Rapport projet 2, TSAT - Solveur SAT/SMT

T. Stérin, A. Torres

Mai 2016

Abstract

TSAT est un solveur SAT/SMT à l'instar de minisat [1] qui a pour but de satisfaire des formules logiques qui lui sont données en entrée. TSAT est écrit en C++ par Tristan Stérin et Alexy Torres–Aurora-Dugo. Réalisé à l'ENS Lyon pour l'année scolaire 2015-2016 et dans le cadre de la matière "Projet2", ce solveur permet de résoudre des formules SAT/SMT sous formes CNF et logique (en appliquant la transformation de Tseitin).

Différentes méthode de résolution ont été implémentées dans TSAT :

- DPLL (Davis–Putnam–Logemann–Loveland) standard [2].
- DPLL avec méthode des litéraux surveillés [3].
- DPLL avec méthode avec apprentissage de clause [4].

1 Organisation du code

L'organisation du projet est la suivante :

```
src/
├── BETHeuristic (Classes contenant les stratégies de pari)
├── Global (Fichiers source contenant les paramètres du programme)
├── NewCore (Noyau TSAT, structures de clause)
│   └── SMT (Noyau SMT)
├── Parser (Parseurs CNF et logique)
├── RandomSatExpGenerator (Générateur de formules SAT)
├── SMTGenerator (FGénérateur de formules SMT)
├── main.cpp (Fichier source principal du solveur)
├── main.h (En tête du fichier main.cpp)
└── Makefile (Fichier de compilation du solveur)
```

2 Structures de données utilisées

3 Choix des patrons de conception

Le patron de conception en statégies à été retenu pour le développement des heuristiques de pari. Celà permet d'ajouter une heuristique sans modifier le

noyau SAT. Ainsi il est aisé de créer une nouvelle classe heuristique et de l'ajouter au programme en l'incluant dans le fichier main.cpp et modifiant quelques lignes de code dans ce fichier. Cette méthode de conception permet aussi de modifier l'heuristique de pari durant le traitement de la formule.

Nous avons aussi développé les différents parseurs de formule à l'aide de ce patron de conception afin de faciliter l'ajout de nouveaux types de parseurs.

4 Choix des implémentations

MAP pour gagner du temps Files ? Lien avec SMT?

5 Détection de l'UIP

6 Critique des performances

La méthode avec apprentissage de clause permet d'obtenir de très bons résultats. Certaine formules alors impossible à traiter à l'aide de DPLL standard et de la méthode avec les litéraux surveillés obtiennent un solution (affféctation ou réponse d'insatisfaisabilité) en un temps très court.

Certaines heuristiques de pari nous donne de meilleurs résultats dans certain cas. Ainsi, RAND permet dans les meilleur cas de satisfaire des formules en moins d'une seconde alors que l'heuristique standard ne me permet pas en moins de dix. Les meilleurs résultats s'obtiennent avec l'heuristique MOMS.

Les optimisations à la compilation (-O3) permettent elles aussi de rendre le solveur beaucoup plus rapide (jusqu'à 70% plus rapide dans certain cas).

7 Amélioration possible

Une des possibilité d'amélioration

References

[1] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[2] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[3] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of*

the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001, pages 530–535. ACM, 2001.

[4] João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.