A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories

Alessandro Cimatti[1], Alberto Griggio[2], and Roberto Sebastiani[2]

[1] ITC-IRST, Povo, Trento, Italy. cimatti@itc.it
[2] DIT, Università di Trento, Italy. {griggio,rseba}@dit.unitn.it

Abstract. Finding small unsatisfiable cores for SAT problems has recently received a lot of interest, mostly for its applications in formal verification. Surprisingly, the same problem in the context of SAT Modulo Theories (SMT) has instead received very little attention in the literature; in particular, we are not aware of any work aiming at producing small unsatisfiable cores in SMT.
The purpose of this paper is to start filling the gap in this area, by proposing a novel approach for computing small unsat cores in SMT. The main idea is to combine an SMT solver with an external propositional core extractor: the SMT solver produces the theory lemmas found during the search; the core extractor is then called on the boolean abstraction of the original SMT problem and of the theory lemmas. This results in an unsatisfiable core for the original SMT problem, once the remaining theory lemmas have been removed.
The approach has several advantages: it is extremely simple to implement and to update, and it can be interfaced with every propositional core extractor in a plug-and-play way, so that to benefit for free of all unsat-core reduction techniques which have been or will be made available.

1 Motivations and goals

In the last decade we have witnessed an impressive advance in the efficiency of SAT techniques, which has brought large and previously intractable problems at the reach of state-of-the-art SAT solvers. In particular, and due to its importance in formal verification, the problem of finding small *unsatisfiable cores* in SAT — i.e., unsatisfiable subsets of unsatisfiable sets of clauses— has been addressed by many authors in the recent years [8, 9, 11, 4, 7, 10].

The formalism of plain propositional logic, however, is often not suitable or expressive enough for representing many interesting real-world problems, which are more naturally expressible as satisfiability problems in decidable first-order theories —Satisfiability Modulo Theories, SMT. Efficient SMT solvers have been developed in the last five years, called *lazy* SMT solvers, which combine DPLL with ad-hoc decision procedures for many theories of interest (e.g., [6, 1, 2, 5]).

Surprisingly, the problem of finding unsatisfiable cores in SMT has received virtually no attention in the literature. Although some SMT tools do compute unsat cores, this is done either as a byproduct of the more general task of producing proofs, or by modifying the embedded DPLL solver so that to apply

basic propositional techniques to produce an unsat core. In particular, we are not aware of any work aiming at producing *small* unsatisfiable cores in SMT.

In this paper we present a novel approach addressing this problem. The main idea is to combine an SMT solver with an external propositional core extractor. The SMT solver stores and returns the theory lemmas it had to prove in order to refute the input formula; the external core extractor is then called on the boolean abstraction of the original SMT problem and of the theory lemmas. The resulting boolean unsatisfiable core is cleaned from (the boolean abstraction of) all theory lemmas, and it is refined back into a subset of the original clauses. The result is an unsatisfiable core of the original SMT problem.

Although simple in principle, the approach is conceptually interesting: basically, the SMT solver is used to dynamically lift the suitable amount of theory information to the boolean level. Furthermore, the approach has several advantages in practice: first, it is extremely simple to implement and to update; second, it is effective in finding small cores; third, the core extraction is not prone to complex SMT reasoning; finally, it can be interfaced with every propositional core extractor in a plug-and-play manner, so that to benefit for free of all unsat-core reduction techniques which have been or will be made available.

For lack of space, in this short version of the paper we omit many details, any related work and the description and the results of our extensive experimental evaluation of the approach. They can be found in the extended version of the paper [3].

2 Background

Given a decidable first-order theory $\mathcal{T}$, we call a *theory solver for* $\mathcal{T}$, *$\mathcal{T}$-solver*, any tool able to decide the satisfiability in $\mathcal{T}$ of sets/conjunctions of ground atomic formulas and their negations (*theory literals* or *$\mathcal{T}$-literals*) in the language of $\mathcal{T}$. If the input set of $\mathcal{T}$-literals μ is $\mathcal{T}$-unsatisfiable, then a typical *$\mathcal{T}$-solver* not only returns `unsat`, but it also returns the subset η of $\mathcal{T}$-literals in μ which was found $\mathcal{T}$-unsatisfiable. (η is hereafter called a *theory conflict set*, and $\neg\eta$ a *theory conflict clause*.) If μ is $\mathcal{T}$-satisfiable, then *$\mathcal{T}$-solver* not only returns `sat`, but it may also be able to discover one (or more) deductions in the form $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$, s.t. $\{l_1, ..., l_n\} \subseteq \mu$ and l is an unassigned $\mathcal{T}$-literal. If so, we call $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ a *theory-deduction clause*. Importantly, notice that both theory-conflict clauses and theory-deduction clauses are valid in $\mathcal{T}$. We call them *theory lemmas* or *$\mathcal{T}$-lemmas*.

Satisfiability Modulo (the) Theory $\mathcal{T}$ ($SMT(\mathcal{T})$) is the problem of deciding the satisfiability of *boolean combinations* of propositional atoms and theory atoms. We call an $SMT(\mathcal{T})$ *tool* any tool able to decide $SMT(\mathcal{T})$. Notice that, unlike a *$\mathcal{T}$-solver*, an $SMT(\mathcal{T})$ tool must handle also boolean connectives.

Hereafter we adopt the following terminology and notation. The bijective function $\mathcal{T}2\mathcal{P}$ ("theory-to-propositional"), called *boolean abstraction*, maps propositional variables into themselves, ground $\mathcal{T}$-atoms into fresh propositional variables, and is homomorphic w.r.t. boolean operators and set inclusion. The func-

tion $\mathcal{P}2\mathcal{T}$ ("propositional-to-theory"), called *refinement*, is the inverse of $\mathcal{T}2\mathcal{P}$. The symbols φ, ψ denote $\mathcal{T}$-formulas, and μ, η denote sets of $\mathcal{T}$-literals; φ^p, ψ^p denote propositional formulas, μ^p, η^p denote sets of propositional literals (i.e., truth assignments) and we often use them as synonyms for the boolean abstraction of φ, ψ, μ, and η respectively, and vice versa (e.g., φ^p denotes $\mathcal{T}2\mathcal{P}(\varphi)$, μ denotes $\mathcal{P}2\mathcal{T}(\mu^p)$). If $\mathcal{T}2\mathcal{P}(\varphi) \models \bot$, then we say that φ is *propositionally unsatisfiable*.

2.1 Lazy techniques for SMT

The idea underlying every lazy $SMT(\mathcal{T})$ procedure is that (a complete set of) the truth assignments for the propositional abstraction of φ are enumerated and checked for satisfiability in $\mathcal{T}$; the procedure either returns sat if one $\mathcal{T}$-satisfiable truth assignment is found, or returns unsat otherwise.

A simplified schema of a lazy $SMT(\mathcal{T})$ procedure is as follows. The propositional abstraction φ^p of the input formula φ is given as input to a modified version of a DPLL solver, and when a satisfying assignment μ^p is found, the refinement μ of μ^p is fed to the *$\mathcal{T}$-solver*; if μ is found $\mathcal{T}$-consistent, then φ is $\mathcal{T}$-consistent; otherwise, *$\mathcal{T}$-solver* returns the conflict set η which caused the $\mathcal{T}$-inconsistency of $\mathcal{P}2\mathcal{T}(\mu^p)$. Then the clause $\neg\eta^p$ is added in conjunction to φ^p, either temporarily or permanently (*$\mathcal{T}$-learning*), and the algorithm backtracks up to the highest point in the search where a literal can be unit-propagated on $\neg\eta^p$ (*$\mathcal{T}$-backjumping*).

Two important optimizations are *early pruning* and *theory propagation*: the $\mathcal{T}$-solver is invoked also on (the refinement of) an intermediate assignment μ: if it is found $\mathcal{T}$-unsatisfiable, then the procedure can backtrack, since no extension of μ can be $\mathcal{T}$-satisfiable; if not, and if the *$\mathcal{T}$-solver* performs a deduction $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$ s.t. $\{l_1, ..., l_n\} \subseteq \mu$, then $\mathcal{T}2\mathcal{P}(l)$ can be unit-propagated, and the boolean abstraction of the $\mathcal{T}$-lemma $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ can be learned.

The above schema is a coarse abstraction of the procedures underlying all the state-of-the-art lazy $SMT(\mathcal{T})$ tools like, e.g., BarceLogic, CVCLite, MathSAT, Yices. The interested reader is pointed to, e.g., [6, 1, 2, 5], for details and further references.

2.2 Techniques for unsatisfiable-core extraction in SAT

Given an unsatisfiable (propositional) CNF formula φ, we say that an unsatisfiable CNF formula ψ is an *unsatisfiable core* of φ iff $\varphi = \psi \wedge \psi'$ for some (possibly empty) CNF formula ψ'. Intuitively, ψ is a subset of the clauses in φ causing the unsatisfiability of φ. An unsatisfiable core ψ is *minimal* iff the formula obtained by removing any of the clauses of ψ is satisfiable. A *minimum* unsat core is a minimal unsat core with the smallest possible cardinality.

In the last few years, several algorithms for computing small [11], minimal [7, 4] or minimum [8–10] unsatisfiable cores of propositional formulas have been proposed. For lack of space, we can not provide details here, and we refer to the extended version [3] of the paper for a detailed description.

```
⟨SatValue,Clause_set⟩ T-Unsat_Core(Clause_set φ) { // φ is {C_1, ..., C_n}
  if (T-DPLL(φ) == sat) then return ⟨sat,∅⟩;
  // D_1, ..., D_k are the T-lemmas stored by T-DPLL
  ψ^p=Boolean_Unsat_Core(T2P({C_1, ..., C_n, D_1, ..., D_k}));
  // ψ^p is T2P({C'_1, ..., C'_m, D'_1, ..., D'_j}));
  return ⟨unsat,{C'_1, ..., C'_m}⟩; }
```

Fig. 1. Schema of the $\mathcal{T}$-Unsat_Core algorithm.

2.3 Techniques for unsatisfiable-core extraction in SMT

To the best of our knowledge, there is no published work in the literature devoted to the computation of unsatisfiable cores in SMT. However, at least three SMT solvers support unsat core generation with techniques adapted from SAT. CVCLite [1] and a recent extension of MathSAT [2] can compute unsatisfiable cores as a byproduct of the generation of proofs, in a way similar to that in [11]. Yices [5] instead uses the following technique: a selector variable is introduced for each original clause, which is forced to false before starting the search. In this way, when a conflict at decision level zero is found, the conflict clause contains only selector variables, and the unsat core returned is the union of the clauses whose selectors appear in such conflict clause.

We remark the fact that none of these solvers aims at producing minimal or minimum unsat cores, nor does anything to reduce their size.

3 A novel approach to building unsat cores in SMT

We present a novel approach in which the unsatisfiable core is computed *a posteriori* w.r.t. the execution of the SMT solver, and only if the formula has been found $\mathcal{T}$-unsatisfiable, by means of an external (and possibly optimized) propositional unsat-core extractor.

In the following we assume that a lazy $SMT(\mathcal{T})$ procedure has been run over a $\mathcal{T}$-unsatisfiable $SMT(\mathcal{T})$ CNF formula $\varphi =_{def} \{C_1, ..., C_n\}$, and that $D_1, ..., D_k$ denote all the $\mathcal{T}$-lemmas, both theory-conflict and theory-deduction clauses, which have been returned by the $\mathcal{T}$*-solver* during the run.

Our novel approach is based on two simple facts.

(i) Under the assumptions above, the conjunction of φ with all the $\mathcal{T}$-lemmas $D_1, ..., D_k$ is propositionally unsatisfiable: $\mathcal{T}2\mathcal{P}(\varphi \wedge \bigwedge_{i=1}^{n} D_i) \models \bot$.

(ii) As $\mathcal{T}$-lemmas D_i are valid in $\mathcal{T}$, they do not affect the $\mathcal{T}$-satisfiability of a formula: $(\psi \wedge D_i) \models_{\mathcal{T}} \bot \iff \psi \models_{\mathcal{T}} \bot$.

These facts suggest the novel algorithm represented in Figure 1. The procedure $\mathcal{T}$`-Unsat_Core` receives as input a set of clauses $\varphi =_{def} C_1, ..., C_n$ and it invokes on it a lazy $SMT(\mathcal{T})$ tool $\mathcal{T}$`-DPLL`, which is instructed to *store* somewhere the $\mathcal{T}$-lemmas returned by $\mathcal{T}$*-solver*, namely $D_1, ..., D_k$. If $\mathcal{T}$`-DPLL` returns

sat, then the whole procedure returns sat. Otherwise, the boolean abstraction of $\{C_1, ..., C_n, D_1, ..., D_k\}$, which is inconsistent because of (i), is passed to an external tool `Boolean_Unsat_Core`, which is able to return the boolean unsat core ψ^p of the input. By construction, ψ^p is the boolean abstraction of a clause set $\{C'_1, ..., C'_m, D'_1, ..., D'_j\}$ s.t. $\{C'_1, ..., C'_m\} \subseteq \{C_1, ..., C_n\}$ and $\{D'_1, ..., D'_j\} \subseteq \{D_1, ..., D_k\}$. As ψ^p is unsatisfiable, then $\{C'_1, ..., C'_m, D'_1, ..., D'_j\}$ is $\mathcal{T}$-unsat. By (ii), the $\mathcal{T}$-valid clauses $D'_1, ..., D'_j$ have no role in the $\mathcal{T}$-unsatisfiability of $\{C'_1, ..., C'_m, D'_1, ..., D'_j\}$, so that the procedure returns unsat and the $\mathcal{T}$-unsat core $\{C'_1, ..., C'_m\}$.

The procedure can be implemented very simply by modifying the SMT solver so that to store the $\mathcal{T}$-lemmas [3] —if it doesn't already— and by interfacing it with some state-of-the-art boolean unsat-core extractor used as an external black-box device (e.g., by a simple exchange of files in DIMACS format). Moreover, if the SMT solver can provide the set of all $\mathcal{T}$-lemmas as output, then the whole procedure may reduce to a control device interfacing with both the SMT solver and the boolean core extractor as black-box external devices.

3.1 Discussion

Though based on an extremely simple concept, the newly-proposed approach is appealing for several reasons.

First, it is extremely simple to implement and update. The building of unsat cores is demanded to an external device, which is fully decoupled from the internal DPLL-based enumerator. Therefore, there is no need to implement any internal unsat-core constructor nor to modify the embedded boolean device. Every possible external device can be interfaced in a plug-and-play manner by simply exchanging a couple of DIMACS files.

Second, from the perspective of effectiveness in reducing the size of unsat cores, every original clause which the boolean unsat-core device is able to drop is dropped also in the final formula. Therefore, this technique exploits for free all unsat-core reduction techniques which have been and will be conceived in the SAT community.

One potential drawback of this approach is the fact that a $SMT(\mathcal{T})$ solver is required to store all the $\mathcal{T}$-lemmas returned by the *$\mathcal{T}$-solver*. However, this is not a real problem. In fact, unlike with plain SAT, in lazy SMT the computational effort is typically dominated by the search in the theory $\mathcal{T}$, so that the number of clauses that can be stored with a reasonable amount of memory is typically much bigger than the number of calls to the *$\mathcal{T}$-solver* which can overall be accomplished within a reasonable amount of time. In our experience, even the hardest SMT formulas at the reach of current lazy SMT solvers rarely need generating more than 10^5 $\mathcal{T}$-lemmas, which have very reasonable memory requirements to store. (E.g., notice that the default choice in MathSAT is to learn all $\mathcal{T}$-lemmas

[3] Notice that here "storing" does not mean "learning": the SMT solver is not required to add the $\mathcal{T}$-lemmas to the formula during the search. This imposes no constraint on the lazy strategy adopted.

permanently anyway, and we have never encountered memory overload problems due to this fact.)

Finally, one limitation of this approach is that the resulting $\mathcal{T}$-unsatisfiable core is not guaranteed to be minimal, even if `Boolean_Unsat_Core` returns minimal boolean unsatisfiable cores. However, to the best of our knowledge, not only the issue of the *minimality* of unsat cores in SMT has never been addressed or even discussed before, but also this is the first time that the problem of the *size* of unsat cores in SMT is addressed.

4 Conclusions

We have presented a novel approach to generating small unsatisfiable cores in SMT, that computes them a posteriori, relying on an external propositional unsat core extractor. The technique is very simple in concept, and straightforward to implement and update. Moreover, it benefits for free of all the advancements in propositional unsat core computation. Our experimental results, available in the extended version [3], have shown that, by using different core extractors, it is possible to reduce significantly the size of cores and to trade core quality for speed of execution (and vice versa), with no implementation effort.

References

1. C. Barrett and S. Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In *Proc. CAV'04*, 2004.
2. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P.van Rossum, S. Schulz, and R. Sebastiani. An incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In *Proc. TACAS'05*, 2005.
3. A. Cimatti, A. Griggio, and R. Sebastiani. A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories. Technical Report DIT-07-006, DIT, Univ. of Trento, 2007. Extended version. Available at `http://dit.unitn.it/~griggio/papers/sat07_extended.pdf`.
4. N. Dershowitz, Z. Hanna, and A. Nadel. A Scalable Algorithm for Minimal Unsatisfiable Core Extraction. In *Proc. SAT'06*, 2006.
5. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc. CAV'06*, 2006.
6. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *Proc. CAV'04*, 2004.
7. R. Gershman, M. Koifman, and O. Strichman. Deriving Small Unsatisfiable Cores with Dominators. In *Proc. CAV'06*, 2006.
8. I. Lynce and J. P. Marques Silva. On computing minimum unsatisfiable cores. In *Proc. SAT'04*, 2004.
9. M. N. Mneimneh, I. Lynce, Z. S. Andraus, J. P. Marques Silva, and K. A. Sakallah. A Branch-and-Bound Algorithm for Extracting Smallest Minimal Unsatisfiable Formulas. In *Proc. SAT'05*, 2005.
10. J. Zhang, S. Li, and S. Shen. Extracting Minimum Unsatisfiable Cores with a Greedy Genetic Algorithm. In *Proc. ACAI'06*, 2006.
11. L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formula. In *Proc. SAT'03*, 2003.