SMT SOLVING FOR THE THEORY OF BIT-VECTORS

MARTIN JONÁŠ

PHD THESIS PROPOSAL

September 2016

Faculty of Informatics
Masaryk University

SUPERVISOR
doc. RNDr. Jan Strejček, Ph.D.

CONTENTS

1 INTRODUCTION

During the last decades, the area of *satisfiability modulo theories* (SMT) has undergone steep development in both theory and practice. Achieved advances of SMT solving opened new research directions in program analysis and verification, where SMT solvers are now seen as standard tools.

The task for an SMT solver is to decide for a given first-order formula in a given first-order theory whether the formula is satisfiable. Usually, if the formula is satisfiable, the SMT solver also has the ability to provide its model. Modern SMT solvers support wide range of different first-order theories – for example theories of integers, real numbers, floating-point numbers, arrays, strings, inductively defined data types, bit-vectors, and various combinations and fragments of formulas over these theories [BFT15; ZZG13; RB16]. From the software analysis and verification point of view, a particularly important of these theories is the theory of bit-vectors, which can be used to describe properties of computer programs, since programs usually use data-types of bounded size instead of mathematical integers. Additionally, operations over these bounded data-types naturally correspond to operations of the bit-vector theory.

The benefit of describing properties of programs by bit-vector formulas is twofold. Formulas in the bit-vector theory allow to model the program's behavior precisely including possible arithmetic overflows and underflows. Furthermore, in contrast to the theory of integers, the satisfiability of the bit-vector theory is decidable even if the multiplication is allowed.

Therefore, quantifier-free bit-vector formulas are used in tools for symbolic execution, bounded model checking, analysis of hardware circuits, static analysis, or test generation [Cad+08; CDE08; Lei10; CFM12]. Most of the current SMT solvers for the quantifier-free bit-vector formulas eagerly or lazily translate the formula to the propositional logic (*bit-blasting*) and use an efficient SAT solver to decide its satisfiability. Therefore, the efficiency of most of the bit-vector SMT solvers is tightly connected to the efficiency of the SAT solvers. Plenty of solvers for the quantifier-free bit-vector formulas exist: Beaver [JLS09], Boolector [NPB14], CVC4 [Bar+11], MathSAT5 [Cim+13], OpenSMT [Bru+10], Sonolar [PVL11], Spear [Hut+07], STP [GD07], UCLID [LS04], Yices [Dut14], and Z3 [MB08b].

In some cases, quantifier-free formulas are not succinct enough and using quantification is necessary to keep the size of the formula reasonable. Quantified bit-vector formulas arise naturally for example in applications that need to decide equality of two symbolically represented

states [BHB14], or in applications that generate loop invariants, ranking functions, or loop summaries [WHM13; KLW13]. However, the current SMT solvers' support of quantified bit-vector logic is modest – only CVC4, Yices, and Z3 officially support quantifiers in bit-vector formulas. Recently, quantifiers have been also implemented in an development version of Boolector [Boo]. All of these SMT solvers solve quantified bit-vector formulas by some variant of quantifier-instantiation and using a solver for quantifier-free formulas as an oracle.

In the last year, we have proposed a different approach. We have implemented a symbolic SMT solver Q3B, which is based on binary decision diagrams. We have shown that it can not only compete with state-of-the art SMT solvers, but outperforms them in many cases [JS16]. However, BDDs are not a silver bullet; the symbolic SMT solver fails on formulas containing non-trivial multiplication and complex arithmetic. Therefore, as the aim of my PhD study, I plan to develop a hybrid approach to solving quantified bit-vectors, which combines symbolic representation by BDDs with search techniques employed by existing state-of-the-art solvers. I also plan to continue developing the symbolic solver Q3B itself, namely improve its efficiency and add support for features as uninterpreted functions and arrays, which are important in the software verification.

The thesis proposal is organized as follows. Chapter 2 summarizes the state of the art and is divided into six sections. Section 2.1 introduces necessary background and notations from the propositional and first-order logic. Section 2.2 describes approaches to solving propositional satisfiability and Section 2.3 describes approaches to solving satisfiability modulo theories. Sections 2.4 and 2.5 are devoted to solving quantifier-free and quantified formulas over the theory of bit-vectors, respectively. Finally, Section 2.6 section describes results concerning the computational complexity of bit-vector logics. Chapter 3 describes results that we have achieved during the first two years of my PhD study. Chapter 4 presents the aim of the thesis and states plans for the remaining part of my PhD study.

2 STATE OF THE ART

2.1 PRELIMINARIES

This section introduces the notation used in the rest of this chapter. The exposition of the propositional logic is mainly based on the work of Nieuwenhuis et al. [NOT06]. The exposition of the first-order logic is based mainly on works of Enderton [End01] and of Barrett et al. [Bar+09]. Note that instead of following the approach of Enderton to the first-order theories, we follow the approach of Barrett et al. and define the theory as a set of first-order *structures*, rather than a set of first-order sentences.

2.1.1 *Propositional formulas, assignments, and satisfaction*

Let $\mathcal{P}$ be a fixed finite set of propositional variables. For every variable $x \in \mathcal{P}$ there are two literals – a *positive literal* x and a *negative literal* $\overline{x}$. For a given literal l, we define $\neg l$ as $\overline{x}$ if $l = x$ for a variable x and as x if $l = \overline{x}$ for a variable x. Literals l and $\neg l$ are called *complementary*. A *clause* is a finite disjunction of literals. The empty clause is denoted by $\bot$. A formula in the *conjunctive normal form* (CNF) is a finite conjunction of clauses. If convenient, we use idempotence and commutativity of disjunction and view clauses as sets of literals and therefore ignore the order and multiple occurrences of literals. Similarly, if convenient, we view CNF formulas as sets of clauses. For example, we write the formula $(x \vee \overline{y}) \wedge (\overline{x} \vee z)$ as the set $\{\{x, \overline{y}\}, \{\overline{x}, z\}\}$.

A *partial assignment* M is a set of literals that does not contain complementary literals, i.e. $\{x, \overline{x}\} \not\subseteq M$ for all $x \in \mathcal{P}$. A literal l is *true* in the assignment M if $l \in M$, *false* in M if $\neg l \in M$, and *undefined* otherwise. A literal is *defined* in M if it is true or false in M. We call an assignment M *total* over $\mathcal{P}$ if all literals of $\mathcal{P}$ are defined in M. A clause is *true* in M if at least one of its literals is true in M and a CNF formula is *true* in M if all of its clauses are true in M. A clause that is false for a given assignment M is called a *conflict clause* for M. If, for a given assignment, all literals of a clause are false except of one literal that is undefined, the clause is called *unit* in that assignment.

If a formula φ is true in M, we call M a *model* of φ and denote it as $M \models \varphi$. A formula is *satisfiable* if it has a model and *unsatisfiable* otherwise. If every model of a formula φ is also a model of a formula ψ, we say that the formula ψ is *entailed* by the formula φ and denote it

as $\varphi \models \psi$. Formulas φ and ψ are called *equisatisfiable* if φ is satisfiable precisely if ψ is satisfiable.

2.1.2 *First-order logic and theories*

A *signature* Σ consists of a set of *function symbols* Σ^f, a set of *predicate symbols* Σ^p, and for each of these symbols of a non-negative number called its *arity*. Given a signature Σ, Σ-*terms*, Σ-*atoms*, Σ-*literals*, Σ-*clauses*, Σ-*formulas*, and Σ-*formulas in CNF* are defined as usual. We are using the logic with equality, i.e. the set of Σ-atoms contains elements $t_1 = t_2$ for each pair of Σ-terms t_1, t_2. We call Σ-terms and Σ-formulas *ground* if they contain no free variables.

A Σ-*structure* $\mathcal{A}$ consists of a non-empty set A, the *universe* of the structure, and an assignment $(\_)^{\mathcal{A}}$, which to each variable x assigns an element $x^{\mathcal{A}} \in A$, to each function symbol $f \in \Sigma^f$ of arity n assigns a function $f^{\mathcal{A}} \colon A^n \to A$, and to each predicate symbol $P \in \Sigma^p$ of arity n assigns a relation $P^{\mathcal{A}} \subseteq A^n$. Given a Σ-structure $\mathcal{A}$, there exists a unique homomorphic extension of $(\_)^{\mathcal{A}}$ to Σ-terms, which to each Σ-term t assigns an element $t^{\mathcal{A}} \in A$. In turn, by using standard definitions, the assignment $(\_)^{\mathcal{A}}$ can be further extended to assign a truth value $\varphi^{\mathcal{A}} \in \{\texttt{true}, \texttt{false}\}$ to each Σ-formula φ. A Σ-structure $\mathcal{A}$ is said to *satisfy* a Σ-formula φ if $\varphi^{\mathcal{A}} = \texttt{true}$. In such case, the structure $\mathcal{A}$ is also called a *model* of the formula φ.

A Σ-*theory* $\mathcal{T}$ is a class of Σ-structures closed under isomorphisms and variable reassignment. A Σ-formula is said to be *satisfiable in the* Σ-*theory* $\mathcal{T}$, or $\mathcal{T}$-*satisfiable*, if there exists a Σ-structure $\mathcal{A} \in \mathcal{T}$ such that $\mathcal{A}$ satisfies φ. The structure $\mathcal{A}$ is then called a $\mathcal{T}$-*model* of φ. If φ is a Σ-formula and Γ is a set of Σ-formulas, we say that Γ *entails* φ *in* $\mathcal{T}$, written $\Gamma \models_{\mathcal{T}} \varphi$, if every Σ-structure that satisfies all formulas in Γ also satisfies φ. If $\emptyset \models_{\mathcal{T}} \varphi$, the formula φ is called $\mathcal{T}$-*valid* or a *theory lemma*. A set Γ of Σ-formulas is called $\mathcal{T}$-*inconsistent*, if $\Gamma \models_{\mathcal{T}} \bot$.

For two signatures Σ and Ω, we call Ω a *sub-signature* of Σ if $\Omega^f \subseteq \Sigma^f$ and $\Omega^p \subseteq \Sigma^p$. Given a Σ-structure $\mathcal{A}$ and a signature Ω that is a sub-signature of Σ, the *reduct* of $\mathcal{A}$ to a sub-signature Ω is an Ω-structure $\mathcal{A}'$ that has the same universe as $\mathcal{A}$ and coincides with $\mathcal{A}$ on all symbols from Ω. For a Σ_1-theory $\mathcal{T}_1$ and a Σ_2-theory $\mathcal{T}_2$, their *combination* $\mathcal{T}_1 + \mathcal{T}_2$ is the largest $(\Sigma_1 \cup \Sigma_2)$-theory that contains all $(\Sigma_1 \cup \Sigma_2)$-structures $\mathcal{A}$ for which the reduct of $\mathcal{A}$ to Σ_1 is in $\mathcal{T}_1$ and the reduct of $\mathcal{A}$ to Σ_2 is in $\mathcal{T}_2$.

We use the notation $\varphi[x_1, \ldots, x_n]$ for a formula whose free variables are from the set $\{x_1, \ldots, x_n\}$. If $\varphi[x_1, \ldots, x_n]$ is a formula and $t_1, \ldots, t_n$ are terms, $\varphi[t_1, \ldots, t_n]$ denotes a results of the simultaneous substitution of every free variable x_i in φ by the term t_i.

In the rest of this chapter, we consider only theories for which the satisfiability of conjunctions of Σ-literals is decidable and we call any

decision procedure for conjunctions of Σ-literals a $\mathcal{T}$-*solver*. Moreover, if the signature or the theory is clear from the context, we drop the respective Σ- or $\mathcal{T}$- prefixes.

2.1.3 *Many-sorted logic*

For some theories, it can be convenient to distinguish several types of objects instead of having only one universe. This can be achieved by using a *many-sorted logic*. In contrast with a single-sorted signature, defined in the previous subsection, a *many-sorted signature* Σ additionally contains a set of *sort symbols* Σ^S. Arities in a many-sorted logic are also modified. Arity of a symbol determines not only a number of its arguments, but also their types – each function symbol has assigned $(n+1)$-tuple of sort symbols for a non-negative integer n and each predicate symbol has assigned a n-tuple of sort symbols for a non-negative integer n. Simultaneous inductive definition of terms of sorts $S \in \Sigma^S$ and definitions of atoms, literals, clauses and formulas are straightforward and can be found for example in Enderton [End01].

For a many-sorted signature Σ, a Σ-*structure* $\mathcal{A}$ consists of an assignment $(\_)^{\mathcal{A}}$, which to each sort symbol $S \in \Sigma^S$ assigns a non-empty set $S^{\mathcal{A}}$, called the *domain of S in $\mathcal{A}$*, to each variable x of sort S assigns an element $x^{\mathcal{A}} \in S^{\mathcal{A}}$, to each function symbol $f \in \Sigma^f$ of an arity $(S_1, \ldots, S_n, S_{n+1})$ assigns a function $f^{\mathcal{A}} : S_1^{\mathcal{A}} \times \ldots \times S_n^{\mathcal{A}} \to S_{n+1}^{\mathcal{A}}$, and to each predicate symbol $P \in \Sigma^p$ of an arity $(S_1, \ldots, S_n)$ assigns a relation $P^{\mathcal{A}} \subseteq S_1^{\mathcal{A}} \times \ldots \times S_n^{\mathcal{A}}$. Definitions of many-sorted satisfiability, models, entailment, and so on are also straightforward and can also be found in Enderton [End01].

2.2 PROPOSITIONAL SATISFIABILITY

A *propositional satisfiability problem* (SAT) is to decide for a given formula φ in CNF, whether it is satisfiable. The restriction to formulas in CNF is without a loss of generality, as the Tseytin transformation can be used to transform every formula to an equisatisfiable formula in CNF with only linear increase of its size [Tse68].

2.2.1 *Davis–Putnam–Logemann–Loveland algorithm*

Historically, the first procedure to solve SAT without explicitly computing the truth table of the formula was proposed by Davis and Putnam [DP60]. During the Davis–Putnam procedure (DP) the propositional variables of the input formula are successively eliminated using the resolution inference rule [Rob65]. If the resolution yields the empty clause, the formula is unsatisfiable; on the other hand, if after elimination of all variables no

clauses remain, the formula is satisfiable. The main problem of DP is its space complexity, since the number of the clauses may grow exponentially even for simple formulas. To alleviate this problem, the refinement of the DP algorithm was introduced in 1962 by Davis, Logemann and Loveland [DLL62].

If you don't know where you are going any road can take you there.
– Alice in Wonderland

The Davis–Putnam–Logemann–Loveland algorithm (DPLL) iteratively tries to build a satisfying assignment by deciding values of the propositional variables and it backtracks if any of the input clauses becomes false in the current assignment. A key procedure guiding the DPLL search is the *unit clause rule*, which is based on the observation that if a formula contains a clause C that is unit in the current assignment, the only way to build a satisfying assignment is to add the sole undefined literal of C to M. The iterated application of the unit clause rule is called *unit propagation* or *Boolean constraint propagation* (BCP) [ZM88]. The DPLL search consists of decision and propagation steps. In decision steps, a variable and its new value are chosen and added to the current partial assignment. After each decision step, the BCP is performed to set values of variables which are implied by the decision. The backtracking used in DPLL is *chronological*, i.e. after a conflict the value of the last decided literal is changed.

2.2.2 *Conflict-Driven Clause Learning*

Although the DPLL algorithm is more efficient than the original DP algorithm, it may unnecessarily explore parts of the search space that contain no solution. This problem can be partially solved by a further refinement of the DPLL algorithm called Conflict-Driven Clause Learning (CDCL), which was proposed by Marques-Silva and Sakallah [MSS99] and is a base of almost all modern SAT solvers [KSMS11]. In addition to DPLL, CDCL based SAT solvers have mechanisms to analyze a conflict and learn a new clause from a conflicting assignment. Moreover, in contrast to a chronological backtracking, CDCL solvers can perform *non-chronological backtracking* to an earlier decision literal.

In particular, CDCL based SAT solvers maintain an *implication graph* during the search, which records reasons for assigning the unit-propagated literals during the search. If a conflict clause is found, a clause that captures the reason of the conflict is computed by analyzing the implication graph or, equivalently, by the backwards resolution process [BKS04]. After computing the clause C that explains the conflict, CDCL based solvers *learn* the clause C, in order not to repeat the similar conflict in the future, and change the value of the last decision literal that occurs in the clause C. Note that this literal does not have to be the last decision literal in the search and therefore a bigger region of the search space can be ruled out from the search, as it is known not to contain any solutions. There are multiple strategies of computing the clausal reason for the conflict, but the

majority of CDCL based SAT solvers are using the *first unique implication point* based learning scheme, which has been shown to produce small clauses in practice [Mos+01].

Beside the learning, the efficiency of the most CDCL based SAT solvers relies on *branching heuristics*, which select the next decision variable during the search. Multiple branching heuristics have been proposed [BF15b] and the majority of modern SAT solver are using VSIDS heuristic [Mos+01], which prefers branching on the variables that have appeared in the biggest number of recent conflicts. More recently, new branching heuristics based on learning rate have been proposed and shown to outperform the currently used VSIDS branching heuristic [Lia+16].

Another important part of modern SAT solvers are dynamic restarts, which allow restarting the search from scratch in hope that clauses learned during the conflict analyses will guide the search from regions of the search space that contain no solutions [GSK98]. Although most commonly used heuristic to decide when to restart the solver is based on the *Luby sequence* [LSZ93], the recent survey has shown that it is outperformed by a heuristic based on the concept of exponential moving averages [BF15a].

In addition to the mentioned heuristics, an efficient implementation of CDCL based SAT solver relies on lazy data structures used in the Boolean constraint propagation. This is important because it has been observed that SAT solvers spend most of the computation time by performing BCP. Therefore, most of the modern SAT solvers use *two-watched literal scheme* [Mos+01] to efficiently identify clauses that have become unit during the search.

The contribution of all mentioned parts of the CDCL based SAT solvers to the overall performance of the MiniSAT solver [ES03] has been experimentally evaluated by Katebi, Sakallah, and Marques-Silva [KSMS11]. The experimental evaluation shows that the most important from the mentioned techniques is clause-learning, followed by the VSIDS branching heuristic.

2.2.3 *Other approaches to SAT*

Aside from conflict-driven SAT algorithms, other variants of approaches to SAT have been proposed. Most notable of those variants are *symbolic SAT solvers* [AV01; JS04; PV04; HD04] and *look-ahead based SAT solvers* [HM09].

Symbolic algorithms rely on a compact representation of the set of satisfying assignments using *Binary Decision Diagrams* (BDDs) – a data structure proposed by Bryant to represent Boolean functions [Bry86]. In particular, symbolic SAT algorithms are using Reduced Ordered BDDs, which represent Boolean functions *canonically* and allow efficiently com-

puting Boolean combinations of the represented Boolean functions using the recursive procedure `Apply` [Bry86]. The main idea of symbolic SAT solvers is to convert a CNF formula to the corresponding ROBDD and check the root of the resulting BDD. If the resulting BDD is equivalent to the BDD for the function that is 0 everywhere, the formula is unsatisfiable, and it is satisfiable otherwise. However, in order to keep the size of the BDD small, it is necessary to existentially quantify the variables during the computation. This technique is known as *early quantification* [HKB96]. A simplified symbolic SAT algorithm can be found for example in the survey of SAT solving by Darwiche and Pipatsriswat [DP09].

Look-ahead based algorithm, in contrast to the CDCL, are employing expensive heuristics to guide the DPLL search to a satisfying assignment instead of using cheap heuristics and learning from the conflicts [HD04]. Therefore, while CDCL solvers are efficient on large industrial formulas, look-ahead-based SAT solvers are efficient on small hard problems, which require sophisticated heuristics [Heu+11]. The strengths of these two approaches were combined to a technique called *cube-and-conquer*, in which a look-ahead-based solver is used to partition the formula to many simpler formulas, which are in turn solved by an efficient CDCL solver [Heu+11]. The cube-and-conquer approach has been shown to be efficient in practice due to an easy parallelization and has been recently used to solve a long-standing open problem in the Ramsey theory [HKM16].

2.3 SATISFIABILITY MODULO THEORIES

Similarly to SAT, the *satisfiability modulo theories problem* (SMT) is to decide for a given a CNF formula φ in a fixed theory $\mathcal{T}$ whether it is $\mathcal{T}$-satisfiable. Depending on the theory $\mathcal{T}$, the complexity of the SMT problem ranges from polynomial to undecidable. However, as the formula φ can contain Boolean connectives, the SMT problem is at least NP-hard for non-trivial theories.

Notable examples of decidable first-order theories include

- the theory of *equality and uninterpreted functions*, which for a given signature contains all structures over this signature and therefore imposes no restriction on the interpretation of function and predicate symbols;
- the theory of *linear integer arithmetic* over the signature $\{+, \leqslant\}$, which consists of all structures isomorphic to integers with the function $+$ and the relation $\leqslant$;
- the theory of *linear real arithmetic* over the signature $\{+, \leqslant\}$, which consists of all structures isomorphic to real numbers with the function $+$ and the relation $\leqslant$;

- the theory of *real arithmetic* over the signature $\{+, \times, \leqslant\}$, which consists of all structures isomorphic to real numbers with functions $+$ and $\times$ and the relation $\leqslant$;
- the theory of *arrays* over the signature $\{\texttt{read}, \texttt{write}\}$, which consists of all structures isomorphic to the set of arrays with a binary function $\texttt{read}(a, i)$ interpreted as a value on the index i of the array a, and a ternary function $\texttt{write}(a, i, v)$ interpreted as an array a modified to contain the value v on the index i;
- the theory of *fixed-size bit-vectors*, in which the structure contains set of all bit-vectors as an universe, bit-wise and arithmetic functions on those bit-vectors, and ordering relations on the corresponding integers. This theory is in detail described in section 2.4.

For a detailed description of these theories and implementation of the respective $\mathcal{T}$-solvers, we refer the reader for example to the book of Bradley and Manna [BM07].

In practice, a satisfiability of formulas using a combination of theories is often needed. For example, a single formula may be using integers, arrays, and uninterpreted functions. In their seminal work, Nelson and Oppen have shown that satisfiability of combination of stably infinite theories with disjoint signatures can be solved by using separate satisfiability solvers for the respective theories, which interchange implied equalities and disequalities between shared variables [NO79]. A theory $\mathcal{T}$ is *stably infinite* if every $\mathcal{T}$-satisfiable formula has a $\mathcal{T}$-model whose universe is infinite. For a theory over a many-sorted logic, the theory is *stably infinite* if every $\mathcal{T}$-satisfiable formula has a model, whose every sort has an infinite domain. Although almost all practically used theories are stably infinite, this is not true for inherently finite theories like the theory of bit-vectors. Therefore, variants of the theory combination in which one of the theories does not have to be stably infinite have been proposed [TZ05; RRZ05; JB10].

2.3.1 *DPLL modulo theories*

Most of the SMT approaches can be classified as *eager* or *lazy* [Bar+09]. In the eager SMT approach, input formula is directly translated to an equivalent propositional formula and an off-the-shelf SAT solver is used to decide satisfiability of this formula. The eager SMT approach is implemented for example in the SMT solver UCLID for the combination of the theory of uninterpreted functions and linear integer arithmetic [LS04].

On the other hand, the lazy SMT approach uses a SAT solver to reason about the Boolean structure of the formula and a specialized $\mathcal{T}$-solver to

reason about conjunctions of Σ-literals. There are two variants of the lazy approach to the SMT solving – *online* and *offline* [Fla+03].

OFFLINE APPROACH In the offline approach, the input formula φ is first converted to a propositional formula φ^p called *Boolean abstraction* by treating each atom as a propositional variable and then a SAT solver is asked to decide satisfiability of φ^p. If φ^p is unsatisfiable, the input formula φ is also unsatisfiable. On the other hand, if φ^p is satisfiable, the SAT solver returns its propositional model M^p, and the $\mathcal{T}$-solver can be used to determine $\mathcal{T}$-satisfiability of the corresponding set of Σ-literals M. If the $\mathcal{T}$-solver decides M to be $\mathcal{T}$-satisfiable, the formula φ is also $\mathcal{T}$-satisfiable. If the propositional model is not $\mathcal{T}$-satisfiable, the corresponding theory lemma $\neg M^p$ is added to the formula and the procedure is repeated. This variant of the lazy SMT approach is called offline, because it uses the SAT solver as a black box and employs the $\mathcal{T}$-solver only to check satisfiability of a complete Boolean assignment.

ONLINE APPROACH In contrast to the offline approach, the SAT and SMT solvers can cooperate more tightly, resulting in an online variant of the lazy SMT approach, which outperforms the offline approach in most of the cases [Fla+03]. In the online variant, the SMT solver is used not only to check validity of total Boolean assignments, but it is also used to check the validity of partial assignments during the DPLL search (known as *early pruning* [Aud+02]), to assign values of literals which are $\mathcal{T}$-entailed by the current assignment (known as *theory propagation* [ACG99]), and to provide smaller theory lemmas that explain the conflict. In order for this combination to be efficient, the $\mathcal{T}$-solver should provide several important features. Namely, it should be *incremental*, so it does not restart the computation after every decided literal, it should be able to provide small theory lemmas explaining conflicts and theory propagations, so the SAT solver can backtrack to a relevant decision literal, and it should be able to efficiently unassign the values of literals during backtracking. In most of the cases, the $\mathcal{T}$-solver is also required to be able to provide models for the satisfiable formulas.

This SMT approach, in which the $\mathcal{T}$-solver is used to guide the search of the underlying DPLL solver has been named DPLL(T) [NOT06]. For the detailed description of DPLL(T) and further improvements, we refer the reader to survey on the topic [Seb07], or to the abstract description of the underlying transition system [NOT06]. The DPLL(T) approach is used in the majority of modern SMT solvers, including solvers Barcelogic [NO05], CVC4 [Bar+11], MathSAT [Cim+13], OpenSMT [Bru+10], Simplify [DNS05], Yices [Dut14], and Z3 [MB08b].

2.3.2 *Natural domain SMT*

Although the separation of the propositional and theory reasoning in DPLL(T) allows the solver to be modular, it can be also restricting in some cases. In particular, DPLL(T)-based solvers can not directly reason about values of first-order variables, but have to rely on the $\mathcal{T}$-solver guiding the search over Boolean valuations. While there are some techniques like *splitting on demand* [Bar+06] , which allow the $\mathcal{T}$-solver to add new atoms to the formula and delegate the case splitting to the underlying SAT solver, their performance depends on the internal heuristics of the SAT solver. To overcome this issue, Cotton has proposed *Natural domain SMT* approach, which allows the SMT solver to reason directly about the natural domains of variables like integers or reals instead of merely choosing Boolean valuations of existing literals [Cot10]. This has led to two main techniques of SMT solving – *abstract conflict-driven clause learning* and *model-constructing satisfiability calculus* [Mon16].

ABSTRACT CDCL Abstract conflict-driven clause learning (ACDCL) is based on the lattice-theoretic abstract interpretation framework of Cusot and Cusot, widely used to reason about properties of computer programs [CC79]. In order to reason about a behavior of the computer program, its semantics is described as a fix point expression over a *concrete domain* equipped with a set of *concrete transformers*. An *abstract domain* is then a lattice equipped with over-approximations or under-approximations of the concrete transformers, called *abstract transformers*, which can be used to approximate the set of program behaviors.

In recent years, CDCL and DPLL(T) algorithms have been shown to be instances of the abstract interpretation framework [DHK13; Bra+13; DHK14]. In this setting, the concrete domain contains all structures and is equipped with a *model transformer*, computing all models, and a *conflict transformer*, computing all counter-models. The satisfiability problem is then to compute whether the greatest fixed point of the model transformer is different from $\bot$. In the CDCL algorithm, an abstract domain is then a lattice of partial assignments, the unit propagation rule is an over-approximation of the model transformer, BCP is a greatest fixed point computation of this abstract model transformer, decisions are downward extrapolations, and a conflict analysis corresponds to computing an under-approximation of the conflict transformer. Further, clause learning can be seen as learning of a new transformer, which is computed by the conflict analysis.

In thus specified framework, an abstract domain can be replaced by an arbitrary domain that satisfies a certain set of requirements and this yields an abstract conflict-driven clause learning ACDCL algorithm using this particular domain. For example, Brain et al. have implemented the FP-

ACDCL solver for the floating point arithmetic using an abstract interval domain and have shown that on a certain class of formulas it significantly outperforms state-of the art SMT solvers based on the encoding floating point numbers into bit-vectors [Bra+14].

MODEL-CONSTRUCTING SATISFIABILITY CALCULUS The other approach to the natural SMT solving is the model-constructing satisfiability calculus (MC-SAT), proposed by de Moura and Jovanović [MJ13; JBM13]. MC-SAT is based on the DPLL style search, but in addition to assigning values of Boolean variables, it can assign the value of theory variables and it can generate new literals not occurring in the original formula. De Moura et al. have implemented the MC-SAT solver supporting linear real arithmetic and uninterpreted functions, which uses model-driven Fourier-Motzkin elimination [Dan63] to learn new predicates from arithmetic conflicts and model-driven Ackermannization [MB08a; Ack54] to learn new predicates from conflicts involving uninterpreted functions. This solver has been shown to have performance comparable with existing state-of-the-art DPLL(T) solvers [JBM13].

2.3.3 *Additional features of SMT solvers*

Aside from using models for satisfiable formulas, algorithms using SAT and SMT-based solvers often rely on additional features of solvers. For example, several efficient formal verification algorithms are using solvers' ability to provide small *unsatisfiable cores* for unsatisfiable formulas, or to compute *Craig interpolants* [McM06; Hei+13; Hei+14].

In the context of SAT and SMT solving, an unsatisfiable core is a subset of clauses of an unsatisfiable formula that is already unsatisfiable [CGS07]. Three main approaches are used for the unsatisfiable core computation in SMT [Bar+09]. In the first, if a solver can compute resolution proofs for unsatisfiable formulas, the clauses that occur in the proof can be returned as an unsatisfiable core. In the second approach each clause can be represented by a fresh selector variable and after a top level conflict, clauses whose selector variable appears in the final conflict clause are returned as an unsatisfiable core. Third approach consists in combination of an SMT solver with an external propositional core extractor [CGS07].

A (Craig) interpolant for a pair of formulas F and G such that $F \wedge G$ is unsatisfiable is a formula I such that $F \models_T I$, $G \wedge I$ is unsatisfiable, and all uninterpreted symbols in I are contained in both F and G. Following the work of Pudlák [Pud97] and McMillan [McM05], an interpolant for the pair (F, G) over the theory $\mathcal{T}$ can be computed from a resolution proof of unsatisfiability of $F \wedge G$ by using a combination of a propositional interpolating algorithm and a theory specific interpolation procedure, which computes interpolants of theory lemmas. Such interpolation algorithms

exist for linear real arithmetic [CHN12], linear integer arithmetic [KLR10; GLS11; Bri+11; CHN12] or quantified linear integer arithmetic with arrays and uninterpreted functions [McM11].

2.4 SATISFIABILITY OF QUANTIFIER-FREE BIT-VECTOR FORMULAS

The *theory of fixed sized bit-vectors ($\mathcal{BV}$)* is a many-sorted first-order theory with infinitely many sorts $[n]$ corresponding to bit-vectors of length n. The only predicate symbols in the $\mathcal{BV}$ theory are $\leqslant_u$ and $\leqslant_s$, interpreted as inequality of binary-encoded unsigned and signed integers, respectively. Function symbols in the theory are $+$, $\times$, $\div$, $\&$, $|$, $\oplus$, $\ll$, $\gg$, $\cdot$, $\texttt{extract}^n_p$, interpreted as addition, multiplication, unsigned division, bit-wise and, bit-wise or, bit-wise exclusive or, left-shift, right-shift, concatenation, and extraction of n bits starting from the position p, respectively. For the detailed description of the $\mathcal{BV}$ theory syntax and semantics, see for example Hadarean's PhD thesis [Had15]. This section focuses on the problem of satisfiability of the quantifier-free fragment of the $\mathcal{BV}$ theory, denoted $\mathcal{QFBV}$. The the full $\mathcal{BV}$ logic is treated in the next section.

Current state-of-the-art SMT solvers for the $\mathcal{QFBV}$ logic rely on rewriting techniques used to simplify the formula during the preprocessing, and eager or lazy translation of the bit-vector formula to the equivalent propositional formula, which is subsequently solved by a SAT solver. The transformation of a bit-vector formula to the equivalent propositional formula is traditionally called *bit-blasting* [KS08]. More lazy approach to the bit-blasting is beneficial when the theory combination is required. For example, solvers Z3 and Yices apply bit-blasting to all operations except for the equality, which is handled by a specialized solver, and dynamically add axioms of the array theory [MB08b; Dut14]. Boolector applies bit-blasting to the bit-vector operations and lazily instantiates definitions of macros and array axioms [NPB14]. Furthermore, CVC4 uses lazy and layered solver, which tries to decide the satisfiability using possibly faster but incomplete sub-solvers for equality and inequality reasoning and if the sub-solvers are not sufficient for deciding the satisfiability of the formula, theory lemmas and propagated literals generated by the sub-solvers are added to the formula and a lazy DPLL(T) bit-blasting solver is employed [Had+14].

2.4.1 *Word-level techniques*

Although bit-blasting is highly efficient for most of practical problems, it can exhaust memory of the solver if the input formula contains complex arithmetic or variables with large bit-width. Several techniques that avoid

the bit-blasting and work directly on the level of individual bit-vectors (*word level*) have been proposed to alleviate this problem.

Some useful fragments of the bit-vector theory can be solved by specialized algorithms for deciding satisfiability. For example, conjunctions of literals in the *core theory of bit-vectors* consisting of equality, concatenation and extraction, are decidable in polynomial time by reduction to the theory of equality [CMR97; BS09]. Additionally, Babić and Musuvathi have described algorithms for solving linear and non-linear modular arithmetic, which can be used for deciding satisfiability of bit-vector formulas that contain only arithmetic and no bit-wise operations [DB05].

Recently, an instance of the model-constructing satisfiability calculus, which was introduced in subsection 2.3.2, was implemented by Zeljić et al. in the solver MCBV [ZWR16]. Zeljić et al. have extended the MC-SAT framework by the ability to perform partial assignments and have proposed heuristics for generalizing explanations of bit-vector conflicts. For example, the solver MC-SAT can perform the partial assignment $\texttt{extract}_0^2(x) \mapsto 10$, denoting that the two least significant bits of x are 10. To be able to efficiently use such partial assignments, the solver MCBV maintains two over-approximations of the set of models that are compatible with the current partial assignment – using *bit-patterns* and *arithmetic intervals*. Bit-patterns are sequences of 0, 1 and u (represents undefined bit), which constrain the values of particular bits in the assignment. On the other hand, arithmetic intervals are pairs of bit-vector values representing lower and upper bounds and constrain integral values of bit-vectors. Both bit-patterns and arithmetic intervals can be ordered to form a lattice in which the solver performs a search for a more general explanation if a conflict is detected.

Another word-level approach for the full bit-vector theory is *stochastic local search* (SLS), proposed for solving bit-vectors by Fröhlich et al. [Frö+15] and subsequently improved by Niemetz et al. [Nie+15; NPB16]. In the SLS approach, the solver randomly chooses initial values of bit-vector variables and tries to find a satisfying assignment by performing random bit flips, which are guided by the scoring function based on the satisfaction of subformulas of the input formula. Niemetz et al. have improved the SLS technique with the *path-based propagation*, which, instead of relying solely on random bit modifications guided by the scoring function, allows computing values of bit-vector variables that are necessary to satisfy randomly selected subformulas. The SLS based solver has been shown to be able to decide several formulas not decided by bit-blasting solvers. To combine benefit of bit-blasting and SLS approaches, the latest version of Boolector, which have won the 2016 SMT competition in the category of quantifier-free bit-vectors, uses a portfolio approach, which consists in first running a SLS based solver for a short period of time

and then running a bit-blasting solver if the SLS solver fails to solve the formula [Boo].

2.4.2 *Preprocessing*

For both bit-blasting and word-level approaches, a preprocessing of the input formula is crucial for the efficiency of the solver. Therefore, modern SMT solvers employ hundreds of rewrite rules in order to simplify the input formula [Fra10]. Franzén describes several classes of simplification methods implemented in the solver MathSAT5: canonization, unconstrained variables propagation, packet splitting, and disjunctive partitioning [Fra10].

The aim of the *canonization* is converting subterms of the formula to their canonical forms. For example, the term $x - x$ can be rewritten to 0. Although efficient algorithms exist for several fragments of the $\mathcal{BV}$ theory, like the core theory of bit-vectors, the problem of computing the canonical form for the full $\mathcal{BV}$ theory is NP-hard [BDL98]. Therefore, SMT solvers usually do not compute the canonical form, but rely on multiple heuristically chosen rewrite rules that produce the canonical form for simple terms, but are not required to produce the canonical form of all terms.

From the remaining simplification techniques presented by Franzén, we focus on propagation of unconstrained variables, as it is highly relevant for software verification.

A variable x in a formula is called *unconstrained* if it occurs only once in the formula. Brummayer [Bru10] and Bruttomesso [Bru08] have independently observed that if an unconstrained variable occurs as an argument to a function symbol that can be *inverted* with respect to this argument, replacing this function with a fresh variable yields an equisatisfiable formula. Moreover, unconstrained variables often occur in the industrial benchmarks and especially in benchmarks produced during a verification of programs in a single static assignment form, such as LLVM bit-code.

With a slight abuse of notation which identifies function symbols with their intended interpretations, a simple definition of invertibility for binary function symbols is as follows. A binary function f can be inverted with respect to its first argument if for every two values a_i, a_o there exists a value b such that $f(b, a_i) = a_o$. This definition can be easily generalized to functions of different arities. For example, as the addition is invertible with respect to its first argument, the formula $\varphi \equiv x+(y*y-30*z*y) = 0$ can be transformed to an equisatisfiable formula $v = 0$, where v is a fresh variable, because x is unconstrained in φ. Note that in contrast to the theory of integers, the function $f(x) = k \times x$ is invertible in bit-vectors precisely if k is odd.

2.5 SATISFIABILITY OF QUANTIFIED BIT-VECTOR FORMULAS

Although the bit-vector theory admits quantifier elimination by expanding all quantifiers with all possible bit-vector values of the corresponding bit-width, this is rarely a practical approach. Instead, the formula is usually converted to an equisatisfiable formula by Skolemization and then instances of the universally quantified formulas are lazily added to the formula until a model is found or the formula is found to be unsatisfiable by a $\mathcal{QFBV}$ solver. There are multiple ways to choose quantifier instances that are sufficient to decide the satisfiability of the formula. For the bit-vector theory, the most widely used approach is the *model-based quantifier instantiation* [GM09], implemented in Z3, CVC4, and Yices, combined with heuristics as *E-matching* or *symbolic quantifier instantiation* [WHM13; Dut15]. Additionally, for dealing with quantifiers, CVC4 supports solving quantified formulas by *counter-example guided quantifier instantiation* [Rey+15]. We describe only the model-based quantifier instantiation in detail, as the counter-example guided quantifier instantiation considers all functions as uninterpreted during the conflict search, and therefore its performance on bit-vector formulas is limited.

2.5.1 *Model-based quantifier instantiation*

Given a closed formula with quantifiers, the first step is to convert the formula to the negation normal form and apply Skolemization to obtain equisatisfiable formula of the form

$$\varphi \wedge \forall x_1, x_2, \ldots, x_n \, (\psi[x_1, \ldots, x_n]),$$

where φ and ψ are quantifier-free formulas. Then the $\mathcal{QFBV}$ solver is invoked to check the satisfiability of the formula φ. If φ is unsatisfiable, then the entire formula is unsatisfiable. If φ is satisfiable, the $\mathcal{QFBV}$ solver returns its model M and another call to the $\mathcal{QFBV}$ solver is made to determine whether M is also a model of $\forall x_1, x_2, \ldots, x_n \, (\psi)$. This is achieved by asking the solver whether the formula $\neg\widehat{\psi}$ is satisfiable, where $\widehat{\psi}$ is the formula ψ after replacing all uninterpreted functions and all variables except for $x_1 \ldots x_n$ and by their values in M. If $\neg\widehat{\psi}$ is not satisfiable, then the structure M is indeed a model of the formula $\forall x_1, x_2, \ldots, x_n \, (\psi)$, therefore the entire formula is satisfiable and M is its model. If $\neg\widehat{\psi}$ is satisfiable, we get values $v_1, \ldots, v_n$ such that $\neg\widehat{\psi}[v_1, \ldots, v_n]$ holds. To rule out M as a model of the input formula, the instance $\psi[v_1, \ldots, v_n]$ of the quantified formula is added to the quantifier-free part, i.e. the formula φ is modified to

$$\varphi' \equiv \varphi \wedge \psi[v_1, \ldots, v_n],$$

and the procedure is repeated.

Example 2.5.1. Consider the formula $3 < a \wedge \forall x\,(\neg(a = 2 \times x))$. The subformula $3 < a$ is satisfiable and $a = 4$ is its model. However, it is not a model of the formula $\forall x\,(\neg(a = 2 \times x))$, since the $\mathcal{QFBV}$ solver called on the formula $\neg(\neg(4 = 2 \times x))$ returns $x = 2$ as a model. The next step is to decide the satisfiability of the formula $3 < a \wedge \neg(a = 2 \times 2)$. This formula is satisfiable and $a = 5$ is its model. Moreover, it is also a model of $\forall x\,(\neg(a = 2 \times x))$ as $\neg(\neg(5 = 2 \times x))$ is unsatisfiable. Hence, the input formula is satisfiable and $a = 5$ is its model.

This algorithm is trivially terminating, since there is only a finite number of distinct models M of φ. However, in some cases exponentially many such models have to be ruled out before the solver is able to find a correct model or decide unsatisfiability of the whole formula. To overcome this issue, state-of-the-art SMT solvers do not use just instances of the form $\psi[v_1, \ldots, v_n]$ with concrete values, but employ mentioned heuristics such as E-matching or symbolic quantifier instantiation to choose instances with ground terms that can potentially rule out more spurious models and thus significantly reduce the number of iterations of the algorithm. In practice, suitable ground terms substituted for quantified variables are selected only from subterms of the input formula.

Note that Skolemization introduces uninterpreted functions to the quantified formulas of form other than $\exists^*\forall^*\varphi$. This class of formulas is usually called *exists/forall*, or simply $\exists\forall$. Therefore, for the model-based quantifier instantiation approach to be usable for formulas not from the exists/forall class, the underlying $\mathcal{QFBV}$ solver has to support reasoning about uninterpreted functions. In case of Z3, this is achieved by *template-based model finding* – uninterpreted functions are assigned templates and the $\mathcal{QFBV}$ solver is used to find satisfying parameters of these templates. For example, an uninterpreted function f may be assigned a template for linear functions $f(x) = ax + b$ and finding a satisfying function f reduces to finding values a and b.

2.6 COMPUTATIONAL COMPLEXITY OF BIT-VECTOR LOGICS

The propositional satisfiability problem is well known from the computational complexity perspective, as it is the first problem that was shown to be NP-complete. The complexity of the satisfiability problems for various variants of bit-vector theory have been recently studied and was shown to range from NP-complete to 2-NEXPTIME-complete [KFB16]. In particular, interesting variants of the bit-vector satisfiability problem differ in allowing uninterpreted functions, allowing quantifiers, and in encoding of the bit-widths (unary vs. binary). In the following, we follow the notation of Kováznai et al. [KFB16] – decision problems for quantifier-free fragments are denoted by the prefix QF_, the combination with

		Quantifiers			
		No		Yes	
		Uninterpreted functions		Uninterpreted functions	
		No	Yes	No	Yes
Encoding	Unary	NP	NP	PSPACE	NEXPTIME
	Binary	NEXPTIME	NEXPTIME	?	2-NEXPTIME

Table 1: Completeness results for various bit-vector logics and encodings [FKB13].

the theory of uninterpreted functions is denoted by the prefix UF, and the problems with unary and binary encoded bit-widths are denoted by suffixes 1 and 2, respectively. For example, QF_UFBV2 is the decision problem for quantifier-free formulas with uninterpreted functions and binary encoded bit-widths. The completeness results for these classes are summarized in table 1, and briefly explained in the rest of this section.

UNARY ENCODED BIT-WIDTHS The bit-blasting yields a polynomial time reduction from QF_BV1 to SAT, showing that QF_BV1 is in NP. A similar reduction from BV1 to QBF can show that BV is in PSPACE. For lower bounds, NP-hardness of QF_BV1 follows from a simple reduction from SAT by encoding each propositional variable as a bit-vector of bit-width 1. Similarly, BV1 can be shown to be PSPACE-hard by considering every bit-vector as a quantified variable of bit-width 1.

In quantifier-free formulas, uninterpreted functions can be eliminated by the Ackermann expansion with only quadratic increase in the size of the formula. Therefore, QF_UFBV1 is in NP. As the set of QF_UFBV1 formulas contains all QF_BV1 formulas, QF_UFBV1 is also NP-hard. The case of UFBV1 is more involved. Wintersteiger et al. have shown that UFBV1 is NEXPTIME-complete by using *effectively propositional* fragment of the first-order logic, which is well known to be NEXPTIME-complete [WHM13]. The class of effectively propositional formulas consists only of formulas in the form $\exists^*\forall^*\varphi$, where φ does not contain any quantifiers or function symbols.

The class of effectively propositional formulas is also known as the Bernays–Schönfinkel class.

BINARY ENCODED BIT-WIDTHS For formulas with binary encoded bit-widths, the bit-blasting may yield propositional formula that is exponentially larger than the original input bit-vector formula, as the number of bits may be exponential with respect to the size of the formula. Therefore, bit-blasting shows that QF_BV2 is in NEXPTIME. On the other hand, Kovásznai et al. have presented a polynomial time reduction of

satisfiability of *dependent quantified Boolean formulas* (DQBF) to QF_BV2. In contrast to QBF, DQBF formulas allow explicitly stating on which variables can a quantified variable depend. Since DQBF is well known to be NEXPTIME-complete, this reduction shows NEXPTIME-hardness of QF_BV2 [KFB12]. In contrast, the precise complexity after adding quantifiers is not known. BV2 is known to be in EXPSPACE and because it contains all formulas from QF_BV2, it is also NEXPTIME-hard.

Similarly to the case with the unary encoding, the complexity of the quantifier-free fragment stays the same when the uninterpreted functions are added – QF_UFBV2 can be shown to be in NEXPTIME by the Ackermann reduction and NEXPTIME-hard by the simple reduction from QF_BV2. The complexity of the problem after adding quantifiers and uninterpreted functions was investigated by Kovásznai et al. [KFB12]. Reencoding of all bit-widths to unary shows that UFBV2 is in 2-NEXPTIME. For the lower bound, Kovásznai et al. present a reduction of an instance of a *domino tiling problem* [Chl84] that is known to be 2-NEXPTIME-hard.

3 ACHIEVED RESULTS

3.1 SYMBOLIC APPROACH TO QUANTIFIED BIT-VECTORS

Our research is focused on employing symbolic methods for deciding satisfiability of quantified bit-vector formulas. In particular, we have developed a BDD-based solver Q3B for quantified bit-vector logic. The solver uses simplifications like early quantification in order to reduce the size of the intermediate BDDs, tailored BDD variable ordering based on dependencies in the input formula, which can further reduce the size of BDDs, and approximations, which partially alleviate infeasibility of representing multiplication with a BDD. This section summarizes all three of these components, which were in detail described in our paper published at the SAT 2016 conference [JS16].

The benefit of a BDD-based solver over the quantifier instantiation-based one is the ability of representing all models of a universally quantified formula even in cases in which finding a suitable set of quantifier instantiations would be infeasible. The following formula, in which all variables have bit-width 32 bits, shows this difference:

$$\varphi \equiv a = 16 \cdot b + 16 \cdot c \wedge \forall(x : [32])\,(a \neq 16 \cdot x).$$

The BDD-based solver can quickly compute the BDD for the quantified subformula $\forall(x : [32])\,(a \neq 16 \cdot x)$, which shows that at least one of the last 4 bits of the variable a has to be 1, i.e. the value of a is not divisible by 16. After conjoining this BDD to the BDD for $a = 16 \cdot b + 16 \cdot c$, the formula is decided unsatisfiable, as the resulting BDD represents the function that is 0 everywhere. On the other hand, if one considers only quantifier instantiations by the subterm of the input formula, exponentially many quantifier instances have to be added to the formula to show its unsatisfiability, as there is no subterm of φ that can be instantiated as x to yield an unsatisfiable quantifier-free formula.

3.1.1 *Simplifications*

Simplifications used in Q3B aim for reducing the size of the intermediate BDDs. This can be achieved, besides partially canonizing the subterms, by reducing scopes of quantifiers occurring in the formula. One such simplification is distributivity of universal quantification over conjunction:

$$\forall x.\,(\varphi[x] \wedge \psi[x]) \quad \rightsquigarrow \quad (\forall x.\,\varphi[x]) \wedge (\forall x.\,\psi[x]).$$

After such transformation, the BDD for $\varphi \wedge \psi$ does not have to be computed and the conjunction of the BDDs is computed only after elimination of the universal quantifier, which usually reduces the size of the formula. Dually, the existential quantification can be distributed over disjunctions.

Moreover, universal quantification can distribute over disjunctions in cases where the variable bound by the quantifier does not occur in one of the disjuncts. This leads to the following rule and its dual version, which is known as *miniscoping* [Haro9]:

$$\forall x.\,(\varphi[x] \vee \psi) \quad \rightsquigarrow \quad (\forall x.\,\varphi[x]) \vee \psi,$$

where x does not occur freely in ψ. Note that the scope of the quantifier is again reduced, which can lead to smaller intermediate BDDs.

Q3B also uses *destructive equality resolution* (DER) rule, which was proposed for quantified bit-vector formulas by Wintersteiger et al:

$$\forall x.\,(x \neq t \vee \varphi[x]) \quad \rightsquigarrow \quad \varphi[t],$$

where t is an arbitrary term that does not contain x. As the DER rule eliminates the quantified variable, it in many cases also reduces the size of the BDDs.

Unlike Z3, for which the DER rule was proposed, our solver does not perform Skolemization before solving. Therefore we also need a dual version of the DER rule, which we have called *constructive equality resolution*:

$$\exists x.\,(x = t \wedge \varphi[x]) \quad \rightsquigarrow \quad \varphi[t],$$

where t is an arbitrary term that does not contain x.

3.1.2 *Variable ordering*

Ordering of BDD variables is crucial for efficiency. The size of a BDD can differ exponentially when choosing a different ordering. Therefore, besides well known methods of dynamic variable reordering during the computation, Q3B computes an initial variable ordering, which is based on the syntactic dependencies among the variables in the formula.

We have implemented and compared three different initial orderings, which we denote $\leqslant_1, \leqslant_2, \leqslant_3$.

- In $\leqslant_1$, all bit variables are ordered according to their significance (from the least to the most significant) and variables with the same significance are ordered by the order of the first occurrences of the corresponding bit-vector variables in the formula. In this ordering, bit variables of the same significance are close to each other.

- In $\leqslant_2$, bit variables are ordered by the order of the first occurrences of the corresponding bit-vector variables in the formula and bit variables corresponding to the same bit-vector variable are ordered according to their significance (from the least to the most significant). In this ordering, bit variables corresponding to the same bit-vector variable are close to each other.

- In $\leqslant_3$, we first identify syntactically dependent variables. Two variables are *syntactically dependent* if they occur in the same atomic subformula. Then the set of variables is decomposed to equivalence classes modulo the transitive closure of the syntactic dependence relation and each of these classes is ordered according to $\leqslant_2$. Therefore, in this ordering, only potentially dependent bit variables of the same significance are close to each other.

From these three orderings, $\leqslant_3$ performs best on the set of our benchmarks even if the dynamic variable reordering by *sifting* [Rud93] is used.

3.1.3 *Approximations*

The size of the BDD for multiplication is exponential regardless the variable ordering [Bry91]. Therefore, to be able to decide formulas with multiplication or complex arithmetic, Q3B uses approximations of the input formula, which allow deciding a satisfiability of the input formula by solving a simpler formula instead. In the context of SMT solving, an underapproximation of a formula is a formula that logically entails the input formula and an overapproximation of an input formula is a formula logically entailed by the input formula. It can be easily seen that satisfiability of an underapproximation implies satisfiability of the input formula and unsatisfiability of an overapproximation implies unsatisfiability of the input formula.

Q3B employs approximations similar to those used by the SMT solver UCLID [Bry+07]. In UCLID, the quantifier-free input formula is underapproximated by representing each bit-vector variable by the smaller number of bit variables than its original bit-width. The number of bits by which the bit-variable is represented is called the *effective bit-width*. For reducing the effective bit-width, UCLID offers multiple ways to represent a variable by a smaller number of bits: *zero extension*, *one-extension*, and *sign-extension*. In all three, the effective bit-width is used to represent the least significant bits of the bit-vector variable and all other bits are set to 0, 1, or the value of the most significant effectively represented bit, respectively. To these extensions, we have proposed their right variants, which use effective-bit width to represent the most significant bits. Figure 1 illustrates left and right variants of zero-extension and sign-extension.

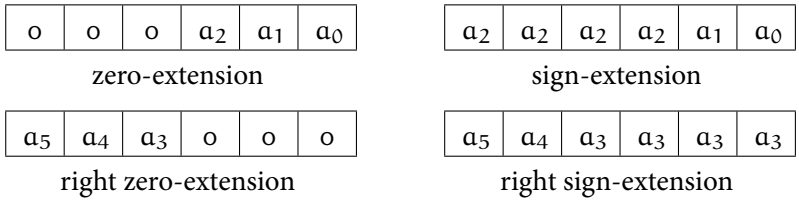


Figure 1: Reductions using zero-extension and sign-extension of a 6-bit variable $a = a_5 a_4 a_3 a_2 a_1 a_0$ to 3 effective bits.

In contrast to UCLID, we can use the same approach to perform overapproximations, because we work with the quantified formulas. For formulas in the negation normal form, underapproximations can be achieved by reducing effective bit-widths of existentially quantified variables and overapproximations can be achieved by reducing effective bit-widths of universally quantified variables.

In Q3B, approximations are used in a portfolio style. The solver without approximations is run in parallel with the solver which employs underapproximations and the solver which employs overapproximations. Our experimental evaluation has shown that this set up can help to decide more formulas, especially formulas containing multiplication.

3.1.4 *Experimental evaluation*

Our experimental evaluation has shown that a BDD-based SMT solver can decide more formulas than CVC4 and Z3, when the set of quantified bit-vector formulas in the SMT-LIB benchmark library [BST10] is considered. Additionally, we have gathered quantified formulas produced by the semi-symbolic model checker SymDIVINE [BHB14], which uses quantified formulas to decide the equality of two symbolic states. On this set of benchmarks, our solver Q3B also has better performance than SMT solvers based on the quantifier instantiation and bit-blasting. Table 2 shows the numbers of formulas solved by each solver and Figure 2 presents the quantile plot of CPU times needed to solve these formulas.

All experiments were performed on a Debian machine with two six-core Intel Xeon E5-2620 2.00GHz processors and 128 GB of RAM. Each benchmark run was limited to use 3 processor cores, 4 GB of RAM and 20 minutes of CPU time (if not stated otherwise).

Recently, these results were confirmed by the SMT competition 2016[1] – our solver is the winner of the quantified-bit vectors category. Table 3 shows official results for this category. The benchmarks are divided into two groups, with *known* and *unknown* status. A benchmark has a known status if at least two solvers in the previous year of the competition agreed

1 http://smtcomp.sourceforge.net/2016/

	SMT-LIB				SymDIVINE			
	sat	unsat	unknown	timeout	sat	unsat	unknown	timeout
CVC4	29	55	32	75	1 124	3 845	2	490
Z3	71	93	5	22	1 135	4 162	22	142
Q3B	**94**	**94**	0	3	**1 137**	**4 202**	0	122

Table 2: For each benchmark set and each solver, the table provides the numbers of formulas decided as satisfiable (*sat*), unsatisfiable (*unsat*), or undecided with the result unknown or because of an error (*unknown*), or a *timeout*.

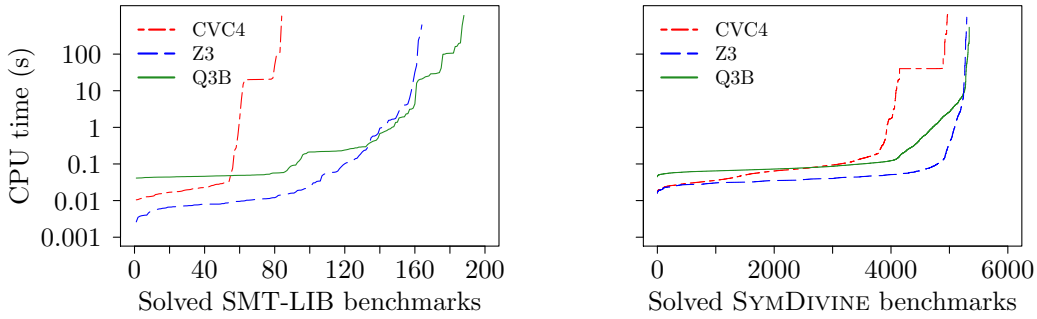


Figure 2: Quantile plot of the number of benchmarks which CVC4, Q3B, and Z3 solved compared by the CPU time.

	Known status		Unknown status		
	# solved	avg. CPU time	# solved	avg. CPU time	avg. WALL time
Boolector	85	1.635	89	**11 431**	11 422
CVC4	85	1.576	56	29 464	29 453
Q3B	85	**0.138**	**99**	12 111	**4 059**
Z3	85	0.339	78	16 721	16 713

Table 3: Official results of quantified bit-vector category of SMT competition 2016 divided into the benchmarks with known status and benchmarks with a previously unknown status. All times are in seconds.

whether the benchmark is satisfiable or unsatisfiable. The results show that Q3B can solve as many known benchmarks as the other solvers, but solves them in the shortest time. Moreover, Q3B can solve more benchmarks with unknown status than any of the other solvers.

3.2 OTHER AREAS OF RESEARCH

Outside the field of SMT solving, my research interests also include software verification. In this field, I have co-authored the following paper on the tool Symbiotic, which combines instrumentation, slicing and, symbolic execution to allow verification of a real-world code [Cha+16].

3.3 PUBLISHED PAPERS

- JONÁŠ, M. and J. STREJČEK. "*Solving Quantified Bit-Vector Formulas Using Binary Decision Diagrams.*" In: Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. 2016, pp. 267-283 [JS16]

  I am the main author of the text of the paper. I have also implemented the SMT solver and conducted all the experiments.

- CHALUPA M., M. JONÁŠ, J. SLABÝ, J. STREJČEK, and M. VITOVSKÁ. "*Symbiotic 3: New Slicer and Error-Witness Generation – (Competition Contribution).*" In: Tools and Algorithms for the Construction and Analysis of Systems – 22nd International Conference, TACAS 2016, Held as Part of the ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings, pp. 946-949 [Cha+16]

  I wrote parts of the paper and prepared the environment that was used to run experiments with the implemented tool Symbiotic.

4 AIM OF THE WORK

4.1 OBJECTIVES AND EXPECTED RESULTS

4.1.1 *Symbolic solver for quantified bit-vectors*

Throughout the rest of my PhD study, I plan to continue developing the implemented symbolic SMT solver Q3B, which is aimed at solving quantified bit-vector formulas. In particular, I plan to add support for uninterpreted functions and the theory of arrays, which is highly desirable for applications in the program verification. I also want to implement extraction of an unsatisfiable core from the intermediate BDDs that were produced during the computation.

Additionally, currently implemented approximations are very simple and could benefit from better refinement in the case that the current approximation is too coarse.

Moreover, I want to implement other variants of decision diagrams such as zero-suppressed decision diagrams introduced by Minato [Min93], binary moment diagrams introduced by Bryant and Chen [BC95], and algebraic decision diagrams introduced by Bahar et al. [Bah+93]. I plan to experimentally evaluate their effect on the performance of the symbolic SMT solver for the quantified bit-vectors.

4.1.2 *Hybrid approach to quantified bit-vectors*

Although results of the symbolic SMT solver for quantified bit-vectors look promising, standard SMT solvers still perform better on queries containing multiplication and complex arithmetic. Therefore, I plan to develop a hybrid approach to SMT solving of quantified bit-vector formulas that combines strengths of both of these approaches. For example, parts of the quantified formula that do not contain multiplication can be converted to the BDD, which can be used to guide the model search in the model-based quantifier instantiation of the other parts of the formula. One possible way of achieving this is adding support for BDDs to the solver MCBV developed by Zeljić et al. The BDD representation can be added to the currently used over-approximations by bit-patterns and arithmetic intervals. Moreover, a tighter cooperation is possible – if a value of a variable is decided during the MCBV computation, it can be used to partially instantiate a quantified part of the formula, for which the BDD will be computed.

4.1.3 *Unconstrained variable propagation for quantified bit-vectors*

Simplifications using unconstrained variables can be extended to quantified formulas. However, in the quantified setting, constraints among variables can be introduced also by the order in which the variables are quantified. We have formulated a hypothesis that describes a sufficient condition for the quantified variable to be considered as unconstrained. Based on this hypothesis and a partial proof of its validity, we have implemented a proof-of-concept simplification procedure that can simplify quantified formulas that contain unconstrained variables. Although the initial experimental results, conducted on the formulas from the semi-symbolic model checker SymDIVINE, look promising, the formal proof of the correctness is not yet complete.

Moreover, the simplifications of formulas with unconstrained variables can be further extended. For example, if a formula contains a term $k \times x$ in which the variable x is unconstrained, this term can be replaced by a simpler term regardless the parity of the value k. In particular, if i is the largest number for which 2^i divides the constant k and the unconstrained variable x has bit-width n, the term $k \times x$ can be rewritten to $extract_0^{n-i}(x) \cdot 0^i$. Intuitively, the term $k \times x$ can have every possible bit-vector value that has the last significant i bits zero. This approach can be also extended to the multiplication of two variables from which one is unconstrained. I plan to investigate further extensions to the terms containing division and remainder operations and to publish a paper concerning propagation of unconstrained variables in quantified formulas and extensions of unconstrained variable simplification to multiplication and division. I will also experimentally evaluate the effect of such simplifications on our solver Q3B and on state-of the art solvers such as Boolector, CVC4, and Z3.

4.1.4 *Complexity of BV2*

As was explained in Section 2.6, the precise complexity of quantified bit-vector formulas with binary-encoded bit-widths and without uninterpreted functions is not known. It is known to be in EXPSPACE and to be NEXPTIME-hard. However, a class for which the problem is complete is not known.

According to our investigation, it is probably complete for neither of those complexity classes. We are working on a proof which shows that BV2 is complete for the class of problems solvable by the *alternating Turing machine* (ATM) with the exponential space and *polynomial number of alternations* with respect to log-space reductions. This class is usually denoted as AEXPTIME(poly) and is known to be in between NEXPTIME

and EXPSPACE [Han+15; Lüc16]. However, whether any of the inclusions is proper is not known.

4.2 PROGRESSION SCHEDULE

The plan of my future study and research activities is following:

NOW – JANUARY 2019
: Improvements and maintaining of the developed symbolic SMT solver Q3B.

NOW – JANUARY 2017
: Extending unconstrained variable propagation to quantified formulas and to non-linear multiplication and division.

JANUARY 2017
: Doctoral exam and defense of this thesis proposal.

FEBRUARY 2017 – APRIL 2017
: Proving a precise complexity class of the quantified bit-vector formulas without uninterpreted functions and with binary encoded bit-widths. Preparing a paper on this topic.

MAY 2017 – DECEMBER 2017
: Development of a hybrid approach to SMT solving of quantified bit-vector formulas combining symbolic representation of formulas with DPLL(T) or MC-SAT.

JANUARY 2018 – AUGUST 2018
: Implementing a hybrid SMT solver

SEPTEMBER 2018 – JANUARY 2019
: Text of the PhD thesis.

JANUARY 2019
: The final version of the thesis.

BIBLIOGRAPHY

[Ack54] Ackermann, W. *Solvable cases of the decision problem*. Amsterdam, 1954.

[AV01] Aguirre, A. S. M. and M. Y. Vardi. "Random 3-SAT and BDDs: The Plot Thickens Further." In: *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*. 2001, pp. 121–136.

[ACG99] Armando, A., C. Castellini, and E. Giunchiglia. "SAT-Based Procedures for Temporal Reasoning." In: *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999, Proceedings*. 1999, pp. 97–108.

[Aud+02] Audemard, G., P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. "A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions." In: *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings*. 2002, pp. 195–210.

[Bah+93] Bahar, R. I., E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. "Algebraic decision diagrams and their applications." In: *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993, Santa Clara, California, USA, November 7-11, 1993*. 1993, pp. 188–191.

[BDL98] Barrett, C. W., D. L. Dill, and J. R. Levitt. "A Decision Procedure for Bit-Vector Arithmetic." In: *DAC*. 1998, pp. 522–527.

[Bar+09] Barrett, C. W., R. Sebastiani, S. A. Seshia, and C. Tinelli. "Satisfiability Modulo Theories." In: *Handbook of Satisfiability*. 2009, pp. 825–885.

[BFT15] Barrett, C., P. Fontaine, and C. Tinelli. *The SMT-LIB Standard: Version 2.5*. Tech. rep. Available at `www.SMT-LIB.org`. Department of Computer Science, The University of Iowa, 2015.

[BST10] Barrett, C., A. Stump, and C. Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. `www.SMT-LIB.org`. 2010.

[Bar+06] Barrett, C., R. Nieuwenhuis, A. Oliveras, and C. Tinelli. "Splitting on Demand in SAT Modulo Theories." In: *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006, Phnom Penh, Cambodia, November 13-17, 2006, Proceedings*. 2006, pp. 512–526.

[Bar+11] Barrett, C., C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. "CVC4." In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. 2011, pp. 171–177.

[BHB14] Bauch, P., V. Havel, and J. Barnat. "LTL Model Checking of LLVM Bitcode with Symbolic Data." In: *MEMICS 2014*. Vol. 8934. LNCS. Springer, 2014, pp. 47–59.

[BKS04] Beame, P., H. A. Kautz, and A. Sabharwal. "Towards Understanding and Harnessing the Potential of Clause Learning." In: *J. Artif. Intell. Res. (JAIR)* 22 (2004), pp. 319–351.

[BF15a] Biere, A. and A. Fröhlich. *Evaluating CDCL Restart Schemes*. 2015.

[BF15b] Biere, A. and A. Fröhlich. "Evaluating CDCL Variable Scoring Schemes." In: *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*. 2015, pp. 405–422.

[Boo] *Boolector at the SMT competition 2016*. Tech. rep. FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2016.

[BM07] Bradley, A. R. and Z. Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 2007.

[Bra+13] Brain, M., V. D'Silva, L. Haller, A. Griggio, and D. Kroening. "An Abstract Interpretation of DPLL(T)." In: *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*. 2013, pp. 455–475.

[Bra+14] Brain, M., V. D'Silva, A. Griggio, L. Haller, and D. Kroening. "Deciding floating-point logic with abstract conflict driven clause learning." In: *Formal Methods in System Design* 45.2 (2014), pp. 213–245.

[Bri+11] Brillout, A., D. Kroening, P. Rümmer, and T. Wahl. "An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic." In: *J. Autom. Reasoning* 47.4 (2011), pp. 341–367.

[Bru10] Brummayer, R. "Efficient SMT solving for bit vectors and the extensional theory of arrays." PhD thesis. Johannes Kepler University of Linz, 2010.

[Bru08] Bruttomesso, R. "TL Verification: From SAT to SMT(BV)." PhD thesis. University of Trento, 2008.

[BS09] Bruttomesso, R. and N. Sharygina. "A scalable decision procedure for fixed-width bit-vectors." In: *2009 International Conference on Computer-Aided Design, ICCAD 2009, San Jose, CA, USA, November 2-5, 2009*. 2009, pp. 13–20.

[Bru+10] Bruttomesso, R., E. Pek, N. Sharygina, and A. Tsitovich. "The OpenSMT Solver." In: *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*. 2010, pp. 150–153.

[Bry86] Bryant, R. E. "Graph-Based Algorithms for Boolean Function Manipulation." In: *IEEE Trans. Computers* 35.8 (1986), pp. 677–691.

[Bry91] Bryant, R. E. "On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication." In: *IEEE Trans. Comput.* 40.2 (1991), pp. 205–213. ISSN: 0018-9340.

[BC95] Bryant, R. E. and Y. Chen. "Verification of Arithmetic Circuits with Binary Moment Diagrams." In: *DAC*. 1995, pp. 535–541.

[Bry+07] Bryant, R. E., D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman, and B. Brady. "Deciding Bit-Vector Arithmetic with Abstraction." In: *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2007*. Vol. 4424. LNCS. Springer, 2007, pp. 358–372.

[CDE08] Cadar, C., D. Dunbar, and D. R. Engler. "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs." In: *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*. 2008, pp. 209–224.

[Cad+08] Cadar, C., V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. "EXE: Automatically Generating Inputs of Death." In: *ACM Trans. Inf. Syst. Secur.* 12.2 (2008).

[Cha+16] Chalupa, M., M. Jonáš, J. Slaby, J. Strejcek, and M. Vitovská. "Symbiotic 3: New Slicer and Error-Witness Generation - (Competition Contribution)." In: *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*. 2016, pp. 946–949.

[Chl84] Chlebus, B. S. "From domino tilings to a new model of computation." In: *Computation Theory - Fifth Symposium, Zaborów, Poland, December 3-8, 1984, Proceedings*. 1984, pp. 24–33.

[CHN12] Christ, J., J. Hoenicke, and A. Nutz. "SMTInterpol: An Interpolating SMT Solver." In: *Model Checking Software - 19th International Workshop, SPIN 2012, Oxford, UK, July 23-24, 2012. Proceedings*. 2012, pp. 248–254.

[CGS07] Cimatti, A., A. Griggio, and R. Sebastiani. "A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories." In: *Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*. 2007, pp. 334–339.

[Cim+13] Cimatti, A., A. Griggio, B. Schaafsma, and R. Sebastiani. "The MathSAT5 SMT Solver." In: *Proceedings of TACAS*. Ed. by Piterman, N. and S. Smolka. Vol. 7795. LNCS. Springer, 2013.

[CFM12] Cordeiro, L. C., B. Fischer, and J. Marques-Silva. "SMT-Based Bounded Model Checking for Embedded ANSI-C Software." In: *IEEE Trans. Software Eng.* 38.4 (2012), pp. 957–974.

[Cot10] Cotton, S. "Natural Domain SMT: A Preliminary Assessment." In: *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*. 2010, pp. 77–91.

[CC79] Cousot, P. and R. Cousot. "Systematic Design of Program Analysis Frameworks." In: *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979*. 1979, pp. 269–282.

[CMR97] Cyrluk, D., M. O. Möller, and H. Ruess. "An Efficient Decision Procedure for the Theory of Fixed-Sized Bit-Vectors." In: *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*. 1997, pp. 60–71.

[DHK13] D'Silva, V., L. Haller, and D. Kroening. "Abstract conflict driven learning." In: *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*. 2013, pp. 143–154.

[DHK14] D'Silva, V., L. Haller, and D. Kroening. "Abstract satisfaction." In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. 2014, pp. 139–150.

[Dan63] Dantzig, G. B. *Linear programming and extensions*. Rand Corporation Research Study. Princeton, NJ: Princeton Univ. Press, 1963. XVI, 625.

[DP09] Darwiche, A. and K. Pipatsrisawat. "Complete Algorithms." In: *Handbook of Satisfiability*. 2009, pp. 99–130.

[DLL62] Davis, M., G. Logemann, and D. W. Loveland. "A machine program for theorem-proving." In: *Commun. ACM* 5.7 (1962), pp. 394–397.

[DP60] Davis, M. and H. Putnam. "A Computing Procedure for Quantification Theory." In: *J. ACM* 7.3 (1960), pp. 201–215.

[DNS05] Detlefs, D., G. Nelson, and J. B. Saxe. "Simplify: a theorem prover for program checking." In: *J. ACM* 52.3 (2005), pp. 365–473.

[DB05] Domagoj Babic, M. M. *Modular Arithmetic Decision Procedure*. Tech. rep. 2005.

[Dut14] Dutertre, B. "Yices 2.2." In: *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. 2014, pp. 737–744.

[Dut15] Dutertre, B. "Solving Exists/Forall Problems With Yices." In: *13th International Workshop on Satisfiability Modulo Theories (SMT 2015)*. 2015.

[ES03] Eén, N. and N. Sörensson. "An Extensible SAT-solver." In: *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*. 2003, pp. 502–518.

[End01] Enderton, H. B. *A Mathematical Introduction to Logic*. Harcourt/Academic Press, 2001.

[Fla+03] Flanagan, C., R. Joshi, X. Ou, and J. B. Saxe. "Theorem Proving Using Lazy Proof Explication." In: *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*. 2003, pp. 355–367.

[Fra10] Franzén, A. "Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT." PhD thesis. University of Trento, 2010.

[FKB13] Fröhlich, A., G. Kovásznai, and A. Biere. "More on the Complexity of Quantifier-Free Fixed-Size Bit-Vector Logics with Binary Encoding." In: *Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings*. 2013, pp. 378–390.

[Frö+15] Fröhlich, A., A. Biere, C. M. Wintersteiger, and Y. Hamadi. "Stochastic Local Search for Satisfiability Modulo Theories." In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. 2015, pp. 1136–1143.

[GD07] Ganesh, V. and D. L. Dill. "A Decision Procedure for Bit-Vectors and Arrays." In: *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*. 2007, pp. 519–531.

[GM09] Ge, Y. and L. M. de Moura. "Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories." In: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*. 2009, pp. 306–320.

[GSK98] Gomes, C. P., B. Selman, and H. A. Kautz. "Boosting Combinatorial Search Through Randomization." In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*. 1998, pp. 431–437.

[GLS11] Griggio, A., T. T. H. Le, and R. Sebastiani. "Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic." In: *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. 2011, pp. 143–157.

[Had15] Hadarean, L. "An Efficient and Trustworthy Theory Solver for Bit-vectors in Satisfiability Modulo Theories." PhD thesis. New York University, 2015.

[Had+14] Hadarean, L., K. Bansal, D. Jovanovic, C. Barrett, and C. Tinelli. "A Tale of Two Solvers: Eager and Lazy Approaches to Bit-Vectors." In: *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. 2014, pp. 680–695.

[Han+15] Hannula, M., J. Kontinen, J. Virtema, and H. Vollmer. "Complexity of Propositional Independence and Inclusion Logic." In: *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*. 2015, pp. 269–280.

[Har09] Harrison, J. *Handbook of Practical Logic and Automated Reasoning*. 1st. Cambridge University Press, 2009.

[Hei+13] Heizmann, M., J. Christ, D. Dietsch, E. Ermis, J. Hoenicke, M. Lindenmann, A. Nutz, C. Schilling, and A. Podelski. "Ultimate Automizer with SMTInterpol - (Competition Contribution)." In: *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*. 2013, pp. 641–643.

[Hei+14] Heizmann, M., J. Christ, D. Dietsch, J. Hoenicke, M. Lindenmann, B. Musa, C. Schilling, S. Wissert, and A. Podelski. "Ultimate Automizer with Unsatisfiable Cores - (Competition Contribution)." In: *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*. 2014, pp. 418–420.

[HKM16] Heule, M. J. H., O. Kullmann, and V. W. Marek. "Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer." In: *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. 2016, pp. 228–245.

[HM09] Heule, M. and H. van Maaren. "Look-Ahead Based SAT Solvers." In: *Handbook of Satisfiability*. 2009, pp. 155–184.

[Heu+11] Heule, M., O. Kullmann, S. Wieringa, and A. Biere. "Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads." In: *Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC 2011, Haifa, Israel, December 6-8, 2011, Revised Selected Papers*. 2011, pp. 50–65.

[HKB96] Hojati, R., S. C. Krishnan, and R. K. Brayton. "Early Quantification and Partitioned Transition Relations." In: *1996 International Conference on Computer Design (ICCD '96), VLSI in Computers and Processors, October 7-9, 1996, Austin, TX, USA, Proceedings*. 1996, pp. 12–19.

[HD04] Huang, J. and A. Darwiche. "Toward Good Elimination Orders for Symbolic SAT Solving." In: *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*. 2004, pp. 566–573.

[Hut+07] Hutter, F., D. Babic, H. H. Hoos, and A. J. Hu. "Boosting Verification by Automatic Tuning of Decision Procedures." In: *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings*. 2007, pp. 27–34.

[JLS09] Jha, S., R. Limaye, and S. A. Seshia. "Beaver: Engineering an Efficient SMT Solver for Bit-Vector Arithmetic." In: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*. 2009, pp. 668–674.

[JS04] Jin, H. and F. Somenzi. "CirCUs: A Hybrid Satisfiability Solver." In: *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*. 2004.

[JS16] Jonáš, M. and J. Strejcek. "Solving Quantified Bit-Vector Formulas Using Binary Decision Diagrams." In: *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. 2016, pp. 267–283.

[JB10] Jovanovic, D. and C. Barrett. "Polite Theories Revisited." In: *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings*. 2010, pp. 402–416.

[JBM13] Jovanović, D., C. Barrett, and L. de Moura. "The design and implementation of the model constructing satisfiability calculus." In: *Formal Methods in Computer-Aided Design (FMCAD), 2013*. 2013, pp. 173–180.

[KSMS11] Katebi, H., K. A. Sakallah, and J. P. Marques-Silva. "Empirical Study of the Anatomy of Modern Sat Solvers." In: *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*. 2011, pp. 343–356.

[KFB12] Kovásznai, G., A. Fröhlich, and A. Biere. "On the Complexity of Fixed-Size Bit-Vector Logics with Binary Encoded Bit-Width." In: *10th International Workshop on Satisfiability Modulo Theories, SMT 2012, Manchester, UK, June 30 - July 1, 2012*. 2012, pp. 44–56.

[KFB16] Kovásznai, G., A. Fröhlich, and A. Biere. "Complexity of Fixed-Size Bit-Vector Logics." In: *Theory Comput. Syst.* 59.2 (2016), pp. 323–376.

[KLR10] Kroening, D., J. Leroux, and P. Rümmer. "Interpolating Quantifier-Free Presburger Arithmetic." In: *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings*. 2010, pp. 489–503.

[KLW13] Kroening, D., M. Lewis, and G. Weissenbacher. "Under-Approximating Loops in C Programs for Fast Counterexample Detection." In: *Computer Aided Verification - 25th International Conference, CAV 2013*. Vol. 8044. LNCS. Springer, 2013, pp. 381–396.

[KS08] Kroening, D. and O. Strichman. *Decision Procedures - An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2008. ISBN: 978-3-540-74104-6.

[LS04] Lahiri, S. K. and S. A. Seshia. "The UCLID Decision Procedure." In: *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*. 2004, pp. 475–478.

[Lei10] Leino, K. R. M. "Dafny: An Automatic Program Verifier for Functional Correctness." In: *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*. 2010, pp. 348–370.

[Lia+16] Liang, J. H., V. Ganesh, P. Poupart, and K. Czarnecki. "Learning Rate Based Branching Heuristic for SAT Solvers." In: *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. 2016, pp. 123–140.

[LSZ93] Luby, M., A. Sinclair, and D. Zuckerman. "Optimal Speedup of Las Vegas Algorithms." In: *Inf. Process. Lett.* 47.4 (1993), pp. 173–180.

[Lüc16] Lück, M. "Complete Problems of Propositional Logic for the Exponential Hierarchy." In: *CoRR* abs/1602.03050 (2016).

[MSS99] Marques-Silva, J. P. and K. A. Sakallah. "GRASP: A Search Algorithm for Propositional Satisfiability." In: *IEEE Trans. Computers* 48.5 (1999), pp. 506–521.

[McM05] McMillan, K. L. "An interpolating theorem prover." In: *Theor. Comput. Sci.* 345.1 (2005), pp. 101–121.

[McM06] McMillan, K. L. "Lazy Abstraction with Interpolants." In: *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*. 2006, pp. 123–136.

[McM11] McMillan, K. L. "Interpolants from Z3 proofs." In: *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*. 2011, pp. 19–27.

[Min93] Minato, S. "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems." In: *DAC*. 1993, pp. 272–277.

[Mon16] Monniaux, D. "A Survey of Satisfiability Modulo Theory." In: *CoRR* abs/1606.04786 (2016).

[Mos+01] Moskewicz, M. W., C. F. Madigan, Y. Zhao, L. Zhang, and S. 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*. 2001, pp. 530–535.

[MB08a] Moura, L. M. de and N. Bjørner. "Model-based Theory Combination." In: *Electr. Notes Theor. Comput. Sci.* 198.2 (2008), pp. 37–49.

[MB08b] Moura, L. M. de and N. Bjørner. "Z3: An Efficient SMT Solver." In: *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*. 2008, pp. 337–340.

[MJ13] Moura, L. de and D. Jovanović. "A Model-Constructing Satisfiability Calculus." In: *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*. 2013, pp. 1–12.

[NO79] Nelson, G. and D. C. Oppen. "Simplification by Cooperating Decision Procedures." In: *ACM Trans. Program. Lang. Syst.* 1.2 (1979), pp. 245–257.

[NPB14] Niemetz, A., M. Preiner, and A. Biere. "Boolector 2.0 system description." In: *Journal on Satisfiability, Boolean Modeling and Computation* 9 (2014 (published 2015)), pp. 53–58.

[NPB16] Niemetz, A., M. Preiner, and A. Biere. "Precise and Complete Propagation Based Local Search for Satisfiability Modulo Theories." In: *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*. 2016, pp. 199–217.

[Nie+15] Niemetz, A., M. Preiner, A. Biere, and A. Fröhlich. "Improving Local Search For Bit-Vector Logics in SMT with Path Propagation." In: *Proc. 4th Intl. Work. on Design and Implementation of Formal Tools and Systems (DIFTS'15)*. 2015, 10 pages.

[NO05] Nieuwenhuis, R. and A. Oliveras. "Decision Procedures for SAT, SAT Modulo Theories and Beyond. The Barcelogic-Tools." In: *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*. 2005, pp. 23–46.

[NOT06] Nieuwenhuis, R., A. Oliveras, and C. Tinelli. "Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)." In: *J. ACM* 53.6 (2006), pp. 937–977.

[PV04] Pan, G. and M. Y. Vardi. "Search vs. Symbolic Techniques in Satisfiability Solving." In: *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*. 2004.

[PVL11] Peleska, J., E. Vorobev, and F. Lapschies. "Automated Test Case Generation with SMT-Solving and Abstract Interpretation." In: *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*. 2011, pp. 298–312.

[Pud97] Pudlák, P. "Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations." In: *J. Symb. Log.* 62.3 (1997), pp. 981–998.

[RRZ05] Ranise, S., C. Ringeissen, and C. G. Zarba. "Combining Data Structures with Nonstably Infinite Theories Using Many-Sorted Logic." In: *Frontiers of Combining Systems, 5th International Workshop, FroCoS 2005, Vienna, Austria, September 19-21, 2005, Proceedings*. 2005, pp. 48–64.

[RB16] Reynolds, A. and J. C. Blanchette. "A Decision Procedure for (Co)datatypes in SMT Solvers." In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. 2016, pp. 4205–4209.

[Rey+15] Reynolds, A., M. Deters, V. Kuncak, C. Tinelli, and C. W. Barrett. "Counterexample-Guided Quantifier Instantiation for Synthesis in SMT." In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*. 2015, pp. 198–216.

[Rob65] Robinson, J. A. "A Machine-Oriented Logic Based on the Resolution Principle." In: *J. ACM* 12.1 (1965), pp. 23–41.

[Rud93] Rudell, R. "Dynamic variable ordering for ordered binary decision diagrams." In: *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993*. 1993, pp. 42–47.

[Seb07] Sebastiani, R. "Lazy Satisfiability Modulo Theories." In: *JSAT* 3.3-4 (2007), pp. 141–224.

[TZ05] Tinelli, C. and C. G. Zarba. "Combining Nonstably Infinite Theories." In: *J. Autom. Reasoning* 34.3 (2005), pp. 209–238.

[Tse68] Tseitin, G. S. "On the complexity of derivations in the propositional calculus." In: *Studies in Mathematics and Mathematical Logic* Part II (1968), pp. 115–125.

[WHM13] Wintersteiger, C. M., Y. Hamadi, and L. M. de Moura. "Efficiently solving quantified bit-vector formulas." In: *Formal Methods in System Design* 42.1 (2013), pp. 3–23.

[ZM88] Zabih, R. and D. A. McAllester. "A Rearrangement Search Strategy for Determining Propositional Satisfiability." In: *Proceedings of the 7th National Conference on Artificial Intelligence. St. Paul, MN, August 21-26, 1988*. 1988, pp. 155–160.

[ZWR16] Zeljic, A., C. M. Wintersteiger, and P. Rümmer. "Deciding Bit-Vector Formulas with mcSAT." In: *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. 2016, pp. 249–266.

[ZZG13] Zheng, Y., X. Zhang, and V. Ganesh. "Z3-str: a z3-based string solver for web application analysis." In: *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*. 2013, pp. 114–124.

A

SUMMARY OF AUTHOR'S PH.D. STUDY

A.1 PRESENTATIONS

I have presented the paper *Solving Quantified Bit-vector Formulas Using Binary Decision Diagrams* at the conference SAT 2016, July 2016, Bordeaux, France.

A.2 OTHER CONFERENCES

- SMT Workshop 2015 (affiliated with CAV 2015, July 2015, San Francisco)
- MEMICS 2015, October 2015, Telč, Czech Republic
- TACAS 2016, April 2016, Eindhoven, The Netherlands

A.3 TEACHING

At my faculty, I have tutored seminar groups for the following courses:

- Automata, Grammars, and Complexity (2014–now)
- Formal Languages and Automata (2016–now)
- Non-Imperative Programming (2015–now)

I have also been an official reader of the following bachelor theses:

- Jan Mrázek – Caching SMT Queries in SymDivine
- Jakub Lédl – Many-sorted equational logic

A.4 SCHOOLS

- SAT/SMT Summer School, Stanford University, July 15–17, 2015
- RiSE & LogiCS Spring School on Logic and Verification, TU Wien, April 15–17, 2016

A.5 PASSED COURSES

Besides the courses that are mandatory for the PhD study, I have passed the following courses:

- Academic Writing in English (spring 2016)
- Academic and professional skills in English for IT (spring 2015)
- C++ Programming (autumn 2014)
- Exercises in Category Theory (autumn 2014)
- Formal Methods in Theory and Practice (autumn 2014, spring 2015, autumn 2015, spring 2016)
- GEB – limits of formal systems (autumn 2014)
- Modeling and Simulation (spring 2015)
- Probability in Computer Science (spring 2015)
- Selected problems of present-day written Czech (spring 2015)
- Seminar on Concurrency (autumn 2014, spring 2015, autumn 2015, spring 2016)
- Teaching Lab (spring 2016)

B

PUBLICATIONS

Solving Quantified Bit-Vector Formulas Using Binary Decision Diagrams*

Martin Jonáš and Jan Strejček

Faculty of Informatics, Masaryk University, Brno, Czech Republic
{martin.jonas, strejcek}@mail.muni.cz

Abstract. We describe a new approach to deciding satisfiability of quantified bit-vector formulas using binary decision diagrams and approximations. The approach is motivated by the observation that the binary decision diagram for a quantified formula is typically significantly smaller than the diagram for the subformula within the quantifier scope. The suggested approach has been implemented and the experimental results show that it decides more benchmarks from the SMT-LIB repository than state-of-the-art SMT solvers for this theory, namely Z3 and CVC4.

1 Introduction

During the last decades, the area of *Satisfiability Modulo Theories* (SMT) [6] solving has undergone steep development in both theory and practice. Achieved advances of SMT solving opened new research directions in program analysis and verification, where SMT solvers are now seen as standard tools.

Common programming languages provide basic datatypes of fixed size. Program variables of these datatypes naturally correspond to variables of the bit-vector logic, which can easily express bit-wise operations or arithmetic overflows. In spite of this natural correspondence, most SMT-based program analysis techniques model program variables by variables in the theory of integers. This may look a bit strange considering the fact that the satisfiability problem for the theory of integers is undecidable whenever an arbitrary usage of addition and multiplication is allowed, while the same problem is decidable for the bit-vector theory. The reasons for using the integer logic instead of the bit-vector logic are twofold. First, the satisfiability problem is NEXPTIME-complete even for formulas of the *quantifier-free fragment of the bit-vector logic* (QF_BV) with binary encoding of bit-vector sizes [21]. In this paper, we consider formulas with quantifiers and without uninterpreted functions. The precise complexity of the problem for this logic is an open question: it is known to be NEXPTIME-hard [21] and trivially solvable in EXPSPACE. Second and from the practical point of view more important, the SMT solvers for the theory of integers are often more efficient than the solvers for the theory of fixed-size bit-vectors.

While there are several SMT solvers for QF_BV formulas, only few of them can decide the *quantified bit-vector (*BV*) logic*. In particular, the logic is supported by CVC4 [3] and Z3 [16]. Relatively modest support of this logic from

* The research was supported by Czech Science Foundation, grant GBP202/12/G061.

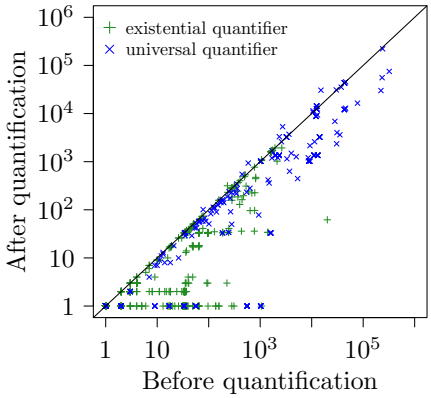


Fig. 1: Comparison of sizes (measured by the number of BDD nodes) of BDDs corresponding to all quantified subformulas in SMT-LIB benchmarks for BV logic, before and after quantification.

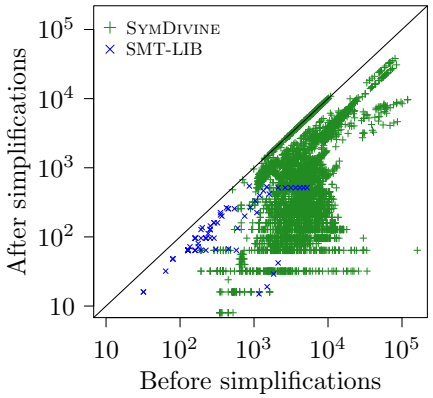


Fig. 2: Effect of simplifications on the number of bit variables in the formulas of the SMT-LIB and SymDivine benchmarks. Formulas simplified to *true* or *false* are not represented.

developers of SMT solvers is definitely not a consequence of a low demand from potential users. For example, in the program analysis community, BV formulas are suitable for description of various properties of program loops like loop invariants, ranking functions, or loop summaries [22], or to describe properties of symbolic representations of sets of program states, such as inclusion [7].

While current solvers for BV logic rely on *model-based quantifier instantiation* [25], we present a new algorithm based on *Binary Decision Diagrams* (BDDs) and approximations. BDDs have been previously used to implement satisfiability decision procedures for the propositional logic, however state-of-the-art CDCL-based solvers usually achieve much better performance. The main disadvantage of BDDs is low scalability: the size of a BDD corresponding to a propositional formula can be exponential in the number of propositional variables, and when a BDD becomes too large, some operations are very slow. Employment of BDDs in SMT solving makes more sense when formulas with quantifiers are considered: quantification usually reduces size of a BDD as it decreases the number of BDD variables. This can be documented by Figure 1, which compares the BDD sizes for formulas before and after existential or universal quantification.

There already exist some BDD-based tools deciding validity of quantified boolean formulas with the performance similar to state-of-the-art solvers for this problem [2, 23].

Our BDD-based algorithm for satisfiability of the BV logic consists of three main components:

– Formula simplifications, which reduce the number of variables in the formula and push quantifiers downwards in the syntax tree of the formula (which later helps to keep intermediate BDDs smaller as they are build in the bottom-up

manner). Formula simplifications can reduce some formulas to *true* or *false* and thus immediately decide their satisfiability.
- Construction of BDD using a specific variable ordering. The ordering has a significant influence on the BDD size.
- Formula approximations, which reduce the width of bit-vector variables in the formula and thus lead to smaller BDDs. Unsatisfiability of a formula over-approximation implies unsatisfiability of the original formula and an analogous statement holds for satisfiability of an under-approximation.

We present a minor contribution in each component. The main contribution of the paper is the fact that the algorithm based on the three parts can compete with leading SMT solvers for the BV logic, which participated in the BV category of SMT-COMP 2015 [1], namely Z3 and CVC4.

In the next section, we recall the definition of BV logic and BDDs, and briefly explain the main idea of the model-based quantifier instantiation technique employed by CVC4 and Z3. The proposed algorithm including the three main components is presented in Section 3. Section 4 is devoted to the implementation of the algorithm and to experimental results showing separately the effect of formula simplification, variable ordering, and approximations. The section also provides an experimental comparison of our solver with Z3 and CVC4. The paper closes with conclusions and intended directions of future work.

2 Preliminaries

2.1 Quantified Bit-Vector Formulas

In what follows, we assume a knowledge of the multi-sorted first-order logic and the model theory [18, 19]. Let $\mathbb{N}$ denote the set of positive integers.

The *bit-vector logic* is a multi-sorted first-order logic with the set of sort symbols $S = \{\texttt{bitvec}_i \mid i \in \mathbb{N}\}$, where $\texttt{bitvec}_i$ represents the sort of bit-vectors of length i, the set of function symbols

$$\begin{aligned} F = &\{c^i_{[n]} \mid n, i \in \mathbb{N}\} \cup \bigcup_{n\in\mathbb{N}} \{0_{[n]}, 1_{[n]}, \ldots, (2^n-1)_{[n]}\} \cup \\ &\cup \bigcup_{n\in\mathbb{N}} \{\texttt{not}_{[n]}, \texttt{and}_{[n]}, \texttt{or}_{[n]}, \texttt{shl}_{[n]}, \texttt{shr}_{[n]}, -_{[n]}, +_{[n]}, \times_{[n]}, /_{[n]}, \%_{[n]}\} \cup \\ &\cup \{\texttt{concat}_{[m,n]} \mid m, n \in \mathbb{N}\} \cup \{\texttt{extract}_{[n,i,j]} \mid n, i, j \in \mathbb{N}, i \leq j < m\}, \end{aligned}$$

and the set of the predicate symbols $P = \{=_{[n]}, <_{[n]} \mid n \in \mathbb{N}\}$. Arities of function and predicate symbols are described in Table 1.

The syntax of bit-vector formulas is defined in the standard way. Every formula can be transformed into the *negation normal form* (NNF), where negation is applied only to atomic subformulas and implication is not used at all.

A structure M is said to be a *model* for formula φ, if the formula φ is true in M and if M interprets all function and predicate symbols according to Table 1.

Symbol	Arity	Interpretation
$0_{[n]}, 1_{[n]}, \ldots$	$\texttt{bitvec}_n$	natural number constants
$c^1_{[n]}, c^2_{[n]}, \ldots$	$\texttt{bitvec}_n$	uninterpreted constants
$\texttt{not}_{[n]}$	$\texttt{bitvec}_n \to \texttt{bitvec}_n$	bit-wise negation
$\texttt{and}_{[n]}, \texttt{or}_{[n]}$	$\texttt{bitvec}_n \times \texttt{bitvec}_n \to \texttt{bitvec}_n$	bit-wise and, or
$\texttt{shl}_{[n]}, \texttt{shr}_{[n]}$	$\texttt{bitvec}_n \times \texttt{bitvec}_n \to \texttt{bitvec}_n$	bit-wise shift left, right
$-_{[n]}$	$\texttt{bitvec}_n \to \texttt{bitvec}_n$	two-complement negation
$+_{[n]}, \times_{[n]}$	$\texttt{bitvec}_n \times \texttt{bitvec}_n \to \texttt{bitvec}_n$	addition, multiplication
$/_{[n]}, \%_{[n]}$	$\texttt{bitvec}_n \times \texttt{bitvec}_n \to \texttt{bitvec}_n$	unsigned division, remainder
$\texttt{concat}_{[m,n]}$	$\texttt{bitvec}_m \times \texttt{bitvec}_n \to \texttt{bitvec}_{m+n}$	concatenation
$\texttt{extract}_{[m,i,j]}$	$\texttt{bitvec}_m \to \texttt{bitvec}_{j-i+1}$	extraction from i-th to j-th bit
$=_{[n]}, <_{[n]}$	$\texttt{bitvec}_n \times \texttt{bitvec}_n$	equality, unsigned less than

Table 1: Function and predicate symbols of the bit-vector logic.

Precise description of function and predicate symbols interpretation can be found in [4]. A closed formula is said to be *satisfiable* if it has a model.

We omit subscripts representing the sorts from the function and predicate symbols if the bit-width can be inferred from the context. If the sort of a variable or a constant is not specified, it is assumed to be $\texttt{bitvec}_{32}$. We also write $a, b, c, \ldots$ instead of uninterpreted constants $c^1, c^2, c^3, \ldots$. For example, $\forall x\,(x < a)$ denotes the formula $\forall x_{[32]}\,(x_{[32]} <_{[32]} c^1_{[32]})$. We write $\varphi[x_1, \ldots, x_n]$ for a formula φ, which may contain free variables $x_1, \ldots, x_n$. If $\varphi[x_1, \ldots, x_n]$ is a formula and $t_1, \ldots, t_n$ are terms of corresponding sorts, then $\varphi[t_1, \ldots, t_n]$ is the result of simultaneous substitution of free variables $x_1, \ldots, x_n$ in the formula φ by terms $t_1, \ldots, t_n$, respectively.

2.2 Model-Based Quantifier Instantiation

Satisfiability of the quantifier-free fragment of the bit-vector logic is traditionally solved by eager or lazy reduction to a propositional formula (bit-blasting) and subsequent call of a SAT solver. In the following, we describe the *model-based quantifier instantiation* algorithm [25], which is used by existing solvers for the full bit-vector logic.

Given a closed formula with quantifiers, the first step is to convert the formula to the negation normal form and apply Skolemization to obtain equisatisfiable formula of the form

$$\varphi \;\wedge\; \forall x_1, x_2, \ldots, x_n\,(\psi[x_1, \ldots, x_n]),$$

where φ and ψ are quantifier-free formulas. Then the QF_BV solver is invoked to check the satisfiability of the formula φ. If φ is unsatisfiable, then the entire formula is unsatisfiable. If φ is satisfiable, the QF_BV solver returns its model

M and another call to the QF_BV solver is made to determine whether M is also a model of $\forall x_1, x_2, \ldots, x_n\,(\psi)$. This is achieved by asking the solver whether the formula $\neg\widehat{\psi}$ is satisfiable, where $\widehat{\psi}$ is the formula ψ with uninterpreted constants replaced by their corresponding values in M. If $\neg\widehat{\psi}$ is not satisfiable, then the structure M is indeed a model of the formula $\forall x_1, x_2, \ldots, x_n\,(\psi)$, therefore the entire formula is satisfiable and M is its model. If $\neg\widehat{\psi}$ is satisfiable, we get values $v_1, \ldots, v_n$ such that $\neg\widehat{\psi}[v_1, \ldots, v_n]$ holds. To rule out M as a model, the instance $\psi[v_1, \ldots, v_n]$ of the quantified formula is added to the quantifier-free part, i.e. the formula φ is modified to

$$\varphi' \;\equiv\; \varphi \wedge \psi[v_1, \ldots, v_n],$$

and the procedure is repeated.

Example 1. *Consider the formula* $3 < a \;\wedge\; \forall x\,(\neg(a = 2 \times x))$. *The subformula* $3 < a$ *is satisfiable and* $a = 4$ *is its model. However, it is not a model of the formula* $\forall x\,(\neg(a = 2 \times x))$, *since the* QF_BV *solver called on the formula* $\neg(\neg(4 = 2 \times x))$ *returns* $x = 2$ *as a model. The next step is to decide the satisfiability of the formula* $3 < a \;\wedge\; \neg(a = 2 \times 2)$. *This formula is satisfiable and* $a = 5$ *is its model. Moreover, it is also a model of* $\forall x\,(\neg(a = 2 \times x))$ *as* $\neg(\neg(5 = 2 \times x))$ *is unsatisfiable. Hence, the input formula is satisfiable and* $a = 5$ *is its model.*

This algorithm is trivially terminating, since there is only a finite number of distinct models M of φ. However, in some cases exponentially many such models have to be ruled out before the solver is able to find a correct model or decide unsatisfiability of the whole formula. To overcome this issue, state-of-the-art SMT solvers do not use just instances of the form $\psi[v_1, \ldots, v_n]$ with concrete values, but employ heuristics such as E-matching [15, 17] or symbolic quantifier instantiation [25] to choose instances with ground terms which can potentially rule out more spurious models and thus significantly reduce the number of iterations of the algorithm. In practice, suitable ground terms substituted for quantified variables are selected only from subterms of the input formula. This strategy brings some drawbacks. For example, the formula

$$a = 2^4 \times b \;+\; 2^4 \times c \;\wedge\; \forall x\,(\neg(a = 2^4 \times x))$$

is unsatisfiable as the subformula $\forall x\,(\neg(a = 2^4 \times x))$ is true precisely when the value of a is not a multiple of 2^4, while $a = 2^4 \times b + 2^4 \times c$ implies that a is a multiple of 2^4. The quantifier instantiation can prove the unsatisfiability easily by using the instance $\psi[b+c]$ of $\psi[x] \equiv \neg(a = 2^4 \times x)$. However, the current tools do not consider this instance as $b+c$ is not a subterm of the formula. As a result, current tools can not decide satisfiability of this formula within a reasonable time limits.

2.3 Binary Decision Diagrams

A *binary decision diagram (BDD)* is a data structure proposed by Bryant [12] to succinctly represent all satisfying assignments of a boolean formula.

A BDD is a rooted directed acyclic graph with inner nodes labeled by boolean variables of the formula and two leaf nodes 0 and 1. Every inner node has two outgoing edges, one labeled with *true* and the other with *false*. Every assignment of boolean variables determines a path from the root to a leaf: from every inner node we follow the edge labeled with the truth value assigned to the variable corresponding to the node. The BDD represents all assignments that determine paths to leaf 1. Fixing an order in which variables can occur on paths from the root yields an *Ordered Binary Decision Diagram (OBDD)* and merging identical subgraphs of an OBDD and deleting every node whose two children are identical yields a *Reduced Ordered Binary Decision Diagram (ROBDD)*. The main advantage of ROBDDs is that for the fixed variable order every set of assignments corresponds to a unique ROBDD [12]. In the following, BDD always stands for ROBDD.

A BDD for a boolean formula can be built from BDDs for atomic subformulas in a bottom-up manner. Application of negation corresponds to switching the leaf nodes 0 and 1. For binary operators, there is a function `Apply` that gets an operator and two BDDs corresponding to the operands and produces the desired BDD. Using this function, one can also build a BDD representing a quantified boolean formula: if B is a BDD representing a formula φ, then the BDD for $\forall x\,(\varphi)$ is obtained by $\texttt{Apply}(\wedge, B[x \leftarrow \mathit{true}], B[x \leftarrow \mathit{false}])$ and the BDD for $\exists x\,(\varphi)$ by $\texttt{Apply}(\vee, B[x \leftarrow \mathit{true}], B[x \leftarrow \mathit{false}])$.

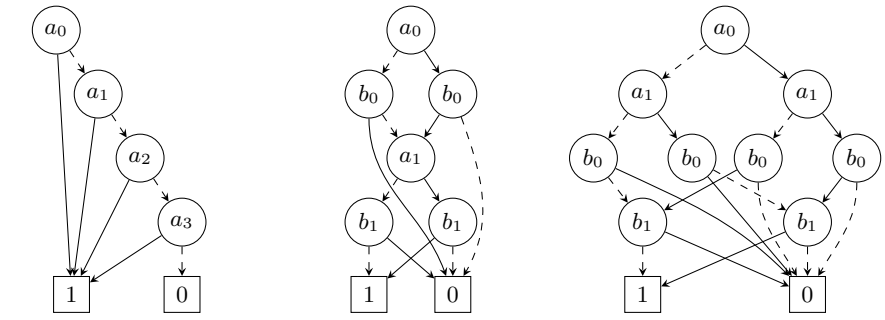
A BDD can also represent a set of all models of a BV formula. It is sufficient to decompose every bit-vector variable and every uninterpreted constant of bit-width n into n boolean variables and perform operations on individual bits. For example, all models of the formula $\forall x\,(\neg(a = 2^4 \times x))$ are represented by the BDD of Figure 3a, where the 32 bits of the uninterpreted constant a are denoted by boolean variables $a_0, a_1, \ldots, a_{31}$ in order from the least significant to the most significant bit. Boolean variables arising from bit-vector variables and uninterpreted constants are called *bit variables* henceforth. As usual, instead of labelling edges as *true* and *false*, edges are drawn as solid and dashed, respectively. Note that every unsatisfiable formula is represented by the BDD with the single node 0. By the BDD size we mean the number of its nodes.

3 Our Approach

This section first describes three main parts of our algorithm, namely formula simplifications, bit variable ordering for BDD construction, and approximations. Subsequently, the main algorithm is presented.

3.1 Formula Simplifications

As in most of modern SMT solvers, the first step of deciding satisfiability is simplification of the input formula. Besides trivial simplifications (e.g. $\varphi \wedge \varphi$ reduces to φ), we apply the following simplification rules.



(a) $\forall x\,(\neg(a = 2^4 \times x))$ (b) $a_{[2]} = b_{[2]}$ (c) $a_{[2]} = b_{[2]}$

Fig. 3: Examples of BDDs representing bit-vector formulas.

Miniscoping. Miniscoping [19] is a technique reducing the scope of universal quantifier over disjunctions whenever one disjunct has no free occurrences of the quantified variable, and over conjunctions by distributivity (existential quantifiers are handled analogously). The simplification rules are as follows:

$$\forall x\,(\varphi[x] \vee \psi) \;\rightsquigarrow\; \forall x\,(\varphi[x]) \vee \psi \qquad \forall x\,(\varphi[x] \wedge \psi[x]) \;\rightsquigarrow\; \forall x\,(\varphi[x]) \wedge \forall x\,(\psi[x])$$
$$\exists x\,(\varphi[x] \wedge \psi) \;\rightsquigarrow\; \exists x\,(\varphi[x]) \wedge \psi \qquad \exists x\,(\varphi[x] \vee \psi[x]) \;\rightsquigarrow\; \exists x\,(\varphi[x]) \vee \exists x\,(\psi[x])$$

Destructive Equality Resolution. Destructive equality resolution (DER) [25] eliminates a universally quantified variable x in a formula $\forall x\,\overline{Qy}\,(\neg(x = t) \vee \varphi[x])$, where t is a term that does not contain the variable x, and $\overline{Qy}$ is a sequence of variable quantifications. The formula is equivalent to $\forall x\,\overline{Qy}\,(x = t \rightarrow \varphi[x])$ and hence also to $\overline{Qy}\,(\varphi[t])$. The simplification rule is formulated as follows:

$$\forall x\,\overline{Qy}\,(\neg(x = t) \vee \varphi[x]) \quad\rightsquigarrow\quad \overline{Qy}\,(\varphi[t])$$

Constructive Equality Resolution. Constructive equality resolution (CER) is a dual version of DER. As far as we know, it was not considered before as solvers for quantified formulas typically work with formulas after Skolemization and thus without any existential quantifiers. CER can be formulated as the following simplification rule, where t and $\overline{Qy}$ have the same meaning as above:

$$\exists x\,\overline{Qy}\,(x = t \wedge \varphi[x]) \quad\rightsquigarrow\quad \overline{Qy}\,(\varphi[t])$$

Theory-Related Simplifications. We also perform several simplifications related to the interpretation of the function and predicate symbols in the BV logic. Examples of such simplifications are reductions $a_{[n]} + (-a_{[n]}) \;\rightsquigarrow\; 0_{[n]}$, $a_{[n]} \times 0_{[n]} \;\rightsquigarrow\; 0_{[n]}$, $a_{[n]}$ `and` $0_{[n]} \;\rightsquigarrow\; 0_{[n]}$, or $\texttt{extract}_{[n,i,j]}(0_{[n]}) \;\rightsquigarrow\; 0_{[j-i+1]}$.

Note that all mentioned simplification rules have no effect on models of the formula and thus they have no direct effect on the resulting BDD. However, a simplified formula has simpler subformulas and thus the intermediate BDDs are often smaller and the computation of the resulting BDD is faster.

3.2 Bit Variable Ordering

When constructing a BDD, one has to specify an order of BDD variables. In our case, BDD variables precisely correspond to bit variables. The order of these variables has a significant effect on the BDD size and its construction time. In some cases, the size of a BDD for a formula is linear in the number of BDD variables with one variable ordering, but exponential with another ordering.

For example, consider the formula $\phi_1 \equiv a_{[n]} = b_{[n]}$ for arbitrary $n \in \mathbb{N}$ and let $a_0, a_1, \ldots, a_{n-1}$ be the bits of a and $b_0, b_1, \ldots, b_{n-1}$ be the bits of b. We define two orderings:

$\leq_1$ All bit variables are ordered according to their significance (from the least to the most significant) and variables with the same significance are ordered by the order of the first occurrences of the corresponding bit-vector variables in the formula. For the considered formula ϕ_1, we get:

$$a_0 \leq_1 b_0 \leq_1 a_1 \leq_1 b_1 \leq_1 \ldots \leq_1 a_{n-1} \leq_1 b_{n-1}$$

$\leq_2$ Bit variables are ordered by the order of the first occurrences of the corresponding bit-vector variables in the formula and bit variables corresponding to the same bit-vector variable are ordered according to their significance (from the least to the most significant). For the considered formula, we get:

$$a_0 \leq_2 a_1 \leq_2 \ldots \leq_2 a_{n-1} \leq_2 b_0 \leq_2 b_1 \leq_2 \ldots \leq_2 b_{n-1}$$

The BDD for ϕ_1 using the ordering $\leq_1$ has $3n+2$ nodes, while the BDD for the same formula and $\leq_2$ has $3 \cdot 2^n - 1$ nodes. Figures 3b and 3c show these BDDs for $n = 2$ and orderings $\leq_1$ and $\leq_2$, respectively.

These orderings can lead to opposite results with other formulas. For example, the size of the BDD for the formula

$$\phi_2 \equiv (c^1_{[2]} = c^2_{[2]} \,\texttt{shr}\, 1_{[2]}) \wedge (c^3_{[2]} = c^4_{[2]} \,\texttt{shr}\, 1_{[2]}) \wedge \ldots \wedge (c^{2n-1}_{[2]} = c^{2n}_{[2]} \,\texttt{shr}\, 1_{[2]})$$

using the ordering $\leq_1$ is $2^{n+2} - 1$, while it is only $4n + 2$ for $\leq_2$. In general, choosing the optimal variable ordering is an NP-complete problem [10]. In the following, we introduce an ordering $\leq_3$ combining advantages of $\leq_1$ and $\leq_2$.

Let V be the set of bit-vector variables and uninterpreted constants appearing in an input formula φ. Elements $x, y \in V$ are *dependent*, written $x \sim y$, if they both appear in some atomic subformula of φ. Let $\simeq$ be the equivalence on V defined as the transitive closure of $\sim$. Every $v \in V$ then defines an equivalence class $[v]_\simeq$ of transitively dependent elements.

$\leq_3$ Bit variables are first ordered according to $\leq_1$ within corresponding equivalence classes of $\simeq$ and the equivalence classes are then ordered by the first occurrences of BV variables in φ. In particular for $u \not\simeq v$, $u_i \leq_3 v_j$ if there is a BV variable in $[u]_\simeq$, which occurs in φ before all BV variables of $[v]_\simeq$.

Note that for both formulas ϕ_1, ϕ_2 mentioned above, $\leq_3$ coincides with the better of the orderings $\leq_1$ and $\leq_2$.

In addition to the initial variable ordering, there are several techniques that dynamically reorder the BDD variables to reduce the BDD size. We use *sifting* [24] as usually the most successful one [20].

3.3 Approximations

For some BV formulas, e.g. formulas containing non-linear multiplication, the size of the BDD representation is exponential for every possible variable ordering [13]. Fortunately, satisfiability of these formulas can be often decided using their over-approximations or under-approximations. Given a formula φ, its *under-approximation* is any formula $\underline{\varphi}$ that logically entails φ, and its *over-approximation* is any formula $\overline{\varphi}$ logically entailed by φ. Clearly, every model of $\underline{\varphi}$ is also a model of φ and if an under-approximation $\underline{\varphi}$ is satisfiable, so is the formula φ. Similarly, if an over-approximation $\overline{\varphi}$ is unsatisfiable, so is φ.

The model-based quantifier instantiation presented in Section 2.2 can be seen as a technique based on iterative over-approximation refinement: the formulas φ, $\varphi \wedge \psi[\bar{v}]$, ... are over-approximations of $\varphi \wedge \forall \bar{x}\,(\psi[\bar{x}])$. A different concepts of approximations can be found in SMT solvers for QF_BV formulas. For example, the SMT solver UCLID over-approximates a formula in the negation normal form by replacing some subformulas with fresh uninterpreted constants [14]. Further, SMT solvers UCLID and Boolector under-approximate a formula by restricting the value of m most significant bits of a bit-vector variable while leaving the remaining bits unchanged [11,14]. The number of bit variables used to represent the bit-vector variable or uninterpreted constant is called its *effective bit-width*. This approach inspired both over- and under-approximation used in our algorithm.

Let $a_{[n]}$ be a variable or an uninterpreted constant of bit-width n and $e \in \mathbb{N}$ be its desired effective bit-width. If $e \geq n$, we leave $a_{[n]}$ unchanged. Otherwise, we consider four different ways to reduce the effective bit-width of $a_{[n]}$ to e:

zero-extension uses the effective bit-width to represent the e least significant bits and sets the $n - e$ most significant bits to 0.

sign-extension also uses the effective bit-width to represent the e least significant bits and sets the $n - e$ most significant bits to the value of the e-th least significant bit.

right zero-extension uses the effective bit-width to represent the e most significant bits and sets the $n - e$ least significant bits to 0.

right sign-extension also uses the effective bit-width to represent the e most significant bits and sets the $n - e$ least significant bits to the value of the e-th most significant bit.

All considered extensions are illustrated in Figure 4. The first two extensions are taken directly from [14], while the other two are original. One can easily see that each extension reduces the domain of $a_{[n]}$ to a different subdomain of size 2^e. Another extensions are suggested in [11], e.g. *one-extension* defined analogously to the zero-extension. Our choice of considered extensions is motivated by exploration of values near corner cases as well as by reduction of BDD size. In particular, we do not consider one-extension because it produces only few zero bits which are desired as they tend to reduce the size of BDDs for multiplication.

In the following, the term *extension* always refers either to zero-extension, or to sign-extension. In an over-approximation, we apply a selected reduction to all

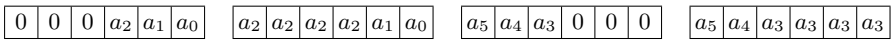


Fig. 4: Reductions of $a_{[6]} = a_5a_4a_3a_2a_1a_0$ to 3 effective bits.

universally quantified variables. Given a formula φ and $e \in \mathbb{N}$, let $\overline{\varphi}_e$ denote the formula φ where the effective bit-width of each universally quantified variable is reduced to e by the chosen extension. Further, $\overline{\varphi}_{-e}$ denotes the formula obtained by application of the right counterpart of the chosen extension.

In an under-approximation, we apply the selected reduction to all existentially quantified variables and uninterpreted constants. Given a formula φ and $e \in \mathbb{N}$, let $\underline{\varphi}_e$ and $\underline{\varphi}_{-e}$ denote the formula φ where the effective bit-width of each existentially quantified variable and uninterpreted constant is reduced to e by the chosen extension or its right counterpart, respectively.

The following theorem establishes that, for each formula φ in the negation normal form, $\overline{\varphi}_e$, $\overline{\varphi}_{-e}$ are over-approximations (and analogously for under-approximations). The theorem can be easily proven by an induction on the structure of the formula φ.

Theorem 1. *For every formula φ in the NNF and any $e \in \mathbb{N}$, it holds:*

1. *If M is a model of φ, then M is also a model of $\overline{\varphi}_e$ and $\overline{\varphi}_{-e}$.*
2. *If M is a model of $\underline{\varphi}_e$ or $\underline{\varphi}_{-e}$, then M is also a model of φ.*

Corollary 1. *For every formula φ in the NNF and any $e \in \mathbb{N}$, it holds:*

1. *If the formula $\overline{\varphi}_e$ or $\overline{\varphi}_{-e}$ is unsatisfiable, so is the formula φ.*
2. *If the formula $\underline{\varphi}_e$ or $\underline{\varphi}_{-e}$ is satisfiable, so is the formula φ.*

3.4 The Algorithm

In this section, we present the complete algorithm deciding satisfiability of BV formulas. In the algorithm, we use a procedure `ConvertToBDD` which converts a formula to the corresponding BDD recursively on the formula structure. For a given input formula φ, the algorithm proceeds in the following steps:

1. Simplify the formula φ using the rules discussed in Section 3.1 up to the fix-point and convert it to the negation normal form. If the result is *true*, return `SAT`. If the result is *false*, return `UNSAT`.
2. Take the simplified formula in NNF φ' and compute a chosen ordering $\leq$ as described in Section 3.2. This ordering will be used as the initial ordering in the procedure `ConvertToBDD`.
3. Call `ConvertToBDD`(φ') to compute the BDD corresponding to φ'. If the root node of the BDD has label 0, return `UNSAT`. Otherwise return `SAT`.
4. If the procedure `ConvertToBDD` called in the previous step has not finished within 0.1 seconds, additionally run in parallel:

(a) *Under-approximations*: Sequentially compute `ConvertToBDD`($\underline{\varphi'}_i$) for $i = 1, -1, 2, -2, 4, -4, 6, -6, \ldots$ until reaching the greatest bit-width of a bit-vector variable in φ'. If any of the resulting BDDs has a root node distinct from the leaf 0, return `SAT`.
(b) *Over-approximations*: Sequentially compute `ConvertToBDD`($\overline{\varphi'}_i$) for $i = 1, -1, 2, -2, 4, -4, 6, -6, \ldots$ until reaching the greatest bit-width of a bit-vector variable in φ'. If any of the produced BDDs has a root node labeled by 0, return `UNSAT`.

The algorithm is parametrized by the choice of an ordering and reductions for approximations. Regardless these parameters, the algorithm is sound and complete. The decision to start the solvers using approximations after 0.1 second is based on our experiments. In practice, the procedure `ConvertToBDD` may need exponential time and memory and thus the algorithm may not finish within reasonable limits.

4 Implementation and Experimental Results

We have implemented the presented algorithm in an experimental SMT solver called Q3B. The implementation is written in C++, relies on the BDD package BuDDy[1], and uses the API of Z3 to parse the input formula in the SMT-LIB 2.5 format [4] and to perform some formula simplifications. As the BuDDy package does not support allocation of multiple BDD instances, we run separate processes for the base solver and for computing over- and under-approximations. The execution of these three processes is controlled by a Python wrapper.

We have evaluated our solver on two sets of BV formulas. The first set consists of all 191 formulas in the category BV of the SMT-LIB benchmark repository [5]. The second set contains 5 461 formulas generated by the model checker SymDivine [7] when run on verification tasks from SV-COMP [8]. These formulas correspond to checking equivalence of two symbolic states of the verified program. In total, SymDivine generated 1 462 500 formulas. For tasks with more than 25 generated formulas, we randomly picked 25 formulas to keep the number of formulas reasonable.

All experiments were performed on a Debian machine with two six-core Intel Xeon E5-2620 2.00GHz processors and 128 GB of RAM. Each benchmark run was limited to use 3 processor cores, 4 GB of RAM and 20 minutes of CPU time (if not stated otherwise). All measured times are CPU times. For reliable benchmarking we employed BenchExec [9], a tool that allocates specified resources for a program execution and measures their use precisely.

All used benchmarks and detailed experimental results are available at `http://www.fi.muni.cz/~xstrejc/sat2016.tar.gz`. Q3B is available under the MIT License and hosted at GitHub: `https://github.com/martinjonas/Q3B`.

In the following, we demonstrate the effect of formula simplifications on the formulas and the effect of various algorithm parameters on its efficiency. At the end, we compare our solver with the best parameters against CVC4 and Z3.

[1] `http://sourceforge.net/projects/buddy`

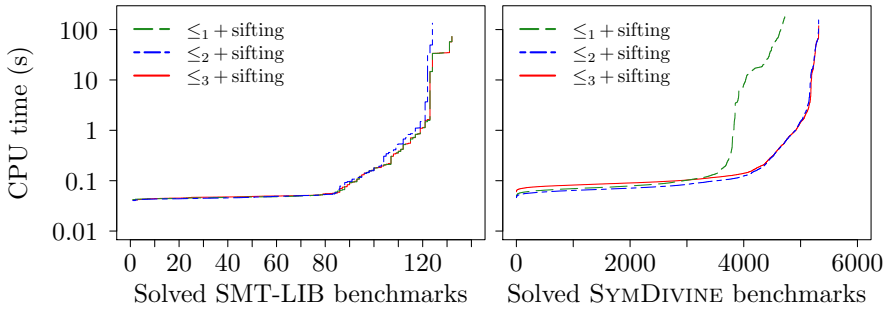


Fig. 5: Quantile plot of the number of solved benchmarks for each of three described initial variable orderings.

Formula Simplifications. Considered formula simplifications reduced 108 of 191 SMT-LIB benchmarks and 300 of 5 461 SymDivine benchmarks to *true* or *false*. Additionally, 1 276 SymDivine benchmarks were reduced to a quantifier-free formulas, which is not the case for any SMT-LIB benchmark. Figure 2 shows the number of bit variables (i.e. the sum of bit-widths of all bit-vector variables and uninterpreted constants in the formula) of each formula before and after simplification.

Variable Ordering. To compare the effect of BDD variable orderings $\leq_1$, $\leq_2$, and $\leq_3$ defined in Section 3.2, we run our tool with each of these initial orderings on all considered benchmarks. Recall that sifting method is used for dynamic variable reordering. The solver has been executed without approximations (to ensure that approximations will not hide the effect of the initial ordering) and with CPU time limited to 3 minutes. The results are shown in Figure 5.

When SMT-LIB are considered, the worst performing initial ordering is $\leq_2$. The results for $\leq_1$ and $\leq_3$ are almost identical as nearly all bit-vector variables in these benchmarks are mutually transitively dependent. For SymDivine benchmarks, initial ordering $\leq_1$ performs the worst. The results for $\leq_2$ and $\leq_3$ are very similar, as SymDivine formulas usually contain a large number of mutually independent groups of variables. The solver using $\leq_3$ decided 3 more formulas than the solver using $\leq_2$. To sum up, since now we always use the ordering $\leq_3$ as it provides better overall performance than $\leq_1$ and $\leq_2$.

Note that the solver runs usually faster when executed without sifting, as the dynamic reordering causes some computational overhead. However, with sifting it decides 2 more SMT-LIB benchmarks and 9 more SymDivine benchmarks.

Approximations. To compare the effect of the considered effective bit-width reductions, we run the solver once with approximations based on (right) zero-extension, and again with approximations based on (right) sign-extension. Quantile plots in Figure 6 show results for zero-extension and sign-extension on SMT-LIB benchmarks. The results are presented separately for satisfiable and unsat-

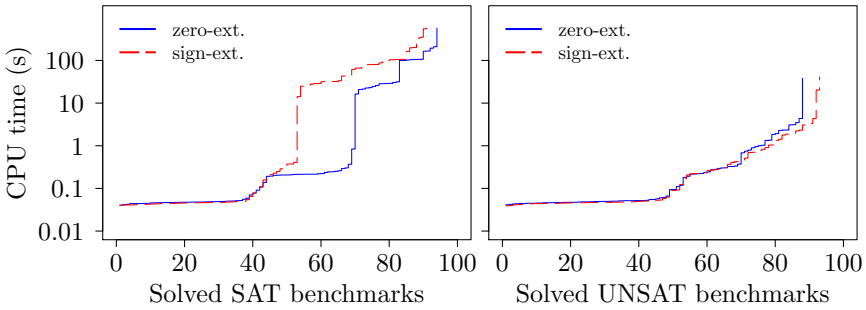


Fig. 6: Quantile plot of the number of solved SMT-LIB benchmarks using approximation via sign-extension and zero-extension compared by the CPU time.

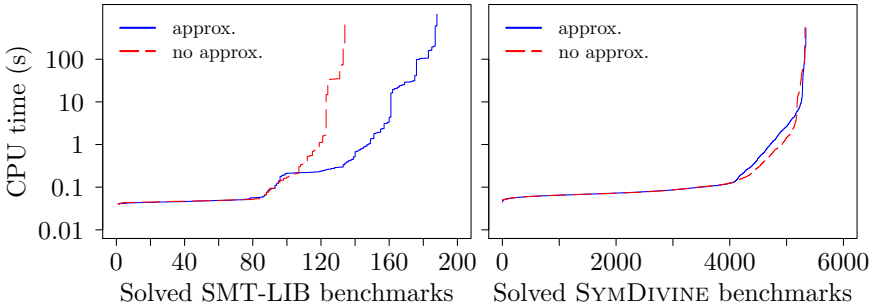


Fig. 7: Quantile plot of the number of solved benchmarks with and without approximations compared by the CPU time.

isfiable formulas. On satisfiable formulas, approximation using zero-extension performs better and can decide 3 more satisfiable formulas. On the contrary, on unsatisfiable formulas sign-extension performs better and can decide 5 formulas more. Corresponding plots for SymDivine formulas are not presented, since the difference in CPU times was insignificant. Based on this observation and the fact that satisfiability can be decided by an under-approximation and unsatisfiability by an over-approximation, the default bit-width reduction method in our solver is zero-extension for under-approximations and sign-extension for over-approximations.

Further, to show the contribution of approximations, we compare the solver using the proposed algorithm as described in the section 3.4 against the same algorithm without approximations. Figure 7 shows quantile plots corresponding to measured CPU times. With approximations, the solver was able to decide 54 more SMT-LIB formulas. The difference is less significant when SymDivine formulas are considered, as they mostly do not contain difficult arithmetic; only 15 more of SymDivine formulas were decided using approximations.

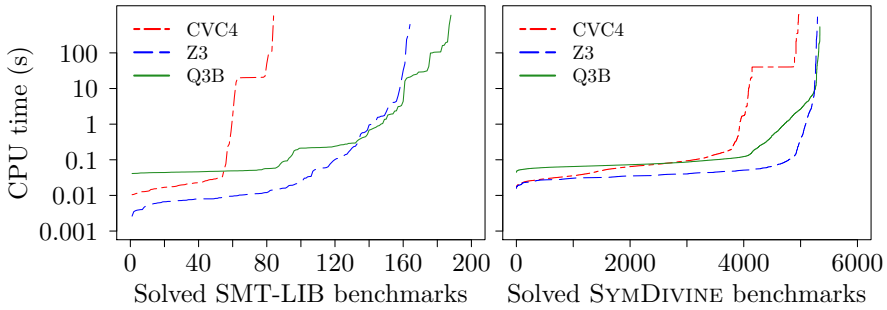


Fig. 8: Quantile plot of the number of benchmarks which CVC4, Q3B, and Z3 solved compared by the CPU time.

Comparison. Finally, we compare our solver (with the parameters selected by the previous experiments) to the current stable versions of leading SMT solvers for BV logic, namely to the version 4.4.1 of Z3 [16] and the version 1.4 of CVC4 [3]. We also tested the latest development version of CVC4 (2016-02-25), but it decided some SymDivine benchmarks incorrectly. The solver Z3 was executed with the default settings, CVC4 was executed with settings supplied for the SMT-competition, where the benchmarks from the SMT-LIB benchmark repository are used.

Table 2 shows summary results of the solvers CVC4, Z3, and Q3B on the two benchmark sets. Additionally, Table 3 shows for each pair of solvers the number of formulas which were decided by one solver, but not by the other one. Out of the 191 SMT-LIB benchmarks, CVC4 solves 84 benchmarks, Z3 decides 164 benchmarks, and our solver can decide 188 benchmarks. Out of the 5 461 SymDivine benchmarks, CVC4 decides 4 969 benchmarks, Z3 solves 5 297 benchmarks, and our solver Q3B decides 5 339 benchmarks. To sum up, in the number of decided benchmarks Q3B outperforms both CVC4 and Z3. Moreover, only 1 of all considered formulas was solved by Z3 and not by Q3B, and no formula was solved by CVC4 and not by Q3B.

Further, quantile plots of Figure 8 show numbers of input formulas each of the solvers was able to decide within different CPU time limits. Note that the y axis has the logarithmic scale. On the easy instances, our experimental solver can not compete with highly optimized solvers as CVC4 and Z3. The initial delay of Q3B is caused by an overhead of a process creation within the Python wrapper. However, as the instances become harder, the difference in solving times decreases. In particular, Q3B solves more SMT-LIB benchmarks than CVC4 whenever the CPU time limit is longer than 0.05 s and more than Z3 for any CPU time limit over 0.39 s. For SymDivine benchmarks, these thresholds are 0.08 s for CVC4 and 8.72 s for Z3. Note that Q3B uses 3 parallel processes and hence its wall times are usually three times shorter than presented CPU times, while wall times are the same as CPU times for Z3 and CVC4.

	SMT-LIB				SymDivine			
	sat	unsat	unknown	timeout	sat	unsat	unknown	timeout
CVC4	29	55	32	75	1 124	3 845	2	490
Z3	71	93	5	22	1 135	4 162	22	142
Q3B	94	94	0	3	1 137	4 202	0	122

Table 2: For each benchmark set and each solver, the table provides the numbers of formulas decided as satisfiable (*sat*), unsatisfiable (*unsat*), or undecided with the result unknown or because of an error (*unknown*), or a *timeout*.

	SMT-LIB			SymDivine		
	CVC4	Z3	Q3B	CVC4	Z3	Q3B
CVC4	–	0	0	–	21	0
Z3	80	–	1	349	–	0
Q3B	104	25	–	370	42	–

Table 3: For each pair of solvers, the table shows the number of benchmarks, which were solved by the solver in the corresponding row, but not by the solver in the corresponding column.

5 Conclusions

We presented a new SMT solving algorithm for quantified bit-vector formulas. While current SMT solvers for this logic typically rely on model-based quantifier instantiation and an SMT solver for quantifier-free bit-vector formulas, our algorithm is based on BDDs (with a specific initial variable ordering) and approximations. We have implemented the algorithm and experimental results indicate that our approach can compete with state-of-the-art SMT solvers CVC4 and Z3. In fact, it decides more formulas than the mentioned solvers.

We plan to further develop the algorithm and the tool. In particular, we plan to add a support for arrays and uninterpreted functions as these are useful for modelling some features of computer programs. We would also like to investigate possible approximations of bit-vector operations and predicates, or to develop some fine-grained methods for a targeted approximation refinement.

References

1. The 10th International Satisfiability Modulo Theories Competition (SMT-COMP 2015). `http://smtcomp.sourceforge.net/2015/`. 2015.
2. Gilles Audemard and Lakhdar Sais. SAT based BDD solver for quantified boolean formulas. In *16th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2004*, pages 82–89, 2004.
3. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification, CAV 2011*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.
4. Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.
5. Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.
6. Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. 2009.
7. Petr Bauch, Vojtěch Havel, and Jiří Barnat. LTL Model Checking of LLVM Bitcode with Symbolic Data. In *MEMICS 2014*, volume 8934 of *LNCS*, pages 47–59. Springer, 2014.
8. Dirk Beyer. Software verification and verifiable witnesses - (report on SV-COMP 2015). In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015*, volume 9035 of *LNCS*, pages 401–416. Springer, 2015.
9. Dirk Beyer, Stefan Löwe, and Philipp Wendler. Benchmarking and resource measurement. In *Model Checking Software - 22nd International Symposium, SPIN 2015, Proceedings*, pages 160–178, 2015.
10. Beate Bollig and Ingo Wegener. Improving the Variable Ordering of OBDDs Is NP-Complete. *Computers, IEEE Transactions on*, 45(9):993–1002, 1996.
11. Robert Brummayer and Armin Biere. Effective Bit-Width and Under-Approximation. In *Computer Aided Systems Theory, EUROCAST 2009*, volume 5717 of *LNCS*, pages 304–311. Springer, 2009.
12. Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.
13. Randal E. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication. *IEEE Trans. Comput.*, 40(2):205–213, 1991.
14. Randal E. Bryant, Daniel Kroening, Joël Ouaknine, Sanjit A. Seshia, Ofer Strichman, and Bryan Brady. Deciding Bit-Vector Arithmetic with Abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2007*, volume 4424 of *LNCS*, pages 358–372. Springer, 2007.
15. Leonardo de Moura and Nikolaj Bjørner. Efficient E-Matching for SMT Solvers. In *Automated Deduction, CADE-21*, volume 4603 of *LNCS*, pages 183–198. Springer, 2007.
16. Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
17. David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A Theorem Prover for Program Checking. *J. ACM*, 52(3):365–473, 2005.

18. Herbert B. Enderton. *A Mathematical Introduction to Logic*. Harcourt/Academic Press, 2001.
19. John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 1st edition, 2009.
20. Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 12th edition, 2009.
21. Gergely Kovásznai, Andreas Fröhlich, and Armin Biere. Complexity of Fixed-Size Bit-Vector Logics. *Theory of Computing Systems*, 7913:1–54, 2015.
22. Daniel Kroening, Matt Lewis, and Georg Weissenbacher. Under-approximating loops in C programs for fast counterexample detection. In *Computer Aided Verification - 25th International Conference, CAV 2013*, volume 8044 of *LNCS*, pages 381–396. Springer, 2013.
23. Oswaldo Olivo and E. Allen Emerson. A More Efficient BDD-Based QBF Solver. In Jimmy Lee, editor, *Principles and Practice of Constraint Programming, CP 2011*, volume 6876 of *LNCS*, pages 675–690. Springer, 2011.
24. Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993*, pages 42–47, 1993.
25. Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo de Moura. Efficiently solving quantified bit-vector formulas. *Formal Methods in System Design*, 42(1):3–23, 2013.