Using Term Rewriting to Solve Bit-Vector Arithmetic Problems

IAGO ABAL RIVAS
Universidade do Minho

We analyze the implementation of an effective simplification algorithm for the theory of fixed-size bit-vectors. The objective is to exploit algebraic properties concerning the domain of bit-vectors to rewrite a given problem into an equisatisfiable but simpler one. We believe that this goal could be accomplished by a term rewriting system and an appropriate set of rewrite rules, with the support of some other simplification techniques. The study of the state of the art seems to support our thesis, evidencing that term rewriting is decisive for the performance of decision procedures for bit-vector arithmetic.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving

General Terms: Algorithms, Verification

Additional Key Words and Phrases: Bit-Vectors, Decision Procedures, SMT, Term Rewriting, Automated Theorem Proving

1. INTRODUCTION

Satisfiability Modulo Theories (SMT) [de Moura et al. 2007] solvers are a recent technology whose popularity has increased very significantly during the last years, and are now being used for almost any application which involves automated theorem proving: formal verification, scheduling, bug-finding, symbolic execution, test-case generation, etc. Satisfiability is the problem of determining if a formula has a model, that is, an assignment of its variables that makes it *true*. The problem of satisfiability for boolean formulas is often abbreviated as SAT, and algorithms to decide this problem are called SAT solvers. SMT solvers combine SAT solving with decision procedures allowing to decide the satisfiability of formulas expressed in a combination of first-order theories. Some commonly supported theories are linear integer arithmetic, fixed-size bit-vectors, and extensional arrays, among others. Despite SAT has plenty of practical applications, many interesting problems require or benefit from the added expressiveness of these SMT-supported theories. In contrast with traditional theorem provers, SMT solvers gained a lot of interest not only in academia but also in industry because they are fully automatic and, what is more, they provide counter-examples whenever a formula cannot be proved valid.

Among the many theories that are supported by SMT solvers, the theory of finite-precision bit-vector arithmetic is one of the most useful, for both hardware and software systems verification. For instance, modern digital design is usually ac-

E-mail: pg16305@alunos.uminho.pt
Departamento de Informática, Universidade do Minho Master PreThesis Reports 2012

complished by using hardware description languages (HDLs) like Verilog or VHDL; both are examples of register-transfer level (RTL) design, where systems are described on the world level. Typically a circuit implementation is verified by checking its equivalence with a reference implementation (specification), using a decision procedure for bit-vector arithmetic. On the software side, bit-vectors are useful to model finite-precision integer arithmetic and bit-wise operations. Program analysis tools generate queries about the feasibility of individual program paths, which tend to be a conjunction of atomic constraints that have to be solved. In combination with a theory of arrays, they can also be used to model the main memory of a computer (e.g. to reason about pointer operations) or a cache. Moreover, there are specific domains such as cryptography or digital signal processing (DSP) in which solutions are naturally expressed as bit-vector programs. Cryptol is an example of a domain-specific language (DSL) and toolset for cryptography [Browning 2010], developed by Galois Inc. It supports equivalence checking of cryptographic algorithms with SMT solvers, by modeling Cryptol specifications as fixed-size bit-vector formulas [Erkök and Matthews 2008].

More precisely, the theory of fixed-size bit-vectors is an equational theory over finite non-empty strings over $\{0, 1\}$, whose length is known and fixed *a priori*. The basic operations that are being considered here are *concatenation*, *extraction of bit strings*, *boolean bit-wise operations* (negation, conjunction, disjunction, exclusive-or, shifts and rotates), *modular arithmetic operations* (addition, multiplication, division, modulo and exponentiation), *boolean comparison operators* (equals, less-than and equals-or-less-than), and *if-then-else expressions*. Other operations like subtraction can be defined in terms of these basic ones. Note that formulas of this theory are just one-sized bit-vector terms.

The decision problem for this theory is known to be NP-hard. There are many approaches for deciding this theory, or subsets of it, as will be discussed later; but the conceptually simplest way to implement a bit-vector decision procedure consists in translating bit-vector problems into pure propositional SAT problems, also known as *bit-blasting*. In fact, most current approaches to decide bit-vector problems are SAT-based; that is, they rely on bit-blasting at some point. Unfortunately bit-blasting can be computationally too expensive in practice, due to the clause explosion produced when bit-blasting operators like multiplication or division. For instance, a 32-bit division circuit is represented as ten thousands of clauses at the pure bit level. SAT-based approaches try to overtake this problem in different ways – many real-world verification problems would otherwise be intractable in practice.

The basic problem of bit-blasting is that it loses the high-level structure present in the original decision problem. For example, for some bit-vector x of size n, the equality $x^2 - 1 = (x + 1) \times (x - 1)$ is a simple consequence of distributivity and associativity laws; but even for small values of n the representation of this formula at the bit-level is so huge that it is intractable by current SAT solvers. The point here is that one can use algebraic properties concerning the domain of bit-vectors to rewrite formulas into equisatisfiable, but computationally less hard, formulas; for some instances, the formulas can even be fully decided at the word-level. For example, a term rewriting system may reduce the above formula as follows:

$$
\begin{aligned}
x^2 - 1 &= (x+1) \times (x-1) \\
&\equiv \{\text{distributivity} \times 3\} \\
x^2 - 1 &= (x^2 + x) + (-x - 1) \\
&\equiv \{\text{associativity}\} \\
x^2 - 1 &= x^2 + x - x - 1 \\
&\equiv \{\text{inverse and identity element}\} \\
x^2 - 1 &= x^2 - 1 \\
&\equiv \{\text{reflexivity}\} \\
&\mathit{true}
\end{aligned}
$$

Most current SMT solver architectures include a simplification phase that performs some word-level rewriting on the input formula. Nevertheless current SMT solvers tend to be very conservative, implementing only simple and relatively inexpensive simplification steps that typically only help in the most trivial cases. Actually, most of them cannot handle the above example. But it is not clear if such preprocessing must be inexpensive: even a potentially expensive technique may be worthwhile if it drastically reduces solving time [Franzen 2010]. Indeed, our previous experience on improving equivalence checking for public-key cryptography has shown how bad current SMT solvers handle problems involving modular arithmetic, and how just adding a few tens of rules to simplify modulo operations make a great difference. Thus, the thesis that we intend to test in the present work is that *it is possible to design a practical rewriting-based simplification algorithm for the theory of fixed-size bit-vectors, that will overcome the preprocessing steps of state-of-the-art SMT solvers by handling a wider range of real-world problems.*

The rest of this paper is organized as follows. Section 2 covers the main objectives and challenges of this work. Next, Section 3 provides a brief introduction to state of the art decision procedures for bit-vector arithmetic. Section 4 broadly describes the research and work methodology to be followed. Finally, Section 5 gives the conclusions that were drawn from the study of the subject, and provides arguments that support the feasibility of this thesis.

2. OBJECTIVES

Our main objective is the design and implementation of an innovative, effective, and efficient simplification algorithm for the theory of fixed-size bit-vectors; most likely, in combination with uninterpreted functions. The theory of uninterpreted functions is particularly useful for achieving compositional verifications: it allows to replace defined functions with uninterpreted symbols together with a set of axioms describing their relevant properties.

This goal is supposed to be achieved by designing a highly-customized term rewriting system for this purpose, but concrete details are still unclear – further research is needed. However, previous experience in this field has already demonstrated the value of *associative-commutative (AC) rewriting* (i.e. rewriting modulo associativity and commutativity laws) as well as of *term graph rewriting* [Kennaway

et al. 1993], so these are techniques that have to be seriously considered. In contrast with state-of-the-art SMT solvers, this algorithm is intended to be more aggressive, applying possibly expensive strategies whenever required to further simplify a formula. Although it would be valuable to provide a bound for the computational cost of this algorithm, that is not the focus of this work: our main interest is in the observable overall behavior of the algorithm.

Additionally, this preprocessing step is going to be packed as a stand-alone simplifier application supporting the SMT-LIB standard input format [Barrett et al. 2008], in order to make integration with other tools easy. The application should take a SMT-LIB benchmark as input and produce a simplified SMT-LIB benchmark as output. Under the hood the input benchmark will first be converted into the internal format used by the simplifier, then simplified, and finally the result will be converted back to the SMT-LIB format.

One of the biggest challenges of this work is to design such a simplification method while keeping it general. Generality is hard to quantify, and any way to do this could be subject to criticism. In this work it will be measured by testing against a large and heterogeneous set of (mostly) real-world case studies. This set of case studies has to contain a significant amount of hard problems for current SMT solvers. The goal is to provide evidence of the usefulness of the simplification algorithm to be developed, by demonstrating that it is able to take a difficult problem as input and produce in reasonable time an easy, equisatisfiable problem as output.

As a secondary objective, this algorithm should be *extensible* to some degree, for instance with the purpose of fine-tuning it for some specific domain. Indeed, some partners of Galois Inc. would be interested in a specialized instance for equivalence checking of public-key cryptographic algorithms.

3. STATE OF THE ART

Decision procedures for bit-vector arithmetic can be broadly classified into three categories. First, approaches that decide a bit-vector problem by converting it into SAT are introduced in Section 3.1. Another approach is to convert bit-vector problems into the integer arithmetic domain, as explained in Section 3.2. Finally, approaches in the third category rely on *canonizing* bit-vector terms and are the subject of Section 3.3. Section 3.4 is dedicated to preprocessing techniques for the theory of bit-vectors, which are the focus of this study. These techniques, despite being used almost exclusively by SAT-based decision procedures, do not depend on any particular approach to decide bit-vector arithmetic.

3.1 SAT-based approaches

Most state-of-the-art decision procedures for bit-vector arithmetic are SAT-based. Recall that the fundamental idea behind this category of decision procedures is to decide bit-vector formulas by *bit-blasting* them, and then using a SAT-solver to decide the resulting boolean formulas. Bit-blasting can be broadly defined as the process of representing any bit-vector operation by a series of operations on individual bits. This very basic strategy to decide bit-vector arithmetic has two major (interrelated) drawbacks: (i) the high-level structure of the original problem is destroyed, and (ii) bit-blasting a formula may blow up, resulting in a SAT problem that is too hard to be solved.

Nevertheless, SAT-based methods for deciding bit-vector theories are still the most efficient and most general approaches [Ganesh et al. 2006]. This is primarily due to the fact that the performance of SAT solvers has been consistently increasing during the last years, and to the existence of effective translations from the bit-vector to the boolean domain. Furthermore, there is an increasing effort in the development of techniques that exploit the high-level structure of bit-vector arithmetic problems to make them easier to solve [Ganesh et al. 2006; Brummayer 2009; Franzen 2010]. Combinations of these techniques are now applied by most current SMT solvers as preprocessing steps that simplify input formulas before bit-blasting [Brummayer and Biere. 2009; Jha et al. 2009a; De Moura and Bjørner 2008]. In fact, the decision procedures of the six participants in the quantifier-free bit-vector (QF_BV) category of the *2011 edition of the Satisfiability Modulo Theories Competition (SMT-COMP)*[1] are all SAT-based, and employ different preprocessing techniques. As said above, such techniques are going to be the focus of Section 3.4.

Under-approximation techniques. The *eager* SAT-based approach decides a bit-vector formula by translating it into pure propositional SAT and then just calling a SAT solver. However, there exist *lazy* versions of this method that use automatic abstraction-refinement, combining under-approximation and over-approximation techniques. The UCLID verification system (predecessor of the Beaver SMT solver) has pioneered the use of these techniques in bit-vector logic [Bryant et al. 2007]. This abstraction-refinement is in the spirit of the *Counter Example Guided Abstraction Refinement Framework* (CEGAR) [Clarke et al. 2003], very popular within the *software model checking* community. Under-approximation techniques are currently being used in some state-of-the-art SMT solvers such as Boolector [Brummayer and Biere. 2009].

The basic idea of under-approximation is to restrict individual bits of bit-vectors. For each bit-vector variable v_i of width w_i, the m_i most significant bits are restricted (e.g. fixed to be 0), whilst the remaining n_i ($0 \leq n_i \leq w_i$) least significant bits remain variable. The parameter n_i can be called the *effective bit-with* or the *encoding size* of the variable v_i for obvious reasons: note that to encode such an under-approximation of v_i only n_i boolean variables are required. Not only do such domain restrictions typically lead to a smaller search space and a speed-up in solving satisfiable formulas, but they also produce "smaller" models (i.e. the values assigned to variables are smaller). Small models are beneficial for diagnosis, specially when the model is analyzed by users for debugging. Furthermore, in the area of test case generation, small models lead to test cases with reduced test data size.

For instance, the abstraction-based method described in [Bryant et al. 2007] works as follows. For a given formula ϕ, a small initial *encoding size* n_i is assigned for each variable v_i in ϕ. Then the procedure generates an under-approximation of ϕ, denoted $\underline{\phi}$, by restricting the values of each v_i to range over a set of cardinality 2^{n_i}. A boolean formula β is generated by bit-blasting $\underline{\phi}$, and it is handed to a SAT solver that must provide an unsatisfiable core when a formula is determined to be

[1] `http://www.smtcomp.org/2011/`

unsatisfiable. If the SAT solver reports that β is satisfiable, then the satisfying assignment is also a model of ϕ, and the procedure terminates. Otherwise, if β is unsatisfiable, the unsatisfiable core $\mathcal{C}$ provided by the SAT solver is used to build an over-approximating abstraction $\overline{\phi}$ of ϕ, which is typically much smaller than ϕ. The key property of $\overline{\phi}$ is that its translation into SAT, using the same n_i that were used for ϕ, would also result in an unsatisfiable formula. The satisfiability of $\overline{\phi}$ is checked and, if it is unsatisfiable, we can conclude that so is ϕ. On the other hand, if $\overline{\phi}$ is satisfiable we cannot conclude anything, but it must be the case that one or more variables v_i were assigned a value that is not representable with n_i boolean variables (otherwise it cannot be satisfiable, recall the key property of $\overline{\phi}$). Such an assignment is used to increase the encoding size for the relevant variables, and the under-approximation step is then repeated.

3.2 Integer-based approaches

Traditional decision procedures use integers in order to approximate bit-vector semantics. These methods can be considerably more efficient than SAT-based approaches for some problems like datapath verification in RTL design [Brinkmann and Drechsler 2002; Parthasarathy et al. 2004], but they are very inefficient in the presence of a large boolean component [Bruttomesso and Sharygina 2009]. These approaches also destroy the original structure of the problem, but not to the same extent as bit-blasting. Bit-vectors are at least kept as individual entities, by encoding words as integers.

Such integer abstractions may not be able to detect important corner cases such as overflows. What is more, and most importantly, reasoning about non-linear operators is undecidable, so these approaches are not as general as the ones based on SAT, which can handle arbitrary bit-vector operations. Some methods have been developed that handle non-linear arithmetic [Babić and Musuvathi 2005], but they are limited and hard to extend to new operators.

3.3 Canonizer-based approaches

Earlier work on deciding bit-vector arithmetic relied on *canonizing* bit-vector terms, thus exploiting the high-level structure of the input problem. Many of these methods [Cyrluk et al. 1997; Barrett et al. 1998] centered on the use of a Shostak-style[2] approach, based on the use of a canonizer and a solver for the theory to be supported. The Stanford Validity Checker (SVC) is one example of that [Barrett et al. 1998]. Alternatively, efficient canonical data structures like Binary Moment Diagrams (BMDs) have been used to represent bit-vector terms [Bryant and Chen 1995].

A *canonizer* is an algorithm that transforms semantically equivalent terms into a unique representation, which we refer to as a *canonical form*. Commonly this unique representation is chosen to be the simplest one, accordingly to some simplicity criteria. Unfortunately, even for a relatively simple language of bit-vector expressions, computing a canonical form for a bit-vector term is in itself NP-hard. For a richer language, such as the one considered here, it is impractical for most

[2]Shostak's methods are one of two main approaches that can be used to combine decision procedures for different theories, the other being based on Nelson-Oppen's methods.

cases [Ganesh et al. 2006]. For instance, SVC uses a smart approach not requiring all terms to be canonized: a class of terms which are called *atomic* are canonized by (polynomial-time) rule-based transformations, whilst the canonization of *non-atomic* terms is delayed as much as possible. Canonizing *non-atomic* terms requires case-splitting, but the incremental approach followed avoid this unless it is absolutely necessary.

These approaches are very elegant, but are usually restricted to a subset of bit-vector arithmetic comprising concatenation, extraction, and linear equations over bit-vectors. Moreover, extending the theory with new operators can be rather difficult, since a new canonical form and solver algorithm have to be designed at each time.

3.4 Preprocessing techniques

The purpose of preprocessing techniques is to exploit the algebraic structure of bit-vector formulas, in order to rewrite them into equisatisfiable but hopefully less hard formulas. For some instances, the input formula can be fully decided by these simplification techniques, thus avoiding bit-blasting at all. Most SMT developers agree that such preprocessing steps are very important for the theory of fixed-size bit-vectors [de Moura 2011]. What is more, given that most state-of-the-art SMT solvers are based on bit-blasting, the actual differences between them lie in the simplification steps employed before bit-blasting [Jha et al. 2009b].

Unfortunately, it is hard to find documentation about the preprocessing steps used by modern SMT solvers [de Moura 2011]: sometimes because they are just little tricks, sometimes because they only work for a particular SMT solver. Nevertheless, there is a small set of general and well-known simplification techniques that are being used by several current SMT solvers; these are presented below.

Normalization. Normalization can be seen as a practical and cheaper way to simplify bit-vector expressions by exploiting their algebraic structure. Ideally one would like to compute canonical forms for bit-vector terms but, as stated earlier, that is impractical for several reasons.

Normalization broadly consists in performing repeated simplifications over a term, by exhaustively applying a set of rewrite rules. A *rewrite rule*, written $s \to t$, denotes that terms *matching* the pattern s are rewritten as specified by t. For example, the rewrite rule $0 + x \to x$ can be used to simplify the term $0 + 1$ into 1 (where $x \mapsto 1$), or the term $0 + 2y$ into $2y$ (where $x \mapsto 2y$). A set of rewrite rules constitutes a *term rewriting system* (TRS). For our purpose, these rules are intended to exploit the algebraic properties concerning the domain of bit-vector arithmetic. When we repeatedly apply the set of rules of a TRS over a term t to a fix-point, obtaining a term t' that cannot be further simplified, we say that t' is the *normal form* of t. Note that terms sharing the same normal form are semantically equivalent, but the opposite is not always true – otherwise we would be canonizing. In practice, normalization allows significant simplification of bit-vector terms while still being reasonably efficient (polynomial-time), in contrast with canonization which is NP-hard.

Normalization is a very important preprocessing technique that is used by the majority of modern SMT solvers, including Z3 [De Moura and Bjørner 2008], Boolec-

tor [Brummayer and Biere. 2009], Beaver [Jha et al. 2009b], MathSAT [Franzen 2010], etc. In fact, some SMT solvers apply normalization not only at the word-level but also at the bit-level, after the simplified bit-vector problem has been converted into a SAT problem. This allows further simplification of the problem's boolean representation before handing it to a SAT solver [Jha et al. 2009b].

Constant propagation. The goal of constant propagation is to discover values that are constant and to propagate these constant values as far forward as possible. For instance, given an equality $t = c$ where t is an arbitrary term and c is a constant, it replaces every occurrence of t by c. For example, the formula $t = 0 \wedge \phi[t]$ can be transformed into $t = 0 \wedge \phi[0]$ by performing constant propagation. Hopefully, $\phi[0]$ will be further simplified.

However, note that once we perform constant propagation we cannot simply ignore the equality $t = c$, otherwise the resulting formula could not be equisatisfiable. The following example illustrates why. Consider the formula $(v\#w = 1100) \wedge (v\#w > 0001) \wedge (v = 00)$, where $\#$ is bit-vector concatenation and v, w are 2-sized bit-vector variables. This formula is unsatisfiable because the first equation implies $v = 11$, which conflicts with the last one. But, if we perform constant propagation and then ignore the equation $v\#w = 1100$, we will end (after simplifying) with the formula $v = 00$, which is obviously satisfiable.

Substitution / Variable elimination. It is very common for real-world problems to contain a number of definitions together with a formula that uses these definitions. Such problems are of the form $(\bigwedge_i v_i = t_i) \wedge \phi$, where v_i is a variable, t_i an arbitrary bit-vector term, and ϕ a bit-vector formula.

These definitions would be encoded into SAT as comparators which may be entirely unnecessary, and which could cause significant overhead in solving. So it is very common that solvers perform substitution/variable elimination, usually followed by a series of simplifications, to hopefully generate a more compact and simpler formula to solve. For instance, $v = t \wedge \phi$ can be rewritten into the equisatisfiable formula $\phi[v \mapsto t]$ if $v \notin \mathcal{V}ar(t)$.

However, variable elimination can also be counterproductive in some circumstances, as when it makes the problem harder to solve or grow too much, without enabling any further simplifications. Therefore, this technique is usually applied in combination with some heuristic to guess when a substitution is going to be useful.

Linear solving. The goal of using linear solvers is to eliminate as many variables as possible to reduce the size of the problem that is eventually received by the SAT solver. In addition, it may enable many other simplifications (such as variable elimination), and can solve purely linear problems outright.

The STP decision procedure, for instance, uses a custom linear solver for bit-vector equations to simplify the input problem [Ganesh and Dill 2007]. In some cases the solver cannot solve an entire variable, but it can solve for some of the lower bits of the variable, so these bits will not appear as boolean variables when the problem is bit-blasted. Non-linear and word-level terms (extractions, concatenations, etc.) appearing in linear equations are treated as bit-vector variables. Note that it is possible to solve linear equations involving extractions and concatenations, but the problem becomes NP-complete. Z3 is another example of a solver

that makes use of linear solving, in particular, it performs Gaussian elimination for bit-vectors [de Moura 2011].

Elimination of unconstrained terms. *Unconstrained variables* are inputs that are referenced only once in a formula, which means that they are not affected by further side conditions, and so their assignment value may be adjusted to make the formula satisfiable. If a term contains a set of unconstrained variables, this term is itself unconstrained iff for any value of the variables in the term which are not unconstrained, it is possible to choose values for the unconstrained ones so that the term evaluates to any desired value. The point of this is that an *unconstrained term* can simply be replaced by a variable, resulting in a simpler but equisatisfiable formula. For example, consider the following formula. Let t be an arbitrary bit-vector term, and let v_1, v_2 and v_3 be bit-vector variables.

$$v_3 + t = v_1 \& v_2$$

Variables v_1 and v_2 are unconstrained, and so is $v_1 \& v_2$. Note that we can trivially make $v_1 \& v_2$ evaluate to an arbitrary constant c, just assigning c to both variables ($c \& c \equiv c$). Hence, we can simply replace $v_1 \& v_2$ by a fresh bit-vector variable v_4 itself. In this way, we have obtained a simpler equisatisfiable formula.

$$v_3 + t = v_4$$

A similar argument can be applied to the left side of the equality. Independently of the assignment of t, we can freely adjust the value of v_3 in order to satisfy the formula. Therefore we can also replace $v_3 + t$ by a fresh bit-vector variable v_5.

$$v_5 = v_4$$

Note that t, which may be an arbitrary complex bit-vector term, has been completely eliminated. It is possible to define propagation rules for other arithmetic and boolean operators, equality, if-then-else expressions and bit-vector predicates [Bruttomesso 2008]. For instance, in our example, one can finally replace the unconstrained term $v_4 = v_5$ by a fresh boolean variable b to obtain a trivially satisfiable formula.

4. METHODOLOGY

First, it is needed to clarify many design decisions concerning the term rewriting system that is going to be developed. For instance, it is not clear if the most convenient is to make use of a general framework like Maude [Clavel et al. 2003] to implement term rewriting systems, or if it would be preferable to implement one from scratch. The former makes possible effective and rapid implementation of relatively simple systems, whilst developing your own rewriting system allows for a high-degree of control and fine-tuning. The question is whether developing such a customized rewriting system is worth the effort. A further study of the topic is required to answer such pending questions, from the foundations of term rewriting [Baader and Nipkow 1998] to state-of-the-art techniques and tools [Eker 2003; Clavel et al. 2003]. Additionally, relevant figures in both term rewriting and decision procedures for bit-vector arithmetic fields are going to be contacted for feedback.

Next, once the most important design details have been clarified, such a (term rewriting based) simplification algorithm is going to be developed. Given the characteristics of this project it looks like an agile software development process [Larman 2004] fits perfectly, so it is the approach that is going to be followed. The code is to be developed in small increments from two to six weeks length, with minimal long-term planning. These development increments will be defined by a set of objectives to be achieved and a lightweight plan, that will be revised and adjusted in a weekly basis. Each iteration is intended to deliver a working prototype for demonstration and testing. Hence, case studies are going to be collected gradually during the whole development process, and they will be used to validate such intermediate prototypes. Prototype validation is expected to provide valuable feedback for each next software increment. A simple issue tracking tool and weekly meetings with the supervisor should help to monitor the progress of the project, and a wiki is to be used to document the work and collect ideas.

When the final version of such simplification algorithm is ready, it will be time to draw conclusions. First, the set of case studies must be substantially extended to make sure that the simplifications are general and not ad-hoc for a specific set of concrete problems. Then, these case studies will be used to measure the performance of the developed algorithm with respect to, and in combination with, current decision procedures for bit-vector arithmetic. At this time, it will become clear if we succeeded in proving this thesis or not. But even if we fail to prove it, it may simply be the case that a wrong direction was taken, and it would be valuable to use the experience acquired during the project to analyze and suggest other promising roads.

5. CONCLUSIONS

Decision procedures for fixed-size bit-vector arithmetic are still the subject of active research. That is justified because they are of great importance for verification of both hardware and software systems, and particularly in some specific domains such as cryptography. Anyone who has used SMT solvers in practice will agree that there is definitely a lot of room for improvement. Significant research has been done since the '90s to improve the very basic approach of deciding bit-vector arithmetic through bit-blasting, but the fact is that the winner of the *SMT-COMP QF_BV 2011* is Z3: a SMT solver developed by Microsoft Research that relies on traditional bit-blasting, but incorporates a sophisticated preprocessing phase in comparison with other SMT solvers. That seems to support the argument that an aggressive preprocessing for exploiting algebraic properties of bit-vectors would have a great impact on performance.

As stated in Section 3, the most practical approach to exploit such high-level structure is *normalization*. But even though this is a powerful and widely used technique in decision procedures, previous experience suggests that there has been no real effort on discovering a set of rewrite rules to cover a wide range of bit-vector problems. For instance, most normalizations steps for bit-vector arithmetic employ only a few hundred rewrite rules. In comparison, as part of a small MSc course project we have ourselves developed a normalizer with about two hundred rewrite rules, a half of them just concerning modular arithmetic. Thus, we plan

to start focusing our work on discovering such a large set of rules, and possibly using other preprocessing techniques such as *unconstrained terms elimination* to support further simplifications. Just the effort to discover such a set of rules, and the new useful rules that will hopefully show up, will in our view already be a great contribution by itself.

REFERENCES

Baader, F. and Nipkow, T. 1998. *Term Rewriting and All That.* Cambridge University Press.

Babić, D. and Hutter, F. 2008. Spear theorem prover. In *SAT'08: Proceedings of the SAT 2008 Race.*

Babić, D. and Musuvathi, M. 2005. Modular Arithmetic Decision Procedure. Tech. Rep. TR-2005-114, Microsoft Research Redmond.

Barrett, C., Ranise, S., Stump, A., and Tinelli, C. 2008. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`.

Barrett, C. W., Dill, D. L., and Levitt, J. R. 1998. A decision procedure for bit-vector arithmetic. In *Proceedings of the 35th annual Design Automation Conference.* DAC '98. ACM, New York, NY, USA, 522–527.

Brinkmann, R. and Drechsler, R. 2002. RTL-datapath verification using integer linear programming. In *Proceedings of the 2002 Asia and South Pacific Design Automation Conference.* ASP-DAC '02. IEEE Computer Society, Washington, DC, USA, 741–.

Browning, S. 2010. Cryptol, a DSL for cryptographic algorithms. In *ACM SIGPLAN Commercial Users of Functional Programming.* CUFP '10. ACM, New York, NY, USA, 9:1–9:1.

Brummayer, R. 2009. Efficient SMT solving for bit-vectors and the extensional theory of arrays. Ph.D. thesis, Johannes Kepler University.

Brummayer, R. and Biere., A. 2009. Boolector: An efficient SMT solver for bit-vectors and arrays. In *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09).* Vol. 5505. Springer, 174–177.

Brummayer, R. and Biere, A. 2009. Effective bit-width and under-approximation. In *Computer Aided Systems Theory - EUROCAST 2009, 12th International Conference, Las Palmas de Gran Canaria, Spain, February 15-20, 2009, Revised Selected Papers.* Lecture Notes in Computer Science, vol. 5717. Springer, 304–311.

Bruttomesso, R. 2008. RTL verification: From SAT to SMT(BV). Ph.D. thesis, University of Trento. Available at `http://www.inf.unisi.ch/postdoc/bruttomesso`.

Bruttomesso, R. and Sharygina, N. 2009. A scalable decision procedure for fixed-width bit-vectors. In *Proceedings of the 2009 International Conference on Computer-Aided Design.* ICCAD '09. ACM, New York, NY, USA, 13–20.

Bryant, R. E. and Chen, Y.-A. 1995. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference.* DAC '95. ACM, New York, NY, USA, 535–541.

Bryant, R. E., Kroening, D., Ouaknine, J., Seshia, S. A., Strichman, O., and Brady, B. 2007. Deciding bit-vector arithmetic with abstraction. In *Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems.* TACAS'07. Springer-Verlag, Berlin, Heidelberg, 358–372.

Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM 50*, 752–794.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. 2003. The Maude 2.0 system. In *Rewriting Techniques and Applications (RTA 2003)*, R. Nieuwenhuis, Ed. Number 2706 in Lecture Notes in Computer Science. Springer-Verlag, 76–87.

Cyrluk, D., Möller, M. O., and Ruess, H. 1997. An efficient decision procedure for the theory of fixed-sized bit-vectors. In *Proceedings of the 9th International Conference on Computer Aided Verification.* CAV '97. Springer-Verlag, London, UK, 60–71.

de Moura, L. 2011. A summary of the pre-processing techniques performed by Z3. http://stackoverflow.com/questions/8273033/use-of-term-rewriting-in-decision-procedures-for-bit-vector-arithmetic. In response to a question in StackOverflow.com.

De Moura, L. and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Proceedings of the Theory and practice of software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'08/ETAPS'08. Springer-Verlag, Berlin, Heidelberg, 337–340.

de Moura, L., Dutertre, B., and Shankar, N. 2007. A tutorial on satisfiability modulo theories. In *Proceedings of the 19th International Conference on Computer Aided Verification*. CAV'07. Springer-Verlag, Berlin, Heidelberg, 20–36.

Eker, S. 2003. Associative-commutative rewriting on large terms. In *Rewriting Techniques and Applications (RTA 2003)*, R. Nieuwenhuis, Ed. Number 2706 in Lecture Notes in Computer Science. Springer-Verlag, 14–29.

Erkök, L. and Matthews, J. 2008. Pragmatic equivalence and safety checking in Cryptol. In *Proceedings of the 3rd workshop on Programming languages meets program verification*. PLPV '09. ACM, New York, NY, USA, 73–82.

Franzen, A. 2010. Efficient solving of the satisfiability modulo bit-vectors problem and some extensions to SMT. Ph.D. thesis, University of Trento.

Ganesh, V., Berezin, S., and Dill, D. L. 2006. A decision procedure for fixed-width bit-vectors. Tech. Rep. CSTR 2007-06, Department of Computer Science, Stanford University.

Ganesh, V. and Dill, D. L. 2007. A decision procedure for bit-vectors and arrays. In *Proceedings of the 19th international conference on Computer aided verification*. CAV'07. Springer-Verlag, Berlin, Heidelberg, 519–531.

Jha, S., Limaye, R., and Seshia, S. A. 2009a. Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. In *21st International Conference on Computer-Aided Verification*. 668–674.

Jha, S. K., Limaye, R. S., and Seshia, S. A. 2009b. Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. Tech. Rep. UCB/EECS-2009-95, EECS Department, University of California, Berkeley. Jun.

Kennaway, J. R., Klop, J. W., Sleep, M. R., and de Vries, F. J. 1993. *An introduction to term graph rewriting*. John Wiley and Sons Ltd., Chichester, UK, 1–13.

Larman, C. 2004. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley.

Parthasarathy, G., Iyer, M. K., Cheng, K.-T., and Wang, L.-C. 2004. An efficient finite-domain constraint solver for circuits. In *Proceedings of the 41st annual Design Automation Conference*. DAC '04. ACM, New York, NY, USA, 212–217.