Effective Bit-Width and Under-Approximation

Robert Brummayer and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria

Abstract. Recently, it has been proposed to use approximation techniques in the context of decision procedures for the quantifier-free theory of fixed-size bit-vectors. We discuss existing and novel variants of under-approximation techniques. Under-approximations produce smaller models and may reduce solving time significantly. We propose a new technique that allows early termination of an under-approximation refinement loop, although the original formula is unsatisfiable. Moreover, we show how over-approximation and under-approximation techniques can be combined. Finally, we evaluate the effectiveness of our approach on array and bit-vector benchmarks of the SMT library.

1 Introduction

The problem of Satisfiability Modulo Theories (SMT) is to decide satisfiability of logical formulas expressed in a combination of first-order theories. SMT solvers are used in many applications, e.g. optimization, scheduling, verification, and test case generation. For text books about SMT see [4, 16].

The quantifier-free theory of fixed-size bit-vectors plays an important role in specifying and verifying software and hardware systems. Modelling programs and digital circuits on the bit-vector level, e.g. addition of fixed-size bit-vectors with two's complement arithmetic, allows bit-precise and exact reasoning. Not taking modular arithmetic into account, i.e. using natural numbers instead of bit-vectors, may lead to unsound verification results, e.g. bugs caused by overflows may not be detected.

The theory of bit-vectors can be combined with the theory of arrays in order to model memory in a bit-precise way. This enables reasoning about pointers and pointer arithmetic which is important in software verification. Even reasoning about assembler programs on a symbolic processor is possible [7].

Typically, a system and its properties are represented by formulas. These formulas are combined to one verification formula which is checked by an SMT solver. The solver tries to satisfy the formula in order to find a counter-example where the system violates a property. If the formula is satisfiable, most modern SMT solvers can generate a model of the formula. A model can be used to construct a concrete execution of the system that leads to a property violation.

2 Formula Approximation Techniques

The main motivation of most approximation techniques is to speed-up decision procedures. While over-approximation technique tend to speed-up unsatisfiable formulas, under-approximation techniques tend to speed-up satisfiable formulas. In order to remain sound and complete, approximation techniques are typically combined with a refinement loop. Recently, approximation techniques are also used in the context of decision procedures for bit-vectors [8, 15].

Over-approximation techniques are in the spirit of the Counter Example Guided Abstraction Refinement Framework [10] (CEGAR) and are typically used in lazy SMT approaches [17]. For example, over-approximation techniques are used to decide complex array formulas in [5, 13]. In the rest of this paper we will focus on under-approximation techniques.

The basic idea of under-approximation techniques in the context of bit-vectors is to restrict individual bits of bit-vectors. While such domain restrictions typically lead to a smaller search space and a speed-up for satisfiable formulas, it additionally produces "smaller" models which means that the domain of the variables are smaller. If a small model can be found, it must also be a model of the original formula. Small models are beneficial for diagnosis, for instance if the model is directly analyzed by users for debugging. Furthermore, in the area of test case generation, small models lead to test cases with reduced test data size.

One way of using over-approximations and under-approximations in bit-vector logic has been pioneered in [8]. In the context of under-approximation, the m most significant bits of variables are additionally restricted. The remaining n least significant bits are not concerned by the under-approximation and remain variable. In the rest of this paper we call n the *effective bit-width*.

in [8] it is proposed to use an under-approximation technique which corresponds to sign-extension. Let n be the effective bit-width. The m most-significant bits of a variable are forced to be equal to the n^{th} least significant bit, the last effective bit. This technique reduces the domains of variables and leads to smaller models where bit-vectors are interpreted in the context of two's complement. An example with an effective bit width of four is shown left in Fig. 1.

It is also possible to force the m most significant bits to zero resp. one which we call *zero-extension* resp. *one-extension*. Zero-extension has been suggested in [8]. While zero-extension leads to smaller models where bit-vectors are interpreted in an unsigned context, one-extension is beneficial if a formula has small models with negative values. Examples with an effective bit-width of four are shown in Fig. 1 resp. Fig. 2. In [14] zero-extension and one-extension were used for bounded model checking of embedded software.

We propose an additional under-approximation technique that *partitions bits* of individual bit-vectors into equivalence classes. All bits in one class are forced to have the same value. An example is shown Fig. 2. The under-approximation refinement increases the number of classes per variable, or splits individual classes. The idea of this technique is that only some individual bits of the vector are important to satisfy the formula. Therefore, the other bits can be forced to the same value in order to reduce the search space.

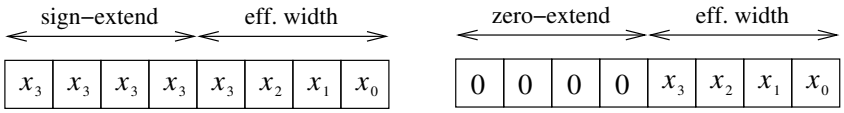

Fig. 1. Under-approximation techniques: Sign-extension is shown left and zero-extension is shown right. The effective bit-width is four in both examples.

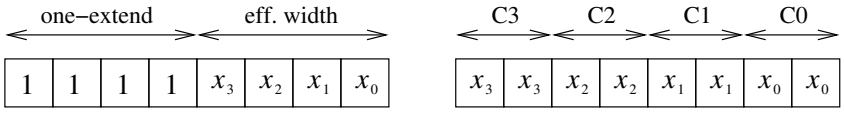

Fig. 2. Under-approximation techniques: One-extension is shown left and class splitting is shown right. The effective bit-width resp. number of classes is four.

3 Under-Approximation Refinement on CNF Layer

We propose to perform under-approximations with the help of additional clauses on the CNF layer. The original formula is translated to CNF once. In each refinement iteration i we perform an under-approximation by adding new clauses to the SAT solver incrementally and also use assumptions as in [9].

First, we introduce a fresh boolean under-approximation variable e. Then, we perform an under-approximation by adding new clauses. Let n be the effective bit width of a bit-vector variable v of bit-width w. To perform a sign-extending under-approximation we add the following clauses:

$$\bigwedge_{i=n}^{w-1} ((v_{n-1} \vee \overline{v_i} \vee \bar{e}) \wedge (\overline{v_{n-1}} \vee v_i \vee \bar{e}))$$

Finally, we assume e to enable the under-approximation.

For example, let v be a bit-vector variable with bit-width eight and an effective bit-width of six. We add the following clauses to perform a sign-extending under-approximation:

$$(v_5 \vee \overline{v_6} \vee \bar{e}) \wedge (\overline{v_5} \vee v_6 \vee \bar{e}) \wedge (v_5 \vee \overline{v_7} \vee \bar{e}) \wedge (\overline{v_5} \vee v_7 \vee \bar{e})$$

Assuming e enforces $v_5 = v_6 = v_7$.

Zero-extension can be encoded as follows. Again, let n be the effective bit width of a bit-vector variable of bit-width w. We add the following clauses:

$$\bigwedge_{i=n}^{w-1} (\overline{v_i} \vee \bar{e})$$

One-extension can be encoded analogously.

If we have to refine our approximation, we add the unit clause $\bar{e}$ in order to disable the current approximation. This gives the SAT solver also the opportunity to recycle clauses. Then, a refined under-approximation is performed with the help of another fresh boolean under-approximation variable.

4 Refinement Strategies

Generally, the effective bit-width can be used as a metric of approximation. Typically, the effective bit-width is initialized to one, i.e. the domain of a bit-vector variable is restricted to $\{-1, 0\}$ in a signed resp. $\{0, 1\}$ in an unsigned context. During the refinement the effective bit width is increased. In order to avoid too many refinement loops, the effective bit-width is typically doubled in each iteration. Traditionally, in the worst case, e.g. if the original formula is unsatisfiable, the effective bit-width reaches the original bit-width.

With the proposed refinement on the CNF layer, two main refinement strategies are possible which we call *global* and *local*. On the one hand, local refinement strategies maintain one fresh boolean under-approximation variable e for each bit-vector in each refinement. The benefit is a precise refinement as we can ask the SAT solver if it has used the respective e to derive unsatisfiability. Only those under-approximations that have been used need to be refined. However, in the worst case we have to introduce $k \cdot r$ fresh variables, where k is the number of bit-vector variables and r is the maximum number of refinements.

On the other hand, the global refinement strategy maintains exactly one under-approximation variable for all bit-vector variables. The benefit is less overhead, as we need only r additional boolean variables, where r is the number of refinements. However, the refinement is imprecise.

5 Early Unsat Termination

Traditional under-approximation techniques perform the under-approximation outside the SAT solver. The CNF is generated from scratch in each refinement iteration. This makes it impossible to find out whether the current under-approximation has been responsible for deriving unsatisfiability or not. In the worst case, the original formula is unsatisfiable and we have the additional overhead of the under-approximation refinement, which is slower than solving the original formula up front.

The proposed under-approximation refinement on the CNF layer enables the decision procedure to terminate earlier, even if the original formula is unsatisfiable. If the under-approximated formula is unsatisfiable, then we can use the under-approximation variables to ask the SAT solver which under-approximations have been used to derive unsatisfiability. If no under-approximation variables have been used, then we can conclude that the original formula is unsatisfiable, and terminate. The early unsat technique is shown in Fig. 3.

Furthermore, the refinement on the CNF layer allows the SAT solver to keep learned conflict clauses over refinement iterations. This would be impossible if the CNF was generated on scratch in each refinement iteration.

In a first implementation we let the SAT solver generate unsat cores modulo assumptions, but simply recording those assumptions [9] that were used in deriving the empty clause is enough, much faster and easier to implement, both on the side of the SAT solver and on the side of the SMT solver.

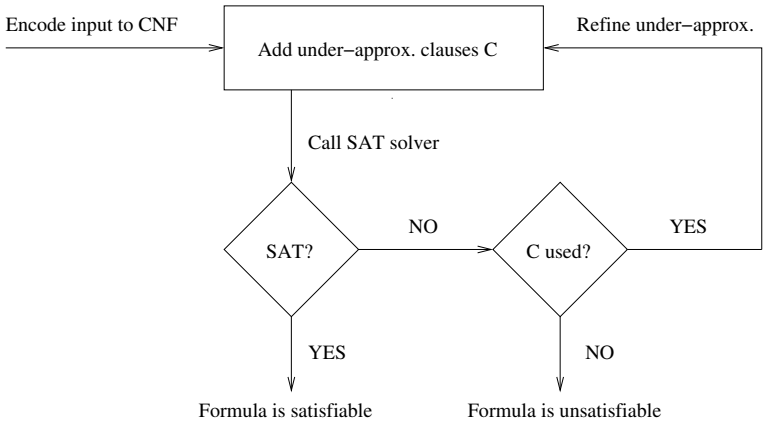

Fig. 3. Early Unsat Termination.

6 Combining Approximation Techniques

Figure 4 shows how over-approximation and under-approximation techniques can be combined to solve complex SMT formulas. We consider the quantifier-free theory of arrays combined with the quantifier-free theory of bit-vectors. The idea is to use over-approximation techniques for the array part [5, 13] and under-approximation techniques for the bit-vector part.

First of all, we perform an over-approximation by replacing reads by fresh bit-vector variables. Then, we translate the bit-vector part of the formula to CNF and add a set C of under-approximation clauses. In each iteration we call the SAT solver. Depending on the result we have to perform an additional check. On the one hand, if the result is *satisfiable* we have to check if the current model σ respects the theory of arrays. If not, we have to refine our over-approximation with a lemma on demand [11, 12, 2]. Otherwise, we can terminate with the model σ and the result *satisfiable*. On the other hand, if the result of the SAT solver is *unsatisfiable* we have to check if the current set of under-approximation clauses has been used. If not, we can terminate with the result *unsatisfiable*. Otherwise, we disable the current under-approximation and continue with a refined approximation.

7 Experiments

We implemented the presented approximation techniques in our SMT solver Boolector [6]. Boolector implements a decision procedure for the quantifier-free theory of fixed-size bit-vectors combined with the quantifier-free extensional theory of arrays [5]. It is the winner of the last SMT competition in 2008 [1] in the bit-vector category (`QF_BV`) and also in the division of bit-vectors with arrays (`QF_AUFBV`). Moreover, Boolector can be used as word-level bounded model

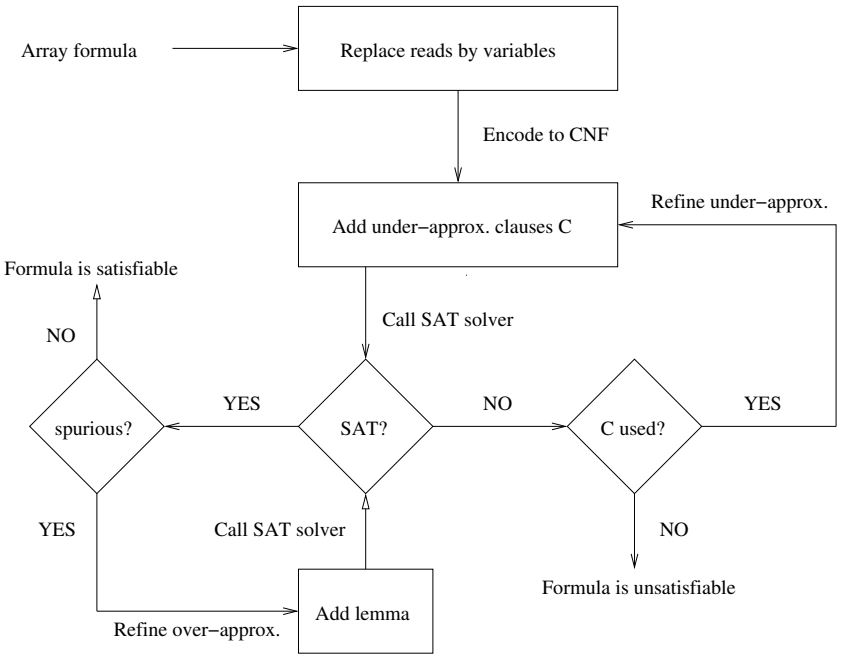

Fig. 4. Combining over-approximation and under-approximation techniques.

checker for synchronous hardware and software systems [7]. For our experiments we used Boolector 1.0.

The results of our experiments are shown in Fig. 5 and Fig. 6. We used benchmarks from `QF_AUFBF` of the SMT library [3] (June 1st, 2008). As expected, under-approximation techniques speed up satisfiable instances, but slow down unsatisfiable instances. We summarize further observations.

First, the under-approximation by classes technique performs as good as the under-approximation by sign-extension technique on the satisfiable instances of `QF_AUFBF`. However, it performs worse on the unsatisfiable benchmarks. Second, the average ratio effective bit-width / original bit-width is 16% (without `egt` examples) on satisfiable and 27% on unsatisfiable benchmarks, which corresponds to an impressive reduction of 84% resp. 73%. Third, early unsat termination occurs in 1553 from 3552 unsat cases. Finally, the global refinement strategy seems to be a good approximation of the local strategy. It is much easier to implement, is often as good as the local strategy, and should be sufficient in most cases.

8 Conclusion

We presented formula approximation techniques, in particular for bit-vector. We discussed different techniques and refinement strategies and showed how they can

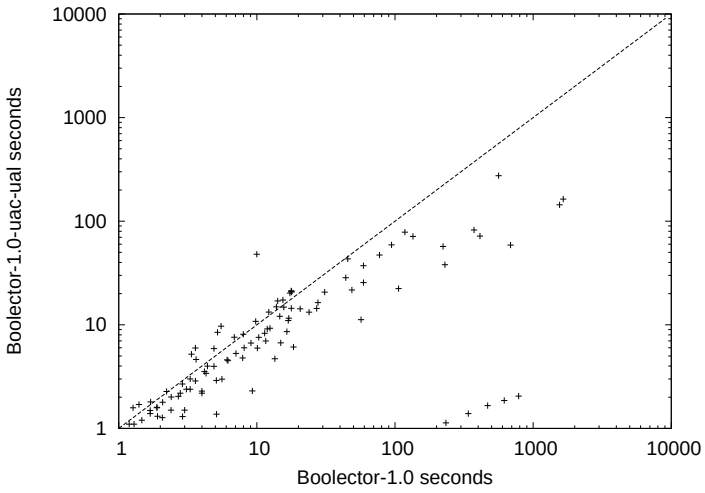

Fig. 5. Boolector (x-axis) vs. Boolector with class under-approximation and local refinement strategy (y-axis). Benchmarks are from `QF_AUFBF` and are all satisfiable.

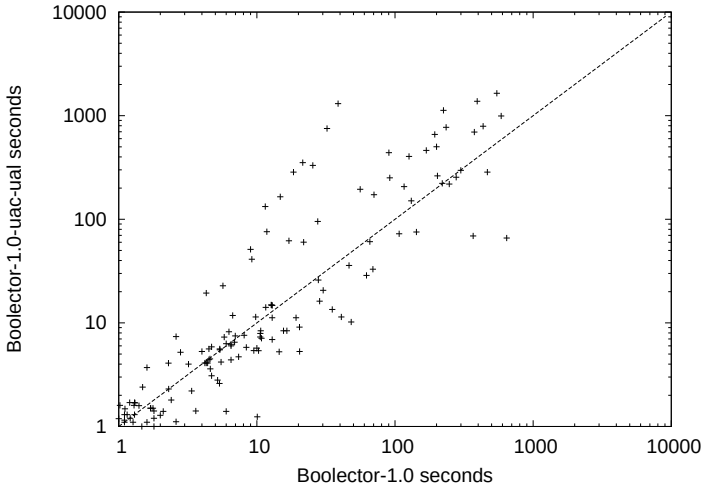

Fig. 6. Boolector (x-axis) vs. Boolector with class under-approximation and local refinement strategy (y-axis). Benchmarks are from `QF_AUFBV` and are all unsatisfiable.

be implemented on the CNF layer which enables further optimizations like early unsat termination. Finally, we showed how under-approximation techniques for bit-vectors can be combined with over-approximation techniques for arrays and evaluated the effectiveness of our approach on benchmarks from the SMT library.

Formula approximation techniques help to handle complex and hard formulas. Under-approximation techniques speed up decision procedures in the context of falsification and generate small models with restricted domains.

References

1. C. Barrett, M. Deters, A. Oliveras, and A. Stump. SMT-Comp, 2008. `www.smtcomp.org`.
2. C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In *Proc. CAV'02*. Springer, 2002.
3. C. Barrett, S. Ranise, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, June 2008.
4. A. Bradley and Z. Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification*. Springer, 2007.
5. R. Brummayer and A. Biere. Lemmas on Demand for the Extensional Theory of Arrays. In *Proc. SMT'08*. ACM, 2008.
6. R. Brummayer and A. Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In *Proc. TACAS'09*. Springer, 2009.
7. R. Brummayer, A. Biere, and F. Lonsing. BTOR: Bit-Precise Modelling of Word-Level Problems for Model Checking. In *Proc. BPR'08*. ACM, 2008.
8. R.E. Bryant, D. Kroening, J. Ouaknine, S. Seshia, O. Strichman, and B. Brady. Deciding Bit-Vector Arithmetic with Abstraction. *Software Tools for Technology Transfer (STTT)* , 2009.
9. K. Claessen and N. Sörensson. New Techniques that Improve MACE-style Finite Model Finding. In *CADE-19, Workshop W4, Model Computation – Principles, Algorithms, Applications*, 2003.
10. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM (JACM)*, 2003.
11. L. de Moura and H. Rueß. Lemmas on Demand for Satisfiability Solvers. In *Proc. SAT'02*. Springer, 2002.
12. C. Flanagan, R. Joshi, and J. Saxe. Theorem Proving Using Lazy Proof Explication. In *Proc. CAV'03*. Springer, 2003.
13. V. Ganesh. *Decision Procedures for Bit-Vectors, Arrays and Integers*. PhD thesis, Computer Science Department, Stanford University, 2007.
14. N. He and M. Hsiao. Bounded Model Checking of Embedded Software in Wireless Cognitive Radio Systems. In *Proc. ICCD*. IEEE, 2007.
15. N. He and M. Hsiao. A new Testability Guided Abstraction to Solving Bit-Vector Formula. In *Proc. BPR'08*. ACM, 2008.
16. D. Kroening and O. Strichman. *Decision Procedures: An algorithmic Point of View*. Springer, 2008.
17. R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 3, 2007.