aux heuristiques et à la détection des cas où le principe de partition du modèle s'applique peuvent aussi être rendues linéaires (dans une implémentation effective de la procédure de Davis et Putnam). Cela complète les descriptions de M. Dalal [169]. À partir de l'implémentation décrite, on montre facilement que la procédure de Davis et Putnam est quadratique sur les clauses de Horn et que, munie du principe de partition du modèle, elle est quadratique sur les clauses binaires (de plus, il est facile de la rendre linéaire dans les deux cas en utilisant le ≪truc≫ de S. Even, A. Itai et A. Shamir [203] consistant a effectuer ≪en parallèle≫ les affectations à vrai et à faux de la variable sélectionnée). J'ai aussi montré que, grâce à une généralisation du principe de partition du modèle, elle est quadratique sur 2-QBF.

J'ai réalisé une étude expérimentale détaillée qui montre que ces complexités au pire ne sont jamais atteintes et qu'à condition de choisir convenablement l'heuristique de branchement (c'est à dire à garantir qu'elle soit en temps linéaire amorti[28]) on observe toujours des temps de calcul linéaires. Dans [421], j'ai donné quelques éléments permettant de comprendre pourquoi. Des résultats partiels allant dans cette direction avaient été obtenus par R. Petreschi et B. Simeone [387].

Je vois plusieurs intérêts à ce travail :

- Il clarifie la situation : Une présentation des algorithmes en termes d'affectation de valeurs aux variables rend leur lecture beaucoup plus facile. Qui a essayé de déchiffrer des articles comme [203] ou [286] peut aisément s'en convaincre. De plus, elle permet de distinguer très vite ce qui est dû à la logique de ce qui est dû aux structures de données.
- Il montre que l'on peut facilement obtenir des algorithmes de complexité optimale en spécialisant des démonstrateurs généraux et que, même sans ces spécialisations, ces derniers ont une très bonne complexité pratique. De plus, ils n'ont pas besoin de reconnaître que l'instance traitée appartient à telle ou telle classe.
- Il montre qu'efficacité et simplicité concordent, ce qui est tout de même assez rassurant d'un point de vue philosophique. Par exemple, l'algorithme ≪PLPD[29]≫ que j'ai proposé dans [415] pour traiter les q-Horn clauses de E. Boros, Y. Crama et P.L. Ham-

[28] Voir [473] pour une introduction à cette notion.

[29] Partial Lookahead Davis and Putnam.

mer [59, 60, 61] est très proche des meilleurs implémentations de la procédure de Davis et Putnam qui comportent un mécanisme de propagation sur les clauses binaires [45, 191, 163].

1.3.3 Classes polynomiales par affectation

Poursuivant l'idée de [415], nous avons étudié, R. Génisson et moi, les classes polynomiales du problème SAT et des problèmes de satisfaction de contraintes qui sont susceptibles d'être traitées efficacement par une procédure énumérative de type Davis et Putnam [224]. Ce travail vient en complément de celui de M. Dalal et D.W. Etherington [171].

Nous avons donc défini une première procédure, appelée «procédure de Davis et Putnam à profondeur bornée» dont le principe est le suivant : Soit ϕ l'ensemble de clauses à traiter. On se donne un entier k. Si la satisfiabilité de ϕ n'est pas immédiatement décidable, alors il y a trois cas possibles :

- ϕ contient un littéral unitaire ou pur q. Dans ce cas, on rappelle l'algorithme sur ϕ_q et k.
- ϕ ne contient pas de littéral unitaire ou pur et $k = 0$. Dans ce cas, on répond "je-ne-sais-pas".
- ϕ ne contient pas de littéral unitaire ou pur et $k > 0$. Dans ce cas, on choisit une variable neutre p et une valeur v et on rappelle l'algorithme d'abord sur $\phi_{p \leftarrow v}$, puis sur $\phi_{p \leftarrow 1-v}$, si $\phi_{p \leftarrow v}$ n'est pas satisfiable (c'est à dire si on a obtenu la réponse non ou "je-ne-sais-pas"), la profondeur devant être bornée par $k-1$ sur une des deux branches et par k sur l'autre.

Le choix de la branche sur laquelle la profondeur est décrémentée n'est pas forcément fait immédiatement. Il peut être décidé au retour de l'appel de la procédure sur $\phi_{p \leftarrow v}$. On montre facilement que cette procédure s'exécute au pire en $\mathcal{O}(n^k \times |\phi|)$.

On montre alors que la procédure de Davis et Putnam bornée à la profondeur 1 est complète sur les clauses de Horn et les clauses binaires et que bornée à la profondeur k elle est complète sur la famille Γ_k de la hièrarchie de G. Gallo et M.G. Scutellà [221]. On peut définir une procédure de Davis et Putnam bornée à la profondeur k et qui teste pour chaque littéral l apparaissant dans les clauses binaires si $\neg l$ peut être dérivé par propagation unitaire (dans l'esprit de [415]). Bornée à la profondeur 1, cette procédure est complète pour les q-Horn clauses [59]. Ces résultats ont été ennoncés sous d'autres formes dans [171] et [415, 421].

Nous avons définit un second algorithme, appelé $\mu - Production$, produisant des clauses impliquées par l'ensemble initial sur le même principe. Ces clauses sont des impliqués (premiers si $k = n$ et si l'on élimine les clauses subsumées) de l'ensemble initial. Elles sont en particulier utilisées dans le cadre de la représentation des connaissances (voir [341] pour un discussion récente sur ces questions). On peut montrer que $\mu - Production$ bornée à la profondeur k est strictement plus puissant que la résolution avec des clauses bornées à la taille k introduite par Z. Galil [217] et strictement moins puissant que ce dernier algorithme avec des clauses bornées à la taille $k+1$.

Nous avons ensuite appliqué ces deux algorithmes à la traduction clausale des problèmes

de satisfaction de contraintes[30]. Il existe en fait deux traductions des CSP en clauses : celle proposée par J. de Kleer [177] et celle proposée par M. Dalal [170].

Une partie importante de la littérature sur les CSP porte sur la notion de consistance partielle. L'idée est d'éliminer des relations d'un CSP les affectations partielles des variables ne pouvant pas faire partie d'une solution. Cette notion sert donc principalement au filtrage des problèmes : en supprimant ces affectations partielles inconsistantes on espère faciliter la résolution du CSP. Dans [206], E. Freuder a formalisé la notion de k-consistance. Dans le cas des CSP binaires, la 2-consistance est aussi appelée *consistance d'arc* et la 3-consistance *consistance de chemin* [332]. On trouvera une discussion détaillée sur ces notions dans [479]. Une chose est de vérifier qu'un CSP est k-consistant (ou fortement k-consistant), une autre est de réaliser cette k-consistance en supprimant des relations les k-uplets inconsistants. Les meilleurs algorithmes de filtrages (c'est à dire réalisant la k-consistance) sont en $n^k d^k$ [130], où n est le nombre de variables et d la taille du plus grand domaine. Ainsi le filtrage par consistance d'arc peut être réalisé en temps linéaire dans la taille du CSP [363]. Depuis ce dernier article, un certain nombre d'améliorations ont été proposées pour les consistances d'arc et de chemin (voir par exemple [249, 42]). Nous avons établi les résultats suivants :

1. On peut définir une variante de la procédure de Davis et Putnam bornée à la profondeur $k-1$ qui retourne "oui" si le CSP considéré est fortement k-consistant. La réciproque est vraie pour le codage de M. Dalal et fausse pour celui de J. de Kleer (mais elle devient vraie en bornant la profondeur à k et non $k-1$).

2. On peut définir une variante de l'algorithme $\mu - production$ borné à la profondeur k qui réalise la k-consistance.

3. La procédure de Davis et Putnam bornée à la profondeur 1 est complète sur l'expression clausale d'un CSP arborescent (voir [207]) vérifiant la consistance d'arc et la procédure PLDP bornée à la profondeur 1 est complète sur cette même expression même si elle ne vérifie pas la consistance d'arc.

4. La procédure de Davis et Putnam bornée à la profondeur 1 est complète sur l'expression clausale des CSP Zéro-Un-Tous définis par M. Cooper, D.A. Cohen et P.G. Jeavons [131] vérifiant la consistance d'arc et la procédure PLDP bornée à la profondeur 1 est complète sur cette même expression même si elle ne vérifie pas la consistance d'arc.

En résumé, l'intérêt principal de [224] est de fournir un cadre unificateur – la procédure de Davis et Putnam à profondeur bornée – pour traiter les principales classes polynomiales du problème SAT et des problèmes de satisfaction de contraintes. De plus, les procéures que nous avons proposées sont efficaces en ce sens qu'elles sont de complexités comparables à celles des meilleurs algorithmes CSP.

[30]Le formalisme des problèmes de satisfaction de contraintes – dans la suite, nous abbrévierons en CSP (pour Constraint Satisfaction Problems) – tel qu'il a été introduit par Montanari dans [366] (voir [479] pour une introduction générale aux CSP).

1.3.4 Horn renommage par affectation

Un autre exemple de problème que la procédure de Davis et Putnam permet de résoudre efficacement est la recherche d'un Horn renommage pour une instance SAT donnée. Renommer les variables $x_1,\ldots,x_k$ d'une formule sous forme normale conjonctive ϕ consiste à substituer dans ϕ les littéraux construits sur les x_i par leurs opposés (il est bien clair que cette opération préserve la satisfiabilité). Un renommage est dit Horn s'il transforme ϕ en un ensemble de clauses de Horn. Comme je l'ai déjà indiqué section 1.3.1, on connaît des algorithmes linéaires pour tester si un ensemble de clauses de Horn est satisfiable [187, 357, 444, 233]. Les clauses de Horn (non nécessairement propositionnelles) sont aussi largement utilisées en démonstration automatique [263]. D'où l'intérêt d'algorithmes de Horn renommage efficaces.

B. Meltzer a été le premier à s'intéresser au Horn renommage (dans le contexte de la démonstration automatique [350]). H.R. Lewis, dans [317], a proposé un algorithme de complexité quadratique en réduisant le problème du Horn renommage à la satisfiabilité d'un ensemble de clauses binaires. L'idée est d'associer à ϕ la formule $H(\phi)$ suivante :

$$H(\phi) \;=\; \bigwedge_{C_k\in\phi} \;\bigwedge_{p_i,p_j\in C_k} p_i \vee p_j$$

La formule $\bigwedge_{p_i,p_j\in C_k} p_i \vee p_j$ est satisfaite par une affectation σ si σ falsifie au plus un des littéraux de C_k. Il y a donc une correspondance bi-univoque entre les solutions de $H(\phi)$ et les Horn-renommage de ϕ. $H(\phi)$ contient au plus $\mathcal{O}(n^2)$ clauses (où n est le nombre de variables de ϕ). Comme le test de satisfiabilité d'un ensemble de clauses binaires peut être fait en temps linéaire [203, 16], on a bien un algorithme quadratique. Dans [15], B. Apsvall a amélioré la méthode en obtenant un codage linéaire de $H(\phi)$ (et donc une complexité linéaire) au prix de l'ajout de nouvelles variables. H. Mannila et K. Mehlorn [338] d'une part, G. Lindhorst et F. Shahrokhi [322] d'autre part, ont réduit le problème à celui de trouver des composantes fortement connexes dans un graphe orienté (leurs algorithmes sont en $\mathcal{O}(mn)$, où m est le nombre de clauses). V. Chandru & col. [95] ont obtenu un algorithme linéaire en codant l'ensemble de clauses à l'aide d'un graphe bipartite et en introduisant des concepts spécifiques à ces graphes. *Last but not least*, J.-J. Hébrard [258] a considéré la relation binaire sur les littéraux qui exprime les contraintes du Horn renommage et a proposé un algorithme lui aussi linéaire qui parcourt en largeur d'abord et avec le contrôle approprié, le graphe de cette relation. Cet algorithme peut être vu comme une extension de celui de S. Even ; A. Itai et A. Shamir [203]. Excepté [95], aucun de ces articles ne présente de résultats expérimentaux. Et c'est bien dommage.

Dans [225], avec R. Génisson, nous avons dérivé un algorithme de Horn renommage de la procédure de Davis et Putnam [172]. L'idée est simplement de changer la définition des clauses pour traduire qu'une clause est satisfaite si et seulement si elle comporte au plus un littéral falsifié. Autrement dit, il consiste à utiliser directement l'idée de H.R. Lewis. Nous avons montré :

- Que toutes les notions usuelles, littéraux unitaires, littéraux purs, théorème partition du modèle [287, 372] (≪autark≫ [365, 203]) se transposent et que l'on peut donc utiliser strictement les mêmes structures de données que pour la satisfiabilité.

- Que l'algorithme obtenu est quadratique dans le pire des cas, mais qu'il est facile de le rendre linéaire en utilisant le ≪truc≫ de S. Even, A. Itai et A. Shamir (ce que fait en fait l'algorithme de J.J. Hébrard sous une présentation beaucoup plus compliquée).
- Qu'en pratique il n'est pas du tout nécessaire de linéariser l'algorithme car sur les instances générées aléatoirement avec le modèle dit de longueur fixe (cf section 1.1.3) on constate toujours une complexité linéaire (nous avons donné quelques éléments permettant de comprendre pourquoi).

Encore une fois, simplicité et efficacité pratique ont fait bon ménage ! À titre de comparaison les temps de calcul de notre algorithme pour des instances avec 50000 variables sont du même ordre que ceux donnés par V. Chandru & col. pour 200 variables (les machines ne sont pas les mêmes – PC Pentium 90 Mz contre Station de travail SUN 3/60 – mais il est douteux qu'un tel écart soit justifié uniquement par le matériel utilisé). La large étude expérimentale que nous avons conduit nous a de plus permis de confirmer les résultats annoncés par O. Dubois et J.J. Hébrard dans [390] sur l'existence de phénomènes de seuil pour le Horn renommage. Nous avons aussi montré que les instances les plus difficiles de ce problème se trouvent, comme on pouvait s'y attendre, dans la zone de transition de phase. Enfin, nous avons montré que notre algorithme est aussi efficace pour déterminer si un ensemble de clauses admet un unique Horn renommage (ce qui diminue l'intérêt de l'algorithme proposé par J.J. Hébrard dans [260]).

Notons aussi que le principe de partition du modèle s'applique pour le Horn-renommage dans les mêmes conditions que pour la satisfiabilité. On peut donc montrer, comme dans [373], que la composition de deux Horn renommages est un Horn renommage. Ce qui simplifie sensiblement la notion de Base de Horn définie par J.J. Hébrard dans [259].

Pour finir, nous avons montré que le Horn renommage est en quelque sorte le problème le plus difficile qui peut être traité par un algorithme de type Davis et Putnam plus autark. Reprenant l'idée de T.J. Schaeffer [437] qui consiste à voir les clauses comme des relations, nous avons montré que les mécanismes de propagation mis en œuvre dans l'algorithme de Horn renommage ne peuvent pas être généralisés plus avant.

Chapitre 2

Toupie

Ce court chapitre est consacré au système de programmation par contraintes Toupie qui est destiné avant tout à l'analyse de programme et que j'ai réalisé au LaBRI entre 1992 et 1994. Les deux articles donnés en annexe offrent une bonne vision des travaux que j'ai effectués sur ce thème. Aussi ce qui suit ici est volontairement succinct afin d'éviter les redondances. La prochaine section présente le projet Toupie tel qu'il s'est déroulé dans le temps et introduit les sections suivantes qui reviennent informellement sur des points qu'il m'a paru intéressant de souligner. Une connaissance, au moins superficielle, de Toupie est sans doute nécessaire à leur compréhension. L'article [141] donné annexe C.2 fournit, je pense, une bonne introduction à Toupie.

2.1 Le projet Toupie en perspective

Les langages de programmation logique avec contraintes ont montré l'intérêt d'outils aussi ouverts et généralistes que possible. L'idée est de fournir à l'utilisateur un langage déclaratif de haut niveau lui permettant de combiner des primitives de résolution de contraintes de bas niveau (et donc très efficaces, cf 3.1 pour une discussion sur ces systèmes). Pour résoudre tel ou tel problème particulier, il peut ainsi construire des prototypes rapides, efficaces, faciles à modifier et à maintenir. Il est évidemment tentant d'utiliser ces langages dans le cadre de l'analyse de programmes en général et de l'analyse de systèmes de processus communicants en particulier. Plusieurs propositions ont été faites en ce sens (voir notamment [371, 209]). Toutefois, les systèmes de programmation par contraintes disponibles sont conçus essentiellement pour résoudre des problèmes d'optimisation combinatoire, c'est à dire des problèmes dont la résolution consiste à trouver une solution à un système de contraintes, éventuellement la meilleure suivant un certain critère. Ils ne permettent, du moins directement, ni quantification universelle, ni quantification du second ordre. Malheureusement, et la quantification universelle, et les calculs de points fixes d'équations ensemblistes sont utiles voire nécessaires pour modéliser les appels récursifs, obtenir les états accessibles d'un système ou exprimer des propriétés de vivacité

ou d'équité[1] [18].

Toupie permet de tels calculs.

Le projet Toupie a débuté au printemps 1992. Il est né d'une part de discussions sur l'interprétation abstraite de programmes Prolog avec les autres membres[2] de la petite équipe «programmation logique» du LaBRI[3] ; D'autre part, de l'intérêt que je portais aux travaux de l'équipe «sémantique du parallélisme» sur les systèmes de processus communicants, le modèle de A. Arnold et M. Nivat [13] et de l'outil MEC [167, 11]. Dans ce cadre, j'ai pris connaissance des résultats spectaculaires obtenus par l'équipe d'E. Clarke d'une part [84, 85, 349] et par J.-C. Madre et O. Coudert [144, 143, 153, 41, 155] d'autre part[4]. En codant symboliquement les automates à l'aide de diagrammes binaires de décision de R. Bryant [77, 70], ils ont réussi à repousser assez loin l'explosion combinatoire inhérente au calcul des états accessibles des systèmes analysés[5]. Les diagrammes binaires de décision faisaient une jonction idéale entre mes travaux sur les résolveurs de contraintes booléennes (cf section 3.1) et ce nouveau centre d'intérêt.

J'ai alors pensé que le μ-calcul – le langage utilisé dans [85] – peut être un bon cadre pour définir des sémantiques abstraites de Prolog. Plus précisément, la sémantique au point fixe de J.W. Loyd [330] d'un programme logique peut-être décrite par des termes du μ-calcul sur l'univers de Herbrand du programme. Par conséquent, si l'on abstrait l'univers de Herbrand en un domaine fini D, le μ-calcul sur D permet de calculer les substitutions réponses abstraites du programme. J'ai alors développé une première version de Toupie dédiée à ce type d'analyses. Comme beaucoup de domaines abstraits intéressants comportent plusieurs éléments, j'ai étendu les diagrammes binaires décision en des diagrammes n-aires (cf section 2.5). Ce travail en collaboration avec M.-M. Corsini et K. Musumbu (pour la partie interprétation abstraite) est paru dans [135]. Il a été grandement amélioré lors de la venue de B. Le Charlier à Bordeaux [132, 133] (ce dernier article est donné annexe C.1, voir aussi section 2.3).

Nous avons ensuite, avec A. Griffault, utilisé Toupie pour prouver des algorithmes d'exclusion mutuelle [134] en particulier ceux de G.L. Peterson [386] et de J.E. Burns [86]. Ce changement de domaine m'a convaincu que Toupie devait être un outil généraliste, un système de programmation par contraintes. Cette évolution s'est faite par étapes [136, 139,

[1] Certaines de ces propriétés peuvent néanmoins être calculées «à la volée»[265]. L'intégration de cette technique et de la programmation par contraintes est une piste intéressante car elle permet d'étudier plusieurs états du programme à la fois [371, 209].

[2] M. Billaud, M.-M. Corsini et K. Musumbu.

[3] Nous avons organisé un colloque sur ce thème en septembre 1991 (JTASPEFL'91 [347], devenu WSA'92 [346], puis WSA'93 (à Padoue) [161] et finalement SAS'94 (à Namur) [102]).

[4] Pour être équitable on doit sans doute aussi citer ceux de H. Touati et col. [477, 476].

[5] En général, dans ce type d'études, chaque composant du système analysé est modélisé par un automate d'états fini et le comportement global du système est décrit par le produit de ces automates individuels. Tous les états du produit ne sont pas accessibles à partir des états initiaux. Une première tâche est donc de calculer l'ensemble des états accessibles, qui peut être très gros (plusieurs millions d'éléments). La fonction de transition de l'automate global peut aussi être donnée directement, comme c'est souvent le cas lorsq'on analyse des circuits séquentiels [144, 143, 153, 41, 155]. Mais cela ne change pas la nécessité de calculer les états accessibles.

422, 141], au fur et à mesure de l'introduction de nouvelles fonctionnalités, notamment du résolveur de contraintes arithmétiques et de la représentation par diagrammes de décision compressés (voir l'article [141] donné annexe C.2, la section 2.2 pour une discussion sur le modèle de calcul et la section 2.5 pour une discussion sur l'implémentation de Toupie).

Dans un premier temps, j'ai codé directement en Toupie la description dans le modèle de A. Arnold et M. Nivat [13] des algorithmes cités ci-dessus ainsi que d'un certain nombre d'autres exemples classiques comme le «scheduler» de R. Milner [353] ou le «dîner des philosophes» [98]. Ces expériences m'ont montré que Toupie permet, en introduisant la notion de contraintes dans la définition même des automates, de donner une autre vision de ce modèle, aussi puissante mais plus facile d'utilisation, au moins pour ce qui me concerne. Là encore, la formalisation s'est faite par étapes [73, 422, 424]. Cette question est discutée section 2.4.

Cette réflexion sur le modèle descriptif sous-jacent conduit à repenser l'architecture même de Toupie. Son évolution est donc loin d'être terminée. Quelques pistes pour le futur sont évoquées section 2.5 qui revient sur l'implémentation du système.

Pour finir, signalons que la majeure partie du travail sur Toupie s'est déroulée dans le cadre du projet inter-PRC «Langages Logiques Concurrents avec Contraintes».

2.2 Toupie = μ-calcul + contraintes

Toupie est un interpréteur pour une extension du μ-calcul propositionnel. Le μ-calcul propositionnel est un formalisme introduit par D. Park [385] pour exprimer des propriétés d'ensembles d'états et de transitions d'automates d'états finis. En plus des variables, constantes, connecteurs et quantificateurs usuels du calcul propositionnel, il permet de définir des relations comme plus grands ou plus petits points fixes d'équations. Sa syntaxe abstraite est décrite figure 2.1. Il peut être vu comme l'assembleur des logiques temporelles [12]. En effet, les opérateurs de logiques comme celle de M. Hennessy et R. Milner [262], celle de A. Dicky [183] ou CTL [199, 108] se traduisent facilement dans cette logique [200, 12]. De plus, on peut aussi y exprimer divers équivalences comportementales [468]. Il existe en fait plusieurs présentations du μ-calcul dans la littérature (voir par exemple [385, 303, 200]). Les différences tiennent essentiellement au type des automates considérés. Un point clef étant de savoir si les transitions sont étiquetées ou non. Les auteurs travaillant avec des automates à transitions étiquetées ajoutent en général au formalisme des connecteurs caractérisant ces transitions et venant typiquement de la logique de M. Hennessy et R. Milner [262]. Toupie est proche de la présentation originale de D. Park [385] (utilisée par exemple dans [85]) et des langages de requêtes étudiés dans le cadre des bases de données [481].

Toupie est différent de ces divers formalismes, pour plusieurs raisons (la syntaxe et la sémantique dénotationelle de Toupie sont présentées en détails dans l'article [141] donné annexe C.2, aussi je n'ai souligné ici que les points de divergence) :

– Les variables individuelles de Toupie prennent valeurs dans des ensembles de constantes symboliques ou des intervalles de $\mathbb{N}$ et non simplement dans $\{0, 1\}$. Cela ne fait bien

Les termes du μ-calcul propositionnel sont définis comme suit :

$$t ::= (X = Y) \mid \neg t \mid t \vee t \mid \exists X t \mid p(\vec{X}) \mid \mu p.t$$

Dans la description ci-dessus, t est un terme, X et Y sont des variables individuelles booléennes, p est une variable de prédicat, $\vec{X}$ est un vecteur de variables individuelles. Si p est d'arité n, $\mu p.t$ dénote le plus petit point fixe de l'équation $p = t$ dans le treillis ensembliste des parties de $\{0,1\}^n$. Les règles usuelles de dualité s'appliquent :

$$\begin{aligned} (f \wedge g) &= \neg(\neg f \wedge \neg g) \\ \forall X f &= \neg \exists X \neg f \\ \nu p.f &= \neg \mu p. \neg f[p \leftarrow \neg p] \end{aligned}$$

FIG. 2.1 – Syntaxe abstraite du μ-calcul propositionnel.

entendu aucune différence quant à la puissance d'expression mais c'est beaucoup plus agréable à manipuler.

- Les atomes du μ-calcul propositionnel sont de la forme $X = Y$, où X et Y sont des variables booléennes. Les atomes de Toupie sont des contraintes n-aires sur des variables individuelles. Toupie intègre en fait un résolveur d'inéquations linéaires en nombres entiers bornés (cf section 2.5). Là encore, pas de différence du point de vue théorique mais une plus grande souplesse du point de vue pratique et des gains d'efficacité non négligeables.
- L'introduction de la notion de contraintes permet d'envisager de traiter des variables individuelles continues et/ou sur des algèbres *ad hoc*, voir section 2.5 pour une discussion sur ce sujet.
- Le μ-calcul ne permet pas *a priori* de nommer des relations. Un programme Toupie au contraire est un système d'équations de points fixes définissant des prédicats[6]. Cette extension aussi est facile et utile, et a été étudiée par R. Cleaveland, M. Klein et B. Steffen dans [110].
- En général, les auteurs se servent de formules du μ-calcul pour exprimer des propriétés sur un automate d'états fini donné au préalable (autrement dit, ses propriétés sont interprétées sur cet automate). Toupie est utilisé à la fois pour écrire des propriétés et pour coder l'automate étudié. En ce sens, Toupie est donc vraiment un langage de programmation. Cette généralité n'est pas contradictoire avec de bonnes performances comme le montre les comparaisons avec les outils spécialisés décrits dans [63, 202] (cf [139]).

L'originalité de Toupie vient de ce qu'il se situe à la confluence de trois courants de

[6]Les systèmes d'équations de points fixes sont définis sur le treillis ensembliste du produit cartésien des domaines des variables. Par le théorème de Knaster-Tarksi [474], ils admettent toujours une solution calculable par itération finie (pour peu que les formules considérées soient monotones).

pensée :

L'analyse de systèmes de transitions. Des travaux sur la vérification symbolique de circuits et de programmes, Toupie prend son modèle de calcul (le μ-calcul) et est donc proche d'outils comme Cesar [402], MEC [11], le Concurrency Workbench [111], Priam [335] ou SMV [349]. La plupart de ces outils ou de leurs successeurs permettent aujourd'hui un codage symbolique des automates à l'aide de diagrammes binaires de décision.

La programmation logique avec contraintes. De la programmation logique avec contraintes, et plus spécialement de langages comme CHIP [489], Toupie prend la volonté d'être le plus universel possible ainsi que les techniques de résolution de contraintes. Par rapport à ces langages, il intègre la quantification universelle et les calculs de points fixes. Ces derniers, qui fournissent une forme de quantification sur les relations, sont tout à fait essentiels pour modéliser les appels récursifs dans les langages de programmation [156, 157]. Bien entendu, la puissance d'expression obtenue en incluant dans le langage une quantification d'ordre supérieur se paye en termes d'efficacité des algorithmes de résolution. Toutefois, l'utilisation de diagrammes de décision pour coder les relations permet de résoudre des problèmes difficiles en des temps tout à fait satisfaisants (en fait, le principal problème est plutôt celui de la place mémoire). Signalons à ce sujet que les diagrammes binaires de décision sont aussi utilisés en programmation logique avec contraintes (cf section 3.1).

Les bases de données relationnelles. En fait, comme je l'ai dit plus haut, Toupie ressemble beaucoup aux langages requêtes et les résultats théoriques obtenus sur ces ceux-là valent donc en général pour celui-ci. C'est par exemple à N. Immerman [278] que l'on doit le résultat qu'un terme de μ-calcul contenant plusieurs points fixes imbriqués peut toujours être traduit en un terme ne contenant qu'un seul point fixe[7].

2.3 Interprétation abstraite de programmes logiques

Ainsi qu'il a été dit plus haut, la première utilisation de Toupie a été l'implémentation d'interprétations abstraites de programmes logiques (voir l'article [133] donné annexe C.1). L'interprétation abstraite est un cadre général pour analyser des programmes. Ce cadre a été défini par P. Cousot et R. Cousot dans [158, 159]. L'idée principale est d'exécuter un programme abstrait dont la sémantique est une abstraction de la sémantique du programme concret à analyser et de tirer de cette exécution abstraite des informations sur les exécutions potentielles du programme concret. Les objectifs sont multiples : preuves de programmes, optimisations des compilateurs, ...

La programmation logique se prête particulièrement bien à ce genre d'exercices, d'une part parce que la sémantique des langages de programmation logique est en général bien définie [330] et d'autre part parce qu'il y a grand besoin de compiler ces langages aussi

[7] Toupie permet toutefois l'imbrication *ad libitum* de points fixes pour des raisons d'efficacité : la transformation en une hiérarchie de points fixes ne peut être faite (du moins à ma connaissance) qu'en augmentant l'arité des prédicats, ce qui est évidemment fort coûteux.

efficacement que possible pour les rendre compétitifs avec les langages impératifs. Aussi, une littérature abondante existe sur le sujet (voir, par exemple, [1, 495, 160, 285, 193, 3, 232, 100] pour un aperçu de ces nombreuses références). Toutefois, à l'exception notable de Aquarius Prolog [495], aucun compilateur Prolog n'intègre, à ma connaissance, ces techniques.

L'idée d'utiliser des langages de contraintes pour implémenter des interprétations abstraites de programmes logiques a été proposée dans [115, 234]. Dans ces deux articles, le domaine considéré est le domaine à deux valeurs `prop`[8]. Ils montrent comment utiliser les contraintes booléennes pour l'implémenter (aucune expérimentation n'est toutefois rapportée). Dans [133], nous avons généralisé cette idée à des domaines abstraits plus complexes, mais toujours ≪downward closed≫ [100], c'est à dire tels que la substitution réponse abstraite σ_{out} pour une substitution d'entrée abstraite σ_{in} s'obtient en appliquant σ_{in} sur la substitution réponse σ_{out}^{gen} obtenue pour la substitution d'entrée σ_{in}^{gen} laissant toutes les variables libres. Autrement dit, si C_{out}^{gen}, C_{in} et C_{out} sont respectivement les contraintes codant σ_{out}^{gen}, σ_{in} et σ_{out}, on a $C_{out} = C_{out}^{gen} \wedge C_{in}$. Cette approche peut donc être reliée aux travaux sur la compilation logique [264] (en abstrayant la syntaxe des programmes comme dans [219, 234]).

L'utilisation de Toupie a montré qu'il est possible d'implémenter des interprétations abstraites efficacement (sur le banc d'essais proposé dans [101], voir aussi [112]) et a participé à la clarification par B. Le Charlier de quelques points théoriques [100] (notamment la notion de domaine ≪downward closed≫) . En dépit de ces résultats intéressants, et qui sans doute s'étendent à d'autres types de langages, j'ai abandonné ce thème de recherche, faute de motivation et d'application pratique. Cela étant, il a participé au développement de Toupie et à ce titre mérite la place qui lui est donné dans ce mémoire.

2.4 Modélisation et vérification

J'ai défini et implémenté Toupie avec le double objectif de me former aux techniques utilisées en modélisation et vérification de systèmes de processus communicants et de voir ce que peut apporter le paradigme programmation par contraintes dans ce domaine. Naturellement, cette démarche s'est appuyée sur de nombreuses discussions avec les membres de l'équipe MVTSI[9] du LaBRI à propos des mérites et des défauts du modèle de A. Arnold et M. Nivat [13] et de l'outil MEC [167, 11]. Dans ce modèle, qui a l'immense avantage d'être très simple, l'essentiel de la sémantique est mise sur les étiquettes des transitions et les vecteurs de synchronisation. Il y a de fortes motivations théoriques à cela, liées précisément à la simplicité des descriptions, et aussi de fortes motivations pratiques, liées

[8] `prop` est le treillis booléen construit sur les deux valeurs `g` (pour ≪ground≫) et `ng` (pour ≪not ground≫). Un terme de Herbrand est abstrait en `g` s'il est clos, c'est à dire s'il ne contient pas de variable libre et par `ng` dans le cas contraire. `prop` sert donc de déterminer les points de programmes auxquels la substitution courante contient des termes clos. Ce qui permet de remplacer l'unification par un algorithme de reconnaissance de structures (≪pattern matching≫) plus efficace en particulier dans le cadre de la machine abstraite de Warren [504, 4].

[9] Modélisation, Vérification et Tests de Systèmes Informatisés.

à la façon dont les automates sont codés dans MEC[10]. Mais cette méthodologie a aussi des inconvénients. Par exemple, la simple description d'un compteur dont la valeur conditionne certaines transitions peut atteindre rapidement une taille redoutable.

Le codage symbolique des automates via les diagrammes de décision ainsi que la notion de contraintes permettent de résoudre partiellement ces inconvénients[11]. Plutôt que de considérer des états et des transitions individuelles, on raisonne sur des ensembles d'états et de transitions, dans l'esprit des travaux de V. Saraswat [435, 436] qui généralisent la programmation impérative – «store as value» – par la programmation par contraintes – «store as constraint». On définit donc la notion d'automates à contraintes [72, 73, 424] (voir aussi [208, 209]). Un *automate à contraintes* est un sextuple $\langle D, n, m, S, T, I\rangle$, où

- D est un ensemble de valeurs, appelé *domaine*,
- n et m sont deux entiers positifs ou nuls,
- S est une *contrainte d'état*, c'est à dire un sous-ensemble de D^n,
- T est une *contrainte de transition*, c'est à dire un sous-ensemble de $D^n \times D^m \times D^n$,
- I est un ensemble d'*états initiaux* c'est à dire un sous-ensemble de D^n.

Les états d'un tel automate sont donc des vecteurs (de taille n) de variables prenant valeur dans D et ses transitions sont étiquetées par des vecteurs (de taille m) de variables prenant elles aussi valeur dans D. La contrainte d'état S peut être vu comme un invariant que doit vérifier chacun des états de l'automate.

Deux états $\vec{s} = \langle s_1, \ldots, s_n\rangle$ et $\vec{t} = \langle t_1, \ldots, t_n\rangle$ sont respectivement la source et le but d'une transition si et seulement si il existe une étiquette $\vec{l} = \langle l_1, \ldots, l_m\rangle$ telle que $\langle \vec{s}, \vec{l}, \vec{t}\rangle \in T$.

L'ensemble des états accessibles d'un automate à contraintes $\mathcal{A}$, noté $acc(\mathcal{A})$, est défini comme le plus petit point fixe de l'équation ensembliste suivante :

$$acc(\mathcal{A}) = (S \cap I) \ \cup \ \ \{\vec{t} \in S | \exists \vec{s} \in acc(\mathcal{A}), \exists \vec{l} \in D^m, \langle \vec{s}, \vec{l}, \vec{t}\rangle \in T\}$$

La composition d'automates à contraintes se définit naturellement de la façon suivante :

Soient $\mathcal{A}_1 = \langle D_1, n_1, m_1, S_1, T_1, I_1\rangle, \ldots, \mathcal{A}_k = \langle D_k, n_k, m_k, S_k, T_k, I_k\rangle$, k automates à contraintes, S_g un sous-ensemble de D^n et T_g un sous-ensemble de $D^n \times D^m \times D^n$. On pose $D = D_1 \cup \ldots \cup D_k$, $n = n_1 + \ldots + n_k$, $m = m_1 + \ldots + m_k$, et pour $R_1 \subseteq D^{l_1}$ et $R_2 \subseteq D^{l_2}$, on note $R_1 \bowtie R_2$ l'ensemble $\{\langle \vec{s_1}, \vec{s_2}\rangle \in D^{l_1+l_2}; \vec{s_1} \in R_1, \vec{s_2} \in R_2\}$. L'automate produit $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_k$ est le sextuple $\langle D, n, m, S, T, I\rangle$, où

- $S = (S_1 \bowtie \ldots \bowtie S_k) \cap S_g$, S_g est donc un invariant global du produit.
- $T = (T_1 \bowtie \ldots \bowtie T_k) \cap T_g$, où par abus de notation on réordonne les champs de la jointure des T_i de façon à garder l'ordre «variables de l'état source / variables de l'étiquette / variables de l'état but» et T_g est une contrainte globale de transition servant typiquement à exprimer les synchronisations.
- $I = I_1 \bowtie \ldots \bowtie I_k$.

[10] C'est à dire explicitement, la taille du codage étant proportionnelle à la taille de l'automate.

[11] Bien entendu, elle en introduit d'autres liés notamment à l'impossibilité de se servir de l'arsenal de l'algorithmique sur les graphes (voir plus loin).

Comme on le voit, cette définition est très proche de celles des automates booléens utilisés pour la preuve de circuits séquentiels (et implémentés par des outils comme SIAM [142] ou BAC [247]). Son apport est donc essentiellement méthodologique. En effet, par rapport à la définition classique d'automates et dans le cas où D est fini, l'introduction de contraintes n'apporte rien de plus que des facilités d'écriture et une meilleure intuition des calculs effectués (dans le cadre d'une représentation symbolique des automates). En particulier, elle permet de passer facilement des transitions individuelles entre états à des ensembles de transitions entre ensembles d'états. On pourrait d'ailleurs définir les états eux-mêmes comme des contraintes, c'est à dire des sous-ensembles de D^n. L'ensemble des états accessibles devant alors être défini via la relation d'équivalence engendrée par T. Le cas où D est infini mériterait qu'on s'y intéresse en détails.

À titre d'illustration, j'ai montré dans [424], sur l'exemple proposé dans [17] – un régulateur de vitesse de croisière pour une automobile – que des spécifications comportementales ≪Software Cost Reduction≫ (cf [261] pour une description de ce formalisme) se traduisent très naturellement en termes d'automates à contraintes (alors que leur traduction dans le modèle de A. Arnold et M. Nivat n'est pas très agréable).

En résumé, les automates à contraintes semblent définir un cadre très souple pour modéliser des systèmes de processus communicants. Ils peuvent être vue comme une méthodologie associée à Toupie. Mon grand regret est de n'avoir pas eu le temps à ce jour d'avoir fait subir le baptème du feu à cette méthodologie sur un problème industriel complet. Pour être équitable, il faut dire ici que le codage symbolique de automates n'a pas que des avantages. Principalement parce qu'il ne permet pas d'utiliser les nombreux algorithmes de manipulation des graphes. En particulier, il devient délicat de déterminer les *scenarii* conduisant à un évènement redouté (c'est à dire un état particulier du produit synchronisé). Le problème n'est donc pas de choisir entre codage explicite et codage symbolique mais de les faire coopérer.

2.5 Quelques réflexions sur l'implémentation

2.5.1 Diagrammes de Décision

Le point clef de l'implémentation de Toupie, d'où vient son efficacité pratique, est l'utilisation de diagrammes de décision pour coder les relations. Cette structure de données est une extension pour les domaines finis des diagrammes binaires de décisions de R. Bryant [5, 77, 70, 78]. La principale différence entre les deux représentations tient à ce que les nœuds des diagrammes de décision sont n-aires, où n est le cardinal du domaine de la variable étiquetant le nœud, et non binaires. Cette extension préserve toutes les bonnes propriétés des diagrammes binaires de décision (voir l'article donné annexe C.2 pour plus de détails) :

- La représentation est canonique, ce qui permet, entre autres, de tester si deux relations sont identiques en temps constant (pour le test d'arrêt des calculs de point fixe).
- Le complémentaire d'une relation est calculé en temps constant en mettant un dra-

peau sur chaque branche sortante pour indiquer si le diagramme de décision pointé doit être considéré positivement où négativement [46, 334, 356].
- On peut effectuer pratiquement la plupart des calculs ($\vee$, $\wedge$, ...) grâce à une unique opération de composition $ITE(\beta, \gamma, \delta) \stackrel{\text{def}}{=} (f \wedge g) \vee (\neg f \wedge h)$. Cette opération est de même complexité pour les diagrammes n-aires que pour les diagrammes binaires, c'est à dire en $\mathcal{O}(|\beta| \times |\gamma| \times |\delta|)$ [70].
- Les opérations de quantifications et de projections sont effectuées de façon similaire et sont de même complexité [70].
- On peut utiliser les mêmes mécanismes de gestion mémoire et de mémorisation des opérations [70, 356].

L'idée de coder des fonctions discrètes à l'aide de diagrammes de décision est due, autant que je puisse en juger, à J.P. Billon [46]. W. Büttner, dans [80] et [81], a proposé de coder de cette façon des formules de logiques multivaluées. Toutefois, aucune de ces deux implémentations n'utilise pleinement les techniques actuelles d'implémentation des diagrammes binaires de décision (tables de hachage, connecteur ITE), et pour cause : l'article de R. Bryant et cols. [70] date de 1990. L'extension des diagrammes binaires aux diagrammes n-aires sous forme presque ≪moderne≫ a été proposée dans [465] mais, jusqu'à une date récente (c.à.d. en 1994), je ne connaissais pas ces travaux sur la vérification de circuits. Signalons aussi [498] qui propose de coder des contraintes sous forme d'automates d'états finis et la structure de données dite des "Arbres Partagés" proposée par D. Zampunieris [514] qui a été implémentée avec succès dans différents outils de vérification de modèle [513, 216, 384].

Quand on travaille avec des variables ayant d'assez grands domaines (cela peut être en particulier le cas avec les variables numériques), de nombreux fils successifs d'un même nœud peuvent être égaux. On peut dans ce cas compresser un peu plus la représentation. L'idée est d'étiqueter les branches sortantes d'un nœud non par des constantes mais par des intervalles [140, 422]. À condition que deux fils successifs et égaux soient réunis, cette nouvelle représentation a, elle aussi, toutes les bonnes propriétés énoncées plus haut. Elle permet de plus d'accélérer les calculs (plusieurs cas étant traités en même temps). Cette représentation est originale (au moins jusqu'à nouvel ordre).

Au début du projet Toupie, j'ai implémenté un module de diagrammes de décision pour tester expérimentalement les avantages éventuels de cette extension. Du point de vue de l'efficacité, l'expérience n'est pas réellement probante : les temps de calculs et l'occupation mémoire sont tout à fait comparables avec ceux des diagrammes binaires. En revanche, la représentation directe des valeurs des variables permet une intégration beaucoup plus simple des mécanismes de résolution de contraintes. De plus, la représentation par intervalles permet d'envisager de traiter des domaines non discrets. Je reste donc convaincu que cette piste est intéressante.

2.5.2 Résolution de systèmes de contraintes

Toupie intègre un résolveur de systèmes d'inéquations linéaires en nombres entiers. Ici, résoudre veut dire construire le diagramme de décision qui code toutes les solutions du système. Pour ce faire, on parcourt implicitement un arbre de décision (en utilisant la technique classique énumération implicite/propagation, voir par exemple [489] pour une discussion détaillée) et on construit le diagramme de décision associé de bas en haut. Toupie se limite aux systèmes d'inéquations linéaires uniquement pour des raisons d'efficacité. Construire le diagramme de décision associé à un tel système n'est pas, en théorie, plus complexe que construire celui associé à un système quelconque : peu ou prou, on doit parcourir un arbre de décision. Il n'en va pas en revanche de même en pratique car on dispose de filtrages efficaces permettant d'élaguer l'arbre de décision dans le premier cas, mais pas dans le second.

Le principe de propagation utilisé dans Toupie est celui de D.L. Waltz [502, 503] (voir aussi [318]). Il consiste à maintenir, pour chaque variable, une valeur minimum et une valeur maximum. Chaque fois que, dans une inéquation, l'intervalle de variation d'une variable est modifié, on recalcule ces valeurs pour toutes les autres variables apparaissant dans l'inéquation. On propage ainsi les modifications sur tout le système jusqu'à atteindre un point fixe.

L'intégration des diagrammes de décision et des techniques de résolution de contraintes est originale, même si ces techniques coexistent dans de nombreux travaux, à commencer par CHIP [489]. D'un certain point de vue, elle s'est arrêtée au milieu du gué (parce que l'effort de programmation pour aller plus loin serait très important). Il serait, à mon avis, intéressant de traiter les inéquations non linéaires en nombres entiers et d'autres types de contraintes, comme les algèbres de Allen [6], ou les systèmes d'inéquations linéaires en nombres flottants ou rationnels (en utilisant des approximations par enveloppes convexes des disjonctions de systèmes [174, 175, 246]).

2.5.3 Discussion

Pour finir, signalons que la version 0.26 de Toupie [417] est distribuée sur le site ftp de Bordeaux. Elle constitue un logiciel de taille appréciable (environ 20000 lignes de code C écrites dans un style «orienté objet»), contenant un forte concentration d'algorithmes non triviaux. Comme pour Pardi (cf section 1.2.2), il est intéressant d'avoir un regard critique sur l'ensemble du logiciel. Quelques remarques peuvent donc être faites :

- Toupie est découpé en couche, chaque couche pouvant *a priori* être utilisée séparément. Néanmoins, il serait intéressant de concevoir systématiquement, pour chacune des couches, une double interface : une consistant en un langage de haut niveau, l'autre consistant par exemple en un ensemble de classes C++ pouvant être intégrées dans n'importe quelle application. Cette approche a été adoptée avec succès par J.-F. Puget pour Ilog-Solver [397].
- Il faudrait permettre à l'utilisateur de manipuler directement les structures de données pour programmer ses propres algorithmes. Par exemple, pour implémenter des calculs

d'implicants premiers (sans passer par des diagrammes binaires de décision comme dans [321]), ou des opérateurs comme `constrain` et `restrict`, introduits par J.-C. Madre et O. Coudert et qui sont si utiles en preuve de circuits [144, 155, 85], ou encore pour travailler sur des diagrammes de décision "libres", c'est à dire dans lesquels on a relaché les contraintes sur l'ordre des variables (voir [231, 506, 71, 456, 40] pour divers travaux sur les *braching programs* et leurs avantages sur les diagrammes de décision ordonnés et réduits). L'idée étant de fournir un jeu de primitives élémentaires permettant d'implémenter ces algorithmes. Cette approche a été proposée dans [492] pour les langages concurrents avec contraintes (cf section 3.1 pour une discussion plus détaillée sur cette question). Plus délicat serait sans doute le déclenchement (par une programmation à haut niveau) du réordonnancement dynamique des variables utilisé aujourd'hui dans la plupart des paquetages BDD [279, 281, 251, 431, 204, 198].

– Il serait aussi intéressant de choisir comme interface un langage applicatif incluant dans ces mécanismes de bases le calcul de point fixe au travers de fonctions de *memoization* (encore appelée *tabulation*) des résultats [47, 471]. L'avantage serait double. D'une part cela permettrait d'adopter une syntaxe familière, d'autre part et plus fondamentalement, cela permettrait d'inclure naturellement des calculs procéduraux (ou fonctionnels) à l'intérieur des mécanismes logiques.

À la lumière des remarques précédentes, je pense qu'il serait donc intéressant de refondre Toupie. C'est un gros projet auquel je m'attaquerais peut-être dans les prochains mois. Il reste que Toupie dans sa version actuelle a permis de montrer l'intérêt d'appliquer à la vérification de systèmes les idées directrices du paradigme programmation par contraintes.

Chapitre 3

Applications des fonctions booléennes

Ce chapitre regroupe deux sections traitant de l'utilisation pratique des méthodes de codage et de manipulation des fonctions booléennes. La première présente les travaux que j'ai effectués sur l'intégration de résolveurs de contraintes booléennes dans des langages de programmation logique avec contraintes. La seconde concerne ceux que j'ai réalisés dans le cadre la sûreté de fonctionnement de systèmes industriels.

3.1 Contraintes booléennes dans Prolog

La programmation par contraintes connaît depuis quelques années un développement important. Des langages comme CLP($\mathcal{R}$) [282, 284], PrologIII [120, 121], CHIP [186, 489] et CAL [2, 432, 433] ont permis à ce paradigme de quitter les laboratoires pour l'industrie, où de réels succès ont été obtenus (voir par exemple [367] où plusieurs articles montrent comment des systèmes de programmation par contraintes se sont avérés plus performants que des codes «recherche opérationnelle» traditionnels). Les avantages de ces systèmes sont bien connus : utilisant des langages déclaratifs comme interface, ils permettent un prototypage rapide ; souples, ils facilitent les modifications et la maintenance ; ouverts, ils permettent à l'utilisateur de définir ses propres mécanismes de contraintes et assurent de ce fait une efficacité qui ne cède en rien aux codes classiques (voir [117, 32, 490, 389, 367] et surtout [283] pour une vue d'ensemble récente et une bibliographie assez complète sur ce paradigme).

Le choix d'un système de contraintes et son intégration dans un langage de haut niveau posent deux questions :

- Pourquoi : quel type d'applications vise-t-on ?
- Comment : quelles structures de données, quels algorithmes de résolution, quelles interfaces avec le langage hôte ?

Ce sont là deux problèmes à la frontière entre l'ingénierie logicielle et la recherche. En la matière, le rôle des chercheurs est sans doute de débroussailler le terrain, d'explorer de nouvelles voies, de dégager les bons concepts, de les expérimenter au travers de prototypes, bref, d'étudier la faisabilité des systèmes. C'est ce que j'ai essayé de faire pour les contraintes

booléennes, lorsque j'étais au Groupe d'Intelligence Artificielle de Marseille puis dans le cadre du groupe ≪Métheol≫ et de l'opération ≪Organon≫ du PRC-PAOIA[1] et enfin du projet inter-PRC ≪Langages Concurrents avec Contraintes≫[2]. La première partie de ce chapitre présente ces travaux. La section 3.1.1 fait quelques rappels sur la programmation logique avec contraintes. La section 3.1.2 expose les fonctionnalités communément attendues d'un résolveur de contraintes. La section 3.1.3 présentent les différents résolveurs proposés dans la littérature (y compris les miens). Finalement, j'ai essayé dans la section 3.1.4 de dégager ce que je pense être la bonne façon d'introduire des contraintes booléennes dans un langage de haut niveau.

3.1.1 Un peu d'histoire

Un premier pas vers l'intégration de la notion de contraintes dans Prolog a été réalisé avec PrologII [118] pour lequel A. Colmerauer a remplacé le concept d'unification par celui de résolution de systèmes d'équations et de diséquations ($\neq$) sur les arbres rationnels [119]. En 1987, J. Jaffar et J.-L. Lassez ont introduit la notion de contraintes numériques dans le modèle théorique de Prolog [282]. L'idée était de palier la difficulté bien connue de traiter les opérations arithmétiques dans Prolog en ajoutant à sa machinerie un résolveur de systèmes d'inéquations linéaires en nombres flottants (un algorithme du simplexe) et en permettant à l'utilisateur de conditionner l'application d'une règle à la satisfaction d'un tel système.

Un programme CLP($\mathcal{R}$) — CLP pour *Constraint Logic Programming* et $\mathcal{R}$ pour $\mathcal{R}$*eal numbers* (voir [284] pour une présentation de ce système) — est donc un ensemble de règles de la forme :

$$h \leftarrow S, t_1, \ldots, t_n$$

où h, t_1, ..., t_n ($n \geq 0$) sont des atomes Prolog ≪classiques≫ et S est un système de contraintes, c'est à dire, dans le cas de CLP($\mathcal{R}$), un système d'inéquations linéaires portant sur un sous-ensemble variables apparaissant dans h, t_1, ..., t_n.

Le moteur d'inférence CLP($\mathcal{R}$) peut être schématiquement décrit de la façon suivante :

$$(t) \quad : \quad \langle S_t \,\square\, a_0, a_1, \ldots, a_k \rangle \tag{3.1}$$

$$h \leftarrow S_r, t_1, \ldots, t_n \tag{3.2}$$

$$(t+1) \quad : \quad \langle S_t \cup S_r \cup \{a_0 = h\} \,\square\, t_1, \ldots, t_n, a_1, \ldots, a_k \rangle \tag{3.3}$$

La ligne 3.1 décrit le but à l'étape t de la résolution. Il est composé de la liste d'atomes a_0, a_1, ..., a_k et du système de contraintes S_t. Pour passer à l'étape $t+1$, on choisit de façon non déterministe une règle du programme (ligne 3.2) et le but courant devient celui décrit ligne 3.3. $\{a_0 = h\}$ dénote l'ensemble de contraintes générées par l'unification du premier atome du but 3.1 et de la tête de la règle 3.2. Le but courant est résolu quand sa

[1] Programme de Recherches Coordonnées Programmation Avancée et Outils pour l'Intelligence Artificielle

[2] Ce projet c'est déroulé du début de l'année 1993 à la fin de l'année 1995.

liste d'atomes est vide. La solution est alors obtenue en projetant le système de contraintes courant sur les variables apparaissant dans la requête. Il y a échec dans la résolution quand on détecte que le système de contraintes est insatisfiable (ce qui est en particulier le cas quand aucune règle ne peut être utilisé pour résoudre a_0). Le non déterminisme est géré par un simple retour arrière permettant d'explorer toutes les branches de l'alternative. CLP($\mathcal{R}$) intègre donc deux résolveurs de contraintes : un algorithme d'unification sur les termes Prolog et l'algorithme du simplexe déjà mentionné. Pour une présentation plus formelle des sémantiques des langages de programmation logique avec contraintes (qui sont en général des extensions de la sémantique de Prolog [330]), voir [283] déjà cité.

D'un point de vue pratique, un exemple assez plaisant de ce que permet CLP($\mathcal{R}$) est donné par le programme suivant qui calcule les versements mensuels du remboursement d'un emprunt [284] :

```
mortgage(P,T,IR,B,MP) :-
    T > 0, T <= 1, B = P*(1+T+IR) - T*MP.
mortgage(P,T,IR,B,MP) :-
    T > 1, mortgage(P*(1+IR)-MP,T-1,IR,B,MP).
```

Les paramètres sont le montant principal `P`, le nombre de remboursements `T`, le taux d'intérêt mensuel `IR`, le solde pour tout compte `B` et le montant des mensualités `MP`. L'intérêt de ce programme est qu'il peut être exécuté dans presque tous les sens : étant donnés un montant emprunté, un taux d'intérêt et un nombre de mensualités, on peut calculer le montant des mensualités ; étant donnés un montant emprunté, un taux d'intérêt et le montant des mensualités, on peut calculer le nombre de mensualités nécessaires au remboursement, et ainsi de suite. On renoue là avec l'aspect déclaratif qui a fait le succès de la programmation logique.

En fait, dans [282], J. Jaffar et J.-L. Lassez ont proposé un cadre général pour introduire des systèmes de contraintes dans Prolog. Ils ont défini une famille de langages CLP($\mathcal{X}$), où $\mathcal{X}$ peut être n'importe quel formalisme ou ensemble de formalismes permettant de poser et de résoudre des systèmes de contraintes.

À la suite de CLP($\mathcal{R}$), plusieurs systèmes de programmation logique avec contraintes ont été créés. De la première génération, citons PrologIII [120, 121], CHIP [186, 489] et CAL [2, 432, 433]. Prolog III et CHIP intègrent tous les deux un résolveur de systèmes d'inéquations linéaires en nombres rationnels. PrologIII intègre aussi un résolveur de contraintes sur les listes. Le succès de CHIP est en grande partie du à son résolveur de contraintes portant sur des variables prenant valeur dans des intervalles entiers finis et qui est l'héritier des travaux de J.-L. Laurière [313]. CAL permet de calculer dans le corps des polynômes à coefficients en nombres flottants. Chacun de ces trois systèmes intègre un résolveur de contraintes booléennes, mais comme nous le verrons plus loin ces résolveurs sont bâtis sur des principes totalement différents les uns des autres.

De la seconde génération, citons `clp(FD)` [182], `cc`($\mathcal{FD}$) [492, 494], Flang [339]. Ces trois systèmes sont plus ou moins dérivés de CHIP et ne permettent pas directement la résolution de contraintes booléennes. CLP(BNR) [370], en revanche, en intégrait un, mais il a été supprimé par la suite au profit d'un traitement des contraintes booléennes grâce à de l'arithmétique sur les intervalles.

La morale de cette courte histoire est que les résolveurs de contraintes booléennes embarqués «en dur» ont été petit à petit abandonnés au profit de résolveurs méta-programmés. La raison la plus souvent invoquée pour justifier cet abandon est le manque d'applications «réelles» des contraintes booléennes. Je pense qu'il s'agit là d'une mauvaise raison qui résulte d'une part de l'ignorance ou du manque d'intérêt pour ce à quoi sert l'algèbre de Boole dans la pratique des ingénieurs, d'autre part de ce que le cadre CLP($\mathcal{X}$) fixé dans [282] est trop étroit pour inclure des contraintes booléennes «utiles». Le reste de cette section est destiné à étayer cette thèse que j'ai défendu, sans grand succès il est vrai, dans la communauté programmation logique.

3.1.2 Fonctionnalités d'un résolveur de contraintes

Le «cahier des charges» d'un résolveur de contraintes pour un langage CLP($\mathcal{X}$) contient, *a priori*, trois fonctionnalités élémentaires induites par la sémantique opérationnelle ébauchée plus haut :

1. Permettre de construire «facilement» l'union de deux systèmes de contraintes et de revenir sur cette construction lors du retour arrière.
2. Déterminer la satisfiabilité d'un ensemble de contraintes. Le test de satisfiabilité conditionne l'application de la règle sélectionnée à l'étape t de la résolution (lignes 3.1, 3.2, 3.3).
3. Projeter un système de contraintes sur un sous-ensemble des variables apparaissant dans ce système. Cette opération à deux objectifs : présenter les solutions d'une requête (en éliminant, autant que faire ce peut, les variables introduites en cours de résolution) et simplifier l'ensemble de contraintes courant.

   À ces trois fonctionnalités, on peut en ajouter deux autres :

4. Détecter les variables astreintes à prendre une valeur unique. Cette opération est particulièrement utile pour mettre en œuvre le prédicat prédéfini `geler` de Prolog II [235]. Elle sert aussi à simplifier l'ensemble de contraintes courant.
5. Détecter si un système de contraintes implique une contrainte donnée. Cette opération est utile dans le cadre plus général des langages concurrents avec contraintes [336, 436] où elle sert à implémenter les mécanismes de `ask` et `tell` (qui sont une généralisation des gardes des Prolog concurrents [452]). Elle sert aussi à implémenter l'opérateur de cardinalité [491] et la disjonction constructive [174, 492].

Ces fonctionnalités, admises par la plupart des auteurs [283] et formalisées par V. Saraswat [435, 436], imposent en fait un cadre relativement restrictif sur le type de systèmes de contraintes intégrables dans un langage CLP($\mathcal{X}$). En un mot, les systèmes de contraintes répondant à ce cahier des charges sont ceux dont la résolution consiste à chercher une affectation des variables satisfaisant le problème posé, ou éventuellement la meilleure affectation possible pour un critère (une fonction objective) donné(e). Les systèmes d'inéquations linéaires (en nombre flottants ou rationnels) et les contraintes sur les domaines finis rentrent effectivement dans ce cadre. De fait, les problèmes pratiques qui ont été résolus avec

succès grâce à des programmes CLP($\mathcal{X}$) sont presque tous de problèmes de recherche opérationnelle et plus particulièrement d'optimisation combinatoire (voir par exemple [184, 185, 277, 489, 494] et [283] pour plus de références). Comme nous le verrons, les applications pratiques de l'algèbre de Boole ne rentrent pas dans cette catégorie.

3.1.3 Les résolveurs de contraintes booléennes existants

Pour mieux appréhender comment le cahier des charges décrit ci-dessus a été rempli dans les différents systèmes et les problèmes que ces implémentations ont soulevés, il faut entrer dans le détail des algorithmes mis en œuvre.

Le résolveur de CAL : C'est une version modifiée de l'algorithme de Buchberger [434]. Il consiste à mettre le système de contraintes courant sous forme normale polynomiale (aussi appelée forme normale de Reed-Muller – voir [505, 75] – qui a été introduite par M. Stone [469]). Cette dernière est définie comme suit. Étant donné un ordre total sur les variables,

- 0 et 1 sont sous forme normale polynomiale.
- Si f et g sont sous forme normale polynomiale et ne contiennent pas de variable plus petite que x, alors $(x \wedge f) \oplus g$ est sous forme normale polynomiale, où $\oplus$ dénote le ou exclusif.

Cette forme normale est canonique. Les auteurs n'ont, à ma connaissance, jamais testé sur une échelle significative leur résolveur. Dans [351], un nouvel algorithme est proposé pour calculer la forme normale polynomiale d'une formule. Quelques exemples montrent que ce nouvel algorithme est plus efficace que celui décrit dans [434]. Toutefois, les temps de calculs donnés sont catastrophiques ! C'est à dire de plusieurs ordres de magnitude plus lents que l'énumération des solutions avec un algorithme du type procédure de Davis et Putnam [172]. Notons qu'un troisième algorithme de mise sous forme normale polynomiale a été proposé par J. Hsiang [273] (il est toutefois douteux qu'il soit beaucoup plus efficace que les deux précédents).

Sans entrer dans les détails, la forme normale polynomiale permet de remplir les fonctionnalités du cahier des charges. Toutefois, et c'est là le point le plus important, que faire d'une formule sous forme normale polynomiale ? Les résultats sont en général illisibles et on voit mal comment ils pourraient être exploités tels quels dans le cadre d'une application concrète. Pour être équitable, il faut signaler que la forme normale de Reed-Muller a été utilisée avec succès dans le cadre de la synthèse de circuits (en particulier en utilisant les techniques similaires à celles mises en œuvre par les diagrammes binaires de décision [26, 296, 188]). Enfin comme nous le verrons, un solveur ouvert du type de celui que j'ai préconisé permet d'implémenter facilement les primitives nécessaires à la manipulation des diagrammes binaires de décision codant des formules sous forme normale polynomiale.

Le résolveur de CHIP : Une première version a été développé par W. Büttner et H. Simonis [82]. Elle consistait en un algorithme d'unification booléenne dont le principe

était similaire à celui proposé par J. Martin et T. Nipkow dans [342]. L'idée générale consiste à substituer les variables $x_1, \ldots, x_m$ apparaissant dans le système de contraintes par des fonctions booléennes de nouvelles variables $y_1, \ldots, y_n$ de telle façon que toute affectation des y_i corresponde à une solution du système et réciproquement (cf [343, 64] pour plus de détails). Par la suite, une nouvelle version a été développée, par les mêmes auteurs qui utilise les diagrammes binaires de décision de R. Bryant (dans la version de 1986 [77]) comme représentation interne des contraintes. L'unification booléenne n'est donc plus qu'une façon de présenter les résultats. Il n'existe pas, à ma connaissance, d'article décrivant cette version *stricto sensu* mais elle est présentée dans des articles de Büttner sur l'unification dans les algèbres finies [80, 81, 89]. Une implémentation similaire a été réalisée par O. Ridoux [426] au dessus de MALI [27]. Notons que l'unification booléenne est un cas particulier de la résolution d'équations booléennes quantifiées [75] et que O. Coudert et J-C. Madre ont proposé un algorithme pour résoudre ce type d'équations avec les diagrammes binaires de décision [142].

On peut faire un certain nombre de remarques sur le résolveur de CHIP :

- Il a été utilisé par l'équipe de CHIP pour faire de la preuve de circuits [458, 462, 475, 457, 205]. Cependant, dans les articles cités, il est assez clair que c'est les diagrammes binaires de décision qui ont été utiles et non l'unification booléenne. De plus, les auteurs ont du intervenir à bas niveau sur le résolveur, en particulier pour fixer l'ordre des variables [205]. Évidemment, cela n'aurait pas été possible pour un utilisateur n'ayant pas accès au code de CHIP.
- L'unification booléenne ne sert donc pas à grand chose et en particulier pas à présenter de façon lisible les résultats. De ce point de vue, elle n'est donc pas plus utile que la forme de Reed-Muller. Cependant, le coût du calcul d'un plus grand unificateur à partir de diagramme binaire de décision ne représente qu'une faible fraction du coût total du traitement du système de contraintes [426].
- L'implémentation des diagrammes binaires de décision dans CHIP n'est pas, loin s'en faut, optimum, en particulier parce qu'elle reprend celle de 1986 [77] et non celle de 1990 [70]. La différence principale étant que, dans la première, l'opération de réduction par partage des sous-arbres isomorphes n'est pas automatiquement réalisée et a un coût non négligeable.
- Dans leurs articles présentant la résolution de contraintes booléennes et comparant CHIP à d'autres systèmes [460, 459, 461], M. Dincbas et H. Simonis n'utilisent quasiment jamais le résolveur de contraintes booléennes, mais plutôt le résolveur de contraintes sur domaines finis ou les démons[3]. Et ce, bien entendu, pour des questions d'efficacité. I. Mitterreiter et F-J. Radermacher ont d'ailleurs confirmé cela en comparant différents résolveurs (y compris ceux de CHIP) [362].

Autrement dit, si le choix des diagrammes binaires de décision comme représentation interne des contraintes était pertinent, ni l'implémentation, ni l'interface du résolveur de contraintes booléennes de CHIP ne pouvaient donner satisfaction.

[3]Les démons sont un moyen de déclencher l'exécution de routines à l'intérieur de la procédure de résolution.

Le résolveur de Prolog III : Il a été développé par F. Benhamou et J-M. Boï [31] (voir aussi [30]). Les contraintes sont codées par des ensembles de clauses et l'algorithme de résolution utilisé est une version *ad hoc* de l'algorithme de production de P. Siegel [455]. Les résultats sont aussi présentés sous forme d'ensembles de clauses. Les temps de calculs sont là encore catastrophiques (voir par exemple [362, 53, 461] pour des comparaisons avec ceux d'autres algorithmes, et [373] pour une comparaison plus «théorique»). Ce n'est pas vraiment surprenant dans la mesure où les algorithmes utilisant le principe de résolution [427] sont sans soute les plus mauvaises des méthodes proposées pour résoudre SAT (cf section 1.1).

Les résolveurs utilisant le paradigme énumération/propagation Comme on le voit, les résolveurs de contraintes booléennes des trois principaux systèmes de programmation logique avec contraintes (compte non tenu de CLP($\mathcal{R}$) qui n'en possède pas) ont d'une part de mauvaises performances et d'autre part ne servent pas à grand chose, les résultats qu'ils fournissent n'étant guère exploitables. À la décharge de leurs concepteurs, il faut dire que l'utilité d'un langage et la pertinence des choix technologiques se vérifient presque toujours *a posteriori*. La résolution de contraintes booléennes étant de bien moindre importance que celles de systèmes d'inéquations linéaires ou de contraintes sur des domaines finis, ils ont consenti beaucoup moins d'efforts pour mettre en œuvre les premières.

Les problèmes d'efficacité peuvent être significativement réduits en réalisant des résolveurs utilisant le paradigme énumération/propagation (c'est à dire implémentant des algorithmes du type procédure de Davis et Putnam [173, 172]). Plusieurs résolveurs de ce type ont été expérimentés.

Afin de valider les résultats que L. Oxusoff et moi avions obtenus avec notre algorithme de production de clauses dérivé de la procédure de Davis et Putnam [287, 372], J.-M. Boï et moi avons intégré cet algorithme dans un prototype de Prolog III. Les résultats obtenus ont bien confirmé la plus grande efficacité des méthodes énumératives [54, 55] par rapport aux méthodes utilisant le principe de résolution [455]. Ce travail d'intégration, ainsi que la recherche de jeux de tests significatifs nous ont conduits à élaborer des techniques de programmation sous contraintes booléennes [56]. Afin de comparer notre résolveur à celui de CHIP, j'ai développé un algorithme d'unification booléenne fondé sur la procédure de Davis et Putnam (publié avec quelque retard dans [409]). Je l'ai étendu à des systèmes de contraintes plus complexes dans [406]. Autant que l'on puisse en juger, il s'est montré plus efficace que celui de CHIP, sur les problèmes du banc d'essai déjà cité [372].

H. Beringer [38] a aussi implémenté un résolveur utilisant une procédure énumérative et une méthode de propagation locale (dérivée de [364]). Ses résultats confirment ceux que J.-M. Boï et moi avions obtenus. Dans [33, 344], J.-L. Massat et F. Benhamou ont proposé un résolveur très similaire dont les performances sont elles aussi du même ordre de grandeur. *Last but not least*, P. Codognet et D. Diaz ont montré que l'on peut programmer un résolveur booléen raisonnablement efficace «au dessus» d'un résolveur pour domaines finis [113, 114]. F. Benhamou et W. Older, dans [34], ont montré comment on peut utiliser l'arithmétique sur les intervalles pour en faire autant.

J. Chabrier et J.-J. Chabrier ont construit un langage programmation logique avec contraintes dont les résolveurs sont des méthodes de réparation locale : SCORE(FD/B) [93] (cf section 1.1.4 pour une présentation de ces méthodes). Compte tenu de l'efficacité de ces méthodes, si l'on s'en tient aux fonctionnalités définies par V. Saraswat, cette idée est sans doute une des plus intéressantes proposées ces dernières années et il est vraiment dommage que ces travaux n'est pas eu la publicité qu'ils méritent.

Pour terminer ce tour d'horizon, signalons les travaux de A. Bockmayr [50, 51, 52] sur ce qu'il appelle les contraintes pseudo-booléennes et qui sont fortement inspirées des travaux de J.N. Hooker sur les liens entre programmation linéaire en 0/1 et logique propositionnelle [267, 266, 268]. Les fonctions pseudo-booléennes sont des fonctions de $\{0,1\}^n$ vers $\mathbb{N}$. Toute fonction pseudo-booléenne peut s'écrire sous forme d'un polynôme multilinéaire de ses variables. Les contraintes pseudo-booléennes sont des inéquations entre de tels polynômes. Elles généralisent donc un peu les contraintes booléennes mais sont moins générales que les contraintes sur les domaines finis. Enfin, pour être complets, citons l'article – sans aucun intérêt – de G. Sidebottom [454] proposant un résolveur utilisant les matrices d'interconnection de W. Bibel [43] comme algorithme de base.

En résumé, les divers résolveurs présentés ci-dessus remplissent avec plus ou moins de bonheur le cahier des charges CLP($\mathcal{X}$) décrit section 3.1.2. Mais, restreints à ces fonctionnalités, ils font à mon sens double emploi avec les résolveurs de contraintes sur les domaines finis ou sur les intervalles. Double emploi que ne justifie pas le faible gain d'efficacité obtenu en spécialisant les résolveurs (par exemple, dans [114] P. Codognet et D. Diaz ont estimé que l'on gagne un facteur trois ou quatre par rapport à un résolveur sur les domaines finis).

3.1.4 Vers des résolveurs utiles

En 1992, j'ai utilisé les diagrammes binaires de décision dans le cadre de l'analyse de sûreté de fonctionnement de systèmes industriels (voir la prochaine section). Cette étude m'a fait changer de point de vue sur ce que doit ou devrait être un résolveur de contraintes booléennes pour Prolog. L'analyse d'arbres de défaillance (c'est de cela qu'il s'agissait) requiert deux opérations, typiques de toutes les utilisations pratiques des fonctions booléennes (voir par exemple [441]), que ne permettaient pas les résolveurs implémentés :

- Le calcul et le stockage des implicants premiers d'une fonction.
- Le calcul de la probabilité d'une fonction étant données les probabilités de chacune de ses variables.

Ces deux opérations demandent de coder et manipuler l'ensemble des solutions d'une contrainte et non d'obtenir une solution individuelle (fût-elle la meilleure par rapport à un certain critère). Autrement dit, elles ne peuvent pas être effectuées au moyen des seuls connecteurs classiques $\vee$, $\wedge$, $\neg$, ...En revanche, elles peuvent être accomplies efficacement à l'aide des diagrammes binaires de décision à condition, et cela est capital, d'avoir accès aux structures de données. C'est le cas, de nombreuses applications pratiques des fonctions booléennes au premier rang desquelles se trouve la preuve de circuits combinatoires et séquentiels (plusieurs autres sont analysées dans [441, 137]). Il y a dès lors deux approches :

- L'approche CLP($\mathcal{X}$), dite aussi «boite noire», qui consiste à fournir à l'utilisateur des résolveurs offrant le maximum de fonctionnalités, en espérant couvrir ses besoins.
- L'approche «boite de verre», préconisé par P. van Hentenryck dans [492], qui consiste à fournir à l'utilisateur un jeu de primitives élémentaires qui lui permettent de programmer, dans le langage hôte, ces propres algorithmes de résolution.

Dans le cas d'un résolveur de contraintes booléennes la première approche n'est pas réaliste. C'est la raison de l'«échec» de CHIP. En effet, cela demanderait non seulement d'intégrer «en dur» le calcul des implicants premiers et de la probabilité d'une fonction, mais encore des opérateurs comme `constrain` et `restrict`, introduits par J.-C. Madre et O. Coudert et qui sont si utiles en preuve de circuits [144, 155, 85], ou des opérateurs d'élargissement comme celui proposé par L. Mauborgne dans [345]. Et d'autres encore à venir. Enfin, il faudrait de toute façon permettre à l'utilisateur de définir l'ordonnancement des variables et de déclancher dynamiquement leur reordonnancement. En effet, les performances des diagrammes binaires de décision en dépendent fortement. Il est impossible de déterminer le "bon" ordonnancement (le meilleur algorithme est en $\mathcal{O}(3^n)$, ou n est le nombre de variables [210, 279]) et les heuristiques mises en œuvre dépendent du domaine d'application (voir par exemple [213, 337, 39, 356, 88, 215, 214] pour les circuits combinatoires, [105, 477] pour les circuits séquentiels, [65, 419, 66, 68] pour les arbres de défaillance et [202, 63, 212] pour les machines d'états finis). Le réordonnancement dynamique des variables est maintenant un "passage obligé" de tous les paquetages de diagrammes binaires de décision (voir par exemple [279, 281, 251, 431, 204] pour des discussions sur cette technique et [198, 201] pour ses applications).

La seconde approche a été mise en œuvre avec succès pour des résolveurs sur les domaines finis (voir par exemple [492, 182, 493, 116]). Dans le cas d'un résolveur utilisant les diagrammes binaires de décision comme codage interne des contraintes, elle est un peu plus délicate à implémenter. En effet, il faut permettre à l'utilisateur d'avoir accès aux tables de codage tout en gardant l'aspect déclaratif de CLP($\mathcal{X}$). Dans cette optique, j'ai réalisé plusieurs résolveurs utilisant différentes implémentations des diagrammes binaires de décision [138, 137, 414]. L'article [414] est donné en annexe D.1. Il étend une idée de K. Karplus [294] consistant à permettre de construire les diagrammes binaires de décision de bas en haut (comme usuellement) ou de haut en bas. L'intérêt de cette deuxième construction est qu'elle permet de calculer partiellement le codage d'une fonction tout en garantissant sa satisfiabilité. On résout ainsi, au moins partiellement, un des problèmes des diagrammes binaires de décision, à savoir leur manque d'efficacité pour s'assurer de la satisfiabilité d'une formule (voir par exemple [482] pour une étude sur ce sujet). L'idée directrice de ces travaux peut être résumée par le slogan suivant :

«Un résolveur de contraintes booléennes doit être un gestionnaire de mémoire,
rien qu'un gestionnaire de mémoire»

Autrement dit, on attache quasiment aucune sémantique au codage. Les diagrammes binaires de décision sont vus comme des termes clos, dont la gestion est différente des termes Prolog usuels. En particulier :

1. Il n'existe qu'une seule copie d'un tel terme $f(t_1, \ldots, t_n)$. Son adresse est calculée par

hachage en fonction de f et des termes $t_1,\ldots,t_n$.

2. L'utilisateur dispose d'un mécanisme permettant de déclarer que deux termes t_1 et t_2 sont équivalents et que l'un des deux est le représentant de la classe d'équivalence à laquelle ils appartiennent tous les deux. Ce mécanisme est en fait le même (parcours des chaînes de déréférenciation compris) que celui mis en œuvre dans l'algorithme de A. Colmerauer pour résoudre les systèmes d'équations sur les arbres rationnels [119].

Le résolveur a alors en charge la gestion de la table de hachage et de la relation d'équivalence. Il fournit les primitives de bas niveau permettant de réaliser les opérations décrites ci-dessus. Tout le reste, algorithmes de calculs compris, est programmé dans le langage hôte, quitte à fournir un certain nombre de bibliothèques avec le système de programmation logique. Dans les résolveurs proposés dans [414], on peut ainsi décider, sans aucun problème, d'utiliser la forme normale de Reed-Muller plutôt que celle de Shannon pour coder les fonctions [26, 296, 188]. L'implémentation d'un paquetage de ≪Zero-Suppressed BDD's≫ à la Minato [354, 355] est un jeu d'enfant. Idem pour celle des ≪Differential BDD's≫ à la Manna [7]. On pourrait également implémenter les diagrammes de décisions ternaires utilisés dans [340] pour analyser des programmes SIGNAL [194]. Ou encore, les diagrammes n-aires mis en œuvre dans Toupie. Ou encore les extensions des diagrammes binaires de décision proposées par exemple dans [20, 109, 308, 307, 79] pour traiter des fonctions pseudo-booléennes (i.e. des circuits arithmétiques). Notons que l'équipe de CHIP a implicitement proposé quelque chose de similaire dans [205], mais sans aller au bout de la démarche ni, à ma connaissance, l'expérimenter. La sémantique opérationnelle complète d'un langage CLP(HT) ou cc(HT) (HT pour Hash Table) reste à définir, mais il s'agit là d'un travail un peu ≪académique≫. En revanche, il semble important, pour des raisons d'efficacité, d'étudier l'intégration des termes HT dans la WAM (Warren Abstract Machine) [504] qui sert de base à la plupart des compilateurs Prolog [57, 4].

Pour conclure cette section, je voudrais dire que je suis convaincu qu'un langage complet CLP($\mathcal{X}$) ou cc($\mathcal{X}$), intégrant un résolveur de contraintes (booléennes) du type de ceux décrits plus haut, pourrait être très utile, au moins pour la conception et le prototypage des applications pratiques utilisant les fonctions booléennes. Cela ouvrirait Prolog à dans de nombreux domaines tels que la sûreté de fonctionnement, l'interprétation abstraite, la validation et la vérification de systèmes de processus communicants, ...

3.2 Sûreté de fonctionnement

La dernière partie de ce mémoire est consacrée à l'utilisation des techniques de manipulations de fonctions booléennes pour l'analyse de sûreté de fonctionnement de systèmes industriels. En préambule à cette section, j'aimerais souligner que la sûreté de fonctionnement est un domaine de recherche actif – en raison de la nécessité d'évaluer le risque dans les systèmes industriels complexes comme les centrales nucléaires, les usines chimiques ou les avions – et que ces recherches portent notamment sur le développement d'algorithmes efficaces [375, 500, 360, 310]. Or, sans doute pour des raisons historiques, les informaticiens ≪pur sucre≫ sont, au moins en France, très peu impliqués dans ces travaux. Et c'est bien

dommage car il y a là une mine de problèmes passionnants et d'applications de techniques mises au point dans d'autres cadres.

3.2.1 CECILIA/Adia

J'ai commencé à travailler sur la sûreté de fonctionnement un peu par hasard, grâce au stage de fin d'année d'un étudiant de DESS de génie logiciel, chez Dassault Aviation. Cet étudiant travaillait sur le logiciel CECILIA, développé par Dassault, qui gère des arbres de défaillance. La méthode des arbres de défaillance est une technique d'ingénierie largement utilisée pour l'analyse des systèmes embarqués [499, 316, 320]. Elle consiste à décomposer le système étudié en sous-systèmes, puis chaque sous-systèmes en sous-systèmes et ainsi de suite jusqu'au degré de granularité voulu. On exprime ensuite les défaillances possibles d'un composant comme une fonction booléenne des défaillances de ses sous-composants. À partir ce cette modélisation, on peut faire deux types d'études :

- Des études quantitatives : par exemple, calculer la probabilité de panne du système – appelée probabilité de l'événement sommet – en fonction du temps et des lois de probabilités de pannes des composants élémentaires.
- Des études qualitatives : par exemple, déterminer les jeux de pannes minimaux de composants élémentaires qui produisent une panne du système. Ce qui revient à calculer les implicants premiers[4] de la fonction booléenne correspondante.

Si la méthode des arbres de défaillance est maintenant bien comprise par les praticiens, sa mise en œuvre se heurte au problème du coût des calculs à effectuer. En effet, déterminer la probabilité de l'événement sommet est un problème #P-difficile[5]. Exhiber tous les implicants premiers d'une formule est, dans le pire des cas, exponentiel en espace[6]. De fait, même des arbres de taille moyenne (quelques dizaines de variables et de connecteurs) peuvent avoir plusieurs milliers d'implicants premiers. On peut approximer la probabilité de l'événement sommet à partir des implicants premiers en utilisant les premiers termes du développement de Sylvester-Poincaré (voir par exemple [320]). De nombreux outils ont été proposés pour résoudre les deux problèmes. Certains travaillent en codant «en dur» les circuits logiques correspondant aux arbres de défaillance (par exemple Escaf [314]), d'autres, les plus nombreux, sont purement logiciels et utilisent différentes techniques : récriture, méthode de Monte-Carlo, ...(voir par exemple, et plus des ouvrages déjà cités, [324, 374, 295, 441, 442, 124]). Pratiquement tous ces outils donnent des résultats approchés, tant pour les calculs de probabilité que pour ceux des implicants premiers (en se limitant au calcul des implicants premiers ayant peu de littéraux – au plus 4 ou 5). Aucun d'eux ne permet de traiter de façon complète et en des temps de calculs raisonnables des

[4] Un *implicant* est un ensemble de littéraux dont la conjonction implique la formule considérée. Il est dit *premier* si il est minimal du point de vue de l'inclusion. Dans le cas des fonctions monotones, les fiabilistes appellent aussi les implicants premiers des *coupes minimales*.

[5] La complexité des problèmes de fiabilité, exprimés en termes de formules booléennes ou de graphes, a été étudiée par divers auteurs [428, 94, 485, 484, 23, 25, 393, 394, 24, 395, 396, 392]. J'ai précisé les résultats connus dans [423], aussi ne les reprendrais-je pas ici.

[6] En effet, il peut y a avoir un nombre exponentiel d'implicants premiers [404, 94].

arbres un peu importants.

Il se trouve que les diagrammes binaires de décision permettent de gagner plusieurs ordres de magnitude en efficacité et en précision des calculs sur les algorithmes classiques pour les études quantitatives comme pour les études qualitatives d'arbres de défaillance. D'abord, étant donné le diagramme binaire de décision associé à un arbre de défaillance, il est trivial de calculer la probabilité de l'événement sommet de ce dernier (en utilisant le théorème de Shannon [440]). Pour le calcul des implicants premiers, deux algorithmes ont été proposés indépendamment : le premier par J.-C. Madre et O. Coudert [146, 147][7], le second par l'auteur de ces lignes [410]. Mon algorithme répondait aux besoins exprimés par Dassault : il ne traite que les fonctions monotones, mais en revanche il est plus rapide et surtout moins coûteux en place mémoire[8]. La réalisation d'un gestionnaire de diagrammes binaires de décision, appelé Adia, contenant cet algorithme et son incorporation dans CECILIA a fait l'objet d'un contrat avec Dassault. L'article [410] est donné annexe D.2.

Pour l'anecdote, E. Ledinot, de Dassault Aviation, a récemment prouvé en Coq [274], l'algorithme que j'avais proposé dans [410] (voir [315], cette contribution se trouve dans la distribution Coq[9]).

3.2.2 Aralia

Depuis le début de l'année 94, Y. Dutuit du LADS[10] et moi collaborons dans le cadre d'un contrat, avec un groupe de partenaires industriels issu du groupe ≪recherches méthodologiques≫ de l'Institut de Sûreté de Fonctionnement[11]. Dans le cadre de ce contrat, j'ai réalisé le logiciel Aralia qui intègre, outre l'algorithme de J.-C. Madre et O. Coudert et le mien, un certain nombre d'autres fonctionnalités jugées utiles par les fiabilistes, comme le dénombrement, le tri, la sélection des implicants premiers par nombres d'éléments, probabilités ...Aralia est aussi un outil pédagogique dans la mesure où l'état courant des tables de hachage, les nœuds d'un diagramme binaire de décision et bien d'autres choses sont affichables.

Aralia m'a donc demandé un effort de programmation non négligeable. À cet effort c'est ajouté celui de la rédaction de divers rapports intermédiaires [411, 412], d'un manuel

[7] En fait J.-C. Madre et O. Coudert ont publié toute une série d'articles sur cette question : [148, 145, 146, 147, 149, 150, 151, 152, 154] pour ne citer que ceux que je connais (voir [333] pour une bibliographie détaillée).

[8] Plus précisément, il permet de déterminer les parties positives d'implicants premiers qui ne sont pas incluses dans la partie positive d'un autre implicant premier (voir [423] pour une comparaison plus détaillée). Cette information positive est souvent aussi intéressante que les implicants premiers complets. En effet, d'une part les variables représentent des pannes et on est donc intéressé par ce qui est en panne plutôt que par ce qui n'est pas en panne ; d'autre part on travaille souvent sur des composants dont les probabilités de panne sont faibles (disons entre 10^{-3} et 10^{-5}) pour lesquels on peut par conséquent sans grand risque approximer la probabilité de non-occurrence d'une panne par 1.

[9] Disponible par ftp à ftp.inria.fr, INRIA/coq/V5.10.

[10] Laboratoire d'Analyse Des Systèmes de l'Université Bordeaux I.

[11] Ce groupe est constitué, outre le LADS et le LaBRI, du CEA, la Cogema (SGN), EdF, Elf Aquitaine, Renault, Schneider Electric et Technicatome.

d'utilisation [425], d'un manuel de développement [418] et d'un «Primer» [413] présentant de façon aussi pédagogique que possible les diagrammes binaires de décision et les différents algorithmes mis en œuvre. C'est le prix à payer pour que les idées passent des laboratoires vers l'industrie. Il est lourd. Aralia est aujourd'hui utilisé par les fiabilistes pour des études réelles, notamment chez Elf Aquitaine et au CEA. Les résultats obtenus ont donné lieu à plusieurs publications [9, 10].

Le contrat a été reconduit pour l'année 1995/1996. Aralia va être diffusé commercialement, à partir du printemps 1996, par une société d'ingénierie. Il n'a, pour le moment pas d'autres concurrents que MetaPrime, le gestionnaire d'arbres de défaillance réalisé – au moins pour ce qui concerne le noyau – par J.-C. Madre et O. Coudert [150, 151, 154]. Les performances des deux outils semblent comparables, avec un léger avantage pour Aralia, pour autant que l'on puisse en juger [66].

3.2.3 La boite à outils de Aralia

Les expériences menées avec Aralia[12] montrent que pratiquement tous les arbres de défaillance construits «à la main» par les fiabilistes sont traités en des temps très raisonnables (quelques minutes au plus) [9, 10]. Cependant, ce résultat est provisoire. Pour deux raisons. La première, c'est que l'appétit vient en mangeant. Les ingénieurs, sachant qu'ils disposent d'un outil beaucoup plus puissant que ceux de la génération précédente, incorporent de plus en plus d'informations à leurs modèles. Les arbres construits aujourd'hui par le CEA, Elf Aquitaine et Dassault (sans parler de EdF) comprennent plusieurs centaines de variables et de connecteurs. La deuxième raison est liée au développement d'ateliers de sûreté de fonctionnement, comme Figaro [67] ou Fiabex [276, 309], qui comprennent des générateurs automatiques d'arbres de défaillance travaillant à partir de descriptions de plus haut niveau des systèmes. Bien entendu, les arbres construits de cette façon sont beaucoup plus gros et beaucoup moins simplifiés que les arbres construits «à la main». Ces nouveaux arbres ne sont plus traitables tels quels par Aralia.

La vraie recherche commence donc ...

Il y a principalement deux voies d'investigations :

- La récriture des arbres avant leur traitement par les diagrammes binaires de décision. Cette voie a été largement explorée par les fiabilistes, mais dans la perspective d'un traitement par les algorithmes «classiques» de calcul des implicants premiers (voir entre autres les nombreux travaux sur la modularisation des arbres et/ou leur simplification [48, 103, 429, 325, 90, 508, 250, 511, 300, 197, 65, 124]).
- L'amélioration du traitement par les diagrammes binaires de décision. Cette amélioration passe par la mise au point de bonnes heuristiques d'ordonnancement (voir plus loin, mais aussi [213, 337, 39, 356, 477, 88, 215, 66]), le réordonnancement dynamique des variables [279, 281] et la mise au point de nouveaux algorithmes approximant les calculs (des premiers essais ont été faits dans ce sens [345], mais pratiquement tout

[12]Pour mener à bien ces expériences, j'ai constitué un banc d'essai de plusieurs dizaines d'arbres venant tous de descriptions de systèmes industriels.

reste à faire).

Tout cela doit être revu, corrigé et surtout expérimenté. Car il faut insister encore sur ce point : aucun outil théorique ne permet de prévoir la complexité pratique de tel ou tel calcul. Pire, les intuitions sont presque à tous les coups de fausses bonnes idées. Il n'y a pas de règle générale, ce qui marche ici ne marche pas forcément là. Autrement dit, il faut essayer, encore essayer, toujours essayer. Fort de cette constatation et de l'expérience de Pardi, j'ai commencé à réaliser une boite à outils associée à Aralia [419]. Elle se présente sous forme d'un ensemble de commandes Unix (partageant du code). Ces commandes implémentent la plupart des heuristiques d'ordonnancement des variables proposées dans la littérature, des traducteurs vers d'autres outils, des algorithmes de récriture et simplification de formules ainsi qu'un certain nombre d'autres petits outils. À cette occasion, j'ai proposé, dans [195], un algorithme linéaire pour isoler les modules[13] d'un arbre de défaillance dérivé de l'algorithme de Tarjan [472]. Certains arbres qui n'étaient pas traitables tels quels par Aralia ont pu être traités après transformation. Il reste un gros travail d'expérimentation et d'analyse à faire pour avoir une idée plus précise des règles à appliquer suivant la forme des arbres étudiés. La conception, du point de vue génie logiciel, de la boite à outils permet justement ce travail par essais/erreurs.

À coté de la boite à outils proprement dite, je développe le logiciel Réseda qui est un compilateur vers Aralia de réseaux de fiabilité. Les réseaux de fiabilité sont une autre méthode de sûreté de fonctionnement (voir [453] pour un présentation complète de cette technique). Elle consiste à décrire les systèmes considérés sous forme d'un graphe dont les sommets et les arêtes sont étiquetées par des formules booléennes décrivant les pannes possibles de ces sommets ou arêtes. On associe une loi de probabilité à chaque variable apparaissant dans ces formules. Étant donnés deux sommets, on se pose des questions très similaires à celles posées sur les arbres de défaillance : quelle est la probabilité pour que les deux sommets soient déconnectés, quels sont les jeux d'arêtes et de sommets minimaux provoquant la déconnection, ...On peut compiler les réseaux de défaillance en formules booléennes de plusieurs façons, toutes n'étant évidemment pas équivalentes du point de vue de l'efficacité de leur traitement par les diagrammes binaires de décision. Dans [196], j'ai fait une première série de comparaisons de deux compilations, l'une proposée dans [69], l'autre dans [453]. Ces résultats partiels montrent que l'on a tout intérêt à travailler sur les formules avant de les traiter.

Comme le lecteur peut s'en convaincre au vu de ce qui précède, la sûreté de fonctionnement offre un champs de recherches passionnantes sur l'utilisation des techniques de manipulations de fonctions booléennes (mais pas seulement). C'est sans aucun doute un domaine dans lequel je mettrais beaucoup d'énergie dans un avenir proche.

[13] Un module est un sous-arbre ne partageant pas de variables avec le reste de l'arbre.

Bibliographie

[1] S. Abramsky and C. Hankin. *Abstract Interpretation of Declarative Languages*. Ellis Horwood Limited, West Sussex, 1987.

[2] A. Aiba, K. Sakai, Y. Sato, D. Hawley, and R. Hasegawa. Constraint Logic Programming Language CAL. In *Proceedings of the International Conference on the Fifth Generation of Computers 1988*, pages 263–276, 1988.

[3] A. Aiken and T.K. Lakshman. Directional Type Checking of Logic Programs. In B. Le Charlier, editor, *Proceedings of the* 1^{rst} *International Symposium on Static Analysis, SAS'94*, volume 864 of *LNCS*, pages 43–60. Springer Verlag, 1994.

[4] H. Aït-Kaci. *Warren's Abstract Machine*. MIT Press, 1991. ISBN 0-262-01123-9.

[5] B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, 27(6) :509–516, 1978.

[6] J.F. Allen. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM*, 26 :832–843, 11 1983.

[7] A. Anuchitanukul and Z. Manna. Differential BDDs. Tech. Report TR-94-1525, Stanford University, 1994.

[8] K.R. Apt and M.H. van Emden. Contributions to the Theory of Logic Programming. *JACM*, 29(3) :841–862, july 1982.

[9] Groupe Aralia. Arbres de Défaillances et diagrammes binaires de décision. In *Actes du* 1^{er} *congrès interdisciplinaire sur la Qualité et la Sûreté de Fonctionnement*, pages 47–56. Université Technologique de Compiègne, 1994. The "Groupe Aralia" is constitued by A. Rauzy (LaBRI – Université Bordeaux I), Y. Dutuit (LADS – Université Bordeaux I), J.P. Signoret (Elf-Aquitaine – Pau), M. Chevalier (Schneider Electric – Grenoble), I. Morlaes (SGN – Saint Quentin en Yvelines), A.M. Lapassat (CEA-LETI – Saclay), S. Combacon (CEA-IPSN – Valduc), F. Brugère (Technicatome – Aix-Les Milles), M. Bouissou (EDF-DER – Clamart).

[10] Groupe Aralia. Computation of Prime Implicants of a Fault Tree within Aralia. In *Proceedings of the European Safety and Reliability Association Conference, ESREL'95*, pages 190–202, Bournemouth – England, June 1995. European Safety and Reliability Association.

[11] A. Arnold. MEC : a System for Constructing and Analysing Transition Systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*. Springer Verlag, June 1989.

[12] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants*. ERI. Masson, 1992.

[13] A. Arnold and M. Nivat. Comportements de processus. In *Colloque AFCET "Les Mathématiques de l'informatique"*, 1982.

[14] V. Arvind and S. Biswas. An $\mathcal{O}(n^2)$ algorithm for the satisfiability problem of a subset of propositional sentences in CNF that includes horn sentences. *Information Processing Letters*, 24 :67–69, 1987.

[15] B. Aspvall. Recognizing Disguised NR(1) Instances of the Satisfiability Problem. *Journal of Algorithms*, 1 :97–103, 1980.

[16] B. Aspvall, M. Plass, and R. Tarjan. A Linear Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulae. *Information Processing Letters*, 8(3) :121–123, 1979.

[17] J.M. Atlee and J. Gannon. State-based model checking of event-driven system requirements. *IEEE Transactions on Software Engineering*, 19(1) :24–40, January 1993.

[18] E. Audureau, P. Enjalbert, and L. Fariñas del Cerro. *Logique temporelle*. études et recherches en informatique. Masson, 1989. ISBN 2-225-81967-X.

[19] G. Ausiello and G. Italiano. On-Line Algorithms for Polynomially Solvable Satisfiability Problems. *Journal of Logic Programming*, 10 :69–90, 1990.

[20] R.I. Bahar, E.A. Frohm, C.M. Goana, G.D. Hatchel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the IEEE International Conference on Computer Aided Design, ICCAD'93*, pages 188–191, November 1993.

[21] Groupe Bahia. Étude comparatives de trois formalismes en calcul propositionnel. In *Proceedings of the* $4^{ième}$ *Journées du PRC Intelligence Artificielle*, pages 239–314. Teknea, 1992.

[22] Groupe Bahia. Étude comparatives de trois formalismes en calcul propositionnel. In *Proceedings of the* $5^{ième}$ *Journées du PRC Intelligence Artificielle*, pages 126–158. Teknea, 1995.

[23] M.O. Ball. Complexity of network reliability computations. *Networks*, 10 :153–165, 1980.

[24] M.O. Ball. Computational complexity of network reliability analysis : an overview. *IEEE Transactions on Reliability*, R-35 :230–239, 1986.

[25] M.O. Ball and J.S. Provan. Bounds on reliability polynomial for shellable indepence systems. *SIAM Journal on Algebraic and Discrete Methods*, 3 :166–181, 1982.

[26] B. Becker, R. Drechsler, and M. Theobald. On the implementation of a package for efficient representation and manipulation of functional decision diagrams. In *IFIP WG 10.5 Workshop on Applications of Reed-Muller Expansion in Circuit Design, Hamburg*, 1993.

[27] Y. Bekkers, B. Canet, O. Ridoux, and L. Ungaro. MALI : a memory with a real-time garbage collector for implementing logic programming languages. In *Proceedings of the* 3^{rd} *Symposium on Logic Programming*. IEEE, 1986.

[28] B. Benhamou, R. Génisson, and A. Rauzy. Une version concurrente de la procédure de Davis et Putnam. *Revue d'Intelligence Artificielle*, 10(4) :499–506, 1996. Note de recherche.

[29] B. Benhamou and L. Sais. Theoretical Study of Symetries in Propositional Calculus and Applications. In *Proceedings of the 11th Conference on Automatic Deduction (CADE-11)*, volume 607. LNAI, Springer Verlag, 1992.

[30] F. Benhamou. Boolean Algorithms in Prolog III. In A. Colmerauer and F. Benhamou, editors, *Constraint Logic Programming : Selected Research*, pages 307–325. MIT Press, 1993. ISBN 0-262-02353-9.

[31] F. Benhamou and J.M. Boï. *Le traitement des contraintes booléennes dans PrologIII.* PhD thesis, GIA - Université Aix-Marseille II, 1988. Thèse de doctorat.

[32] F. Benhamou and A. Colmerauer, editors. *Constraint Logic Programming : Selected Research.* MIT Press, Cambridge, Mass., 1993. ISBN 0-262-02353-9.

[33] F. Benhamou and J.L. Massat. Boolean Pseudo-Equations in Constraint Logic Programming. In *Proceedings of the* 10^{th} *International Conference on Logic Programming*, pages 517–531. MIT Press, 1993. ISBN 0-262-73105-3.

[34] F. Benhamou and W. Older. Applying Interval Arithmetic to Real, Integer, and Boolean Constraints. *to appear in Journal of Logic Programming*, 1994.

[35] H. Bennaceur and G. Plateau. FAST : une méthode de résolution du problème linéaire de satisfaction de contraintes. *Technique et Science Informatique*, 11(3) :33–57, 1992.

[36] H. Bennaceur and G. Plateau. An exact algorithm for the constraint satisfaction problem : Application to logical inference. *Information Processing Letters*, 48 :151–158, 1995.

[37] A. Beringer, G. Aschemann, H.H. Hoos, M. Metzger, and A. Weiß. GSAT versus Simulated Annealing. In A. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 130–134. Wiley & Sons, Ltd, 1994.

[38] H. Beringer. CLP(B) : Logic Programming with Boolean Constraints. Technical report, IBM European Center of Applied Mathematics, December 1991.

[39] C.L. Berman. Ordered Binary Decision Diagrams and Circuit Structure. In *Proceedings of the IEEE International Conference on Computer Aided Design, ICCAD'89*, September 1989. Cambridge MA, USA.

[40] J. Bern, C. Meinel, and A. Slobodova. Efficient OBDD-Based Boolean Manipulation in CAD Beyond Current Limits. In *Proceedings of the 32th ACM/IEEE Design Automation Conference, DAC'95*, pages 408–413, 1995.

[41] C. Berthet, O. Coudert, and J.-C. Madre. New Ideas on Symbolic Manipulations of Finite State Machines. In *Proceedings of the IEEE International Conference on Computer Design, ICCD'90*, Cambridge MA, USA, September 1990.

[42] C. Bessière. Arc Consistency and Arc Consistency again. *Artificial Intelligence*, 65 :179–190, 1994.

[43] W. Bibel. On Matrix with Connections. *JACM*, 28(4), october 1981.

[44] W. Bibel. Short Proofs of the Pigeonhole Formulas Based on the Connection Method. *Journal of Automated Reasoning*, 6, 1990.

[45] A. Billionnet and A. Sutter. An efficient algorithm for the 3-satisfiability problem. *Operation Research Letters*, 12 :29–36, 1992.

[46] J.P. Billon. Perfect Normal Forms for Discrete Functions. Technical Report DSG/CRG/87014, Centre de Recherche, BULL, 1987.

[47] R.S. Bird. Tabulation techniques for recursive programs. *Computing Surveys*, 12(4), 1980.

[48] Z.W. Birnbaum and J.P. Esary. Modules of coherent binary systems. *SIAM J. of Applied Mathematics*, 13 :442–462, 1965.

[49] C.E. Blair, R.G. Jerolsow, and J.K. Lowe. Some results and experiments in programming techniques for propositional logic. *Comput. Oper. Res.*, 13(5) :633–645, 1986.

[50] A. Bockmayr. Logic Programming with Pseudo-Boolean Constraints. Research report MPI-I-91-227, Max Planck Institut, Saarbrucken, Germany, 1991.

[51] A. Bockmayr. Logic Programming with Pseudo-Boolean Constraints. In A. Colmerauer and F. Benhamou, editors, *Constraint Logic Programming : Selected Research*, pages 327–350. MIT Press, 1993. ISBN 0-262-02353-9.

[52] A. Bockmayr. Solving Pseudo-Boolean Constraints. In A. Poldeski, editor, *Constraint Programming : Basic and Trends*, volume 910, pages 22–38. LNCS, 1995.

[53] J.M. Boï, E. Innocente, A. Rauzy, and P. Siegel. Production Fields : a New Approach to Deduction Problems and two Algorithms for Propositional Calculus. *Revue Française d'Intelligence Artificielle*, 06(3) :235–253, 1992.

[54] J.M. Boï and A. Rauzy. La démonstration automatique en calcul propositionnel au service de la programmation par contraintes. In *Actes du Séminaire de Programmation en Logique de Trégastel, SPLT'90*, pages 511–521. CNET, 1990.

[55] J.M. Boï and A. Rauzy. Two algorithms for constraints system solving in propositional calculus and their implementation in prologIII. In P. Jorrand and V. Sugrev, editors, *Proceedings Artificial Intelligence IV Methodology, Systems, Applications (AIMSA'90)*, pages 139–148. North-Holand, September 1990. Alba-Varna bulgarie.

[56] J.M. Boï and A. Rauzy. Using Boolean Constraints in Prolog. In *Proceedings of the Italian Conference on Logic Programming GULP'91*. GULP, 1991.

[57] P. Boizumault. *Prolog, l'implantation*. études et recherches en informatique. Masson, 1988. ISBN 2-225-81479-1.

[58] G. Boole. *An Investigation of the Laws of Though*. Dover Publications, 1854. Reprint, no place or date.

[59] E. Boros, Y. Crama, and P.L. Hammer. Polynomial-time inference of all implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, 1 :21–32, 1990.

[60] E. Boros, Y. Crama, P.L. Hammer, and M. Saks. A complexity index for satisfiability problems. *SIAM Journ. Comp.*, 23 :45–49, 1994.

[61] E. Boros, P.L. Hammer, and X. Sun. Recognition of q-Horn formulae in linear time. *Discrete Applied Mathematics*, 55 :1–13, 1994.

[62] G. Bossu and P. Siegel. Saturation, non monotonic reasonning and the closed world assumption. *Artificial Intelligence*, 25(1) :13–63, 1985.

[63] A. Bouali. *Études et mises en œuvre d'outils de vérification basée sur la bisimulation*. PhD thesis, Université Paris VII, 03 1993. in french.

[64] A. Boudet and J.P. Jouannaud. Unification in Boolean Rings and Abelian Groups. *Special Issue of Journal of Symbolic Computation*, May 1988.

[65] M. Bouissou. Une heuristique d'ordonnancement des variables pour la construction des diagrammes de décision binaires à partir d'arbre de défaillance. In *Actes du congrès* $\lambda - \mu$, pages 547–556, 1994.

[66] M. Bouissou. An Ordering Heuristics for Building Binary Decision Diagrams from Fault-Trees. In *Proceedings of the Annual Reliability and Maintenability Symposium, ARMS'96*, 1996.

[67] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte. Knowledge modelling and reliability processing : presentation of the FIGARO language and of associated tools. In *Proceedings of SAFECOMP'91*, Trondheim, Norway, 1991.

[68] M. Bouissou and C. Brizec. Application of Two Generic Availability Allocation Methods on a Real Life Example. In C. Cacciabue and I.A. Papazoglou, editors, *Proceedings of European Safety and Reliability Association Conference, ESREL'96*, volume 3, pages 2099–2104. Springer Verlag, 1996. ISBN 3-540-76051-2.

[69] M. Bouissou, J.-C. Madre, O. Coudert, and H. Fraïsse. Application d'un ATMS fondé sur les diagrammes de décision binaires à l'analyse de digraphes et de connexité dans un réseau. Technical Report HT-53/93/063A, EDF-DER, November 1993.

[70] K. Brace, R. Rudell, and R. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 40–45. IEEE 0738, 1990.

[71] Y. Breitbart, H. Hunt III, and D. Rosenkrantz. On the size of binary decision diagrams representing Boolean functions. *Theoretical Computer Science*, 145 :45–69, 1994.

[72] S. Brlek and A. Rauzy. Implementation of Constrained Transition Systems : a Unified Approach. In *Proceedings of the Bordeaux–Montréal Workshop, BMW'94*, pages 125–141, Montreal, 1994. Éditions du LACIM.

[73] S. Brlek and A. Rauzy. Synchronization of Constrained Transition Systems. In H. Hong, editor, *Proceedings of the First International Symposium on Parallel Symbolic Computation (PASCO'94)*, pages 54–62, Linz, Ostreich, 1994. World Scientific Publishing.

[74] A. Broder, A. Frieze, and E. Upfal. On the Satisfiability and Maximum Satisfiability of Random 3-CNF Formulas. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993.

[75] F.M. Brown. *Boolean Reasoning*. Kluwer Academic Publishers, 1990.

[76] M. Bruynooghe. Solving Combinatorial Search Problems by Intelligent Backtracking. *Information Processing Letters*, 12(1) :36–39, 1981.

[77] R. Bryant. Graph Based Algorithms for Boolean Fonction Manipulation. *IEEE Transactions on Computers*, 35(8) :677–691, August 1986.

[78] R. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24 :293–318, September 1992.

[79] R.E. Bryant and Y.-A. Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the 32th Design Automation Conference*, pages 535–541, June 1995.

[80] W. Buettner. Unification in Finite Algebras is Unitary (?). In *Proceedings of* 9^{th} *International Conference on Automated Deduction, CADE'9*, volume 310, pages 368–377. LNCS, 1988.

[81] W. Buettner. Implementing Complex Domains of Application in an Extended Prolog System. *Journal of General System*, 15 :129–139, 1989.

[82] W. Buettner and H. Simonis. Embedding Boolean Expressions into Logic Programming. *Journal of Symbolic Computation*, 4 :191–205, 1987.

[83] K. Bugrara, Y. Pan, and P. Purdom. Exponential Average Time for the Pure Literal Rule. *SIAM Journal of Computing*, 18 :409–418, 1989.

[84] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential Circuit Verification Using Symbolic Model Checking. In *Proceedings of the* 27*th ACM/IEEE Design Automation Conference, DAC'90*, pages 46–51, june 1990.

[85] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking : 10^{20} States and Beyond. *IEEE transactions on computers*, 1990.

[86] J.E. Burns. Symmetry in systems of synchronous processes. *IEEE Transactions on Computers*, pages 169–174, October 1981.

[87] S.R. Buss and G. Turán. Resolution proofs of generalized pigeonhole principles. *Theoretical Computer Science*, 62 :311–317, 1988.

[88] K.M. Butler, D.E. Ross, R. Kapur, and M.R. Mercer. Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered BDDs. In *Proceedings of the 28th Design Automation Conference, DAC'91*, June 1991. San Francisco, California.

[89] W. Büttner, K. Estenfeld, R. Schmid, H.-A. Schneider, and E. Tidèn. Symbolic Constraint Handling through Unification in Finite Algebras. *Communication and Computing*, pages 97–118, 1990.

[90] L. Camarinopoulos and J. Yllera. An Improved Top-down Algorithm Combined with Modularization as Highly Efficient Method for Fault Tree Analysis. *Reliability Engineering and System Safety*, 11 :93–108, 1985.

[91] T. Castell. Résolution Arrière dans la Procédure de Davis et Putnam, 1995. Actes du Congrès AFCET Reconnaissance de Formes et Intelligence Artificielle, RFIA'96.

[92] C. Cayrol, M. Cayrol, and O. Palmade. P-arbres de déduction : application aux atms. *Revue d'Intelligence Artificielle*, 9(1) :35–52, 1995.

[93] J. Chabrier, V. Juliard, and J.-J. Chabrier. SCORE(FD/B) An efficient complete local-based search method for satisfiability problems. In *Proceedings of the post-conference workshop on hard problems, CP'95*, pages 25–42. Laboratoire d'Informatique de Marseille, 1995.

[94] A.K. Chandra and G. Markowsky. On the number of prime implicants. *Discrete Mathemathics*, 24 :7–11, 1978.

[95] V. Chandru, C.R. Coulard, P.L. Hammer, M. Montanez, and X. Sun. On renamable Horn and generalized Horn functions. In *Annals of Mathematics and Artificial Intelligence*, volume 1. J.C. Baltzer AG, Scientific Publishing Company, Basel Switzerland, 1990.

[96] V. Chandru and J.N. Hooker. Extended Horn sets in Propositional Logic. *Journal of the ACM*, 38 :205–221, 1990.

[97] V. Chandru and J.N. Hooker. Detecting embedded Horn structure in propositional logic. *Information Processing Letters*, 42 :109–111, 1992.

[98] K.M. Chandy and J. Misra. *Parallel Program Design.* Addison Wesley, 1988. ISBN 0-201-05866-9.

[99] M. Chao and J. Franco. Probabilistic Analysis of a Two Heuristics for the 3-Satisfiability Problem. *SIAM Journal on Computing*, 15 :1106–1118, 1986.

[100] B. Le Charlier. Abstract Interpretation and Finite Domain Symbolic Constraints. In A. Poldeski, editor, *Constraint Programming : Basic and Trends*, volume 910, pages 145–170. LNCS, 1995.

[101] B. Le Charlier and P. van Hentenryck. Groundness Analysis for Prolog : Implementation and Evaluation of the Domain `prop`. In *Proceedings of the 1993 ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (*PEPM'93*)*, 1993.

[102] B. Le Charlier and D. Zampunieris, editors. *Proceedings of the first international Static Analysis Symposium, SAS'94*, volume 864, Namur, Belgium, September 1994. LNCS.

[103] P. Chatterjee. Modularization of fault trees : A method to to reduce the cost of analysis. *Reliability and Fault Tree Analysis, SIAM*, pages 101–137, 1975.

[104] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the International Joint Conference of Artificial Intelligence, IJCAI'91*, 1991.

[105] H. Cho, G. Hatchel, S.W. Jeong, B. Plessier, E. Swharz, and F. Somenzi. ATPG Aspect of FSM Verification. In *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'90*, November 1990.

[106] V. Chvátal and B. Reed. Miks gets some (the odds are on his side). In *Proceedings of the 33rd IEEE Symp. on Foundations of Computer Science*, pages 620–627, 1992.

[107] V. Chvátal and E. Szemerédi. Many Hard Examples for Resolution. *JACM*, 35(4) :759–768, october 1988.

[108] E.M. Clarke, E.A. Emmerson, and A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Trans. on Prog. Lang Syst*, 8 :244–263, 1986.

[109] E.M. Clarke, K.L. McMillan, X. Zhao, M. Fujita, and J.C.-Y. Yang. Spectral transforms for large Boolean functions with application to technology mapping. In *Proceedings of the* 30^{th} *ACM/IEEE Design Automation Conference, DAC'93*, pages 54–60, 1993.

[110] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In G.C. Bochmann and D.K. Probst, editors, *Proceedings of the Fourth international workshop on Computer Aided Verification, CAV'92*, volume 663 of *LNCS*, pages 410–422. Springer Verlag, 1992.

[111] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench : A Semantics-Based Tool for the Verification of Concurrent Systems. *ACM Transactions on Programming Languages and Systems*, 15(1) :36–72, January 1993.

[112] M. Codish and B. Demoen. Analysing Logic Programs using `prop`-ositional Logic Programs and a Magic Wand. In *Proceedings of ILPS'93*, 1993.

[113] P. Codognet and D. Diaz. Boolean Constraint Solving Using `clp(FD)`. In D. Miller, editor, *Proceedings of the International Logic Programming Symposium, ILPS'93*. MIT Press, 1993.

[114] P. Codognet and D. Diaz. clp(B) : Combining Simplicity and Efficiency in Boolean Constraint Solving. In *Proceedings of the Conference on Programming Language Implementation and Logic Programming, PLILP'94*, volume 844. LNCS, 1994.

[115] P. Codognet and G. Filè. Computations, Abstractions and Constraints in Logic Programs. In *Proceedings of the IEEE International Conference on Computer Languages, ICCL'92*. IEEE Press, 1992.

[116] P. Codognet and G. Nardiello. Enhancing the constraint-solving power of `clp(FD)` by means of path-consistency methods. In A. Poldeski, editor, *Constraint Programming : Basic and Trends*, volume 910, pages 39–61. LNCS, 1995.

[117] J. Cohen. Constraint logic programming languages. *Communications of the ACM*, 33 :52–68, July 1990.

[118] A. Colmerauer. Prolog in 10 Figures. In *Proceedings of the* 8^{th} *International Joint Conference on Artificial Intelligence, IJCAI'83*, pages 487–499, 1983.

[119] A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *Proceedings of the* 2^{nd} *International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.

[120] A. Colmerauer. Opening the PrologIII Universe. *Byte*, August 1987.

[121] A. Colmerauer. An Introduction to PrologIII. *Communications of the ACM*, 28(4), July 1990.

[122] A. Colmerauer, H. Kanouis, P. Roussel, and R. Pasero. Un système de Communication Homme-Machine en Français. Technical report, Groupe de Recherche en Intelligence Artificielle, Université d'Aix-Marseille II, 1973.

[123] M. Conforti and G. Cornuéjols. A class of logical inference problems solvable by linear programming. In *Proceedings FOCS'92*, volume 33, pages 670–675, 1992.

[124] S. Contini. A new hybrid method for fault tree analysis. *Reliability Engineering and System Safety*, 49 :13–21, 1995.

[125] S. Cook and T. Pitassi. A feasibly constructive lower bound for resolution proofs. *Information Processing Letters*, 34 :81–85, 1990.

[126] S.A. Cook. The Complexity of Theorem Proving Procedures. In *Proceedings of the 3rd Ann. Symp. on Theory of Computing, ACM*, pages 151–158, 1971.

[127] S.A. Cook. A short proof of the pigeon hole principle using extended resolution. *ACM SIGACT News*, 8 :28–32, 1976.

[128] S.A. Cook and R.A. Reckhow. On the Length of Proofs in the Propositional Calculus. In *Proceedings of the 6th ACM Symposium on Theory of Computing, STOC'74*, pages 135–148, 1974.

[129] S.A. Cook and R.A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1) :39–50, 1979.

[130] M.C. Cooper. An optimal k-consistency Algorithm. *Artificial Intelligence*, 41 :89–95, 1989.

[131] M.C. Cooper, D.A. Cohen, and P.G. Jeavons. Characterizing Tractable Constraints. *Artificial Intelligence*, 65 :347–361, 1994.

[132] M-M. Corsini, B. Le Charlier, K. Musumbu, and A. Rauzy. Efficient Abstract Interpretation of Prolog Programs by means of Constraint Solving over Finite Domains. Research Report 93–16, LaBRI, Université Bordeaux I, 1993.

[133] M-M. Corsini, B. Le Charlier, K. Musumbu, and A. Rauzy. Efficient Abstract Interpretation of Prolog Programs by means of Constraint Solving over Finite Domains (extended abstract). In *Proceedings of the 5th Int. Symposium on Programming Language Implementation and Logic Programming, PLILP'93*, volume 714, Tallin, Estonia, 1993. LNCS.

[134] M-M. Corsini, A. Griffault, and A. Rauzy. Yet another Application for Toupie : Verification of Mutual Exclusion Algorithms. In A. Voronkov, editor, *Proceedings of Logic Programming and Automated Reasonning, LPAR'93*, volume 698, pages 86–97. LNAI, 1993.

[135] M-M. Corsini, K. Musumbu, and A. Rauzy. The μ-calculus over Finite Domains as an Abstract Semantics of Prolog. In *WSA'92 Workshop on Static Analysis (Bordeaux)*, volume 81–82 of *Bigre*, pages 51–59. Atelier Irisa, Sept. 23–25 1992.

[136] M-M. Corsini, K. Musumbu, and A. Rauzy. Toupie : a Symbolic Finite Domains Constraint Language, 1992. Post-conference wokshop of the 1992 Joint International Conference & Symposium on Logic Programming.

[137] M-M. Corsini and A. Rauzy. CLP($\mathcal{B}$) : Do it Yourself. In *Proceedings of the 8th Italian Conference on Logic Programming, GULP'93*. GULP, June 1993.

[138] M-M. Corsini and A. Rauzy. CLP($\mathcal{B}$) joue la transparence. In P. Ezequel, editor, *Actes des Journées Francophones sur la Programmation en Logique, JFPL'93*, pages 245–260. Teknea, June 1993.

[139] M-M. Corsini and A. Rauzy. Symbolic Model Checking and Constraint Logic Programming : a Cross-Fertilization. In Don Sannella, editor, *Proceedings of the European Symposium on Programming ESOP'94*, volume 788 of *LNCS*, pages 180–194. Springer Verlag, 1994.

[140] M.-M. Corsini and A. Rauzy. Toupie : un langage de programmation par contraintes pour l'analyse formelle de programmes concurrents. *Technique et Science Informatiques*, 14(6) :753–782, June 1995.

[141] M-M. Corsini and A. Rauzy. Toupie : the μ-calculus over finite domains as a constraint language. *Journal of Automated Reasonning*, 17 :143–171, 1997.

[142] O. Coudert. *SIAM : Une Boite à Outils Pour la Preuve Formelle de Systèmes Séquentiels.* PhD thesis, École Nationales Supérieure des Télécommunications, 1991.

[143] O. Coudert, C. Berthet, and J.-C. Madre. Verification of Sequential Machines using Boolean Fonctional Vectors. In L.J.M. Claesen, editor, *Formal VLSI Corectness Verification*, pages 179–196. North Holland, 1989.

[144] O. Coudert, C. Berthet, and J-C. Madre. Verification of Synchronous Sequential Machines Based on Symbolic Execution. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407, pages 365–373. LNCS, 1989.

[145] O. Coudert and J.-C. Madre. A Logically Complete Reasoning Maintenance System Based on Logical Constraint Solver. In *Proceedings of the International Join Conference on Artificial Intelligence, IJCAI'91*, pages 294–299, August 1991.

[146] O. Coudert and J.-C. Madre. A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions. In T. Knight and J. Savage, editors, *Advanced Research in VLSI and Parallel Systems*, pages 113–128, March 1992.

[147] O. Coudert and J.-C. Madre. Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions. In *Proceedings of the 29th ACM/IEEE Design Automation Conference, DAC'92*, June 1992.

[148] O. Coudert and J.-C. Madre. A New Implicit Graph Based Prime and Essential Prime Computation Technique. In *Proceedings International Symposium on Logic Synthesis and Microprocessor Architecture, ISKIT'92, Japan*, pages 125–131, 1992.

[149] O. Coudert and J.-C. Madre. A New Implicit Graph Based Prime and Essential Prime Computation Technique. In *Proceedings International Symposium on Logic Synthesis and Microprocessor Architecture, ISKIT'92, Japan*, pages 125–131, 1992.

[150] O. Coudert and J.-C. Madre. Fault Tree Analysis : 10^{20} Prime Implicants and Beyond. In *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'93*, January 1993. Atlanta NC, USA.

[151] O. Coudert and J.-C. Madre. MetaPrime : an Iteractive Fault Tree Analyser. *IEEE Transactions on Reliability*, 43(1) :121–127, March 1994.

[152] O. Coudert and J.-C. Madre. Une approche intentionnelle du calcul des implicants premiers et essentielles des fonctions booléennes. *RAIRO Informatique Théorique et Applications*, 28(2) :125–149, 1994.

[153] O. Coudert, J-C. Madre, and C. Berthet. Verifying Temporal Properties of Sequential Machines Without Building their State Diagrams. In E.M. Clarke and R.P. Kurshan, editors, *Proceedings ACM Computer-Aided Verification'90*, pages 75–84. DIMACS Series, june 1990.

[154] O. Coudert, J.-C. Madre, and H. Fraissé. New Qualitative Analysis Strategies in META-PRIME. In *Proceedings of the Annal Reliability and Maintainability Symposium, ARMS'94*, January 1994. Anaheim CA, USA.

[155] O. Coudert and J.C. Madre. A Unified Framework for the Formal Verification of Sequential Circuits. In *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'90*, pages 294–299, November 1990.

[156] B. Courcelle. Recursive Applicative Program Scheme. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 459–491. Elsevier, 1990.

[157] P. Cousot. Methods and Logics for Proving Programs. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 841–993. Elsevier, 1990.

[158] P. Cousot and R. Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction of approximations of fixpoints. In *Proceedings of the* 4^{th} *ACM Symposium on Principles of Programming Languages, POPL'77*, pages 238–252, 1977.

[159] P. Cousot and R. Cousot. Systematic design of program analysis framework. In *Proceedings of the* 6^{th} *ACM Symposium on Principles of Programming Languages, POPL'79*, pages 269–282, 1979.

[160] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3), 1992.

[161] P. Cousot, M. Falaschi, G.Filè, and A. Rauzy, editors. *Proceedings of the Workshop on Static Analysis, WSA'93*, volume 724, Padova, Italy, september 1993. LNCS.

[162] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (Washington, D.C., AAAI'1993)*, pages 21–27, 1993.

[163] J.M. Crawford and L.D. Auton. Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence, Special issue on transition phases in problem states*, 81 :31–57, 1996.

[164] N. Creignou and J.-J. Hébrard. On generating all satisfying truth assignments of a generalized cnf-formula. personal communication, 1996.

[165] N. Creignou and M. More. Complexity of satisfiability problems with symmetric polynomial clauses. *Journal of Logic and Computation*, 1996. To appear.

[166] N. Creigou and M. Hermann. Complexity of Generalized Satisfiability Counting Problems. *Information and Computation*, To appear, 1996.

[167] P. Crubillé. *Réalisation de l'outil MEC. Spécification fonctionnelle et architecture.* PhD thesis, LaBRI, Université Bordeaux I, 1989.

[168] C. Cubadda and M-D. Mousseigne. *Variantes de l'algorithme de SL-Resolution avec retenue d'informations.* PhD thesis, Université d'Aix-Marseille II, Faculté de Luminy, Laboratoire A.P.I., 1989.

[169] M. Dalal. Efficient Propositional Constraint Propagation. In *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence, AAAI'92*, pages 409–414, 1992. San Jose, California.

[170] M. Dalal. Tractable Deduction in Knowledge Representation Systems. In *Proceedings of the* 3^{th} *International Conference on Principles of Knowledge Representation and Reasoning, KR'92*, pages 393–402, 1992. Boston MA.

[171] M. Dalal and D.W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing Letters*, 44 :173–180, 1992.

[172] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *CACM*, 5 :394–397, 1962.

[173] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *JACM*, 7 :201–215, 1960.

[174] B. de Backer and H. Beringer. A CLP Language Handling Disjunctions of Linear Constraints. In *Proceedings of the* 10^{th} *International Conference on Logic Programming, ICLP'93*, pages 550–563. MIT Press, 1993.

[175] B. de Backer and H. Beringer. Satisfiability of Boolean formulas over linear constraints. In *Proceedings of the* 13*th International Joint Conference on Artificial Intelligence, IJCAI'1993)*, pages 296–301, 1993.

[176] K.A. de Jong and W.M. Spears. Using genetic algorithms to solve NP-complete problems. In *Proceedings of the Int. Conf. on Genetic Algorithms, ICGA'89*, pages 124–132, Fairfax, Virginia, june 1989.

[177] J. de Kleer. A Comparison of ATMS and CSP Techniques. In *Proceedings IJCAI'89*, 1989.

[178] R. Dechter and I. Rish. Directional Resolution : The Davis-Putnam Procedure, Revisited. In *Proceedings of the* 4^{th} *Conference on Principles of Knowledge Representation, KR'94*, pages 134–145, 1994.

[179] R. Dechter and P. van Beek. On the Minimality and Global Consistency of Row-Convex Constraint Networks. *Journal of the ACM*, 42(3) :543–461, 1995.

[180] Groupe DedAut. Aspects de la démonstration automatique en France. In *Proceedings of the* $3^{ième}$ *Journées du PRC Intelligence Artificielle*, pages 139–238. Teknea, 1990.

[181] J.P. Delahaye. Les lois de tout ou rien. *Pour la science*, 213, July 1995.

[182] D. Diaz and P. Codognet. A Minimal Extension of the WAM for `clp(FD)`. In *Proceedings of the* 10^{th} *International Conference on Logic Programming, ICLP'93*, pages 774–790. MIT Press, 1993.

[183] A. Dicky. An algebraic and algorithmic method for analyzing transition systems. *Theoretical Computer Science*, 46 :285–303, 1986.

[184] M. Dincbas, H.Simonis, and P. van Hentenryck. Solving a Cutting-Stock Problem in CLP. In *Proceedings of the* 5^{th} *Internation Conference on Logic Programming, ICLP'88*. MIT Press, 1988.

[185] M. Dincbas, H.Simonis, and P. van Hentenryck. Solving the sequencing car in constraint logic programming. In *Proceedings of the European Conference on Artificial Intelligence, ECAI'88*, 1988.

[186] M. Dincbas, P. van Hentenryck, H.Simonis, A. Aggoun, T. Graf, and F. Berthier. The Constraint Logic Programming Language CHIP. In *proceedings of the 2nd International Conference on Fifth Generation Computer Systems*, pages 249–264, 1988.

[187] W.F. Dowling and J.H. Gallier. Linear-time Algorithms for Testing the Satisfiablity of Propositional Horn Formulae. *J. Logic Programming*, 3 :267–284, 1984.

[188] R. Dreshler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Proceedings of the* 31*st Design Automation Conference, DAC'94*, pages 415–419, june 1994.

[189] O. Dubois. On the r,s-SAT satisfiability problem and a conjecture of Tovey. *Discrete Applied Mathematics*, 26 :51–60, 1990.

[190] O. Dubois, P. André, Y. Boufkhad, and J. Carlier. Can very simple algorithm be efficient for solving the sat problem, 1993. Position paper, DIMACS chalenge on Satisfiability Testing, to appear.

[191] O. Dubois, P. André, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. In *SAT Chalenge*, volume 26, pages 415–436. AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1996.

[192] O. Dubois and J. Carlier. Sur le problème de satisfiabilité. Communication at the Barbizon Workshop on SAT, october 1991.

[193] V. Dumortier, G. Janssens, M. Bruynooghe, and M. Codish. Freeness Analysis in Presence of Numerical Constraints. In D.S. Warren, editor, *Proceedings of the* 10^{th} *International International Conference on Logic Programming, ICLP'93*. MIT Press, 1993.

[194] B. Dutertre. *Spécification et preuves de systèmes dynamiques : Application à* SIGNAL. PhD thesis, IRISA, Université de Rennes, 1992.

[195] Y. Dutuit and A. Rauzy. A Linear Time Algorithm to Find Modules of Fault Trees. *IEEE Transactions on Reliability*, 45(3) :422–425, 1996.

[196] Y. Dutuit, A. Rauzy, and J.-P. Signoret. Réséda : a Reliability Network Analyser. In C. Cacciabue and I.A. Papazoglou, editors, *Proceedings of European Safety and Reliability Association Conference, ESREL'96*, volume 3, pages 1947–1952. Springer Verlag, 1996. ISBN 3-540-76051-2.

[197] Y. Dutuit, J. Dos Santos, and T. Bouhoufani. Modularisation et disjonction : une association pour traiter qualitativement et quantitativement les arbres de défaillance. In *Actes du* $8^{ième}$ *Colloque de fiabilité et maintenabilité,* $\lambda - \mu'89$, pages 189–198, 1992.

[198] C.A.J. Eijk and G.L.J.M. Janssen. Exploiting Structural Similarities in a BDD-based Verification Method. In T. Kropf and R. Kumar, editors, *Proceedings of* 2*nd International Conference on Theorem Provers in Circuit Design, TPCD'94*, volume 901 of *LNCS*, pages 110–125. Springer Verlag, 1994.

[199] E.A. Emerson and E.M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2 :241–266, 1982.

[200] E.A. Emerson and C.-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proceedings of Logics in Computer Science, LICS'86*, pages 267–278, 1986.

[201] E. Encrenaz. *Une méthode de vérification de propriétés de programmes VHDL basée sur des modèles formels de réseaux de Petri.* Thèse de doctorat, Université Pierre et Marie Curie, Paris VI, 1995.

[202] R. Enders, T. Filkorn, and D. Taubner. Generating BDDs for Symbolic Model Checking in CCS. *Journal of Distributed Computing*, 6 :155–164, 6 1993.

[203] S. Even, A. Itai, and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM J. Comput.*, 5 :691–703, 1976.

[204] E. Felt, G. York, R. Brayton, and A. Sangiovanni-Vincentelli. Dynamic Variable Reordering for BDD Minimization. In *Proceedings of the European Design Automation Conference EURO-DAC'93/EURO-VHDL'93*, pages 130–135. IEEE Computer Society Press, 1993.

[205] T. Filkorn, R. Schmid, E. Tidén, and P. Warkentin. Experiences from a Large Industrial Circuit Design Application. In *Proceedings of the North American Conference on Logic Programming (NACLP'91)*, pages 581–595. MIT Press, 1991.

[206] E. Freuder. Synthesizing Constraint Expressions. *Communications of the ACM*, 21(11) :958–966, 1978.

[207] E. Freuder. A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM*, 29(1) :24–32, 1982.

[208] L. Fribourg and M. Veloso Peixoto. Concurrent Constraint Automata. Research Report 93-10, École Normal Supérieure, May 1993. poster at ILPS'93, Vancouver.

[209] L. Fribourg and M. Veloso Peixoto. Automates concurrents à contraintes. *Technique et Science Informatiques*, 13(6) :837–866, 1994.

[210] S.J. Friedman and K.J. Supowit. Finding the Optimal Variable Ordering for Binary Decision Diagrams. *IEEE Transactions on Computers*, 39(5) :710–713, May 1990.

[211] A. Frieze and S. Suen. Analysis of Two Simple Heuristics on a Random Instance of k-sat. *Journal of Algorithms*, 20(2) :312–355, March 1996.

[212] H. Fujii, G. Ootomo, and C. Hori. Interleaved Based Variables Ordering Methods for Ordered Binary Decision Diagrams. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 38–41, 1993.

[213] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams. In *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'88*, pages 2–5, 1988.

[214] M. Fujita, H. Fujisawa, and Y. Matsugana. Variable Ordering Algorithm for Ordered Binary Decision Diagrams and Their Evalutation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(1) :6–12, January 1993.

[215] M. Fujita, Y. Matsunaga, and N. Kakuda. On the Variable Ordering of Binary Decision Diagrams for the Application of Multilevel Logic Synthesis. In *Proceedings of European Conference on Design Automation, EDAC'91*, pages 50–54, 1991.

[216] F. Gagnon, J.-C. Grégoire, and D. Zampunieris. Sharing Trees for "on-the-fly" Verification. In *Proceedings of International IFIP Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, FORTE'95*, 1995. Also Research Paper RP-95-026, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium.

[217] Z. Galil. On resolution with clauses of bounded size. *SIAM J. Comput.*, 6(3) :444–459, 9 1977.

[218] Z. Galil. On the Complexity of Regular Resolution and the Davis and Putnam's Procedure. *Theoretical Computer Science*, 4 :23–46, 1977.

[219] J.P. Gallagher and D.A. de Waal. Regular Approximations of Logic Programs and their uses. Technical Report CSTR-92-06, Computer Science Department of Bristol University, March 1992.

[220] G. Gallo and D. Pretolani. A new algorithm for the propositional satisfiability problem. *Discrete Applied Mathematics*, 60 :159–179, 1995.

[221] G. Gallo and M.G. Scutellà. Polynomially Solvable Satisfiability Problems. *Information Processing Letters*, 29 :221–227, 1988.

[222] G. Gallo and G. Urbani. Algorithms for Testing the Satisfiablity of Propositional Formulae. *Journal of Logic Programming*, 7 :45–61, 1989.

[223] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Freeman, San Fransisco, 1979.

[224] R. Génisson and A. Rauzy. Aspects algorithmiques des classes polynomiales du problème sat et des problèmes de satisfaction de constraintes. In *Actes du* $10^{ième}$ *congrès Reconnaissance de Formes et Intelligence Artificielle, RFIA'96*, pages 97–108. AFCET-AFIA, 1996.

[225] R. Génisson and A. Rauzy. Efficient Horn renaming : yet a Davis and Putnam's procedure. In *Résolution Pratique de Problèmes NP-Complets, CNPC'96*, pages 169–184. Teknea, 1996. Also, Technical Report 1092-95, LaBRI – URA CNRS 1304 – Université Bordeaux-I, 1995.

[226] R. Génisson and L. Saïs. Some ideas on random generation of k-SAT instances. In J.M. Crawford and B. Selman, editors, *Proceedings of post-conference workshop on Experimental Evaluation of Reasoning and Search Methods, AAAI'94*, pages 91–92, 1994.

[227] I.P. Gent and T. Walsh. Towards an Understanding of Hill-climbing Procedures for SAT. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (Washington, D.C., AAAI'1993)*, pages 28–33, 1993.

[228] I.P. Gent and T. Walsh. An Empirical Analysis of Search in GSAT. *Journal of Artificial Intelligence Research*, pages 47–59, 1994.

[229] I.P. Gent and T. Walsh. The SAT Phase Transition. In A.G. Cohn, editor, *Proceedings of 11th European Conference on Artificial Intelligence, ECAI'94*, pages 105–109. Wiley, 1994.

[230] I.P. Gent and T. Walsh. Unsatisfied Variables in Local Search. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions*, pages 73–85. IOS Press, Amsterdam, 1995.

[231] J. Gergov and C. Meinel. Efficient Boolean Manipulation with OBDDs can be extended to FBDDs. *IEEE Transactions on Computers*, 43 :1197–1209, October 1994.

[232] T. Getzinger. The Costs and Benefits of Abstract Interpretation-driven Prolog Optimization. In B. Le Charlier, editor, *Proceedings of the 1^{rst} International Symposium on Static Analysis*, volume 864, pages 1–25. LNCS, 1994.

[233] M. Ghallab and E. Escalada-Imaz. A linear control algorithm for a class of rule-based systems. *Journal of Logic Programming*, 11 :117–132, 1991.

[234] R. Giacobazzi, S. Debray, and G. Levi. A Generalized Semantics for Constraint Logic Programs. In *Proceedings of FGCS'92*, pages 581–591, 1992.

[235] F. Giannesini, H. Kanoui, R. Paséro, and M. van Caneghem. *Prolog*. InterEditions, 1985. ISBN 2 7296 0076 0.

[236] F. Glover. Tabu search – Part I. *ORSA Journal of Computing*, 1 :190–206, 1989.

[237] F. Glover. Tabu search – Part II. *ORSA Journal of Computing*, 2 :4–32, 1990.

[238] A. Goerdt. Davis-Putnam resolution versus unrestricted resolution. *Annals of Mathematics and Artificial Intelligence*, 6 :169–184, 1992.

[239] A. Goerdt. A threshold for unsatifiability. In I.M. Havel and V. Koubek, editors, *Proceedings of Mathematical Foundations of Computer Science, MFCS'92*, pages 264–272, August 1992.

[240] A. Goerdt. Unrestricted resolution versus n-resolution. *Theoretical Computer Science*, 93 :159–167, 1992.

[241] A. Goldberg. Average case complexity of the satisfiability problem. In *Proceedings of the 4th workshop on Automated Deduction*, pages 1–6, 1979.

[242] A. Goldberg, P.W. Purdom Jr, and C.A. Brown. Average time analysis of simplified Davis and Putnam Procedures. *Information Processing Letters*, 15 :72–75, 1982. see also errata vol. 16, p213,1983.

[243] Jun Gu. Global Optimization for Satisfiability (SAT) problem. *IEEE Transactions on Knowledge and data engineering*, 6(3) :361–381, June 1994.

[244] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66, 1994.

[245] A. Haken. The intractability of the resolution. *Theoretical Computer Science*, 39 :297–308, 1985.

[246] N. Halbwachs. Delay Analysis in Synchronous Programs. In *Proceedings of the 5th international conference on Computer Aided Verification CAV'93*, volume 697 of *LNCS*. Springer Verlag, June 1993.

[247] N. Halbwachs. BAC : A Boolean Automaton Checker (Version 1). Verimag, Miniparc-Zirst, 38330 - Montbonnot, France, email : Nicolas.Halbwachs@imag.fr, feb 1994.

[248] P. Hall. On representatives of subsets. *J. London Math. Society*, 10 :26–30, 1935.

[249] C.C. Han and C.H. Lee. Comments on Mohr and Henderson's Path Consistency Algorithm. *Artificial Intelligence*, 36 :125–130, 1988.

[250] Sang Hoon Han, Tae Woon Kim, and Kun Joong Yoo. Development of an Integrated Fault Tree Analysis Computer Code MODULE by Modularization Technique. *Reliability Engineering and System Safety*, 21 :145–154, 1988.

[251] J.P. Hansen and N. Goto. Adaptative Variable Ordering in Shared Binary Decision Diagrams. ULSI Research Laboratories, Toshiba Corporation.

[252] P. Hansen and B. Jaumard. Uniquely solvable quadratic boolean equations. *Discrete Applied Mathematics*, 12 :147–154, 1985.

[253] P. Hansen, B. Jaumard, and M. Minoux. A linear expected-time algorithm for deriving all logical conclusion implied by a set of Boolean inequalities. *Math. Programming*, 34 :223–231, 1986.

[254] J.K. Hao and R. Dorne. A new population-based method for satisfiability problems. In *Proceedings of the 11th European Conf. on Artificial Intelligence, ECAI'94*, pages 135–139, Amsterdam, August 1994.

[255] R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problem. *Artificial Intelligence*, 14 :263–313, 1980.

[256] F. Harche, J.N. Hooker, and G.L. Thompson. A Computational Study of Satisfiability Algorithms for Propositional Logic. Management Science Research Report MSRR-567, Working Paper 1991-27, Graduate School of Industrial Administration Carnegie Mellon University, 1991.

[257] J.-J. Hébrard. Résolution unitaire, Horn-renommage et procédure de Davis et Putnam. Rapport interne, les cahiers du laiac, LAIAC, Université de Caen, 1993.

[258] J.-J. Hébrard. A linear algorithm for renaming a set of clauses as a Horn set. *Theoretical Computer Science*, 124 :343–350, 1994.

[259] J.-J. Hébrard. Base de Horn d'un ensemble de clauses. Les cahiers de greyc, numéro 2, GREYC, Université de Caen, France, 1995.

[260] J.-J. Hébrard. Unique Horn renaming and Unique 2-Satisfiability. *Information Processing Letters*, 54 :235–239, 1995.

[261] K. Heninger. Specifying Software Requirements for Complex Systems : New Techniques and Their Applications. *IEEE Transactions on Software Engineering*, 6 :2–12, 1 1980.

[262] M. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *J. Assoc. Comput. Mach.*, 32 :137–161, 1985.

[263] L. Henschen and L. Wos. Unit refutations and Horn sets. *JACM*, 21(4) :590–605, October 1974.

[264] M. Hermenegildo, R. Warren, and S. Debray. Global Flow Analysis as a Practical Compilation Tool. *Journal of Logic Programming*, 13(4) :349–367, 1992.

[265] G.J. Holzmann. *Design and validation of computer protocols.* software series. Prentice hall, 1991. ISBN 0-13-539925-4.

[266] J.N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4 :45–69, 1988.

[267] J.N. Hooker. Resolution vs. cutting plane solution of inference problems in propositional logic : some computational experience. *Operation Research Letters*, 7(1) :1–7, 1988.

[268] J.N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathemathics and Artificial Intelligence*, 6 :271–286, 1992.

[269] J.N. Hooker. Solving the incremental satisfiability problem. *Journal of Logic Programming*, 15 :177–186, 1993.

[270] J.N. Hooker and C. Fedjki. Branch-and-cut solution of inference problems in propositional logic. *Annals of Mathemathics and Artificial Intelligence*, 1 :123–139, 1990.

[271] J.N. Hooker and V. Vinay. Branching Rules for Satisfiability. *Journal of Automated Reasoning*, 15 :359–383, 1995.

[272] A. Horn. On sentences which are true of direct unions algebras. *Journal of Symbolic logic*, 16 :14–21, 1951.

[273] J. Hsiang. Refutational Theorem Proving using Term-Rewriting Systems. *Artificial Intelligence*, 25 :255–300, 1985.

[274] G. Huet, G. Kahn, and C. Paulin-Mohring. The Coq Proof Assistant, A Tutorial. Technical report, INRIA Rocquencourt and CNRS ENS Lyon, 1995.

[275] E. Humbert. *Études de complexité au pire dans le problème de satisfiabilité logique*. Thèse de doctorat, Université René Descartes - Paris V, Novembre 1995.

[276] T. Hutinet, S. Lajeunesse, and L. Martin. Atelier FIABEX, vers une intégration des études SdF en phase de conception. In *Actes du Congrès $\lambda\mu$ 94, ESREL'94*, pages 694–700, La Baule, 1994.

[277] T. Huynh and C. Lassez. A CLP($\mathcal{R}$) Option Trading Analysis System. In *Proceedings of the 5^{th} Internation Conference on Logic Programming, ICLP'88*, pages 59–69. MIT Press, 1988.

[278] N. Immerman. Relational Queries Computable in Polynomial Time. *Information and Control*, 68 :86–104, 1986.

[279] N. Ishuira, H. Sawada, and S. Yajima. Minimization of Binary Decision Diagrams Based on Exchanges of Variables. In *Proceedings of the IEEE International Conference on Computer Aided Design, ICCAD'91*, pages 472–475, November 1991. Santa Clara CA, USA.

[280] G.F. Italiano. Amortized efficiency of path retrieval data structure. *Theoretical Computer Science*, 48, 1986.

[281] R. Jacobi, N. Calazans, and C. Truellemans. Incremental Reduction of Binary Decision Diagrams. In *Proceedings of the International Symposium on Circuit Design'91*, june 1991.

[282] J. Jaffar and J.L. Lassez. Constraint Logic Programming. In *Proceedings of the 14^{th} ACM Symposium on Principles of Programming Languages, POPL'87*, pages 111–119, January 1987.

[283] J. Jaffar and J.L. Lassez. Constraint Logic Programming : A Survey. *Journal of Logic Programming*, 19 & 20 :503–581, 1994.

[284] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) Language and System. *ACM Transactions on Programming Languages*, 14(3) :339–395, 1992.

[285] G. Janssens and M. Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. *Journal of Logic Programming*, 13(3), 1992.

[286] B. Jaumard, P. Marchioro, A. Morgana, R. Petreschi, and B. Simeone. On line 2-Satisfability. In M.C. Golombic, editor, *Annals of Mathematics and Artificial Intelligence*, volume 1, pages 155–165. J.C. Baltzer AG, 1990.

[287] S. Jeannicot, L. Oxusoff, and A. Rauzy. Évaluation Sémantique en Calcul Propositionnel. *Revue d'Intelligence Artificielle*, 2 :41–60, 1988.

[288] P. Jeavons, D. Cohen, and M. Gyssens. A Unifying Framework for Tractable Constraints. In U. Montanari and F. Rossi, editors, *Proceedings of the International Conference on Principle of Constraint Programming, CP'95*, volume 976 of *LNCS*, pages 276–291. Springer Verlag, 1995.

[289] R.J. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. *Annals of Mathematics and Artificial Intelligence*, 1 :167–188, 1990.

[290] D.S. Johnson. A Catalog of Complexity Classes. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 67–162. Elsevier, 1990.

[291] J.L. Johnson. A Neural Network Approach to the 3-Satisfiability Problem. *Journal of Parallel and Distributed Computing*, 6 :435–449, 1989.

[292] A. Kamath, R. Motwani, K. Palem, and P. Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. In *Proceedings of the* 35*th Annual Symposium on Foundations of Computer Science*, pages 592–603, 1994.

[293] R.M. Karp. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–104. Plenun, New York, 1972.

[294] K. Karplus. Using if-then-else DAGs for multi-level logic minimization. *Advanced Research in VLSI*, pages 101–118, 1989.

[295] A. Kean and H. Tsiknis. An incremental Method for Generating Prime Implicants. *Journal of Symbolic Computation*, 9(2) :185–206, 1990.

[296] U. Kebschull and R. Rosenstiel. Efficient Graph-Based Computation and Manipulation of Functional Decision Diagrams. In *Proceedings of European Conference on Design Automation, EDAC'93*, pages 278–282, 1993.

[297] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220 :671–680, 1983.

[298] S. Kirkpatrick and B. Selman. Critical Behavior in the Satisfiability of Random Boolean Expression. *Science*, 264 :1297–1301, may 1994.

[299] D.E. Knuth. Nested Satisfiability. *Acta Informatica*, 28 :1–6, 1990.

[300] T. Kohda, E.J. Henley, and K. Inoue. Finding Modules in Fault Trees. *IEEE Transactions on Reliability*, 38(2) :165–176, June 1989.

[301] E. Kousoupias and C.H. Papadimitriou. On the greedy algorithm for satisfiability. *Information Processing Letters*, 43 :53–55, 1992.

[302] R. Kowalski and D. Kuehner. Linear Resolution with Selection Function. *Artificial Intelligence*, pages 227–259, 1971.

[303] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27 :333–354, 1983.

[304] J. Kratochvil, P. Savicky, and Z. Tuza. One more occurrence of variable makes satisfiability jump from trivial to NP-complete. *SIAM Journal of Computing*, 22 :203–210, 1993.

[305] B. Krishnamurty. Short Proofs for Tricky Formulas. *Acta Informatica*, 22 :253–275, 1985.

[306] B. Krishnamurty and R.N. Moll. Examples of Hard Tautologies in the Propositional Calculus. In *Proceedings of the* 13^{th} *Symposium on Theory of Computing*, 1981.

[307] Y.-T. Lai, M. Pedram, and S.B.K. Vradhula. Formal verification Usign Edge-Valued Binary Decision Diagrams. *IEEE Transactions on Computers*, 45(2) :247–255, February 1996.

[308] Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proceedings of the 29th Design automation Conference, DAC'92*, pages 608–613, 1992.

[309] S. Lajeunesse, T. Hutinet, and J.-P. Signoret. Automatic Fault Trees Generation on Dynamic Systems. In *Proceedings of the European Safety and Reliability Association Conference, ESREL'96*, pages 1553–1559. European Safety and Reliability Association, 1996.

[310] J.-C. Laprie, editor. *Guide de la sûreté de fonctionnement*. Cépaduès, 1995. ISBN 2-85428-382-1.

[311] T. Larrabee. Test Pattern Generation Using Boolean Satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1) :4–15, January 1992.

[312] T. Larrabee and Y. Tsuji. Evidence for a Satisfiability Threshold for Random 3CNF Formulas. In H. Hirsh and al., editors, *Proceedings of Spring Symposium on Artificial Intelligence and NP-Hard Problems (Stanford CA 1993)*, pages 112–118, 1993.

[313] J.L. Laurière. A Language and a Program for Stating and Solving Combinatorial Problems. *Artificial Intelligence*, 10 :29–127, 1 1978.

[314] A. Laviron, A. Carnino, and J.-C. Manarache. Escaf, a new a cheap system for complex reliability analysis and computation. *IEEE Transactions on Reliability*, R-31 :339–348, 1982.

[315] E. Ledinot. Programs proved correct by construction, application to bdd algorithms. Personal Communication, 1994.

[316] W.S. Lee, D.L. Grosh, F.A. Tillman, and C.H. Lie. Fault tree analysis, methods and applications : a review. *IEEE Transactions on Reliability*, 34 :194–303, 1985.

[317] H.R. Lewis. Renaming a Set of Clauses as a Horn Set. *JACM*, 25(1) :134–135, 1978.

[318] O. Lhomme. Consistency techniques for numeric CSPs. In *Proceedings of the* 13^{th} *International Conference on Artificial Intelligence, IJCAI'93*, pages 232–238, 1993.

[319] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal of Computing*, 11(2) :329–343, May 1982.

[320] N. Limnios. *Arbres de défaillance*. Traité des Nouvelles Technologies. Hermes, 1991. in french.

[321] B. Lin, O. Coudert, and J.-C. Madre. Symbolic Prime Generation for Multiple-Valued Functions. In *Proceedings of the 29th ACM/IEEE Design Automation Conference, DAC'92*, pages 40–44, 1992.

[322] G. Lindhorst and F. Shahrokhi. On renaming a set of clauses as a Horn set. *Information Processing Letters*, 30 :289–293, 1989.

[323] R.J. Lipton. DNA Solution of Hard Computational Problems. *Science*, 268 :542–545, 1995.

[324] M.O. Locks. Fault trees, prime implicants and noncoherence. *IEEE Transactions on Reliability*, R-29 :130–135, June 1980.

[325] M.O. Locks. Modularizing, minimizing and interpreting the K & H fault tree. *IEEE Transactions on Reliability*, R-30 :411–415, December 1981.

[326] D. Loveland. *Automated Theorem Proving : A Logical Basis.* North Holland, 1978.

[327] L. Löwenheim. Über das Auflösungsproblem im logischen Klassenkalkul. *Sitzunger. Berlin Math. Ges.*, 7 :89–94, 1908.

[328] L. Löwenheim. Über die Auflösung von Gleichungen im logischen Gebietkalkul. *Math. Ann.*, 68 :169–207, 1910.

[329] L. Löwenheim. Über Transformationen im Gebietkalkul. *Math. Ann.*, 73 :245–272, 1913.

[330] J.W. Loyd. *Foundations of Logic Programming.* Symbolic Computation. Springer Verlag, 1984.

[331] E.L. Lozinskii. A simple test improves checking satisfiability. *Journal of Logic Programming*, 15 :99–111, 1993.

[332] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1) :99–118, 1977.

[333] J.-C. Madre. *Conception et Applications d'une Procédure de Décision en Logique Propositionnelle Basée sur les Graphes de Décision Binaires.* Rapport scientifique présenté pour obtenir l'habilitation à diriger des recherches, Centre de Recherche BULL, 1993.

[334] J.-C. Madre and J.P. Billon. Correctness using Formal Comparison Between Expected and Extracted Behaviour. In *Proceedings of the 25th ACM/IEEE Design Automation Conference, DAC'88*, pages 205–210, June 1988.

[335] J.C. Madre. *Priam un outil de vérification formelle de circuits intégrés digitaux.* PhD thesis, École Nationale Supérieure des Télécommunications, 1990. In french.

[336] M.J. Maher. Logic semantics for a class of committed-choice programs. In *Proceedings of the* 4^{th} *International Conference on Logic Programming, ICLP'87*, pages 858–876. MIT-Press, 1987. Melbourne, Australia.

[337] S. Malik, A.R. Wang, R.K Brayton, and A. Sangiovanni-Vincentelli. Logic Verification using Binary Decision Diagrams in Logic Synthesis Environment. In *Proceedings of the IEEE International Conference on Computer Aided Design, ICCAD'88*, pages 6–9, November 1988. Santa Clara CA, USA.

[338] H. Mannila and K. Mehlorn. A fast algorithm for renaming a set of clauses as a Horn set. *Information Processing Letters*, 21 :269–272, 1985.

[339] A. Mantsivoda. Flang and its Implementation. In *Proceedings of the Symposium on Programming Language Implementation and Logic Programming, PLILP'93*, volume 714, pages 151–166. LNCS, 1993.

[340] H. Marchand and M. Le Borgne. Typage des graphes de décision ternaires. Technical Report 2185, INRIA Rennes – Programme 2, 1994.

[341] P. Marquis. Knowledge Compilation Using Theory Prime Implicate. In *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI'95*, pages 837–845, 1995. 1 55860-363-8.

[342] J. Martin and T. Nipkow. Unification in Boolean Rings. In *Proceedings of the* 8^{th} *Conference on Automated Deduction, CADE'86*, pages 506–513, 1986. Oxford, England.

[343] J. Martin and T. Nipkow. Boolean Unification : The Story So Far. *Journal of Symbolic Computation*, 7 :275–293, 1989.

[344] J-L. Massat. Using Local Consistency Technics to Solve Boolean Constraints. In A. Colmerauer and F. Benhamou, editors, *Constraint Logic Programming : Selected Research*, pages 223–235. MIT Press, 1993. ISBN 0-262-02353-9.

[345] L. Mauborgne. Abstract interpretation using TDGs. Rapport de stage de dea, LIENS, École Normale Supérieure, September 1993.

[346] M.Billaud, P.Castéran, M-M.Corsini, K.Musumbu, and A.Rauzy, editors. *Proceedings of the Workshop on Static Analysis, WSA'92*, Bordeaux, september 1992. Bigre numéro 81–82, Atelier Irisa.

[347] M.Billaud, P.Castérana, M-M.Corsini, K.Musumbu, and A.Rauzy, editors. *Journées de Travail sur l'Analyse Statique de Programmes Equationnels Fonctionnels et Logiques JTAS-PEFL'91*, Bordeaux, october 1991. Bigre numéro 74, Atelier Irisa.

[348] J.C.C McKinsey. The decision problem for some classes of sentences without quantifiers. *Journal of Symbolic Logic*, 8 :61–76, 1943.

[349] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publisher, 1993. ISBN 0-7923-9380-5.

[350] B. Meltzer. Theorem-proving for computers : some results on resolution and renaming. *Comp. Journal*, 8 :493–495, 1966.

[351] S. Menju, K. Sakai, Y. Sato, and A. Aiba. A Study on Boolean Constraints Solvers. In A. Colmerauer and F. Benhamou, editors, *Constraint Logic Programming : Selected Research*, pages 253–267. MIT Press, 1993. ISBN 0-262-02353-9.

[352] M. Mezard, G. Parisi, and M.-A. Virasoro. *Spin glass theory and beyond*, volume 9 of *Lecture notes in Physics*. World Scientific, 1987. ISBN 9971-50-115-5.

[353] R. Milner. *A calculus of communicating systems*. LNCS. Springer Verlag, Berlin, 1980.

[354] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proceedings of the 30th ACM/IEEE Design Automation Conference, DAC'93*, pages 272–277, 1993.

[355] S. Minato. Calcultation of Unate Cube Set Using zero-Suppressed BDDs. In *Proceedings of the 31th ACM/IEEE Design Automation Conference, DAC'94*, pages 420–424, 1994.

[356] S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagrams with Attributed Edges for Efficient Boolean Function Manipulation. In L.J.M Claesen, editor, *Proceedings of the 27th ACM/IEEE Design Automation Conference, DAC'90*, pages 52–57, June 1990.

[357] M. Minoux. LTUR : A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and its Computer Implementation. *Information Processing Letters*, 29 :1–12, 1988.

[358] M. Minoux. The Unique Horn-Satisfiability problem and quadratic Boolean equations. In *Annals of Mathematics and Artificial Intelligence*, volume 6, pages 253–266. J.C. Baltzer A.G. Scientific Publishing Compagny, 1988.

[359] S. Minton, M.D. Johnson, A.B. Philips, and P. Laird. Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the 9th National Conference on Artificial Intelligence, AAAI'90*, pages 17–24, 1990.

[360] K.R. Misra. *Reliability analysis and prediction.* Elsevier, 1992.

[361] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings Tenth National Conference on Artificial Intelligence (AAAI'92)*, 1992.

[362] I. Mitterreiter and F-J. Radermacher. Running Time Experiments on some Algorithms for Solving Propositional Satisfiability Problems. Technical report, Forschungsintitut für Anwendungsorientierte Wissensverarbeitung, Helmholtzste. 16, Postfach 2060, D-7900 Ulm Germany, 1991.

[363] R. Mohr and T. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 38 :225–233, 1986.

[364] G. Möller. A Logic Programming Tool for Qualitative System Design. In *APL Quote Quad Proc., APL86*, volume 16 :4, pages 266–271, 1986.

[365] B. Monien and E. Speckenmeyer. Solving Satisfiability in Less than 2^n Steps. *Discrete Applied Math.*, 10 :287–295, 1985.

[366] H. Montanari. Network of Constraints : Fundamental Properties and Applications to Picture Processing. *Information Sciences*, 7 :95–132, 1974.

[367] U. Montanari and F. Rossi, editors. *Proceedings of the First International Conference on Principles and Practice of Constraint Programming, CP'95*, volume 976 of *LCNS*. Springer Verlag, 1995.

[368] P. Morris. The Breakout Method For Escaping From Local Minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (Washington, D.C., AAAI'1993)*, pages 40–45, 1993.

[369] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995. ISBN 0-521-47465-5.

[370] W.J. Older and F. Benhamou. Programming in CLP(BNR). In *Proceedings of the 1^{fst} Workshop on Principles and Practice of Constraint Programming, PPCP'93*, 1993. Newport RI, USA.

[371] J.S. Ostroff. Constraint Logic Programming for Reasoning about Discrete Event Processes. *Journal of Logic Programming*, 11 :243–270, 11 1991.

[372] L. Oxusoff and A. Rauzy. *L'Évaluation Sémantique en Calcul Propositionnel.* PhD thesis, GIA – Université de Aix-Marseille II, Janvier 1989.

[373] L. Oxusoff and A. Rauzy. Towards a Better Understanding of SL-Resolution. In P. Jorrand and V. Sugrev, editors, *Proceedings Artificial Intelligence V Methodology, Systems, Applications, AIMSA'94.* North Holland, 1994.

[374] L.B. Page and J.E. Perry. An algorithm for exact fault-tree probabilities. *IEEE Transactions on Reliability*, 35(5) :544–558, 1986.

[375] A. Pagès and M. Gondran. *Fiabilité des systèmes*. Collection de la Direction des Études et Recherches d'Électricité de France. Eyrolles, 1980. ISSN 0399-4198.

[376] C.H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4 :237–244, 1977.

[377] C.H. Papadimitriou. On selecting a Satisfing Truth Assignment. In *Proceedings of the Conference on Foundations of Computer Science, FCS'91*, pages 163–169, 1991.

[378] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994. ISBN 0-201-53082-1.

[379] O. Papini and A. Rauzy. A Mixed Approach of Revision in Propositional Calculus. In R. Kruse M. Clarke and S. Moral, editors, *Proceedings of European Conference on Symbolic and Quantitative Approaches to Reasonning about Uncertainty, ECSQARU'93*, volume 747, pages 296–303. LNCS, November 1993.

[380] O. Papini and A. Rauzy. Revision in Propositional Calculus. In *Proceedings of the post-conference Workshop of the International Joint Conference of Artificial Intelligence, IJCAI'93 : Revision and Change*, 1993.

[381] O. Papini and A. Rauzy. Révision : mettons un bémol (résumé étendu). In *Actes des Rencontres Françaises sur l'Intelligence Artificielle, RFIA'94*, 1994.

[382] O. Papini and A. Rauzy. Revision in Extended Propositionnal Calculus. In C. Froidevaux and J. Kohlas, editors, *Proceedings of European Conference on Symbolic and Quantitative Approaches to Reasonning about Uncertainty, ECSQARU'95*, volume 946, pages 328–335. LNAI, 1995.

[383] O. Papini and A. Rauzy. Révision : mettons un bémol. *Revue Française d'Intelligence Artificielle*, 9(4) :455–474, 1995.

[384] R. Paquay and D. Zampunieris. MEC with Sharing Trees. Research Paper RP-96-006, Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1996.

[385] D. Park. Fixpoint Induction and Proofs of Program Properties. *Machine Intelligence*, 5, 1970.

[386] G.L. Peterson. MYTHS about the Mutual Exclusion Problem. *Information Processing Letters*, 12(3) :115–116, June 1981.

[387] R. Petreschi and B. Simeone. Experimental Comparison on 2-Satisfiability Algorithms. *RAIRO Recherche Opérationelle*, 25 :241–264, 8 1991.

[388] D.A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2 :293–304, 1986.

[389] A. Podelski, editor. *Constraint Programming : Basics and Trends*, volume 910. LCNS, 1995.

[390] Projet Classes Polynomiales. Projet inter-prc "classes polynomiales" : Travaux et résultats. In *Actes des* $5^{ième}$ *Journées du PRC Intelligence Artificielle*, pages 5–28. Teknea, 1995.

[391] D. Pretolani. A linear time algorithm for unique Horn satisfiability. *Information Processing Letters*, 48 :61–66, 1993.

[392] G.M. Provan. The Computational Complexity of Multiple-Context Truth Maintenance System. In *Proceedings of the European Conference on Artificial Intelligence, ECAI'90*, pages 523–527, 1990.

[393] G.M. Provan and M.O. Ball. The complexity of counting cuts and computing the probability that a graph is connected. *SIAM Journal of Computing*, 12(4) :777–788, November 1983.

[394] G.M. Provan and M.O. Ball. Computing network reliability in time polynomial in the number of cuts. *Operation Research*, 32 :516–526, 1984.

[395] J.S. Provan. Bounds on the reliability of networks. *IEEE Transactions on Reliability*, R-35 :260–268, 1986.

[396] J.S. Provan. The complexity of reliability computations in planar and acyclic graphs. *SIAM Journal on Computing*, 15 :694–702, 1986.

[397] J.-F. Puget. A C++ implementation of CLP. In *Proceedings of SPICIS'94*, Singapore, 1994. Available at http ://www.ilog.com.

[398] P.W. Purdom. Search Rearrangement Backtracking and Polynomial Average Time. *Artificial Intelligence*, 21 :117–133, 1983.

[399] P.W. Purdom. Solving Satisfiability with Less Searching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(4) :510–513, July 1984.

[400] P.W. Purdom and C.A. Brown. The Pure Literal Rule and Polynomial Average Time. *SIAM Journal of Computing*, 14 :943–953, 1985.

[401] P.W. Purdom, C.A. Brown, and E.L. Robertson. Backtracking with multi-level dynamic search rearrangement. *Acta Informatica*, 15 :99–113, 1981.

[402] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proceedings of the International Symposium on Programming*, volume 137, pages 337–351. LNCS, 1982.

[403] W.V. Quine. A proof procedure for quantification theory. *Journal of Symbolic Logic*, pages 141–149, june 1955.

[404] W.V.O. Quine. On cores and prime implicants of truth functions. *American Mathematics Monthly*, 66 :755–760, 1959.

[405] R. Raghvan, J. Cohoon, and S. Sahni. Single bend wiring. *Journal of Algorithms*, 7 :232–257, 1986.

[406] A. Rauzy. Introducing Constraints over a Ring in Prolog. In *Proceedings of Russian Conference of Logic Programming, RCLP'91*. LNCS, 1991.

[407] A. Rauzy. Knowledge extraction in trivalued propositional logic. In P. Siegel and K. Kruse, editors, *Proceedings of European Conference on Symbolic and Quantitative Approaches of Uncertainty, ECSQAU'91*, volume 548, pages 287–291. LNCS, 1991.

[408] A. Rauzy. A knowledge extraction mechanism from formulae over a ring. In M. de Glas and D. Gabbay, editors, *Proceedings of the First World Conference in the Fundamentals of Artificial Intelligence, WOFCAI'91*, pages 427–438. Angkor, july 1991.

[409] A. Rauzy. Boolean unification : an efficient algorithm. In A. Colmerauer and F. Benhamou, editors, *Constraint Logic Programming : Selected Research*, pages 237–251. MIT Press, 1993. ISBN 0-262-02353-9.

[410] A. Rauzy. New Algorithms for Fault Trees Analysis. *Reliability Engineering & System Safety*, 05(59) :203–211, 1993.

[411] A. Rauzy. Aulne Version 0.2 : User's Guide. Technical Report 834-94, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1994.

[412] A. Rauzy. Binary Decision Diagrams : a Tutorial Construction. Technical Report 829–94, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1994.

[413] A. Rauzy. Binary Decision Diagrams and Fault Tree Analysis : a Tutorial Presentation. Technical Report 1066–95, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1994.

[414] A. Rauzy. Notes on the Design of an Open Boolean Solver. In P. van Hentenryck, editor, *Proceedings of the* 11^{th} *International Conference on Logic Programming, ICLP'94*, pages 354–368. MIT Press, 1994.

[415] A. Rauzy. On the Complexity of the Davis and Putnam's Procedure on Some Polynomial Sub-Classes of SAT. Technical Report 806-94, LaBRI, URA CNRS 1304, Université BordeauxI, 9 1994.

[416] A. Rauzy. Pardi Version 1.2 : User's Manual. Technical Report 876–94, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1994.

[417] A. Rauzy. Toupie Version 0.25 : User's Manual. Technical Report 959-94, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1994.

[418] A. Rauzy. Aralia Version 1.0 : Developer's Guide. Technical report, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1995.

[419] A. Rauzy. Aralia version 1.0 : the Toolbox Manual. Technical report 1093-95, LaBRI – URA CNRS 1304 – Université Bordeaux-I, 1995.

[420] A. Rauzy. On the Random Generation of 3-SAT-Instances. Technical Report 1060–95, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1995. Submitted to Revue Française d'Intelligence Artificielle.

[421] A. Rauzy. Polynomial restrictions of SAT : What can be done with an efficient implementation of the Davis and Putnam's procedure. In U. Montanari and F. Rossi, editors, *Proceedings of the International Conference on Principle of Constraint Programming, CP'95*, volume 976 of *LNCS*, pages 515–532. Springer Verlag, 1995.

[422] A. Rauzy. Toupie : a Constraint Language for Model Checking. In M. Nivat and A. Podelski, editors, *Constraints : Basics and Trends*, volume 910, pages 193–207. Springer Verlag (LNCS), 1995.

[423] A. Rauzy. An Introduction to Binary Decision Diagrams and some of their Applications to Risk Assessment. In O. Roux, editor, *Actes de l'école d'été, Modélisation et Vérification de Processus Parallèles, MOVEP'96*, 1996. Also Technical Report LaBRI number 1121-96.

[424] A. Rauzy. Utilisation du langage de contraintes toupie pour analyser des spécifications "à la parnas". In J.J. Lesage, editor, Actes du congrès AFCET sur la Modélisation de systèmes réactifs, pages 81–89, 1996.

[425] A. Rauzy and B. Sanscartier. Aralia Version 1.0 : User's Guide. Technical Report 1065-95, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1995.

[426] O. Ridoux and H. Tonneau. Une mise en œuvre de l'unification d'expression booléenne. In S. Bourgault et M. Dincbas, editor, *Actes du Séminaire de Programmation Logique de Trégastel*, pages 551–570. CNET, 1990.

[427] J.A. Robinson. A Machine-oriented Logic Based on the Resolution Principle. *J.ACM*, 12(1), January 1965.

[428] A. Rosenthal. A computer scientist looks at reliability computations. In R.E. Barlow, J.B. Fussel, and N.D. Signpurwalla, editors, *Reliability and fault tree analysis*, pages 133–152. SIAM, 1975.

[429] A. Rosenthal. Decomposition methods for fault tree analysis. *IEEE Transactions on Reliability*, R-29 :136–138, 1980.

[430] P. Rossa. Formules bien imbriquées : reconnaissance et satisfaisabilité. Mémoire de dea, Université de Caen, Laboratoire d'Informatique, 1993.

[431] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'93*, pages 42–47, November 1993.

[432] K. Sakai and A. Aiba. CAL : a Theorical Background of Constraint Logic Programming and its Applications. In *Proceedings of the Workshop Languages and Constraints Providence*, April 1988.

[433] K. Sakai and A. Aiba. CAL : a Theorical Background of Constraint Logic Programming and its Applications. *Journal of Symbolic Computation*, 8 :589–603, 1989.

[434] K. Sakai and Y. Sato. Application of Ideal theory to Boolean Constraint Solving. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence 90*, 1990.

[435] V. Saraswat, M. Rinard, and P. Panangen. Semantic Foundation of Concurrent Constraint Programming. In *Proceedings of the* 18^{th} *Symposium on Principles of Programming Languages, POPL'91*, pages 333–352, 1991.

[436] V.A. Saraswat. *Concurrent Constraint Programming*. ACM Doctoral Dissertation Awards, Logic Programming. MIT Press, Cambridge, Massachusetts, U.S.A., 1993.

[437] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the* 10^{th} *Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.

[438] I. Schiermeyer. Solving 3-satisfiability in less than $1,579^n$ steps. In *Proceedings of CSL'92*, volume 702, pages 379–394. LCNS, 1992.

[439] J.S. Schlipf, F.S. Annexstein, J.V. Franco, and R.P. Swaminathan. On finding solutions for extended Horn formulas. *Information Processing Letters*, 54 :133–137, 1995.

[440] W.G. Schneeweiss. Disjoint Boolean Products via Shannon's Expansion. *IEEE Transaction on Reliability*, 33(4) :329–332, 1984.

[441] W.G. Schneeweiss. *Boolean Functions with Engineering Applications and Computer Programs*. Springer Verlag, 1989. ISBN 3-540-18892-4.

[442] W.G. Schneeweiss. Approximate Fault-Tree Analysis Without Cut Sets. In *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'92*, pages 370–375. IEEE, 1992.

[443] R. Schrag and J.M. Crawford. Implicates and Prime Implicates in Random 3SAT. *Artificial Intelligence, special issue on phase transitions in problem spaces*, 1996.

[444] M.G. Scutellà. A Note on Dowling and Gallier's Top-Down Algorithm for Propositional Horn Satisfiability. *Journal of Logic Programming*, 8 :265–273, 1990.

[445] B. Selman. Stochastic Search and Phase Transitions : AI Meets Phyiscs, 1995. available by ftp research.att.com dist/ai.

[446] B. Selman and H.A. Kautz. Knowledge Compilation Using Horn Approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, 1991.

[447] B. Selman and H.A. Kautz. Domain Independent Extensions to GSAT : Solving Large Structured Satisfiability Problems. In *Proceedings of the International Conference on Artificial Intelligence (IJCAI'93)*, 1993.

[448] B. Selman and H.A. Kautz. An Empirical Study of Greedy Local Search for Satisfiability Testing. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (Washington, D.C., AAAI'1993)*, pages 46–51, 1993.

[449] B. Selman, H.A. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In *Proceedings of the* 12^{th} *National Conference on Artificial Intelligence, AAAI'94*, 1994.

[450] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence, AAAI'92*, 1992.

[451] C.E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions AIEE*, 57 :713–719, 1938.

[452] E. Shapiro. *Concurrent Prolog, Volume 1 and 2.* MIT Press, December 1987.

[453] D.R. Shier. *Network Reliability and Algebraic Structures.* Oxford Science Publications, 1991.

[454] G. Sidebottom. Implementing CLP($\mathcal{B}$) using the Connection Theorem Proving Method and a Clause Management System. *Journal of Symbolic Computation*, 15 :27–48, 1993.

[455] P. Siegel. Représentation et utilisation des connaissances en calcul propositionnel, 1987. Thèse d'état, Groupe d'Intelligence Artificielle, Université Aix-Marseille II.

[456] D. Sieling and I. Wegener. Graph driven BDDs – a new data structures for Boolean functions. *Theoretical Computer Science*, 141 :283–310, 1995.

[457] H. Simonis. Test Generation Using the Constraint Logic Programming Language CHIP. In *Proceedings of the* 6^{th} *Conference on Logic Programming.* MIT Press, 1989. Lisbon, Portugal.

[458] H. Simonis and M. Dincbas. Using an Extended Prolog for Digital Circuit Design. In *Proceedings IEEE Workshop on AI Applications to CAD Systems for Electronics*, pages 165–188, 1987. Munich, W. Germany.

[459] H. Simonis and M. Dincbas. Propositional Calculus in CHIP. In H. Kirchner, editor, *Proceedings of the* 2^{nd} *International Conference on Algebraic and Logic Programming*, 1990.

[460] H. Simonis and M. Dincbas. Propositional Calculus Problems in CHIP. Technical report TR-LP-48, ECRC, 1990.

[461] H. Simonis and M. Dincbas. Propositional Calculus Problems in CHIP. In A. Colmerauer and F. Benhamou, editors, *Constraint Logic Programming : Selected Research*, pages 269–285. MIT Press, 1993. ISBN 0-262-02353-9.

[462] H. Simonis, N. Nguyen, and M. Dincbas. Verification of Digital Circuits using CHIP. In G. Milne, editor, *The Fusion of Hardware Design and Verification.* North Holland, 1988.

[463] R. Smullyan. *First Order Logic*. Springer Verlag – New York, 1968.

[464] J. Spencer. A Survey/Expository Paper : Zero-one Laws with Variable Probability. *Journal of Symbolic Logic*, 58(1) :1–14, 1983.

[465] A. Srinivasan, T. Kam, S. Malik, and R.K. Brayton. Algorithms for Discrete Function Manipulation. In *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'90*, pages 92–95. IEEE, 1990.

[466] R. Stallman and G. Sussman. Forward reasoning and dependency directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9 :135–196, 1977.

[467] D. Stauffer and A. Aharony. *Introduction to percolation theory*. Taylor & Francis, 1992. ISBN 0 7484 0027 3.

[468] B.U. Steffen. Characteristic Formulae. In *Proceedings ICALP'89*, volume 372. LNCS, 1989.

[469] M. Stone. The Theory of Representation for Boolean Algebra. *Transactions AMS*, 40 :37–111, 1936.

[470] R.P. Swaminathan and D.K. Wagner. The arborescence-realization problem. *Discrete Applied Mathematics*, 59, 3 1995.

[471] H. Tamaki and T. Sato. OLD resolution with Tabulation. In *Proceedings of the* 3^{th} *International Conference on Logic Programming, ICLP'86*, volume 225. LNCS, 1986.

[472] R.E. Tarjan. Depth First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1 :146–160, 1972.

[473] R.E. Tarjan. Amortized Computational Complexity. *SIAM Journal of Algebraic Discrete Methods*, 6 :306–318, 1985.

[474] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific. J. Math.*, 5 :285–309, 1955.

[475] E. Tidén. Symbolic Verification of Switch-Level Circuits using a Prolog Enhanced with Unification in Finite Algebra. In G. Milne, editor, *The Fusion of Hardware Design and Verification*. North Holland, 1988.

[476] H. Touati, R.K. Brayton, and R. Kurshan. Testing Language Containment for ω-Automata using BDD's. In *Proceedings IMEC-IFIP International Workshop on Formal Methods in VLSI Design*, 1991.

[477] H.J. Touati, H. Savoj, B. Lin, and R.K. Brayton. Implicit State Enumeration of Finite State Machines using BDD's. In *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'90*, pages 130–133, November 1990.

[478] C.A. Tovey. A Simplified NP-complete Satisfiability Problem. *Discrete Applied Mathematics*, 8 :85–89, 1984.

[479] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993. ISBN 0-12-701610-4.

[480] G.S. Tseitin. On the complexity of derivations in the propositional calculus. In A.O. Slisenko, editor, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, New York, 1970.

[481] J. Ullman. *Principles of database systems. Second edition*. Computer software engineering series. Computer Science Press, 1982.

[482] T.E. Uribe and M.A. Stickel. Ordered Binary Decision Diagrams and the Davis-Putnam Procedure. In J.P. Jouannaud, editor, *Proceedings of the first international conference on Constraints in Computational Logic (CCL'94)*, 1994.

[483] A. Urquhart. Hard Examples for Resolution. *JACM*, 34(1) :209–219, January 1987.

[484] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8 :410–421, 1979.

[485] L.G. Valiant. On the complexity of computing the permanent. *Theoretical Computer Science*, 8 :189–201, 1979.

[486] P. van Beek. On the Minimality and Decomposability of Constraint Networks. In *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence, AAAI'92*, pages 447–452, 1992. San José, California.

[487] A. van Gelder. A Satisfiability Tester for Non-clausal Propositional Calculus. *Information and Computation*, 79 :1–21, 1988.

[488] A. van Gelder. Linear Time Unit Resolution for Propositional Formulas - in Prolog, Yet. submitted to the Journal of Logic Programming, 1994.

[489] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, 1989. ISBN 0-262-08181-4.

[490] P. van Hentenryck. Special Issue on Constraint Logic Programming. *Journal of Logic Programming*, 16(3 & 4), 1993.

[491] P. van Hentenryck and Y. Deville. The Cardinality Operator : A New Logical Connective and its Application to Constraint Logic Programming. In *Proceedings of the* 8^{th} *International Conference on Logic Programming, ICLP'91*, pages 745–759. MIT Press, 1991.

[492] P. van Hentenryck, V. Saraswat, and Y. Deville. Constraint Processing in cc($\mathcal{FD}$). Research Report TR-CS-93-02, Brown University, 1993.

[493] P. van Hentenryck, V. Saraswat, and Y. Deville. Design, Implementation, and Evalutation of the Constraint Language `cc(FD)`. In A. Poldeski, editor, *Constraint Programming : Basic and Trends*, volume 910, pages 293–316. LNCS, 1995.

[494] P. van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using Constraint Logic Programming. *Artificial Intelligence*, 58 :113–159, 1992.

[495] P. van Roy and A.M. Despain. The Benefits of Global Dataflow Analysis for an Optimizing Prolog Compiler. In *Proceedings of the 1990 North American Conference on Logic Programming*, pages 501–515. MIT Press, 1990.

[496] A. Vellino. *The Complexity of Automated Reasonning.* PhD thesis, University of Toronto, Department of Philosophy, 1989.

[497] A. Vellino. A Prolog Implementation of an Analytic Tableau Theorem Prover for the Propositional Calculus. Technical Report CRL-89024, Bell Nothern Research, Computing Research Laboratory, 1989.

[498] N.R. Vempaty. Solving Constraint Satisfaction Problem Using Finite State Automata. In *Proceedings of American Conference on Artificial Intelligence, AAAI'92*, pages 453–458, 1992.

[499] W.E. Vesely, F.F. Goldberg, N.H. Robert, and D.F. Haasl. Fault Tree Handbook. Technical Report NUREG 0492, U.S. Nuclear Regulatory Commission, 1981.

[500] A. Villemeur. *Sûreté de fonctionnement des systèmes industriels*. Eyrolles, 1988.

[501] F. Vlach. Simplification in a Satisfiability Checker for VLSI Applications. *Journal of Automated Reasoning*, 10 :115–136, 1993.

[502] D.L. Waltz. Generating semantic descriptions for drawings of scenes with shadows. Technical Report AI271, M.I.T., Cambridge MA, 1972.

[503] D.L. Waltz. Generating semantic descriptions for drawings of scenes with shadows. In P.H. Winston, editor, *The Psychology of Computer Vision*, pages 19–91. Mc Graw Hill, New York, 1975.

[504] D.H.D. Warren. An abstract Prolog instruction set. Technical Note 309, SRI International, Menlo Park CA, USA, October 1983.

[505] I. Wegener. *The Complexity of Boolean Functions*. Wiley, New-York, 1987.

[506] I. Wegener. Efficient data structures for Boolean functions. *Discrete Mathematics*, 136 :347–372, 1994.

[507] G.A. Wilson and C. Minker. Resolution, refinements and search strategies :A comparative study. *IEEE Trans. Comput.*, C-25 :782–801, 1976.

[508] J.M. Wilson. Modularizing and minimizing fault trees. *IEEE Transactions on Reliability*, R-34 :320–322, 1985.

[509] J.M. Wilson. An Improved Minimizing Algorithm for Sum of Disjoint Products. *IEEE Transactions on Reliability*, 39 :42–45, 1990.

[510] S. Yamasaki and S. Doshita. The satisfiability problem for a class consisting of Horn sentences and some non-Horn sentences in propositional logic. *Information and Computation*, 59 :1–12, 1983.

[511] J. Yllera. Modularization methods for evaluating fault trees of complex technical systems. In A. Kandel and E. Avni, editors, *Engineering Risk and Hazard Assessment*, volume 2, chapter 5. CRC Press, 1988. ISBN 0-8493-4655-X.

[512] R. Zabih and D. Mac Allester. A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of the National Conference on Artificial Intelligence, AAAI'88*, pages 155–160, 1988.

[513] D. Zampunieris and B. Le Charlier. An Efficient Algorithm to Compute the Synchronized Product. In *Proceedings of Workshop on Modeling, Analysis and Simulation of Computer and Telecomunication Systems, MASCOTS'95*, January 1995. Also Research Paper RP-94-003, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium.

[514] D. Zampunieris and B. Le Charlier. Efficient Handling of Large Sets of Tuples with Sharing Trees. In *Proceedings of Data Compression Conference, DCC'95*, October 1995. This paper appeared as poster in the cited conference, and also as Research Paper RP-94-004, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium.

Annexe A

Curriculum vitae

Annexe B

Articles autour du problème SAT

B.1 Towards a Better Understanding of SL-Resolution

– L. Oxusoff and A. Rauzy. Towards a Better Understanding of SL-Resolution. In P. Jorrand and V. Sugrev, editors, *Proceedings Artificial Intelligence V Methodology, Systems, Applications, AIMSA'94*. North Holland, 1994.

B.2 Polynomial restrictions of SAT : What can be done with an efficient implementation of the Davis and Putnam's procedure.

– A. Rauzy, Polynomial restrictions of SAT : What can be done with an efficient implementation of the Davis and Putnam's procedure. In U. Montanari and F. Rossi, editors, *Proceedings of the International Conference on Principle of Constraint Programming, CP'95*, volume 976 of *LNCS*, pages 515–532. Springer Verlag, 1995.

Annexe C

Articles sur Toupie

C.1 Efficient Abstract Interpretation of Prolog Programs by means of Constraint Solving over Finite Domains

– M-M. Corsini, B. Le Charlier, K. Musumbu, and A. Rauzy. Efficient Abstract Interpretation of Prolog Programs by means of Constraint Solving over Finite Domains (extended abstract). In *Proceedings of the 5th Int. Symposium on Programming Language Implementation and Logic Programming, PLILP'93*, volume 714, Tallin, Estonia, 1993. LNCS.

C.2 Toupie : the μ-calculus over finite domains as a constraint language

– M-M. Corsini et A. Rauzy, Toupie : the μ-calculus over finite domains as a constraint language. À paraître dans Journal of Automated Reasonning.

Annexe D

Articles sur les applications du calcul propositionnel

D.1 Notes on the Design of an Open Boolean Solver

– A. Rauzy. Notes on the Design of an Open Boolean Solver. In P. van Hentenryck, editor, *Proceedings of the* 11^{th} *International Conference on Logic Programming, ICLP'94*, pages 354–368. MIT Press, 1994.

D.2 New Algorithms for Fault Trees Analysis

– A. Rauzy, New Algorithms for Fault Trees Analysis. *Reliability Engineering & System Safety*, 05 59, pp 203–211, 1993.