TouIST again... Formalisez et résolvez facilement des problèmes avec des solveurs SAT, SMT et QBF

Olivier Gasquet, Andreas Herzig, Dominique Longin, Frédéric Maris, Maël Valais

TouIST again...

(formalisez et résolvez facilement des problèmes avec des solveurs SAT, SMT et QBF)

Olivier Gasquet[†], Andreas Herzig[‡], Dominique Longin[‡], Frédéric Maris[†], Maël Valais[†]

IRIT (Institut de Recherche en Informatique de Toulouse)

[‡]CNRS, [†]Université Paul Sabatier, Toulouse, France

{Olivier.Gasquet,Andreas.Herzig,Dominique.Longin,Frederic.Maris,Mael.Valais}@irit.fr

1 Introduction

Depuis 2010, nous développons TouIST [1], un logiciel dédié à la logique propositionnelle dont les principales fonctionnalités sont (1) d'offrir un langage logique de haut niveau pour exprimer succinctement des formules complexes et (2) de trouver des modèles à ces formules en utilisant un solveur SAT performant.

Dans ce qui suit, nous présentons une extension de TouIST à QBF (*Quantified Boolean Formulas*) au travers d'un exemple : le jeu de Nim.

Tout d'abord, nous survolons succinctement les principales caractéristiques de TouIST (Section 2) et montrons comment modéliser le jeu de Nim dans le langage d'entrée de TouIST (Section 3). Enfin, après une brève présentation de QBF nous montrons comment modéliser la recherche d'une stratégie gagnante dans TouIST pour ce jeu (Section 4).

2 Présentation générale de TouIST

Dès le départ, nos exigences étaient que l'outil devait : être très facile à installer et à utiliser, sans syntaxe complexe ; pouvoir être utilisé comme une boîte noire sans connaître son fonctionnement interne (éditeur intégré de formules traduites automatiquement et de manière transparente dans le langage d'entrée du solveur considéré) ; ne requérir aucune mise en forme normale, ordonnancement de clauses, *etc.* ; ne nécessiter qu'un minimum de connaissances en logique.

Avec TouIST, les étudiants peuvent expérimenter par eux-même qu'un langage logique n'est pas seulement descriptif mais peut aussi conduire à des calculs qui résolvent des problèmes concrets. En particulier, TouIST leur permet de résoudre des Sudokus assez facilement, ainsi que beaucoup d'autres problèmes combinatoires (emplois du temps, coloration de carte, circuits électroniques, *etc.*).

Voici les principales facilités qu'offre TouIST :

- les formules entrées n'ont pas besoin d'être sous forme clausale et des connecteurs arbitraires peuvent être utilisés, la mise sous forme normale est faite dynamiquement pendant la saisie au clavier de l'utilisateur ;
- facilités d'utilisation de conjonctions ou disjonctions indicées comme :

$$\bigwedge_{i\in\{1..9\}} \bigvee_{j\in\{1..9\}} \bigwedge_{n\in\{1..9\}} \bigwedge_{m\in\{1..9\},m\neq n} (p_{i,j,n} \rightarrow \neg p_{i,j,m})$$

  exprimant qu'une case de coordonnées (i, j) contient au plus un nombre compris entre 1 et 9.
- plusieurs solveurs sont disponibles (SAT, QF_LRA, QF_LIA, QF_RDL, QF_IDL et QBF) et le langage admissible par TouIST s'adapte facilement ;
- définition d'ensembles de domaines : $\bigwedge_{i\in A}$ vs. $\bigwedge_{i\in\{Paris,London,Roma,Madrid\}}$
- liaisons multiples sur les indices : $\bigwedge_{i\in A,j\in B}$ vs. $\bigwedge_{i\in\cdots}\bigwedge_{j\in\cdots}$
- calculs riches sur les indices ainsi que sur les ensembles de domaines : $\bigwedge_{i\in(A\cup(B\cap C))}$
- primitives de contraintes de cardinalité : *au moins*, *au plus*, *exactement*, *n valeurs sont vraies parmi un ensemble de valeurs données*, *etc.*
- prédicats pouvant être des variables définies sur des ensembles de domaines : $\bigwedge_{X\in\{A,B\},i\in\{1,2\}} X(i)$ *vs.* $\bigwedge_{i\in\{1,2\}}(A(i)\wedge B(i))$

1. Historiquement, TouIST est le successeur de SATouLOUSE, présenté pour la première fois lors de la conférence ICTTL'2011 [2].

— littéraux spéciaux définissant des contraintes entre nombres entiers ou réels : $(x + y \leq z)$
— parcours facile des modèles successivement calculés par les solveurs
— expressions régulières permettant un filtrage des littéraux pertinents
— possibilité d'utiliser le logiciel en ligne de commande et/ou batch
— nombreuses fonctionnalités d'édition et améliorations

Ainsi, il est possible de montrer la puissance de la logique propositionnelle à des étudiants qui ont été formés quelques heures à la formalisation de phrases en logique et qui ont acquis les notions de bases de validité et satisfiabilité pour résoudre automatiquement des Sudokus par exemple.

Une présentation plus complète de TouIST peut être trouvée dans [4]. TouIST est téléchargeable à l'adresse https://www.irit.fr/touist. Cette page donne aussi accès au manuel complet du langage et de l'utilisation du logiciel.

3 Description du jeu de Nim

Le principe du jeu de Nim est le suivant : on dispose au départ d'un nombre NA non nul d'allumettes et un nombre NJ de joueurs peuvent prendre 1 ou plusieurs allumette(s). Le joueur qui perd est celui qui, le premier, ne peut plus prendre d'allumette. [2] Le nombre de tours de jeu possibles est au plus égal à celui des allumettes (à minima, chaque joueur ne prend à chaque tour qu'une seule allumette). Ainsi, l'ensemble des indices des tours possibles est $T = \{0, ..., NA\}$ où 0 est l'indice de l'état initial. De même, l'ensemble des nombres possibles d'allumettes encore disponibles est $A = \{0, 1, ..., NA\}$.

Afin de simplifier au maximum le langage utilisé, nous modélisons ici une variante où $NA = 4$ et $NJ = 2$. Les joueurs sont notés 0 et 1 et c'est au tour de 0 de jouer au tour t ssi $tour\_de\_0(t)$ est vrai (considérant que si ce n'est pas le tour de 0 alors c'est celui de 1). De plus, $reste(t, n)$ est vrai ssi au tour t il reste n allumettes.

Ainsi, l'état initial du jeu est le suivant :

$$reste(0, NA) \wedge tour\_de\_0(0) \tag{1}$$

indiquant qu'au tour 0 il y a encore NA de disponible et que c'est au tour de 0 de jouer.

2. Il existe différentes variantes de ce jeu, notamment en faisant varier les nombres d'allumettes et de joueurs, mais également en faisant varier les actions possibles ou en introduisant des contraintes (par exemple, on ne peut pas prendre le même nombre d'allumettes que le joueur précédent).

Dans la version présentée ici du jeu de Nim, on limite également le nombre des actions possibles à deux : un joueur peut prendre soit 1 allumette, soit 2 allumettes. Ainsi, $prend(t, 2)$ est vrai ssi un agent prend 2 allumettes au tour t (considérant ainsi que $prend(t, 2)$ est faux ssi il n'en prend qu'une).

Ainsi :

$$\bigwedge_{\substack{t\in T\\ n\in A\\ n\geq 2}} \Big(\Big(reste(t,n) \wedge prend(t,2) \rightarrow reste(t+1, n-2) \Big) \wedge \Big(reste(t,n) \wedge \neg prend(t,2) \rightarrow reste(t+1, n-1) \Big) \Big) \tag{2}$$

capture le fait que si au tour t il reste au moins 2 allumettes et qu'un joueur en prend 2 alors au tour suivant il en reste 2 de moins, et que si il n'en prend qu'une alors au tour suivant il en reste 1 de moins.

En revanche, si au tour t il reste exactement 1 allumette, alors nécessairement le joueur en prendra 1 et il en restera 0 au tour suivant :

$$\bigwedge_{t\in T} \Big(reste(t,1) \rightarrow \neg prend(t,2) \wedge reste(t+1, 0) \Big) \tag{3}$$

Notre modèle spécifie ensuite que :

$$\bigwedge_{t\in T} \bigvee_{n\in A} reste(t,n) \tag{4}$$

$$\bigwedge_{\substack{t\in T\\ n1,n2\in A\\ n1\neq n2}} \Big(reste(t,n1) \rightarrow \neg reste(t,n2) \Big) \tag{5}$$

La première formule stipule qu'à chaque tour t il existe au moins un nombre n d'allumettes restant, et la seconde que ce nombre est unique.

Il faut maintenant définir quand un joueur a perdu :

$$0\_perd \leftrightarrow \bigvee_{\substack{t\in T\\ t>0}} \Big(tour\_de\_0(t) \wedge reste(t,0) \Big) \tag{6}$$

signifie que le jouer 0 a perdu ssi il existe un tour t où il reste 0 allumettes alors qu'à l'instant d'avant il y en avait au moins une.

Finalement, à chaque tour t, ce n'est pas au joueur 0 de jouer ssi c'est à lui de jouer au tour suivant :

$$\bigwedge_{t\in T\setminus\{NA\}} \Big(\neg tour\_de\_0(t) \leftrightarrow tour\_de\_0(t+1) \Big) \tag{7}$$

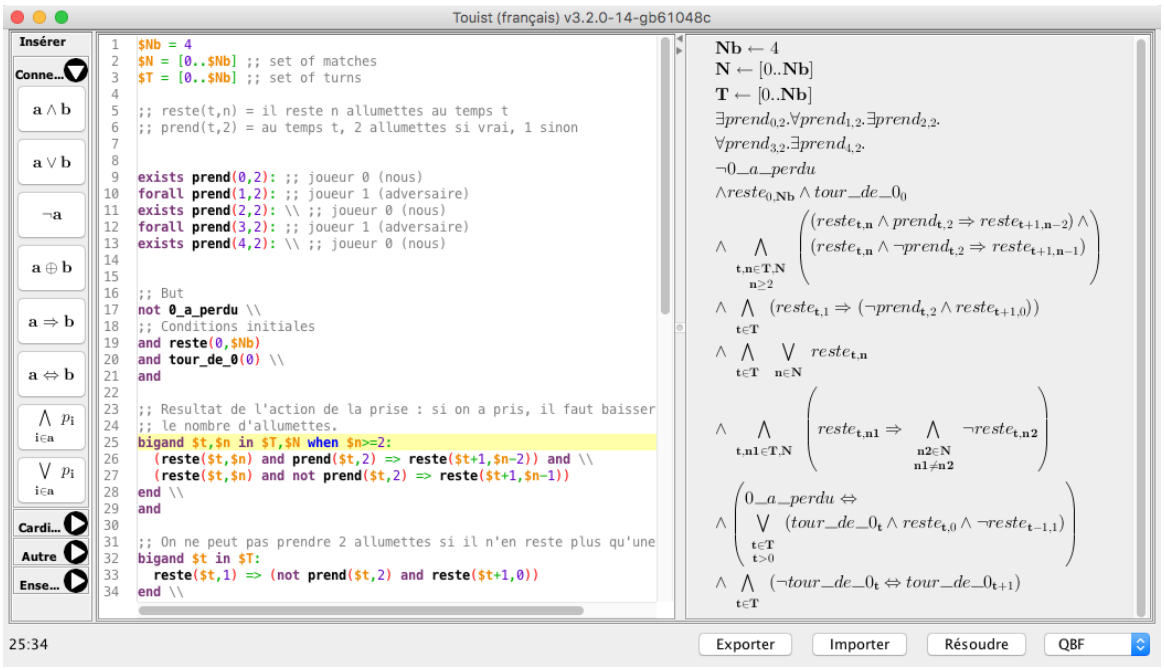

FIGURE 1 – Capture d'écran de TOUIST avec le jeu de Nim. Le fichier est disponible à l'adresse https://github.com/maelvalais/allumettes

4 Formalisation d'une stratégie gagnante à l'aide de QBF

Dans cette section, nous allons présenter à l'aide de notre exemple du jeu de Nim l'extension de TOUIST à Quantified Boolean Formulas (QBF) connu comme étant le problème de référence pour la classe de complexité PSPACE ([5]). C'est une extension de la logique propositionnelle permettant de quantifier sur les variables propositionnelles.

Par exemple, $\forall p \exists q.p \leftrightarrow q$ se lit : pour toute valeur de vérité de p, il existe une valeur de vérité de q tel que $p \leftrightarrow q$ est vrai. Cette formule est vraie (il suffit de choisir la même valeur pour q que pour p). Alors que $\exists p \forall q.p \vee q$ ne l'est pas. Ainsi, une formule booléenne quantifiée est toujours SOIT vraie SOIT fausse.

De fait, à toute formule QBF peut être associée une formule propositionnelle sans variables car par définition : $\forall p.\Phi$ est vraie ssi $\Phi_{[p:=\top]} \wedge \Phi_{[p:=\bot]}$ l'est, et $\exists p.\Phi$ est vraie ssi $\Phi_{[p:=\top]} \vee \Phi_{[p:=\bot]}$.

La formule QBF peut être exponentiellement plus compacte que la formule propositionnelle correspondante.

Par exemple à la formule $\forall p \exists q.p \leftrightarrow q$ correspond la formule propositionnelle

$$\Big((\top \leftrightarrow \top) \vee (\top \leftrightarrow \bot)\Big) \wedge \Big((\bot \leftrightarrow \top) \vee (\bot \leftrightarrow \bot)\Big)$$

Le langage de QBF permet d'exprimer naturellement et de manière concise l'existence de stratégies gagnantes ainsi que décrit dans [3]. Les coups du joueur 0 (pour lequel on cherche une stratégie gagnante) seront existentiellement quantifiés alors que ceux de son adversaire seront universellement quantifiés. (On cherche les coups du joueur 0 qui le mèneront à la victoire quels que soient les coups joués par le joueur 1.)

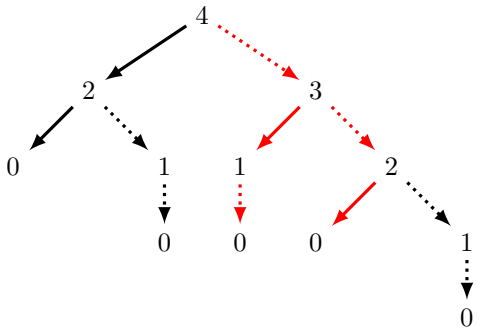

FIGURE 2 – Solutions pour le jeu de Nim (4 all./2 joueurs), en rouge : stratégie gagnante du joueur 0

TOUIST a été étendu pour être compatible avec le solveur QBF *Quantor 3.2* [1]. La sélection de ce prouveur dans TOUIST autorise *de facto* l'utilisation des quantificateurs $\forall$ et $\exists$ (respectivement définis par `exists` et `forall` dans TOUIST).

FIGURE 2 présente l'ensemble exhaustif des solutions de notre exemple. La racine de l'arbre représente le nombre initial d'allumettes, et chaque flèche l'action de retirer 1 (.....▸) ou 2 (——▸) allumette(s). Au

bout de la flèche, le nombre d'allumettes après exécution de l'action concernée. D'après cette figure, on voit que si le joueur 0 commence (ce qui est imposé par (1)) et qu'il retire une seule allumette (il en reste donc 3) on voit qu'il a une stratégie gagnante :

— si le joueur 1 retire ensuite 2 allumettes il en restera 1 seule que le joueur 0 peut retirer pour gagner (puisque le joueur 1 ne pourra ensuite plus retirer d'allumette) ;
— si le joueur 1 retire une seule allumette, il en restera 2 et le joueur 0 pourra au coup suivant les retirer en un seul coup et le joueur 1 perd.

Nous tirons parti de QBF pour écrire cette stratégie dans TouIST. Si on note Φ la conjonction des formules (1) à (7) alors la recherche d'une stratégie gagnante pour le joueur 0 s'écrit simplement :

$$\begin{array}{l} \exists prend(0,2) \forall prend(1,2) \\ \quad \exists prend(2,2) \forall prend(3,2) \\ \exists prend(4,2) \ . \ \neg 0\_perd \wedge \Phi \end{array} \quad (8)$$

Autrement dit, on cherche à satisfaire le fait qu'il existe une action du joueur 0 au tour 0 telle que quelle que soit l'action du joueur 1 au tour 1, il existe une action du joueur 0 au tour 2, telle que pour toute action du joueur 1 au tour 3 il existe une action du joueur 0 (qui sera donc le dernier à jouer) telle que le joueur 0 ne perd pas et que les contraintes inhérentes au jeu de Nim soient satisfaites.

L'exécution du programme dans TouIST indique que cette formule est vraie, ce qui signifie l'existence d'une stratégie gagnante pour le joueur 0. Le solveur retourne la valeur des (ici une seule) variables existentielles correspondant au prochain coup du joueur 0. À ce stade, le joueur adverse doit fournir son coup qui fixe la valeur des variables universelles correspondant à ses possibles prochains coups. On exécute alors de nouveau le programme modifié de la façon suivante (de manière à prendre en compte le calcul de la valuation de $prend(0,2)$) :

$$\begin{array}{l} \exists prend(0,2) \exists prend(1,2) \\ \quad \exists prend(2,2) \forall prend(3,2) \\ \exists prend(4,2) \ . \ \neg 0\_perd \wedge c_0 \wedge c_1 \wedge \Phi \end{array}$$

où c_0 est soit $prend(0,2)$ soit $\neg prend(0,2)$ en fonction du coup du joueur 0, et similairement pour c_1 en fonction du coup choisi par l'adversaire. La situation après ces deux coups est la nouvelle situation initiale pour le solveur et la recherche du coup suivant du joueur 0... jusqu'à sa victoire ! On réitère ce processus jusqu'à ce que toutes les variables aient reçu une valeur.

5 Conclusion

TouIST peut être vu comme un compilateur de langages logiques étendus et de haut niveau vers des prouveurs efficaces indépendants. Ces deux facettes lui confèrent une grande facilité d'utilisation, un large spectre d'application et de bonnes performances calculatoires. À ce titre, il constitue un outil complètement original et unique en son genre.

Nous l'utilisons dans le cadre du cours d'initiation à la logique en licence d'informatique, mais aussi en master, dans le cadre de travaux pratiques et de projets. Les étudiants sont ainsi appelés à parcourir tout le processus allant de la formalisation à la résolution de problèmes qui vont au-delà des problèmes-jouets faisables sur papier.

Mais plus encore, TouIST est d'ores et déjà utilisé par des chercheurs dans le cadre de travaux mené au sein de notre laboratoire et impliquant une modélisation logique (planification, raisonnement épistémique via traduction en QBF,...), il comble un manque existant au sein des logiciels de calculs formels comme Maple, SageMath, Mathematica ou Maxima qui n'intègrent qu'anecdotiquement des outils logiques.

Références

[1] Armin Biere. Resolve and expand. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, SAT'04, pages 59–70, Berlin, Heidelberg, 2005. Springer-Verlag.

[2] Olivier Gasquet, François Schwarzentruber, and Martin Strecker. Satoulouse : The computational power of propositional logic shown to beginners. In Patrick Blackburn, Hans van Ditmarsch, María Manzano, and Fernando Soler-Toscano, editors, *Tools for Teaching Logic : Third International Congress, TICTTL 2011, Salamanca, Spain, June 1-4, 2011. Proceedings*, pages 77–84. Springer, Berlin, Heidelberg, 2011.

[3] Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[4] Khaled Skander Ben Slimane, Alexis Comte, Olivier Gasquet, Abdelwahab Heba, Olivier Lezaud, Frederic Maris, and Mael Valais. Twist your logic with touist. *CoRR*, abs/1507.03663, 2015.

[5] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 1–9, New York, NY, USA, 1973. ACM.