UTILISATION DE LA LOGIQUE PROPOSITIONNELLE POUR RESOUDRE DES INEQUATIONS LINEAIRES

Mariam Bouzid – Justine Evrard

satō

UFR des Sciences – Université d'Angers
Licence informatique – Année scolaire 2015-2016

AVANT-PROPOS

Encadré par le maître conférencier Igor Stephan, ce travail d'étude et de recherche s'inscrit dans le cadre d'un stage clôturant les trois années de licence informatique à l'université d'Angers. Le sujet porte sur le travail accompli par les quatre chercheurs japonais Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa et Mutsunori Banbara, dans l'article ***Compiling finite linear CSP into SAT***, publié en 2009.

Prenant conscience du contexte de cette étude, nous essayerons de lever toutes ambiguïtés concernant la méthodologie des quatre chercheurs japonais cités précédemment. L'extraction de ce substrat théorique nous préparera méthodiquement en vue de créer notre propre script en PHP illustrant le fonctionnement de l'encodage décrit dans cette publication.

TABLE DES MATIERES

INTRODUCTION

Le domaine de notre étude se partage entre celui des mathématiques et celui de l'informatique théorique. En effet, nous allons résoudre des problèmes d'aspect mathématiques grâce à la logique informatique, et plus précisément la logique propositionnelle, bien que la limite entre l'ordre 0 et l'ordre 1 devienne parfois ténue. De manière plus formelle, nous voulons transformer un problème de satisfaction de contraintes (CSP) linéaire sous nombres entiers en un problème de satisfiabilité SAT.

Une telle manipulation nécessite d'encoder les contraintes du CSP en formules propositionnelles, permettant ainsi l'utilisation d'un solveur SAT. La méthode d'encodage proposée dans cette étude est nommée *order encoding* qui représente par des symboles propositionnels des comparaisons de la forme $x \leq a$, x étant une variable entière et a une valeur entière. Cet encodage diffère à celui – plus classique – du *sparse encoding* qui se base plutôt sur de l'assignation de valeurs aux variables entières comme $x = a$.

I - EXPLORATION ET ANALYSE DU SUJET

1. CONTEXTE

a. Complexité d'un problème

Un problème est spécifié *combinatoire* lorsque sa résolution se heurte à une explosion du nombre de combinaisons à explorer. Cette notion est caractérisée par la théorie de la complexité qui propose une classification en fonction de l'estimation, dite dans le pire des cas, du nombre d'instructions à exécuter pour résoudre les instances du problème lié.

La classe qui nous intéresse dans ce contexte de résolution de problèmes de décision est la classe NP. Celle-ci contient l'ensemble des problèmes polynomiaux non déterministes, c'est-à-dire pouvant être résolus par un algorithme de complexité polynomiale par une machine non déterministe - une machine capable d'exécuter en parallèle un nombre fini d'alternatives. Intuitivement, cela signifie que la résolution des problèmes de NP peut nécessiter l'examen d'un très grand nombre de cas, chacun en un temps polynomial.

Cependant, certains problèmes de NP apparaissent plus difficiles que d'autre dans le cas où aucun algorithme polynomial pour machine non-déterministe n'est trouvé. Ces problèmes définissent alors la classe NP-complet, dont la résolution de ceux-ci implique un nombre exponentiel de cas.

b. Problème de satisfaction de contrainte

Les problèmes de satisfaction de contrainte, notés CSPs pour *Constraint Satisfaction Problem*, sont des problèmes définis par des contraintes dont la solution doit respecter ces dernières. Dans le cas général et de façon formelle, un tel problème est défini par un triplé (X, D, C) où X est un ensemble fini de variables - les inconnues du problème ; D une fonction associant à chaque variable $x_i \in X$ son domaine de définition $D(x_i)$; C un ensemble fini de contraintes.

Les CSPs ne sont pas tous des problèmes combinatoires et leur complexité théorique dépend de la nature des contraintes et des domaines de variables. Par exemple, lorsque les contraintes sont des équations, ou inéquations linéaires, et les domaines des variables continus, les CSPs peuvent être polynomiaux. Le cadre très général des CSPs permet de modéliser de nombreux problèmes réels. Ainsi, exprimer un problème mathématique sous la forme d'un CSP est un moyen permettant de le résoudre.

c. Problème SAT

Un problème SAT, connu aussi sous le nom de *boolean / propositional SATisfiability problem*, est un problème de logique propositionnelle. Ce type de problème a été le premier prouvé NP-complet, grâce au théorème de *Cook* (1971). Ce dernier a ainsi démontré que tout problème appartenant à la classe NP-complet peut se réduire polynomialement en un problème SAT.

Formellement, un problème SAT consiste à décider si une formule F en forme normale conjonctive est satisfiable, c'est-à-dire s'il existe une valuation v telle que $v^*(F) = vrai$ - v étant donc un modèle de F, noté $v \vDash F$.

d. Solveur SAT

Un solveur SAT est un programme qui détermine si une formule propositionnelle sous forme normale conjonctive est satisfiable. Si tel est le cas, celui-ci nous retourne une valuation qui satisfait cette formule. De cette manière, il suffit de transformer un problème en une instance de SAT, si possible, et de laisser un solveur déterminer sa satisfiabilité, plutôt que d'écrire un algorithme pour chaque problème.

Cependant, les solveurs SAT modernes implémentent et optimisent de diverses façons un même algorithme, celui de *Davis-Logemann-Loveland*, pouvant être illustré par un arbre binaire dans lequel chaque nœud représente un appel récursif à cette même procédure, qui est une extension de l'algorithme de *Davis-Putnam* produisant parfois un nombre trop conséquent de résolvantes.

2. MORPHOLOGIE

a. Cadre mathématique

Dans un premier temps, il est nécessaire d'introduire la notion mathématique d'inéquation linéaire de façon formelle. Par ce biais, nous pourrons définir l'espace de recherche de notre étude.

DEFINITION 1 (INEQUATION LINEAIRE)

Dans $\mathbb{Z}^n$, une inéquation linéaire est exprimée par

$$f(\vec{x}) \leq c$$

où f est une application linéaire, $\vec{x} = (x_1, \dots, x_n)$ un vecteur de $\mathbb{Z}^n$ et c une constante entière relative.

De manière plus concrète, une inéquation linéaire s'écrit comme suit

$$\sum_{i=1}^{n} a_i x_i \leq c$$

où les x_i sont des inconnues, les a_i des coefficients entiers relatifs et c une constante entière relative.

b. Ensembles de référence

La définition étant posée, nous allons déterminer les ensembles pertinents dans le contexte de satisfiabilité des inéquations linéaires.

- $\boldsymbol{Z} \subseteq \mathbb{Z}$ *: un ensemble d'entiers relatifs.*
- $\mathcal{V}$ *: un ensemble infini dénombrable*[1] *de variables entières à valeurs dans* $\boldsymbol{Z}$.
- $E(V)$ *: un ensemble d'expressions linéaires dont les variables sont dans* $V \subset \mathcal{V}$*, et les expressions algébriques, de la forme*

  $$\sum_{i=1}^{n} a_i x_i$$

  où $a_i \in \mathbb{Z} \setminus \{0\}$ *et* $x_i \in V$, $\forall\, x_i \neq x_j$ *(tous distincts).*
- $\boldsymbol{B}$ **:** *un ensemble de valeurs booléennes* $\{\boldsymbol{true}, \boldsymbol{false}\}$.
- $\mathcal{B}$ *: un ensemble infini dénombrable*[1] *de variables booléennes à valeurs dans* $\boldsymbol{B}$.
- $L(V, B)$ *: un ensemble de littéraux sur* $V \subset \mathcal{V}$ *et* $B \subset \mathcal{B}$*, de la forme :*
  - $p \in B$*, une variable booléenne*
  - $\neg p \in B$*, la négation d'une variable booléenne*
  - $\{e \leq c | e \in E(V), c \in \boldsymbol{Z}\}$*, une comparaison (toujours exprimée sous forme positive)*
- $C(V, B)$ *: un ensemble de clauses*[2] *sur* $V \subset \mathcal{V}$ *et* $B \subset \mathcal{B}$*, celles-ci définies comme la disjonction de littéraux* $l \in L(V, B)$.

[1] Pour l'introduction de nouvelles variables.

[2] Les variables booléennes apparaissant dans les clauses sont traitées comme des variables libres.

c. Représentation

Les inéquations linéaires discrètes peuvent facilement être réduites à un problème de satisfaction de contraintes. En effet, leurs caractères discrets nous permettent de définir un domaine borné continu dans lequel les variables prendront leurs valeurs, et la notion d'inégalité pourra s'exprimer comme une clause.

DEFINITION 2 (CSP LINEAIRE FINIE)

Un CSP (Problème de Satisfaction de Contrainte) est défini par un quintuple (V, ℓ, u, B, S) *où :*

- V *est un sous ensemble fini de variables entières de* $\mathcal{V}$.
- $\ell : V \to \boldsymbol{Z}$ *est une application représentant la borne inférieure du domaine d'une variable entière.*
- $u : V \to \boldsymbol{Z}$ *est une application représentant la borne supérieure du domaine d'une variable entière.*
- B *est un sous ensemble fini de variables booléennes de* $\mathcal{B}$.
- S *est un ensemble fini de clauses représentant la contrainte devant être satisfaite.*

Ces cinq points correspondent en fait aux trois données d'un CSP classique défini comme le triplet (X, D, C) *où* $X \equiv V$ *l'ensemble des variables entières,* D *le domaine de ces variables entières, i.e. pour chaque* $x_i \in V, D(x_i) = [\![\ell(x_i); u(x_i)]\!]$ *, et* $C(X) \equiv S(V, B)$ *l'ensemble des clauses dont les variables sont soit dans* V*, soit dans* B.

On fournit une extension aux applications ℓ et u définies précédemment qui permet de déterminer les bornes d'une expression linéaire donnée. La publication ne détaillant pas assez l'utilisation de cette extension, nous avons dû trouver l'implémentation de celle-ci directement dans le code source du programme *Sugar*.

DEFINTION 3 (BORNES DES EXPRESSIONS LINEAIRES)

Soit $e \in E(V)$ *de la forme* $\sum_{i=1}^{n} a_i x_i + c$ *où* $a_i \in \mathbb{Z}^*$ *,* $c \in \mathbb{Z}$ *, et* $x_i \in V$ *pour tout* $i \in [\![1; n]\!]$. *Ainsi l'extension* $\ell^* : E(V) \to \boldsymbol{Z}$ *de l'application* ℓ *s'exprime comme :*

$$\ell^*(e) = \ell\left(\sum_{i=1}^{n} a_i x_i + c\right) = \sum_{i=1}^{n} (a_i x_i)^* + c$$

Où l'on définit la translation $(\,)^*$ *de* ℓ^* *telle que*

$$(ax)^* = \begin{cases} a \times \ell(x) & si\ a > 0 \\ a \times u(x) & si\ a < 0 \end{cases}.$$

Et l'extension de u *notée* $u^* : E(V) \to \boldsymbol{Z}$ *s'exprime comme :*

$$u^*(e) = u\left(\sum_{i=1}^{n} a_i x_i + c\right) = \sum_{i=1}^{n} (a_i x_i)^* + c$$

Où l'on définit la translation $(\,)^*$ *propre à* u^* *telle que*

$$(ax)^* = \begin{cases} a \times u(x) & si\ a > 0 \\ a \times \ell(x) & si\ a < 0 \end{cases}.$$

DEFINITION 4 (SAT)

Un SAT (Problème de Satisfiabilité) est défini comme une instance de CSP sans variable entière, c'est-à-dire, une forme particulière de CSP exprimée par le quintuple $(\emptyset, \emptyset, \emptyset, B, S)$.

DEFINITION 5 (COMPARAISON PRIMITIVE)

Une comparaison primitive est une comparaison restreinte à la forme $x \leq c$ *où* x *est une variable entière satisfaisant* $\ell(x) \leq x \leq u(x)$ *et* c *un entier relatif constant. La valeur de ce dernier est restreinte à* $\ell(x) \leq c \leq u(x) - 1$. *Or, il est préféré dans cette étude d'utiliser un intervalle plus large tel que* $\ell(x) - 1 \leq c \leq u(x)$.

3. SEMANTIQUE

La sémantique prend une place significative dans notre étude puisque c'est dans celle-ci que nous pourrons apporter une cohérence vis-à-vis de l'objet mathématique modélisé.

a. Satisfiabilité

La notion de satisfiabilité est intrinsèquement liée aux problèmes SAT. De ce fait, nous devons la définir de façon rigoureuse.

DEFINITION 6 (ASSIGNATION)

Une assignation est un couple (α, β) d'applications telles que $\alpha : V \to \mathbf{Z}$ affecte à une variable entière une valeur de $\mathbf{Z}$ et $\beta : \mathcal{B} \to \mathbf{B}$ affecte à une variable booléenne une valeur de $\mathbf{B}$.

DEFINITION 7 (SATISFIABILITE)

Soit un CSP (V, ℓ, u, B, S). Une clause $C \in C(V, B)$ est satisfiable par l'assignation (α, β) si cette dernière rend vraie la clause C pour tout $\in V$, $\ell(x) \leq \alpha(x) \leq u(x)$, ce qui est noté $(\alpha, \beta) \vDash C$.

Une clause C est satisfiable si C est satisfiable par une assignation. Un ensemble de clause est satisfiable lorsque toutes ses clauses sont satisfiables par une même assignation. Une formule propositionnelle est satisfiable si sa forme clausale est satisfiable (théorème du cours de logique du premier ordre). De ce fait, un CSP est satisfiable si son ensemble de clauses S est satisfiable.

b. Réduction d'une inéquation linéaire

La difficulté dans cette partie est de réduire une inéquation linéaire à une forme normale conjonctive de comparaison primitive, facilitant ainsi la manipulation de notre problème.

LEMME 1 (DICHOTOMIE D'UNE INEQUATION LINEAIRE)

Soit (V, ℓ, u, B, S) un CSP, alors pour toute assignation (α, β), pour toutes expressions linéaires $e, f \in E(V)$ et pour tout entier $c \geq \ell(e) + \ell(f)$,

$$(\alpha, \beta) \vDash e + f \leq c$$

$$\Leftrightarrow (\alpha, \beta) \vDash \bigwedge_{a+b=c-1} (e \leq a \vee f \leq b)$$

Les paramètres a et b sont à valeurs dans $\mathbb{Z}$ et satisfont $a + b = c - 1$, $\ell(e) - 1 \leq a \leq u(e)$, $\ell(f) - 1 \leq b \leq u(f)$. La conjonction représente $\top$ s'il n'existe pas de tels a et b.

PROPOSITION 1 (RECURSIVITE DE LA DICHOTOMIE)

Soit (V, ℓ, u, B, S) un CSP, alors pour toute assignation (α, β), pour toute expression linéaire $\sum_{i=1}^{n} a_i x_i \in E(V)$ et pour tout entier $c \geq \ell(\sum_{i=1}^{n} a_i x_i)$, on a :

$$(\alpha, \beta) \vDash \sum_{i=1}^{n} a_i x_i \leq c$$

$$\Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{n} b_i = c-n+1} \bigvee_i (a_i x_i \leq b_i)^{\#}$$

Les paramètres b_i sont à valeurs dans $\boldsymbol{Z}$ et satisfont $\sum_{i=1}^{n} b_i = c - n + 1$ et pour tout i, $\ell(a_i x_i) - 1 \leq b_i \leq u(a_i x_i)$. La translation $(\,)^{\#}$est définie par

$$(ax \leq b)^{\#} \equiv \begin{cases} x \leq \left\lfloor \frac{b}{a} \right\rfloor & (a > 0) \\ \neg\left(x \leq \left\lceil \frac{b}{a} \right\rceil - 1\right) & (a < 0) \end{cases}$$

PREUVES

1) *La satisfiabilité par l'assignation (α,β) de $\sum_{i=1}^{n} a_i x_i \leq c$ est équivalente à la satisfiabilité par l'assignation (α,β) de $\bigwedge \bigvee_i a_i x_i \leq b_i$.*

Méthodologie : Pour démontrer cette équivalence, on utilise la récurrence sur $n \geq 1$ et la propriété du Lemme 1.

Soit pour tout $n \geq 1$, la proposition

$$P(n) : (\alpha,\beta) \vDash \sum_{i=1}^{n} a_i x_i \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{n} b_i = c-n+1} \bigvee_i a_i x_i \leq b_i$$

Cas de base n°1 : Pour $n = 1$, la proposition se formule par

$$P(1) : (\alpha,\beta) \vDash \sum_{i=1}^{1} a_i x_i \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{1} b_i = c-1+1} \bigvee_i a_i x_i \leq b_i$$

$$P(1) : (\alpha,\beta) \vDash a_1 x_1 \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{1} b_i = c} \bigvee_i a_i x_i \leq b_i$$

$$P(1) : (\alpha,\beta) \vDash a_1 x_1 \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{b_1 = c} a_1 x_1 \leq b_1$$

$$P(1) : (\alpha,\beta) \vDash a_1 x_1 \leq c \Leftrightarrow (\alpha,\beta) \vDash a_1 x_1 \leq c$$

⇨ *Cette proposition est toujours vraie.*

Cas de base n°2 : Pour $n = 2$, la proposition est

$$P(2) : (\alpha,\beta) \vDash \sum_{i=1}^{2} a_i x_i \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{2} b_i = c-2+1} \bigvee_i a_i x_i \leq b_i$$

$$P(2) : (\alpha,\beta) \vDash a_1 x_1 + a_2 x_2 \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{2} b_i = c-1} \bigvee_i a_i x_i \leq b_i$$

$$P(2) : (\alpha,\beta) \vDash a_1 x_1 + a_2 x_2 \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{b_1 + b_2 = c-1} a_1 x_1 \leq b_1 \vee a_2 x_2 \leq b_2$$

⇨ *Cette proposition est tout à fait analogue au Lemme 1.*

Hypothèse de récurrence : Supposons la proposition $P(n)$ vraie pour tout $n > 2$.

<u>Hérédité :</u> Soit la proposition

$$P(n+1) : (\alpha,\beta) \vDash \sum_{i=1}^{n+1} a_i x_i \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{n+1} b_i = c-(n+1)+1} \bigvee_i a_i x_i \leq b_i$$

$$P(n+1) : (\alpha,\beta) \vDash \sum_{i=1}^{n} a_i x_i + a_{n+1} x_{n+1} \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{n+1} b_i = c-n} \bigvee_i a_i x_i \leq b_i$$

Par le Lemme 1,

$$(\alpha,\beta) \vDash \sum_{i=1}^{n} a_i x_i + a_{n+1} x_{n+1} \leq c \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{a+b=c-1} \sum_{i=1}^{n} a_i x_i \leq a \vee a_{n+1} x_{n+1} \leq b$$

Par hypothèse de récurrence,

$$(\alpha,\beta) \vDash \sum_{i=1}^{n} a_i x_i \leq a \Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{n} b_i = a-n+1} \bigvee_i a_i x_i \leq b_i$$

D'où,

$$(\alpha,\beta) \vDash \sum_{i=1}^{n} a_i x_i + a_{n+1} x_{n+1} \leq c$$

$$\Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{a+b=c-1} \left(\bigwedge_{\sum_{i=1}^{n} b_i = a-n+1} \bigvee_i a_i x_i \leq b_i \right) \vee a_{n+1} x_{n+1} \leq b$$

En regroupant les conjonctions,

$$\Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{n} b_i + a + b = c-1+a-n+1} \left(\bigvee_i a_i x_i \leq b_i \vee a_{n+1} x_{n+1} \leq b \right)$$

Et les disjonctions,

$$\Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{n} b_i + b = c-n} \left(\bigvee_i a_i x_i \leq b_i \right)$$

$$\Leftrightarrow (\alpha,\beta) \vDash \bigwedge_{\sum_{i=1}^{n+1} b_i = c-n} \left(\bigvee_i a_i x_i \leq b_i \right)$$

⇨ *Ainsi, $P(n+1)$ est vérifié.*

<u>Conclusion :</u> La proposition est vraie aux rangs $n = 1$ et $n = 2$ et elle est héréditaire, donc elle est vraie pour tout $n \geq 1$. ■

2) *La satisfiabilité de chaque comparaison $a_i x_i \leq b_i$ est équivalente à la satisfiabilité de sa translation $(a_i x_i \leq b_i)^{\#}$. On observe qu'il existe deux cas distincts pour cette translation paramétrés par le signe de a.*

Mathématiquement, nous avons :

$$(ax \leq b) = \begin{cases} \left(x \leq \dfrac{b}{a}\right) & si\ a > 0 \\ \left(x \geq \dfrac{b}{a}\right) = \left(x > \dfrac{b}{a} - 1\right) & si\ a < 0 \end{cases}$$

Or, d'un point de vue logique, la négation de $\leq$ est $>$. Ainsi, nous conservons le symbole $\leq$ afin de construire la forme $\neg\left(x \leq \frac{b}{a} - 1\right)$ pour le cas où a est négatif. De plus, le fait que nous soyons dans l'ensemble $\mathbb{Z}$ nous force à seulement garder la partie entière par excès ou par défaut, suivant le signe de a, de notre quotient. De cette manière, nous obtenons :

$$(ax \leq b)^{\#} \equiv \begin{cases} x \leq \left\lfloor \frac{b}{a} \right\rfloor & (a > 0) \\ \neg\left(x \leq \left\lceil \frac{b}{a} \right\rceil - 1\right) & (a < 0) \end{cases} . \blacksquare$$

4. ENCODAGE

Pour encoder un problème SAT, il est nécessaire d'exprimer toutes ses contraintes en une unique forme clausale permettant ainsi d'introduire cette dernière dans un solveur SAT. L'encodage doit également garantir la connexité entre le passage de la sémantique à la syntaxe. Ceci implique que les notions mathématiques abordées doivent être préservées, d'où la description d'axiomes pour encoder la relation d'ordre.

a. Elimination des variables entières

Un problème SAT étant défini comme un CSP sans variable entière, l'encodage d'un CSP vers SAT doit donc éliminer ce type de variable, sans pour autant changer la satisfiabilité du problème.

On définit ainsi un ensemble de variables booléennes B'

$$B' = \{p(x,i) | x \in V, \forall i \in [\![\ell(x) - 1; u(x)]\!]\}.$$

Cet ensemble B' est le résultat du remplacement de chaque comparaison primitive de la forme $x \leq i$, $x \in V, \ell(x) - 1 \leq i \leq u(x)$ par une variable booléenne $p(x,i)$ provenant de $\mathcal{B}$.

Supprimer ces variables entières implique la suppression de la notion d'ordre des entiers relatifs dans $\mathbb{Z}$ ce qui nous amène à créer une nouvelle relation d'ordre en terme logique, i.e. un ensemble de clauses comme énoncé dans la définition suivante.

b. Relation d'ordre

La relation d'ordre que nous allons définir est une relation d'ordre totale sur une partie $\mathcal{P} = [\![\ell(x); u(x)]\!]$ du corps $\mathbb{Z}$ des entiers relatifs.

DEFINITION 8 (RELATION D'ORDRE TOTALE, BORNES)

Pour chaque variable entière $x \in V$, il est nécessaire de définir un axiome représenté par un ensemble de clauses

$$A(x) = \{[\neg p(x, \ell(x) - 1)], [p(x, u(x))]\} \cup \{[\neg p(x, i-1), p(x,i)] \mid \forall i \in [\![\ell(x); u(x)]\!]\}$$

Cet axiome exprime la notion de bornes et la relation d'ordre permettant de limiter l'affectation de nos variables à leur domaine de définition.

PREUVE

Les axiomes suivant représentent la notion de bornes $\ell(x) \leq x \leq u(x)$:

$$\neg p(x, \ell(x) - 1) = \neg(x \leq l(x) - 1) \equiv l(x) \leq x$$

$$p(x, u(x)) = x \leq u(x)$$

La relation d'ordre totale[3] est exprimée par la clause $[\neg p(x, i-1), p(x,i)]$:

$$\neg p(x, i-1) = \neg(x \leq i - 1) \equiv (i \leq x)$$

$$p(x,i) = (x \leq i)$$

$$d'où\ ((i \leq x) \vee (x \leq i))$$

[3] Une relation d'ordre total sur un ensemble E est une relation d'ordre $\leq$ pour laquelle deux éléments de E sont toujours comparables, c'est-à-dire que $\forall x, y \in E, ((x \leq y) \vee (y \leq x))$.

c. Préservation d'une forme clausale générale

L'utilisation de la récursivité de la dichotomie sur toute inéquation linéaire - non réduite à une comparaison primitive - d'une clause provoque la rupture structurelle de celle-ci, le résultat étant une union de FNC. Pour pallier à cette complication, nous élaborons un ordonnancement de traitements sur cette clause.

LEMME 2

Soient (V, ℓ, u, B, S) *un CSP et une clause* $C = [L_1, \dots, L_n]$ *et*

$\Sigma = \{[p_1; \dots; p_n], [L_1; \neg p_1], \dots, [L_n; \neg p_n]\}$, *alors* C *est satisfiable* $\Leftrightarrow$ Σ *est satisfiable.*

PREUVE

$\Rightarrow$ *Soit* (α, β) *une assignation telle que* $(\alpha, \beta) \vDash L_i$ *pour tout* $i \in [\![1; n]\!]$ *et* $\beta(p_j) = \begin{cases} \textbf{\textit{true}} \text{ si } i = j \\ \textbf{\textit{false}} \text{ sinon} \end{cases}$. *Dans ce cas,* $(\alpha, \beta) \vDash \Sigma$.

$\Leftarrow$ *Soit* (α, β) *une assignation telle que* $(\alpha, \beta) \vDash \Sigma$. *Il existe donc un* p_i, $i \in [\![1; n]\!]$, *tel que* $\beta(p_i) = \top$. *Or,* $(\alpha, \beta) \vDash \neg p_i \vee L_i$, *donc* $(\alpha, \beta) \vDash L_i$. *Dans ce cas,* $(\alpha, \beta) \vDash C$. ■

DEFINITION 9 (TRANSFORMATION D'UNE CLAUSE EN UNE FNC)

Soient un CSP (V, l, u, B, S), *une clause* $C = [L_1; \dots; L_n]$ *appartenant à* S *et* $\Sigma = \{[p_1; \dots; p_n], [\neg p_1; L_1], \dots, [\neg p_n; L_n]\}$. *On définit l'ensemble de clause suivant :*

$$\Sigma' = \{[p_1; \dots; p_n]\} \cup \bigcup_{i=1}^{n} \Sigma_i$$

Où chaque L_i *de la forme comparaison de la clause* C *est transformé en une FNC de comparaisons primitives* F_i *par la récursivité de la dichotomie, et chaque* Σ_i *représentant ainsi la forme clausale de chaque formule* $\neg p_i \vee F_i$.

L'utilisation de cet ensemble de clauses Σ nous permet d'éviter le remaniement d'une disjonction de FNC de comparaisons primitives provenant de la clause C. De ce fait cela réduit raisonnablement le nombre de ces dernières qui sont contenues dans la forme clausale Σ'.

PROPOSITION 2

Soient un CSP (V, l, u, B, S), *une clause* $C = [L_1; \dots; L_n]$ *appartenant à* S *et un ensemble de clauses* Σ' *tel qu'il est défini précédemment. Alors, il existe une assignation* (α, β) *telle que*

$$(\alpha, \beta) \vDash C \Leftrightarrow (\alpha, \beta) \vDash \Sigma'.$$

PREUVE

D'après le lemme 2, si C *est satisfiable alors,* Σ *(c.f. définition 9) l'est aussi. Et d'après la proposition 1, on sait que la satisfiabilité de chaque* L_i *est équivalente à celle de chaque* F_i *correspondantes.* ■

DEFINITION 10 (FNC GENERALE)

Soit un CSP (V, l, u, B, S) *où* S *est l'ensemble de clauses défini comme* $S = \{C_i \mid i \in [\![1; n]\!]\}$.

On définit l'ensemble de clause suivant :

$$S^* = \bigcup_{i=1}^{n} \Sigma'_i$$

Où chaque C_i appartenant à S est transformé en un ensemble de clause Σ'_i comme énoncé dans la définition précédente.

PROPOSITION 3

Soient un CSP (V, l, u, B, S) et un ensemble de clauses S^* tel qu'il est défini précédemment. Alors, il existe une assignation (α, β) telle que

$$(\alpha, \beta) \vDash S \Leftrightarrow (\alpha, \beta) \vDash S^*.$$

PREUVE

Trivialement, la satisfiabilité de chaque C_i est équivalente à celle de chaque Σ'_i, par la proposition 2. ■

d. CSP vers SAT

Ayant tous les éléments nécessaires à la transformation d'un CSP vers SAT, nous pouvons désormais établir le lien entre les deux types de problème grâce à la proposition suivante.

PROPOSITION 4

Soient (V, ℓ, u, B, S) un CSP composé de contraintes exprimées uniquement par une formule en forme normale conjonctive de comparaisons primitives, et S^* la forme clausale de la formule obtenue à partir de la définition 9. Considérons aussi $A = \bigcup_{x \in V} A(x)$. Alors,

$$(V, \ell, u, B, S) \text{ est satisfiable} \Leftrightarrow (\emptyset, \emptyset, \emptyset, B \cup B', S^* \cup A) \text{ est satisfiable.}$$

Ce qui pourrait être formulé par $(\alpha, \beta) \vDash S \Leftrightarrow (\emptyset, \beta^*) \vDash S^* \cup A$;

Où l'on définit les applications de ces assignations comme suit :

- *Pour tout $p(x, c) \in B'$, on a $\alpha(x) = min\{c \mid \ell(x) \leq c \leq u(x)\}$*
- *$\beta : B \to \boldsymbol{B}$*
- *$\beta^* : B \cup B' \to \boldsymbol{B}$, une extension de β, tel que*

$$\beta^*(p) = \begin{cases} \beta(p) & (p \in B) \\ \alpha(x) \leq c & (p = p(x, c) \in B') \end{cases}$$

II - EXPLOITATION DU SUJET

A bien des égards, la programmation est un processus méthodologique concret qui, pour autant, nécessite une réflexion scientifique. Nous avons convenu dans un premier temps de souligner l'importance de la méthodologie de la programmation, en ce sens qu'elle exprime la connexité entre la théorie et la pratique ; puis de développer l'aspect technique, celui-ci soulignant les limites du sujet.

Notre démarche fut double : à la fois satisfaire la syntaxe de sortie et être conforme aux concepts décrits dans l'article. Nous avons donc adopté simultanément la méthode descendante (*top-down*), partir du préexistant, et la méthode ascendante (*bottom-up*), création des briques de bases de nos concepts. De fait, la première décision à prendre concernait l'interfaçage de la syntaxe d'un fichier *dimacs*, et la deuxième fut de modéliser nos différents concepts mathématiques, sans perdre d'informations à leurs égards.

1. LANGAGE DE PROGRAMMATION

Plus qu'un choix esthétique, syntaxique ou technique, le langage est un choix d'ordre méthodologique qui doit être adapté en fonction du problème : à chaque problématique son langage. Le sujet fait autorité sur le programmeur : si la bonne connaissance d'un langage peut orienter celui-ci, le sujet suffit à motiver la préférence d'un langage à un autre.

Dans la mesure où nous disposions déjà d'une bonne connaissance de ce langage sous toutes ses formes, *PHP* fut adopté pour notre projet. En effet, portable et faiblement typé, ce qui lui confère une grande flexibilité, il a également les ressources pour une utilisation en orienté objet et comme script *shell* ; sans compter qu'il manipule de manière aisée les expressions régulières.

De surcroît, l'aspect *développement web* de ce langage permettrait de déployer ce programme à l'image d'une *API web* où ce code ferait office de *backend*. Il faudrait cependant quelque peu le modifier, afin d'avoir des sorties sous format *JSON* plutôt que texte, l'exploitation par du JavaScript en *frontend* devenant ainsi possible.

2. IMPLEMENTATION

Nous avons choisi de représenter chaque entité issue de la théorie par une classe d'un point de vue *orienté objet*. Ce paradigme nous permet de construire une organisation rigoureuse de nos éléments et de leurs fonctionnalités. Le script principal exploite de cette façon les objets générés par ces modèles.

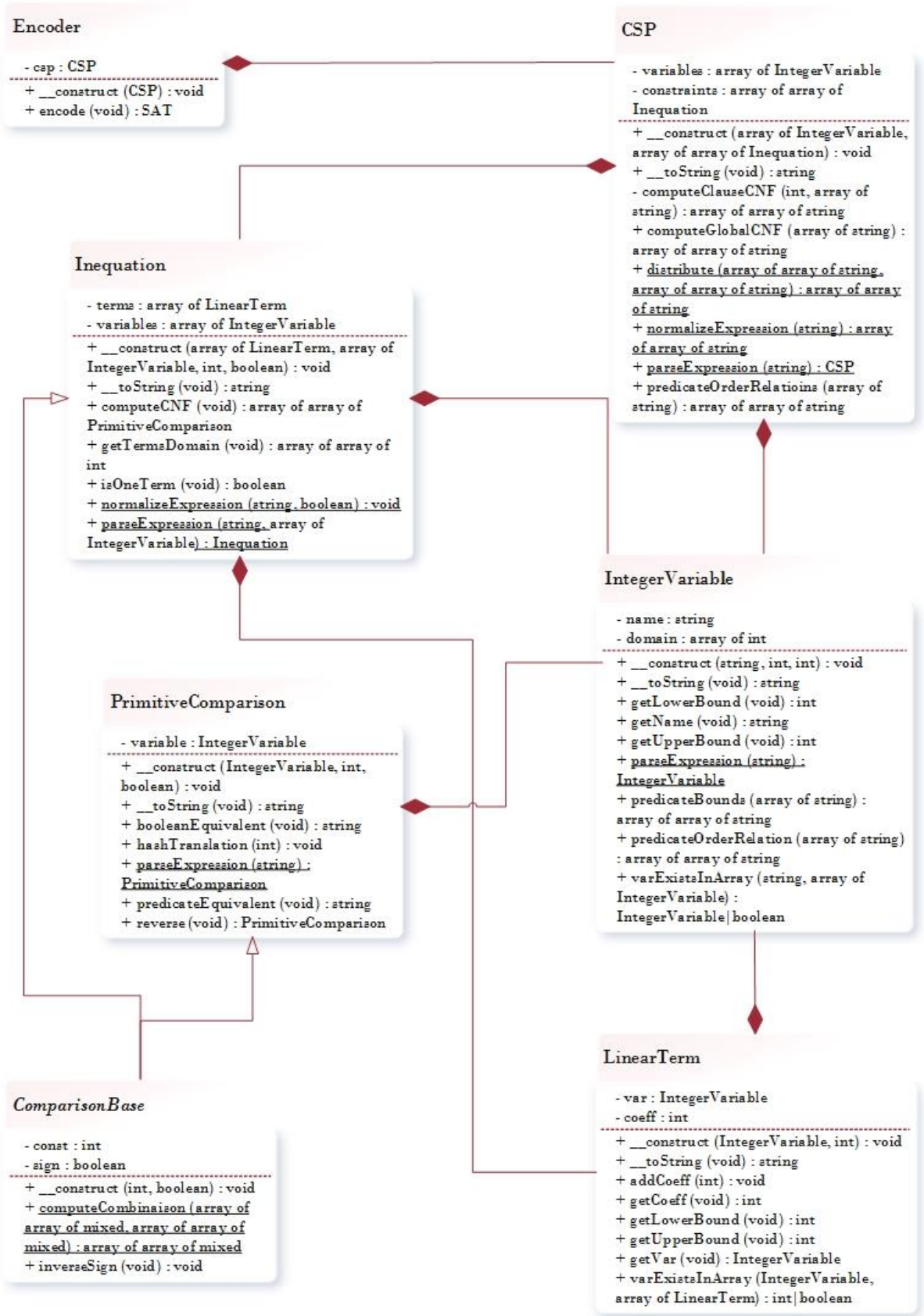
L'architecture définie, nous avons agencé les différentes composantes entre elles : chacune d'entre elles correspondant à un concept pouvant rassembler d'autres composantes. Ceci exprime la bijection qui existe entre les différentes structures représentées.

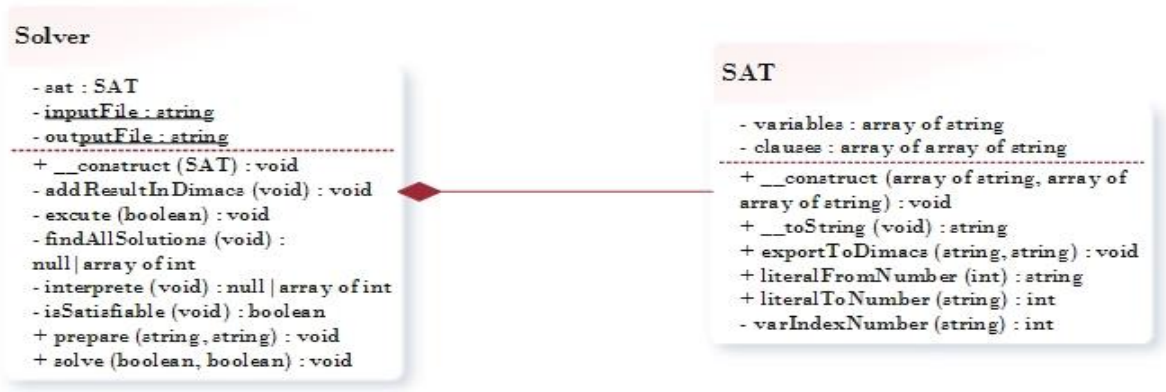
Expliquer en détail les mécanismes de notre code source n'aurait pas grand intérêt dans ce rapport, hormis celui de répéter et traduire les commentaires que nous avons disposé dans ce premier. C'est pourquoi nous nous contenterons de justifier les quelques particularités qui caractérisent notre programme.

a. Nom du projet

SATO peut paraitre incongru comme nom aux premiers abords. Toutefois, ceci est en fait une petite plaisanterie qui fait référence au thème choisi par les chercheurs japonais pour leurs propres noms de projets : *sugar*, *azucar*, *glucose*, *etc*… En effet, ***satō*** est le terme japonais désignant le sucre, et contient l'acronyme SAT.

b. Diagramme de classe





c. Options disponibles du script

L'option –h du script permet d'afficher l'aide de celui-ci ainsi que des exemples d'utilisation et les autres options disponibles.

```
$ ./sato -h

-- Project SATO using glucose 4.0 based on MiniSAT --
-- SATO Copyright (C) 2016 Justine Evrard & Mariam Bouzid --
For the full copyright and license information, please view the
LICENSE file that was distributed with this source code.

INSTRUCTIONS:
Script's arguments can be :
  - an integer variable's name and its domain like x 0 2
  - constraints on clausal form like ['<constraint>' '<constraint>']
['<constraint>']
    where constraints are composed of two linear expressions separated by
one of these symbol : <=, >=, <, >, !=, =

Arguments' order doesn't matter but if constraints' form isn't respected or
if an integer variable's declaration is missing, the script won't succeed.

USAGE EXAMPLES:
  ~ SATISFIABLE ~
  ./sato x 0 2 y 0 2 ['x-y<=-1' '-x+y<=-1']
  ./sato x 1 3 y -1 0 ['x+y<1'] ['-x+y>=-3']
  ./sato x 2 6 y 2 6 ['x+y<=7']
  ./sato x 0 1 y 0 1 ['x!=y' 'x=y']
  ~ UNSATISFIABLE ~
  ./sato x 1 3 y -1 0 ['x+y<1'] ['-(-x+y>=-3)']
  ./sato x 1 3 y 0 1 ['x+y<1'] ['-x+y>=-3']
  ./sato x 0 2 y 0 2 ['x!=y'] ['x=y']

AVAILABLE OPTIONS:
  -h or --help : display this documentation
  -v : verbose - display solver SAT output
  -i : interprete and display the solver's solutions
  -f <file_name.dimacs> <comment>: only generate the dimacs file with name
'file_name.dimacs' and a comment 'comment'
```

3. PROCESSION

De façon naturelle, nous avons commencé par étudier la syntaxe de fichiers *dimacs*, imposée par l'utilisation d'un *solver SAT*, et donc par créer la classe qui représente une entité SAT. Dans un second temps, nous nous sommes étendues sur l'aspect de la représentation mathématiques des éléments inéquations, variables entières et termes linéaires ; ceci nous amenant dès lors à écrire une ébauche du script principal afin de se rendre compte de la bonne cohésion des composantes.

Par la suite, l'élaboration du concept de CSP s'est imposé à ces dernières puisque les contraintes de celui-ci sont exprimées par un ensemble d'ensembles d'inéquations, *i.e.* forme clausale. Le changement d'une syntaxe – CSP – à une autre – SAT – a été attribué à l'abstraction de l'encoder qui garantit la connexion entre ces deux entités. La dernière phase consistait à modéliser la gestion du *solver SAT* par la détermination de la satisfiabilité du problème et de son interprétation mathématique, rendue possible de par la signification mathématique des prédicats utilisés.

4. OBSTACLES FRANCHIS

Peu de problèmes d'ordre technique et conceptuel se sont manifestés à notre égard. Cependant, nous nous sommes confrontés à deux incidents pouvant perturber la cohérence de nos résultats. Le premier fut une mauvaise interprétation de la notation utilisée pour décrire la relation d'ordre ; ce que nous avons explicité dans la partie théorique. Le second portait sur la traduction en FNC d'une inéquation. En effet, lorsque cette transposition était vide, la satisfiabilité du problème pouvait être incohérente avec le résultat attendu. Si bien qu'il était nécessaire, pour ce cas de figure, de retourner un tableau avec un tableau vide, *i.e.* un ensemble avec une clause vide, au lieu de seulement retourner un tableau vide.

5. EXEMPLES D'EXECUTION

Voici des exemples d'exécution permettant de visualiser l'entrée et la sortie du programme dans un terminal.

a. Exemple satisfiable avec interprétation

Existe-t-il des valeurs pour $x \in [\![0;2]\!]$ *et* $y \in [\![0;2]\!]$ *tels que* $(x = y)$?

```
$ ./sato x 0 2 y 0 2 [‘x=y’] -i

-- Project SATO using glucose 4.0 based on MiniSAT --
-- SATO Copyright (C) 2016 Justine Evrard & Mariam Bouzid --
For the full copyright and license information, please view the
LICENSE file that was distributed with this source code.

CSP {
      [(x+(-y) <= 0); ],
      [-(x+(-y) <= -1); ],
}

(x+(-y) <= 0) {
      [(x <= 2); -(y <= 2); ],
      [(x <= 1); -(y <= 1); ],
      [(x <= 0); -(y <= 0); ],
      [(x <= -1); -(y <= -1); ],
}
-(x+(-y) <= -1) {
      [-(x <= 1); -(x <= 0); -(x <= -1); ],
      [(y <= 2); -(x <= 0); -(x <= -1); ],
      [-(x <= 1); (y <= 1); -(x <= -1); ],
      [(y <= 2); (y <= 1); -(x <= -1); ],
      [-(x <= 1); -(x <= 0); (y <= 0); ],
      [(y <= 2); -(x <= 0); (y <= 0); ],
      [-(x <= 1); (y <= 1); (y <= 0); ],
      [(y <= 2); (y <= 1); (y <= 0); ],
}

SAT {
      [q00; ],
      [-q00; px2; -py2; ],
      [-q00; px1; -py1; ],
      [-q00; px0; -py0; ],
      [-q00; px-1; -py-1; ],
      [q10; ],
      [-q10; -px1; -px0; -px-1; ],
      [-q10; py2; -px0; -px-1; ],
      [-q10; -px1; py1; -px-1; ],
      [-q10; py2; py1; -px-1; ],
      [-q10; -px1; -px0; py0; ],
      [-q10; py2; -px0; py0; ],
      [-q10; -px1; py1; py0; ],
      [-q10; py2; py1; py0; ],
      [-px-1; ],
      [px2; ],
      [-px-1; px0; ],
      [-px0; px1; ],
      [-px1; px2; ],
      [-py-1; ],
      [py2; ],
      [-py-1; py0; ],
      [-py0; py1; ],
```

```
      [-py1; py2; ],
}
Variables : q00 q10 px-1 px2 px0 px1 py-1 py2 py0 py1

--> The SAT problem is satisfiable.

There are 3 solution(s) :
      x = 2       y = 2
      x = 1       y = 1
      x = 0       y = 0
```

b. Exemple insatisfiable

Existe-t-il des valeurs pour $x \in [\![0;2]\!]$ *et* $y \in [\![3;5]\!]$ *tels que* $(x = y)$?

```
$ ./sato x 0 2 y 3 5 ['x=y']

-- Project SATO using glucose 4.0 based on MiniSAT --
-- SATO Copyright (C) 2016 Justine Evrard & Mariam Bouzid --
For the full copyright and license information, please view the
LICENSE file that was distributed with this source code.

CSP {
      [(x+(-y) <= 0); ],
      [-(x+(-y) <= -1); ],
}

(x+(-y) <= 0) {
      [(x <= 2); -(y <= 2); ],
}

-(x+(-y) <= -1) {
      [-(x <= 2); -(x <= 1); ],
      [(y <= 3); -(x <= 1); ],
      [-(x <= 2); (y <= 2); ],
      [(y <= 3); (y <= 2); ],
}

SAT {
      [q00; ],
      [-q00; px2; -py2; ],
      [q10; ],
      [-q10; -px2; -px1; ],
      [-q10; py3; -px1; ],
      [-q10; -px2; py2; ],
      [-q10; py3; py2; ],
      [-px-1; ],
      [px2; ],
      [-px-1; px0; ],
      [-px0; px1; ],
      [-px1; px2; ],
      [-py2; ],
      [py5; ],
      [-py2; py3; ],
      [-py3; py4; ],
      [-py4; py5; ],
}
Variables : q00 q10 px-1 px2 px0 px1 py2 py5 py3 py4

--> The SAT problem is unsatisfiable.
```

c. Exemple infructueux

Existe-t-il des valeurs pour $x \in [\![0;2]\!]$ *et* $y \in [\![0;2]\!]$ *tels que* $(x + y = z)$?

```
$ ./sato x 0 2 y 0 2 ['x+y=z']

-- Project SATO using glucose 4.0 based on MiniSAT --
-- SATO Copyright (C) 2016 Justine Evrard & Mariam Bouzid --
For the full copyright and license information, please view the
LICENSE file that was distributed with this source code.

Inequation class : variable z in expression is not in the array of
variables.
CSP class : cannot parse expression.
```

6. EXTENSION AUX DOMAINES NON CONTINUS

Nous souhaitions approfondir la théorie en travaillant avec des variables pouvant être élevées à une puissance $n \in \mathbb{N}$. La solution aurait été de faire un changement de variable pour chaque variable élevée à une certaine puissance, amenant à lier un nouveau domaine de définition à celle-ci.

PROPOSITION 5 (TRANSFORMATION DE L'EXPONENTIATION)

Soit l'expression (E) *définie par :*

$$(E) : \sum_{i=0}^{n} a_i x_i^k$$

Où $a_i \in \mathbb{Z}^*, x_i \in [\![b_{1i}; b_{2i}]\!] \subset \mathbb{Z}, k \in \mathbb{N}$. *Il est possible d'avoir* $x_i = x_j$, *pour* $i \neq j \in [\![1; n]\!]$.

Soit la transformation suivante sur (E) *:* $\forall k > 1$, *on remplace la variable* x_i^k *par une nouvelle variable* $x_{ik} \in \{ c^k \mid \forall c \in [\![b_{1i}; b_{2i}]\!] \}$. *De cette façon, il se peut que les* x_{ik} *prennent leurs valeurs dans un domaine non continu.*

EXEMPLE

Soit l'équation (e) *suivante :* $2x^3 + 4x^2 - x + 3 = 0$ *où* $x \in [\![0; 4]\!]$. *La transformation de* (e) *est :* $2x_3 + 4x_2 - x + 3 = 0$ *où* $x_3 \in \{0, 1, 8, 27, 64\}$ *et* $x_2 \in \{0, 1, 4, 9, 16\}$.

Ce qui nous a manqué fut la possibilité d'appliquer, dans le temps imparti de notre stage, l'**order encoding** sur des variables dont les domaines sont non continus.

CONCLUSION

En des termes généraux, notre étude concernait la modélisation d'un problème de mathématiques discrètes en un problème de logique informatique. Le passage entre ces deux discours devait ainsi garantir la consistance et la cohérence de la théorie, et préserver l'intégrité de l'objet modélisé. Il fut donc essentiel de poser les termes de notre approche avec l'utilisation de deux types de discours : celui des mathématiques – très succinctement – et celui de l'informatique.

Dans ce cadre théorique, notre dialectique s'est décomposée en deux parties : l'étude morphologique et l'étude sémantique. Ces deux approches ont naturellement convergées lors de la formalisation de l'encodage, celui-ci devant préserver la connexité entre la syntaxe et le sens. La difficulté résidait dans le fait que nous allions utiliser un autre degré d'abstraction – celui de l'informatique – alors que l'objet en lui-même était déjà sous forme abstraite de par son caractère mathématique.

L'implémentation de cette théorie et l'architecture de ce programme s'est effectuée aisément grâce à une approche constructive et logique issue de la première partie. De ce fait, l'écriture de ce code en langage PHP s'est déroulée sans grandes difficultés ; bien que quelques petites incohérences dans les résultats nous aient amené à nous rendre compte d'une méprise à propos de certains détails du sujet, dont nous nous sommes appliquées à en trouver la source.

En définitive, nous avons été très enthousiastes et curieuses devant ce sujet qui nous a toutes deux donné pleinement satisfaction, autant d'un point de vue théorique de par sa nature mathématique et logique, que d'un point de vue pratique au sens de l'application d'une certaine méthodologie dans un langage particulier. Cependant, nous exprimons une petite déception concernant les possibilités d'améliorations qui n'ont pu être abouties, faute de temps.

BIBLIOGRAPHIE

a. Sites internet

- *Complexity Zoo*
- *Sugar web site*
- *The Glucose SAT Solver*
- *MiniSAT Solver*
- *CNF – Conjunctive Normal Form (Dimacs format) Explained*

b. Publications et mémoires

- *Résolution de problèmes combinatoires et optimisation par colonies de fourmis - Christine SOLNON*
- *Méthode sat et algorithme dpll appliqués à un problème de recherche opérationnelle*
- *Fast Algorithms for Generating Integer Partitions*

c. Cours

- *Logique d'ordre 0 et 1* - Igor STEPHAN
- *Démonstration automatique* - Igor STEPHAN
- *Théorème de Cook - Denis Trystram*
- *Calcul propositionnel - Le problème SAT*
- *Le problème SAT et ses variantes*
- *Informatique Fondamentale : Les Solveurs SAT - Emmanuel Filiot*
- *Logique et principe de résolution - D. Pastre*

d. Livres

- *Intelligence artificielle et informatique théorique*, Jean-Marc ALLIOT et Thomas SCHIEX