La XOR-résolution

Nicolas Prcovic

La XOR-résolution

Nicolas Prcovic
LSIS - Université d'Aix-Marseille
nicolas.prcovic@lsis.org

Résumé

Nous présentons la XOR-résolution, une généralisation de la résolution standard. Elle part de l'idée qu'à partir de deux clauses CNF contenant deux littéraux opposées $x \vee y \vee C_1$ et $\overline{x} \vee \overline{y} \vee C_2$, la résolution ne permet d'inférer que la tautologie $y \vee \overline{y} \vee C1 \vee C_2$ alors que la sous-formule non tautologique $(x \oplus y) \vee C1 \vee C_2$ (où $\oplus$ est le "ou exclusif") pourrait être produite. En généralisant la règle de résolution, nous obtenons un système plus puissant (ie, capable d'obtenir une réfutation de longueur polynômiale plutôt qu'exponentielle de certaines classes d'instances). Nous montrons aussi comment les méthodes de résolution arborescente de type DPLL ou CDCL peuvent intégrer la XOR-Résolution pour être théoriquement plus puissantes.

Abstract

We present XOR-resolution, a generalization of Resolution. It starts from the idea that from two CNF clauses containing two opposite literals $x \vee y \vee C_1$ and $\overline{x} \vee \overline{y} \vee C_2$, Resolution only allows to infer the tautology $y \vee \overline{y} \vee C_1 \vee C_2$ while the non tautological subformula $(x \oplus y) \vee C_1 \vee C_2$ (where $\oplus$ is the "exclusive or") could be produced. By generalizing the Resolution rule, we obtain a more powerful proof system (ie, able to generate a polynomial, instead of exponential, length refutation of some classes of instances). We also show how the Resolution-based tree search methods DPLL or CDCL can integrate XOR-Resolution to be more powerful.

1 Introduction

Les systèmes de preuve basés sur la résolution (Res) [3] sont utilisés pour permettre de trouver la réfutation d'une formule booléenne supposée insatisfiable ou de montrer qu'il existe un modèle. Les méthodes classiques de recherche arborescente de modèle (DPLL, CDCL) peuvent être interprétées comme une manière d'appliquer une suite de résolutions pour obtenir une solution. Cependant, certaines familles d'instances [4, 13, 1] nécessitent de produire un nombre exponentiel de résolvantes. La résolution étendue [12] ajoute une régle supplémentaire à la résolution. Personne n'a jamais trouvé de classe d'instances nécessitant la production d'un nombre exponentiel de résolvantes avec la résolution étendue. Mais, bien que rendant un système de preuve par résolution plus puissant, la règle d'extension est très difficile à mettre en pratique car le nombre d'extensions possibles est très élevé et il est difficile de savoir quand l'ajout d'une extension sera utile ou dommageable.

Nous présentons ici un système de preuve intermédiaire entre la résolution et la résolution étendue. Il est basé sur la remarque que lorsque la règle de résolution génère une tautologie, donc une clause inutile, il est possible à la place d'inférer une sous-formule non tautologique qui apporte une "information" potentiellement utile.

La règle de résolution permet d'obtenir une clause $C_1 \vee C_2$ à partir des clauses $x \vee C_1$ et $\overline{x} \vee C_2$. Mais si un littéral est dans C_1 et que son opposé est dans C_2 alors on a affaire à une tautologie donc $C_1 \vee C_2$ n'est pas ajoutée à l'ensemble des clauses.

Or, il s'avère que l'on peut inférer logiquement une formule non tautologique à partir de $x \vee C_1$ et $\overline{x} \vee C_2$ en généralisant la règle de résolution. Nous allons le voir en rappelant la raison pour laquelle on peut ajouter la résolvante $C_1 \vee C_2$ à l'ensemble des clauses.

L'ensemble des clauses représente une conjonction de clauses F qu'on peut exprimer ainsi : $F = (x \vee C_1) \wedge (\overline{x} \vee C_2) \wedge F'$.

Comme on a la propriété $A = A \wedge (A \vee B)$, la formule peut se réécrire successivement :
$F = ((x \vee C_1) \wedge (x \vee C_1 \vee C_2)) \wedge ((\overline{x} \vee C_2) \wedge (\overline{x} \vee C_2 \vee C_1)) \wedge F'$
$F = (x \vee C_1) \wedge (\overline{x} \vee C_2) \wedge ((x \wedge \overline{x}) \vee C_1 \vee C_2) \wedge F'$
$F = (x \vee C_1) \wedge (\overline{x} \vee C_2) \wedge (C_1 \vee C_2) \wedge F'$

Or, si nous avons deux clauses $x \vee y \vee C_1$ et

$\overline{x} \vee \overline{y} \vee C_2$, le même type de raisonnement s'applique :
$F = (x \vee y \vee C_1) \wedge (\overline{x} \vee \overline{y} \vee C_2) \wedge F'$
$F = ((x \vee y \vee C_1) \wedge (x \vee y \vee C_1 \vee C_2)) \wedge ((\overline{x} \vee \overline{y} \vee C_2) \wedge (\overline{x} \vee \overline{y} \vee C_2 \vee C_1)) \wedge F'$
$F = (x \vee y \vee C_1) \wedge (\overline{x} \vee \overline{y} \vee C_2) \wedge (((x \vee y) \wedge (\overline{x} \vee \overline{y})) \vee C_1 \vee C_2) \wedge F'$

Or, $(x \vee y) \wedge (\overline{x} \vee \overline{y}) = x \oplus y$, $\oplus$ étant le "ou exclusif". On peut donc ajouter $(x \oplus y) \vee C_1 \vee C_2$ à la formule (plutôt qu'une tautologie inutile).

Bien sûr, si z est dans C_1 et $\overline{z}$ est dans C_2, $(x \oplus y) \vee C_1 \vee C_2$ est une tautologie. Mais nous pouvons généraliser le raisonnement, quelque soit le nombre de littéraux en opposition dans les deux clauses : à partir des clauses $x_1 \vee ... \vee x_n \vee C_1$ et $\overline{x_1} \vee ... \vee \overline{x_n} \vee C_2$, on peut ajouter $((x_1 \vee ... \vee x_n) \wedge (\overline{x_1} \vee ... \vee \overline{x_n})) \vee C_1 \vee C_2$. Or, $x_1 \vee ... \vee x_n$ signifie que tous les x_i ne sont pas égaux à 0 tandis que $\overline{x_1} \vee ... \vee \overline{x_n}$ signifie que tous les x_i ne sont pas égaux à 1. La conjonction des deux équivaut au fait que parmi les x_i, il y en a au moins un qui diffère des autres. On peut donc réexprimer $(x_1 \vee ... \vee x_n) \wedge (\overline{x_1} \vee ... \vee \overline{x_n})$ par $(x_1 \oplus x_2) \vee ... \vee (x_1 \oplus x_n)$ qui signifie que x_1 (on aurait pu choisir n'importe quelle autre variable de manière équivalente) diffère au moins d'un des autres x_i. Au final, la formule inférée est $(x_1 \oplus x_2) \vee ... \vee (x_1 \oplus x_n) \vee C_1 \vee C_2$ qui est une disjonction d'exclusions, non tautologique.

Nous voyons que les formules que nous pouvons inférer ne sont plus des clauses CNF (des disjonctions de littéraux) mais des disjonctions d'exclusions, ce qui nous amène à généraliser la notion de clause afin de définir un système de preuve qui généralise Res.

Dans cet article, nous allons donc définir la notion de x-clauses, de formule XCNF et la XOR-résolution, qui généralisent respectivement les clauses CNF, les formules CNF et la résolution standard. Nous montrerons que la XOR-résolution constitue un système de preuve plus puissant que la résolution mais moins puissant que la résolution étendue. Nous montrerons ensuite comment prendre en compte la XOR-résolution dans les méthodes de recherche arborescente pour rendre les solveurs plus puissant (en théorie).

2 Notions préliminaires

Une *formule booléenne*, sous sa forme dite *CNF*, est une conjonction de clauses. Une clause est une disjonction de littéraux. Un littéral est soit une variable booléenne, soit sa négation. On définit var(l) comme étant la variable du littéral l. Les variables peuvent être affectées à la valeur 1 (vrai) ou 0 (faux). Une clause peut être représentée par l'ensemble de ses littéraux. Une formule SAT peut être représentée par l'ensemble de ses clauses. Un *modèle* d'une formule F est une affectation de variables qui est telle que F est vraie. Une formule n'ayant pas de modèle est dite *insatisfiable*. La clause vide, notée $\square$, est toujours fausse et si une formule la contient alors elle est insatisfiable. Si une clause contient un littéral l et son opposé $\neg l$, elle est toujours vraie et on l'appelle *tautologie* sur var(l).

2.1 Resolution

Les systèmes formels de preuve propositionnelle permettent de dériver d'autres formules à partir d'une formule F, grâce à des règles d'inférence. En particulier, le système de Robinson [10], que nous appelons `Res`, permet de dériver des clauses induites à partir des clauses d'une formule F grâce à la règle de *résolution* :

$$\frac{C_1 \qquad C_2}{C_1 \backslash \{l\} \cup C_2 \backslash \{\bar{l}\}}$$

La clause inférée par résolution de C_1 avec C_2 sera appelé *résolvante* sur var(l) de C_1 et C_2. L'intérêt de `Res` est qu'il permet d'établir des réfutations de formules (ie, des preuves de leur insatisfiabilité) dans la mesure où une formule est insatisfiable si et seulement si il existe une suite de résolutions qui dérive la clause vide. `Res` fonctionne par saturation de l'application de sa règle : si une formule est satisfiable, `Res` le détecte quand plus aucune résolvante non redondante ne peut être dérivée.

Une dérivation est une suite d'applications de la règle de résolution permettant de dériver une clause. Elle constitue la preuve que cette clause est une conséquence logique de la formule. Une dérivation de la clause vide s'appelle une *réfutation*. On appelle *longueur d'une preuve*, le nombre de résolvantes qu'elle contient.

Nous rappelons qu'on dit qu'un système de preuves P *p-simule* un système de preuves Q ssi il existe une fonction reformulant en temps polynomial une preuve produite par P à partir d'une preuve produite par Q. Ainsi, P est dit *plus puissant* que Q si P peut p-simuler Q mais Q ne peut pas p-simuler P. Informellement, un système de preuves propositionnel P est plus puissant que Q si P peut générer des preuves de longueur polynômiale pour des classes d'instances pour lesquelles Q ne peut générer que des preuves de longueur exponentielle, et que lorsque Q peut générer une preuve de longueur polynômiale, P le peut aussi.

2.2 Le "ou exculsif" (XOR)

Nous rappelons succintement quelques propriétés de l'opérateur $\oplus$. Il est commutatif et associatif.

$a \oplus b = a \wedge \bar{b} \vee \bar{a} \wedge b = (a \vee b) \wedge (\bar{a} \vee \bar{b})$.

$a \oplus b$ peut s'interpréter comme "a diffère de b". $a_1 \oplus a_2 \oplus ... \oplus a_n$ peut s'interpréter comme "il y a

un nombre impair de variables a_i qui sont égales à 1 (vraies)" ou comme une équation linéaire dans le corps Z/2Z, c'est-à-dire "la somme modulo 2 des variables a_i égale 1". C'est pourquoi nous les appellerons *équations booléennes*, mais on les retrouve aussi ailleurs sous le nom de *contraintes de parité*.

Voici quelques relations que nous utilisons dans cet article :
$a \oplus 1 = \overline{a}$ (1)
Simplifications :
$a \oplus a = 0$ (2)
$a \oplus b \vee a = a \vee b$ (3)
$a \oplus b \vee \overline{a} = \overline{a} \vee \overline{b}$ (4)
$a \oplus b \wedge a = \overline{b}$ (5)
$a \oplus b \wedge \overline{a} = b$ (6)
Redistribution des variables dans les équations :
$a \oplus b \vee a \oplus c = a \oplus b \vee b \oplus c$ (7)
$a \oplus b \wedge a \oplus c = a \oplus b \wedge b \oplus c \oplus 1$ (8)

En fait, les relations (3), (4), (5) et (6) sont des cas particuliers des relations (7) et (8) pour lesquelles on a $b = 0$ ou $b = 1$.

L'opérateur d'équivalence $\Leftrightarrow$ est la négation du $\oplus$: $x \Leftrightarrow y = \overline{x \oplus y} = x \oplus y \oplus 1$. Toute équation booléenne est reformulable en une chaîne d'équivalences en remplaçant chaque $\oplus$ par un $\Leftrightarrow$ et en ajoutant "$\oplus 1$" si le nombre de $\oplus$ était impair.

On normalise l'expression d'une équation booléenne en utilisant d'abord la relation (1) pour faire disparaître les occurences de littéraux négatifs, puis en utilisant la relation (2) pour faire disparaître les doublons de variable ou de 1. À la fin, une équation booléenne ne contient plus de négation, aucune variable n'apparaît plus d'une fois, et 1 apparaît une fois au plus.

Les systèmes linéaires d'équations booléennes (ie, les conjonctions d'équations booléennes) se résolvent grâce à la méthode du pivot de Gauss. Quand un tel système a au moins une solution, l'ensemble des équations s'exprime d'une manière normalisée en ordonnant arbitrairement les variables. On fait alors en sorte que la première variable n'apparaisse plus à partir de la deuxième équation, que la deuxième variable n'apparaisse plus à partir de la troisième équation, etc. Cela se fait en utilisant un nombre polynômial de fois la relation (8).

Ex : $a \oplus b \oplus c \wedge a \oplus d \oplus e \wedge b \oplus d \oplus 1 =$
$a \oplus b \oplus c \wedge (b \oplus c \oplus 1) \oplus d \oplus e \wedge b \oplus d \oplus 1 =$
$a \oplus b \oplus c \wedge b \oplus c \oplus 1 \oplus d \oplus e \wedge (c \oplus d \oplus e) \oplus d \oplus 1 =$
$a \oplus b \oplus c \wedge b \oplus c \oplus 1 \oplus d \oplus e \wedge c \oplus e \oplus 1.$
À ce moment, on réécrit les équations, toujours en utilsant la relation (8) de manière à ce qu'elles contiennent des variables dont l'ordre est le plus éloigné possible : $b \oplus c \oplus 1 \oplus d \oplus e = b \oplus e \oplus 1 \oplus d \oplus e = b \oplus d \oplus 1$ et $a \oplus b \oplus c = a \oplus d \oplus e$. Au final, la normalisation de la conjonction d'équations donne $a \oplus d \oplus e \wedge b \oplus d \oplus 1 \wedge c \oplus e \oplus 1$.

Lorsqu'on normalise un système d'équations booléennes, on le résout : le système est inconsistent ssi l'équation vide (0) est produite pendant la normalisation. Le problème XORSAT de décision sur l'existence d'une solution d'une conjonction d'équations booléennes est donc polynomial.

2.3 Formules XCNF

On appelle *x-clause* (ou simplement *clause* à partir de maintenant, lorsque le contexte n'est pas ambigü), une disjonction d'équations booléennes. Les x-clauses ont déjà été introduites (sous le nom de *clauses étendues*) dans [6] dans le cadre de la compilation de connaissances dédiée au comptage de modèles. En imitant la normalisation des systèmes d'équations booléennes, on normalise la formulation des x-clauses grâce à la relation (7). Précisément, on ordonne arbitrairement les variables et on applique la relation (7) de manière à ce que la première variable n'apparaisse pas dans les équations booléennes suivantes, que la deuxième variable n'apparaisse plus à partir de la troisième équation, etc. Ensuite, comme pour la normalisation des conjonctions d'équations, on réécrit les équations de manière à ce qu'elles contiennent des variables dont l'ordre est le plus éloigné possible. Cette normalisation permet de diminuer au maximum le nombre d'équations dans la disjonction.
Ex : $a \oplus b \vee a \oplus c \vee b \oplus c =$
$a \oplus b \vee b \oplus c \vee b \oplus c =$
$a \oplus b \vee b \oplus c \vee c \oplus c =$
$a \oplus b \vee b \oplus c$

Elle permet aussi de détecter les tautologies.
Ex : $a \oplus b \vee a \oplus c \vee b \oplus c \oplus 1 =$
$a \oplus b \vee b \oplus c \vee b \oplus c \oplus 1 =$
$a \oplus b \vee b \oplus c \vee c \oplus c \oplus 1 = 1$

Une *formule XCNF* (appelée E-CNF dans [6]) est une conjonction de x-clauses. Toute formule CNF peut se réécrire comme une formule XCNF dans la mesure où il suffit de remplacer chaque littéral négatif $\overline{x}$ par $1 \oplus x$. Une instance XCNF dont toutes les clauses sont unaires (ie, ne contiennent qu'une équation) équivaut à une instance XORSAT (et l'ensemble de ces instances constitue donc une classe polynômiale du problème XCNF).

3 La XOR-résolution

Etant données deux x-clauses, on définit deux règles qui permettent d'en inférer une troisième pour l'ajouter à l'ensemble des x-clauses de la formule. Ces deux

règles constituent le système de preuves que nous appelons XOR-résolution (`XRes`).

$$\text{R1} : \frac{Q_1 \vee ... \vee Q_n \vee C_1 \qquad \overline{Q_1} \vee ... \vee \overline{Q_n} \vee C_2}{Q_1 \oplus Q_2 \vee ... \vee Q_1 \oplus Q_n \vee C_1 \vee C_2}$$

où chaque Q_i est une équation booléenne et $\overline{Q_i}$ désigne son opposée.

$$\text{R2} : \frac{x \oplus Q_1 \vee C_1 \qquad x \oplus Q_2 \vee C_2}{1 \oplus Q_1 \oplus Q_2 \vee C_1 \vee C_2}$$

où x est une variable, et Q_1 et Q_2 sont des équations booléennes. R2 se justifient par la relation $x \oplus Q_1 \wedge x \oplus Q_2 = 1 \oplus Q_1 \oplus Q_2 \wedge x \oplus Q_1$. Cette règle permet d'inférer une x-clause quand deux x-clauses ont chacune une équation booléenne qui possède une variable x commune.

R1 et R2 sont des généralisations de la règle de résolution. R1 équivaut à la règle de résolution si $n = 1$, R2 si $Q_1 = 0$ et $Q_2 = 1$.

La règle R2 permet d'obtenir une x-clause sans x dans la mesure où il est toujours possible de normaliser une x-clause de telle manière que x n'apparaissent qu'une seule fois dans chacune des deux x-clauses. Il suffit que x soit considérée comme la première variable de son ensemble. Cependant, on ne doit appliquer R2 que si elle ne produit pas une tautologie. Sinon, c'est la règle R1 qu'il faut appliquer.

3.1 Puissance de la XOR-résolution

Définition 1 *Soit $X = \{x_1, x_2, ..., x_k\}$. On note $Eq(X)$ l'équation booléenne $x_1 \oplus x_2 \oplus ... \oplus x_k$.*
On note $CNF(X)$ l'ensemble (insatisfiable) de toutes les clauses CNF de longueur $|X|$ que l'on peut faire avec les variables de X.
On note $CNF(X)^{\oplus}$ l'ensemble de toutes les clauses CNF dont la conjonction équivaut à $Eq(X)$.
On note $CNF(X)^{\oplus 1}$ le complémentaire de $CNF(X)^{\oplus}$ par rapport à $CNF(X)$.

Ex : $CNF(\{a,b,c\}) = \{a \vee b \vee c, a \vee b \vee \overline{c}, a \vee \overline{b} \vee c, a \vee \overline{b} \vee \overline{c}, \overline{a} \vee b \vee c, \overline{a} \vee b \vee \overline{c}, \overline{a} \vee \overline{b} \vee c, \overline{a} \vee \overline{b} \vee \overline{c}\}$. $CNF(\{a,b,c\})^{\oplus} = \{a \vee b \vee c, a \vee \overline{b} \vee \overline{c}, \overline{a} \vee b \vee \overline{c}, \overline{a} \vee \overline{b} \vee c\}$. $CNF(\{a,b,c\})^{\oplus 1} = \{a \vee b \vee \overline{c}, a \vee \overline{b} \vee c, \overline{a} \vee b \vee c, \overline{a} \vee \overline{b} \vee \overline{c}\}$. La conjonction des clauses de $CNF(\{a,b,c\})^{\oplus}$ équivaut à $Eq(\{a,b,c\}) = a \oplus b \oplus c$ et celle de $CNF(\{a,b,c\})^{\oplus 1}$ à $Eq(\{a,b,c\}) \oplus 1 = a \oplus b \oplus c \oplus 1$.

Proposition 1 *Soit $X = \{x_1, x_2, ..., x_k\}$. À partir des clauses de $CNF(X)^{\oplus}$, on obtient la x-clause $Eq(X)$ en appliquant $2^{k-1} - 1$ fois la règle R1 de `XRes`.*

On procède en faisant des résolutions avec la règle R1 entre couples de clauses de la forme $C \vee x_i \vee x_{i+1} \oplus ... \oplus x_k$ et $C \vee \overline{x_i} \vee x_{i+1} \oplus ... \oplus x_k \oplus 1$, qui produit $C \vee x_i \oplus x_{i+1} \oplus ... \oplus x_k$, et couples de clauses de la forme $C \vee \overline{x_i} \vee x_{i+1} \oplus ... \oplus x_k$ et $C \vee x_i \vee x_{i+1} \oplus ... \oplus x_k \oplus 1$, qui produit $C \vee x_i \oplus x_{i+1} \oplus ... \oplus x_k \oplus 1$. Au début, $i = n-1$, on a les 2^{k-1} clauses de $CNF(X)^{\oplus}$ et on produit 2^{k-2} résolvantes. Puis, à partir de ces résolvantes, pour $i = k-2$, on produit 2^{k-3} résolvantes selon le même schéma, et on continue pour toutes les valeurs de i décroissante jusque $i = 0$, où on obtient la résolvante $x_1 \oplus x_2 \oplus ... \oplus x_k$. En tout, on a appliqué $2^{k-1} - 1$ fois la règle R1. □

Proposition 2 *La XOR-résolution est strictement plus puissante que la Résolution standard.*

Dans la mesure où les x-clauses sont des généralisations des clauses CNF et que les règles de `XRes` sont des généralisations de la règle de résolution, `XRes` est au moins aussi puissant que `Res`. Nous allons montrer que les problèmes d'Urquhart, qui se réfutent en temps nécessairement exponentiel avec `Res` [13], se réfutent en temps polynomial avec `XRes`. Les problèmes d'Urquhart ne contiennent que des clauses appartenant à des ensembles $CNF(X_i)^{\oplus}$ ou $CNF(X_i)^{\oplus 1}$, où les X_i sont des sous-ensembles non disjoints de l'ensemble de toutes les variables. Par ailleurs, chaque X_i contient sept variables au maximum. À une équation booléenne portant sur k variables correspond 2^{k-1} clauses CNF. Un problème d'Urquhart qui serait formulé sous la forme d'un ensemble d'équations booléennes se résoudrait en temps polynômial grâce par exemple à la méthode du pivot de Gauss, qui se simule par l'application de la règle R2 (avec $C_1 = 0$ et $C_2 = 0$). Or, en partant des clauses d'un ensemble $CNF(X_i)^{\oplus}$ ou $CNF(X_i)^{\oplus 1}$, on peut obtenir chacune de ces équations grâce à $2^7 - 1$ applications de la règle R2 au maximum d'après la proposition 1. Donc toute instance du problème d'Urquhart se résout en temps polynômial. □

3.2 XOR-résolution et résolution étendue

La résolution étendue [12] consiste à ajouter à `Res` la *règle d'extension* en plus de la règle de résolution. Nous appellerons `ER` ce système de preuve. En toute généralité, la règle d'extension permet d'ajouter (à l'ensemble des clauses) les clauses correspondant à la formule $x \Leftrightarrow F$, où x est une nouvelle variable et F est n'importe quelle formule portant sur des variables existant déjà. L'intérêt de rajouter la règle d'extension est qu'elle rend le système plus puissant dans la mesure où certains problèmes dont la preuve est de longueur nécessairement exponentielle dans `Res` ont une preuve polynômiale dans `ER`. C'est le cas du fameux problème des pigeons [4, 2].

Chaque x-clause peut se réécrire comme la conjonction d'une clause CNF et d'équations booléennes :

$Q_1 \vee ... \vee Q_n =$
$(z_1 \vee ... \vee z_n) \wedge (z_1 \Leftrightarrow Q_1) \wedge ... \wedge (z_n \Leftrightarrow Q_n) =$
$(z_1 \vee ... \vee z_n) \wedge (z_1 \oplus Q_1 \oplus 1) \wedge ... \wedge (z_n \oplus Q_n \oplus 1)$. Les z_i sont des variables introduites par la règle d'extension. Ainsi, `XRes` peut être vu comme une façon restreinte d'appliquer la résolution étendue. Ses deux règles se simulent ainsi :

$$\text{R1'} : \frac{x_1 \vee ... \vee x_n \vee C_1 \qquad \overline{x_1} \vee ... \vee \overline{x_n} \vee C_2}{z_1 \vee ... \vee z_{n-1} \vee C_1 \vee C_2,\ z_1 \Leftrightarrow x_1 \oplus x_2,\ ...,\ z_{n-1} \Leftrightarrow x_1 \oplus x_n}$$

$$\text{R2'} : \frac{z_1 \vee C_1 \qquad z_2 \vee C_2 \qquad z_1 \Leftrightarrow x \oplus y_1 \qquad z_2 \Leftrightarrow x \oplus y_2}{z_3 \vee C_1 \vee C_2,\ z_3 \Leftrightarrow 1 \oplus y_1 \oplus y_2}$$

Chaque formule de type $a \Leftrightarrow b \oplus c$ équivaut à la conjonction des quatre clauses CNF $\overline{a} \vee b \vee c$, $a \vee \overline{b} \vee c$, $a \vee b \vee \overline{c}$, $\overline{a} \vee \overline{b} \vee \overline{c}$. On peut donc réécrire les règles R1' et R2' uniquement avec des clauses CNF.

4 XDPLL : la recherche arborescente sur une formule XCNF

Il paraît peu probable que la définition d'un solveur basé directement sur la résolution d'une instance CNF grâce à l'application répétée des règles de la XOR-résolution lui permette d'être compétitif avec les solveurs SAT complets les plus récents, tous basés sur la recherche arborescente. Il nous faut donc trouver comment effectuer une recherche arborescente qui simule la XOR-résolution, au moins partiellement. Dans cette section, nous montrons comment simuler l'application de la règle R2.

Pour résoudre une instance F, un solveur de type DPLL ou CDCL utilise le fait que $F = (F \wedge x) \vee (F \wedge \overline{x})$, où x est une variable apparaissant dans F. Il procède par dichotomie en considérant $F \wedge x$ et $F \wedge \overline{x}$. $F \wedge x$ se déduit de F en retirant les clauses contenant x, car la clause unitaire x les subsume, et en retirant le littéral $\overline{x}$ des clauses qui le contenait, obtenant ainsi les résolvantes entre les clauses contenant $\overline{x}$ et la clause unitaire x (grâce à la propagation unitaire).

On peut généraliser le procédé en remarquant que $F = (F \wedge Q) \vee (F \wedge (Q \oplus 1))$ où Q est une équation booléenne contenant des variables de F (comme cela a déjà été proposé dans [6], mais pour compiler de la connaissance). Ainsi, on remplace le choix de variable par le choix d'un sous-ensemble de variables, et le choix de valeur par le choix d'une des deux équations booléennes sur ces variables. Cela étant, afin de gérer correctement l'adjonction d'une équation booléenne dans la formule correspondant à un sous-arbre de recherche, nous allons devoir généraliser la notion de propagation unitaire.

4.1 Propagation unitaire dans les XCNF

Dans le contexte d'une instance XCNF, une clause unitaire est une clause uniquement constituée d'une équation booléenne. La propagation unitaire va consister à choisir une équation booléenne Q et une variable x de Q et d'appliquer jusqu'à saturation la règle R2 entre la clause unitaire $Q = x \oplus Q_1$ et des clauses $x \oplus Q_2 \vee C_2$. Précisément, on va remplacer toutes les occurences de x par $Q_1 \oplus 1$ dans toutes les autres clauses (ce qui correspond à la suppression de x dans la propagation unitaire classique). Si des clauses unitaires sont produites, on recommence le processus avec une clause unitaire produite en choisissant une nouvelle variable de cette clause, qui disparaîtra à son tour de la formule.

En fait, l'application de R2 entre une clause unitaire Q et une clause C permet aussi de détecter si Q subsume C et donc d'éliminer C de la formule, comme l'indique la proposition suivante.

Proposition 3 *Si l'application de R2 entre une x-clause unitaire Q et une x-clause C produit une tautologie alors Q subsume C.*

Pour que R2 s'applique, il faut qu'il existe une variable x telle que $Q = x \oplus Q_1$ et $C = x \oplus Q_2 \vee C_1$. La résolvante est $Q_1 \oplus Q_2 \oplus 1 \vee C_1$. Pour que cette résolvante soit tautologique, il faut que C_1 soit vraie quand $Q_1 \neq Q_2$. Or, $x \oplus Q_1$ est vraie quand $x \neq Q_1$ et alors :

- soit $x \neq Q_2$ et alors $x \oplus Q_2 \vee C_1$ est vrai.
- soit $x = Q_2$ donc $Q_2 \neq Q_1$ donc C_1 est vrai donc $x \oplus Q_2 \vee C_1$ est vrai.

Donc, quand $x \oplus Q_1$ est vrai, C est vrai donc $x \oplus Q_1$ subsume C. □

Comme on l'a vu, la résolvante se détecte comme étant une tautologie lorsque qu'elle est normalisée.

Nous pouvons maintenant définir XDPLL, une généralisation de DPLL :

```
XDPLL(F) :
  % Entrée : F, une formule XCNF
  % Sortie : vrai ssi F est satisfiable
   Si F est vide
    retourne vrai
   Si F contient la clause vide
      retourne faux
   Si F contient une clause unitaire Q
      x = choisit-variable(Q)
      Q1 = Q[x|0]
      retourne XDPLL(F[x|Q1+1])
   Q = choisit-equation(F)
   x = choisit-variable(Q)
   Q1 = Q[x|0]
   retourne XDPLL(F[x|Q1]) ou XDPLL(F[x|Q1+1])
```

Q[x|0] désigne l'équation où x est supprimé (en le mettant à 0). F[x|Q] désigne la transformation de la formule F où toutes les occurrences de x de F ont été remplacées par Q puis chacune des x-clauses a été renormalisée afin d'être simplifiée voire supprimée si c'est une tautologie.

4.2 Puissance de XDPLL

Proposition 4 *XDPLL est plus puissante que DPLL.*

Nous montrons que les instances d'Urquhart peuvent être aussi résolue en temps polynômial par XDPLL. En effet, une instance d'Urquhart F représente un ensemble d'équations booléennes sur des ensembles de variables X_i, c'est-à-dire une union d'ensembles $CNF(X_i)^{\oplus}$ ou $CNF(X_i)^{\oplus}1$. Lors de la recherche, à chaque point de choix, il suffit de choisir un ensemble X_i de variables de $CNF(X_i)^{\oplus}$ et d'explorer les deux branches de l'arbre de recherche $F \wedge Eq(X_i)$ et $F \wedge Eq(X_i) \oplus 1$. Supposons que F contient les clauses $CNF(X_i)^{\oplus}$ (le cas où ce serait $CNF(X_i)^{\oplus 1}$ est équivalent). Alors :

- Chacune des clauses de $CNF(X_i)^{\oplus}$ est subsumée par $Eq(X_i)$. Cela est détecté par la propagation unitaire. Donc dans le sous-arbre de recherche qui explore l'alternative $F \wedge Eq(X_i)$, on supprime toutes les clauses de $CNF(X_i)^{\oplus}$.
- La sous-formule $CNF(X_i)^{\oplus} \wedge Eq(X_i) \oplus 1$ est insatisfiable et ne contient que sept variables au maximum. Donc le sous-arbre de recherche qui explore l'alternative $F \wedge Eq(X_i) \oplus 1$ mène à un échec qui peut être détecté en temps constant si on choisit systématiquement une (équation constituée d'une seule) variable de X_i.

Globalement, l'arbre de recherche est déterminé par une succession d'alternatives où on se demande si chacune des équations booléennes induites par une instance du problème d'Urquhart est vraie ou fausse. Quand on suppose qu'elle est fausse, cela est réfuté en temps constant. On se retrouve donc à la fin à supposer que chacune des équations est vraie et il ne reste plus qu'à vérifier si le système d'équations est vrai, ce qui se fait par propagation unitaire. Si une instance d'Urquhart représente m équations booléennes, on a vérifié m fois qu'aucune équation n'est fausse dans l'instance puis on a résolu une fois le système d'équations. La résolution du problème s'est donc effectuée en temps polynômial. □

Nous avons montré que XDPLL pouvait être plus efficace en théorie que la recherche arborescente traditionnelle DPLL avec choix d'une seule variable. Cependant, il faut admettre qu'elle est plus difficile à mettre en œuvre que DPLL car il ne s'agit plus seulement de choisir une variable à chaque point de choix mais un sous-ensemble des variables. Cet aspect pratique est traité dans [6] où le choix des sous-ensembles de variables (qui constituent une équation) est basé sur le score VSIDS de leur variable.

5 Travaux connexes

Nous présentons quelques travaux ayant été effectués sur la prise en compte de relations d'équivalence ou de "ou exclusif" dans les instances SAT et les comparons avec notre approche.

Fondamentalement, ces travaux se résument à extraire syntaxiquement ou sémantiquement d'une formule CNF des équations booléennes puis à résoudre l'instance en faisant cohabiter une formule CNF et un ensemble d'équations booléennes. Les détections syntaxiques se font typiquement dans la phase de prétraitement de la formule, avant de lancer le solveur. Les détections sémantiques se font en chaque point de choix.

Le solveur eqSatz décrit dans [8] maintient un ensemble de chaînes d'équivalences de taille maximum 3 qu'il met à jour en chaque nœud de l'arbre de recherche. Il utilise ces chaînes pour faire de la propagation (par exemple par substitution quand une chaîne est de taille 2) et les détecte sémantiquement par propagation unitaire. On peut voir eqSatz comme une forme limitée mais très pragmatique de recherche arborescente à la XDPLL dans la mesure où il ne gère que des chaînes d'équivalence au plus ternaire (ce qui correspondrait à des équations booléennes sur trois variables).

Dans [7] est proposé un solveur qui gère séparément deux ensembles de clauses : des clauses CNF et des équations booléennes. Un solveur SAT s'occupe des clauses CNF et communique avec un autre solveur qui gère un système d'équations booléennes. Les deux solveurs communiquent afin que le filtrage dans l'un ait des répercussions dans l'autre. Les travaux dans [5] utilisent aussi cette technique des deux solveurs mais en permettant une détection sémantique plus poussée d'équations (non limitée en taille) et en indiquant comment obtenir des équations booléennes syntaxiquement pendant le pré-traitement de la formule (fondamentalement : en repérant les ensembles de clauses CNF qui équivalent à une équation booléennes). Dans [9], une approche syntaxique permet non seulement de détecter des chaînes d'équivalences mais aussi d'autres types de fonctions entre variables booléennes.

Si on considère la réécriture R1' et R2' des clauses R1 et R2 dans le cadre de la résolution étendue (cf section 3.2), il suffit de remarquer que $a \Leftrightarrow b \oplus c = a \oplus b \oplus c \oplus 1$, pour que R1' et R2' puissent être considérée comme mettant en jeu uniquement des clauses CNF et

des équations booléennes :

$$\frac{x_1 \vee ... \vee x_n \vee C_1 \qquad \overline{x_1} \vee ... \vee \overline{x_n} \vee C_2}{z_1 \vee ... \vee z_{n-1} \vee C_1 \vee C_2,\ 1 \oplus z_1 \oplus x_1 \oplus x_2,\ ...,\ 1 \oplus z_{n-1} \oplus x_1 \oplus x_n}$$

$$\frac{z_1 \vee C_1 \qquad z_2 \vee C_2 \qquad 1 \oplus z_1 \oplus x \oplus y_1 \qquad 1 \oplus z_2 \oplus x \oplus y_2}{z_3 \vee C_1 \vee C_2,\ z_3 \oplus y_1 \oplus y_2}$$

Les travaux de [7] peuvent donc aussi être interprétés comme une façon de faire partiellement de la XOR-résolution. La différence majeure est que l'ensemble des équations booléennes est donné et ils ne peuvent en produire d'autres qu'en les combinant entre elles après avoir affecté certaines variables pendant la recherche. Les points de choix de la recherche arborescente sont celui d'un choix de variable et pas un choix d'équations comme nous le faisons. En conséquence, cela revient à ne simuler l'application de la règle R2 que dans le cas où $C_1 = C_2 = 0$, celle qui correspond à de la propagation unitaire d'équations.

À note connaissance, les seules techniques d'inférences (syntaxiques) qui ont été proposées se limitent à la découverte d'équations booléennes à partir de clauses CNF et n'atteignent jamais le cas plus général que nous avons présenté où il s'agit d'obtenir une disjonction d'équations booléennes à partir d'autres disjonctions d'équations booléennes.

Dans cet article, nous n'avons pas abordé l'aspect pratique d'un solveur basé sur la XOR-résolution et les difficultés que cela engendre. Ces difficultés ont été abordés dans d'autres articles à travers la gestion des équations booléennes : comment choisir une équation booléenne à chaque point de choix [6], comment gérer efficacement des équations booléennes plutôt que des littéraux [7, 11, 5], etc.

6 Conclusion et perspectives

En définissant la XOR-résolution, nous avons défini un cadre théorique général pour la prise en compte d'équations booléennes (ou de chaînes d'équivalences) sous-jacentes à une formule que son expression CNF empêche de traiter efficacement. En définissant les x-clauses, nous intégrons le ou-exclusif dans les formules à traiter au lieu de maintenir d'un côté un ensemble de clauses CNF et d'un autre côté un ensemble d'équations booléennes, de raisonner sur les deux ensembles séparément et de les faire communiquer. Les deux règles de la XOR-résolution unifient tous les traitements possibles et permettent d'inférer plus de clauses que ce que font les solveurs dédiés jusqu'à présent.

Une fois posé ce cadre, nous avons indiqué comment la recherche arborescente pouvait simuler l'application de la règle R2 de la XOR-résolution, qui généralise la règle de résolution standard et permet donc d'être plus efficace en théorie. Il nous reste à voir comment on pourrait aussi simuler l'application de la règle R1. Une façon d'y parvenir serait d'abord de définir une variante de DP60 [3], un algorithme complet qui applique la règle de résolution jusqu'à trouver la clause vide (problème insatisfiable) ou jusqu'à obtenir la formule vide (problème satisfiable) en éliminant une par une les variables de la formule. Nous pourrions nous en inspirer pour définir une méthode complète permettant d'appliquer R1 et R2 pour décider de la satisfiabilité d'une formule CNF. Cependant, il n'est pas évident de procéder par éliminations successives de variable de la formule. En effet, seule la règle R2 permet de se "débarasser" d'une variable x mais elle n'est pas toujours applicable et c'est alors R1 qu'il faut appliquer, mais toutes les variables subsistent dans la résolvante.

La prochaine étape de nos travaux pourrait donc être de trouver une variante de DP60 pour la XOR-résolution afin d'en déduire une méthode de recherche arborescente qui simule complètement la XOR-résolution, à la façon dont DPLL avait été défini à partir de DP60.

Références

[1] Vasek Chvátal and Endre Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4) :759–768, 1988.

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

[3] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3) :201–215, 1960.

[4] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39 :297–308, 1985.

[5] Marijn Heule and Hans van Maaren. Aligning CNF- and equivalence-reasoning. In *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, pages 145–156, 2004.

[6] Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Knowledge compilation for model counting : Affine decision trees. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

[7] Tero Laitinen, Tommi A. Junttila, and Ilkka Niemelä. Conflict-driven xor-clause learning. In

Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings, pages 383–396, 2012.

[8] Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 291–296, 2000.

[9] Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Sais. Recovering and exploiting structural knowledge from CNF formulas. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, pages 185–199, 2002.

[10] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1) :23–41, 1965.

[11] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, pages 244–257, 2009.

[12] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logics*, pages 115–125, 1968.

[13] Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1) :209–219, 1987.