Translations to propositional satisfiability

Juan Antonio Navarro-Pérez
Supervisor: Andrei Voronkov, Adviser: Renate Schmidt

The University of Manchester
School of Computer Science
Oxford Rd. Manchester M13 9PL, UK.
navarroj@cs.manchester.ac.uk

1 Introduction

Propositional satisfiability (SAT) solvers have recently been found as a competitive approach in many industrial applications. In component verification, for example, one is given a formal description of a system (usually a finite state machine) and a property that is to be checked against the system. Clarke et al. (2001) proposed a method, *bounded model checking*, where both the finite state machine and the property are encoded into a propositional logic formula. The formula is then fed into a satisfiability solver and, if a satisfying model is found, it can be used as a counterexample to show an execution trace in which the system fails to satisfy the required property.

Dramatic improvements in SAT solver technology (Mitchell, 2005) and the availability of efficient implementations (Eén and Sörensson, 2005) have been continuously improving the applicability of this approach in industrial settings. However, researchers have also observed that a very relevant decision is how to encode or translate the problem into a form that is suitable for use by a satisfiability solver. Even a slight modification in syntax of the resulting formula could change the search space explored by the solver in a rather unpredictable way. In this short report we present an overview of our experiences translating bounded model checking problems into satisfiability solving, outlining both ideas that seem promising and others that did not prove to be very successful.

2 An economic clause normal form translation

In order to simplify its design and implementation, modern satisfiability solvers typically work on formulae in *clause normal form* (CNF) so that arbitrary propositional formulae first need to be translated into this format. Tseitin (1968) proposed a translation that works by introducing *definitions* that map each subformula of the original expression to a new atom, e.g. $x \leftrightarrow (a_1 \wedge \cdots \wedge a_n)$. The newly introduced atoms can be used instead of the corresponding subformulae while their definitions, which are simpler flat formulae, can be easily translated to CNF. Plaisted and Greenbaum (1986) further improve this idea showing that, by considering the *polarity* of subformulae (i.e. how it is affected by the scope of negations), it is sometimes enough to include one implication of the definition, e.g. $x \rightarrow (a_1 \wedge \cdots \wedge a_n)$ only.

It has been noted, however, that the translation tends to introduce many unnecessary atoms. One of the simplifications of the SATELITE preprocessor (Eén and Biere, 2005), for example, is specifically targeted to identify atoms introduced by Tseitin's translations and 'undo' their definition. This motivated our interest to design a translation which is still polynomial, but trying to introduce as few new atoms as possible. Some of our main ideas follow:

- We consider logic connectives with multiple arity. In particular equivalence and exclusive disjunction are replaced by more general *parity constraints* (i.e. a parity constraint is true if an odd/even number of their operands is true).
- The formula is translated to a *negation normal form* first, where the negation connective is only applied to atoms, but parity connectives are still allowed anywhere. This simplifies the computation of *polarity*: every formula has positive polarity, except for subformulae under the scope of parity constraints which have polarity zero.
- Subformulae with polarity zero are translated doing a *full renaming* as Tseitin does. Other subformulae can be more carefully translated, adding only one atom for each conjunction, using the idea that a *half-definition* $x \rightarrow F$, where F is already in CNF, can be translated without adding any more atoms (simply append $\neg x$ to each clause in F).
- *Parity clauses*, which are parity constraints applied to literals only, are left untouched until the very end of the translation. At this point, parity clauses of length 2 and 3 can be translated using a standard clausification. Longer parity clauses, of length n, can be split into $n - 2$ parity clauses of length 3 by adding $n - 3$ new atoms.

An important difference of our implementation, with respect to common approaches (e.g. Clarke et al., 2001), is that we do not use subformula sharing. This is under the rationale that, in problems from real life applications, it is unlikely to expect large blocks of identical subformulae to share. If two large identical formulae were needed in the description of a system, the human writing such description would probably recognise it and emulate the 'sharing' by introducing a definition of a new atom himself. This can be observed, for instance, in typical benchmarks of model checking problems where system descriptions consist mainly of definitions.

In some preliminary experimental results we did obtained some significant improvements compared to the translation provided by the BMC tool of Clarke et al. (2001). For a particular example (a safety property on a barrel shifter), our translation halved in average both the total number of clauses and variables of the formula produced, allowing speedups of solving time by an order of magnitude. However, more experimentation is needed in order to give any conclusive results.

3 A SAT solver with direct support of parity clauses

During the development of our translation we observed that, potentially, most atoms are introduced while clausifying parity clauses. It then made sense to implement a solver with direct support of parity clauses. Moreover, common implementation tricks such as the *watched literals scheme* (Mitchell, 2005) could be easily adapted to work with parity clauses.

We implemented this idea in a version of MINISAT (Eén and Sörensson, 2005) with support of custom constraints but, unfortunately, it did not quite work as expected. First we observed that problems from typical applications do contain large amounts of parity clauses, but most of them are of size 2 or 3 which do not introduce new atoms when clausified anyway. Moreover the possible advantages of the approach, such as having a more compact representation and a reduced number of literals to watch, turned out to be negligible for this short parity clauses.

Another problem we faced is that there was no obvious way to write a formula such as $x \rightarrow (a \oplus b)$,[1] which are often produced in half-definitions of the translation, using clauses and parity clauses only. One approach was to clausify the right hand side and loose any advantage of having support of such parity constraints. The other approach was to ignore the polarity optimisation in such cases and write a single parity clause $(\neg x \oplus a \oplus b)$. This, however, dramatically decreased the effectiveness of the solver, making it explore a considerably larger search space in all our experiments.

4 Conclusions

This is currently work in progress. We have been trying many ideas and exploring possible paths to get better representations of problems stemming from real life applications and solve them with propositional satisfiability methodologies. Some of this ideas have proved useful, and some others have not. We have yet a few ideas to try in order to get concise propositional representations of formulae and, in our immediate future work, we also plan to study the applicability of alternative translations into decidable fragments of first order logic.

References

Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, 2001. ISSN 0925-9856.

Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proc. 8th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *Lecture Notes in Computer Science*. Springer, 2005.

Niklas Eén and Niklas Sörensson. MINISAT — a SAT solver with conflict-clause minimization. Poster presented at the SAT 2005 Competition, 2005.

David G. Mitchell. A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science*, 85: 112–133, February 2005. Logic in Computer Science Column.

David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304, 1986. ISSN 0747-7171.

Grigori S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part II*, 1968.

[1] We use $\oplus$ to denote the odd parity connective.