Université de Provence - Aix-Marseille I
École Doctorale en Mathématiques et Informatique de Marseille, ED 184

Symétries locales et globales en logique propositionnelle et leurs extensions aux logiques non monotones

THÈSE

pour l'obtention du

Doctorat de l'Université de Provence - Aix-Marseille I
(Spécialité : Informatique)

Présentée et soutenue par
Tarek Nabhani

soutenue le 09 décembre 2011

Jury :

Rapporteurs :	Chu Min Li	-	Université de Picardie
	Lakhdar Saïs	-	Université d'Artois
Examinateurs :	Pierre Siegel	-	Université de Provence
	Souhila Kaci	-	Université de Montpellier II
Directeur :	Belaïd Benhamou	-	Université de Provence
Co-directeur :	Richard Ostrowski	-	Université de Provence

Laboratoire des Sciences de l'Information et des Systèmes (**LSIS**) – UMR 6168
http://www.lsis.org

Remerciements

Cette thèse n'aurait vu le jour sans la confiance, la patience et la générosité de mon directeur de recherche M. Belaïd Benhamou que je veux vivement remercier. Je voudrais le remercie aussi pour les conseils qui m'ont orienté au long de la production de cette thèse. En fait, je ne saurais exprimer toute ma gratitude en quelques lignes.

Je veux remercie aussi Richard Ostrowski pour m'avoir fait connaître le problème SAT et pour tout le temps et les efforts qu'il m'a consacrés pendant ces trois annnées.

Je suis très reconnaissant à Monsieur Chu Min Li, professeur à l'université de Picardie, et à Monsieur Lakhdar Saïs, professeur à l'université d'Artois, d'avoir accepté d'être rapporteurs de ma thèse.

Je remercie chaleureusement tous les autres membres du jury : Monsieur Pierre Siegel, professeur à l'université de Provence, et Madame Souhila Kaci, professeur à l'université de Montpellier II, pour avoir accepté de juger ce travail.

Je voudrais aussi remercier mes chers amis : Mohamed Réda Saïdi et Farid Nouioua pour leurs nombreuses relectures et leurs commentaires qui m'ont permis de clarifier ma rédaction.

Je tiens également à remercier toute l'équipe INCA.

Un grand merci aux membres du CMI pour m'avoir permis de réaliser cette thèse dans de bonnes conditions, ainsi qu'aux membres de l'École Doctorale « Mathématiques et Informatique de Marseille » pour leur excellent travail jusqu'à la dernière minute.

Un remerciement particulier pour les doctorants Laurent Battisti pour son aide précieuse et Saurabh Trivedi pour les agréables moments passés ensemble durant ces trois années de thèse.

Enfin, j'adresse mes plus sincères remerciements à tous mes proches, mes parents, mes sœurs et amis, sans oublier mon épouse et ma chère fille qui m'ont toujours soutenu et encouragé au cours de la réalisation de cette thèse.

Table des matières

Introduction Générale v

I Introduction générale 1

1 Logique propositionnelle et le problème SAT 3
1.1 Introduction . . . 3
1.2 Logique propositionnelle . . . 4
1.2.1 Syntaxe . . . 4
1.2.2 Sémantique . . . 7
1.3 Problème SAT . . . 9
1.3.1 Définition du problème . . . 9
1.3.2 Classes polynomiales de SAT . . . 10
1.4 Algorithmes de résolution de SAT . . . 12
1.4.1 Principe de résolution . . . 12
1.4.2 Procédure de Davis & Putnam . . . 14
1.4.3 Apprentissage dirigé par les conflits (CDCL) . . . 17

2 Logiques non monotones 25
2.1 Introduction . . . 25
2.1.1 Raisonnement révisable . . . 25
2.1.2 Logique classique et raisonnement valide . . . 26
2.1.3 Logique des prédicats (Premier ordre) . . . 27
2.2 Logiques non monotones . . . 30
2.2.1 Logique préférentielle . . . 31
2.2.2 $X-$logique . . . 33
2.2.3 Logique des défauts . . . 35

3 Structure de groupes et groupes symétriques 43
3.1 Introduction . . . 43
3.2 Notion de groupe . . . 43

3.2.1 Loi de composition interne . . . 43
3.2.2 Groupe . . . 44
3.2.3 Sous-groupe . . . 46
3.3 Groupes symétriques S_n . . . 48
3.3.1 Notion de $\sigma-$orbite . . . 48
3.3.2 Cycles dans S_n . . . 49

II Contributions 51

4 Élimination dynamique des symétries dans SAT 53
4.1 Introduction . . . 53
4.2 Symétrie dans la logique propositionnelle . . . 54
4.3 La détection et l'élimination de la symétrie locale . . . 60
4.3.1 Détection dynamique de la symétrie . . . 61
4.3.2 Élimination des symétries : . . . 62
4.4 Exploitation de la symétrie . . . 62
4.5 Expérimentations . . . 62
4.5.1 Résultats sur des instances SAT diverses . . . 64
4.5.2 Résultats sur des instances du problème de coloration de graphes . . . 64
4.6 Conclusion . . . 67

5 Apprentissage par symétrie dans les solveurs SAT 69
5.1 Introduction . . . 69
5.2 Symétries et apprentissage de clauses . . . 70
5.3 Avantages de la symétrie dans les solveurs CDCL . . . 73
5.4 Expérimentation . . . 73
5.4.1 Résultats sur différents problèmes symétriques . . . 76
5.4.2 Résultats sur les problèmes de coloration de graphe . . . 76
5.4.3 Résultats sur des problèmes des dernières compétitions SAT . . . 78
5.5 Conclusion . . . 82

6 Symétries dans les logiques non monotones 83
6.1 Introduction . . . 83
6.2 Symétrie dans la logique préférentielle . . . 84
6.3 Symétries dans les $X-$Logiques . . . 86
6.4 Symétrie dans la logique des défauts . . . 88
6.5 Conclusion . . . 93

Conclusion et perspectives 95

Bibliographie 99

Introduction Générale

La symétrie est par définition un concept multidisciplinaire. Il apparaît dans de nombreux domaines allant des mathématiques à l'intelligence artificielle, la chimie et la physique. En général, elle revient à une transformation qui laisse invariant (ne modifie pas sa structure fondamentale et/ou ses propriétés) un objet (un chiffre, une molécule, un système physique, une formule ou un réseau de contraintes, etc. . .). Par exemple, la rotation d'un échiquier 180 degrés donne un état qui ne se distingue pas de celui d'origine. La symétrie est une propriété fondamentale qui peut être utilisée pour étudier ces différents objets, analyser finement ces systèmes complexes ou pour réduire la complexité de calcul lorsqu'il s'agit de problèmes combinatoires.

Le *problème de satisfaisabilité d'une formule de la logique propositionnelle mise sous forme normale conjonctive*, appelé plus communément le problème SAT occupe un rôle central en théorie de la complexité. C'est un problème générique, de nombreux problèmes provenant d'autres domaines peuvent être réduits au problème de satisfiabilité. Par exemple, la déduction automatique, la configuration, la planification, l'ordonnancement, etc. . . . Le problème SAT est un problème de décision qui consiste à déterminer si une formule booléenne mise sous forme normale conjonctive CNF admet ou non une valuation qui la rend vraie. Le problème SAT est le premier problème à avoir été montré complet pour la classe NP (Non déterministe Polynomial) par Cook en 1971 [Cook, 1971].

D'autre part, Krishnamurthy dans [Krishnamurty, 1985] a introduit le principe des symétries dans le calcul propositionnel. Une symétrie d'une formule logique est une permutation de littéraux qui laisse invariante cette formule. Krishnamurthy a montré que certaines formules difficiles à prouver peuvent avoir des preuves courtes si on ajoute au système de résolution la règle de symétrie. La définition des symétries à été élargie aux littéraux dans [Benhamou, 1993, Benhamou and Sais, 1994, Benhamou and Sais, 1992], un premier algorithme de détection des symétries à été élaboré et l'exploitation des symétries à été réalisé dans les algorithmes de déduction automatique telle que DP et la SL-résolution.

Les symétries ont été utilisées bien avant pour la résolution de nombreux problèmes comme celui des huit reines [Glaisher, 1874]. Elles ont aussi été introduites dans la résolution des problèmes de satisfaction de contraintes [Freuder, 1991, Puget, 1993, Benhamou, 1994], dans un algorithme de Backtracking intelligent [Brown et al., 1988] et dans la logique du premier ordre [Crawford, 1992].

L'importance de l'exploitation des symétries lors de la résolution peut être mise en évidence sur de nombreux problèmes que les méthodes de résolution classiques ont du mal à résoudre efficacement. Prenons par exemple le problème des pigeons [Bibel, 1990, Haken, 1985], ou le problème de Ramsey [Kalbfleisch and Stanton, 1969]. Les deux problèmes sont connus pour être difficiles à résoudre pour les méthodes de résolution classiques et ils sont représentés en logique du premier ordre par un petit ensemble de formules qui devient très grand lorsque l'on tente de calculer toutes les instanciations propositionnelles terminales. L'ensemble des clauses propositionnelles obtenues contient un nombre important de symétries. C'est-à-dire que l'ensemble des clauses reste invariant par rapport à plusieurs permutations de variables. Il en résulte que prouver la satisfiabilité par l'exploitation de ces permutations a une complexité polynomiale alors que l'on sait que les méthodes de résolution classiques sont toutes de complexité exponentielle pour résoudre ces deux problèmes, si l'élimination de la symétrie n'est pas prise en compte.

Durant la dernière décennie, plusieurs travaux sur l'exploitation des symétries dans la résolution des problèmes SAT et CSP sont apparus. Cependant, peu de travaux exploitent la détection et l'élimination dynamique des symétries [Benhamou and Sais, 1992, Benhamou and Sais, 1994, Benhamou et al., 1994, Benhamou and Saïdi, 2007, Gent et al., 2007]. La plupart des méthodes n'exploitent que les symétries globales [Crawford et al., 1996, Aloul et al., 2003, Aloul et al., 2004], c'est-à-dire, les symétries du problème initial correspondant à la racine de l'arbre de recherche.

La première contribution de ce mémoire est l'étude d'une nouvelle méthode complète qui élimine toutes les symétries locales pour la résolution du problème SAT en exploitant le groupe des symétries. Les résultats obtenus sont encourageants et montrent que l'exploitation des symétries locales donne de meilleurs résultats que l'exploitation des symétries globales sur certaines instances SAT et que la combinaison de l'exploitation des deux types de symétries sont complémentaires.

D'autre part, de nombreuses améliorations ont été apportées dans les solveurs actuels du problème SAT. Les solveurs de type CDCL (Conflict Driven Clauses Learning) sont aujourd'hui capables de résoudre de manière efficace des problèmes industriels de très grande taille (en nombre de variables et de clauses). Ces derniers utilisent des structures de données

plus sophistiquées permettant d'optimiser le coût des propagation de mono-littéraux, appliquent l'apprentissage de clauses [Silva and Sakallah, 1996, Zhang et al., 2001], effectuent le retour arrière non chronologique [Silva and Sakallah, 1996, Jr. and Schrag, 1997] et différentes politiques de redémarrage [Gomes et al., 1997, Huang, 2007, Audemard et al., 2008]. Il a été montré que l'apprentissage de clauses est une notion importante [Beame et al., 2004, Hertel et al., 2008, Pipatsrisawat and Darwiche, 2008, Pipatsrisawat and Darwiche, 2009] qui améliore sensiblement l'efficacité des solveurs. Bien que l'utilisation des symétries et l'apprentissage de clauses s'avèrent être des principes puissants, mais la combinaison des deux notions n'a pas été investi par les chercheurs du domaine.

Dans la deuxième contribution de cette thèse, nous proposons une approche d'apprentissage de clauses en utilisant la symétrie. Cette approche est dynamique, elle génère au cours de la recherche toutes les clauses symétriques des différentes clauses assertives afin d'éviter l'exploration de sous-espaces isomorphes. L'apprentissage par symétrie est différent des approches consistant à éliminer les symétries globales du problème. Ces dernières sont des méthodes statiques qui ajoutent en prétraitement des clauses afin d'éliminer les interprétations symétriques du problème. L'avantage de notre nouvelle méthode est qu'elle n'élimine pas les modèles symétriques comme font les méthodes statiques. Elle évite d'explorer des sous-espaces correspondant aux no-goods symétriques de l'interprétation partielle courante. Les résultats obtenus sont très encourageants et montrent que l'utilisation des symétries dans l'apprentissage est profitable pour des solveurs à base de CDCL.

En Intelligence Artificielle, nous avons l'habitude de manipuler des informations incomplètes (données incertaines, des informations révisables, etc...), qui nécessite un type de raisonnement sur les connaissances qui gère convenablement les exceptions : c'est le raisonnement non monotone.

Une composante essentielle de l'intelligence (qu'elle soit humaine, animale ou artificielle) est en effet liée à une certaine capacité d'adaptation du raisonnement. Contrairement au mode de raisonnement formalisé par une logique conventionnelle ou classique, le résultat déduit d'une information (à partir d'une connaissance, ou de croyances) n'est pas vrai indéfiniment, mais seulement plausible dans le sens où il peut être invalidé ou révisé lors de l'ajout d'une nouvelle information.

Par exemple, il est admis qu'un oiseau normale vole. Ainsi, si l'on sait que Titi est un oiseau, alors on en conclut que, naturellement, Titi vole. Si l'on apprend par la suite que Titi est une autruche, cette conclusion devra être révisée. Ceci est impossible dans une logique classique ayant la propriété de monotonie : une information déduite d'une base de connaissances C sera toujours vraie si C est augmentée par d'autres connaissances inférées.

Pour gérer le problème des exceptions, plusieurs approches logiques ont

été introduites en intelligence artificielle. De nombreux formalismes non monotones ont été présentés depuis une trentaine d'années, mais le problème de la symétrie dans ce cadre n'a pas été étudié. Le raisonnement en utilisant la symétrie est cependant pertinent pour la représentation des connaissances et le raisonnement non monotone. Par exemple, dans l'exemple précédent, il est intéressant de considérer que les oiseaux normaux appartiennent à la même classe en respectant certaines propriétés de base, puis ils sont tous symétriques en ce sens.

La troisième contribution de cette thèse consiste en l'extension de la notion de symétrie à des logiques non classiques telles que les logiques préférentielles [Bossu and Siegel, 1982, Bossu and Siegel, 1985, SHOAM, 1987, Besnard and Siegel, 1988, Kraus et al., 1990], les $X-$logiques [Siegel et al., 2001] et les logiques des défauts [Reiter, 1980]. Nous montrons comment raisonner par symétrie dans ces logiques et nous mettons en évidence l'existence de certaines symétries dans des logiques non classiques qui n'existent pas dans les logiques classiques.

Ce mémoire est structuré en deux parties. La première partie contient des rappelles et brefs états de l'art sur la logique propositionnelle, les logiques non monotones et la structure de groupe et groupes symétriques, qui sont nécessaires pour la compréhension des contributions de la thèse. Cette partie contient trois chapitres :

Dans le premier chapitre nous décrivons brièvement la logique propositionnelle en donnant sa syntaxe et sa sémantique et nous définissons le problème SAT. Nous proposons ensuite un aperçu sur les techniques de recherche principales pour résoudre le problème SAT : le principe de résolution, la procédure de Davis & Putnam (DP et DPLL) et l'apprentissage dirigé par les conflits (CDCL).

Dans le deuxième chapitre nous présentons la notion de raisonnement révisable. Nous présentons ensuite les logiques non monotones en décrivant les logiques préférentielles, les $X-$logiques et les logiques des défauts.

Dans le troisième chapitre nous faisons des rappels sur la notion de groupe et sous-groupe afin de présenter la notion de groupe symétriques.

La deuxième partie contient les trois contributions de la thèse. Cette partie est structuré en trois chapitres :

Dans le quatrième chapitre nous présentons une méthode qui élimine toutes les symétries locales dans la résolution du problème SAT, en exploitant le groupe total des symétries [Benhamou et al., 2009, Benhamou et al., 2010c, Benhamou et al., 2010b, Benhamou et al., 2010e]. Cette méthode consiste à réduire incrémentalement la sous-formule logique définie à chaque nœud de l'arbre de recherche à un graphe sur lequel nous utilisons un outil de calcul d'automorphismes de graphes tel que Saucy.

Dans le cinquième chapitre nous présentons une nouvelle approche pour l'apprentissage de clause dans les solveurs CDCL qui utilise les symétries du

problème [Benhamou et al., 2010a, Benhamou et al., 2010d]. Cette méthode consiste à détecter toutes les symétries globales du problème et de les utiliser lorsqu'une nouvelle clause (clause assertive) est déduite au cours de la recherche. Cela nous permet de déduire et considérer toutes les clauses assertives symétriques à la clause de référence du problème.

Dans le sixième et dernier chapitre nous nous sommes intéressés à étendre la notion de symétrie à des logiques non classiques telles que les logiques préférentielles, les $X-$logiques et les logiques des défauts [Benhamou et al., 2010g, Benhamou et al., 2010h, Benhamou et al., 2010f].

En fin pour terminer le mémoire, une conclusion général et plusieurs perspectives ouvrant des voies de recherches future sont présentées.

Première partie

Introduction générale

Chapitre 1

Logique propositionnelle et le problème SAT

Sommaire

1.1 Introduction 3
1.2 Logique propositionnelle 4
1.2.1 Syntaxe 4
1.2.2 Sémantique 7
1.3 Problème SAT 9
1.3.1 Définition du problème 9
1.3.2 Classes polynomiales de SAT 10
1.4 Algorithmes de résolution de SAT 12
1.4.1 Principe de résolution 12
1.4.2 Procédure de Davis & Putnam 14
1.4.3 Apprentissage dirigé par les conflits (CDCL) 17

1.1 Introduction

Traiter un problème en Intelligence Artificielle consiste, dans un premier temps, à mettre les connaissances dans un langage « lisible » pour la machine qu'on appelle *modélisation* ou *représentation*, puis appliquer un mode de raisonnement pour déduire des résultats de cette représentation.

Par contre, on peut toujours poser la question suivante : pourquoi faut-il utiliser un langage spécifique pour représenter les problèmes logiques et non pas la langue française (ou toute autre langue naturelle), par exemple ? La première raison est que le français est d'une telle richesse qu'on ne peut pas décrire formellement les problèmes logiques. La deuxième raison est que la signification d'une phrase en français peut être ambiguë.

En fait, pour qu'on puisse profiter du développement des technologies informatiques modernes et de ses outils, on utilise une représentation binaire des informations pour s'approcher du langage machine. Le langage de la logique propositionnelle permet de réaliser cet objectif.

En logique propositionnelle, une connaissance peut s'exprimer de différentes manières : la forme normale conjonctive [1] (CNF), la forme normale disjonctive [2] (DNF), ... etc.... Pour notre étude, nous allons utiliser la forme normale conjonctive pour exprimer les connaissances.

Dans ce chapitre, nous décrivons brièvement la logique propositionnelle, nous donnons sa syntaxe et sa sémantique et nous définissons le problème SAT. Nous proposons ensuite un aperçu sur les techniques de recherche principales pour résoudre le problème SAT : le principe de résolution, la procédure de Davis & Putnam (DP et DPLL) et l'apprentissage dirigé par les conflits (CDCL).

1.2 Logique propositionnelle

1.2.1 Syntaxe

Définition 1.2.1 (variable propositionnelle).
Une variable propositionnelle*, parfois appelée* proposition atomique *ou* atome*, est une variable booléenne, c'est-à-dire une variable qui prend ses valeurs de vérité dans l'ensemble* $\{faux, vrai\}$ *ou encore* $\{0, 1\}$ *ou encore* $\{\bot, \top\}$.

Une variable propositionnelle représente une proposition, et la valeur booléenne 0 (respectivement 1) décrit que la proposition est *fausse* (respectivement *vraie*).

Définition 1.2.2 (Langage de la logique propositionnelle).
Le langage de la logique propositionnelle, noté LP, est défini par :

1. *les valeurs booléennes* 0 *et* 1 *;*
2. *les symboles* $\bot$ *et* $\top$ *;*
3. *un ensemble dénombrable* $\mathcal{P}$ *de variables propositionnelles :* $p_0, p_1, \ldots, p_n, \ldots$ *;*
4. *les connecteurs logiques :*
   - *de négation :* $\neg$ *(non) ;*
   - *de conjonction :* $\wedge$ *(et) ;*
   - *de disjonction :* $\vee$ *(ou) ;*
   - *d'implication :* $\Longrightarrow$ *(si ... alors ...) ;*
   - *d'équivalence :* $\Longleftrightarrow$ *(... si et seulement si ...) ;*

1. En anglais : Conjunctive Normal Form.
2. En anglais : Disjunctive Normal Form.

5. *les symboles auxiliaires : la parenthèse gauche « (» et la parenthèse droite «) ».*

Les éléments de ce langage permettent de définir inductivement des expressions, appelées *formules propositionnelles.*

Définition 1.2.3 (Formule propositionnelle).
Une formule propositionnelle $\mathcal{F}$ *d'un langage propositionnel LP se construit récursivement en appliquant un nombre* fini *de fois les règles suivantes :*

1. *un atome,* 0, 1, $\bot$ *et* $\top$ *sont des formules ;*
2. *si* $\mathcal{F}$ *est une formule, alors* $(\mathcal{F})$ *est une formule ;*
3. *si* $\mathcal{F}$ *est une formule, alors* $\neg\mathcal{F}$ *est une formule ;*
4. *si* $\mathcal{F}$ *et* $\mathcal{F}'$ *sont des formules, alors :*
   - $\mathcal{F} \wedge \mathcal{F}'$ *est une formule ;*
   - $\mathcal{F} \vee \mathcal{F}'$ *est une formule ;*
   - $\mathcal{F} \implies \mathcal{F}'$ *est une formule ;*
   - $\mathcal{F} \iff \mathcal{F}'$ *est une formule.*

On note par $\mathscr{F}_{LP}$ (ou bien par $\mathscr{F}$, dans le cas où il n'y a pas d'ambiguïté) l'ensemble de toutes les formules propositionnelles d'un langage propositionnel LP.

Définition 1.2.4 (Littéral).
Un littéral est une variable propositionnelle p *ou sa négation* $\neg p$.

La variable propositionnelle p est appelée *littéral positif* et $\neg p$ *littéral négatif.* Les deux littéraux p et $\neg p$ son dits *complémentaires.* Si on note par $\mathcal{L}$ l'ensemble de littéraux de la logique propositionnelle LP, on a alors :

$$\mathcal{L} = \{\ell \mid \ell \in \mathcal{P} \quad \text{ou} \quad \neg\ell \in \mathcal{P}\}$$

Définition 1.2.5 (Littéral pur).
Un littéral ℓ *est dit* pur *pour la formule* $\mathcal{F}$ *si et seulement si* ℓ *apparaît dans* $\mathcal{F}$ *et* $\neg\ell$ *n'apparaît pas dans* $\mathcal{F}$.

Nous pouvons associer à chaque formule $\mathcal{F} \in \mathscr{F}$ un ensemble fini $vars(\mathcal{F})$ contenant les variables propositionnelles qui composent $\mathcal{F}$. On peut aussi associer à chaque formule $\mathcal{F} \in \mathscr{F}$ un ensemble fini de littéraux, noté $lits(\mathcal{F})$, qui contient chaque littéral de $\mathcal{F}$ et son opposé. On appelle $lits(\mathcal{F})$ l'*ensemble complet de littéraux* de $\mathcal{F}$. On a alors $|lits(\mathcal{F})| = 2\,|vars(\mathcal{F})|$.

1.2.1.1 Formes normales

Parmi les formules propositionnelles de LP, nous pouvons distinguer certaines formules particulières :

Définition 1.2.6 (Clause).
On appelle clause *une formule constituée d'une disjonction finie de littéraux.*

Définition 1.2.7 (Monôme).
On appelle monôme *une formule constituée d'une conjonction finie de littéraux.*

Définition 1.2.8 (Longueur d'une clause ou d'un monôme).
On appelle longueur de la clause C *(respectivement* longueur du monôme M*), le nombre de littéraux différents dans* C *(respectivement dans* M*). Cette longueur sera notée* $|C|$ *(respectivement* $|M|$*).*

Définition 1.2.9 (Différentes variantes de clauses).
- Clause vide *: Une clause* C *est dite* vide *(notée* $\perp$ *ou* $\square$*) si et seulement si elle ne contient aucun littéral, c'est-à-dire* $|C| = 0$.
- Clause unitaire *: Une clause* C *est dite* unitaire *(ou* unaire *ou* mono-littéral*) si et seulement si elle contient seulement un seul littéral, c'est-à-dire* $|C| = 1$.
- Clause binaire *: Une clause* C *est dite* binaire *si et seulement si elle contient exactement deux littéraux, c'est-à-dire* $|C| = 2$.
- Clause n-aire *: Une clause* C *est dite* n-aire *si et seulement si elle contient exactement* n *littéraux, c'est-à-dire* $|C| = n$.
- Clause positive, négative, mixte *: Une clause* C *est dite* positive *(respectivement* négative*) si et seulement si elle ne contient pas de littéraux négatifs (respectivement positifs). Une clause constituée de littéraux positifs et négatifs est appelée clause* mixte.
- Clause tautologique *: Une clause* C *est dite* tautologique *si et seulement si elle contient un littéral et son complémentaire.*
- Clause fondamental *: Une clause* C *est dite* fondamental *si et seulement si elle ne contient pas de littéraux complémentaires.*
- Clause de Horn, clause reverse-Horn *: Une clause de Horn (respectivement clause reverse-Horn) est une clause qui contient au plus un littéral positif (respectivement négatif).*
- Clause sous-sommant une autre *: Une clause* C_1 subsume *(ou* sous-somme*) une autre clause* C_2 *si et seulement si tous les littéraux de* C_1 *sont aussi des littéraux de* C_2.

Définition 1.2.10 (Formes normales).
- Forme normale conjonctive « CNF » *: une formule propositionnelle est sous forme normale conjonctive si et seulement si elle est une conjonction de clauses.*
- Forme normale disjonctive « DNF » *: une formule propositionnelle est sous forme normale disjonctive si et seulement si elle est une disjonction de monômes.*

- Forme normale négative « NNF » *: une formule propositionnelle est sous forme normale négative si et seulement si elle est une formule telle que :*
  - *les seuls connecteurs logiques utilisés sont : la négation, la conjonction et la disjonction, c'est-à-dire* $\{\neg, \vee, \wedge\}$ *;*
  - *la négation ne s'applique que sur les variables propositionnelles.*

Dans la suite, nous noterons par *formule CNF* (respectivement *formule* DNF) toute formule propositionnelle en forme normale conjonctive (respectivement disjonctive).

Remarque 1.2.1. On peut représenter les formules CNF (respectivement DNF) sous forme ensembliste. En fait, on considère une formule CNF (respectivement DNF) comme un ensemble de clauses (respectivement monômes), où chaque clause (respectivement monôme) est représenté par l'ensemble de ses littéraux.

1.2.2 Sémantique

Pour définir le problème de la satisfiabilité SAT, il est nécessaire de définir le sens d'une formule propositionnelle.

Définition 1.2.11 (Interprétation des variables).
Une interprétation $\mathcal{I}_P$ *(ou* affectation *ou* instanciation*) est une application d'un sous-ensemble de variables propositionnelles* $P \subseteq \mathcal{P}$ *dans l'ensemble* $\{vrai, faux\}$*, qui associe à toute variable* $p \in P$ *une valeur de l'ensemble* $\{vrai, faux\}$*, formellement :*

$$\mathcal{I}_P \colon P \to \{vrai, faux\} \qquad \mathcal{I}_P \colon p \mapsto \mathcal{I}_P(p) \in \{vrai, faux\}$$

S'il n'y a pas d'ambiguïté, on notera l'interprétation $\mathcal{I}_P$ par $\mathcal{I}$.

L'interprétation $\mathcal{I}$ sur les variables propositionnelles peut se généraliser d'une façon naturelle aux formules propositionnelles.

Définition 1.2.12 (Interprétation de formules).
Soit $\mathcal{F} \in \mathscr{F}_{LP}$ *une formule propositionnelle. Une* interprétation $\mathcal{I}$ *(ou* affectation *ou* instanciation*) de la formule* $\mathcal{F}$ *est une application de* $\mathscr{F}_{LP}$ *dans l'ensemble* $\{vrai, faux\}$*, qui associe à toute formule* $\mathcal{F}$ *une valeur de l'ensemble* $\{vrai, faux\}$*, formellement :*

$$\mathcal{I} \colon \mathscr{F}_{LP} \to \{vrai, faux\} \qquad \mathcal{I} \colon \mathcal{F} \mapsto \mathcal{I}(\mathcal{F}) \in \{vrai, faux\}$$

Supposant que $\mathcal{G}$ *et* $\mathcal{G}'$ *sont deux formules quelconques de* $\mathscr{F}_{LP}$*, l'interprétation* $\mathcal{I}$ *satisfait les propriétés suivantes :*

1. $\mathcal{I}(\top) = vrai$ *;*
2. $\mathcal{I}(\bot) = faux$ *;*

3. $\mathcal{I}(\neg\mathcal{G}) = vrai$ *si et seulement si* $\mathcal{I}(\mathcal{G}) = faux$;
4. $\mathcal{I}(\mathcal{G} \wedge \mathcal{G}') = vrai$ *si et seulement si* $\mathcal{I}(\mathcal{G}) = \mathcal{I}(\mathcal{G}') = vrai$;
5. $\mathcal{I}(\mathcal{G} \vee \mathcal{G}') = faux$ *si et seulement si* $\mathcal{I}(\mathcal{G}) = \mathcal{I}(\mathcal{G}') = faux$;
6. $\mathcal{I}(\mathcal{G} \implies \mathcal{G}') = faux$ *si et seulement si* $\mathcal{I}(\mathcal{G}) = vrai \quad et \quad \mathcal{I}(\mathcal{G}') = faux$;
7. $\mathcal{I}(\mathcal{G} \iff \mathcal{G}') = vrai$ *si et seulement si* $\mathcal{I}(\mathcal{G}) = \mathcal{I}(\mathcal{G}')$;

L'interprétation $\mathcal{I}$ d'une formule $\mathcal{F}$ est dite *interprétation complète* de la formule $\mathcal{F}$ si et seulement si $P = vars(\mathcal{F})$, sinon elle est *partielle* (ou *incomplète*).

En général, on peut considérer une interprétation $\mathcal{I}$ comme un *ensemble* des littéraux vrais dans l'interprétation $\mathcal{I}$. Par exemple, pour la formule $\mathcal{F}$: $(a \implies b) \vee c$ et l'interprétation $\mathcal{I} : \mathcal{I}(a) = vrai, \mathcal{I}(b) = faux, \mathcal{I}(c) = vrai$; l'interprétation $\mathcal{I}$ est alors définie par : $\mathcal{I} = \{a, \neg b, c\}$. En utilisant cette représentation, l'interprétation $\mathcal{I}(\mathcal{F})$ est complète si et seulement si $|\mathcal{I}(\mathcal{F})| = |vars(\mathcal{F})|$, sinon elle est partielle ($|\mathcal{I}(\mathcal{F})| < |vars(\mathcal{F})|$).

Définition 1.2.13 (Formules satisfaites et falsifiées).
- *On dit qu'une formule propositionnelle* $\mathcal{F}$ *est* satisfaite *par une interprétation* $\mathcal{I}$ *si et seulement si* $\mathcal{I}(\mathcal{F}) = vrai$.
- *On dit qu'une formule propositionnelle* $\mathcal{F}$ *est* falsifiée *par une interprétation* $\mathcal{I}$ *si et seulement si* $\mathcal{I}(\mathcal{F}) = faux$.

D'après cette définition, une clause est satisfaite par une interprétation $\mathcal{I}$ si et seulement si au moins un de ses littéraux est *vrai* dans $\mathcal{I}$. Dans le cas où tous ses littéraux sont *faux* dans $\mathcal{I}$, la clause est falsifiée.

Définition 1.2.14.
Une formule $\mathcal{F}$ *sous forme CNF est* satisfaite *si et seulement si toutes ses clauses sont satisfaites.*

Définition 1.2.15 (Modèle et contre-modèle).
- *On appelle* modèle *(ou* interprétation consistante*) d'une formule* $\mathcal{F}$ *une interprétation* $\mathcal{I}$ *qui la satisfait.*
- *On appelle* contre-modèle *(ou* interprétation inconsistante *ou* no-good *en anglais) d'une formule* $\mathcal{F}$ *une interprétation* $\mathcal{I}$ *qui la falsifie.*

On note par $\mathscr{M}(\mathcal{F})$ l'*ensemble des modèles* de la formule propositionnelle $\mathcal{F}$.

Définition 1.2.16 (Consistance et inconsistance).
- *Une formule est dite* consistante *(ou* satisfiable *ou* satisfaisable *ou* cohérente*) si et seulement si elle admet au moins un modèle.*
- *Une formule est dite* inconsistante *(ou* insatisfaisable *ou* incohérente*) si et seulement si elle n'admet pas de modèle.*

Définition 1.2.17 (Validité et invalidité).

- *Une formule $\mathcal{F}$ est dite* valide *(ou* tautologie*) si et seulement si elle est satisfaite par toutes les interprétations, c'est-à-dire : pour toute interprétation $\mathcal{I}$, on a $\mathcal{I}(\mathcal{F}) = vrai$ (notée $\mathcal{F} \equiv \top$).*
- *Une formule $\mathcal{F}$ est dite* invalide *si et seulement si elle n'admet que des contre-modèles (notée $\mathcal{F} \equiv \bot$).*

D'après cette définition, une clause est une tautologie si elle contient au moins deux littéraux complémentaires.

Définition 1.2.18 (Conséquence sémantique).
Soient $\mathcal{F}$ et $\mathcal{G}$ deux formules de $\mathscr{F}$. On dit que la formule $\mathcal{G}$ est une conséquence sémantique *(ou une* conséquence logique *ou un* impliquant*)*[3] *de la formule $\mathcal{F}$ si et seulement si $\mathscr{M}(\mathcal{F}) \subseteq \mathscr{M}(\mathcal{G})$. On le note par : $\mathcal{F} \models \mathcal{G}$.*

Définition 1.2.19 (Équivalence sémantique).
Soient $\mathcal{F}$ et $\mathcal{G}$ deux formules de $\mathscr{F}$. On dit que les formules $\mathcal{F}$ et $\mathcal{G}$ sont sémantiquement équivalentes *si et seulement si $\mathcal{F} \models \mathcal{G}$ et $\mathcal{G} \models \mathcal{F}$. On le note par : $\mathcal{F} \equiv \mathcal{G}$.*

Si deux formules $\mathcal{F}$ et $\mathcal{G}$ sont sémantiquement équivalentes, alors elles admettent exactement les mêmes modèles.

Remarque 1.2.2 (Subsumption ou sous-sommation). Si la clause C_1 subsume la clause C_2, on a alors $C_1 \models C_2$.

1.3 Problème SAT

1.3.1 Définition du problème

Définition 1.3.1 (Problème SAT).
Soit $\mathcal{F}$ une formule CNF. Le problème SAT est un problème de décision qui consiste à déterminer si $\mathcal{F}$ admet ou non un modèle.

En d'autre termes, il est décrit par la question suivante : « Est-ce que la formule CNF est satisfiable ? ». Remarquons que cela n'implique pas de trouver un modèle de la formule, mais qu'il faut en prouver l'existence. Parfois, la formule propositionnelle $\mathcal{F}$ est appelée une *instance* SAT.

Exemple 1.3.1.

3. On dit aussi que $\mathcal{F}$ implique $\mathcal{G}$

Donnée *Soit $\mathcal{F}$ une formule CNF définie comme suit :*

$$\begin{aligned}\mathcal{F} = \quad & (a \vee b \vee c) \quad \wedge \\ & (\neg a \vee b) \quad \wedge \\ & (\neg b \vee c) \quad \wedge \\ & (\neg c \vee a)\end{aligned}$$

Remarquons qu'on peut définir cette formule aussi comme suit :

$$\mathcal{F} = \{(a \vee b \vee c), (\neg a \vee b), (\neg b \vee c), (\neg c \vee a)\}$$

Question *Est-ce que la formule $\mathcal{F}$ est satisfaisable ?*

Réponse *Pour cet exemple, la réponse est oui : l'instanciation $\mathcal{I}$: $a = vrai$, $b = vrai$, $c = vrai$ satisfait la formule $\mathcal{F}$.*

La formule $\mathcal{F}$ admet un seul modèle $m = \{a, b, c\}$. Prenons la formule $\mathcal{F}' = \mathcal{F} \wedge (\neg a \vee \neg b \vee \neg c)$. Comme $\mathcal{F}$ admet un seul modèle m qui ne satisfait pas la clause $(\neg a \vee \neg b \vee \neg c)$, alors $\mathcal{F}'$ est insatisfaisable.

Le problème SAT est un des problèmes qui a été très étudié en informatique, du fait de son importance aussi bien théorique que pratique. En fait, le problème SAT est le premier problème prouvé comme étant NP-complet (Non déterministe Polynomial) par Cook en 1971 [Cook, 1971]. Par sa simplicité, la forme CNF est généralement utilisée dans les solveurs SAT modernes. En effet, de nombreux problèmes s'expriment naturellement comme une conjonction de clauses.

1.3.2 Classes polynomiales de SAT

Nous rappelons que définir une classe polynomiale nécessite deux algorithmes de complexité polynomiale :

- un algorithme de reconnaissance, qui décide si l'instance du problème appartient à cette classe ou pas ;
- un algorithme de décision de la satisfaisabilité des instances de cette classe.

Il est clair qu'il existe des classes d'instances ne répondant qu'à l'un ou l'autre des critères. Ces classes sont quasiment inexploitables en pratique. Nous pouvons citer, par exemple, la classe constituée de l'ensemble des instances de SAT ayant un nombre polynomial de résolvantes. Clairement cette classe admet un algorithme de décision polynomial[4]. Nous appelons

4. Il suffit de calculer l'ensemble de ses résolvantes, ce qui est fait en temps et en espace polynomial par définition de la classe, et de répondre « oui l'instance admet une solution » ou « non l'instance n'admet pas de solution », en fonction de la production ou non de la clause vide.

restriction polynomiale une classe pour laquelle seul un algorithme de décision polynomial existe. Très peu de classes polynomiales de SAT sont connues à ce jour. Nous en établissons dans ce paragraphe une liste non exhaustive.

1.3.2.1 Clauses binaires

La classe polynomiale 2-SAT [Cook, 1971], contient les instances de SAT formées de clauses binaires. L'algorithme de reconnaissance pour cette classe est trivial et clairement linéaire. L'algorithme de décision de satisfaisabilité proposé par S.A. Cook se base sur le principe de résolution et il est assurément polynomial puisque la résolvante de deux clauses binaires est au pire une clause binaire et que le nombre de clauses binaires pouvant être construites, pour un nombre donné de variables, est une fonction quadratique de ce nombre de variables.

Propriété 1.3.1. 2-SAT $\in$ P [Cook, 1971].

Par la suite, d'autres algorithmes de complexité linéaire furent proposés. Citons celui de S. Even, A. Itai et A. Shamir [Even et al., 1976] se basant sur un algorithme énumératif avec un principe de retour arrière intelligent, ou encore celui de B. Aspvall, M. Plass et R. Tarjan [Aspvall et al., 1979] utilisant un codage des clauses en termes de graphes et l'algorithme de R. Tarjan [Tarjan, 1972] pour calculer les composantes fortement connexes de ce graphe.

1.3.2.2 Clauses de Horn

Une autre classe polynomiale bien connue de SAT est la classe Horn-SAT.

Définition 1.3.2 (Horn-SAT).
Horn-SAT est la restriction de SAT aux ensembles de clauses de Horn où une clause de Horn est une clause qui contient au plus un littéral positif.

L'algorithme de reconnaissance, comme pour les clauses binaires, est trivial et linéaire : il suffit de passer en revue les clauses une à une et de s'assurer qu'il y a bien au plus un littéral positif par clause. En ce qui concerne le test de satisfaisabilité d'un ensemble de clauses de Horn, plusieurs auteurs ont montré que la résolution des clauses positives unitaires, qui s'effectue en temps et en espace linéaire, est suffisante [Minoux, 1988], [Dalal, 1992] et [Rauzy, 1995]. D'autres algorithmes de complexité linéaire, se basant sur la théorie des graphes, ont été proposés [Dowling and Gallier, 1984], [Ghallab and Gonzalo, 1991] et [Scutellà, 1990].

Propriété 1.3.2. Horn-SAT $\in$ P.

En effet, les clauses binaires se retrouvent très fréquemment dans les problèmes réels, un traitement particulier sur les clauses binaires améliore considérablement l'efficacité des algorithmes de résolution. On sait aussi que les clauses de Horn sont à la base des langages de programmation logique. On retrouve ensuite les formules Horn-renommables (des formules qui se réduisent à Horn par un renommage adéquat des littéraux) et les q-Horn (une généralisation de Horn-SAT et 2-SAT). Il existe bien d'autres classes polynomiales comme par exemple celles exhibées par l'introduction de différentes formes de hiérarchies d'instances, par des restrictions sur le nombre d'occurrences etc....

1.4 Algorithmes de résolution de SAT

Dans la pratique, de nombreux algorithmes ont été proposés pour résoudre le problème SAT. Les principes à la base de ces méthodes sont très variés : résolution [Robinson, 1963, Robinson, 1965], réécriture [Boole, 1854, Boole, 1992, Boole, 2009, C.E.Shannon, 1938], énumération [Davis et al., 1962], comptage du nombre de solutions [Iwama, 1989], recherche locale [Selman et al., 1992], programmation linéaire en nombres entiers [Blair et al., 1986, Bennaceur, 1995], diagrammes binaires de décision [Akers, 1978], algorithmes génétiques et évolutionnistes [Hao et al., 2002] etc....

Parmi les algorithmes complets, nous distinguons deux types : les algorithmes syntaxiques (basés sur le principe de résolution de J.A. Robinson [Robinson, 1963, Robinson, 1965]) et les algorithmes sémantiques (à la Davis & Putnam) très utilisés en logique propositionnelle. Ces derniers semblent être les plus efficaces pour tester la satisfiabilité d'une formule propositionnelle.

1.4.1 Principe de résolution

Le principe de *résolution* est un principe élémentaire et fondamental en logique propositionnelle. Il fut initialement introduit par Robinson ([Robinson, 1965]). C'est un simple processus itératif où à chaque étape une nouvelle clause est générée par l'application de la règle de résolution.

Définition 1.4.1 (Résolvante).
Soient C_1 et C_2 deux clauses contenant respectivement les littéraux ℓ et $\neg\ell$. On définit la résolvante *entre C_1 et C_2 sur ℓ, notée $Res[C_1, C_2 : \ell]$, comme la clause $R = C_1 \cup C_2 - \{\ell, \neg\ell\}$.* [5]

La résolvante entre C_1 et C_2 sur ℓ est égale à la résolvante entre C_1 et C_2 sur $\neg\ell$, c'est-à-dire, $Res[C_1, C_2 : \ell] = Res[C_1, C_2 : \neg\ell]$

5. On a considéré les clauses comme des ensembles de littéraux.

Exemple 1.4.1.
Soit $\mathcal{F}$ une formule CNF contenant les deux clauses suivantes :

$$C_1 : a \vee b \quad et \quad C_2 : \neg a \vee c \vee d$$

La résolvante entre C_1 et C_2 sur a est : $Res[C_1, C_2 : a] = b \vee c \vee d$.

Propriété 1.4.1. Soient $\mathcal{F}$ une formule propositionnelle, C_1 et C_2 deux clauses de $\mathcal{F}$ contenant respectivement les littéraux ℓ et $\neg\ell$ et R la résolvante de C_1 et C_2. Les deux propriétés suivantes sont vérifiées :
- $C_1 \wedge C_2 \models R$ (R est une conséquence logique de C_1 et C_2) ;
- $\mathcal{F} \equiv \mathcal{F} \cup \{R\}$.

La règle de résolution est le processus de dérivation d'une résolvante à partir de deux clauses contenant deux littéraux complémentaires.

Définition 1.4.2 (Dérivation par résolution et réfutation).
Une dérivation par résolution *(ou* preuve par résolution*) d'une clause C à partir de $\mathcal{F}$ est une séquence $\pi = [C_1, C_2, \ldots, C_k = C]$ telle que pour chaque $i \in \{1, \ldots, k\}$:*
- *ou bien $C_i \in \mathcal{F}$,*
- *ou bien $C_i = Res[C_\alpha, C_\beta : \ell]$ avec $1 \leq \alpha, \beta < i$.*

Remarquons que chaque clause C_i (avec $1 \leq i \leq k$) est une conséquence logique (ou un impliquant) de la formule $\mathcal{F}$, c'est-à-dire : $\mathcal{F} \models C_i$. Toute dérivation par résolution d'une clause vide ($C = \bot$) à partir de $\mathcal{F}$ est appelée réfutation. *Dans ce cas, $\mathcal{F}$ est insatisfaisable ($\mathcal{F} \models \bot$). On dit que la formule $\mathcal{G}$ est* déduite par réfutation *de la formule $\mathcal{F}$ si et seulement si $\mathcal{F} \wedge \neg\mathcal{G} \models \bot$.*

Propriété 1.4.2. La résolution est un système de preuve complet pour la réfutation.

Une autre règle souvent omise, appelée *fusion* consiste à remplacer les multiples occurrences d'un littéral par une seule occurrence de ce littéral. Il est également possible d'ajouter une troisième étape au principe de résolution qui consiste à supprimer certaines clauses redondantes de la base de connaissances.

L'application de la subsumption et de la fusion à chaque étape de la résolution rend le système complet. Si la formule est insatisfaisable, on obtient une clause vide ; sinon, un point fixe (toute nouvelle résolvante est subsumée par une clause existante) est atteint, et la formule est donc satisfiable.

Ce principe de résolution est relativement inefficace car le nombre de résolvantes à effectuer est souvent énorme ; l'espace mémoire requis devient donc très important et les temps de calculs prohibitifs. De nombreuses stratégies ont été proposées afin de minimiser le nombre de clauses produites, parmi elles citons :
- la N-résolution [Robinson, 1983] ;

- la résolution régulière [Davis et al., 1962, Tseitin, 1968, Galil, 1977] ;
- la résolution sémantique (PI-clash) [Chang and Lee, 1973] ;
- la résolution linéaire et la SL-résolution [Kowalski and Kuehner, 1971, Cubbada and Mouseigne, 1988] ;
- la résolution étendue [Tseitin, 1968] ;
- la résolution par entrée [Chang and Lee, 1973] ;
- la résolution unitaire [Dowling and Gallier, 1984, Escalada-Imaz, 1989] ;
- la résolution dirigée [Dechter and Rish, 1994] ;
- la résolution bornée [Génisson and Siegel, 1994].

1.4.2 Procédure de Davis & Putnam

La procédure de Davis & Putnam (DP) [Davis and Putnam, 1960] est un algorithme énumératif simple qui consiste en une exploration systématique de l'ensemble des interprétations d'une formule afin de déterminer si l'une d'entre elles est un modèle. Cette procédure peut être assimilée à une forme de résolution que l'on nomme résolution dirigée [Dechter and Rish, 1994]. Une grande partie des algorithmes complets les plus efficaces pour SAT s'appuient sur une version améliorée de Davis & Putnam (DPLL) [Davis et al., 1962].

1.4.2.1 Propagation Unitaire

La propagation unitaire (PU), aussi connue sous le nom de propagation de contraintes booléennes (BCP), est l'équivalent sémantique de la résolution unitaire. La PU est l'un des procédés clés dans les algorithmes de résolution de SAT. Son principe de fonctionnement est le suivant : tant que la formule contient une clause unitaire, affecter son littéral à *vrai*, c'est-à-dire la PU consiste à supprimer de la base de clauses, toutes les clauses contenant le littéral formant la clause unitaire et à supprimer toutes les occurrences du littéral complémentaire dans les autres clauses. C'est-à-dire « raccourcir » les clauses contenant le littéral complémentaire. La PU est l'application répétée de cette simplification jusqu'à ce que la base de clauses ne contienne plus de clauses unitaires ou jusqu'à l'obtention d'une clause vide. Nous présentons dans l'algorithme 1 une version récursive de la propagation unitaire.

Formellement, si $\mathcal{F}$ est une formule propositionnelle contenant une clause unitaire $C = \{\ell\}$, on définit la formule $\mathcal{P}u[\mathcal{F}, \ell]$ comme suit :

$$\mathcal{P}u[\mathcal{F}, \ell] = \{C \mid C \in \mathcal{F},\ \{\ell, \neg\ell\} \cap C = \varnothing\} \cup \{C - \{\neg\ell\} \mid C \in \mathcal{F},\ \neg\ell \in C\}$$

C'est-à-dire $\mathcal{P}u[\mathcal{F}, \ell]$ est une formule obtenue à partir de $\mathcal{F}$ en éliminant les clauses contenant ℓ et en supprimant $\neg\ell$ dans les autres clauses où il apparaît (c'est-à-dire la formule $\mathcal{F}$ avec l'affectation de ℓ à *vrai*). Cette notation est étendue aux interprétations : soit $\mathcal{I} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ une interprétation, on

Algorithme 1: Propagation unitaire (PU)

Entrée : $\mathcal{F}$ un ensemble de clauses
Sortie : $\mathcal{F}|_{pu}$ l'ensemble de clauses $\mathcal{F}$ simplifié par la propagation unitaire

```
1  si F ne contient aucune clause unitaire alors
2  |  retourner F|_pu = F
3  sinon
4  |  tant que F contient une clause unitaire c = {ℓ} faire
5  |  |  si F contient la clause unitaire {¬ℓ} alors
6  |  |  |  retourner F|_pu = ⊥
7  |  |  sinon
8  |  |  |  F = Pu[F, ℓ] c'est-à-dire :
9  |  |  |  Supprimer de F toutes les clauses contenant ℓ
10 |  |  |  Supprimer dans les clauses de F toutes les occurrences de ¬ℓ
11 |  retourner F|_pu = F
```

définit

$$\mathcal{F}_{\mathcal{I}} \stackrel{\text{def}}{=} \mathcal{P}u[\mathcal{F}, \mathcal{I}] = \mathcal{P}u[\ldots \mathcal{P}u[\mathcal{P}u[\mathcal{F}, \ell_1], \ell_2] \ldots, \ell_n]$$

Définition 1.4.3 (Propagation unitaire).
On note par $\mathcal{F}|_{pu}$ la formule obtenue à partir de $\mathcal{F}$ par application de la propagation unitaire. $\mathcal{F}|_{pu}$ est définie récursivement comme suit :

- *$\mathcal{F}|_{pu} = \mathcal{F}$ si $\mathcal{F}$ ne contient aucune clause unitaire.*
- *$\mathcal{F}|_{pu} = \bot$ si $\mathcal{F}$ contient deux clauses unitaires $\{\ell\}$ et $\{\neg\ell\}$.*
- *autrement, $\mathcal{F}|_{pu} = \mathcal{P}u[\mathcal{F}, \ell]|_{pu}$ tel que ℓ est le littéral qui apparaît dans une clause unitaire de $\mathcal{F}$.*

Définition 1.4.4 (U-inconsistance, U-consistance).
Soit $\mathcal{F}$ une formule propositionnelle. On dit que $\mathcal{F}$ est U-inconsistante *si et seulement si $\mathcal{F}|_{pu} = \bot$. Sinon, $\mathcal{F}$ est* U-consistante .

Définition 1.4.5 (Déduction par propagation unitaire).
Soit $\mathcal{F}$ une formule CNF et $\ell \in lits(\mathcal{F})$. On dit que ℓ est déduit par propagation unitaire de $\mathcal{F}$ (notée $\mathcal{F} \models_{pu} \ell$) si et seulement si la formule $\mathcal{F} \wedge \neg\ell$ est U-inconsistante. Une clause C est déductible par propagation unitaire de $\mathcal{F}$ (notée $\mathcal{F} \models_{pu} C$) si et seulement si $\mathcal{F} \wedge \neg C$ est U-inconsistante.

Définition 1.4.6 (Littéral pur).
Soit $\mathcal{F}$ une formule CNF et $\ell \in lits(\mathcal{F})$. Le littéral ℓ est appelé littéral pur *de $\mathcal{F}$ si et seulement si $\neg\ell$ n'apparait pas dans les clauses de $\mathcal{F}$.*

Propriété 1.4.3 (Simplification par les littéraux purs). Soit $\mathcal{F}$ une formule CNF et ℓ un littéral pur de $\mathcal{F}$. Le littéral ℓ peut être propagé à *vrai* tout en préservant la valeur logique de la formule $\mathcal{F}$.

Par conséquent, la simplification d'une formule $\mathcal{F}$ sous forme CNF par un littéral pur consiste à supprimer de $\mathcal{F}$ toutes les clauses contenant ce littéral pur. On note $\mathcal{F}|_{slp}$ la formule $\mathcal{F}$ simplifiée par l'ensemble de tous les littéraux purs.

1.4.2.2 Procédure DPLL

La procédure de Davis, Putnam, Logemann et Loveland (communément appelée DPLL) [Davis et al., 1962] est basée sur une recherche systématique dans l'espace des interprétations. Cet espace peut être représenté par un *arbre binaire de recherche* (ou simplement *arbre de recherche*) où chaque nœud représente une sous-formule simplifiée par l'instanciation partielle courante. À chaque nœud de l'arbre, une variable de décision est choisie et affectée à une valeur de vérité *vrai* ou *faux* et éventuellement une séquence de propagation des littéraux purs et des mono-littéraux est effectuée (cf. Algorithme 2).

Algorithme 2: Procédure DPLL

Entrée : $\mathcal{F}$ un ensemble de clauses
Sortie : *vrai* si $\mathcal{F}$ est consistante, *faux* sinon
Fonction : `DPLL(`*une formule propositionnelle*`)`

```
1  I = ∅
2  début DPLL(F)
3      si I(F) = vrai alors
4          retourner vrai
5      F = F|_pu ;                    // Propagation des clauses unitaires
6      F = F|_slp;                    // Simplification par les littéraux purs
7      si F contient une clause vide alors
8          retourner faux ;                                  // Réfutation
9      Choisir une variable x de vars(F) non affecté.
10     ℓ = x ou ℓ = ¬x;                                      // Séparation
11     I = I ∪ {ℓ}
12     DPLL(Pu[F, ℓ])
```

Le choix de la prochaine variable à instancier fait l'objet en général d'une heuristique (e.g. choix de la variable figurant le plus souvent dans les clauses les plus courtes). DPLL est une méthode de type « *choix* + *propagation* ». D'un autre point de vue, la procédure DPLL procède par séparation : Pour

une formule $\mathcal{F}$ contenant la variable x, on a $\mathcal{F}$ est satisfaisable si et seulement si $\mathcal{P}u[\mathcal{F}, x]$ ou $\mathcal{P}u[\mathcal{F}, \neg x]$ est satisfaisable.

La procédure DPLL est une procédure de recherche de type « *backtrack* » (retour-arrière). À chaque nœud de l'arbre de recherche les littéraux affectés (le littéral de décision et les littéraux propagés) sont étiquetés avec le même *niveau de décision* (noté par *Niveau*(ℓ) pour le niveau d'affectation du littéral ℓ) initialisé à 1 à la racine et incrémenté de 1 à chaque nouveau point de décision. Le niveau de décision courant est le niveau le plus élevé dans la pile de propagation. Lors d'un backtrack, les variables ayant un niveau supérieur au niveau du backtrack sont désaffectées et le niveau de décision courant est décrémenté en conséquence (égal au niveau du backtrack).

Remarque 1.4.1. Au niveau i, l'interprétation partielle courante $\mathcal{I}$ peut être représentée comme une *séquence décision–propagations* de la forme $\mathcal{I} = \prod_{i=1}^{m} \langle (x^i), y_1^i, y_2^i, \ldots, y_{k_i}^i \rangle$ telle que le premier littéral x^i correspond au littéral de décision x affecté au niveau i et chaque $y_{k_j}^i$, avec $1 \leq k_j \leq k_i$, représente un littéral propagé à ce même niveau i.

Définition 1.4.7.
Soient $\mathcal{F}$ une formule CNF et $\mathcal{I}$ une interprétation de $\mathcal{F}$. Pour un niveau d'affectation donné α, $\mathcal{I}^\alpha$ représente la projection de l'interprétation $\mathcal{I}$ *aux littéraux affectés au niveau $\leq \alpha$. C'est-à-dire $\mathcal{I}^\alpha = \prod_{i=1}^{\alpha} \langle (x^i), y_1^i, y_2^i, \ldots, y_{k_i}^i \rangle$.*

Si $\alpha = 0$, alors $\mathcal{I}^\alpha = \varnothing$, c'est-à-dire, aucun littéral de la formule propositionnelle n'est affecté.

1.4.3 Apprentissage dirigé par les conflits (CDCL)

Cette section présente le schéma d'apprentissage dirigé par les conflits CDCL (« Conflict Driven Clauses Learning » en anglais), une des composantes clés des solveurs SAT modernes.

La structure de données principale d'un solveur basé sur un schéma CDCL est le *graphe d'implication* qui enregistre les différentes propagations faites durant la construction d'une interprétation partielle. Ce graphe d'implication permet d'analyser les conflits. Il est utilisé pour effectuer du retour-arrière intelligent, et pour apprendre des no-goods. Il est important de noter que ce graphe est construit de manière incomplète et ne donne qu'une vue partielle des implications entre les littéraux.

Regardons par exemple ce qui se passe lorsqu'un littéral y est déduit à un niveau donné. Cette déduction est faite parce qu'une clause, par exemple $C = (\neg x_1 \vee \neg x_2 \vee y)$, est devenue unitaire sous l'interprétation courante. Cette interprétation contient donc les affectations $x_1 = \mathit{vrai}$ et $x_2 = \mathit{vrai}$ qui ont eu lieu à des niveaux inférieurs ou égaux au niveau d'affectation de y. Lorsque y est impliqué, la raison de cette implication est enregistrée, on ajoute donc

au graphe d'implication des arcs entre x_1 et y et entre x_2 et y ramifiés par la clause C.

1.4.3.1 Graphe d'implications

Le graphe d'implication est une représentation capturant les variables affectées durant la recherche, que ce soit des variables de décision ou des variables propagées.

Définition 1.4.8 (Cause d'un littéral propagé).
Soit $\mathcal{F}$ une formule CNF, $C = x_1 \vee \cdots \vee x_n \vee y$ une clause de $\mathcal{F}$ non unitaire et $\mathcal{I}$ une interprétation partielle de $\mathcal{F}$. Si $\mathcal{I}(x_i) =$ faux pour $1 \leq i \leq n$ et $\mathcal{I}(y)$ est indéfini, alors :

- *La clause C devient une clause unitaire sous l'interprétation courante $\mathcal{I}$;*
- *La propagation unitaire appliquée sur la formule courante $\mathcal{F}' = \mathcal{P}u[\mathcal{F}, \mathcal{I}]$ conduit à propager le littéral y.*

La clause d'origine $C = x_1 \vee \cdots \vee x_n \vee y$ est appelée la cause de la propagation *du littéral y sous l'interprétation courante $\mathcal{I}$ et on la note $Cause(y)_{\mathcal{I}} = C$ (ou bien $Cause(y)$ s'il n'y a pas de confusion).*

La cause de la propagation d'un littéral y propagé sous une interprétation partielle $\mathcal{I}$ est la même sous toutes les interprétations étendues de $\mathcal{I}$.

Quand un littéral y n'est pas obtenu par propagation unitaire mais qu'il correspond à un point de choix, $Cause(y)$ est indéfinie, on la note par convention $Cause(y) = \bot$.

Définition 1.4.9 (Ensemble des explications).
Soient $\mathcal{F}$ une formule CNF, $\mathcal{I}$ une interprétation partielle de $\mathcal{F}$ et $\ell \in lits(\mathcal{F})$ tel que $Cause(y) \neq \bot$. On définit l'ensemble des explications *de y par : $Exp(y) = \{\neg x \mid x \in Cause(y) - \{y\}\}$*

Autrement dit, les explications sont les littéraux x_i qui constituent la condition sous laquelle la clause $Cause(y)$ devient une clause unitaire $\{y\}$. Notons que pour tout $x \in Exp(y)$ on a $Niveau(x) \leq Niveau(y)$, c'est-à-dire, toutes les explications de la déduction ont un niveau inférieur à celui de y.

Quand $Cause(y) = \bot$, on dit par convention que $Exp(y) = \varnothing$.

Définition 1.4.10 (Graphe d'implication).
Soient $\mathcal{F}$ une formule CNF et $\mathcal{I}$ une interprétation partielle de $\mathcal{F}$. Le graphe d'implication *associé à $\mathcal{F}$ et $\mathcal{I}$ est $\mathcal{G}^{\mathcal{I}}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$ tel que :*

- *$\mathcal{N} = \mathcal{I}$, c'est-à-dire, on définit un nœud de $\mathcal{N}$ pour chaque littéral, de décision ou impliqué, de $\mathcal{I}$;*
- *$\mathcal{E} = \{(x, y) \mid x \in \mathcal{I},\ y \in \mathcal{I},\ x \in Exp(y)\}$*

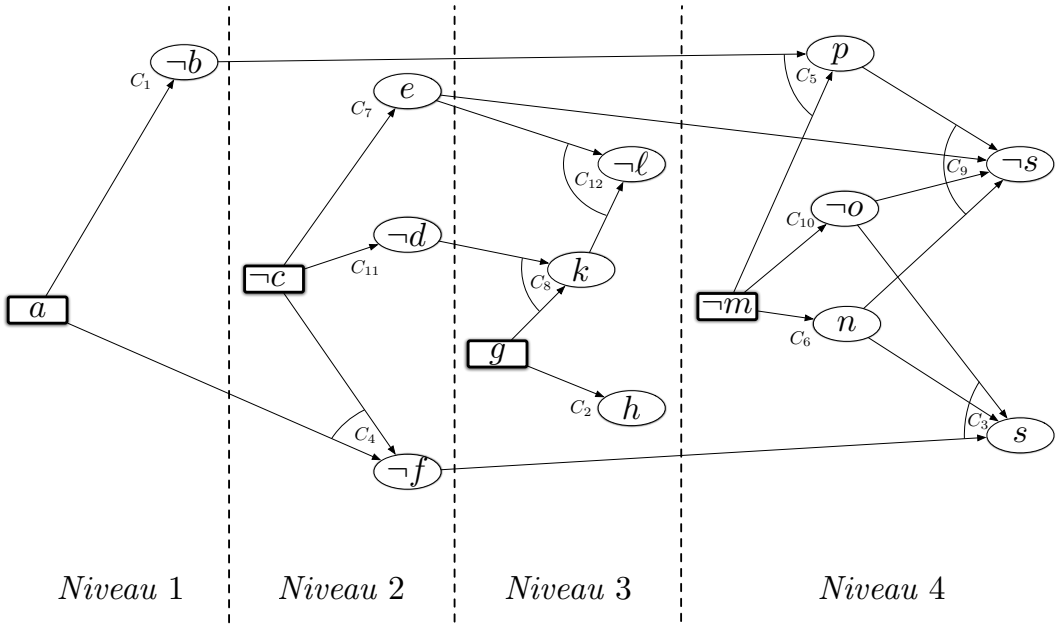

FIGURE 1.1 – graphe d'implication associé à $\mathcal{F}$ et $\mathcal{I}$: $\mathcal{G}^{\mathcal{I}}_{\mathcal{F}} = (\mathscr{N}, \mathscr{E})$

Exemple 1.4.2.
Soit $\mathcal{F}$ une formule CNF contenant, entre autres, les clauses suivantes :

$$
\begin{array}{llll}
\mathcal{F} = & C_1 : \neg a \vee \neg b & & C_7 : c \vee e \\
& C_2 : \neg g \vee h & & C_8 : d \vee \neg g \vee k \\
& C_3 : f \vee \neg n \vee o \vee s & & C_9 : \neg n \vee o \vee \neg e \vee \neg p \vee \neg s \\
& C_4 : \neg a \vee c \vee \neg f & & C_{10} : m \vee \neg o \\
& C_5 : b \vee m \vee p & & C_{11} : c \vee \neg d \\
& C_6 : m \vee n & & C_{12} : \neg k \vee \neg e \vee \neg \ell
\end{array}
$$

Soit $\mathcal{I}$ l'interprétation partielle inconsistante suivante :

$$\mathcal{I} = \langle (a^1), \neg b^1 \rangle \langle (\neg c^2), \neg f^2, \neg d^2, e^2 \rangle \langle (g^3), h^3, k^3, \neg \ell^3 \rangle \langle (\neg m^4), n^4, \neg o^4, p^4, s^4, \neg s^4 \rangle$$

Le niveau de décision courant est 4. Le graphe d'implication $\mathcal{G}^{\mathcal{I}}_{\mathcal{F}}$ associé à $\mathcal{F}$ et $\mathcal{I}$ est schématisé dans la figure 1.1 (page 19). Une seule partie du graphe est affichée.

Remarque 1.4.2. Les clauses qui sont les causes de propagation de littéraux sont utilisées pour étiqueter les arcs ramifiés représentant l'explication du littéral propagé.

1.4.3.2 Génération des clauses assertives

Dans cette section, on décrit formellement le schéma d'apprentissage classique utilisé dans les solveurs SAT modernes. Dans les définitions suivantes

$\mathcal{F}$ est une formule CNF, $\mathcal{I}$ est une interprétation partielle telle que $\mathcal{P}u[\mathcal{F},\mathcal{I}]$ est U-inconsistante (c'est-à-dire $\mathcal{P}u[\mathcal{F},\mathcal{I}]|_{pu}= \bot$) et $\mathcal{G}^{\mathcal{I}}_{\mathcal{F}} = (\mathscr{N},\mathscr{E})$ est le graphe d'implication associé.

Supposons que le niveau de décision courant est μ, puisque l'on a atteint le conflit, il existe donc un littéral ℓ tel que $\{\ell,\neg\ell\} \subset \mathscr{N}$ et $Niveau(\ell) = \mu$. L'analyse du conflit est basée sur l'application de la résolution (sur ℓ) sur la clause devenue fausse et en remontant dans le graphe d'implication, et ceci en utilisant les clauses $Cause(x)$ à chaque nœud $x \in \mathscr{N}$. Le processus s'arrête lorsqu'il ne reste plus qu'un littéral x du dernier niveau d'affectation. On appelle ce processus *la preuve par résolution basée sur les conflits*.

Définissons formellement les différents concepts de clause assertive et de preuve par résolution basée sur les conflits.

Définition 1.4.11 (Clause assertive).
Soient $\mathcal{F}$ une formule CNF, $\mathcal{I}$ une interprétation partielle inconsistante de $\mathcal{F}$ et μ le niveau de décision courant. Une clause C de la forme $C = x_1 \vee \cdots \vee x_n \vee y$ est appelée une clause assertive *si et seulement si $\mathcal{I}(C) = faux$, $Niveau(y) = \mu$ et $Niveau(x_i) < Niveau(y)$ pour $i \in \{1,\ldots,n\}$. Le littéral y est appelé* littéral assertif*, qu'on notera $Assertif(C)$. On définit également le* niveau assertif $Saut(C) \stackrel{\text{def}}{=} \max\{Niveau(x) \mid x \in C - \{y\}\}$.

Il se peut que la clause assertive C soit une clause unitaire de la forme $C = y$, dans ce cas le littéral assertif est y et le niveau assertif est $Saut(C) = 0$.

Définition 1.4.12 (Preuve par résolution basée sur les conflits).
Une preuve par résolution basée sur les conflits est une preuve par résolution

$$\pi = [Cause(\zeta), Cause(\neg\zeta), R_1, Cause(\neg\alpha_1), R_2, Cause(\neg\alpha_2), \ldots, Cause(\neg\alpha_{k-1}), R_k]$$

(voir définition 1.4.2 page 13) satisfaisante les conditions suivantes :

1. $R_1 = Res[Cause(\zeta), Cause(\neg\zeta) : \zeta]$;
2. *Pour tous $i \in \{2,\ldots,k\}$, R_i est construite en sélectionnant un littéral $\alpha_{i-1} \in R_{i-1}$ pour lequel $Cause(\neg\alpha_{i-1})$ est défini. On a alors : $\alpha_{i-1} \in R_{i-1}$ et $\neg\alpha_{i-1} \in Cause(\neg\alpha_{i-1})$, deux clauses pouvant entrer en résolution. La clause R_i est définie comme : $R_i = Res[R_{i-1}, Cause(\neg\alpha_{i-1}) : \alpha_{i-1}]$;*
3. *Enfin, R_k est une clause assertive.*

Une remarque importante est que dans les solveurs SAT modernes, les littéraux α utilisés dans la condition 2 de la définition 1.4.12 sont restreints à ceux du niveau courant. On dit que la preuve par résolution basée sur les

conflits $\pi = [R_1, R_2, \ldots, R_k]$ est une *preuve élémentaire* si π ne contient qu'une seule clause assertive R_k (toutes les clause R_i, avec $1 \leq i < k$ ne sont pas des clauses assertives).

Remarque 1.4.3. La définition précédente et la propriété 1.4.1 (page 13) assurent qu'on peut rajouter la clause assertive R_k à l'ensemble de clauses $\mathcal{F}$. En fait, toutes les clauses de la preuve par résolution basée sur les conflits sont des conséquences logiques de la formule $\mathcal{F}$, et pour chaque résolvante R_i on a $\mathcal{F} \equiv \mathcal{F} \cup \{R_i\}$ (avec $1 \leq i \leq k$). En particulier, la clause assertive R_k est une conséquence logique de la formule $\mathcal{F}$ qui est équivalente à $\mathcal{F} \cup \{R_k\}$.

Exemple 1.4.3.
Pour illustrer les définitions précédentes, considérons à nouveau l'exemple 1.4.2 (page 18). Le parcours du graphe $\mathcal{G}^{\mathcal{I}}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$ nous permet de générer quatre clauses assertives possibles (voir figure 1.1 page 19). Illustrons la preuve par résolution basée sur les conflits qui conduit à la première clause assertive Δ_1 :

$$\begin{array}{lcllcllcl}
R_1 & = & Res[C_9, C_3 & : s] & = & f^2 \vee o^4 \vee \neg e^2 \vee \neg p^4 \vee \neg n^4 & & \\
R_2 & = & Res[R_1, C_6 & : n] & = & f^2 \vee o^4 \vee \neg e^2 \vee \neg p^4 \vee m^4 & & \\
R_3 & = & Res[R_2, C_{10} & : o] & = & f^2 \vee \neg e^2 \vee \neg p^4 \vee m^4 & & \\
R_4 & = & Res[R_3, C_5 & : p] & = & f^2 \vee \neg e^2 \vee b^1 \vee m^4 & = & \Delta_1
\end{array}$$

Comme on peut le voir, R_4 permet de déduire une première clause assertive (qu'on notera Δ_1) car tous ses littéraux sont affectés avant le niveau actuel (leur niveau est inférieur à 4) sauf m qui est affecté au niveau courant 4. La preuve $[C_9, C_3, R_1, C_6, R_2, C_{10}, R_3, C_5, R_4]$ est une preuve élémentaire par résolution basée sur les conflits, et m est le littéral assertif de la clause Δ_1 ($m = Assertif(\Delta_1)$) et le niveau assertif de Δ_1 est $Saut(\Delta_1) = \max\{1, 2\} = 2$.

Si on continue un tel processus, on obtient en plus les trois clauses assertives :

$$\begin{array}{lcllcl}
\Delta_2 & = & Res[R_4, C_7 : e] & = & f^2 \vee c^2 \vee b^1 \vee m^4 \\
\Delta_3 & = & Res[\Delta_2, C_4 : f] & = & \neg a^1 \vee c^2 \vee b^1 \vee m^4 \\
\Delta_4 & = & Res[\Delta_3, C_1 : b] & = & \neg a^1 \vee c^2 \vee m^4
\end{array}$$

Propriété 1.4.4 (Clause assertive et retour-arrière). Soient $\mathcal{F}$ une formule CNF, $\mathcal{I}$ une interprétation partielle inconsistante et $C = x_1 \vee \cdots \vee x_n \vee y$ une clause assertive déduite à partir de l'analyse du conflit avec $Saut(C) = \max\{Niveau(x_i) \mid 1 \leq i \leq n\}$. Le solveur effectue un saut arrière au niveau $Saut(C)$ et considère la nouvelle interprétation partielle $\mathcal{I}^{Saut(C)} \cup \{y\}$

Exemple 1.4.4.
Dans l'exemple précédent, si on utilise la clause assertive Δ_1, alors le solveur

fait un saut du niveau 4 *au niveau* 2 *et considère l'interprétation partielle* $\mathcal{I} = \{a, \neg b, \neg c, \neg f, \neg d, e\} \cup \{m\}$

Propriété 1.4.5. Soient $\mathcal{F}$ une formule CNF, $\mathcal{I}$ une interprétation partielle inconsistante et $C = x_1 \vee \cdots \vee x_n \vee y$ une clause assertive déduite à partir de l'analyse du conflit avec $Saut(C) = \max\{Niveau(x_i) \mid 1 \leq i \leq n\}$. Alors, le littéral assertif y peut être déduit par propagation unitaire au niveau $Saut(C)$, c'est-à-dire, $\mathcal{P}u[\mathcal{F}, \mathcal{I}^{Saut(C)}] \wedge \neg y \models \bot$.

Démonstration. L'affectation de tous les littéraux x_i avec $1 \leq i \leq n$ à *faux* conduit exactement au même conflit par propagation unitaire. □

Exemple 1.4.5.
Considérons à nouveau l'exemple 1.4.2 (page 18). On a trouvé dans l'exemple 1.4.3 (page 21) que la clause Δ_1 *est une clause assertive,* m *est le littéral assertif de la clause* Δ_1 *et le niveau assertif de* Δ_1 *est* $Saut(\Delta_1) = 2$. *L'interprétation* $\mathcal{I}^{Saut(C)}$ *sera* $\mathcal{I}^{Saut(C)} = \langle(a^1), \neg b^1\rangle\langle(\neg c^2), \neg f^2, \neg d^2, e^2\rangle$ *et la formule* $\mathcal{P}u[\mathcal{F}, \mathcal{I}^{Saut(C)}]$ *contiendra, les clauses suivantes :*

$$
\begin{array}{lll}
C_1 : & \checkmark & \\
C_2 : & \neg g \vee h & \\
C'_3 : & \neg n \vee o \vee s & \\
C_4 : & \checkmark & \\
C'_5 : & m \vee p & \\
C_6 : & m \vee n &
\end{array}
\qquad
\begin{array}{lll}
C_7 : & \checkmark & \\
C'_8 : & \neg g \vee k & \\
C'_9 : & \neg n \vee o \vee \neg p \vee \neg s & \\
C_{10} : & m \vee \neg o & \\
C_{11} : & \checkmark & \\
C'_{12} : & \neg k \vee \neg \ell &
\end{array}
$$

Illustrons la dérivation du littéral assertif m *par réfutation à partir de* $\mathcal{P}u[\mathcal{F}, \mathcal{I}^{Saut(C)}]$, *c'est-à-dire,* $\mathcal{P}u[\mathcal{F}, \mathcal{I}^{Saut(C)}] \wedge \neg y \models \bot$ *:*

$$
\begin{aligned}
C'_5 \wedge \neg m &\models_{pu} p \\
C_6 \wedge \neg m &\models_{pu} n \\
C_{10} \wedge \neg m &\models_{pu} \neg o \\
C'_3 \wedge n \wedge \neg o &\models_{pu} s \\
C'_9 \wedge p \wedge n \wedge \neg o \wedge s &\models_{pu} \bot
\end{aligned}
$$

Le littéral assertif m *est déduit par propagation unitaire au niveau assertif* $Saut(\Delta_1) = 2$.

La propriété précédente montre que le littéral assertif peut être déduit par réfutation au niveau $Saut(C)$. Cela signifie que si on décide de tester par propagation unitaire à chaque niveau un ensemble de littéraux (lookahead local), alors on peut détecter ce conflit bien avant d'atteindre un niveau plus bas dans l'arbre de recherche.

L'algorithme 3 décrit globalement le principe de fonctionnement d'un solveur moderne. L'état S correspondant à une interprétation partielle I de la formule $\mathcal{F}$ dans l'arbre de recherche du solveur SAT considéré est donné par $S = (\mathcal{F}, \Gamma, D)$. L'ensemble $D = (\ell_1, \ell_2, \ldots, \ell_k)$ est l'ensemble des littéraux de décision de I où ℓ_i est le littéral de décision au niveau i. La formule Γ est une formule sous forme CNF telle que $\mathcal{F} \models \Gamma$. Les clauses de Γ sont des clauses assertives ajoutées à la formule initiale $\mathcal{F}$ et qui correspondent aux interprétations partielles générées qui sont des no-goods. Un état $S = (\mathcal{F}, \Gamma, D)$ est U-inconsistant (respectivement U-consistant) si et seulement si $\mathcal{F} \wedge \Gamma \wedge D$ est U-inconsistante (respectivement U-consistante).

Algorithme 3: Apprentissage dirigé par les conflits : CDCL

entrée : Une formule CNF $\mathcal{F}$
sortie : Une solution de $\mathcal{F}$ ou unsat si $\mathcal{F}$ n'est pas satisfiable

```
1  D ⟵ ⟨⟩ // Littéraux de décision;
2  Γ ⟵ vrai // Clauses apprises;
3  pour vrai faire
4      si S = (F, Γ, D) est U-inconsistant alors
5          // Il y a un conflit.
6          si D = ⟨⟩ alors
7              retourner unsat
8          c ⟵ une clause assertive de S
9          m ⟵ le niveau assertif de c
10         Γ ⟵ Γ ⋀_{c∈A} c
11         D ⟵ D_m // les m premières décisions
12     sinon
13         // Pas de conflit.
14         si redémarrage alors
15             D ⟵ ⟨⟩
16             S = (F, Γ, D)
17         Choisir un littéral ℓ tel que S ⊭_pu ℓ et S ⊭_pu ¬ℓ
18         si ℓ = null alors
19             retourner D // satisfiable
20         D ⟵ D, ℓ
```

La majorité des solveurs SAT modernes ont une stratégie de *redémarrage*, qui consiste à démarrer une nouvelle recherche et a priori avec un nouvel ordre des variables. Cette stratégie a été proposée par [Gomes et al., 1998]. La plupart de ces solveurs effectuent un redémarrage après qu'un certain seuil

de clauses apprises (ou de conflits) soit atteint. Le parcours de l'espace de recherche change d'un redémarrage à l'autre, ceci étant dû soit à l'aspect aléatoire présent dans les algorithmes de recherche, soit au changement de la formule d'entrée augmentée par les clauses apprises ou les deux à la fois.

La vraie difficulté, cependant, réside dans la manière de sélectionner efficacement les littéraux à pré-traiter par propagation unitaire. Une tentative de réponse à cette question est donnée dans les approches proposées par Dubois et al [DUBOIS et al., 1996] et par Li et Anbulagan [Li and Anbulagan, 1997].

Cette propriété simple montre que les solveurs SAT modernes explorent un arbre de recherche binaire, et peuvent être considérés comme une variante de la procédure DPLL. Ce point de vue n'est pas partagé par tous les membres de la communauté SAT.

Chapitre 2

Logiques non monotones [1]

Sommaire

2.1 Introduction 25
2.1.1 Raisonnement révisable 25
2.1.2 Logique classique et raisonnement valide 26
2.1.3 Logique des prédicats (Premier ordre) 27
2.2 Logiques non monotones 30
2.2.1 Logique préférentielle 31
2.2.2 $X-$logique 33
2.2.3 Logique des défauts 35

2.1 Introduction

2.1.1 Raisonnement révisable

Pour créer un système informatique d'intelligence artificielle fiable, nous avons besoin d'outils rigoureux pour formaliser et implanter des raisonnements au caractère révisable. Une grande part de notre intelligence dépend des raisonnements judicieux en présence d'une information qui n'est souvent qu'incomplète et évolutive. Dans l'absolu, ces raisonnements sont souvent simplement plausibles et peuvent devoir être révisés lorsque de nouvelles informations sont prises en compte. Ils ne peuvent pas être formalisés directement à l'aide de la logique classique.

Exemple 2.1.1.
Nous pouvons par exemple tenir le raisonnement trivial suivant. Sachant que

1. Pour plus de détails concernant les notions introduites dans ce chapitre, le lecteur peut se référer au livre de E. Grégoire [Grégoire, 1990].

la plupart des oiseaux peuvent voler et que Titi est un oiseau, on en conclut que Titi peut voler. *Cette inférence semble acceptable. Elle ne peut cependant être qualifiée d'absolument correcte car elle ne tient pas compte d'exceptions possibles. Elle est donc incertaine et peut être sujette à révision.*

En particulier, s'il est précisé que « Titi est une autruche » et que « les autruches sont des oiseaux qui ne volent pas », l'assertion Titi peut voler *doit être rétractée.*

2.1.2 Logique classique et raisonnement valide

Les systèmes déductifs de la logique classique se limitent à la formalisation du raisonnement *valide*. Tels quels, ils ne sont donc pas appropriés à la formalisation d'un raisonnement incertain et révisable.

– Un système formel de déduction de la logique classique est composé d'un ensemble de schémas d'axiomes et de règles d'inférence. Un tel système permet d'inférer des conclusions à partir de prémisses et définit donc une relation d'inférabilité entre formules, notée $\vdash$. Cette relation possède les propriétés suivantes ([Gabbay, 1985]) :

Propriété de réflexivité : $\{p_1, \ldots, p_n, q\} \vdash q$.
Inférer une conclusion identique à l'une des prémisses est une opération valide.

Propriété de monotonie : Si $\{p_1, \ldots, p_n\} \vdash q$ alors $\{p_1, \ldots, p_n, r\} \vdash q$.
Un résultat acquis n'est pas remis en cause par des résultats ultérieurs.

Propriété de transitivité :
Si $\{p_1, \ldots, p_n\} \vdash r$ et $\{p_1, \ldots, p_n, r\} \vdash q$ alors $\{p_1, \ldots, p_n\} \vdash q$.
Des résultats intermédiaires peuvent être utilisés pour établir la validité d'une conclusion.

Les systèmes formels de déduction de la logique classique apparaissent comme des systèmes algébriques de réécriture respectant ces différentes exigences. En particulier, la propriété de monotonie s'oppose à la formalisation directe d'un raisonnement révisable.

> *D'un point de vue purement syntaxique, construire un système d'inférence non monotone nécessite donc d'affaiblir les propriétés caractérisant les systèmes déductifs de la logique classique.*

– Une caractérisation sémantique d'un système logique s'établit en attribuant des valeurs sémantiques aux expressions du langage au moyen d'une *interprétation*. Cette caractérisation s'effectue par l'attribution de valeurs sémantiques aux expressions de base du langage et par la définition de règles permettant l'évaluation sémantique des expressions plus complexes. Lorsqu'un

ensemble A de formules sont vraies pour une interprétation, on appelle cette interprétation *modèle* de A.

La relation de conséquence sémantique entre un ensemble A de prémisses et une conclusion p est notée $\models$ et est classiquement définie comme suit : $A \models p$ si tout modèle de A est aussi un modèle de p. Dans cette définition le mot *tout* implique que seules les conséquences « certaines » (« valides ») de A peuvent être inférées.

Un raisonnement révisable n'est pas valide au sens classique du terme. Inférer p depuis un ensemble A de prémisses et rétracter p lorsqu'une information q est ajoutée à A signifie en effet que l'on accepte d'inférer p, alors qu'il existe un modèle de $A \cup \{q\}$ qui est a fortiori un modèle de A, qui ne vérifie pas p.

> *D'un point de vue sémantique, construire une logique non monotone nécessite la définition d'une relation d'inférence permettant de tirer des conclusions qui ne sont pas vérifiées dans tous les modèles des prémisses.*

Avant de présenter la logique des prédicats, on veut présenter la notation suivante :

Définition 2.1.1 (Formules impliquées).
Soit $\mathscr{L}$ une logique classique et $\mathscr{F}$ l'ensemble des formules de $\mathscr{L}$. Si $\mathcal{A}$ est un sous-ensemble de formules (ou une formule) de $\mathscr{F}$, alors $\overline{\mathcal{A}}$ (ou bien $Th(\mathcal{A})$) est l'ensemble des formules logiquement impliquées *par (ou les* théorèmes *associés à) $\mathcal{A}$. L'ensemble des formules $\mathcal{A}$ est* déductivement fermé *si $\mathcal{A} = \overline{\mathcal{A}}$ (ou bien $\mathcal{A} = Th(\mathcal{A})$).*

2.1.3 Logique des prédicats (Premier ordre)

La logique des prédicats, appelée également logique du premier ordre, est un formalisme logique plus expressif que la logique propositionnelle. Cependant, cette amélioration de l'expressivité a un coût : la logique des prédicats n'est que semi-décidable [Church, 1936] (un système est semi-décidable s'il n'existe pas de méthodes pour prouver en un temps fini qu'une formule n'est pas un théorème).

Comme nous l'avons fait pour la logique propositionnelle, nous commençons par définir la syntaxe de la logique du premier ordre avant de passer à la sémantique.

2.1.3.1 Syntaxe

Tout comme en logique des propositions, nous devons tout d'abord définir la logique des prédicats.

Définition 2.1.2 (Logique des prédicats).
La logique de la logique des prédicats LPr est construit à partir des ensembles dénombrables suivants (non nécessairement finis) :

- Un ensemble de variables *: notées x, y, z etc.... ;*
- Un ensemble de constantes individuelles *ou* constantes *notées a, b, c etc... ;*
- Un ensemble de constantes fonctionnelles *ou* noms de fonctions *notées f, g, h etc.... Chaque constante fonctionnelle f a une arité k ;*
- Un ensemble de constantes prédicatives *ou* noms de prédicat *notées P, Q, R etc.... Chaque constante prédicative P a une arité k ;*
- Un ensemble de connecteurs *:* $\neg, \vee, \wedge, \Longrightarrow, \Longleftrightarrow$ *(respectivement : non, ou, et, implique, équivalent) ;*
- Un ensemble de quantificateurs *: le quantificateur universel* $\forall$ *et existentiel* $\exists$ *;*
- Un ensemble de symboles *:* $\top$ *et* $\bot$ *et de symboles de ponctuation « (» et «) ».*

Le langage des prédicats contient le langage propositionnel, car une proposition (dans LP) n'est rien d'autre qu'une constante prédicative (dans LPr) sans arguments ou, plus précisément, une constante prédicative (dans LPr) d'arité nule.

Définition 2.1.3 (Terme).
Un terme *de la logique du premier ordre est une variable, une constante individuelle ou une forme fonctionnelle.*

Définition 2.1.4 (Forme fonctionnelle et forme prédicative).

- *Soit f une constante fonctionnelle n-aire (d'arité $k = n$) et soient $t_1, t_2, \ldots, t_n$ des termes, alors $f(t_1, t_2, \ldots, t_n)$ est une* forme fonctionnelle*. Les formes fonctionnelles d'arité $k = 0$ sont des constantes individuelles.*
- *Soit P une constante prédicative n-aire (d'arité $k = n$) et soient $t_1, t_2, \ldots, t_n$ des termes, alors $P(t_1, t_2, \ldots, t_n)$ est une* forme prédicative*. Les formes prédicatives d'arité $k = 0$ sont appelées* propositions*.*

Définition 2.1.5 (Atome).
Un atome est une forme prédicative ou une égalité, c'est-à-dire, une expression du type $(s = t)$, où s et t sont des termes.

Les mots « fonction » et « prédicat » sont parfois utilisés à la place des expressions « forme fonctionnelle » et « forme prédicative ».

Définition 2.1.6 (Formule).
L'ensemble des formules de la logique du premier ordre est le plus petit ensemble d'expressions tel que :

- *Les atomes sont des formules.*
- *Si A, B sont des formules alors (A), $\neg A$, $A \vee B$, $A \wedge B$, $A \implies B$ et $A \iff B$ sont des formules.*
- *Si A est une formule et x une variable alors $\forall x A$ et $\exists x A$ sont des formules.*

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

Définition 2.1.7 (Variables liées, variables libres et formules closes). *Lorsqu'une variable x appartient à une sous-formule précédée d'un quantificateur universel noté $\forall$, ou d'un quantificateur existentiel noté $\exists$, elle est dite* liée par ce quantificateur. *Une variable est dite* libre *si elle n'est liée par aucun quantificateur.*

Une formule close *(ou* formule fermée*) est une formule dont toutes les variables sont liées.*

Définition 2.1.8 (Substitution).
On appelle substitution le remplacement d'une variable x par le terme t, on note une telle substitution par (x/t). Si A est une formule on note alors $A(x/t)$ la formule obtenue en remplaçant dans A toutes les occurrences de x par t.

2.1.3.2 Sémantique

D'un point de vue sémantique, la logique des prédicats a pour but de déterminer quelles formules sont vraies ou fausses.

Définition 2.1.9 (Interprétation).
On appelle interprétation le triplet $\mathcal{I} = (\mathcal{D}, I_c, I_v)$ où :
- *$\mathcal{D}$ est un ensemble non vide appelé* domaine d'interprétation *;*
- *I_c une fonction qui associe :*
  - *à tout constante fonctionnelle f d'arité n une fonction $I_c(f)$ de $\mathcal{D}^n$ dans $\mathcal{D}$;*
  - *et à toute constante prédicative P d'arité m une fonction $I_c(P)$ de $\mathcal{D}^m$ dans $\{vrai, faux\}$;*
- *I_v est une fonction qui associe à toute variable un élément de $\mathcal{D}$.*

Définition 2.1.10.
L'interprétation d'une formule de la logique des prédicats A associe une interprétation $\mathcal{I}(A)$ à A de la manière suivante :
- *Si x est une variable libre alors $\mathcal{I}(x) = I_v(x)$;*
- *$\mathcal{I}(f(t_1, t_2, \ldots, t_n)) = (I_c(f))(\mathcal{I}(t_1), \mathcal{I}(t_2), \ldots, \mathcal{I}(t_n))$;*
- *$\mathcal{I}(P(t_1, t_2, \ldots, t_m)) = (I_c(P))(\mathcal{I}(t_1), \mathcal{I}(t_2), \ldots, \mathcal{I}(t_m))$;*
- *Si s et t sont des termes, alors $\mathcal{I}(s = t)$ est vrai si $\mathcal{I}(s) = \mathcal{I}(t)$ et faux sinon ;*

- *Si A et B sont des formules alors : $\neg A$, $A \vee B$, $A \wedge B$, $A \Longrightarrow B$ et $A \Longleftrightarrow B$ s'interprètent comme dans la logique propositionnelle (voir la définition 1.2.12 page 7) ;*
- *Si A est une formule et x une variable alors $\forall x A$ = vrai si et seulement si $\mathcal{I}(A(x/d)) = vrai$ pour tout élément d de $\mathcal{D}$;*
- *Si A est une formule et x une variable alors $\exists x A$ = vrai si et seulement si $\mathcal{I}(A(x/d)) = vrai$ pour au moins un élément d de $\mathcal{D}$.*

Comme pour la logique propositionnelle, la relation de conséquence entre deux formules peut se comprendre en termes d'inclusion des interprétations qui satisfont les formules.

Définition 2.1.11 (Conséquence).
Soient A et B deux formules de la logique du premier ordre. B est une conséquence *de A et on écrit $A \models B$ si et seulement si pour toute interprétation $\mathcal{I}$ vérifiant $\mathcal{I}(A) = vrai$ on a $\mathcal{I}(B) = vrai$.*

Le fait qu'une formule est la conséquence d'un ensemble de formules est défini par :

Définition 2.1.12.
Soient $\mathcal{F}$ un ensemble de formules de la logique du premier ordre et B une formule quelconque de la logique du premier ordre. B est une conséquence *de $\mathcal{F}$ on écrit $\mathcal{F} \models B$ si et seulement si pour toute formule A de $\mathcal{F}$ on a $A \models B$.*

Comme pour la logique propositionnelle, on dira d'une formule vraie pour toute interprétation qu'elle est *valide* (ou qu'elle est une *tautologie*). Une formule fausse pour toute interprétation est dite *insatisfaisable* ou *incohérente*. S'il existe une interprétation qui satisfait une formule, on dit qu'elle est *satisfaisable* ou *cohérente*. Une interprétation qui satisfait une formule A (ou toutes les formules d'un ensemble de formules A) est appelée *modèle* de A. À l'inverse, une interprétation qui ne satisfait pas une formule A (ou toutes les formules d'un ensemble de formules A) est appelée *contre-modèle* de A.

2.2 Logiques non monotones

Les logiques non monotones ont pour but la mise au point de systèmes d'inférence permettant la modélisation de raisonnements révisables et donc non valides au sens classique du terme. D'un point de vue sémantique, ceci revient à inférer des formules qui sont consistantes avec les prémisses en ce sens qu'elles sont vérifiées dans au moins un modèle de celles-ci.

Par exemple, la conclusion *Titi vole* n'est pas une conséquence de l'ensemble des deux prémisses *La plupart des oiseaux volent* et *Titi est un oiseau*. Elle est simplement une formule consistante avec cet ensemble de

prémisses ; elle appartient donc à une image du monde qu'il est possible de se donner de façon consistante sur la base de ces deux prémisses.

2.2.1 Logique préférentielle

On va présenter, dans cette section, la logique préférentielle qui a été initiée par Bossu-Siegel [Bossu and Siegel, 1982, Bossu and Siegel, 1985], reprise ensuite par Shoam [SHOAM, 1987] et Besnard-Siegel [Besnard and Siegel, 1988], puis par Kraus, Lehmann et Magidor dans [Kraus et al., 1990]. Toutes ces approches sont construites sur une *logique classique* (calcul propositionnel, calcul des prédicats, logique modale), où la sémantique de l'inférence a été donnée par « une formule A implique une formule B si *chaque* modèle de A est un modèle de B » (voir la définition 1.2.18 page 9). Cependant, une approche préférentielle, dans sa forme la plus générale, dit « A implique B si tous les modèles préférés de A sont des modèles de B ». Les modèles préférés de A sont des modèles qui ont des propriétés utiles pour la gestion des exceptions.

Ce concept de préférence peut être défini par une relation $\prec$ de *préordre* entre les modèles de la base de connaissance. Rappelons qu'une relation de préordre est une relation réflexive et transitive. Les modèles préférés étant les modèles minimaux pour cette relation. Soient M et N deux modèles de la base de connaissance, $M \prec N$ signifie que le modèle M est plus important que le modèle N. Insistons sur le fait que si un modèle M est « inférieur » à un modèle N et vice versa, alors M et N ne sont pas nécessairement identiques.

Définition 2.2.1 (Relation préférentielle).
Une relation préférentielle $\prec$ *est une relation de préordre sur l'ensemble des modèles de la base de connaissance. D'ailleurs, si la relation $\prec$ est antisymétrique, alors $\prec$ devient un ordre.*

Intuitivement, on peut considérer les étudiants qui ne sont pas jeunes comme des exceptions (étudiants anormaux). Par conséquent, si M et N sont deux modèle de cette base de connaissance, alors $M \prec N$ si l'ensemble des exceptions de M (c'est-à-dire l'ensemble des étudiants anormaux dans M) est inclus dans l'ensemble des exceptions de N.

Définition 2.2.2 (Modèle minimal).
Soit $\mathscr{L}$ une logique classique, $\mathscr{F}$ l'ensemble des formules de $\mathscr{L}$ et $\mathcal{A}$ un sous-ensemble de $\mathscr{F}$. Un modèle minimal M *de $\mathcal{A}$ est une interprétation qui satisfait $\mathcal{A}$ et qui est minimale par rapport à la relation préférentielle $\prec$ définie sur l'ensemble des modèles de $\mathcal{A}$. Autrement dit, si M' est un modèle de $\mathcal{A}$ tel que $M' \prec M$, alors $M \prec M'$ (ou, de manière équivalente, $M' = M$ si la relation préférentielle $\prec$ est antisymétrique).*

Définition 2.2.3 (Inférence Préférentiel).
De façon classique, si A et B sont deux ensembles de formules d'une logique classique et $\prec$ est une relation préférentielle, on définit l'inférence logique de modèle préférentiel $\underset{\prec}{\models}$ *comme suit : $A \underset{\prec}{\models} B$ si et seulement si chaque modèle minimal de A est un modèle de B.*

Nous pouvons trouver dans [Siegel et al., 2001] la proposition suivante :

Proposition 2.2.1 (Existence d'un modèle minimal)**.**
Si un langage $\mathcal{L}$ a un ensemble fini de variables, alors chacune de ses formules consistantes F a au moins un modèle minimal, et pour chaque modèle de M de F, il existe un modèle minimal M' tel que $M' \prec M$.

Remarque 2.2.1. La logique propositionnelle satisfait les conditions de la Proposition 2.2.1. Alors, chaque formule propositionnelle cohérente F admet au moins un modèle minimal, et pour chaque modèle M de F, il existe un modèle minimal M' de F tel que $M' \prec M$.

Exemple 2.2.1.
Pour représenter les informations suivantes : « En général les étudiants sont jeunes » *et* « Léa est un étudiant »*, nous pouvons utiliser une approche préférentielle, qui est fermée pour la circonscription [McCarthy,]. Un prédicat supplémentaire* anormal *est ajouté.*

Notre première information est traduite en « Un étudiant qui n'est pas anormal est jeune ». Maintenant, si les constantes prédicatives Étudiant*, Anormal, et Jeune indiquent respectivement : « étudiant », « anormal » et « jeune », alors, dans la logique du premier ordre, on obtient l'ensemble suivant des formules :*

$$\mathcal{A} \equiv \left\{ \text{É}tudiant(Léa), \left(\forall x \left(\left(\text{É}tudiant(x) \wedge \neg Anormal(x) \right) \implies Jeune(x) \right) \right) \right\}$$

Par la substitution $(x/Léa)$ (voir la définition 2.1.8 page 29), nous traduisons l'ensemble des formules $\mathcal{A}$ en logique propositionnelle et on obtient l'ensemble suivant de formules :

$$\mathcal{A}(x/Léa) \equiv \left\{ \text{É}tudiant(Léa), \left(\left(\text{É}tudiant(Léa) \wedge \neg Anormal(Léa) \right) \implies Jeune(Léa) \right) \right\}$$

En d'autres termes :

$$\mathcal{A}(x/Léa) \equiv \left\{ \text{É}tudiant(Léa), \left(Anormal(Léa) \vee Jeune(Léa) \right) \right\}$$

L'ensemble $\mathcal{A}(x/Léa)$ a huit interprétations, parmi elles, les trois suivantes

sont des modèles :

$$\begin{aligned} M_1 &= \Big\{\text{É}tudiant(Léa),\quad Anormal(Léa),\quad Jeune(Léa)\Big\} \\ M_2 &= \Big\{\text{É}tudiant(Léa), \neg Anormal(Léa),\quad Jeune(Léa)\Big\} \\ M_3 &= \Big\{\text{É}tudiant(Léa),\quad Anormal(Léa), \neg Jeune(Léa)\Big\} \end{aligned}$$

Dans une logique classique, il est impossible de déduire de $\mathcal{A}(x/Léa)$ que *Léa* est jeune (c'est-à-dire $Jeune(Léa)$). En effet, $\mathcal{A}(x/Léa)$ a deux modèles dans lesquels *Léa* est jeune (M_1 et M_2), et des modèles dans lesquels *Léa* n'est pas jeune, en particulier ceux où *Léa* est anormal (M_3). En d'autre termes $\mathscr{M}\Big(\mathcal{A}(x/Léa)\Big) \nsubseteq \mathscr{M}\Big(\{Jeune(Léa)\}\Big)$ (voir la définition 1.2.18 page 9).

Pour obtenir le résultat « Léa est jeune », on préférera les modèles qui ont moins d'étudiants anormaux. Ainsi, dans une approche de modèle préférentiel, la relation préférentielle $\prec$ peut être définie comme : $M \prec M'$ si et seulement si « chaque individu qui est anormal dans M est anormal dans M' ». Selon cette relation, nous obtenons les préférences suivantes entre les modèles de $\mathcal{A}(x/Léa)$:

- $M_1 \prec M_3$;
- $M_3 \prec M_1$;
- $M_2 \prec M_1$;
- $M_2 \prec M_3$.

Par conséquent, $\mathcal{A}(x/Léa)$ n'a qu'un seul modèle minimal qui est M_2, et *Léa* est jeune dans ce modèle. On peut donc en déduire que *Léa* est jeune dans cette approche préférentielle.

2.2.2 $X-$logique

On va présenter, dans cette section, la $X-$logique, qui a été introduite par Siegel et Forget dans [Lionel and Siegel, 1996].

La $X-$logique se présente comme une tentative de constitution d'une théorie de la preuve pour les logiques non monotones, à partir de la logique propositionnelle et d'un ensemble X de formules.

Remarque 2.2.2 (Inférence classique). En logique classique, une formule f est un *théorème* de (ou *impliquée* par) A si et seulement si son ajout à A ne change pas l'ensemble des conclusions dérivées de A, autrement dit :

$$A \vdash f \quad \text{si et seulement si} \quad \overline{A \cup \{f\}} = \overline{A}$$

où la barre au dessus d'un ensemble dénote les théorèmes associés à (ou les formules impliquées de) l'ensemble en question (voir la définition 2.1.1 page 27).

La $X-$logique est une *généralisation* (donc un affaiblissement) de la relation de conséquence classique $\vdash$:

Définition 2.2.4 ($X-$inférence).
Soit X un ensemble de formules de la logique classique $\mathscr{L}$ (X n'est pas nécessairement déductivement fermé). La relation d'inférence non monotone $\underset{X}{\vdash}$ est définie par :

$$A \underset{X}{\vdash} f \quad \textit{si et seulement si} \quad \overline{(A \cup \{f\})} \cap X \subseteq \overline{A}$$

Ainsi, $A \underset{X}{\vdash} f$ si et seulement si tout théorème classique de $A \cup \{f\}$ qui est dans X, est un théorème de A. Cela veut dire, l'ajout de f à A ne fait pas croître l'ensemble des théorèmes classique qui sont dans X.

La propriété suivante correspond à la définition originale de la $X-$logique telle qu'elle a été introduite dans [Lionel and Siegel, 1996]. Elle est équivalente à la définition 2.2.4.

Propriété 2.2.1 ($X-$inférence [Lionel and Siegel, 1996]).

$$A \underset{X}{\vdash} f \quad \text{si et seulement si} \quad \overline{(A \cup \{f\})} \cap X = \overline{A} \cap X$$

Démonstration. De la définition 2.2.4, $A \underset{X}{\vdash} f$ si et seulement si $\overline{(A \cup \{f\})} \cap X$ est inclus dans $\overline{A}$. Comme $\overline{A \cup \{f\}} \cap X$ est inclus aussi dans X, alors $\overline{(A \cup \{f\})} \cap X$ est inclus dans $\overline{A} \cap X$. Par la monotonie de l'inférence classique, $\overline{A} \cap X$ est inclus dans $\overline{(A \cup \{f\})} \cap X$.

Ainsi, $\overline{(A \cup \{f\})} \cap X \subseteq \overline{A}$ si et seulement si $\overline{(A \cup \{f\})} \cap X = \overline{A} \cap X$, et donc, $A \underset{X}{\vdash} f$ si et seulement si $\overline{(A \cup \{f\})} \cap X = \overline{A} \cap X$ □

La $X-$inférence n'est pas une inférence monotone. Si $A \underset{X}{\vdash} C$, il est possible d'avoir $A \cup B \underset{X}{\nvdash} C$.

Exemple 2.2.2.
Soit $X = \{c\}$. On a $\{a\} \underset{X}{\vdash} \{b \Longrightarrow c\}$, mais $\{a, b\} \underset{X}{\nvdash} \{b \Longrightarrow c\}$. En fait, d'une part, on a $\overline{\{a, b \Longrightarrow c\}} \cap \{c\} = \overline{\{a\}} \cap \{c\} = \varnothing$, et d'autre part, $\overline{\{a, b, b \Longrightarrow c\}} \cap \{c\} = \{c\}$ et $\overline{\{a, b\}} \cap \{c\} = \varnothing$

Propriété 2.2.2. D'après la définition de la $X-$inférence et la monotonie de l'inférence classique, on a les propriétés suivantes :

Supraclassicalité : Si $A \vdash B$, alors $A \underset{X}{\vdash} B$. En fait, si $A \vdash B$, alors $\overline{A \cup B} = \overline{A}$. Par conséquent, $\overline{(A \cup B)} \cap X = \overline{A} \cap X$.

Si $X \supseteq B$: alors $A \underset{X}{\vdash} B$ si et seulement si $A \vdash B$. En utilisant la propriété précédente, il suffit de montrer que, si $A \underset{X}{\vdash} B$, alors $A \vdash B$. D'après

la propriété 2.2.1 et comme $B \subseteq X$, on a donc $B \subseteq \left((\overline{A \cup B}) \cap X\right)$, d'où $B \subseteq (\overline{A} \cap X)$. Ainsi, $B \subseteq \overline{A}$. Ce qui permet de conclure que $\underset{X}{\vdash}$ est monotone sur X.

Si $\mathbf{X} = \mathscr{F}$: l'inférence $\underset{X}{\vdash}$ est identique à l'inférence classique $\vdash$, où $\mathscr{F}$ est l'ensemble de toutes les formules possibles de la logique $\mathscr{L}$. En fait, on peut considérer ce cas comme un cas particulier du cas précédent.

Si $\mathbf{X} \subseteq \overline{\mathbf{A}}$: alors $A \underset{X}{\vdash} f$ pour toute formule f. En fait, il suffit de remarquer que $X \supseteq (\overline{A \cup \{f\}}) \cap X \supseteq \overline{A} \cap X = X$ pour toute formule f. En particulier, $A \underset{\varnothing}{\vdash} f$ pour toute formule f.

Si $\mathbf{X} = \{\bot\}$ et A un ensemble de formules consistantes, alors $A \underset{X}{\vdash} f$ est équivalent à $A \nvdash \neg f$. En fait, d'après la propriété 2.2.1 (page 34), on voit que $A \underset{X}{\vdash} f$ est équivalent à $(A \cup \{f\}) \nvdash \bot$. En fait, on a dans ce cas $(\overline{A \cup \{f\}}) \cap X = \varnothing$, d'où $\bot \notin (\overline{A \cup \{f\}})$. Ce cas décrit la relation de consistance entre A et f.

Propriété 2.2.3 ($X-$inférence [S.Sadok, 2000]).

$$A \underset{X}{\vdash} f \quad \text{si et seulement si} \quad \left(A \cup \{f\} \nvdash x \quad \text{pour tout} \quad x \in X - \overline{A}\right)$$

Démonstration. Si $X \subseteq \overline{A}$ la propriété est vraie, puisque $X - \overline{A} = \varnothing$ et $A \underset{X}{\vdash} f$ est vraie pour toute formule f. Maintenant, soit $x \in X - \overline{A}$.

Supposons $A \underset{X}{\vdash} f$, alors $(\overline{A \cup \{f\}}) \cap X = \overline{A} \cap X$. Puisque $x \in X - \overline{A}$, alors $x \notin \overline{A} \cap X$, c'est-à-dire, $x \notin (\overline{A \cup \{f\}})$. Ainsi $A \cup \{f\} \nvdash x$.

Enfin, supposons que $A \cup \{f\} \nvdash x$. remarquons que $\left((\overline{A \cup \{f\}}) \cap X\right) \subseteq X$ et $X = \left((X - \overline{A}) \cup (X \cap \overline{A})\right)$. Comme $A \cup \{f\} \nvdash x$ pour tout $x \in X - \overline{A}$, alors $x \notin (\overline{A \cup \{f\}})$, d'où $x \notin \left((\overline{A \cup \{f\}}) \cap X\right)$. Ainsi $\left((\overline{A \cup \{f\}}) \cap X\right) \subseteq (X \cap \overline{A})$, c'est-à-dire $A \underset{X}{\vdash} f$. □

Intuitivement, l'ensemble de formules X permet de contraindre notre mode de raisonnement : un ensemble d'informations A implique un ensemble d'informations B, pour la $X-$inférence, si l'ajout de B à A ne produit pas plus de formules de X qu'avec A seulement. En d'autres termes, l'ensemble X peut être considéré comme un potentiomètre qui règle l'inférence : si A encode un ensemble d'informations (des connaissances, ou certaines croyances ...), X peut être considéré comme l'ensemble des informations « pertinentes ».

2.2.3 Logique des défauts

2.2.3.1 Principe et définition

La logique des défauts a été introduite et développée par Reiter [Reiter, 1980] pour formaliser le raisonnement simplement consistant. La logique des défauts est une approche de la non monotonie couramment utilisée comme référence dans le domaine des logiques non monotones.

En présence d'une information incomplète, nous sommes amenés à tirer des conclusions conjecturales simplement plausibles. En particulier, nous interprétons parfois comme absolument générales des lois qui apparaissent correctes dans la plupart des cas, mais qui admettent certaines exceptions.

Si Titi est un oiseau, alors j'en infère que Titi vole. Même s'il y a des oiseaux qui ne volent pas, je me sens autorisé à conclure que Titi vole s'il n'existe rien dans « mes croyances » qui m'interdise de tirer cette conclusion. Comme le fait que Titi vole est consistant avec mes croyances, je conclus que Titi vole, parce que cette conclusion est la plus naturelle pour moi. Ce genre de raisonnement est appelé *raisonnement par défaut*.

La logique des défauts augmente les règles d'inférences de la logique classique par un type particulier de règles d'inférence, appelées *défauts*, qui permettent de capter un type de raisonnement couramment utilisé par les humains et qui peut s'exprimer intuitivement comme suit :

$$\frac{A \ : \ B}{C}$$

Si A est cru et si B est consistant avec tout ce qui est cru, alors C peut être cru également. Il s'agit là d'une inférence par défaut, la conclusion C n'est pas valide au sens strict du terme mais seulement « plausible » dans la situation en cours : la présence d'éléments supplémentaires dans cette même situation peut remettre en cause cette conclusion en inhibant l'application du défaut, d'où le caractère non monotone de la logique des défauts.

Ainsi, on peut exprimer la loi énonçant que les oiseaux généralement volent de la manière que voici :

$$\frac{Oiseau(x) \ : \ Vole(x)}{Vole(x)}$$

L'interprétation intuitive que l'on peut associer à cette règle est : « si x est un oiseau et s'il est consistant de supposer que x vole, alors on peut inférer que x vole ».

Définition 2.2.5 (Règle de défaut).
Soit LPr un langage prédicatif du premier ordre (voir la définition 2.1.2 page 28). Une règle de défaut *(en abrégé* défaut*)* $\mathcal{D}$ *est une expression de la forme :*

$$\frac{\alpha(x) \ : \ \beta_1(x), \ldots, \beta_m(x)}{\gamma(x)}$$

où

- $\alpha(x), \beta_1(x), \ldots, \beta_m(x), \gamma(x)$ *sont des formules de LPr dont les variables libres sont choisies parmi* $\{x_1, \ldots, x_n\}$
- $\alpha(x)$ *est appelé* prérequis *du défaut* $\mathcal{D}$*,*
  $\beta_i(x)$*, avec* $i \in \{1, \ldots, m\}$*, sont appelés* justifications *du défaut* $\mathcal{D}$*,*
  $\gamma(x)$ *est appelé* conséquent *du défaut* $\mathcal{D}$*.*

Intuitivement, on peut lire ce défaut comme suit : « si $\alpha(x)$ est prouvé vrai, et si aucun des $\neg\beta_i(x)$ n'appartient à l'ensemble des conclusions de nos croyances actuelles, alors ajouter $\gamma(x)$ à cet ensemble de conclusions »

Un défaut $\mathcal{D}$ est dit *fermé* si et seulement si $\alpha(x), \beta_1(x), \ldots, \beta_m(x)$ et $\gamma(x)$ ne contiennent pas de variable libre. Dans ce cas $\alpha(x), \beta_1(x), \ldots, \beta_m(x)$ et $\gamma(x)$ sont notés plus simplement $\alpha, \beta_1, \ldots, \beta_m$ et γ.

Les variables libres d'un défaut sont interprétées comme universellement quantifiées, leur portée s'étendant sur les trois membres du défaut. Un défaut qui n'est pas fermé est appelé *ouvert*. Un défaut ouvert représente un « schéma général d'inférence ». Une instance d'un défaut ouvert est un défaut fermé obtenu par remplacement de toutes les variables libres du défaut ouvert par des constantes de *LPr* (en respectant la loi implicite de portée des variables libres du défaut).

Définition 2.2.6 (Théorie des défauts).
Une théorie avec règles de défaut *(en abrégé* théorie des défauts *ou bien* une théorie avec défauts*)* $\mathscr{T}$ *est une paire* $\langle \mathscr{D}, \mathscr{W} \rangle$ *où :*

- $\mathscr{D}$ *est un ensemble de défauts ;*
- $\mathscr{W}$ *(World) est un ensemble de formules fermées de LPr appelé* la théorie de base*. Il décrit l'état du monde modélisé et formalise les faits qui sont connus avec certitude.*

Une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ est dite *fermée* si et seulement si tous les défauts de $\mathscr{D}$ sont fermés.

Exemple 2.2.3.
Voyons tout d'abord un exemple de représentation d'une connaissance incomplète ; cet exemple est repris de [Reiter, 1980]. La connaissance est donnée sous la forme de deux règles de défauts dont nous indiquons un énoncé en français et sa traduction formelle.

- **Français :** *« Une personne habite généralement avec son conjoint. »*
  Traduction formelle : $\dfrac{Conjoint(x,y) \wedge (Habite(y) = z) \ : \ (Habite(x) = z)}{Habite(x) = z}$
- **Français :** *« Une personne habite généralement dans la ville de son employeur. »*

Traduction formelle : $\dfrac{Employeur(x,y) \wedge (Ville(y) = z) \ : \ (Habite(x) = z)}{Habite(x) = z}$

Supposons que le conjoint de Marie *habite* Bruxelles *tandis que son employeur est installé à* Paris. *Les deux règles d'inférence induisent des conclusions mutuellement inconsistantes. Si l'on applique d'abord la première règle, il faut empêcher toute inférence qui amènerait la conclusion : «* Marie *habite* Paris *», c'est-à-dire, cela empêcherait l'application de la deuxième règle. On dira que «* Marie *habite* Bruxelles *» et «* Marie *habite* Paris *» sont des formules appartenant à deux* extensions *« mutuellement inconsistantes » de la théorie ayant les deux règles de défaut énoncées ci-dessus.*

2.2.3.2 Extensions d'une théorie des défauts

Une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ sous-tend un certain nombre d'ensembles de croyances (zéro, un ou plusieurs ensembles) que l'on peut inférer de façon consistante avec l'ensemble $\mathscr{W}$ de formules. Ces ensembles de croyances sont appelés *extensions de la théorie des défauts.*

Les extensions d'une théorie avec défauts ne sont explicitement définies ici que pour les théories fermées. Les théories ouvertes avec défauts peuvent être transformées en des théories fermées correspondantes. Les résultats obtenus pour les théories fermées avec défauts peuvent être étendus aux théories ouvertes avec défauts lorsque ces théories sont finies.

Une extension d'une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ est un ensemble de formules déductivement fermé $\mathcal{E}$ contenant $\mathscr{W}$ et qui vérifie : si $\dfrac{\alpha \ : \ \beta_1, \ldots, \beta_m}{\gamma} \in \mathscr{D}$ est un défaut tel que $\alpha \in \mathcal{E}$ et $\forall i \in \{1, \ldots, m\}, \neg\beta_i \notin \mathcal{E}$, alors $\gamma \in \mathcal{E}$. Formellement :

Définition 2.2.7 (Extension).
Soit $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ une théorie fermée des défauts. Étant donné un sous-ensemble S de LPr, désignons par $\Gamma(S)$ le plus petit sous-ensemble de LPr, satisfaisant les trois conditions :

- $\mathscr{W} \subseteq \Gamma(S)$,
- $Th(\Gamma(S)) = \Gamma(S)$,
- *si* $\dfrac{\alpha \ : \ \beta_1, \ldots, \beta_m}{\gamma} \in \mathscr{D}$, *si* $\alpha \in \Gamma(S)$ *et si* $\forall i \in \{1, \ldots, m\}, \neg\beta_i \notin S$, *alors* $\gamma \in \Gamma(S)$

Un ensemble de formules $\mathcal{E}$ de LPr est une extension *pour $\mathscr{T}$ si et seulement si $\Gamma(\mathcal{E}) = \mathcal{E}$, c'est-à-dire, si et seulement si $\mathcal{E}$ est un point fixe de l'opérateur Γ.*

Une telle extension $\mathcal{E}$ peut être caractérisée de la manière pseudo-itération suivante :

Définition 2.2.8 (Extension (méthode par pseudo-itération)).
Soient $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ une théorie des défauts et $\mathcal{E}$ un ensemble de formules. Construisons une suite de formules $(\mathcal{E}_i)_{i\in\mathbb{N}}$ en posant $\mathcal{E}_0 = \mathscr{W}$ et

$$\mathcal{E}_{i+1} = Th(\mathcal{E}_i) \cup \left\{ \gamma \;\middle|\; \frac{\alpha \;:\; \beta_1, \ldots, \beta_n}{\gamma} \in \mathcal{D},\ \text{où } \alpha \in \mathcal{E}_i \text{ et } \forall j \in \{1, \ldots, n\}, \neg\beta_j \notin \mathcal{E} \right\}$$

pour $i \in \mathbb{N}$. Alors l'ensemble donné $\mathcal{E}$ est une extension pour $\mathscr{T}$ si et seulement si :

$$\mathcal{E} = \bigcup_{i\in\mathbb{N}} \mathcal{E}_i$$

Une règle de défaut $\dfrac{\alpha \;:\; \beta_1, \ldots, \beta_n}{\gamma}$ peut être appliquée à une théorie donnée des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ si son prérequis α est dans $\mathscr{W}$ (dans la première étape, et après dans $Th(\mathcal{E}_i)$), et la négation de chacun de ses justifications $\neg\beta_j$ n'est pas dans l'extension $\mathcal{E}$.

On note particulièrement le caractère « non constructif » du calcul des extensions d'une théorie des défauts. Ce qui rend la logique des défauts en toute généralité indécidable, c'est-à-dire, il n'existe pas d'algorithme qui, pour toute formule, décide au bout d'un temps fini si cette formule appartient ou non à une extension.

Remarque 2.2.3. Les règles de défaut peuvent être appliquées dans un ordre différent et cela peut conduire à différentes extensions $\mathcal{E}$ pour la même théorie $\mathscr{T}$.

Une théorie des défauts peut parfois permettre d'inférer plusieurs extensions à partir d'un même ensemble de prémisses.

Exemple 2.2.4 (Théorie ayant une extension ([McDermott and Doyle, 1980])).
Soit $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ une théorie des défauts où $\mathscr{D} = \left\{ \dfrac{:\ A}{\neg P}, \dfrac{:\ P}{\neg Q}, \dfrac{:\ Q}{\neg S} \right\}$ et $\mathscr{W} = \varnothing$. Cette théorie possède une extension : $\mathcal{E} = Th(\{\neg P, \neg S\})$.

Exemple 2.2.5 (Théorie n'ayant pas d'extension).
Soit $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ une théorie des défauts où $\mathscr{D} = \left\{ \dfrac{:\ A}{\neg A} \right\}$ et $\mathscr{W} = \varnothing$. Cette théorie n'a pas d'extension.

Exemple 2.2.6 (Théorie ayant deux extensions ([Reiter, 1980])).
Soit $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ une théorie des défauts où la théorie de base est $\mathscr{W} = \varnothing$ et $\mathscr{D} = \left\{ \dfrac{A \;:\; \exists x P(x)}{\exists x P(x)}, \dfrac{:\ A}{A}, \dfrac{:\ \neg A}{\neg A} \right\}$. Cette théorie possède deux extensions : $\mathcal{E}_1 = Th(\{\neg A\})$ et $\mathcal{E}_2 = Th(\{A, \exists x P(x)\})$.

2.2.3.3 Théories normales et semi-monotonie

Dans [Reiter and Criscuolo, 1981] Reiter et Criscuolo étudient les théories normales.

Définition 2.2.9 (Défaut normal).
Un défaut normal *est un défaut avec une seule justification équivalente à la conclusion. Un défaut normal est donc de la forme :* $\dfrac{\alpha \ : \ \beta}{\beta}$

Une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ est dite *normale* si tous les défauts de $\mathscr{D}$ sont normaux. Une des propriétés intéressantes des théories normales est la « semi-monotonie ».

Théorème 2.2.1 (Semi-monotonie)**.**
Soient $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ *et* $\mathscr{T}' = \langle \mathscr{D}', \mathscr{W} \rangle$ *deux théories normales des défauts telles que* $\mathscr{D} \subseteq \mathscr{D}'$*. Alors, chaque extension de* $\mathscr{T}$ *est incluse dans une extension de* $\mathscr{T}'$*.*

Ce théorème permet d'affirmer une autre propriété intéressante : « toute théorie normale des défauts possède au moins une extension ». En fait, la théorie normale des défauts $\mathscr{T} = \langle \varnothing, \mathscr{W} \rangle$ dont l'ensemble des défauts est vide a comme seule extension $Th(\mathscr{W})$. Pour toute théorie normale $\mathscr{T}' = \langle \mathscr{D}', \mathscr{W} \rangle$, on a $\varnothing \subseteq \mathscr{D}'$. Donc par le théorème de semi-monotonie, on déduit que $\mathscr{T}'$ a une extension $\mathcal{E}$ telle que $Th(\mathscr{W}) \subseteq \mathcal{E}$.

Cela a permis à Reiter de définir une notion de preuve en logique des défauts concernant les théories normales des défauts [Reiter, 1980].

Propriété 2.2.4. Lorsque la théorie $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ est une théorie normale des défauts, la contrainte sur les justifications β_j dans la définition 2.2.8 (page 38), c'est-à-dire $\neg\beta_j \notin \mathcal{E}$, est simplifiée en $\neg\beta_j \notin \mathcal{E}_i$. En d'autres termes, la suite de formules $(\mathcal{E}_i)_{i\in\mathbb{N}}$ est défini en posant $\mathcal{E}_0 = \mathscr{W}$ et

$$\mathcal{E}_{i+1} = Th(\mathcal{E}_i) \cup \left\{ \gamma \;\middle|\; \frac{\alpha \ : \ \beta_1, \ldots, \beta_n}{\gamma} \in \mathcal{D}, \text{ où } \alpha \in \mathcal{E}_i \text{ et } \forall j \in \{1, \ldots, n\}, \neg\beta_j \notin \mathcal{E}_i \right\}$$

pour $i \in \mathbb{N}$.

Nous obtenons, alors, une méthode plus simple et constructive pour le calcul de l'extension $\mathcal{E}$.

Exemple 2.2.7.
Prenons par exemple la règle de défaut « généralement, les oiseaux volent *» qui est formalisée par le défaut suivant :* $\mathcal{D} = \dfrac{Oiseau(x) \ : \ Vole(x)}{Vole(x)}$*. Cette règle signifie que, si* x *est un oiseau et si l'on peut supposer qu'il vole, alors nous pouvons conclure qu'il vole. Une théorie de base contenant des faits sur*

les oiseaux est la suivante :

$$\mathscr{W} = \{Oiseau(Condor), Oiseau(Manchot), \neg Vole(Manchot), Vole(Aigle)\}$$

On obtient alors la théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ *où* $\mathscr{D} = \{\mathcal{D}\}$.

Selon cette règle de défaut, un Condor vole parce que $Oiseau(Condor)$ *est vraie et la justification* $Vole(Condor)$ *n'est pas incompatible avec ce qui est actuellement connu. Au contraire, nous savons que* $\Big(\neg Vole(Manchot)\Big) \in \mathscr{W}$, *c'est-à-dire un Manchot ne vole pas. Ainsi,* $Oiseau(Manchot)$ *ne permet pas de conclure* $Vole(Manchot)$*, même si le prérequis* $Oiseau(Manchot)$ *est vraie, car la justification* $Vole(Manchot)$ *est incompatible avec ce qui est connu.*

Par conséquent, nous obtenons une extension unique pour la théorie $\mathscr{T}$ *qui est la suivante :*
$\mathcal{E} = Th\Big(\{Oiseau(Condor), Oiseau(Manchot), \neg Vole(Manchot), Vole(Aigle), Vole(Condor)\}\Big)$

2.2.3.4 Inférence en logique des défauts

La notion la plus importante dans une logique des défauts est de « calculer des extensions d'une théorie des défauts ». Selon cet ensemble d'extensions, les chercheurs ont défini des sémantiques différentes pour l'inférence en logique des défauts. L'implication d'une formule par une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ peut être défini de différentes façons :

Définition 2.2.10.
Étant donnée une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ *et l'ensemble de tous ses extensions* $\mathscr{E}_{\mathscr{T}}$ *(ou bien* $\mathscr{E}$ *s'il n'y a pas d'ambiguïté).*

- **Approche sceptique** *: une formule* f *est* impliquée au sens sceptique *par la théorie des défauts* $\mathscr{T}$ *si elle est impliquée par toutes ses extensions, c'est-à-dire :* $\mathscr{T} \underset{S}{\vdash} f$ *si et seulement si* $\forall \mathcal{E} \in \mathscr{E}$, $\mathcal{E} \vdash f$.
- **Approche crédule** *: une formule* f *est* impliquée au sens crédule *par la théorie des défauts* $\mathscr{T}$ *si elle est impliquée au moins par une de ses extensions, c'est-à-dire :* $\mathscr{T} \underset{C}{\vdash} f$ *si et seulement si* $\exists \mathcal{E} \in \mathscr{E}$ *tel que* $\mathcal{E} \vdash f$.
- **Approche semi-crédule** *: une formule* f *est* impliquée au sens semi-crédule *par la théorie des défauts* $\mathscr{T}$ *si elle est impliquée au moins par une de ses extensions et toutes ses extensions n'impliquent pas sa négation, c'est-à-dire :* $\mathscr{T} \underset{SC}{\vdash} f$ *si et seulement si* $\exists \mathcal{E} \in \mathscr{E}$ *tel que* $\mathcal{E} \vdash f$ *et* $\forall \mathcal{E} \in \mathscr{E}$, $\mathcal{E} \nvdash \neg f$.

Chapitre 3

Structure de groupes et groupes symétriques

Sommaire

3.1 Introduction 43
3.2 Notion de groupe 43
3.2.1 Loi de composition interne 43
3.2.2 Groupe 44
3.2.3 Sous-groupe 46
3.3 Groupes symétriques S_n 48
3.3.1 Notion de $\sigma-$orbite 48
3.3.2 Cycles dans S_n 49

3.1 Introduction

Les groupes symétriques sont les structures de base dans tous les travaux sur les symétries. C'est pour cette raison que nous allons présenter dans ce chapitre la notion de groupe symétrique. Mais avant cette présentation, on préfère faire des rappels sur la notion de groupe et de sous-groupe.

Pour plus de détails sur la théorie des groupes et des groupes symétriques, le lecteur peut se référer au livre de Calais [Calais, 1984].

3.2 Notion de groupe

3.2.1 Loi de composition interne

Définition 3.2.1.
Étant donné un ensemble E, on appelle loi de composition interne *sur E toute*

application de $E \times E$ dans E, où $E \times E = \{(x, y) \mid x \in E \text{ et } y \in E\}$.

Supposons que $E \neq \varnothing$ et désignons par $(E, \cdot)$ l'ensemble E muni de la loi de composition définie par l'application : $(x, y) \mapsto x \cdot y$

Définitions 3.2.2.

- *Dans E, la loi $\cdot$ est dite :*

  associative *si, $(x \cdot y) \cdot z = x \cdot (y \cdot z)$, quels que soient x, y, z dans E ;*

  commutative *si, $x \cdot y = y \cdot x$, quels que soient x, y dans E.*
- *S'il existe $e \in E$ tel que, quel que soit $x \in E$, $x \cdot e = e \cdot x = x$, on dit que e est* élément neutre *dans $(E, \cdot)$.*
- *Si $(E, \cdot)$ possède un élément neutre e, alors un élément $x \in E$ est dit* symétrisable, *s'il existe $x' \in E$ tels que : $x' \cdot x = x \cdot x' = e$; x' est alors appelé* symétrique *de x dans $(E, \cdot)$.*

Exemple 3.2.1.
Dans $\mathbb{N}$ et dans $\mathbb{Z}$ les opérations habituelles d'addition et de multiplication sont des lois de composition internes associatives et commutatives ; 0 et 1 sont, respectivement, élément neutre pour l'addition et pour la multiplication.

3.2.2 Groupe

Définition 3.2.3.
Soit G un ensemble non vide, muni d'une loi de composition interne définie par : $(x, y) \mapsto x \cdot y$. On dit que la loi « $\cdot$ » définit sur G une structure de groupe, *ou que G est un* groupe *relativement à cette loi, si les trois axiomes suivants sont vérifiés :*

($\mathbf{G}_1$) : la loi « $\cdot$ » est associative ;

($\mathbf{G}_2$) : il existe dans $(G, \cdot)$ un élément neutre e ;

($\mathbf{G}_3$) : tout élément de $(G, \cdot)$ est symétrisable.

Un groupe G est dit *abélien*[1](ou *commutatif*) si la loi de composition interne de G est commutative.

Lorsque le contexte le permet, nous dirons G est un groupe au lieu de dire $(G, \cdot)$ est un groupe.

Définition 3.2.4 (Ordre d'un groupe).
Un groupe G est dit fini *s'il n'a qu'un nombre fini d'éléments. Dans ce cas, le cardinal de G s'appelle l'*ordre *du groupe G ; il est noté $o(G)$ (ou $|G|$).*

Remarque 3.2.1. Par analogie avec les notations utilisées dans les ensembles de nombres pour les opérations habituelles de multiplication et

1. Du nom du mathématicien norvégien **N. H. Abel** (1802 – 1829).

d'addition, la loi de composition interne d'un groupe G sera couramment notée :

multiplicativement : $(x, y) \mapsto x \cdot y$, ou

additivement : $(x, y) \mapsto x + y$.

Dans le premier cas, $x \cdot y$ s'appelle le *produit* de x et y pris dans cet ordre ; l'élément neutre se note, en général, e ou 1 et s'appelle l'*élément unité* du groupe ; le symétrique d'un élément $x \in G$ s'écrit x^{-1} et est appelé *inverse* de x ; en abrégé, on dira que le groupe G est « *multiplicatif* ».

Dans le second cas, $x+y$ s'appelle la *somme* de x et y pris dans cet ordre ; l'élément neutre est en général noté 0 ; le symétrique de $x \in G$ s'écrit $-x$ et est appelé *opposé* de x ; en abrégé, on dira que le groupe G est « *additif* ».

Définition 3.2.5 (Puissance n-ième d'un élément).
Si x est un élément d'un groupe $(G, \cdot)$ et n un entier, alors la puissance n-ième de l'élément x sera couramment notée :

si G est multiplicatif *par* x^n *où* $x^n = \underbrace{x \cdot x \cdot \cdots \cdot x}_{n \text{ fois}}$

si G est additif *par* nx *où* $nx = \underbrace{x + x + \cdots + x}_{n \text{ fois}}$

Exemple 3.2.2 (Groupes de nombres).
$(\mathbb{Z}, +)$, $(\mathbb{Q}, +)$, $(\mathbb{R}, +)$ et $(\mathbb{C}, +)$ sont des groupes abéliens d'élément neutre 0, où $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ et $\mathbb{C}$ désignant respectivement l'ensemble des nombres entiers, rationnels, réels et complexes.

Posons $\mathbb{Q}^* = \mathbb{Q} - \{0\}$ et définissons de même $\mathbb{R}^*$ et $\mathbb{C}^*$; alors $(\mathbb{Q}^*, \times)$, $(\mathbb{R}^*, \times)$ et $(\mathbb{C}^*, \times)$ sont des groupes abéliens d'élément neutre 1. On remarque que $(\mathbb{Q}, \times)$, par exemple, n'est pas un groupe, car 0 n'a pas d'inverse dans $\mathbb{Q}$.

3.2.2.1 Morphismes de groupes

Définition 3.2.6 (Morphisme de groupe).
Étant donné deux groupes $(G, \cdot)$ et $(G', *)$. Un morphisme de groupes *(ou* homomorphisme de groupes*) de G dans G' est une application $f : G \to G'$ telle que, quels que soient x et y dans G, on a :$f(x \cdot y) = f(x) * f(y)$.*

L'ensemble des morphismes d'un groupe G dans un groupe G' sera noté $Hom(G, G')$. Un morphisme d'un groupe G dans lui-même est appelé *endomorphisme de groupe*. L'ensemble des endomorphismes d'un groupe G sera noté $End(G)$.

Exemple 3.2.3.
Si « id_G » désigne l'application « identité » du groupe E dans lui-même, alors $\mathrm{id}_E \in End(G)$.

Définition 3.2.7 (Isomorphisme de groupe).
Une application f d'un groupe G dans un groupe G' est un isomorphisme de groupes *si $f \in Hom(G, G')$ et s'il existe $g \in Hom(G', G)$ tel que : $g \circ f = \mathrm{id}_G$ et $f \circ g = \mathrm{id}_{G'}$.*

Définition 3.2.8 (Groupes isomorphes).
S'il existe un isomorphisme d'un groupe G sur un groupe G', on dit que G et G' sont des groupes isomorphes *; dans ce cas on écrit : $G \simeq G'$.*

Un isomorphisme de G sur lui-même est appelé un *automorphisme* du groupe G. L'ensemble des automorphismes d'un groupe G est noté $Aut(G)$.

3.2.3 Sous-groupe

Sauf mention contraire, G désigne toujours un groupe multiplicatif d'élément neutre e.

Définition 3.2.9.
Soient G un groupe et H une partie non vide de G. H est un sous-groupe *de G si les deux conditions suivantes sont satisfaites :*
- *si $(x, y) \in H \times H$, alors $x \cdot y \in H$;*
- *si $x \in H$, alors $x^{-1} \in H$.*

On appelle *sous-groupe propre* d'un groupe G tout sous-groupe de G distinct de G. Dans tout groupe G ayant plus d'un élément, $\{e\}$ est un sous-groupe propre.

Exemple 3.2.4 (Sous-groupes de $(\mathbb{Z}, +)$).
Pour tout entier $n \in \mathbb{N}$, $n\mathbb{Z} = \{nx \mid x \in \mathbb{Z}\}$ est un sous-groupe de $\mathbb{Z}$ et tout sous-groupe de $\mathbb{Z}$ est de cette forme.

Proposition 3.2.1.
Soit G un groupe et $\{H_i\}_{i \in I}$ une famille de sous-groupes de G,quel que soit l'ensemble non vide I ; alors, $\bigcap_{i \in I} H_i$ est un sous-groupe de G.

Définition 3.2.10 (Sous-groupe engendré).
Soient G un groupe et S une partie non vide de G ; désignons par $\mathscr{H}_S$ l'ensemble des sous-groupes de G contenant S et posons $\langle S \rangle = \bigcap_{H \in \mathscr{H}_S} H$.

$\langle S \rangle$ est un sous-groupe de G (proposition précédente) appelé sous-groupe de G engendré par S.

Si S est une partie finie non vide de G, c'est-à-dire, $S = \{x_1, x_2, \ldots, x_n\}$, alors $\langle S \rangle$ s'écrit $\langle x_1, x_2, \ldots, x_n \rangle$.

Proposition 3.2.2.

Soit S une partie non vide d'un groupe G, on a :

$$\langle S\rangle = \left\{ x_1 \cdot x_2 \cdot \cdots \cdot x_n \;\middle|\; n \in \mathbb{N}^* \text{ et quel que soit } i \in \{1,\dots,n\}, x_i \in S \text{ ou } x_i^{-1} \in S \right\}$$

Exemple 3.2.5.
Soient G un groupe et $S = \{x\}$ tel que $x \in G$, $\langle S\rangle$ s'écrit alors $\langle x\rangle$, et on a :

$$\langle x\rangle = \{x^n \mid n \in \mathbb{Z}\}$$

Si $x = e$, alors $\langle x\rangle = \langle e\rangle = \{e\}$

Définitions 3.2.11.
Soit G un groupe.

- *Si S est une partie non vide de G telle que $\langle S\rangle = G$, on dit que S est une* partie génératrice *du groupe G, ou que S est* un ensemble de générateurs *de G, ou encore que S* engendre *G.*
- *S'il existe $x \in G$ tel que $\langle x\rangle = G$, le groupe G est dit* monogène. *Plus généralement, s'il existe une partie non vide et finie de G, $S = \{x_1, x_2, \dots, x_n\}$, telle que $\langle S\rangle = G$, on dit que le groupe G est* de type fini. *Un groupe fini est nécessairement de type fini, mais la réciproque est fausse (voir l'exemple 3.2.6 qui suit).*

Exemple 3.2.6.
Puisque tout $n \in \mathbb{Z}$ s'écrit $\underbrace{1+1+\cdots+1}_{n \text{ fois}}$ si $n > 0$, $\underbrace{(-1)+(-1)+\cdots+(-1)}_{n \text{ fois}}$ si $n < 0$ et $0 = 1 + (-1)$, $\mathbb{Z}$ est un groupe monogène engendré par 1.

On remarque que $\mathbb{Z}$ peut aussi être considéré comme engendré par -1 (en générale, la partie génératrice n'est pas unique).

On remarque aussi que $\mathbb{Z}$ est un groupe non fini, alors que $\mathbb{Z}$ est bien de type fini ($\mathbb{Z} = \langle 1\rangle$ par exemple).

Définition 3.2.12 (Groupe cyclique).
On appellera groupe cyclique *tout groupe monogène fini.*

Définitions 3.2.13 (Ordre d'un élément d'un groupe).
Soit G un groupe quelconque et x un élément de G.

- *Si le sous-groupe de G engendré par x est de cardinal infini, on dit que x est d'*ordre infini *dans G.*
- *Si le sous-groupe de G engendré par x est fini, on dit que x est d'*ordre fini *dans G et le cardinal du sous-groupe $\langle x\rangle$ s'appelle l'*ordre *de x dans G ; on le note $o(x)$, c'est-à-dire, $o(x) = o(\langle x\rangle)$ (voir la définition 3.2.4 page 44).*

Exemples 3.2.7.

- *Dans tout groupe G, l'élément neutre est le seul élément d'ordre 1.*
- *Dans $\mathbb{Z}$, tout élément $x \neq 0$ est d'ordre infini.*

3.3 Groupes symétriques S_n

Exemple 3.3.1 (Exemple élémentaire).
Soit E un ensemble non vide ; notons S_E l'ensemble des permutations *de E (c'est-à-dire des bijections de E dans lui-même). On sait que si f et g sont deux éléments de S_E, alors $f \circ g \in S_E$.*

La composition des applications est donc une loi de composition interne dans S_E. Cette loi est associative et possède un élément neutre id_E *:*

$$\mathrm{id}_E \circ f = f \circ \mathrm{id}_E = f \quad \textit{pour tout} \quad f \in S_E$$

de plus, toute permutation f de E admet une application réciproque f^{-1} qui est aussi une permutation de E et qui vérifie

$$f \circ f^{-1} = f^{-1} \circ f = \mathrm{id}_E$$

On en déduit que $(S_E, \circ)$ est un groupe. Le groupe S_E est appelé groupe symétrique *de E. Si $E = \{1, 2, \ldots, n\}$, le groupe symétrique S_E est noté S_n, et s'appelle* le groupe symétrique de degré n. *L'ensemble E est noté désormais $\mathbb{N}_n$. On sait, d'après un résultat d'analyse combinatoire que le nombre des permutations d'un ensemble de n éléments est $n!$; par suite :*

S_n est un groupe fini d'ordre $n!$.

Un élément $\sigma \in S_n$ s'écrit : $\sigma = \begin{pmatrix} 1 & 2 & \cdots & n \\ \sigma(1) & \sigma(2) & \cdots & \sigma(n) \end{pmatrix}$. Ainsi, les 6 *éléments de S_3 peuvent s'écrire :*

$$e = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}, \quad \sigma_1 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \quad \sigma_2 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix},$$
$$\tau_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}, \quad \tau_2 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}, \quad \tau_3 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}.$$

On remarque que le groupe S_3 est non abélien (par exemple $\sigma_1 \circ \tau_3 \neq \tau_3 \circ \sigma_1$). D'une façon générale, pour tout ensemble non vide E tel que $|E| > 2$, le groupe symétrique S_E est non abélien.

Pour tout groupe G, $Aut(G)$ est un sous-groupe du groupe symétrique S_G.

Plus généralement, S_n pouvait être considéré comme le groupe des permutations de tout ensemble fini de cardinal n.

3.3.1 Notion de $\sigma-$orbite

Définition 3.3.1 (Support d'une permutation).
Soit $\sigma \in S_n$. Le support *de σ est l'ensemble : $supp(\sigma) = \{i \in \mathbb{N}_n \mid \sigma(i) \neq i\}$.*

Dans S_n, $\sigma = e$ si et seulement si $supp(\sigma) = \varnothing$. Dans S_n, si $\sigma \neq e$, alors $|supp(\sigma)| > 1$.

Proposition 3.3.1.
Dans tout groupe S_n deux permutations σ_1, σ_2 dont les supports sont disjoints commutent (c'est-à-dire $\sigma_1 \circ \sigma_2 = \sigma_2 \circ \sigma_1$).

Définition 3.3.2 ($\sigma-$orbite).
Pour tout $\sigma \in S_n$ et tout $i \in \mathbb{N}_n$, $\Omega_\sigma(i) = \{\sigma^r(i) \mid r \in \mathbb{Z}\}$ s'appelle la $\sigma-$ orbite *de i (ou l'*orbite de i suivant σ*).*

3.3.2 Cycles dans S_n

Á titre d'exemple, considérons la permutation :

$$\gamma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 3 & 6 & 2 & 5 \end{pmatrix}$$

Le support de γ est $\{2, 4, 5, 6\}$ et on remarque que :

$$\gamma(2) = 4, \quad \gamma(4) = 6, \quad \gamma(6) = 5 \quad \text{et} \quad \gamma(5) = 2.$$

On dit que, dans le groupe S_6 , γ est un cycle de longueur 4 et γ est noté $(2, 4, 6, 5)$. La notion générale de cycle dans S_n est la suivante :

Définition 3.3.3 (Cycle).
Une permutation $\gamma \in S_n$, est un cycle de longueur r *($1 \leq r \leq n$ dans $\mathbb{N}$) s'il existe un ensemble ordonné de r entiers distincts dans $\mathbb{N}_n$: $j_1, j_2, \ldots, j_r$ tels que : $\gamma(j_1) = j_2, \quad \gamma(j_2) = j_3, \ldots, \gamma(j_{r-1}) = j_r, \quad \gamma(j_r) = j_1$ et pour tout $k \in \mathbb{N}_n - \{j_1, j_2, \ldots, j_r\}$, $\gamma(k) = k$.*

Un tel cycle sera noté $\gamma = (j_1, j_2, \ldots, j_r)$. L'ensemble $\{j_1, j_2, \ldots, j_r\}$ est le support de γ. Dans S_n, un cycle γ de longueur r (c'est-à-dire $|supp(\gamma)| = r$), avec $1 \leq r \leq n$, sera appelé un *$r-$cycle*. Dans un groupe S_n, on dira que deux cycles sont *disjoints*, si leurs supports sont disjoints.

Définition 3.3.4 (Transposition).
Un cycle de longueur 2 dans S_n (avec $n \geq 2$) est appelé transposition.

Si τ est une transposition, alors il existe $j_1, j_2 \in \mathbb{N}_n$ tels que : $\tau = (j_1, j_2)$ et on a $\tau(j_1) = j_2$, $\tau(j_2) = j_1$ et $\tau(k) = k$ pour tout $k \in \mathbb{N}_n - \{j_1, j_2\}$. Une transposition dans S_n est donc une permutation qui, dans $\mathbb{N}_n$, échange deux éléments et laisse invariants tous les autres (lorsque $n \geq 3$).

Exemples 3.3.2.
– *$S_2 = \{e, \tau\}$ où τ est la transposition qui échange 1 et 2.*

– *Dans S_3 , les permutations τ_1, τ_2 et τ_3 (notations de l'exemple 3.3.1 page 48) sont des transpositions et σ_1 et σ_2 sont des cycles de longueur 3 (3−cycle).*

Théorème 3.3.1.
Toute permutation $\sigma \neq e$ dans S_n s'écrit sous la forme :

$$\sigma = \gamma_1 \circ \gamma_2 \circ \cdots \circ \gamma_s$$

où $s \in \mathbb{N}^*$ et $\gamma_1, \gamma_2, \ldots, \gamma_s$ sont des cycles disjoints, tous différents de e et cette décomposition est unique à l'ordre des facteurs près.

Cette décomposition sera appelée : *décomposition canonique* de σ en un produit de cycles.

Ce théorème exprime que tout groupe S_n est engendré par l'ensemble de ses cycles.

Exemple 3.3.3.
Soit σ la permutation du groupe S_6 suivante :

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 2 & 1 & 6 & 3 & 4 \end{pmatrix}$$

La décomposition de $\mathbb{N}_6$ en $\sigma-$orbites implique $\sigma = \gamma_1 \circ \gamma_2$ où $\gamma_1 = (1, 5, 3)$ et $\gamma_2 = (4, 6)$. On écrira $\sigma = (1, 5, 3)(4, 6)$.

Deuxième partie

Contributions

Chapitre 4

Détection et élimination dynamique de la symétrie dans le problème de satisfiabilité

Sommaire

4.1 Introduction 53
4.2 Symétrie dans la logique propositionnelle 54
4.3 La détection et l'élimination de la symétrie locale . . 60
4.3.1 Détection dynamique de la symétrie 61
4.3.2 Élimination des symétries : 62
4.4 Exploitation de la symétrie 62
4.5 Expérimentations 62
4.5.1 Résultats sur des instances SAT diverses 64
4.5.2 Résultats sur des instances du problème de coloration de graphes 64
4.6 Conclusion 67

4.1 Introduction

Plusieurs méthodes d'élimination de la symétrie pour le problème de satisfiabilité ont été introduites [Crawford et al., 1996, Aloul et al., 2003, Aloul et al., 2004]. Cependant, pratiquement toutes ces méthodes exploitent uniquement les symétries globales et ignorent le traitement des symétries locales. Ceci est dû à la difficulté de la détection et de l'élimination dynamique de ces symétries. Contrairement aux symétries globales qui peuvent être exploitées par des approches statiques faciles à implémenter.

Une approche qui détecte et élimine dynamiquement les symétries locales en logique propositionnelle est proposée dans [Benhamou and Sais, 1992, Benhamou and Sais, 1994, Benhamou et al., 1994]. Mais cette méthode est incomplète dans le sens où elle ne détecte qu'une partie des symétries locales, et non pas tout le groupe total de symétries locales. Une alternative à cette méthode est d'adapter et d'utiliser l'outil Saucy qui permet de calculer les automorphismes de graphes [Aloul et al., 2003] afin de détecter les symétries locales durant la recherche, puisque le groupe d'automorphismes du graphe déduit à partir de l'instance SAT est identique au groupe de symétries de l'instance SAT.

Dans ce chapitre, nous présentons une méthode alternative qui élimine les symétries locales pour la résolution du problème SAT, en exploitant le groupe total des symétries[Benhamou et al., 2009, Benhamou et al., 2010c, Benhamou et al., 2010b, Benhamou et al., 2010e]. Cette méthode consiste à réduire incrémentalement la sous-formule logique définie à chaque nœud de l'arbre de recherche à un graphe sur lequel nous utilisons un outil de calcul d'automorphismes de graphes tel que Saucy [Aloul et al., 2003]. L'élimination des symétries est implémentée dans un solveur SAT que nous avons expérimenté et comparé sur plusieurs instances SAT. Les résultats obtenus sont encourageants et montrent que l'exploitation des symétries locales donne de meilleurs résultats que l'exploitation des symétries globales sur certaines instances SAT et que la combinaison de l'exploitation des deux types de symétries sont complémentaires.

Pour des rappels sur les notions élémentaires sur le problème de satisfiabilité, le lecteur peut se référer au chapitre 1, et sur les permutations et les groupes symétriques, il peut se référer au chapitre 3. Le reste du chapitre est organisé comme suit : La section 4.2 définit le principe de la symétrie et donne quelques propriétés. La section 4.3 décrit la nouvelle méthode de détection et d'élimination de la symétrie que nous proposons. La section 4.4 montre comment la coupure de symétrie est intégrée dans une procédure du type Davis & Putnam. Nous évaluons la méthode proposée dans la section 4.5 où plusieurs instances SAT ont été testées et où une comparaison avec d'autres méthodes est donnée. Finalement, nous concluons ce travail dans la section 4.6.

4.2 Symétrie dans la logique propositionnelle

Depuis la définition de symétrie de Krishnamurthy [Krishnamurty, 1985] dans la logique propositionnelle, plusieurs autres définitions ont été données par la communauté de programmation par contraintes [1] (CP). Benhamou et Saïs dans leur papier [Benhamou and Sais, 1992] ont raffiné la définition de

1. En anglais : Contraint Programming.

la symétrie en la généralisant au cas de littéraux, et ils ont montré pour la première fois comment détecter et exploiter les symétries dans les algorithmes de déduction automatique. Freuder dans son papier [Freuder, 1991] a introduit les notions d'interchangeabilité globale et locale, où deux valeurs d'un domaine sont interchangeables dans un CSP, si elles peuvent être substituées l'une à l'autre sans aucun effet sur le CSP. En revanche Benhamou dans [Benhamou, 1994] a défini deux niveaux de symétrie sémantique et une notion de symétrie syntaxique. Il a également montré que l'interchangeabilité globale de Freuder est un cas particulier de symétrie sémantique et que l'interchangeabilité de voisinage « locale » est un cas particulier de la symétrie syntaxique.

Plus récemment, Cohen et al [Cohen et al., 2005] ont fait un état de l'art sur les définitions de symétrie les plus connues dans les CSPs et les ont regroupées dans deux définitions : *les symétries de solutions* (sémantiques) et *les symétries de contraintes* (syntaxiques). Presque toutes ces définitions peuvent être identifiées comme appartenant à l'une des deux familles de symétrie : *la symétrie syntaxique* ou *la symétrie sémantique*. Nous allons définir dans ce qui suit les deux symétries sémantiques et syntaxiques dans la logique propositionnelle et nous allons montrer leur relation avec les symétries de solutions ainsi que les symétries de contraintes dans les CSPs.

Définition 4.2.1 (Permutation d'une formule propositionnelle).
Soit $\mathcal{F}$ une formule propositionnelle donnée sous la forme CNF. Une permutation de la formule propositionnelle *$\mathcal{F}$ est une permutation σ de $lits(\mathcal{F})$* [2]*, c'est-à-dire, un bijection de $lits(\mathcal{F})$ dans lui-même. On calcule $\sigma(\mathcal{F})$ par l'intermédiaire des règles suivantes (en supposant que a et b sont deux littéraux quelconques de $lits(\mathcal{F})$) :*

1. $\sigma(\top) = \top$
2. $\sigma(\bot) = \bot$
3. $\sigma(a \wedge b) = \sigma(a) \wedge \sigma(b)$
4. $\sigma(a \vee b) = \sigma(a) \vee \sigma(b)$

On a vu (voir l'exemple élémentaire 3.3.1 page 48) que l'ensemble des permutations $S_{lits(\mathcal{F})}$ définis sur $lits(\mathcal{F})$, muni de la loi de composition des applications $\circ$ forme un groupe noté $(S_{lits(\mathcal{F})}, \circ)$. Ce groupe symétrique appelé *groupe de permutations* de $\mathcal{F}$.

Définition 4.2.2 (Symétrie sémantique).
Soit $\mathcal{F}$ une formule propositionnelle donnée sous la forme CNF. Une symétrie sémantique *de $\mathcal{F}$ est une permutation σ de la formule $\mathcal{F}$ telle que*

$$\mathcal{F} \models \sigma(\mathcal{F}) \quad et \quad \sigma(\mathcal{F}) \models \mathcal{F}.$$

2. $lits(\mathcal{F})$ est l'ensemble des littéraux contenant chaque littéral de $\mathcal{F}$ et son opposé (voir page 5).

Nous rappelons dans la suite, la définition de la symétrie syntaxique donnée dans [Benhamou and Sais, 1992, Benhamou and Sais, 1994].

Définition 4.2.3 (Symétrie syntaxique).
Soit $\mathcal{F}$ une formule propositionnelle donnée sous la forme CNF. Une symétrie syntaxique *de $\mathcal{F}$ est une permutation σ de la formule $\mathcal{F}$ telle que les conditions suivantes sont vérifiées :*

1. $\forall \ell \in lits(\mathcal{F}), \quad \sigma(\neg\ell) = \neg\sigma(\ell)$,
2. $\sigma(\mathcal{F}) = \mathcal{F}$

En d'autres mots, une symétrie syntaxique d'une formule est une permutation de littéraux qui laisse invariante cette formule. Si on dénote par $Sym(\mathcal{F})$ l'ensemble des symétries syntaxiques de la formules $\mathcal{F}$, alors on peut montrer facilement que $(Sym(\mathcal{F}), \circ)$ est un sous-groupe de $(S_{lits(\mathcal{F})}, \circ)$.

Exemple 4.2.1.
Soit $\mathcal{F}$ l'ensemble des clauses suivantes :

$$\mathcal{F} = \{a \vee b \vee c, \quad \neg a \vee b, \quad \neg b \vee c, \quad \neg c \vee a, \quad \neg a \vee \neg b \vee \neg c\}$$

et σ_1 et σ_2 deux permutations de $\mathcal{F}$ définies comme suit :

$$\sigma_1 = (a, b, c)(\neg a, \neg b, \neg c) \qquad \sigma_2 = (a, \neg a)(b, \neg c)(c, \neg b)$$

Les deux permutations σ_1 et σ_2 sont des symétries syntaxiques de $\mathcal{F}$, puisque

$\sigma_1(\neg a) = \neg b = \neg\sigma_1(a)$
$\sigma_1(\neg b) = \neg c = \neg\sigma_1(b)$
$\sigma_1(\neg c) = \neg a = \neg\sigma_1(c)$
$\sigma_1(\mathcal{F}) = \{b \vee c \vee a, \quad \neg b \vee c, \quad \neg c \vee a, \quad \neg a \vee b, \quad \neg b \vee \neg c \vee \neg a\} = \mathcal{F}$

et

$\sigma_2(\neg a) = a = \neg(\neg a) = \neg\sigma_2(a)$
$\sigma_2(\neg b) = c = \neg(\neg c) = \neg\sigma_2(b)$
$\sigma_2(\neg c) = b = \neg(\neg b) = \neg\sigma_2(c)$
$\sigma_2(\mathcal{F}) = \{\neg a \vee \neg c \vee \neg b, \quad a \vee \neg c, \quad c \vee \neg b, \quad b \vee \neg a, \quad a \vee c \vee b\} = \mathcal{F}$

Remarque 4.2.1. Les définitions de symétrie introduites dans les CSPs [Cohen et al., 2005] sont liées à celles introduites dans la logique propositionnelle. Si l'on considère par exemple l'encodage SAT direct d'un problème CSP [Kleer, 1989] où :

- une variable booléenne est introduite pour chaque paire « variable-valeur » du CSP ;
- une clause interdisant chaque $n-$tuple rejeté par une contrainte spécifique est ajoutée ;
- une clause garantissant que la valeur choisie pour chaque variable est préalablement dans son domaine.

Il est trivial alors de voir que la symétrie de solutions du problème CSP est équivalente à la symétrie sémantique de son encodage SAT (la définition 4.2.2) et que la symétrie de contraintes de ce problème CSP est équivalente à la symétrie syntaxique de son encodage SAT (la définition 4.2.3).

Théorème 4.2.1.
Chaque symétrie syntaxique d'une formule $\mathcal{F}$ est une symétrie sémantique de cette formule.

Démonstration. Il est trivial de voir que la symétrie syntaxique est une condition suffisante pour la symétrie sémantique. En fait, si σ est une symétrie syntaxique d'une formule $\mathcal{F}$, alors $\sigma(\mathcal{F}) = \mathcal{F}$. Donc, il en résulte que $\mathcal{F}$ et $\sigma(\mathcal{F})$ ont les mêmes modèles. Ainsi, chaque symétrie syntaxique de la formule $\mathcal{F}$ est une symétrie sémantique de cette formule. □

Remarque 4.2.2. En général, l'inverse de ce théorème n'est pas vrai. Prenons, par exemple, la formule propositionnelle suivante :

$$\mathcal{F} = \{a \vee \neg b,\ b\}$$

et la permutation σ de cette formule $\mathcal{F}$ définie comme suite :

$$\sigma = (a, b)(\neg a, \neg b)$$

On voit, facilement, que σ n'est pas une symétrique syntaxique de $\mathcal{F}$. En fait, on a :

$$\sigma(\mathcal{F}) = \{b \vee \neg a,\ a\} \qquad \text{et donc} \qquad \sigma(\mathcal{F}) \neq \mathcal{F}$$

Par contre, les deux formules $\mathcal{F}$ et $\sigma(\mathcal{F})$ ont un seul modèle commun : $M = \{a, b\}$. Ainsi $\mathcal{F} \models \sigma(\mathcal{F})$ et $\sigma(\mathcal{F}) \models \mathcal{F}$ et donc σ est une symétrie sémantique de $\mathcal{F}$.

Dans la suite, nous travaillerons uniquement sur les symétries syntaxiques. Nous utiliserons donc le mot *symétrie* pour désigner la symétrie syntaxique.

Définition 4.2.4 (Littéraux symétriques).
Deux littéraux ℓ et ℓ' d'une formule $\mathcal{F}$ sont symétriques dans $\mathcal{F}$ *et on note $\ell \sim \ell'$, s'il existe une symétrie σ de $\mathcal{F}$ (c'est-à-dire $\sigma \in Sym(\mathcal{F})$) telle que : $\sigma(\ell) = \ell'$.*

Remarquons que, si $\ell \sim \ell'$ dans $\mathcal{F}$, alors $\ell' \sim \ell$ dans $\mathcal{F}$. En fait, comme $\ell \sim \ell'$ dans $\mathcal{F}$, alors il existe une symétrie $\sigma \in Sym(\mathcal{F})$ telle que $\sigma(\ell) = \ell'$ et en remarquant que $(Sym(\mathcal{F}), \circ)$ est un groupe, on assure l'existence de la symétrie $\sigma^{-1} \in Sym(\mathcal{F})$ telle que $\sigma^{-1}(\ell') = \ell$. Donc $\ell' \sim \ell$ dans $\mathcal{F}$.

Définition 4.2.5 (Orbite d'un littéral).
*Soit $\mathcal{F}$ une formule de LP et ℓ un littéral de $\mathcal{F}$. L'*orbite *du littéral ℓ notée $Orbite_{\mathcal{F}}(\ell)$ est définie par : $Orbite_{\mathcal{F}}(\ell) = \{\sigma(\ell) \mid \sigma \in Sym(\mathcal{F})\}$.*

Définition 4.2.6 (Ensemble de générateurs de symétries).
Soit $\mathcal{F}$ une formule de LP et $(Sym(\mathcal{F}), \circ)$ le groupe de symétries syntaxiques de la formule $\mathcal{F}$. L'ensemble de générateurs de symétries $Gen_{\mathcal{F}}$ de la formule $\mathcal{F}$ est un sous-ensemble de $Sym(\mathcal{F})$ tel que $Sym(\mathcal{F}) = \langle Gen_{\mathcal{F}} \rangle$.

Proposition 4.2.1.
Tous les éléments d'une orbite d'un littéral sont symétriques deux à deux.

Démonstration. La preuve est une conséquence des deux définitions précédentes.

Soit ℓ, a et b des littéraux de $\mathcal{F}$ tels que : $a \neq b$, $a \in Orbite_{\mathcal{F}}(\ell)$ et $b \in Orbite_{\mathcal{F}}(\ell)$. D'après la définition 4.2.5, il existe deux symétries σ_1 et σ_2 telles que :

$$\sigma_1(\ell) = a \qquad \text{et} \qquad \sigma_2(\ell) = b$$

La symétrie $\sigma_2 \circ \sigma_1^{-1}$ est telle que :

$$\sigma_2 \circ \sigma_1^{-1}(a) = \sigma_2\Big(\sigma_1^{-1}(a)\Big) = \sigma_2(\ell) = b$$

Ainsi a et b sont symétriques dans $\mathcal{F}$ (d'après la définition 4.2.4). □

Exemple 4.2.2.
Dans l'exemple 4.2.1 (page 56), l'orbite du littéral a est la suivante :

$$Orbite_{\mathcal{F}}(a) = \{a, b, c, \neg a, \neg b, \neg c\}$$

Nous pouvons voir que tous les littéraux sont dans la même orbite. Donc, ils sont tous symétriques.

Si $\mathcal{I}$ est un modèle de $\mathcal{F}$ et σ une symétrie, nous pouvons avoir un autre modèle de $\mathcal{F}$ en appliquant σ sur les littéraux qui apparaissent dans $\mathcal{I}$. Autrement dit, si $\mathcal{I}$ est un modèle de $\mathcal{F}$ alors $\sigma(\mathcal{I})$ est un modèle de $\mathcal{F}$.

Une symétrie σ transforme chaque modèle en un autre modèle et chaque contre-modèle en un autre contre-modèle.

Proposition 4.2.2.
Soient $\mathcal{F}$ une formule propositionnelle, ℓ un littéral de $\mathcal{F}$, $\mathcal{I}$ une interprétation de $\mathcal{F}$ et σ une symétrie de $\mathcal{F}$:

$$Si \quad \mathcal{I}(\ell) = vrai, \quad alors \quad \sigma(\mathcal{I})\Big(\sigma(\ell)\Big) = vrai$$

Démonstration. Comme on peut considérer une interprétation $\mathcal{I}$ de $\mathcal{F}$ comme l'ensemble des littéraux de $lits(\mathcal{F})$ qui sont vrais sous $\mathcal{I}$ (voir la page 8), alors si ℓ est *vrai* dans une interprétation $\mathcal{I}$, alors $\sigma(\ell)$ doit être *vrai* dans l'interprétation $\sigma(\mathcal{I})$. □

Nous déduisons la proposition suivante.

Proposition 4.2.3.
Si un littéral ℓ est vrai dans un modèle $\mathcal{I}$ de la formule propositionnelle $\mathcal{F}$ et σ est une symétrie de $\mathcal{F}$, alors $\sigma(\ell)$ doit être vrai dans le modèle $\sigma(\mathcal{I})$ de $\mathcal{F}$.

Théorème 4.2.2.
Soient $\mathcal{F}$ une formule propositionnelle et $\ell, \ell' \in lits(\mathcal{F})$ tels que ℓ et ℓ' dans la même orbite.

Le littéral ℓ est vrai dans un modèle de $\mathcal{F}$ si et seulement si le littéral ℓ' est vrai dans un modèle de $\mathcal{F}$.

Démonstration. Soient ℓ et ℓ' deux littéraux de la formule propositionnelle $\mathcal{F}$ tels qu'ils sont dans la même orbite et soit $\mathcal{I}$ un modèle de $\mathcal{F}$ tel que le littéral ℓ est *vrai* dans $\mathcal{I}$. Comme les deux littéraux ℓ et ℓ' sont des littéraux quelconques dans la même orbite, alors pour démontrer ce théorème il suffit de trouver un modèle de $\mathcal{F}$ tel que ℓ' est *vrai* dans ce modèle. Par hypothèse, on a les deux littéraux ℓ et ℓ' de la formule propositionnelle $\mathcal{F}$ sont dans la même orbite. D'après la proposition 4.2.1 (page 58) ℓ et ℓ' sont symétriques. Il existe alors une symétrie σ de $\mathcal{F}$ tel que $\sigma(\ell) = \ell'$. Comme ℓ est *vrai* dans $\mathcal{I}$, par utilisation de la proposition 4.2.3 on déduit que $\ell' = \sigma(\ell)$ est *vrai* dans le modèle $\sigma(\mathcal{I})$ de $\mathcal{F}$. □

D'après ce théorème, on peut énoncer la conséquence suivante :

Corollaire 4.2.1.
Soit ℓ un littéral de $\mathcal{F}$. Si ℓ n'est vrai dans aucun modèle de $\mathcal{F}$, alors aucun littéral ℓ' de $Orbite_{\mathcal{F}}(\ell)$ n'est vrai dans aucun modèle de $\mathcal{F}$.

Ce corollaire exprime une propriété importante que nous allons utiliser pour éliminer les symétries locales à chaque nœud de l'arbre de recherche. C'est-à-dire, si un échec est détecté après avoir affecté la valeur *vrai* au littéral courant ℓ, alors nous calculons l'orbite de ℓ et nous assignons la valeur *faux* à chaque littéral dans l'orbite, puisque par symétrie nous savons que la valeur *vrai* ne peut être assignée à ces littéraux sous peine de produire un échec. En effet, les littéraux assignés à *vrai* de l'orbite ne participent à aucun modèle de la sous formule considérée.

De nombreux problèmes difficiles pour la résolution ont été démontrés de complexités polynomiales si la symétrie est considérée. Par exemple, trouver des nombres de Ramsey ou résoudre le problème des pigeons sont connus pour être exponentiels pour la résolution classique, tandis que de courtes démonstrations peuvent être faites pour les deux problèmes lorsqu'on exploite la symétrie dans le système de résolution. Nous allons montrer maintenant comment détecter dynamiquement la symétrie locale.

4.3 La détection et l'élimination de la symétrie locale

Les symétries locales doivent être détectées dynamiquement à chaque nœud de l'arbre de recherche. La détection dynamique de la symétrie a été étudiée dans [Benhamou and Sais, 1992, Benhamou and Sais, 1994] où une méthode de recherche des symétries syntaxiques locales a été donnée. Cependant, cette méthode n'est pas complète, elle détecte uniquement une symétrie σ à chaque nœud de l'arbre de recherche lors de l'échec dans l'affectation du littéral courant ℓ. Une heuristique est utilisée sur les permutations de variables de σ afin d'avoir le nombre maximal de littéraux dans le même cycle de permutation dans lequel apparait ℓ. En dépit de cette heuristique, cette méthode ne détecte pas tous les littéraux symétriques avec ℓ correspondant à l'orbite de ℓ, puisqu'il n'utilise pas toutes les symétries locales.

Comme alternative à cette méthode incomplète de recherche de symétries, nous avons adapté Saucy [Aloul et al., 2003] pour détecter toutes les symétries syntaxiques et nous montrons comment éliminer ces symétries durant la recherche. Saucy est un outil pour le calcul des automorphismes de graphes. Il existe d'autres outils comme Nauty [McKay, 1981] et plus récemment AUTOM [Puget, 2005] ou aussi celui décrit dans [Mears et al., 2006] qui peuvent être adaptés pour la détection des symétries locales. Il est montré dans [Puget, 2005] que AUTOM est l'un des meilleurs outils. Cependant, cet outil n'est pas gratuit et comme depuis peu, une nouvelle version plus performante de Saucy vient de sortir [Darga et al., 2008] ; nous avons choisit Saucy. Il est montré dans [Crawford et al., 1996, Aloul et al., 2003, Aloul et al., 2004] que chaque formule CNF $\mathcal{F}$ peut être représentée par un graphe $G_{\mathcal{F}}$ qui est construit de la façon suivante :

- Chaque variable booléenne est représentée par deux sommets (sommet littéral) dans $G_{\mathcal{F}}$: le littéral positif et son opposé. Ces deux sommets sont connectés par une arête dans le graphe $G_{\mathcal{F}}$.
- Chaque clause non binaire est représentée par un sommet (sommet clause). Une arête connecte ce sommet à chaque sommet représentant un littéral de la clause.
- Chaque clause binaire est représentée par une arête connectant les deux sommets représentant les deus littéraux de la clause. Les sommets correspondants aux clauses binaires ne sont pas ajoutés.

Une propriété importante du graphe $G_{\mathcal{F}}$ est qu'il préserve le groupe de symétries syntaxiques de $\mathcal{F}$. En effet, le groupe de symétries syntaxiques de la formule $\mathcal{F}$ (c'est-à-dire $(Sym(\mathcal{F}), \circ)$) est identique au groupe d'automorphismes de sa représentation graphique $G_{\mathcal{F}}$, donc nous utilisons Saucy sur $G_{\mathcal{F}}$ pour détecter le groupe de symétries syntaxiques de $\mathcal{F}$.

Saucy retourne un ensemble des générateurs $Gen_{\mathcal{F}}$ du groupe de symétries $(Sym(\mathcal{F}), \circ)$ à partir duquel on peut déduire chaque symétrie de $\mathcal{F}$. Saucy offre la possibilité de colorer les sommets du graphe tels que, chaque sommet est autorisé à être permuté avec un autre sommet s'ils ont la même couleur. Ceci restreint les permutations aux nœuds ayant la même couleur. Deux couleurs sont utilisées dans $G_{\mathcal{F}}$, une pour les sommets correspondant aux clauses de $\mathcal{F}$ et l'autre couleur pour les sommets représentant les littéraux de $lits(\mathcal{F})$. Ceci permet de distinguer les sommets clauses des sommets littéraux, et évite donc la génération de symétries entre clauses et littéraux. Le code source de Saucy peut être trouvé à l'adresse suivante :
http://vlsicad.eecs.umich.edu/BK/SAUCY/.

Exemple 4.3.1.
Soit $\mathcal{F}$ la formule CNF donnée dans l'exemple 4.2.1 (page 56). Son graphe associé $G_{\mathcal{F}}$ est donné dans la figure 4.1.

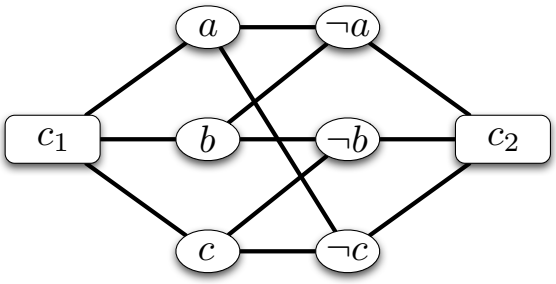

FIGURE 4.1 – Le graphe $G_{\mathcal{F}}$ correspondant à $\mathcal{F}$

4.3.1 Détection dynamique de la symétrie

Soient $\mathcal{F}$ une formule CNF et $\mathcal{I}$ une interprétation partielle de $\mathcal{F}$ où ℓ est le littéral courant en cours d'instanciation. L'interprétation $\mathcal{I}$ simplifie la formule donnée $\mathcal{F}$ en une sous formule $\mathcal{F}_{\mathcal{I}}$ qui définit un état de l'espace de recherche correspondant au nœud courant $N_{\mathcal{I}}$ de l'arbre de recherche. L'idée est de maintenir dynamiquement le graphe $G_{\mathcal{F}_{\mathcal{I}}}$ de la sous formule $\mathcal{F}_{\mathcal{I}}$ correspondant au sous problème local défini au nœud courant $N_{\mathcal{I}}$, alors on colorie le graphe $G_{\mathcal{F}_{\mathcal{I}}}$ et on calcule son groupe d'automorphismes $Aut(G_{\mathcal{F}_{\mathcal{I}}})$.

La sous formule $\mathcal{F}_{\mathcal{I}}$ peut être vue comme le sous problème restant correspondant à la partie non encore résolue de $\mathcal{F}$. En appliquant Saucy sur ce graphe coloré, nous pouvons déduire l'ensemble des générateurs $Gen_{\mathcal{F}_{\mathcal{I}}}$ du sous groupe de symétries $(Sym(\mathcal{F}_{\mathcal{I}}), \circ)$, à partir desquels on peut calculer l'orbite du littéral courant ℓ qui sera utilisée pour faire des coupures de symétrie dans l'arbre de recherche.

4.3.2 Élimination des symétries :

Nous utilisons le corollaire 4.2.1 (page 59) pour élaguer l'espace de recherche des méthodes de résolution. En effet, s'il est montré que l'instanciation de la valeur *vrai* au littéral courant ℓ défini à un nœud donné $N_{\mathcal{I}}$ de l'arbre de recherche mène à un échec, alors l'instanciation de la valeur *vrai* à n'importe quel littéral de l'orbite de ℓ mènera aussi à un échec. Donc, la valeur *faux* doit être assignée à chaque littéral de l'orbite de ℓ. Nous élaguons alors le sous espace correspondant à l'instanciation de la valeur alternative *vrai* à ces littéraux dans l'arbre de recherche. C'est ce que nous appelons la coupure de symétrie.

4.4 Exploitation de la symétrie

Maintenant nous allons montrer comment ces littéraux symétriques peuvent être utilisés pour augmenter l'efficacité des algorithmes de résolution du problème SAT. Nous choisissons dans notre implémentation la procédure de Davis & Putnam (DP) comme méthode de base que nous souhaitons améliorer par l'exploitation de la symétrie.

Si $\mathcal{I}$ est une interprétation partielle inconsistante dans laquelle l'instanciation de la valeur *vrai* au littéral courant ℓ est montrée être en conflit, alors en accord avec le corollaire 4.2.1 (page 59), tous les littéraux dans l'orbite de ℓ calculés en utilisant le groupe $Sym(\mathcal{F}_{\mathcal{I}})$ retourné par Saucy sont symétriques à ℓ. Ainsi, nous assignons la valeur *faux* à chaque littéral de $Orbite_{\mathcal{F}_{\mathcal{I}}}(\ell)$ puisque la valeur *vrai* est montrée être contradictoire, et alors nous élaguons le sous espace correspondant aux instanciations à la valeur *vrai*. La procédure résultant de l'exploitation de cette propriété des symétries, appelée *Satisfiable*, est donnée dans l'algorithme 4.

4.5 Expérimentations

Nous allons maintenant étudier les performances de notre méthode par le biais d'expérimentations. Nous choisissons pour notre étude des instances SAT diverses pour mettre en évidence l'intérêt de l'exploitation de la symétrie dans la satisfiabilité. Nous supposons que l'apport de la symétrie sera plus important dans des instances correspondant à des applications réelles. Ici, nous avons testé et comparé 4 méthodes :

1. No-sym : recherche sans élimination de symétries, c'est le solveur LSAT de base [Ostrowski et al., 2002] ;
2. Global-sym : recherche avec détection et élimination des symétries globales. Cette méthode exploite un programme nommé SHATTER,

Algorithme 4: La procédure Davis Putnam muni de l'élimination des symétries locales

Procedure : Satisfiable($\mathcal{F}$)

début
  si $\mathcal{F} = \varnothing$ **alors** $\mathcal{F}$ est satisfaisable
  sinon si $\mathcal{F}$ *contient la clause vide* **alors** $\mathcal{F}$ est insatisfaisable
  sinon début
    si *il existe un mono-littéral ou un littéral monotone* ℓ **alors**
    si *Satisfiable*($\mathcal{F}_\ell$) **alors** $\mathcal{F}$ est satisfaisable
    sinon $\mathcal{F}$ est insatisfaisable
    sinon début
      Choisir un littéral non assigné ℓ de $\mathcal{F}$
      si *Satisfiable*($\mathcal{F}_\ell$) **alors** $\mathcal{F}$ est satisfaisable
      sinon début
        Gen = Saucy($\mathcal{F}$) ;
        $Orbite_{\mathcal{F}}(\ell) = \{\ell_1, \ell_2, ..., \ell_n\}$ en utilisant Gen ;
        si *Satisfiable*($\mathcal{F}_{\neg\ell_1 \wedge \neg\ell_2 \wedge ... \wedge \neg\ell_n}$) **alors** $\mathcal{F}$ est satisfaisable
        sinon $\mathcal{F}$ est insatisfaisable
      fin
    fin
  fin
fin

comme préprocesseur [Aloul et al., 2003, Aloul et al., 2004] qui détecte et élimine les symétries globales de l'instance considérée en ajoutant à l'instance des clauses éliminant ces symétries. L'instance obtenue est alors résolue par le solveur LSAT. Le temps CPU de Global-sym dans la table 4.1 inclut le temps nécessaire à SHATTER pour le calcul et l'élimination des symétries globales.

3. Local-sym : recherche avec détection et élimination des symétries locales. Cette méthode implémente dans LSAT la stratégie de détection et d'élimination dynamique décrite dans ce travail. Le temps CPU de Local-sym inclut le coût en temps de l'exploitation des symétries locales.
4. Global-Local-sym : recherche qui combine l'exploitation des symétries globales et locales. La méthode consiste à utiliser LSAT avec exploitation des symétries locales (à savoir donc la méthode Local-sym) sur les instances produites par l'utilisation de SHATTER comme préprocesseur.

sur différentes instances SAT comme les FPGA (Field Programmable Gate Array), Chnl, Urquhart et quelques instances issues du problème de coloration de graphes. Nous rappelons que le point commun entre les différentes méthodes présentées précédemment est qu'elles ont toutes en commun la méthode LSAT

comme méthode de base. Les indicateurs de complexité sont le nombre de nœuds de l'arbre de recherche ainsi que le temps d'exécution (en secondes). Le temps nécessaire pour le calcul et l'exploitation des symétries globales et locales est ajouté au temps CPU total pour la résolution des instances considérées. Le code source est en C, il est compilé et exécuté sur une machine équipée d'un Pentium 4, 2.8 GHZ et 1 Gb de RAM.

4.5.1 Résultats sur des instances SAT diverses

La table 4.1 (page 65) montre les résultats obtenus des différentes méthodes sur les instances SAT choisies. Elle donne le nom des instances, la taille de celles-ci (*variables*/*clauses*), le nombre de nœuds et le temps CPU pour la résolution des instances pour chaque méthode.

La table 4.1 montre que Global-sym est en général meilleure que Local-sym et No-sym en nombre de nœuds et en temps CPU sur les problèmes FPGA et Chnl, mais Local-sym est capable de les résoudre aussi en un temps raisonnable. Ces problèmes contiennent un nombre très important de symétries globales, leur élimination est largement suffisante pour résoudre efficacement ces instances. Éliminer les symétries locales sur ces problèmes peut parfois s'avérer être du travail en trop rendre la résolution plus lente. Les instances Urq sont connus pour être plus durs que les FPGA et les Chnl, nous pouvons voir que No-sym n'est pas capable de les résoudre et que Global-sym n'a résolu que l'instance $Urq3\_5$ et n'a pas pu résoudre les autres dans la limite de temps imposée. La méthode Local-sym a résolu toutes les instances Urq efficacement. L'élimination de la symétrie locale dans ce cas est plus avantageuse que l'élimination des symétries globales. Nous pouvons constater que la méthode Local-sym surpasse la méthode Global-sym sur ces instances. Nous pouvons voir qu'en moyenne la méthode Global-Local-sym est meilleure que toutes les autres méthodes, puisqu'elle a résolu toutes les instances plus efficacement. Ses performances son comparables à la méthode Global-sym sur les instances FPGA et Chnl et à la méthode Local-sym sur les instances Urq. Il est donc plus avantageux de combiner les deux types d'élimination de la symétrie (symétrie global et symétries locales) pour ces problèmes. Les résultats confirment que les deux méthodes Global-sym et Local-sym peuvent être complémentaires.

4.5.2 Résultats sur des instances du problème de coloration de graphes

Les instances du problème de coloration de graphes sont générées en fonction des paramètres suivants :

- n : le nombre de sommets ;
- *Couleurs* : le nombre de couleurs ;

Instance	*Vars* :*clauses*	No-sym		Global-sym		Local-sym		Global-Local-sym	
		Nœuds	*Temps*	*Nœuds*	*Temps*	*Nœuds*	*Temps*	*Nœuds*	*Temps*
fpga10_8_SAT	120 : 448	6637776	44.41	**449**	**0.02**	9835	2.09	**449**	0.71
fpga10_9_SAT	135 : 549	-	>1000	**284**	**0.02**	57080	20.37	**284**	0.53
fpga12_8_SAT	144 : 560	6637776	35.79	**165**	**0.00**	9835	2.14	**165**	0.32
fpga13_10_SAT	195 : 905	-	>1000	**4261**	**0.41**	304830	134.89	**4261**	14.08
Chnl10_11	220 : 1122	3628800	100.09	**382**	**0.09**	512	2.42	**382**	3.33
Chnl10_12_3	240 : 1344	3628800	120.72	**322**	**0.10**	512	2.63	**322**	3.41
Chnl11_12_3	264 : 1476	-	>1000	**1123**	**0.26**	1024	6.28	**1123**	12.09
Chnl11_13	286 : 1742	-	>1000	**814**	**0.25**	1024	7.38	**814**	10.96
Chnl11_20	440 : 4220	-	>1000	**523**	**0.38**	1024	18.93	**523**	18.90
Urq3_5	46 : 470	-	>1000	16384	0.16	30	0.09	**15**	**0.00**
Urq4_5	74 : 694	-	>1000	-	>1000	44	0.32	**31**	**0.10**
Urq5_5	121 : 1210	-	>1000	-	>1000	73	1.43	**44**	**0.27**
Urq6_5	180 : 1756	-	>1000	-	>1000	110	4.76	**84**	**2.03**
Urq7_5	240 : 2194	-	>1000	-	>1000	147	9.32	**108**	**3.44**
Urq8_5	327 : 3252	-	>1000	-	>1000	225	27.11	**171**	**9.18**

TABLE 4.1 – Quelques résultats sur des instances SAT

– d : la densité qui est un nombre entre 0 et 1 qui exprime le *ratio* qui est égal au nombre de contraintes (le nombre d'arêtes dans le graphe) sur le nombre de toutes les contraintes possibles.

Pour chaque test correspondant aux paramètres n, *Couleurs* et d fixés, un échantillon de 100 instances est générée aléatoirement et les mesures (temps CPU, nombre de nœuds) sont prises en moyenne.

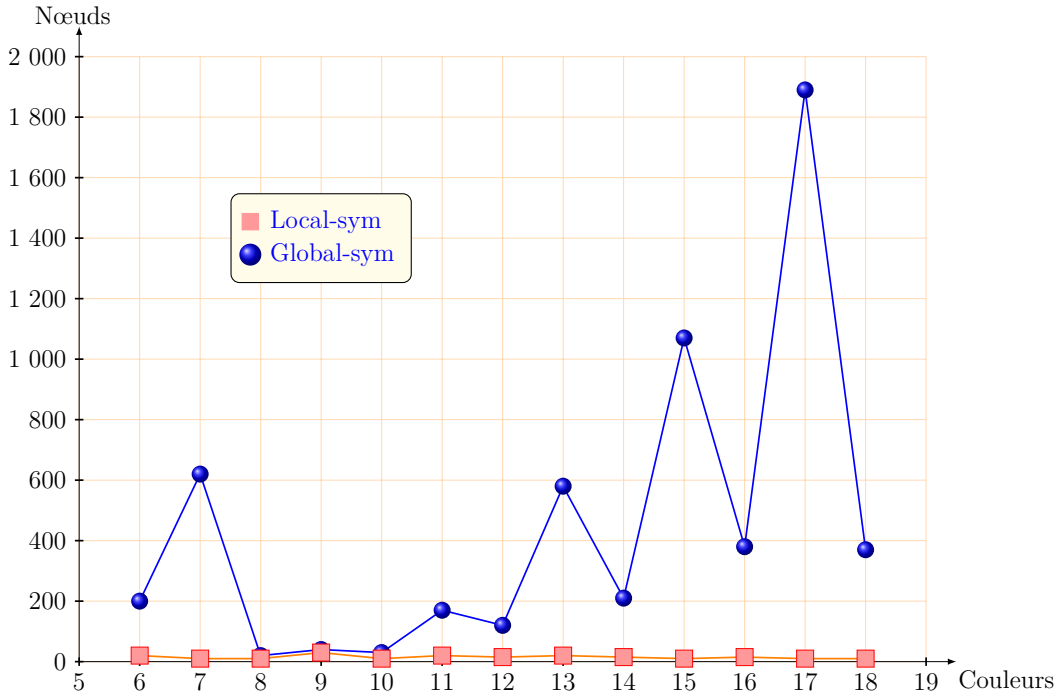

FIGURE 4.2 – Les courbes de nœuds en moyenne pour les deux méthodes d'élimination de la symétrie sur le problème de coloration de graphes où $n = 30$ et $d = 0.5$

Nous avons reporté dans les figures 4.2 et 4.3 les résultats obtenus à partir des méthodes testées : Global-sym, et Local-sym, sur des instances générées aléatoirement où nous avons fixé le nombre de variables à $n = 30$ et où la densité est fixée à $(d = 0.5)$. Les courbes donnent le nombre moyen de nœuds (Figure 4.2), respectivement, le temps CPU moyen (Figure 4.3) en fonction du nombre de couleurs pour chaque méthode.

Nous pouvons voir à l'aide des courbes représentant le nombre moyen de nœuds (Figure 4.2) que la méthode Local-sym détecte et élimine plus de symétries que la méthode Global-sym et que Global-sym n'est pas stable pour le problème de coloration de graphe. À partir des courbes de temps (Figure 4.3), nous pouvons voir que Local-sym est en moyenne plus rapide que Global-sym bien que Saucy est exécuté à chaque nœud pour la méthode Local-sym.

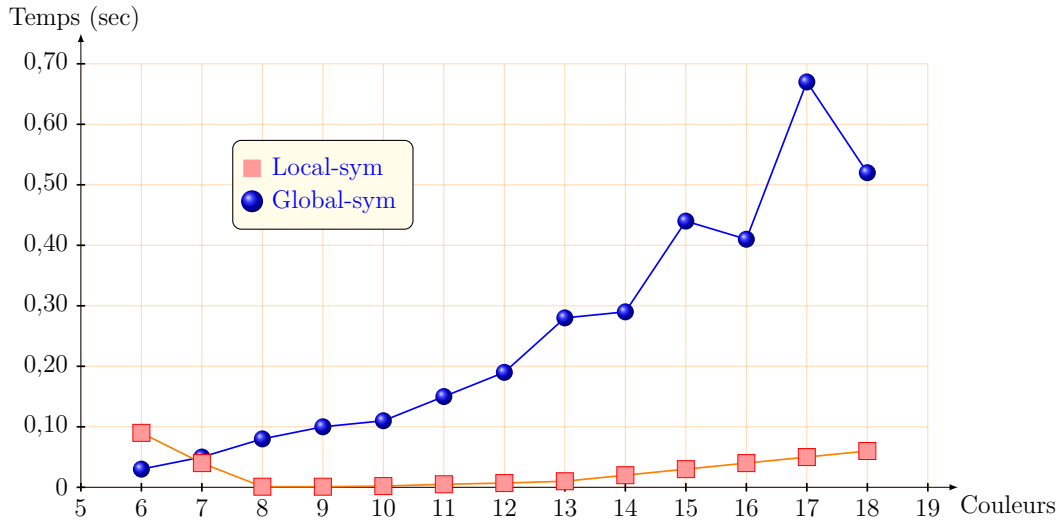

FIGURE 4.3 – Les courbes de temps en moyenne pour les deux méthodes d'élimination de la symétrie sur le problème de coloration de graphes où $n = 30$ et $d = 0.5$

L'élimination des symétries locales est plus avantageuse pour la résolution du problème de coloration de graphes que l'élimination des seules symétries globales sur ces problèmes.

4.6 Conclusion

Ici, nous avons élaboré une méthode complète de détection et d'élimination des symétries aux symétries locales. Les symétries locales sont les symétries de chaque sous formule CNF définie en chaque nœud de l'arbre de recherche et qui est dérivée de la formule initiale en considérant l'interprétation partielle correspondant à ce nœud. Nous avons adapté Saucy pour calculer ces symétries locales en maintenant dynamiquement le graphe correspondant à la sous formule définie en chaque nœud de l'arbre de recherche.

On donne à Saucy en entrée le graphe de la sous formule locale et il nous retourne alors l'ensemble des générateurs du groupe d'automorphismes du graphe donné, sachant que ce groupe est équivalent au groupe de symétries locales de la sous formule considérée. La technique de détection et d'élimination des symétries locales a été implémentée dans le solveur LSAT et testée sur différents problèmes afin de voir les avantages d'exploiter

les symétries locales dans la résolution du problème SAT. Les résultats expérimentaux confirment que l'élimination des symétries locales est d'un apport non négligeable pour la résolution des problèmes SAT et améliore les méthodes n'exploitant que l'élimination des symétries globales sur certains problèmes, et qu'elle peut être complémentaire à l'élimination des symétries globales si on les combine.

Chapitre 5

Symétries et apprentissage des clauses dans les solveurs SAT modernes (CDCL)

Sommaire

5.1 Introduction 69
5.2 Symétries et apprentissage de clauses 70
5.3 Avantages de la symétrie dans les solveurs CDCL . . 73
5.4 Expérimentation 73
5.4.1 Résultats sur différents problèmes symétriques 76
5.4.2 Résultats sur les problèmes de coloration de graphe . . 76
5.4.3 Résultats sur des problèmes des dernières compétitions SAT 78
5.5 Conclusion 82

5.1 Introduction

Des nombreuses améliorations ont été apportées dans les solveurs actuels du problème SAT. Les solveurs de type CDCL (Conflict Driven Clauses Learning) sont aujourd'hui capables de résoudre de manière efficace des problèmes industriels de très grande taille (en nombre de variables et de clauses). Ces derniers utilisent des structures de données paresseuses, des politiques de redémarrage et apprennent de nouvelles clauses à chaque échec au cours de la recherche. Bien que l'utilisation des symétries et l'apprentissage de clauses s'avèrent être des principes puissants, la combinaison des deux n'a pas été investi par les chercheurs du domaine.

Dans ce chapitre, nous présentons une nouvelle approche pour l'apprentissage qui utilise les symétries du problème [Benhamou et al., 2010a, Benhamou et al., 2010d]. Cette méthode consiste, dans un premier temps, à détecter toutes les symétries globales du problème et de les utiliser lorsqu'une nouvelle clause (clause assertive) est déduite au cours de la recherche. Cela nous permet de déduire toutes les clauses assertives symétriques à la clause de référence du problème.

Nous avons mis en application l'apprentissage par symétries dans MiniSat [Eén and S'orensson, 2003] [1] et nous l'avons expérimenté sur différents problèmes. Nous avons comparé MiniSat avec et sans apprentissage par symétries. Les résultats obtenus sont très encourageants et montrent que l'utilisation des symétries dans l'apprentissage est profitable pour des solveurs à base de CDCL.

Pour des rappels sur les notions élémentaires sur le problème de satisfiabilité, le lecteur peut se référer au chapitre 1 (page 3), et sur les permutations et les groupes symétriques, il peut se référer au chapitre 3 (page 43). La section 4.2 (page 54) définit la symétrie et donne les résultats théoriques sur la symétrie que nous utilisons pour l'apprentissage.

5.2 Symétries et apprentissage de clauses

Nous allons montrer comment la symétrie peut être utilisée afin d'améliorer l'apprentissage dans des solveurs de type CDCL. Pour cela, nous introduisons le lemme suivant que nous utiliserons afin de prouver nos résultats sur les clauses assertives symétriques.

Lemme 5.2.1.
Soient $\mathcal{F}$ une formule CNF et σ une symétrie de $\mathcal{F}$. Si $\mathcal{I} = \langle (x), y_1, y_2, \ldots, y_n \rangle$ est une interprétation partielle constituée par le seul littéral de décision x et tous ses littéraux propagés ordonnés $y_1, y_2, \ldots, y_n$, alors :

- $\sigma(\mathcal{I}) = \langle (\sigma(x)), \sigma(y_1), \sigma(y_2), \ldots, \sigma(y_n) \rangle$;
- *Niveau*(x) *dans* $\mathcal{I}$ *est identique à Niveau*$(\sigma(x))$ *dans* $\sigma(\mathcal{I})$;
- $\forall i \in \{1, \ldots, n\}$, *Niveau*$(y_i)$ *dans* $\mathcal{I}$ *est identique à Niveau*$(\sigma(y_i))$ *dans* $\sigma(\mathcal{I})$.

Démonstration. Pour montrer ce lemme, il faut prouver que si $\mathcal{F} \wedge \{x\} \underset{pu}{\models} y_i$, alors nous avons $\mathcal{F} \wedge \{\sigma(x)\} \underset{pu}{\models} \sigma(y_i)$ et ce, pour tout $i \in \{1, \ldots, n\}$ (voir la remarque 1.4.1 page 17). On veut le prouver par induction sur i.

Pour $i = 1$, montrons que si $\mathcal{F} \wedge \{x\} \underset{pu}{\models} y_1$, alors $\mathcal{F} \wedge \{\sigma(x)\} \underset{pu}{\models} \sigma(y_1)$.

1. L'apprentissage par symétrie est générique. Il peut être utilisé dans n'importe quel solveur qui fait de l'apprentissage

Nous avons $\mathcal{F} \wedge \{x\} \underset{pu}{\models} y_1$ par hypothèse. Cela signifie qu'il existe une clause $c \in \mathcal{F}$ telle que $c = \neg x \vee y_1$. Comme σ est une symétrie de la formule $\mathcal{F}$, alors la clause $\sigma(c) = \neg\sigma(x) \vee \sigma(y_1)$ est aussi une clause de la formule $\mathcal{F}$. Cela implique que $\mathcal{F} \wedge \{\sigma(x)\} \underset{pu}{\models} \sigma(y_1)$.

Supposons maintenant que la propriété est vraie jusqu'au rang $i-1$ et montrons qu'elle est vraie pour i. Nous supposons que $\mathcal{F} \wedge \{x\} \underset{pu}{\models} y_i$, et devons montrer que $\mathcal{F} \wedge \{\sigma(x)\} \underset{pu}{\models} \sigma(y_i)$. De $\mathcal{F} \wedge \{x\} \underset{pu}{\models} y_i$ nous déduisons qu'il existe une clause $c \in \mathcal{F}$ telle que $c = \alpha \vee y_i$ où $\alpha \subseteq \neg x \vee \neg y_1 \vee \neg y_2 \vee \cdots \vee \neg y_{i-1}$. Par conséquent $\sigma(c) = \sigma(\alpha) \vee \sigma(y_i)$ est une clause de $\mathcal{F}$ telle que $\sigma(\alpha) \subseteq \neg\sigma(x) \vee \neg\sigma(y_1) \vee \neg\sigma(y_2) \vee \cdots \vee \neg\sigma(y_{i-1})$. Comme $\mathcal{F} \wedge \{x\} \underset{pu}{\models} y_j, \forall j \in \{1, \ldots, i-1\}$, alors par hypothèse de récurrence nous avons $\mathcal{F} \wedge \{\sigma(x)\} \underset{pu}{\models} \sigma(y_j), \forall j \in \{1, \ldots, i-1\}$. En faisant la résolution entre la clause $\sigma(c)$ et les mono-littéraux $\sigma(x), \sigma(y_1), \ldots, \sigma(y_{i-1})$ on déduit $\sigma(y_i)$ donc $\mathcal{F} \wedge \{\sigma(x)\} \underset{pu}{\models} \sigma(y_i)$.

D'après les étapes précédentes, on a $\mathit{Niveau}(x) = \mathit{Niveau}(\sigma(x)) = 1$ et $\forall i \in \{1, \ldots, n\}$, $\mathit{Niveau}(y_i) = \mathit{Niveau}(\sigma(y_i)) = 1$. Alors, $\mathit{Niveau}(x)$ dans $\mathcal{I}$ est identique à $\mathit{Niveau}(\sigma(x))$ dans $\sigma(\mathcal{I})$, et $\forall i \in \{1, \ldots, n\}$, $\mathit{Niveau}(y_i)$ dans $\mathcal{I}$ est identique à $\mathit{Niveau}(\sigma(y_i))$ dans $\sigma(\mathcal{I})$. □

Du lemme précédent, nous pouvons déduire la proposition suivante.

Proposition 5.2.1.
Étant données une formule CNF $\mathcal{F}$ et une symétrie σ de $\mathcal{F}$, si $\mathcal{I}=\prod_{i=1}^{m}\langle(x_i), y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}\rangle$ est une interprétation partielle composée de m littéraux de décisions x_i, et les littéraux $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$ affectés par propagation unitaire dans l'ordre, nous avons alors $\sigma(\mathcal{I})=\prod_{i=1}^{m}\langle(\sigma(x_i)), \sigma(y_{i,1}), \sigma(y_{i,2}), \ldots, \sigma(y_{i,k_i})\rangle$. En plus, $\forall x \in \mathcal{I}$, $\mathit{Niveau}(x)$ dans $\mathcal{I}$ est identique à $\mathit{Niveau}(\sigma(x))$ dans $\sigma(\mathcal{I})$.

Démonstration. La preuve peut être dérivée par application du précédent lemme de manière récursive sur les littéraux de décision x_i et leurs littéraux propagés associés $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$. □

Maintenant, nous introduisons la propriété principale sur la symétrie que nous utiliserons pour améliorer l'apprentissage de clauses dans les solveurs SAT de type CDCL.

Proposition 5.2.2.
Étant données une CNF $\mathcal{F}$, une symétrie σ de $\mathcal{F}$, et une interprétation partielle $\mathcal{I}=\prod_{i=1}^{m}\langle(x_i), y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}\rangle$ constituée de m littéraux de décision x_i et leurs littéraux propagés $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$ dans l'ordre. Si

$c = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_k \vee x$ est une clause assertive correspondant à $\mathcal{I}$ avec x son littéral assertif, alors $\sigma(c)$ est une clause assertive correspondant à l'interprétation partielle symétrique $\sigma(\mathcal{I})$ avec $\sigma(x)$ son littéral assertif.

Démonstration. Pour prouver cette proposition, nous devons montrer que $\sigma(\mathcal{I})[\sigma(c)] = Faux$, $Niveau(\sigma(x)) = m$ et $\forall i \in \{1, \ldots, k\}$, $Niveau(\sigma(\ell_i)) < m$ dans $\sigma(\mathcal{I})$.

Par hypothèse, c est une clause assertive correspondant à $\mathcal{I}$, donc $\mathcal{I}[c] = Faux$. Comme σ est une symétrie et en utilisant la proposition 4.2.2 (page 58), on a $\sigma(\mathcal{I})[\sigma(c)] = Faux$. Par utilisation de la proposition 5.2.1, nous pouvons déduire la condition sur les niveaux : $Niveau(\sigma(x)) = m$ et $\forall i \in \{1, \ldots, k\}$ $Niveau(\sigma(\ell_i)) < m$ dans $\sigma(\mathcal{I})$. □

Cette propriété permet aux solveurs CDCL d'ajouter à l'ensemble de clauses en même temps la clause assertive principale c correspondant à $\mathcal{I}$ ainsi que toutes ses clauses assertives symétriques $\sigma(c)$. Cela permet d'éviter de parcourir des sous-espaces isomorphes correspondant aux interprétations partielles symétriques $\sigma(I)$ de I. Les expérimentations qui vont suivre montreront que cette propriété améliore considérablement l'efficacité des solveurs CDCL.

Le cas le plus intéressant est lorsque la clause assertive est réduite à un mono-littéral. Nous pouvons déduire par symétrie tous les littéraux appartenant à l'orbite de ce mono-littéral et ainsi propager directement tous les littéraux opposés de cet orbite. Formellement, nous avons la propriété suivante :

Proposition 5.2.3.
Étant données une CNF $\mathcal{F}$ et une interprétation partielle I, si ℓ est une clause assertive unitaire, alors pour tout $\ell' \in Orbite_{\mathcal{F}}(\ell)$, ℓ' est une clause assertive unitaire. En plus, $\mathcal{F}$ est satisfiable si et seulement si $(\mathcal{F} \wedge \neg\ell \bigwedge_{\ell_i \in Orbite_{\mathcal{F}}(\ell)} \neg\ell_i)$ est satisfiable.

Démonstration. La proposition est un cas particulier de la proposition 5.2.1. □

Le schéma d'apprentissage par symétrie (SLS) est alors un schéma classique d'apprentissage (SL) incluant les deux propriétés sur les symétries qui permettent d'inférer des clauses assertives symétriques afin de booster l'apprentissage. Dans ce qui suit, nous montrons comment ce schéma est implanté dans un solveur CDCL.

5.3 Avantages de la symétrie dans les solveurs CDCL

Dans cette partie, nous présentons le nouveau schéma d'apprentissage par symétrie (SLS) pour les solveurs CDCL. Ce nouveau schéma SLS induit un solveur moderne basé sur la propagation unitaire, l'apprentissage de clauses par symétrie, une politique de redémarrage [Gomes et al., 1997] et le retour arrière non-chronologique [Silva and Sakallah, 1996, Jr. and Schrag, 1997]. Dans l'algorithme 5, nous décrivons le solveur CDCL adoptant le nouveau schéma d'apprentissage de clauses par symétrie. Le pseudo-code de cette procédure appelé SCLR (Algorithme 5) est basé sur celui présenté dans [Pipatsrisawat and Darwiche, 2009] (pour plus de détails voir l'algorithme 3 page 23).

La principale différence entre la procédure SCLR que nous proposons et les procédures classiques de type CDCL est l'implantation des deux propositions 5.2.2 et 5.2.3 dans SCLR (lignes 10 à 14, de l'algorithme 5). En effet, lorsqu'il y a une interprétation partielle conflictuelle $\mathcal{I}$ et que l'ensemble des points de décision D est non vide, une clause assertive c est déduite et deux cas sont testés :

- Si la clause assertive c est unitaire($c = \ell$, ligne 10, Algorithme 5), alors tous les littéraux symétriques de ℓ (littéraux de son orbite, ligne 11, Algorithme 5) sont alors ajoutés à Γ (ligne 14, Algorithme 5) et leur négation est propagée à la racine de l'arbre de recherche.
- Si la clause assertive c n'est pas unitaire (ligne 12), alors c et toutes les clauses symétriques $\sigma(c)$ induites par les générateurs $\sigma \in Gen_{\mathcal{F}}$ (ligne 13) sont ajoutées à Γ (ligne 14). Pour des raisons d'efficacité, dans notre implantation, nous limitons la génération de clauses symétriques (non unitaires) à celles produites par l'ensemble des générateurs du groupe de symétries de la formule.

Nous avons choisi pour notre implantation le solveur MiniSat [Eén and S'orensson, 2003] auquel nous avons ajouté l'apprentissage par symétrie. Cependant, notre approche est générique et peut donc être utilisée dans d'autres solveurs de type CDCL. Nous présentons les résultats expérimentaux dans la partie suivante.

5.4 Expérimentation

Nous étudions les performances de notre approche par une analyse expérimentale. Nous avons pour cela choisi certaines instances afin de tester notre méthode d'apprentissage par symétrie. Nous nous attendons à ce que l'apprentissage par symétrie soit profitable aux solveurs pour des problèmes contenant des symétries. Nous avons testé et comparé MiniSat et MiniSat avec

Algorithme 5: SCLR : solveurs SAT avec apprentissage par symétries et redémarrages

entrée : Une formule CNF $\mathcal{F}$
sortie : Une solution de $\mathcal{F}$ ou unsat si $\mathcal{F}$ n'est pas satisfiable

```
1  D ⟵ ⟨⟩ // Littéraux de décision;
2  Γ ⟵ vrai // Clauses apprises;
3  pour vrai faire
4      si S = (F, Γ, D) est U-inconsistant alors
5          // Il y a un conflit.
6          si D = ⟨⟩ alors
7              retourner unsat
8          c ⟵ une clause assertive de S
9          m ⟵ le niveau assertif de c
10         si c est unitaire (c = ℓ) alors
11             A ⟵ Orbite_F(ℓ)
12         sinon
13             A ⟵ {σ(c) | σ ∈ Gen_F}
14         Γ ⟵ Γ ⋀_{c∈A} c
15         D ⟵ D_m // les m premières décisions
16     sinon
17         // Pas de conflit.
18         si redémmarage alors
19             D ⟵ ⟨⟩
20             S = (F, Γ, D)
21         Choisir un littéral ℓ tel que S ⊭_pu ℓ et S ⊭_pu ¬ℓ
22         si ℓ = null alors
23             retourner D // satisfiable
24         D ⟵ D, ℓ
```

apprentissage par symétrie (MiniSat+SymCDCL) sur plusieurs et diverses instances.

D'abord, nous avons exécuté les deux méthodes sur différentes instances comme les FPGA (Field Programmable Gate Array), Chnl (Allocation de fréquences), Urquhart, S3 (global routing) et Hole (Pigeon) qui possèdent des symétries. Ensuite, nous avons testé ces deux méthodes sur des problèmes de coloration de graphes difficiles. Pour finir, différents problèmes des dernières compétitions SAT sont testés. Nous en comparons les performances en nombre de nœuds et en temps CPU. Le temps nécessaire pour le calcul

			MiniSat		MiniSat+SymCDCL	
Instance	*#V*	*#C*	*Nœuds*	*Temps*	*Nœuds*	*Temps*
chnl10_12	240	1344	2009561	67.26	**28407**	**1.58**
chnl10_13	260	1586	3140061	128.68	**29788**	**1.92**
chnl11_12	264	1476	– – –	>1200	**246518**	**18.44**
chnl11_13	286	1742	– – –	>1200	**185417**	**15.20**
chnl11_20	440	4220	– – –	>1200	**61287**	**7.84**
fpga10_8_sat	120	448	**264**	0.00	463	0.00
fpga10_9_sat	135	549	**250**	**0.00**	494	0.01
fpga12_11_sat	198	968	**421**	**0.00**	982	0.02
fpga12_12_sat	216	1128	403	0.00	**343**	0.00
fpga12_8_sat	144	560	**390**	**0.00**	568	0.01
fpga12_9_sat	162	684	**383**	**0.00**	1089	0.03
fpga13_10_sat	195	905	**499**	**0.00**	2311	0.06
fpga13_12_sat	234	1242	**335**	0.00	336	0.00
fpga13_9_sat	176	759	**408**	**0.00**	603	0.01
hole7	56	204	10123	0.08	**233**	**0.00**
hole8	72	297	40554	0.37	**8323**	**0.15**
hole9	90	415	202160	2.69	**15184**	**0.35**
hole10	110	561	1437244	27.54	**73844**	**2.10**
hole11	132	738	23096626	778.46	**249897**	**9.49**
hole12	156	949	– – –	>1200	**837072**	**43.23**
s3-3-3-10	1056	10862	**31043**	**0.53**	33372	1.45
s3-3-3-1	864	7592	**9746**	**0.12**	11858	0.37
s3-3-3-3	960	9156	21607	0.35	**7518**	**0.23**
s3-3-3-4	912	8356	**35909**	**0.56**	127035	6.02
s3-3-3-8	912	8356	**10871**	**0.17**	98649	5.03
Urq3_5	46	470	9403639	79.09	**1830**	**0.04**
Urq4_5	74	694	– – –	>1200	**18442**	**0.61**
Urq5_5	121	1210	– – –	>1200	**1798674**	**167**
Urq6_5	180	1756	– – –	>1200	– – –	>1200
Urq7_5	240	2194	– – –	>1200	– – –	>1200
Urq8_5	327	3252	– – –	>1200	– – –	>1200
Urq8_5	327	3252	– – –	>1200	– – –	>1200

TABLE 5.1 – Résultats sur quelques instances SAT

des symétries du problème considéré est ajouté au temps CPU total pour MiniSat+SymCDCL. Le code est écrit en C++ et compilé sur un Pentium 4, 2.8 GHZ avec 1 Gb de RAM.

5.4.1 Résultats sur différents problèmes symétriques

Le tableau 5.1 montre la comparaison de notre approche avec MiniSat sur des problèmes SAT connus. Les colonnes nous donnent le nom de l'instance, la taille ($\#V/\#C$), le nombre de nœuds de l'arbre de recherche et le temps CPU utilisé pour résoudre l'instance.

Le tableau 5.1 montre que notre approche MiniSat+SymCDCL est en général meilleure que MiniSat tant en temps qu'en nombre de nœuds sur les problèmes Hole et Chnl. Pour les instances FPGA, nous obtenons des résultats similaires en temps. Ceci est dû au fait que ces problèmes sont satisfiables et MiniSat arrive à trouver une solution rapidement. Les problèmes Urquhart sont plus difficiles que les problèmes FPGA ou les problèmes Chnl. Nous voyons que MiniSat n'est capable de résoudre que le premier problème alors que notre approche permet de les résoudre tous efficacement.

5.4.2 Résultats sur les problèmes de coloration de graphe

Les problèmes de coloration de graphes aléatoires sont générés selon les paramètres suivants :

- le nombre de sommets du graphe (n),
- le nombre de couleurs ($Couleurs$),
- la densité du graphe (d), qui est un réel compris entre 0 et 1 correspondant au ratio du nombre de contraintes du graphe sur le nombre total de contraintes possibles.

Les paramètres n, $Couleurs$ et d étant fixés, nous générons 100 problèmes aléatoirement et les résultats (temps CPU moyen , nombre de nœuds moyen) sont reportés. Le temps limite est fixé à 800 secondes pour résoudre l'instance.

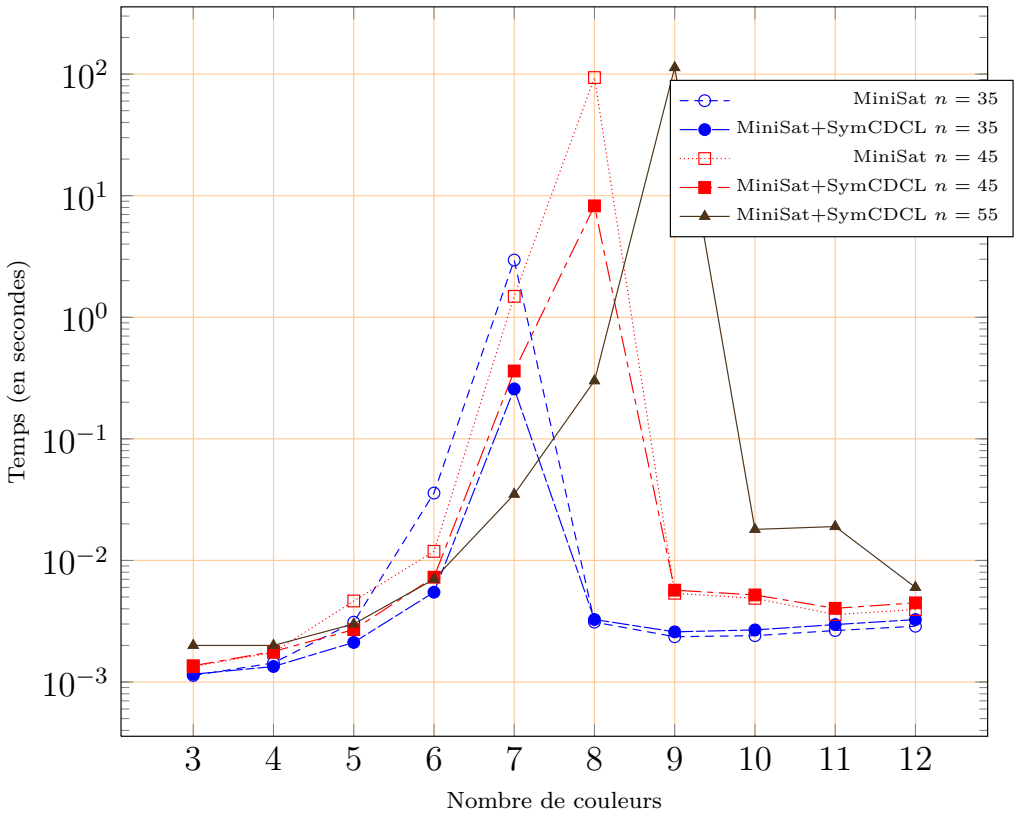

FIGURE 5.1 – Courbes des temps CPU de MiniSat et MiniSat+SymCDCL sur des problèmes de coloration de graphe générés aléatoirement pour $n = \{35, 45, 55\}$ et $d = 0.5$

Nous reportons dans les figures 5.1 et 5.2 les résultats pratiques de MiniSat, et MiniSat+SymCDCL, sur trois classes de problèmes où le nombre de variables (les sommets du graphe) est fixé à 35, 45 et 55 respectivement et où la densité est fixée à ($d = 0.5$). La figure 5.1 représente le temps CPU moyen en fonction du nombre de couleurs pour les deux méthodes et pour chaque classe de problème. La figure 5.2 reporte le nombre moyen de nœuds.

Des deux figures, nous remarquons qu'en moyenne MiniSat+SymCDCL explore moins de nœuds que MiniSat et les résout plus rapidement. L'apprentissage de clauses par symétrie est profitable autour de la région difficile de ces problèmes et notre méthode MiniSat+SymCDCL peut résoudre des problèmes contenant 55 sommets dans le temps imparti, alors que MiniSat n'y arrive pas. Ceci explique que nous ne reportons pas les courbes de MiniSat pour 55 sommets dans les deux figures.

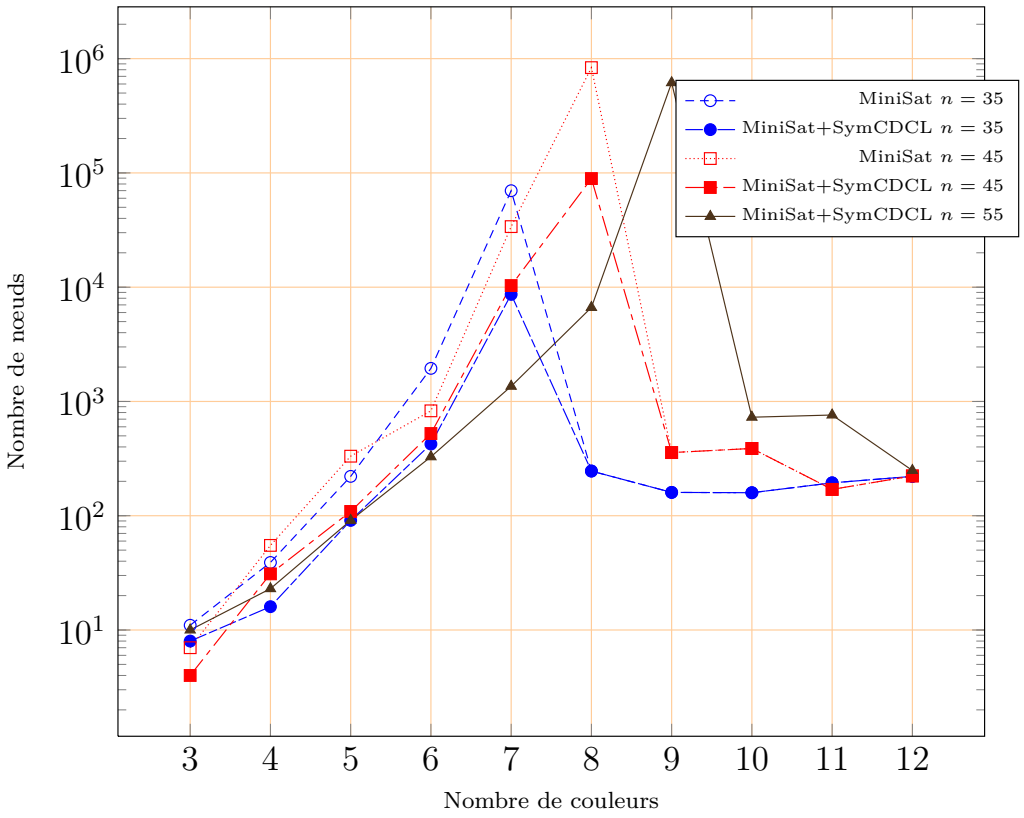

FIGURE 5.2 – Courbes des nœuds de MiniSat et MiniSat+SymCDCL sur des problèmes de coloration de graphe générés aléatoirement pour $n = \{35, 45, 55\}$ et $d = 0.5$

5.4.3 Résultats sur des problèmes des dernières compétitions SAT

Nous présentons les résultats des deux méthodes (MiniSat et MiniSat+SymCDCL) sur 180 classes de problèmes issues des dernières compétitions SAT. Le temps CPU limite est fixé à 1 200 secondes pour la résolution de chaque problème.

Notre premier but est ici de trouver des problèmes contenant des symétries et de regarder le comportement de l'apprentissage par symétrie sur ces derniers. La sélection des problèmes s'est faite en recherchant si le problème comporte des symétries globales. Si c'est le cas, nous la sélectionnons. Sinon nous utilisons le prétraitement de MiniSat et nous testons la présence de symétries sur le problème simplifié.

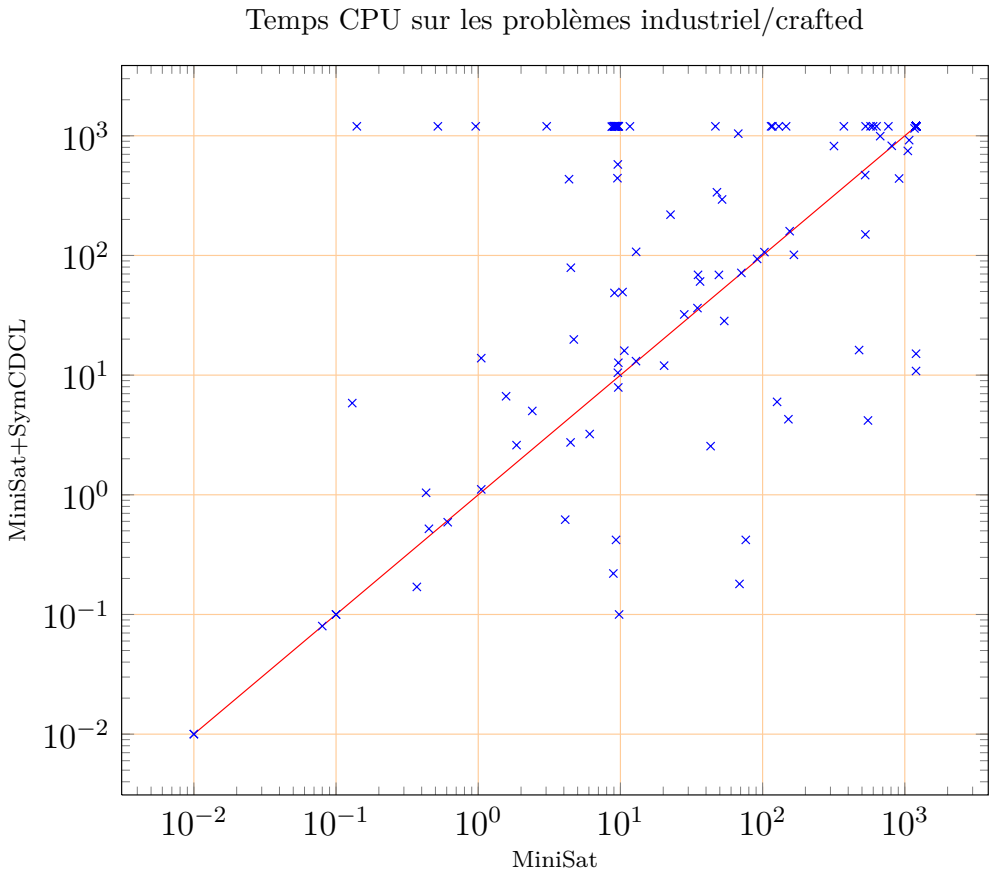

FIGURE 5.3 – Temps CPU sur quelques problèmes SAT

La figure 5.3 montre la comparaison de MiniSat+SymCDCL et MiniSat en temps CPU et la figure 5.4 montre la comparaison en nombre de nœuds sur les 180 problèmes sélectionnés. Sur les deux figures, l'axe des y (resp. l'axe des x) représente les résultats de MiniSat+SymCDCL (resp. MiniSat). Un point sur la figure 5.3 (respectivement la figure 5.4) représente le temps CPU (respectivement le nombre de nœuds) des deux solveurs pour un problème donné.

La projection du point sur l'axe des y (resp. l'axe des x) représente la performance de MiniSat+SymCDCL (resp. MiniSat) en temps CPU pour la figure 5.3 et en nombre de nœuds pour la figure 5.4. Les points en dessous de la diagonale indiquent que MiniSat+SymCDCL est meilleur que MiniSat. Les points aux alentours de la diagonale montrent des résultats similaires et les points au dessus indiquent que MiniSat est meilleur. Les figures 5.3 et 5.4 montrent qu'il existe de nombreux problèmes où MiniSat+SymCDCL est meilleur que MiniSat. Sur certains problèmes, MiniSat est meilleur que MiniSat+SymCDCL en temps CPU. Cela arrive généralement lorsque le

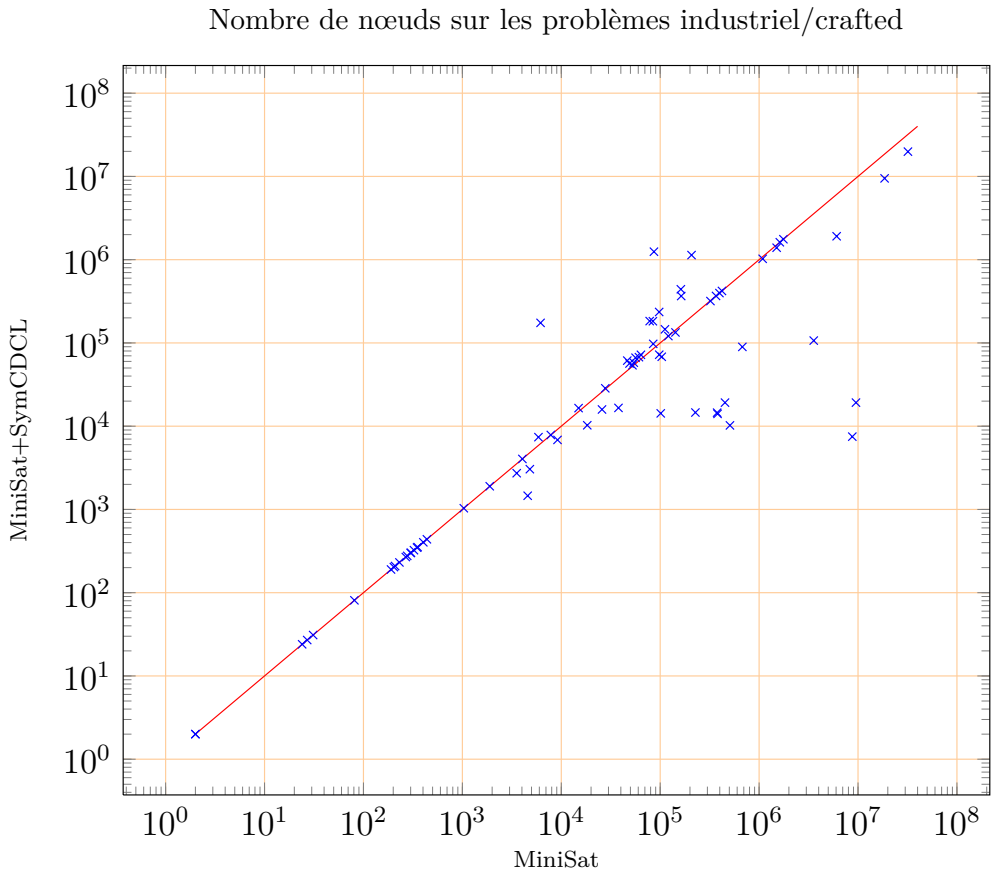

FIGURE 5.4 – Nombre de nœuds sur quelques problèmes SAT

problème admet une solution et que MiniSat trouve rapidement celle-ci.

Le tableau 5.2 donne le détail des résultats des deux méthodes sur certains problèmes issus des dernières compétitions SAT. Nous pouvons voir dans le tableau 5.2 que la considération des symétries influent positivement sur plusieurs problèmes. Cependant, il reste plusieurs instances sur lesquelles MiniSat est meilleur que MiniSat+SymCDCL. Ceci s'explique d'une part par le fait que ces instances sont satisfiables et MiniSat trouve facilement la solution. D'autre part, il reste à faire un travail d'adaptation des heuristiques et paramètres de MiniSat au nouveau schéma d'apprentissage avec symétrie.

			MiniSat		MiniSat+SymCDCL	
Instance	*#V*	*#C*	*Temps*	*Nœuds*	*Temps*	*Nœuds*
acg-10-5p0	123 925	539 763	**154,99**	421 171	159,53	421 171
aprove09-06	77 262	263 137	**35,14**	**56 319**	68,86	66 443
aprove09-10	67 186	245 335	4,46	9 159	**2,74**	**6 842**
aprove09-11	20 192	78 082	**0,45**	**7 838**	0,52	7 843
aprove09-12	27 495	102 011	**1,05**	**27 860**	1,11	28 512

			MiniSat		MiniSat+SymCDCL	
Instance	*#V*	*#C*	*Temps*	*Nœuds*	*Temps*	*Nœuds*
aprove09-13	7 606	26 317	**0,08**	4 043	0,08	4 043
aprove09-19	30 537	112 780	0,61	**1 891**	**0,59**	1 899
aprove09-21	29 964	91 044	**28,12**	**60 986**	32,15	67 428
aprove09-22	11 557	38 505	0,37	4 573	**0,17**	**1 461**
aprove09-24	61 164	209 228	**10,65**	**52 439**	16,01	53 910
aprove09-25	37 821	124 485	**1,05**	**5 878**	13,86	7 386
blocks-4-ipc5-h22-unknown	148 153	937 242	**4,35**	**6 173**	433,68	174 210
cmu-bmc-barrel6	2 306	8 931	4,09	101 246	**0,62**	**14 256**
contest02-mat26.sat05	744	2 464	528,38	6 077 958	**149,83**	**1 911 094**
counting-clqcolor-unsat-set-b	132	1 527	8,92	507 160	**0,22**	**10 222**
counting-easier-php-012-010	120	672	151,89	3 565 803	**4,28**	**106 726**
een-tip-sat-texas-tp-5e	17 985	52 128	**0,43**	**14 958**	1,04	16 492
een-tip-uns-nusmv-t5.b	59 265	170 983	**2,40**	25 804	5,02	**15 876**
gus-md5-04	68 679	223 994	6,08	3 542	**3,22**	**2 724**
gus-md5-05	68 827	224 473	20,29	4 811	**12,00**	**3 044**
gus-md5-06	68 953	224 868	53,74	18 311	**28,35**	**10 256**
gus-md5-07	69 097	225 325	165,98	37 867	**101,22**	**16 583**
gus-md5-09	69 487	226 581	1 050,09	103 672	**747,90**	**68 221**
manol-pipe-c9	15 214	45 082	9,66	97 806	**7,89**	**72 188**
manol-pipe-g10id	159 638	473 596	9,79	170 859	**0,10**	**0**
manol-pipe-g6bi	23 891	69 895	**1,86**	**54 035**	2,60	57 960
manol-pipe-g7nidw	75 496	222 379	9,31	226 984	**0,42**	**14 599**
mod2-3cage-unsat-9-12.sat05	87	232	76,14	9 533 790	**0,42**	**19 204**
mod2-3cage-unsat-9-14.sat05	87	232	68,79	8 754 084	**0,18**	**7 490**
mod2-rand3bip-sat-210-2.sat05	210	840	**70,89**	1 760 256	71,50	1 760 256
mod2c-rand3bip-unsat-120-3.sat05	160	640	**317,55**	31 958 268	820,82	**19 856 764**
mod3block-3vars-9gates-restr	784	209 392	**4,48**	**78 273**	79,00	182 952
partial-5-13-s	184 675	842 981	**9,67**	**63 838**	12,69	72 047
pmg-11-unsat.sat05	169	562	911,78	18 562 286	**439,57**	**9 474 789**
q32ink09	36	7 938	43,08	452 083	**2,55**	**19 195**
q32ink10	45	38 430	126,41	380 172	**5,98**	**14 059**
q32ink11	55	139 590	477,15	376 321	**16,20**	**14 543**
q3ink08	28	1 680	0.00	24	0,00	24
q3ink09	36	15 120	0.01	27	0,01	27
q3ink10	45	75 600	0.10	31	0,10	31
q-query-3-l37-lambda	26 241	128 361	**4,70**	**48 924**	19,86	56 616
q-query-3-l38-lambda	26 986	132 388	**12,90**	**84 339**	107,24	182 241
q-query-3-l39-lambda	27 735	136 449	**10,35**	**85 022**	49,47	97 491
q-query-3-l40-lambda	28 488	140 544	**51,88**	**163 093**	293,73	365 857
q-query-3-l41-lambda	29 245	144 673	**47,73**	**161 772**	337,25	440 925
q-query-3-l42-lambda	30 006	148 836	**67,43**	**207 417**	1 041,98	1 135 417
rbcl-xits-06-unsat	980	47 620	**36,32**	**111 719**	60,60	145 755
rpoc-xits-07-unsat	1 128	63 345	**671,75**	1 086 266	992,96	**1 024 438**
rpoc-xits-12-unknown	1 898	128 735	**9,08**	**46 476**	48,62	61 337
rpoc-xits-17-sat	2 718	215 000	0.10	1 034	0,10	1 034
sat-strips-gripper-12t23.sat05	4 940	93 221	**22,51**	**97 638**	219,20	235 387
ucg-10-5p0	104 481	500 875	**102,97**	397 176	106,70	397 176
ucg-15-5p0	162 696	821 000	526,29	1 503 561	**469,29**	**1 392 041**
unsat-pbl-00090.sat05	322	9 322	**9,59**	677 178	10,47	**89 640**
ur-10-5p0	104 479	500 869	**91,50**	366 803	93,56	366 803
ur-10-5p1	104 856	508 020	**12,87**	120 912	13,09	120 912
ur-15-5p0	162 682	820 958	**809,33**	1 619 470	824,67	1 619 470
uti-10-10p0	130 859	627 223	**9,59**	**86 448**	576,18	1 247 680
velev-engi-uns-1.0-4nd	7 000	67 586	**34,82**	142 665	36,34	**133 675**
velev-live-uns-2.0-ebuf	14 628	161 477	**49,27**	322 116	68,80	**318 384**

TABLE 5.2 – Résultats sur quelques instances des Compétitions SAT

Le tableau 5.3 donne les résultats sur des instances de compétition SAT pour lesquels MiniSat+SymCDCL est meilleur que MiniSat, tant en temps CPU qu'en nombre de nœuds. Nous remarquons notamment que MiniSat+SymCDCL résout l'instance *gus-md5-10* alors que MiniSat n'y parvient pas dans le temps imparti.

			MiniSat		MiniSat+SymCDCL	
Instance	*#V*	*#C*	*Nœuds*	*Temps*	*Nœuds*	*Temps*
cmu-bmc-barrel6	2 306	8 931	101 246	4,09	14 256	0,62
counting-easier-php-012-010	120	672	3 565 803	151,89	106 726	4,28
gus-md5-04	68 679	223 994	3 542	6,08	2 724	3,22
gus-md5-05	68 827	224 473	4 811	20,29	3 044	12,00
gus-md5-06	68 953	224 868	18 311	53,74	10 256	28,35
gus-md5-07	69 097	225 325	37 867	165,98	16 583	101,22
gus-md5-09	69 487	226 581	103 672	1 050,09	68 221	747,90
gus-md5-10	69 503	226 618	– – –	>1200	97 146	1 143,27
mod2-3cage-9-12	87	232	9 533 790	76,14	19 204	0,42
mod2-3cage-9-14	87	232	8 754 084	68,79	7 490	0,18
pmg-11-unsat	169	562	18 562 286	911,78	9 474 789	439,57
Q32inK09	36	7 938	452 083	43,08	19 195	2,55
Q32inK10	45	38 430	380 172	126,41	14 059	5,98
Q32inK11	55	139 590	376 321	477,15	14 543	16,20

TABLE 5.3 – Avantage des symétries sur certains problèmes des compétitions SAT

5.5 Conclusion

Dans ce chapitre, nous avons exploité la symétrie dans l'apprentissage des clauses pour les solveurs CDCL. Nous avons introduit un nouveau schéma d'apprentissage exploitant les symétries (SLS) qui peut être utilisé par n'importe quel solveur SAT à base de CDCL.

L'apprentissage par symétrie permet, en plus de l'ajout de la clause assertive correspondant à un nœud de l'arbre de recherche, d'ajouter toutes les clauses assertives symétriques à celle-ci. Considérer les clauses assertives symétriques permet d'éviter aux solveurs SAT d'explorer des sous-espaces isomorphes. Nous avons implémenté le schéma d'apprentissage par symétrie (SLS) dans MiniSat et avons expérimenté MiniSat et MiniSat avec le schéma SLS sur une grande variété de problèmes. Les résultats expérimentaux obtenus sont très encourageants, et montrent que l'utilisation de la symétrie dans l'apprentissage est profitable sur la plupart des problèmes testés.

Chapitre 6

Symétries dans les logiques non monotones

Sommaire

6.1 Introduction 83
6.2 Symétrie dans la logique préférentielle 84
6.3 Symétries dans les $X-$Logiques 86
6.4 Symétrie dans la logique des défauts 88
6.5 Conclusion 93

6.1 Introduction

Dans ce chapitre, nous étendons la notion de la symétrie à des logiques non classiques telles que les logiques préférentielles [Bossu and Siegel, 1982, Bossu and Siegel, 1985, SHOAM, 1987, Besnard and Siegel, 1988, Kraus et al., 1990], les $X-$logiques [Siegel et al., 2001] et les logiques des défauts [Reiter, 1980]. Nous donnons ensuite de nouvelles règles d'inférence par symétrie pour les $X-$logiques et les logiques des défauts. Enfin, nous montrons comment raisonner par symétrie dans ces logiques et nous mettons en évidence l'existence de certaines symétries dans ces logiques non classiques qui n'existent pas dans les logiques classiques [Benhamou et al., 2010g, Benhamou et al., 2010h, Benhamou et al., 2010f].

Le reste du chapitre est organisé comme suit : Nous étudions tout d'abord la symétrie dans les logiques préférentielles, ensuite dans le cadre du formalisme des $X-$logiques. Enfin, nous étudions la symétrie dans la logique des défauts.

Pour des rappels sur les permutations et les groupes symétriques, on peut se référer au chapitre 3 (page 43). La section 4.2 (page 54) définit la symétrie

en logique propositionnelle et donne les résultats théoriques sur la symétrie.

Avant d'aborder la symétrie dans les logiques non monotones, nous rappelons *la règle de la symétrie* en logique propositionnelle introduite par Krishnamurthy pour améliorer le système de preuves par résolution.

Proposition 6.1.1.
Si LP est la logique propositionnelle, A un ensemble de formules de LP, B une formule de LP et σ une symétrie syntaxique de A, alors la règle de la symétrie est définie comme suit :

$$\frac{A \vdash B}{A \vdash \sigma(B)}$$

Plusieurs problèmes difficiles pour la résolution classique ont été démontrés avoir des preuves polynomiales en utilisant la symétrie dans la résolution.

Maintenant, nous décrivons la contribution principale de ce chapitre qui consiste à étendre la symétrie aux logiques non monotones.

6.2 Symétrie dans la logique préférentielle

Ici, nous étendons la définition de la symétrie sémantique à la logique préférentielle et on montre comment des littéraux peuvent être symétriques dans cette logique non classique,alors qu'ils ne le sont pas en logique classique.

Définition 6.2.1 (Symétrie préférentielle sémantique).
Si $\models_{\prec}$ est une inférence de modèles préférentiels, A un ensemble de formules et σ une permutation définie sur les littéraux de A, alors σ est une symétrie de A, si et seulement si A et $\sigma(A)$ ont le même ensemble de modèles minimaux.

Définition 6.2.2.
Deux littéraux ℓ et ℓ' sont symétriques dans A si et seulement si, il existe une symétrie préférentielle sémantique σ de A tel que $\sigma(\ell) = \ell'$.

Exemple 6.2.1.
Si on reprend l'exemple 2.2.1 (page 32) qui représente l'information « en général les étudiants sont jeunes et Léa est un étudiant » auquel on ajoute le fait que « John est un étudiant », alors nous obtenons la formule suivante :

$$\begin{aligned} A' \;=\; \Big\{ &\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \Big(Anormal(\text{Léa}) \vee Jeune(\text{Léa})\Big), \\ &\Big(Anormal(John) \vee Jeune(John)\Big)\Big\} \end{aligned}$$

qui admet neuf modèles :

$$\begin{aligned}
M_1 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \quad Anormal(\text{Léa}), \quad Jeune(\text{Léa}), \\
&\quad Anormal(John), \quad Jeune(John)\} \\
M_2 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \quad Anormal(\text{Léa}), \quad Jeune(\text{Léa}), \\
&\quad \neg Anormal(John), \quad Jeune(John)\} \\
M_3 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \quad Anormal(\text{Léa}), \quad Jeune(\text{Léa}), \\
&\quad Anormal(John), \neg Jeune(John)\} \\
M_4 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \neg Anormal(\text{Léa}), \quad Jeune(\text{Léa}), \\
&\quad Anormal(John), \quad Jeune(John)\} \\
M_5 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \neg Anormal(\text{Léa}), \quad Jeune(\text{Léa}), \\
&\quad \neg Anormal(John), \quad Jeune(John)\} \\
M_6 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \neg Anormal(\text{Léa}), \quad Jeune(\text{Léa}), \\
&\quad Anormal(John), \neg Jeune(John)\} \\
M_7 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \quad Anormal(\text{Léa}), \neg Jeune(\text{Léa}), \\
&\quad Anormal(John), \quad Jeune(John)\} \\
M_8 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \quad Anormal(\text{Léa}), \neg Jeune(\text{Léa}), \\
&\quad \neg Anormal(John), \quad Jeune(John)\} \\
M_9 &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \quad Anormal(\text{Léa}), \neg Jeune(\text{Léa}), \\
&\quad Anormal(John), \neg Jeune(John)\}
\end{aligned}$$

Il est facile de voir que les personnes Léa et John sont symétriques dans les deux logiques : classique et préférentielle. En effet, chaque littéral où Léa apparaît est symétrique au littéral que l'on obtient en remplaçant John par Léa. Si nous considérons le fait d'être anormal comme une information pertinente sur laquelle elle est basée la préférence et si on a la permutation :

$$\begin{aligned}
\sigma &= \Big(\text{É}tudiant(\text{Léa}), \text{É}tudiant(John)\Big)\Big(Anormal(\text{Léa}), Anormal(John)\Big) \\
&\quad \Big(Jeune(\text{Léa}), Jeune(John)\Big)
\end{aligned}$$

alors on peut facilement voir que σ est une symétrie préférentielle sémantique de la formule A'. En effet, il existe un modèle minimal, parmi les neuf modèles, qui est :

$$\begin{aligned}
M &= \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \neg Anormal(\text{Léa}), \neg Anormal(John), \\
&\quad Jeune(\text{Léa}), Jeune(John)\}
\end{aligned}$$

et qui est conservé par σ. Nous pouvons voir que les littéraux $Jeune(\text{Léa})$ et $Jeune(John)$ sont symétriques ainsi que les littéraux $Anormal(\text{Léa})$ et $Anormal(John)$.

Maintenant, si l'on ajoute à A' l'information « John n'est pas anormal », nous obtenons la formule :

$$A'' = \Big\{\text{É}tudiant(Léa), \text{É}tudiant(John), \Big(Anormal(Léa) \vee Jeune(Léa)\Big), \Big(Anormal(John) \vee Jeune(John)\Big), \neg Anormal(John)\Big\}$$

où les littéraux $Jeune(Léa)$ et $Jeune(John)$ ainsi que les littéraux $\neg Anormal(Léa)$ et $\neg Anormal(John)$ restent symétriques deux à deux en logique préférentielle, puisque M reste le seul modèle minimal de A''. Cependant, les littéraux $Jeune(Léa)$ et $Jeune(John)$ ne sont pas symétriques dans la logique classique. En effet, la nouvelle formule A'' contient les trois modèles étendus suivants :

$$\begin{aligned}
M'_1 &= \{\text{É}tudiant(Léa), \text{É}tudiant(John), \quad Anormal(Léa), \quad Jeune(Léa), \\
&\quad \neg Anormal(John), \quad Jeune(John)\} \\
M'_2 &= \{\text{É}tudiant(Léa), \text{É}tudiant(John), \neg Anormal(Léa), \quad Jeune(Léa), \\
&\quad \neg Anormal(John), \quad Jeune(John)\} \\
M'_3 &= \{\text{É}tudiant(Léa), \text{É}tudiant(John), \quad Anormal(Léa), \neg Jeune(Léa), \\
&\quad \neg Anormal(John), \quad Jeune(John)\}
\end{aligned}$$

où ces littéraux ne sont pas symétriques. Nous pouvons voir par exemple que le modèle M'_3 ne reste pas un modèle si l'on permute $Jeune(Léa)$ et $Jeune(John)$.

Il est alors important de noter que certains littéraux pourraient être symétriques dans une approche préférentielle, mais non symétriques dans une logique classique. Cette idée est nouvelle et prometteuse pour le raisonnement en utilisant la symétrie dans des logiques non classiques.

Dans la section suivante, nous essayons d'étendre la notion de la symétrie syntaxique. Pour ce faire nous avons choisi les $X-$logiques [Siegel et al., 2001] comme un cadre de référence.

6.3 Symétries dans les $X-$Logiques

Nous avons vu qu'il est facile d'étendre la notion de la symétrie sémantique aux logiques préférentielles, mais la définition de la symétrie syntaxique dans ces logiques semble ne pas être triviale.

Nous étendrons la définition de la symétrie syntaxique en logique classique au cadre des $X-$logiques et donnerons une règle étendue de la symétrie qui peut être utilisés pour faire des courtes démonstrations en utilisant des formules symétriques dans ce cadre.

Définition 6.3.1 (Symétrie syntaxique dans les $X-$logiques).
Soient A un ensemble de formules de la logique propositionnelle, X le sous-ensemble de formules pertinentes sur lesquels il est construite l'inférence $\underset{X}{\vdash}$ dans la $X-$logique et σ une permutation de littéraux. La permutation σ est une symétrie syntaxique de A dans la $X-$logique considérée, si les conditions suivantes sont valables :

1. $\sigma(A) = A$,
2. $\sigma(X) = X$.

Maintenant, nous étendons la règle de la symétrie de Krishnamurthy au cadre des $X-$logiques.

Proposition 6.3.1.
Soient A et B deux formules ou deux ensembles de formules et σ une symétrie syntaxique de A dans la $X-$logique considérée. Nous avons la règle suivante :

$$\frac{A \underset{X}{\vdash} B}{A \underset{X}{\vdash} \sigma(B)}$$

Démonstration. Pour prouver que $A \underset{X}{\vdash} \sigma(B)$ nous allons prouver que $\overline{\Big(A \cup \sigma(B)\Big)} \cap X = \overline{A} \cap X$. Nous avons par hypothèse que $A \underset{X}{\vdash} B$, d'où on a $\overline{(A \cup B)} \cap X = \overline{A} \cap X$. Puisque σ est une symétrie syntaxique de A dans la $X-$logique, alors elle préserve les théorèmes de la logique propositionnelle. Ainsi, nous avons $\sigma\Big(\overline{(A \cup B)} \cap X\Big) = \sigma(\overline{A} \cap X)$, d'où on a $\sigma\Big(\overline{(A \cup B)}\Big) \cap \sigma(X) = \sigma(\overline{A}) \cap \sigma(X)$, ce qui est équivalent à $\overline{\sigma(A \cup B)} \cap \sigma(X) = \overline{\sigma(A)} \cap \sigma(X)$. Cela donne $\overline{\Big(\sigma(A) \cup \sigma(B)\Big)} \cap \sigma(X) = \overline{\sigma(A)} \cap \sigma(X)$. Comme A et X sont invariantes par σ, alors on en déduit $\overline{\Big(A \cup \sigma(B)\Big)} \cap X = \overline{A} \cap X$. Par conséquent, $A \underset{X}{\vdash} \sigma(B)$. □

Exemple 6.3.1.
Reprenons l'exemple précédent des étudiants codé par l'ensemble de formules suivant :

$$\begin{aligned} A' \;=\; \Big\{ & \text{É}tudiant(\text{Léa}), \text{É}tudiant(John), \Big(Anormal(\text{Léa}) \vee Jeune(\text{Léa})\Big), \\ & \Big(Anormal(John) \vee Jeune(John)\Big)\Big\} \end{aligned}$$

Considérons l'ensemble $X = \{\neg Anormal(\text{Léa}), \neg Anormal(John)\}$ d'informations pertinentes sur lequel est construite la $X-$logique et la

permutation :

$$\sigma = \Big(\text{É}tudiant(\text{L}éa), \text{É}tudiant(John)\Big)\Big(Anormal(\text{L}éa), Anormal(John)\Big)\Big(Jeune(\text{L}éa), Jeune(John)\Big)$$

La permutation σ est une symétrie de A' dans la $X-$logique. Nous avons $A' \underset{X}{\vdash} Jeune(\text{L}éa)$, et par la symétrie, nous avons $A' \underset{X}{\vdash} Jeune(John)$ puisque $\sigma\Big(Jeune(\text{L}éa)\Big) = Jeune(John)$.

Cette règle peut être utilisée pour déduire toutes les formules symétriques d'une formule inférée dans la $X-$logique considérée. Sa mise en œuvre dans un démonstrateur permettra de raccourcir la preuve d'un théorème.

Dans la prochaine section, nous étendons la notion de symétrie à une logique plus générale et bien connue dans le domaine du raisonnement non monotone, qui est la logique des défauts introduite par Reiter [Reiter, 1980].

6.4 Symétrie dans la logique des défauts

Maintenant, nous introduisons la notion de symétrie dans la logique des défauts et on montre comment l'inférence logique associée est améliorée par la propriété de la symétrie. Pour cette logique, nous distinguons deux niveaux de symétrie : la symétrie sémantique et la symétrie syntaxique. On va définir dans ce qui suit les deux symétries et on étudiera par la suite leur relation.

Définition 6.4.1 (Symétrie sémantique).
Soit une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ où $L_{\mathscr{T}}$ est l'ensemble de ses littéraux et $\mathscr{E}_{\mathscr{T}}$ l'ensemble de toutes ses extensions. Une symétrie sémantique σ est une permutation de littéraux définie sur $L_{\mathscr{T}}$ telle que $\mathscr{E}_{\mathscr{T}} = \mathscr{E}_{\sigma(\mathscr{T})}$.

En d'autres termes, une symétrie sémantique d'une théorie des défauts $\mathscr{T}$ est une permutation de littéraux qui laisse invariant l'ensemble de ses extensions. Il résulte de là que chaque extension $\mathcal{E}_i \in \mathscr{E}_{\mathscr{T}}$ est transformée par la symétrie σ à une autre extension $\mathcal{E}_j = \sigma(\mathcal{E}_i)$. Ces extensions sont ce que nous appelons *extensions symétriques*. Il est alors possible d'obtenir une famille d'extensions symétriques, sans duplication d'effort, si nous savons que $\mathcal{E}_i$ est une extension et nous avons le groupe de symétrie de la théorie $\mathscr{T}$. Malheureusement, le calcul de la symétrie sémantique est coûteux en termes de temps, car on a besoin de calculer toutes les extensions.

Dans ce qui suit, nous allons définir la symétrie syntaxique et on montre qu'il y a une condition suffisante pour cette symétrie qui pourrait être calculée d'une manière plus efficace.

Définition 6.4.2 (Symétrie syntaxique).
Étant donnée une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$. Une symétrie syntaxique est une permutation σ définie sur l'ensemble $L_{\mathscr{T}}$ de littéraux de $\mathscr{T}$, qui laisse la théorie $\mathscr{T}$ invariante, c'est-à-dire $\sigma(\mathscr{T}) = \mathscr{T}$. Plus précisément, les conditions suivantes sont vérifiées : $\sigma(\mathscr{D}) = \mathscr{D}$ et $\sigma(\mathscr{W}) = \mathscr{W}$.

Exemple 6.4.1.
Prenons, á nouveau, l'exemple des étudiants discuté plus haut et considérons la théorie des défauts suivante :

$$\mathscr{T} = \left(\left\{ \frac{\text{É}tudiant(X) \ : \ \neg Anormal(X)}{Jeune(X)} \right\}, \left\{ \text{É}tudiant(\text{Léa}), \text{É}tudiant(John) \right\} \right)$$

La permutation :

$$\sigma \ = \ \Big(\text{É}tudiant(\text{Léa}), \text{É}tudiant(John) \Big) \Big(Anormal(\text{Léa}), Anormal(John) \Big) \Big(Jeune(\text{Léa}), Jeune(John) \Big)$$

est une symétrie syntaxique de $\mathscr{T}$, car elle la laisse invariante. En considérant toutes les instanciations terminales des variables libres des défauts de la théorie $\mathscr{T}$, nous pouvons voir que $\mathscr{T}$ a une seule extension :

$$\mathcal{E} = \{\text{É}tudiant(\text{Léa}), \text{É}tudiant(John), Jeune(\text{Léa}), Jeune(John)\}$$

qui reste invariante sous σ. On conclut que σ est aussi une symétrie sémantique.

Maintenant, si nous ajoutons $Jeune(John)$ à l'ensemble des faits de la théorie $\mathscr{T}$ de l'exemple précédent, nous obtenons une nouvelle théorie $\mathscr{T}'$ pour laquelle σ n'est pas une symétrie syntaxique. Cependant, σ reste une symétrie sémantique de $\mathscr{T}'$, puisque la permutation σ vérifie $\mathscr{E}_{\mathscr{T}'} = \mathscr{E}_{\sigma(\mathscr{T}')} = \{\mathcal{E}\}$. Cet exemple illustre bien le fait que la notion de symétrie sémantique est plus générale que la notion de symétrie syntaxique.

Nous donnons dans le théorème suivant la relation entre la symétrie syntaxique et la symétrie sémantique d'une logique des défauts.

Théorème 6.4.1.
Étant donnée une théorie des défauts $\mathscr{T}$. Si σ est une symétrie syntaxique de $\mathscr{T}$, alors σ est une symétrie sémantique de $\mathscr{T}$.

Démonstration. La preuve est évidente. Puisque σ est une symétrie syntaxique, alors $\sigma(\mathscr{T}) = \mathscr{T}$. Donc, $\mathscr{E}_{\mathscr{T}} = \mathscr{E}_{\sigma(\mathscr{T})}$. □

Maintenant, nous pouvons introduire la nouvelle règle d'inférence par symétrie dans les logiques des défauts. Prenons, par exemple, l'inférence sceptique $\underset{S}{\vdash}$.

Proposition 6.4.1.
Soient $\mathscr{T}$ est une théorie des défauts, f une formule et σ une symétrie syntaxique de $\mathscr{T}$, alors on a la règle suivante :

$$\frac{\mathscr{T} \underset{S}{\vdash} f}{\mathscr{T} \underset{S}{\vdash} \sigma(f)}$$

Démonstration. Pour montrer que $\mathscr{T} \underset{S}{\vdash} \sigma(f)$, on a besoin de prouver que $\forall \mathcal{E} \in \mathscr{E}_{\mathscr{T}}$, $\mathcal{E} \vdash \sigma(f)$.

Soit $\mathcal{E} \in \mathscr{E}_{\mathscr{T}}$. Par la définition de la symétrie σ, il existe $\mathcal{E}' \in \mathscr{E}_{\mathscr{T}}$ telle que $\mathcal{E} = \sigma(\mathcal{E}')$. Par hypothèse on a $\mathscr{T} \underset{S}{\vdash} f$, alors $\mathcal{E}' \vdash f$ d'où on a $\sigma(\mathcal{E}') \vdash \sigma(f)$. Par conséquent, $\mathcal{E} \vdash \sigma(f)$. Alors, on conclut que $\mathscr{T} \underset{S}{\vdash} \sigma(f)$. □

Remarque 6.4.1. La règle précédente est également valable pour l'inférence crédule ($\underset{C}{\vdash}$) et aussi pour l'inférence semi-crédule ($\underset{SC}{\vdash}$).

Exemple 6.4.2.
Prenons l'exemple 6.4.1. Dans la théorie $\mathscr{T}$, nous avons $\mathscr{T} \underset{S}{\vdash} Jeune(Léa)$. En appliquant la règle d'inférence par symétrie, on peut déduire que $\mathscr{T} \underset{S}{\vdash} \sigma(Jeune(Léa))$ et donc $\mathscr{T} \underset{S}{\vdash} Jeune(John)$.

Maintenant, nous donnons une proposition importante qui utilise la symétrie pour calculer l'ensemble des extensions.

Proposition 6.4.2.
Étant donnée une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$, un sous-ensemble de formules $\mathcal{E}$ et une symétrie syntaxique σ de $\mathscr{T}$, alors $\mathcal{E}$ est une extension de $\mathscr{T}$ si et seulement si $\sigma(\mathcal{E})$ est une extension de $\mathscr{T}$.

Démonstration. Supposons que $\mathcal{E}$ soit une extension. En utilisant le théorème 6.4.1 et comme la permutation σ est une symétrie syntaxique, alors σ est une symétrie sémantique. Alors la symétrie σ conserve l'ensemble $\mathscr{E}_{\mathscr{T}}$ des extensions de $\mathscr{T}$. En conséquence, $\sigma(\mathcal{E})$ est une extension de $\mathscr{T}$.

On peut prouver l'inverse de la même manière en considérant la symétrie inverse σ^{-1}. □

Maintenant, nous discutons la relation entre les extensions symétriques d'une théorie des défauts et leurs sous-ensembles des défauts qui sont utilisés pour les construire.

Proposition 6.4.3.
Étant donnée une théorie des défauts $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$, un sous-ensemble $\mathscr{D}' \subset \mathscr{D}$ et une symétrie syntaxique σ, alors il existe une extension $\mathcal{E}^{\mathscr{D}'}$ de $\mathscr{T}$ obtenue par l'application des défauts de $\mathscr{D}'$, si et seulement si, il existe une extension $\mathcal{E}^{\sigma(\mathscr{D}')}$ de $\mathscr{T}$ obtenue par l'application des défauts de $\sigma(\mathscr{D}')$.

Démonstration. Supposons que $\mathcal{E}^{\mathscr{D}'}$ soit une extension de $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ obtenue par l'application des défauts de $\mathscr{D}'$. Nous allons montrer qu'il existe une extension $\mathcal{E}^{\sigma(\mathscr{D}')}$ de $\mathscr{T}$ qui peut être obtenue par l'application des défauts de $\sigma(\mathscr{D}')$.

Puisque $\mathcal{E}^{\mathscr{D}'}$ est une extension de $\mathscr{T}$ et à partir de la définition 2.2.8, on a $\mathcal{E}^{\mathscr{D}'} = \bigcup_{i=0,\ldots,\infty} \mathcal{E}_i$ où : $\mathcal{E}_0 = \mathscr{W}$ et pour tous $i \geq 0$ on a

$\mathcal{E}_{i+1} = Th(\mathcal{E}_i) \cup \Big\{ \gamma \;\Big|\; \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in \mathscr{D}', \text{ où } \alpha \in \mathcal{E}_i \text{ et } \forall j \in \{1, \ldots, n\}, \neg\beta_j \notin \mathcal{E}^{\mathscr{D}'} \Big\}$.

Pour montrer l'existence de l'extension $\mathcal{E}^{\sigma(\mathscr{D}')}$ de $\mathscr{T}$, on pose :

- $\mathcal{E}' = \sigma(\mathcal{E}^{\mathscr{D}'})$,
- $\mathcal{E}'_0 = \mathscr{W}$ et
- pour tous $i \geq 0$ on a

  $\mathcal{E}'_{i+1} = Th(\mathcal{E}'_i) \cup$
  $\Big\{ \sigma(\gamma) \;\Big|\; \frac{\sigma(\alpha) : \sigma(\beta_1), \ldots, \sigma(\beta_n)}{\sigma(\gamma)} \in \sigma(\mathscr{D}'), \sigma(\alpha) \in \mathcal{E}'_i, \forall j \in \{1, \ldots, n\}, \neg\sigma(\beta_j) \notin \mathcal{E}' \Big\}$,

et on va prouver que $\mathcal{E}'$ est une extension de $\mathscr{T}$ et $\mathcal{E}' = \mathcal{E}^{\sigma(\mathscr{D}')}$.

Pour montrer que $\mathcal{E}'$ est une extension de $\mathscr{T}$, il faut montrer que $\mathcal{E}' = \bigcup_{i=0,\ldots,\infty} \mathcal{E}'_i$. Pour ce faire, il faut d'abord montrer que $\forall i \geq 0$, $\mathcal{E}'_i = \sigma(\mathcal{E}_i)$. On va démontrer cette propriété par récurrence sur i.

Pour la première étape ($i = 0$), on a $\mathcal{E}'_0 = \mathscr{W}$. Comme σ est une symétrie syntaxique de $\mathscr{T}$, alors $\mathscr{W} = \sigma(\mathscr{W})$. Il en résulte que $\mathcal{E}'_0 = \sigma(\mathscr{W}) = \sigma(\mathcal{E}_0)$.

Maintenant, on suppose que la propriété est vérifiée jusqu'à l'étape i, c'est-à-dire $\mathcal{E}'_i = \sigma(\mathcal{E}_i)$ et on va montrer que cette propriété est vérifiée à l'étape $i+1$, c'est-à-dire $\mathcal{E}'_{i+1} = \sigma(\mathcal{E}_{i+1})$.

Par la définition de $\mathcal{E}'_{i+1}$, on a

$\mathcal{E}'_{i+1} = Th(\mathcal{E}'_i) \cup$
$\Big\{ \sigma(\gamma) \;\Big|\; \frac{\sigma(\alpha) : \sigma(\beta_1), \ldots, \sigma(\beta_n)}{\sigma(\gamma)} \in \sigma(\mathscr{D}'), \sigma(\alpha) \in \mathcal{E}'_i, \forall j \in \{1, \ldots, n\}, \neg\sigma(\beta_j) \notin \mathcal{E}' \Big\}$,

et par l'hypothèse de récurrence et la définition de $\mathcal{E}'$, on peut réécrire $\mathcal{E}'_{i+1}$ comme :

$\mathcal{E}'_{i+1} = Th\Big(\sigma(\mathcal{E}_i)\Big) \cup$
$\Big\{ \sigma(\gamma) \;\Big|\; \frac{\sigma(\alpha) : \sigma(\beta_1), \ldots, \sigma(\beta_n)}{\sigma(\gamma)} \in \sigma(\mathscr{D}'), \sigma(\alpha) \in \sigma(\mathcal{E}_i), \forall j \in \{1, \ldots, n\}, \neg\sigma(\beta_j) \notin \sigma(\mathcal{E}^{\mathscr{D}'}) \Big\}$.

D'autre part on a $\sigma\Big(Th(\mathcal{E}_i)\Big) = Th\Big(\sigma(\mathcal{E}_i)\Big)$ puisque la symétrie σ vérifie la propriété suivante : si $A \vdash B$, alors $\sigma(A) \vdash \sigma(B)$. Par conséquent,

$\mathcal{E}'_{i+1} = \sigma\Big(Th(\mathcal{E}_i)\Big) \cup \sigma\Big(\Big\{ \gamma \;\Big|\; \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in \mathscr{D}', \text{ où } \alpha \in \mathcal{E}_i \text{ et } \forall j \in \{1, \ldots, n\}, \neg\beta_j \notin \mathcal{E}^{\mathscr{D}'} \Big\}\Big)$.

On en déduit que

$\mathcal{E}'_{i+1} = \sigma\Big(Th(\mathcal{E}_i) \cup \Big\{ \gamma \;\Big|\; \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in \mathscr{D}', \alpha \in \mathcal{E}_i, \forall j \in \{1, \ldots, n\}, \neg\beta_j \notin \mathcal{E}^{\mathscr{D}'} \Big\}\Big)$.

Cela implique que $\mathcal{E}'_{i+1} = \sigma(\mathcal{E}_{i+1})$ et alors on a prouvé la propriété désirée.

Maintenant, on va prouver que $\mathcal{E}' = \bigcup_{i=0,\ldots,\infty} \mathcal{E}'_i$ et donc $\mathcal{E}'$ est une extension

de $\mathscr{T}$. Par la définition de $\mathcal{E}'$ on a $\mathcal{E}' = \sigma(\mathcal{E}^{\mathscr{D}'})$ et comme $\mathcal{E}^{\mathscr{D}'}$ est par l'hypothèse une extension de $\mathscr{T}$, alors $\mathcal{E}^{\mathscr{D}'} = \bigcup_{i=0,\ldots,\infty} \mathcal{E}_i$. On en conclut que $\mathcal{E}' = \sigma\Big(\bigcup_{i=0,\ldots,\infty} \mathcal{E}_i\Big)$, donc $\mathcal{E}' = \bigcup_{i=0,\ldots,\infty} \sigma(\mathcal{E}_i)$. Comme on a montré que $\forall i \geq 0$, $\mathcal{E}'_i = \sigma(\mathcal{E}_i)$, alors $\mathcal{E}' = \bigcup_{i=0,\ldots,\infty} \mathcal{E}'_i$. Il en résulte que $\mathcal{E}'$ est une extension de $\mathscr{T}$.

Comme $\mathcal{E}'$ est une extension de $\mathscr{T}$ obtenue par l'application des défauts de $\sigma(\mathscr{D}')$, alors $\mathcal{E}' = \mathcal{E}^{\sigma(\mathscr{D}')}$. C'est-à-dire, on conclut qu'il existe une extension $\mathcal{E}^{\sigma(\mathscr{D}')}$ de $\mathscr{T}$ qui peut être obtenue par l'application des défauts de $\sigma(\mathscr{D}')$.

L'inverse peut être établi de la même manière en considérant la symétrie inverse σ^{-1} de σ. □

Maintenant, nous allons montrer que si l'application d'un défaut conduit à une extension de la théorie considérée, alors l'application de chacun de ses défauts symétriques conduit aussi à une extension de la même théorie.

Proposition 6.4.4.
Si $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ est une théorie des défauts et σ une de ses symétries syntaxiques. Il existe une extension de $\mathscr{T}$ où le défaut d est appliqué, si et seulement si, il existe une extension où le défaut symétrique $\sigma(d)$ est appliqué.

Démonstration. Soit $\mathcal{E}$ est une extension de $\mathscr{T}$ où d est appliqué. Cela implique qu'il existe un sous-ensemble $\mathscr{D}' \subset \mathscr{D}$ tel que $d \in \mathscr{D}'$ et $\mathcal{E} = \mathcal{E}^{\mathscr{D}'}$. Par la proposition 6.4.3, on en déduit que $\mathcal{E}^{\sigma(\mathscr{D}')}$ est une extension de $\mathscr{T}$, où $\sigma(d) \in \sigma(\mathscr{D}')$ est appliqué. □

De la proposition précédente, on déduit la propriété suivante :

Corollaire 6.4.1.
Soient $\mathscr{T} = \langle \mathscr{D}, \mathscr{W} \rangle$ une théorie des défauts, $d \in \mathscr{D}$ un défaut et σ une symétrie syntaxique de $\mathscr{T}$. S'il n'existe aucune extension de $\mathscr{T}$ où le défaut d est appliqué, alors il n'existe aucune extension de $\mathscr{T}$ où le défaut symétrique $\sigma(d)$ sera appliqué.

Dans la logique des défauts, nous avons l'avantage, car les deux symétries (syntaxiques et sémantiques) sont définies. La logique des défauts est augmentée avec une règle d'inférence par symétrie qui peut être utilisée pour raccourcir les preuves. Par ailleurs, dans la logique des défauts, la symétrie peut être utilisée pour calculer l'ensemble des extensions. En effet, lorsque l'extension est identifiée lors de l'énumération, on peut déduire toutes ses extensions symétriques, sans effort supplémentaire (Proposition 6.4.2). En d'autres termes les branches isomorphes dans l'arbre de recherche correspondant à l'application des sous-ensembles symétriques des défauts conduisent à des extensions symétriques. Nous avons alors besoin d'explorer

qu'une seule branche et élaguer l'espace de recherche correspondant aux autres. En cas d'échec pour obtenir une extension lors de l'application d'un défaut d, le corollaire 6.4.1 nous permet de couper tous les sous-espaces de recherche correspondant aux défauts symétriques $\sigma(d)$, car ils ne conduisent pas à des extensions.

6.5 Conclusion

Le but principal de ces travaux est d'étendre la notion de la symétrie aux formalismes logiques non classiques. Nous avons défini la notion de la symétrie sémantique dans les logiques préférentielles et nous avons montré que des symétries existent dans ces logiques non classiques et qui n'apparaissent pas dans les logiques classiques. L'autre point étudié ici est l'extension de la définition de la symétrie syntaxique dans le cadre des $X-$logiques où une nouvelle règle d'inférence a été introduite. Enfin, nous avons défini à la fois, la symétrie sémantique et syntaxique dans le cadre plus général de la logique des défauts. Nous avons montré comment la symétrie peut être utilisée pour améliorer la recherche des extensions et on a introduit une nouvelle règle d'inférence par symétrie valabe pour les différentes types d'inférence connues pour la logique des défauts et qui peut être utilisée pour trouver de courtes preuves.

Conclusion et perspectives

Conclusion

Nous avons proposé dans cette thèse différentes contributions à fin de bénéficier de la force de la symétrie dans le problème SAT et les logiques non monotones.

Une première contribution consistait à utiliser l'élimination des symétries locales dans les algorithmes de résolution du problème SAT. Les symétries locales sont une généralisation des symétries globales à chaque sous formule CNF en chaque nœud de l'arbre de recherche. Cette sous formule est la formule initiale elle-même mais en considérant l'interprétation partielle correspondant à ce nœud. Nous avons représenté chaque formule $\mathcal{F}$ en CNF par un graphe $G_{\mathcal{F}}$. Une propriété importante du graphe $G_{\mathcal{F}}$ est qu'il préserve le groupe de symétries syntaxiques de $\mathcal{F}$. Nous avons adapté un outil de calcul des automorphismes de graphes (nous avons utilisé Saucy) pour calculer ces symétries locales en maintenant dynamiquement le graphe correspondant à la sous formule définie en chaque nœud de l'arbre de recherche.

On donne à Saucy en entrée le graphe de la sous formule locale et il nous retourne alors l'ensemble des générateurs du groupe d'automorphismes du graphe donné. Cet ensemble est équivalent à l'ensemble des générateurs de groupe de symétries locales de la sous formule considérée.

Ensuite, nous avons implémenté cette méthode de détection et d'élimination des symétries locales dans le solveur LSAT. Le solveur LSAT est basé sur la procédure classique de DPLL. Il propose une implémentation classique de cette procédure, c'est-à-dire sans intégration de structures paresseuses (type « watched litterals »), ni technique d'analyse de conflits ou apprentissage. Cette méthode a été testée sur différents problèmes afin de voir les avantages d'exploiter les symétries locales dans la résolution du problème SAT. Les résultats obtenus confirment le rôle important d'exploiter les symétries locales pour résoudre des problèmes SAT, et montrent que la symétrie locale améliore la symétrie globale sur certains problèmes, et qu'elle

peut être complémentaire aux symétries globales si on les combine.

Dans la deuxième contribution nous avons exploité la symétrie dans l'apprentissage des clauses pour les solveurs du problème SAT à base de CDCL. L'apprentissage par symétrie permet, en plus de l'ajout de la clause assertive correspondant à un nœud de l'arbre de recherche, d'ajouter toutes les clauses assertives symétriques à celle-ci. L'importance de clauses assertives symétriques est d'éviter l'exploration de sous-espaces isomorphes correspondants aux « no-goods » symétriques de l'interprétation partielle courante.

Nous avons introduit un nouveau schéma d'apprentissage exploitant les symétries (SLS) qui peut être utilisé par n'importe quel solveur SAT à base de CDCL. Ensuite, nous avons implémenté ce schéma dans le solveur MiniSat et nous avons expérimenté ce schéma sur une grande variété de problèmes. Les résultats expérimentaux obtenus sont très encourageants, et montrent que l'utilisation de la symétrie dans l'apprentissage est profitable sur la plupart des problèmes testés.

Dans la dernière contribution de cette thèse nous avons étendu la notion de la symétrie aux formalismes logiques non classiques et nous avons montré que des symétries existent dans les logiques non classiques et qui n'apparaissent pas dans les logiques classiques. Nous avons défini la notion de la symétrie sémantique dans les logiques préférentielles, la notion de la symétrie syntaxique dans le cadre des $X-$logiques et les deux symétries (sémantique et syntaxique) dans le cadre de la logique des défauts.

Nous avons introduit une nouvelle règle d'inférence dans le cadre des $X-$logiques. Enfin, nous avons montré comment la symétrie peut être utilisée pour améliorer la recherche des extensions dans le cadre de la logique des défauts et nous avons introduit une nouvelle règle d'inférence par symétrie qui peut être utilisée pour trouver de courtes preuves dans ce cadre.

Perspectives

Cependant il reste encore de nombreuses perspectives à ces différentes contributions. En ce qui concerne la détection et l'élimination dynamique des symétries dans les algorithmes de résolution du problème SAT, nous envisageons d'implémenter une version affaiblie des conditions de détection des symétries locales que nous supposons plus avantageuse pour détecter un plus grand nombre de symétries, dont nous comparerons les résultats avec ceux que nous avons présenté ici.

Un autre point important est de tenter de détecter les symétries de variables locales et de poster dynamiquement des contraintes qui les éliminent. Il serait alors important de comparer les approches statiques qui détectent uniquement les symétries globales avec cette approche.

Sur la partie d'apprentissage par symétrie dans les solveurs du problème SAT, nous chercherons à trouver une bonne stratégie de suppression de clauses qui préservera les clauses symétriques apprises pendant la recherche. Actuellement, MiniSat peut supprimer des clauses assertives symétriques qui peuvent être utiles pour couper des sous-espaces isomorphes.

Seules les symétries du problème initial (symétries globales) sont utilisées dans ce schéma d'apprentissage. Nous prévoyons d'étendre ce schéma d'apprentissage à l'exploitation des symétries locales, qui peuvent être détectées au cours de la recherche.

En ce qui concerne la dernière contribution, nous envisageons d'étudier en profondeur la relation entre la symétrie sémantique en logique préférentielle et la symétrie syntaxique définie dans les $X-$logiques et d'établir le lien entre elles, en utilisant les définitions de la symétrie introduites dans le cadre de la logique des défauts.

Un autre point que nous voulons étudier est d'inclure la symétrie dans les algorithmes d'énumération des extensions de la logique des défauts pour améliorer leurs performances.

Enfin, nous envisageons d'étudier la relation entre la logique des défauts et l'« Answer Set Programming » (ASP) afin de calculer les extensions symétriques en utilisant des modèles stables symétriques.

Bibliographie

[Akers, 1978] Akers, S. (1978). Binary decision diagrams. *IEEE Transactions on Computers*, 27 :509–516.

[Aloul et al., 2003] Aloul, F. A., A.Ramani, Markov, I. L., and Sakallak, K. A. (2003). Solving difficult sat instances in the presence of symmetry. *In IEEE Transaction on CAD, vol. 22(9)*, pages 1117–1137.

[Aloul et al., 2004] Aloul, F. A., Ramani, A., Markov, I. L., and Sakallak, K. A. (2004). Symmetry breaking for pseudo-boolean satisfiabilty. *In ASPDAC'04*, pages 884–887.

[Aspvall et al., 1979] Aspvall, B., Plass, M. F., and Tarjan, R. E. (1979). A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3) :121 – 123.

[Audemard et al., 2008] Audemard, G., Bordeaux, L., Hamadi, Y., Jabbour, S., and Sais, L. (2008). A generalized framework for conflict analysis. In *SAT*, pages 21–27.

[Beame et al., 2004] Beame, P., Kautz, H. A., and Sabharwal, A. (2004). Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res. (JAIR)*, 22 :319–351.

[Benhamou, 1993] Benhamou, B. (1993). *Étude des symétries et de la cardinalité en calcul propositionnel : Applications aux algorithmes syntaxiques*. PhD thesis, France.

[Benhamou, 1994] Benhamou, B. (1994). Study of symmetry in constraint satisfaction problems. *In Proceedings of the 2nd International workshop on Principles and Practice of Constraint Programming - PPCP'94*.

[Benhamou et al., 2009] Benhamou, B., Nabhani, T., Ostrowski, R., and Saïdi, M. R. (2009). Dynamic symmetry detection and elimination for the satisfiability problem. In *The Ninth International Workshop on Symmetry and Constraint Satisfaction Problems - A Satellite Workshop of CP 2009*, Symcon'2009, Lisbon, Portugal.

[Benhamou et al., 2010a] Benhamou, B., Nabhani, T., Ostrowski, R., and Saïdi, M. R. (2010a). Amélioration de l'apprentissage des clauses par symétrie dans les solveurs sat. In *Actes des 6ème Journées Francophones de Programmation par Contraintes*, JFPC'2010, pages 71–80, Caen, France.

[Benhamou et al., 2010b] Benhamou, B., Nabhani, T., Ostrowski, R., and Saïdi, M. R. (2010b). Détection et élimination dynamique de la symétrie dans le problème de satisfiabilité. In *Actes des 6ème Journées Francophones de Programmation par Contraintes*, JFPC'2010, pages 81–90, Caen, France.

[Benhamou et al., 2010c] Benhamou, B., Nabhani, T., Ostrowski, R., and Saïdi, M. R. (2010c). Dynamic symmetry breaking in the satisfiability problem. In *Proceedings of the 16th international conference on Logic for programming, artificial intelligence, and reasoning*, LPAR-16, Dakar, Senegal.

[Benhamou et al., 2010d] Benhamou, B., Nabhani, T., Ostrowski, R., and Saïdi, M. R. (2010d). Enhancing clause learning by symmetry in sat solvers. In *Proceedings of the 22th International Conference on Tools with Artificial Intelligence*, ICTAI'2010, pages 329 – 335, Arras, France.

[Benhamou et al., 2010e] Benhamou, B., Nabhani, T., Ostrowski, R., and Saïdi, M. R. (2010e). Local symmetry breaking in the satisfiability problem. In *Colloque sur l'Optimisation et les Systèmes d'Information*, COSI'2010, Ouargla, Algérie.

[Benhamou et al., 2010f] Benhamou, B., Nabhani, T., and Siegel, P. (2010f). Reasoning by symmetry in non-monotonic inference. In *International Conference on Machine and Web Intelligence*, ICMWI'2010, pages 264 – 269, Algiers, Algeria.

[Benhamou et al., 2010g] Benhamou, B., Nabhani, T., and Siegel, P. (2010g). Study of symmetry in non-monotonic logics. In *13th international workshop on Non-Monotonic Reasoning*, NMR'2010, Sutton Place, Toronto, Canada.

[Benhamou et al., 2010h] Benhamou, B., Nabhani, T., and Siegel, P. (2010h). Symétries dans les logiques non monotones. In *Actes des 6èmes Journées Francophones de Programmation par Contraintes*, JFPC'2010, pages 91–100, Caen, France.

[Benhamou and Saïdi, 2007] Benhamou, B. and Saïdi, M. R. (2007). Local symmetry breaking during search in csps. In Springer, editor, *The 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *LNCS*, pages 195–209, Providence, USA.

[Benhamou and Sais, 1992] Benhamou, B. and Sais, L. (1992). Theoretical study of symmetries in propositional calculus and applications. In Kapur, D., editor, *Automated Deduction—CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 281–294. Springer Berlin / Heidelberg.

[Benhamou and Sais, 1994] Benhamou, B. and Sais, L. (1994). Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning (JAR),*, 12 :89–102.

[Benhamou et al., 1994] Benhamou, B., Sais, L., and Siegel, P. (1994). Two proof procedures for a cardinality based language. *in proceedings of STACS'94, Caen France*, pages 71–82.

[Bennaceur, 1995] Bennaceur, H. (1995). Boolean approch for representing and solving constraint-satisfaction problems. In *Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence on Topics in Artificial Intelligence*, pages 163–174, London, UK. Springer-Verlag.

[Besnard and Siegel, 1988] Besnard, P. and Siegel, P. (1988). The preferential-models approach in nonmonotonic logics - in non-standard logic for automated reasoning. In *Academic Press*, pages 137–156. ed. P. Smets.

[Bibel, 1990] Bibel, W. (1990). Short proofs of the pigeon hole formulas based on the connection method. *Automated reasoning*, (6) :287–297.

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

[Boole, 1854] Boole, G. (1854). *An Investigation Of The Laws Of Thought.* Dover Publications Inc.

[Boole, 1992] Boole, G. (1992). *Les lois de la pensée.* Mathésis (Paris).

[Boole, 2009] Boole, G. (2009). *An Investigation of the Laws of Thought : on Which Are Founded the Mathematical Theories of Logic and Probabilities.* Cambridge University Press, Cambridge.

[Bossu and Siegel, 1982] Bossu, G. and Siegel, P. (1982). Nonmonotonic reasoning and databases. In *Advances in Data Base Theory*, pages 239–284.

[Bossu and Siegel, 1985] Bossu, G. and Siegel, P. (1985). Saturation, nonmonotonic reasoning and the closed-world assumption. *Artif. Intell.*, 25(1) :13–63.

[Brown et al., 1988] Brown, A.Finkelstein, C., and Purdom, L. P. W. (1988). Bactrack searching in the presence of symmetry. *In t. Mora (ed), Applied algebra, algebraic algorithms and error-correcting codes, 6th International Conference. Springer-Verlag*, (6) :99–110.

[Calais, 1984] Calais, j. (1984). *éléments de théorie des groupes.* Presses Universitaires de France (PUF).

[C.E.Shannon, 1938] C.E.Shannon (1938). A symbolic analysis of relay and switching circuits. *AIEE Transactions*, 57 :713–723.

[Chang and Lee, 1973] Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic logic and mechanical theorem proving*, volume 67. Academic press New York.

[Church, 1936] Church, A. (1936). A note on the entscheidungsproblem. *The Journal of Symbolic Logic*, 1(1) :pp. 40–41.

[Cohen et al., 2005] Cohen, D., Jeavons, P., Jefferson, C., Petrie, K., and Smith, B. (2005). Symmetry definitions for constraint satisfaction problems. *In, proceedings of CP*, pages 17–31.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In *STOC*, pages 151–158.

[Crawford, 1992] Crawford, J. (1992). A theoretical analysis of reasoning by symmetry in first-order logic. *Workshop on Tractable Resonning, AAAI-92, San Jose.*

[Crawford et al., 1996] Crawford, J., Ginsberg, M. L., Luck, E., and Roy, A. (1996). Symmetry-breaking predicates for search problems. In *KR'96 : Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California.

[Cubbada and Mouseigne, 1988] Cubbada, C. and Mouseigne, M. D. (1988). *Variantes de l'algorithme de SL-Résolution avec retenue d'information.* PhD thesis, GIA Luminy (Marseille).

[Dalal, 1992] Dalal, M. (1992). Efficient propositional constraint propagation. In *Proceedings of the tenth national conference on Artificial intelligence*, AAAI'92, pages 409–414. AAAI Press.

[Darga et al., 2008] Darga, P. T., Sakallah, K. A., and Markov, I. L. (2008). Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45st Design Automation Conference, Anaheim, California.*

[Davis et al., 1962] Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5 :394–397.

[Davis and Putnam, 1960] Davis, M. and Putnam, H. (1960). A computing procedure for quatification theory. *JACM*, (7) :201–215.

[Dechter and Rish, 1994] Dechter, R. and Rish, I. (1994). Directional resolution : The davis-putnam procedure, revisited. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 134–145. Morgan Kaufmann.

[Dowling and Gallier, 1984] Dowling, W. F. and Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3) :267 – 284.

[DUBOIS et al., 1996] DUBOIS, O., ANDRÉ, P., BOUFKHAD, Y., and CARLIER, J. (1996). Sat versus unsat. In Johnson, D. S. and Trick, M. A., editors, *Cliques, Coloring and Satis ability, Second DIMACS Implementation Challenge October 1993, DIMACS Series in Discrete Mathematics and Theoretical Science*, volume 26, page 415–436. American Mathematical Society.

[Eén and S'orensson, 2003] Eén, N. and S'orensson, N. (2003). An extensible sat-solver. In Giunchiglia, E. and Tacchella, A., editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer.

[Escalada-Imaz, 1989] Escalada-Imaz, G. (1989). *Optimisation d'algorithmes d'inférence monotone en logique des propositions et du premier ordre*. PhD thesis, Université Paul Sabatier, Toulouse (France).

[Even et al., 1976] Even, S., Itai, A., and Shamir, A. (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4) :691–703.

[Freuder, 1991] Freuder, E. (1991). Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, pages 227–233.

[Gabbay, 1985] Gabbay, D. (1985). *Theoretical foundations for non-monotonic reasoning in expert systems*, pages 439–457. Springer-Verlag New York, Inc., New York, NY, USA.

[Galil, 1977] Galil, Z. (1977). On the complexity of regular resolution and the davis-putnam procedure. *Theoretical Computer Science*, 4(1) :23 – 46.

[Génisson and Siegel, 1994] Génisson, R. and Siegel, P. (1994). A polynomial method for sub-clauses production. In *Proceedings of the sixth international conference on Artificial intelligence : methodology, systems, applications : methodology, systems, applications*, pages 25–34, River Edge, NJ, USA. World Scientific Publishing Co., Inc.

[Gent et al., 2007] Gent, I. P., Kelsey, T., Linton, S. A., Pearson, J., and Roney-Dougal, C. M. (2007). Groupoids and conditional symmetry. In *CP*, pages 823–830.

[Ghallab and Gonzalo, 1991] Ghallab, M. and Gonzalo, E.-I. (1991). A linear control algorithm for a class of rule-based systems. *The Journal of Logic Programming*, 11(2) :117 – 132.

[Glaisher, 1874] Glaisher, J. W. L. (1874). On the problem of the eight queens. *Philosophical Magazine*, 48(4) :457–467.

[Gomes et al., 1997] Gomes, C. P., Selman, B., and Crato, N. (1997). Heavy-tailed distributions in combinatorial search. In *CP*, pages 121–135.

[Gomes et al., 1998] Gomes, C. P., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 431–437, Menlo Park, CA, USA. American Association for Artificial Intelligence.

[Grégoire, 1990] Grégoire, E. (1990). *Logiques non monotones et intelligence artificielle*. Hermès, Paris, France.

[Haken, 1985] Haken, A. (1985). The intractability of resolution. *Theorical Computer Science*, 39 :297–308.

[Hao et al., 2002] Hao, J., Lardeux, F., and Saubion, F. (2002). A hybrid genetic algorithm for the satisfiability problem. In *Proceedings of the First International Workshop on Heuristics*, volume 189. Citeseer.

[Hertel et al., 2008] Hertel, P., Bacchus, F., Pitassi, T., and Gelder, A. V. (2008). Clause learning can effectively p-simulate general propositional resolution. In *AAAI*, pages 283–290.

[Huang, 2007] Huang, J. (2007). The effect of restarts on the efficiency of clause learning. In *IJCAI'07 : Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2318–2323, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Iwama, 1989] Iwama, K. (1989). Cnf satisfiability test by counting and polynomial average time. *SIAM J. Comput.*, 18 :385–391.

[Jr. and Schrag, 1997] Jr., R. J. B. and Schrag, R. (1997). Using csp look-back techniques to solve real-world sat instances. In *AAAI/IAAI*, pages 203–208.

[Kalbfleisch and Stanton, 1969] Kalbfleisch, J. and Stanton, R. (1969). On the maximal triangle-free edge-chromatic graphs in three colors. *combinatorial theory*, (5) :9–20.

[Kleer, 1989] Kleer, J. D. (1989). A comparison of atms and csp techniques. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 290–296.

[Kowalski and Kuehner, 1971] Kowalski, R. and Kuehner, D. (1971). Linear resolution with selection function. *Artificial Intelligence*, (2) :227–260.

[Kraus et al., 1990] Kraus, S., Lehmann, D. J., and Magidor, M. (1990). Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1-2) :167–207.

[Krishnamurty, 1985] Krishnamurty, B. (1985). Short proofs for tricky formulas. *Acta informatica*, (22) :253–275.

[Li and Anbulagan, 1997] Li, C. M. and Anbulagan, A. (1997). Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th international joint conference on Artifical intelligence - Volume 1*, pages 366–371, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Lionel and Siegel, 1996] Lionel, P. S. and Siegel, P. (1996). A representation theorem for preferential logics.

[McCarthy,] McCarthy, J. Circumscription-a form of non-monotonic reasoning. *Artificial Intelligence*, 13.

[McDermott and Doyle, 1980] McDermott, D. V. and Doyle, J. (1980). Non-monotonic logic i. *Artif. Intell.*, 13(1-2) :41–72.

[McKay, 1981] McKay, B. (1981). Practical graph isomorphism. In *Congr. Numer. 30*, pages 45–87.

[Mears et al., 2006] Mears, C., de la Banda, M. G., and Wallace, M. (2006). On implementing symmetry detection. In *The CP 2006 Workshop on Symmetry and Constraint Satisfaction Problems (SymCon'06)*, pages 1–8, Cité des Congrès - Nantes, France.

[Minoux, 1988] Minoux, M. (1988). Ltur : a simplified linear-time unit resolution algorithm for horn formulae and computer implementation. *Information Processing Letters*, 29(1) :1 – 12.

[Ostrowski et al., 2002] Ostrowski, R., Mazure, B., and Sais, L. (2002). Lsat solver. In *Fifth International Symposium on the Theory and Applications of Satisfiability Testing*.

[Pipatsrisawat and Darwiche, 2008] Pipatsrisawat, K. and Darwiche, A. (2008). A new clause learning scheme for efficient unsatisfiability proofs. In *AAAI*, pages 1481–1484.

[Pipatsrisawat and Darwiche, 2009] Pipatsrisawat, K. and Darwiche, A. (2009). On the power of clause-learning sat solvers with restarts. In *CP*, pages 654–668.

[Puget, 1993] Puget, J. F. (1993). On the satisfiability of symmetrical constrained satisfaction problems. In *In J. Kamorowski and Z. W. Ras,editors, Proceedings of ISMIS'93, LNAI 689*.

[Puget, 2005] Puget, J. F. (2005). Automatic detection of variable and value symmetries. In Springer, L., editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP-2005)*, pages 474–488, Sitges, Spain.

[Rauzy, 1995] Rauzy, A. (1995). Polynomial restrictions of sat : What can be done with an efficient implementation of the davis and putnam's procedure ? In Montanari, U. and Rossi, F., editors, *Principles and Practice of Constraint Programming ? CP '95*, volume 976 of *Lecture Notes in Computer Science*, pages 515–532. Springer Berlin / Heidelberg. 10.1007/3-540-60299-2_31.

[Reiter, 1980] Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, pages 81–132.

[Reiter and Criscuolo, 1981] Reiter, R. and Criscuolo, G. (1981). On interacting defaults. In *IJCAI*, pages 270–276.

[Robinson, 1963] Robinson, J. A. (1963). Teorem proving on computer. *JACM*, pages 163–174.

[Robinson, 1965] Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *JACM, 12*, pages 23–81.

[Robinson, 1983] Robinson, J. A. (1983). *Automatic Deduction with Hyper Resolution*, pages 416–423. Springer-Verlag.

[Scutellà, 1990] Scutellà, M. G. (1990). A note on dowling and gallier's top-down algorithm for propositional horn satisfiability. *The Journal of Logic Programming*, 8(3) :265 – 273.

[Selman et al., 1992] Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the tenth national conference on Artificial intelligence*, AAAI'92, pages 440–446. AAAI Press.

[SHOAM, 1987] SHOAM, Y. (1987). A semantical approach to nonmonotonic logic. In *IJCAI*, pages 388–392.

[Siegel et al., 2001] Siegel, P., Forget, L., and Risch, V. (2001). Preferential logics are x-logics. *Journal of Logic and Computation*, 11(1) :71–83.

[Silva and Sakallah, 1996] Silva, J. P. M. and Sakallah, K. A. (1996). Grasp - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227.

[S.Sadok, 2000] S.Sadok (2000). Une caractérisation des X-logiques en termes de tableaux analytiques. *Mémoire de DEA*.

[Tarjan, 1972] Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2) :146–160.

[Tseitin, 1968] Tseitin, G. S. (1968). On the complexity of derivation in propositional calculus. In *Structures in the constructive Mathematics and Mathematical logic*, pages 115–125. H.A.O Shsenko.

[Zhang et al., 2001] Zhang, L., Madigan, C. F., Moskewicz, M. W., and Malik, S. (2001). Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD*, pages 279–285.

Résumé

La symétrie est par définition un concept multidisciplinaire. Il apparaît dans de nombreux domaines. En général, elle revient à une transformation qui laisse invariant un objet.

Le problème de satisfaisabilité (SAT) occupe un rôle central en théorie de la complexité. Il est le problème de décision de référence de la classe NP-complet (Cook, 71). Il consiste à déterminer si une formule booléenne admet ou non une valuation qui la rend vraie.

Dans la première contribution de ce mémoire, nous avons introduit une nouvelle méthode complète qui élimine toutes les symétries locales pour la résolution du problème SAT en exploitant son groupe des symétries. Les résultats obtenus montrent que l'exploitation des symétries locales est meilleure que l'exploitation des symétries globales sur certaines instances SAT et que les deux types de symétries sont complémentaires, leur combinaison donne une meilleure exploitation.

En deuxième contribution, nous proposons une approche d'apprentissage de clauses pour les solveurs SAT modernes en utilisant les symétries. Cette méthode n'élimine pas les modèles symétriques comme font les méthodes statiques d'élimination des symétries. Elle évite d'explorer des sous-espaces correspondant aux no-goods symétriques de l'interprétation partielle courante. Les résultats obtenus montrent que l'utilisation de ces symétries et ce nouveau schéma d'apprentissage est profitable pour les solveurs CDCL.

En Intelligence Artificielle, on inclut souvent la non-monotonie et l'incertitude dans le raisonnement sur les connaissances avec exceptions. Pour cela, en troisième et dernière contribution, nous avons étendu la notion de symétrie à des logiques non classiques (non-monotones) telles que les logiques préférentielles, les $X-$logiques et les logiques des défauts. Nous avons montré comment raisonner par symétrie dans ces logiques et nous avons mis en évidence l'existence de certaines symétries dans ces logiques qui n'existent pas dans les logiques classiques.

Mots-clés : Problème SAT, symétrie locale, symétrie sémantique, symétrie syntaxique, solveurs CDCL, logiques non monotones, logiques préférentielles, $X-$logiques, logiques des défauts.

Abstract

Symmetry is by definition a multidisciplinary concept. It appears in many fields. In general, it is a transformation which leaves an object invariant.

The problem of satisfiability (SAT) is one of the central problems in the complexity theory. It is the first decision NP-complete problem (Cook, 71). It deals with determining if a Boolean formula admits a valuation which makes it true.

First we introduce a new method which eliminates all the local symmetries during the resolution of a SAT problem by exploiting its group of symmetries. Our experimental results show that for some SAT instances, exploiting local symmetries is better than exploiting just global symmetries and both types of symmetries are complementary.

As a second contribution, we propose a new approach of Conflict-Driven Clause Learning based on symmetry. This method does not eliminate the symmetrical models as the static symmetry elimination methods do. It avoids exploring sub-spaces corresponding to symmetrical No-goods of the current partial interpretation. Our experimental results show that using symmetries in clause learning is advantageous for CDCL solvers.

In artificial intelligence, we usually include non-monotony and uncertainty in the reasoning on knowledge with exceptions. Finally, we extended the concept of symmetry to non-classical logics that are preferential logics, $X-$logics and default logics. We showed how to reason by symmetry in these logics and we prove the existence of some symmetries in these non-classical logics which do not exist in classical logics.

Keywords : SAT problem, local symmetry, semantic symmetry, syntaxic symmetry, solvers CDCL, non monotonic logics, preferential logics, $X-$logics, default logics.