La logique facile avec TouIST

(formalisez et résolvez facilement des problèmes du monde réel)

Skander Ben Slimane[2], Alexis Comte[2], Olivier Gasquet[1 2], Abdelwahab Heba[2], Olivier Lezaud[2], Frédéric Maris[1 2], Maël Valais[2]

[1]IRIT (Institut de Recherche en Informatique de Toulouse)
[2]Université Paul Sabatier
Toulouse, France
{gasquet,maris@irit.fr}

Résumé

Les solveurs SAT sont des outils puissants pour résoudre des problèmes logiques de taille réelle, mais leur utilisation nécessite des connaissances solides. Elle peut être vue par rapport à la logique comme l'utilisation d'un langage d'assemblage par rapport à la programmation. Il manque un langage de haut niveau pour permettre à des utilisateurs divers de tirer facilement profit de ces outils. TouIST vise à combler cette lacune. Il est dédié à la logique propositionnelle et ses principales fonctions sont (1) d'offrir un langage logique de haut niveau pour exprimer succictement des formules complexes (par exemple des formules décrivant les règles du Sudoku, des problèmes de planification...) et (2) de trouver des modèles à ces formules en utilisant un solveur adéquat et performant, que l'utilisateur n'a pas besoin de connaître. Il consiste en une interface conviviale qui propose plusieurs facilités syntaxiques et qui fait appel à des solveurs suffisamment puissants pour permettre de résoudre automatiquement de grandes instances de problèmes difficiles (emplois du temps, Sudokus...). Il peut interagir avec différents démonstrateurs : solveurs SAT pur mais également solveurs SMT (SAT modulo théories - comme la théorie linéaire sur les réels, etc). Il peut donc être utilisé aussi bien par des débutants pour des problèmes purement propositionnels, que par des étudiants de cycles supérieurs ou même des chercheurs et ingénieurs, par exemple pour résoudre des problèmes de planification impliquant de grands ensembles d'actions et des contraintes numériques.

1 Historique

O. Gasquet et F. Maris enseignent à l'Université Paul Sabatier à Toulouse. Ils enseignent la logique à différents niveaux, en démarrant des cours d'introduction à la logique propositionnelle, jusqu'à des sujets avancés pour les étudiants en Master, comme la logique modale ou la planification basée sur la logique. S. Ben Slimane, A. Comte, A. Heba, O. Lezaud et M. Valais sont des étudiants en Master de la même université. Ils ont mis en oeuvre TouIST durant les trois mois de leur projet de Master.

Motivation des étudiants

Au début des études de premier cycle, nous (enseignants) avons constaté que la motivation des étudiants peut être améliorée en leur montrant que la logique est très utile pour les informaticiens et que l'informatique ne consite pas seulement à écrire du code C ou Java. Classiquement, la logique est motivée par des exemples abstraits ou, au mieux, par des exemples ludiques. A un moment, nous avons pensé qu'il serait préférable de leur montrer et pas seulement de leur dire qu'avec un peu de connaissance, la logique peut être utilisée pour résoudre des problèmes difficiles que la taille empêche de résoudre facilement à la main ou exigerait une programmation assez complexe en C ou tout autre langage de programmation.

Genèse de SAToulouse

Lors de la conférence ICTTL'2011, nous avions présenté SAToulouse [2], dédié à la logique propositionnelle, dont les principales fonctions étaient (1) d'offrir un langage logique de haut niveau pour exprimer

succictement des formules complexes et (2) de trouver des modèles à ces formules en utilisant un solveur SAT performant.

Bien sûr, il existe de nombreux outils logiques comme des prouveurs, assistants de preuves, éditeurs de tables de vérité... sur Internet, même PROLOG aurait pu être utilisé, mais aucun ne correspond à nos exigences qui sont :

— l'outil doit être très facile à installer et à utiliser, sans syntaxe complexe ;
— le prouveur peut être utilisé comme une boîte noire sans savoir comment il fonctionne ;
— aucune mise en forme normale, aucun ordonnancement de clauses, ou aucune coupure PROLOG ne doivent être requis ;
— seulement une petite connaissance en logique devrait être nécessaire.

Comme nous ne pouvions pas trouver un outil existant satisfaisant ces exigences, en 2010, nous avons commencé à développer le nôtre, et nous sommes arrivés à l'idée de développer simplement une interface qui permet d'utiliser très confortablement un prouveur SAT (à savoir SAT4J [1]) : cet outil avait été applelé SATouLouse et est décrit dans [2]. Avec cet outil, les étudiants pouvaient expérimenter par eux-même qu'un langage logique n'est pas seulement descriptif mais peut conduire à des calculs qui résolvent des problèmes de la vie réelle. En particulier, avec SATouLouse, ils pouvaient résoudre des Sudokus assez facilement, ainsi que beaucoup d'autres problèmes combinatoires (emplois du temps, coloration de carte, circuits électroniques...).

Voici les principales facilités qu'offraient SATouLouse :

— 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 ;
— des facilités d'utilisation de grandes conjonctions ou disjonctions sont offertes comme dans :

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

— démarrer le solveur consiste à cliquer sur un bouton ;
— l'outil affiche un modèle dans la syntaxe de la formule entrée.

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.

Travaux pratiques avec SATouLouse

Mais ce n'est pas toute l'histoire, puisque le même solveur SAT peut être utilisé pour résoudre de nombreux autres problèmes combinatoires aussi facilement que pour le Sudoku : ils suffit juste de formaliser les contraintes. Nos étudiants sont invités à le faire pour : des emplois du temps, des colorations de cartes,... SATouLouse a été utilisé pendant trois ans par environ 400 étudiants avec une grande satisfaction. En particulier, les étudiants l'ont utilisé pour effectuer des devoirs à long terme dans l'esprit de la programmation de projets : nous leur donnons un problème logique à résoudre (trop gros pour être résolu à la main), ils doivent le formaliser et ensuite utiliser cette formalisation pour le résoudre. Par exemple, un problème de stockage de produits chimiques qui doivent être stockés dans des salles identiques/contigües/non-contigües en fonction de leur degré de compatibilité. Les étudiants doivent résoudre un cas impliquant beaucoup de produits chimiques.

Limites de SATouLouse et genèse de TouIST

Mais pendant ces années, nous avons remarqué quelques limitations dommageables de SATouLouse : de nombreux bugs, des défauts dans l'interface, le manque de modularité (si l'on souhaite changer le prouveur SAT utilisé), l'ambiguïté et les limites de son langage, etc.

Par exemple, des problèmes impliquant des contraintes de cardinalité, comme les règles du jeu de Takuzu [1] qui nécessitent de compter des 0 et des 1, ne peuvent être facilement formalisés : il manque des fonctionnalités permettant d'exprimer des choses comme "exactement 5 parmi 10 propositions sont vraies".

De plus, SATouLouse n'offre pas la possibilité de parcourir l'ensemble des modèles fournis par le solveur, il en retourne seulement un.

Les leçons tirées de trois années d'utilisation de SATouLouse sont que beaucoup de nos étudiants en informatique ont clairement pris conscience que la logique avait des applications réelles en ce qui concerne la résolution de problèmes, et beaucoup d'entre-eux ont acquis une capacité dans la formalisation de problèmes. Mais les défauts de SATouLouse rendent le débogage vraiment difficile, d'une part parce qu'un seul modèle est affiché et en raison de la façon dont ce modèle est affiché, et d'autre part à cause des faibles capacités d'édition dont il dispose. En outre, seuls des problèmes combinatoires purs peuvent être traités, ce qui limite lourdement la prétention de résolution d'une

1. Connu aussi sous le nom de Binero. http://fr.wikipedia.org/wiki/Takuzu

large gamme de problèmes par SATouLouse concernant les problèmes du monde réel.

Un autre inconvénient de SATouLouse, pas nécessairement lié à l'enseignement de la logique, est son incapacité à être utilisé à partir de la ligne de commande : des chercheurs ou des ingénieurs qui souhaiteraient l'utiliser intensivement trouveraient fastidieux de taper des problèmes en entrée.

Enfin, l'extension à des théories plus riches est également quelque-chose qui peut intéresser les chercheurs, les ingénieurs ou les étudiants de cycles supérieurs. SATouLouse n'est certainement pas adapté pour la satisfiabilité modulo théories ou pour résoudre des problèmes de planification alors que la même architecture logicielle pourrait être utilisée en changeant juste le solveur.

Il y a quelques mois, nous avons commencé à développer un nouveau logiciel qui comblerait ces lacunes et remplirait nos attentes. Nous l'avons appelé TouIST qui signifie TOUlouse Integrated Satisfiability Tool et doit être prononcé "twist". TouIST est bien sûr à la disposition du public pour téléchargement à partir du site suivant :

`https://github.com/olzd/touist/releases`

Pour résumer, voici quelques fonctionnalités offertes par TouIST et que SATouLouse ne propose pas :

- 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. $\bigwedge_{i \in \ldots} \bigwedge_{j \in \ldots}$
- 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 intégrées : "auMoins" (resp. "auPlus", "exactement") *tant* de valeurs sont vraies parmi *ces valeurs*
- les prédicats peuvent également être des variables définies sur des ensembles de domaines : $\bigwedge_{X \in \{A,B\}, i \in \{1,2\}} X(i)$ vs. $\bigwedge_{i \in \{1,2\}} (A(i) \wedge B(i))$
- 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

2 Vue d'ensemble de TouIST

TouIST est composé de trois modules, mais l'utilisateur standard ne verra que l'un d'entre eux : l'interface. Dans la suite nous insistons principalement sur cette dernière plutôt que sur le traducteur et le solveur. L'architecture globale est illustrée par la figure 1 :

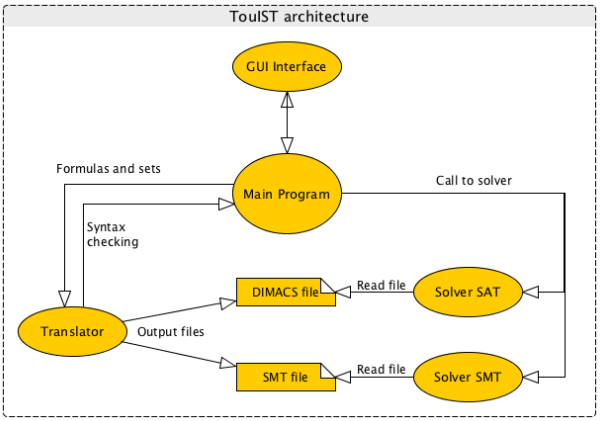

FIGURE 1 – Architecture de TouIST

Avec TouIST on accède à un éditeur puissant et convivial pour éditer des formules logiques complexes et des contraintes variées comme :

$$\bigwedge_{i \in \{1..9\}} (P_i \longrightarrow Q_{i+1})$$

qui abrège confortablement :

$$(P_1 \longrightarrow Q_2) \wedge (P_2 \longrightarrow Q_3) \wedge \ldots (P_9 \longrightarrow Q_{10}).$$

Une fois qu'un ensemble de formules a été donné à l'interface, sa satisfiabilité peut être vérifiée : l'interface peut l'envoyer au prouveur qui retourne un modèle, affiché comme le montre la figure 2 si un tel modèle existe. Ensuite par l'intermédiaire de l'interface, l'utilisateur peut par exemple demander d'autres modèles (bouton "Next" de l'interface). Contrairement à SATouLouse qui aurait nécessité de modifier les formules pour interdire le modèle et de relancer le solveur, TouIST conserve une instance du solveur en attente, ce qui permet d'obtenir les modèles suivants bien plus rapidement.

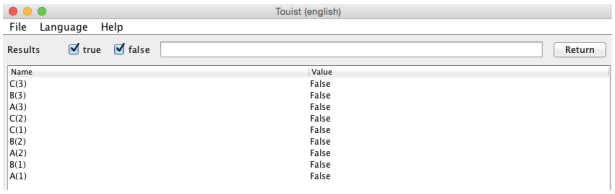

FIGURE 2 – Affichage de modèle

Les modèles renvoyés par le solveur sont totaux : une valeur est affectée à chacune des variables appa-

raissant dans les formules envoyées au solveur. L'utilisateur peut sélectionner uniquement les propositions vraies ou les propositions fausses. Il peut également sélectionner des sous-ensembles de variables en tapant une expression régulière pour les filtrer.

3 Détail de ce qui peut être fait avec TouIST

3.1 Ensembles de domaines

Avec le temps, nous avons remarqué que nous avions souvent besoin d'écrire des choses comme :

$$\bigwedge_{i\in\{1..9\}} \bigwedge_{j\in\{1..9\}} \bigwedge_{m\in\{A,B,C,D,E,F,G,H,I\}} \left(P_{i,j,m} \longrightarrow \bigwedge_{n\in\{A,B,C,D,E,F,G,H,I\}|m\neq n} \neg P_{i,j,n} \right)$$

Si on lit $P_{i,j,m}$ comme "il y a la lettre m dans la case (i, j)" d'une grille 9×9, la formule ci-dessus exprime qu'il y a *au plus* une lettre parmi 'A' ... 'I' dans chaque case.

Ces ensembles $\{1..9\}$ et $\{A, B, C, D, E, F, G, H, I\}$ sont des *ensembles de domaines*, et avec TouIST l'utilisateur peut définir autant d'ensembles de domaines qu'il le souhaite, par exemple :

```
$N=[1..9]
$L=[A,B,C,D,E,F,G,H,I]
```

et ainsi écrire la formule précédente comme :

$$\bigwedge_{i\in N} \bigwedge_{j\in N} \bigwedge_{m\in L} P_{i,j,m} \longrightarrow \bigwedge_{n\in L|m\neq n} \neg P_{i,j,n}$$

De plus, les opérations usuelles sur les ensembles ($\cup$, $\cap$, $\setminus$, ...) peuvent être utilisées pour définir d'autres ensembles.

3.2 Formules propositionelles

Les formules de TouIST sont basées sur des variables propositionnelles (qui peuvent être indicées) et les opérateurs logiques usuels ($\wedge$, $\vee$, $\longrightarrow$, $\neg$, $\leftrightarrow$). Ainsi on peut taper des formules usuelles simples comme $Pluie \longrightarrow Nuages$. Mais en plus, nous fournissons des opérateurs logiques de haut niveau qui permettent d'exprimer des assertions complexes dans une forme très compacte.

Conjonctions et disjonctions généralisées

Elles permettent d'exprimer des conjonctions et des disjonctions sur des formules contenant des paramètres qui varient, par exemple :

— $\bigwedge_{i\in N} P_i$, où N est l'ensemble de domaine défini précédemment.
Elle représente la conjonction $P_1 \wedge P_2 \wedge \ldots \wedge P_9$.
— $\bigvee_{i\in E} P_i$.

Bien sûr, ces opérateurs peuvent être imbriqués comme dans :

$$\bigwedge_{i\in N} \bigwedge_{j\in N} \bigvee_{m\in L} P_{i,j,m}$$

qui indique que dans chaque cellule se trouve au moins une lettre.

Contraintes de cardinalité

C'était l'un des sujets "laissé pour le futur" de [2]. Ces opérateurs logiques moins classiques sont disponibles dans TouIST : il permettent de réduire drastiquement la taille de certaines formules, ce sont : $\leqslant$, $\geqslant$ et $\#$.
L'exemple suivant décrit leur signification :

— $\leqslant^2_{i\in N} P_i$ représente "pour au plus deux valeurs de $i \in N$ $P(i)$ est vraie" ;
— $\geqslant^2_{i\in N} P_i$ représente "pour au moins deux valeurs de $i \in N$ $P(i)$ est vraie" ;
— $\#^2_{i\in N} P_i$ représente "pour exactement deux valeurs de $i \in N$ $P(i)$ est vraie".

La disjonction généralisée est en fait un cas particulier de ceci (au moins une est vraie), la conjonction aussi (au plus 0 est fausse), et le ou exclusif peut être vu comme exactement une parmi deux est vraie.
Rappelons qu'avec des opérateurs logiques classiques et avec N contenant 9 éléments, $\leqslant^3_{i\in N} P_i$ devrait nécessiter une formule contenant 84 propositions P_i puisque cela revient à choisir 3 parmi 9 ce qui donne $\binom{9}{3}$ possibilités, et ni $\bigwedge$ ni $\bigvee$ n'aideraient beaucoup.

Contraintes et calculs sur des indices

Souvent nous avons besoin d'ajouter des contraintes sur les indices, par exemple :

$$\bigwedge_{i\in E} \bigwedge_{j\in E|i\neq j} P_{i,j}$$

qui signifie que $P_{i,j}$ est vraie lorsque $i \neq j$.

C'était la seule contrainte disponible dans SATouLOUSE, maintenant dans TouIST la gamme de possibilités à été largement enrichie. Les contraintes peuvent inclure des opérateurs usuels de comparaison comme $<$, $>$, $\leq$, $\geq$, $\neq$, $=$ et ces comparaisons peuvent s'appliquer non seulement aux indices, mais aussi à toute expression arithmétique impliquant des indices et $+$, $-$, $*$, $/$, $\mod$, $\sqrt{\ }$.

Exprimer une phrase comme "chaque case (i, j) contient un nombre qui n'est pas égal à $i + j$" donnera :

$$\bigwedge_{i \in N} \bigwedge_{j \in N} \bigvee_{k \in N | k \neq i+j} P_{i,j,k}$$

Bien sûr, *toutes ces phrases* pourraient être exprimées avec les opérateurs logiques usuels bruts, mais ceci serait un travail fastidieux. Néanmoins, les étudiants doivent savoir ce qui est derrière la scène, et qu'une telle formule est l'abréviation de quelque chose comme :

$$P_{1,1,1} \vee P_{1,1,3} \vee P_{1,1,4} \ldots P_{1,2,1} \vee P_{1,2,2} \vee P_{1,2,4} \vee \ldots$$

qui est très long et terne.

3.3 Aspects techniques

Langage d'entrée vs Langage d'affichage

Les formules que nous avons vues précédemment sont écrites dans le *langage d'affichage* (style LaTeX), mais tous ces symboles ne sont pas disponibles sur les claviers, ainsi pour écrire les formules et ensembles de domaines, l'utilisateur utilisera le *langage d'entrée*. Par exemple, la formule précédente avec l'ensemble associé N sera tapée (les variables sont préfixées par \$) :

```
bigand $i,$j in $N,$N:
  bigor $k in $N when $k != $i+$j:
    P($i,$j,$k)
  end
end
```

Mais TouIST les affiche immédiatement en style LaTeX comme on peut le voir dans le panneau droit montré sur la figure 3. La définition de l'ensemble N est faite dans l'onglet "Ensembles".

En outre, les formules peuvent être tapées à la main dans la fenêtre d'édition, ou introduites dans une sorte d'éditeur dirigé par la syntaxe, en raffinant progressivement l'arbre syntaxique, ou encore elles peuvent être importées à partir d'un fichier externe.

4 Sujets avancés pour les étudiants de cycles supérieurs

Dans ce qui suit, nous présentons très succictement quelques fonctionnalités avancées de TouIST. Elles devraient intéresser les étudiants de cycles supérieurs et leurs enseignants bien sûr. Elles concernent SMT (SAT Modulo Theories), la planification par satisfaction de base de clauses (planification SAT) et leur combinaison la planification SMT.

4.1 SMT : SAT Modulo Theories

Certains problèmes combinatoires nécessitent néanmoins de traiter des calculs sur les nombre naturels ou réels. Ceci peut être fait en utilisant seulement la logique propositionnelle (par exemple, $2+3 = 5$ pourrait être codé par $add_{2,3,5}$), mais c'est très inconfortable à moins qu'il n'y ait que quelques additions à faire. Ne parlons même pas des opérations de multiplication ou plus complexes. L'idée derrière la genèse de SMT a été de combiner des solveurs SAT avec un solveur arithmétique dans le but d'améliorer le traitement de la partie arithmétique du raisonnement. Dans de nombreux cas, ceci n'améliorera pas seulement l'efficacité du solveur, mais permettra aussi d'exprimer les contraintes arithmétiques des problèmes d'une manière radicalement plus compacte.

Pensez au jeu de Kamaji [2] où le joueur doit grouper des nombres adjacents dans une grille de sorte que leur somme soit égale à un nombre fixe. Résoudre le jeu nécessite essentiellement un raisonnement logique mais a aussi besoin d'un peu d'arithmétique (addition).

Si $x_{i,j}$ pour chaque case (i, j) est un entier et $G(i, j, i, k)$ représente le fait que les cases (i, j) à (i, k) de la ligne i forment un groupe, la contrainte de somme pourrait être exprimée par :

$$\sum_{m \in E} x_{i,m} = N$$

où N est le nombre fixe et E est $\{j, j + 1, \ldots, k\}$. La logique propositionnelle pure n'est certainement pas adaptée pour de telles phrases !

4.2 TouIST pour la planification classique SAT

En Intelligence Artificielle, la *planification* est un processus cognitif qui consiste à générer automatiquement, au travers d'une procédure formelle, un résultat articulé sous la forme d'un système de décision intégré appelé *plan*. Le plan est généralement sous la forme d'une collection organisée d'*actions* et il doit permettre à l'univers d'évoluer de l'*état initial* à un état qui satisfait le *but*. Dans le cadre classique, le plus restrictif, on considère les actions comme des transitions instantanées sans prendre en compte le temps.

La planification par satisfaction de base de clauses (planification SAT) a été introduite avec le planificateur SATPLAN [4]. Dans cette approche, on travaille directement sur un ensemble fini de variables propositionnelles. Deux actions identiques pouvant apparaître à des endroits différents d'un même plan doivent pouvoir être différenciées et on leur associe donc des propositions différentes. Comme on ne connaît pas à l'avance

2. http://fr.wikipedia.org/wiki/Kamaji

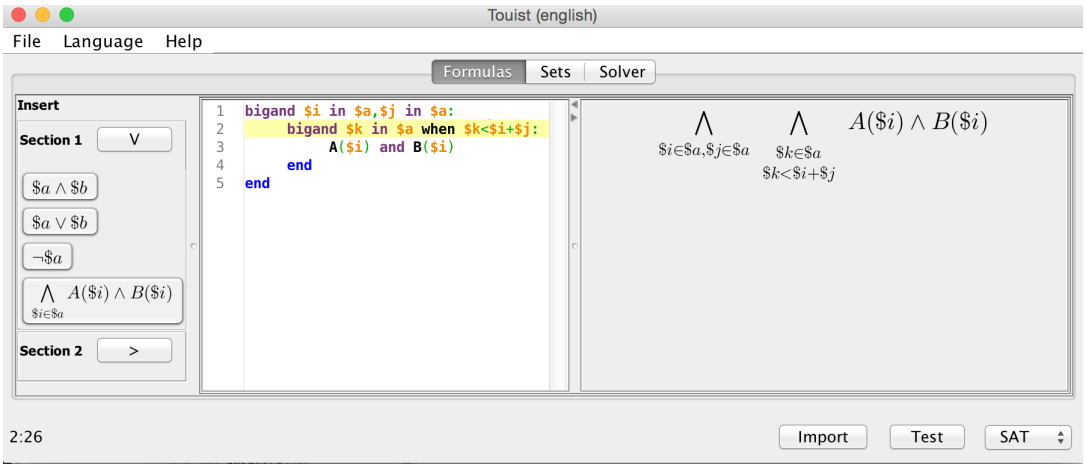

FIGURE 3 – Affichage en style LaTeX

la longueur d'un plan solution d'un problème, on ne peut pas créer un codage unique permettant de le résoudre puisqu'il faudrait créer une infinité de variables propositionnelles pour représenter toutes les actions de tous les plans possibles. La solution la plus commune consiste alors à créer un codage représentant tous les plans d'une longueur k fixée. La base de clause ainsi obtenue est donnée en entrée à un solveur SAT qui retourne, lorsqu'il existe, un modèle de cette base. Le décodage de ce modèle permet alors d'obtenir un plan-solution. Si la résolution du codage ne donne pas de modèle, la valeur de k est augmentée et le processus réitéré. Pour la complétude du procédé, tous ces modèles doivent correspondre exactement à tous les plans solutions d'une longueur fixée du problème.

Une différence importante de TouIST en comparaison avec SATouLouSE est sa capacité à prendre en compte à la fois des formules logiques et ensembles de domaines. Par exemple, si l'on veut résoudre un problème de planification particulier, SATouLouSE est facile à utiliser pour décrire le problème et le résoudre via un solveur SAT. Mais si notre objectif est de résoudre plusieurs problèmes de planification génériques, nous pouvons profiter de la flexibilité de TouIST qui permet à l'utilisateur de décrire une méthode générique de résolution avec des règles encodées comme formules et d'utiliser des ensembles de domaines pour décrire chaque problème de planification particulier. De nombreuses règles de codage pour la résolution de problèmes de planification ont déjà été proposées [4, 5, 7]. Comme exemple d'une telle règle nous donnons ci-après un codage des *frame-axioms*. Si un fait est faux à une étape $i-1$ d'un plan solution et devient vrai à l'étape i, alors la disjonction des actions qui peuvent établir le fait à l'étape i du plan est vraie. C'est à dire, au moins une action qui établit le fait doit avoir été appliquée.

$$\bigwedge_{i\in\{1..longueur\_plan\}} \bigwedge_{f\in Faits} \left((\neg f(i-1) \wedge f(i)) \Rightarrow \bigvee_{a\in Actions/f\in Effets(a)} a(i) \right)$$

4.3 TouIST pour la planification temporelle SMT

Pour résoudre des problèmes de planification réels, l'une des principales difficultés à surmonter consiste à prendre en compte la dimension temporelle. En effet, de nombreux problèmes du monde réel nécessitent, pour être résolus ou exécutés plus efficacement, la prise en compte de la durée des actions, des instants auxquels des évènements se produisent, ou encore la concurrence d'actions.

En complément de SAT, notre nouvelle plateforme TouIST est capable de gérer des théories comme la différence logique ou l'arithmétique linéaire sur les nombres entiers (QF_IDL, QF_LIA) ou les nombres réels (QF_RDL, QF_LRA), et d'appeler un solveur SMT pour trouver une solution. Pour être résolus, les problèmes de planification temporelle issus du monde réel nécessitent une représentation continue du temps, et donc, l'utilisation de nombres réels dans les codages logiques. TouIST peut être utilisé pour résoudre de tels problèmes impliquant des actions avec durée, des évènements exogènes et des buts temporellement étendus, par exemple avec les règles de codage proposées dans [6]. Nous donnons ci-après un codage des exclusions mutuelles temporelles d'actions. Si deux actions a et b

produisant respectivement une proposition p et sa négation sont actives dans le plan ($a \wedge b$ est vrai), alors l'intervalle de temps $[\tau(a \mid\!\rightarrow p), \tau(a \rightarrow\!\mid p)]$ correspondant à l'activation de p, et l'intervalle de temps $[\tau(b \mid\!\rightarrow \neg p), \tau(b \rightarrow\!\mid \neg p)]$ correspondant à l'activation de $\neg p$ sont disjoints. Dans ce cas, il faut ajouter une disjonction pour imposer que la fin de l'un de ces intervalles soit strictement avant le début de l'autre.

$$\bigwedge_{a \in Actions} \bigwedge_{b \in Actions} \bigwedge_{f \in Faits | f \in Effets(a) \wedge \neg f \in Effets(b)}$$

$$\left((a \wedge b) \Rightarrow \left(\begin{array}{c} (\tau(b \rightarrow\!\mid \neg f) < \tau(a \mid\!\rightarrow f)) \\ \vee (\tau(a \rightarrow\!\mid f) < \tau(b \mid\!\rightarrow \neg f)) \end{array} \right) \right)$$

5 Conclusion

A notre connaissance, il n'existe pas d'autre outil qui cible un public aussi large, ni la même grande classe de problèmes, ni avec le même confort d'utilisation. La plupart des outils pédagogiques existants (soit l'implémentation de tables de vérité ou tableaux sémantiques) qui pourraient faire le travail de recherche d'un modèle ne peuvent pas gérer efficacement de gros probèmes, et de vrais outils capables de les traiter ne sont cetainement pas conçus pour être utilisés par des débutants en logique, et même pas par la plupart des étudiants de cycles supérieurs. Des outils avancés conçus pour des sujets d'études supérieures, comme Mozart [8] ou Alloy [3] ont une courbe d'apprentissage raide qui peut dissuader les débutants et les utilisateurs non-spécialistes.

Nous croyons que `TouIST` sera utile pour les débutants en logique ainsi que pour les utilisateurs avancés grâce à sa grande gamme d'applications et sa facilité d'utilisation.

Références

[1] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3) :59–6, 2010.

[2] Olivier Gasquet, François Schwarzentruber, and Martin Strecker. Satoulouse : the computational power of propositional logic shown to beginners. In P. Blackburn, H. van Ditmarsch, M. Manzano, and F. Soler-Tosca, editors, *Third International Congress on Tools for Teaching Logic (ICTTL'2011)*, volume 6680 of *Lecture Notes in Computer Science*, pages 77–84. Springer, 2011.

[3] Daniel Jackson. *Software Abstractions : Logic, Language, and Analysis*. The MIT Press, 2006.

[4] Henry Kautz and Bart Selman. Planning as satisfiability. In *IN ECAI-92*, pages 359–363. Wiley, 1992.

[5] Amol Dattatraya Mali and Subbarao Kambhampati. On the utility of plan-space (causal) encodings. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, 1999*, pages 557–563, 1999.

[6] Frédéric Maris and Pierre Régnier. Tlp-gp : New results on temporally-expressive planning benchmarks. In *International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 507–514. IEEE Computer Society, 2008.

[7] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability : Parallel plans and algorithms for plan search. *Artif. Intell.*, 170(12) :1031–1080, 2006.

[8] Peter Van Roy, editor. *Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004, Charleroi, Belgium, October 7-8, 2004, Revised Selected and Invited Papers*, volume 3389 of *Lecture Notes in Computer Science*. Springer, 2005.