On Simplification of Formulas with Unconstrained Variables and Quantifiers*

Martin Jonáš and Jan Strejček

Masaryk University
Brno, Czech Republic
{xjonas,strejcek}@fi.muni.cz

Abstract. Preprocessing of the input formula is an essential part of all modern SMT solvers. An important preprocessing step is formula simplification. This paper elaborates on simplification of quantifier-free formulas containing unconstrained terms, i.e. terms that can have arbitrary values independently on the rest of the formula. We extend the idea in two directions. First, we introduce partially constrained terms and show some simplification rules employing this notion. Second, we show that unconstrained terms can be used also for simplification of formulas with quantifiers. Moreover, both these extensions can be merged in order to simplify partially constrained terms in formulas with quantifiers. We experimentally evaluate the proposed simplifications on formulas in the bit-vector theory.

1 Introduction

For most of the modern SMT solvers, preprocessing of the input formula is a crucial step for the efficiency of the solver. Therefore, modern SMT solvers employ hundreds of rewrite rules in order to simplify the input formula [10]. The aim of most of the simplifications is to reduce the size of the input formula and to replace expensive operations by easier ones. One class of these simplification rules focuses on formulas containing unconstrained variables. An unconstrained variable is a variable that occurs only once in the formula and therefore can be set to any suitable value without affecting the rest of the formula. For example, the formula $x + (5 * y + z) = y * z$ can be rewritten to an equisatisfiable formula $u = y * z$ because, regardless of the values of y and z, the term $x + (5 * y + z)$ can be evaluated to any value of u by choosing a suitable value of x. Such terms, which can be set to an arbitrary value by a well-suited choice of values of unconstrained variables, are called *unconstrained terms*. The principle of simplifications of unconstrained terms is recalled in more detail in Section 3. This simplification technique was proposed by Bruttomesso [8] and Brummayer [7], who independently observed that industrial benchmarks often contain non-trivial amount of unconstrained variables. For example, consider SMT queries coming from symbolic execution of a program, where a query is satisfiable if and only

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

if the symbolically executed program path is feasible. There are basically two sources of unconstrained variables in such queries. One source is input variables: such a variable is unconstrained in all queries corresponding to the symbolic execution of a path that reads the input variable at most once. The second source is program variables that are assigned on an executed path, but not read yet. For instance, the execution of an assignment `y:=x+5` leads to a conjunct $y = x + 5$ in the path condition query, where y does not appear anywhere else in the query (unless it is read) and thus it is unconstrained. Such situations are especially frequent when analyzing Static Single Assignment (SSA) code such as LLVM, which uses many program variables.

1.1 Contribution and Structure of the Paper

In this paper, we extend the notion of unconstrained terms in several ways:

– In some cases, the definition of unconstrained term is too restrictive by allowing only terms that can evaluate to every possible value by a suitable choice of values of unconstrained variables. For example, Bruttomesso and Brummayer describe the simplification rule that replaces the bit-vector term $c \cdot x$ by a fresh variable y, if x is an unconstrained variable and c is an odd constant. However, if c is even, the simplification is no longer possible. We describe a less restrictive simplification using *partially constrained terms*, which for example allows replacing the term $6 \cdot x$ by the term $2 \cdot y$; although these two terms can not evaluate to all possible values, they can evaluate to precisely the same set of values.
  Partially constrained terms are studied in Section 4. This section also shows that several *ad-hoc* simplification rules introduced by Bruttomesso can be seen as instances of simplification of partially constrained terms. Our definition of partially constrained terms allows construction of more similar rules.
– Previously, the simplifications of unconstrained terms were described only on quantifier-free formulas. In Section 5, we formalize the conditions under which a simplification of unconstrained terms can be performed on *quantified* formulas.
– Section 6 combines techniques from the two preceding sections and describes simplification of *partially constrained terms in quantified formulas*. Furthermore, the resulting technique is combined with quantifier-specific simplification rules to allow more efficient and straightforward applications.

Section 7 experimentally evaluates the influence of proposed simplifications on performance of state-of-the-art SMT solvers Z3 [9], Boolector [13], and Q3B [12] on quantified bit-vector formulas arising in software and hardware verification.

We emphasize that the presented approach is not tied to any particular theory. We use the bit-vector theory in many examples and in evaluation as its functions tend to produce unconstrained terms when at least one argument is an unconstrained.

2 Preliminaries

This section briefly recalls the *theory of fixed sized bit-vectors* (*BV* or *bit-vector theory* for short). It is a multi-sorted first-order theory with infinitely many sorts corresponding to bit-vectors of various lengths. The BV theory uses only three predicates, namely *equality* ($=$), *unsigned inequality* of binary-encoded natural numbers ($\leq_u$), and *signed inequality* of integers in two's complement representation ($\leq_s$). The theory also contains various functions, namely *addition* ($+$), *multiplication* ($\cdot$), *unsigned division* ($\div$), bit-wise *and* (`bvand`), bit-wise *or* (`bvor`), bit-wise *exclusive or* (`bvxor`), *left-shift* ($\ll$), *right-shift* ($\gg$), *concatenation* (`concat`), and *extraction* of n bits starting from position p ($\texttt{extract}_p^n$). The signature of BV theory also contains constants $c^{[n]}$ for each bit-width $n > 0$ and a number $0 \leq c \leq 2^n - 1$. Additionally, as in SMT-LIB [1] and in Hadarean [11], we suppose a distinguished sort *Boolean* and instead of treating formulas and terms differently, formulas are merely the terms of sort *Boolean*. This sort is similar to bit-vectors of length 1, but *Boolean* uses standard logic operators ($\wedge, \vee, \neg$) and not the bit-vector ones. If a bit-width of a constant or a variable is not specified, we suppose that it is equal to 32. The precise description of the multi-sorted logic can be found for example in Barrett et al. [3]. For a precise description of the syntax and semantics of the BV theory, we refer the reader to Hadarean [11].

For a valuation μ that assigns to each variable a value in its domain, $[\![\_]\!]_\mu$ denotes the evaluation function, which assigns to each formula φ the value obtained by substituting free variables in φ by values given by μ and evaluating all functions, predicates, logic operators etc. A formula φ is *satisfiable* if $[\![\varphi]\!]_\mu = \top$ for some valuation μ; it is *unsatisfiable* otherwise. Formulas φ and ψ are *equivalent* if they have the same set of free variables and for each valuation μ of these free variables, the equality $[\![\varphi]\!]_\mu = [\![\psi]\!]_\mu$ holds. Formulas φ and ψ are *equisatisfiable*, if either both are satisfiable, or both are unsatisfiable.

If φ is a formula and t, s are terms of the same sort, we use $\varphi[t \leftarrow s]$ to denote the formula φ with every occurrence of the term t replaced by the term s. In particular, if x is a variable, $\varphi[x \leftarrow t]$ is the result of substituting the variable x by the term t. Further, $\mathit{vars}(\varphi)$ denotes the set of free variables in φ. Finally, a variable $v \in \mathit{vars}(\varphi)$ is called *unconstrained in* φ, if it occurs only once in the formula φ and it is called *constrained* otherwise.

If convenient, we work with functions as with sets of pairs. For example, the union of functions $f\colon A \to B$ and $g\colon C \to D$ where $A \cap C = \emptyset$ is a function $f \cup g\colon (A \cup C) \to (B \cup D)$. Similarly, $\{(a, b)\}$ is a function from the set $\{a\}$ to the set $\{b\}$. This function is also denoted as $\{a \mapsto b\}$.

3 Unconstrained Terms in Quantifier-Free Formulas

This section formalizes known simplifications of quantifier-free formulas containing *unconstrained terms*. Intuitively, a term t is unconstrained in the formula φ if for every assignment to the variables occurring in the term, every possible value of the sort of the term t can be obtained by changing values of only variables that are unconstrained in φ. The idea of simplification is to replace a nontrivial unconstrained term by a fresh variable, which leads to a smaller equisatisfiable formula. For example, consider the formula $(x + 3y = 0 \ \wedge \ y > 0)$ in the theory of integers. The formula contains one unconstrained variable x. The term $x+3y$ is unconstrained as it can attain any integer value, regardless of the value of y. If we replace the term $x + 3y$ by a fresh variable v, we get the equisatisfiable formula $(v = 0 \ \wedge \ y > 0)$. Alternatively, one can realize that the whole term $x + 3y = 0$ is unconstrained and thus it can be replaced by a fresh *Boolean* variable w. In this way, we get an equisatisfiable formula $(w \ \wedge \ y > 0)$. In both cases, the variable y of the simplified formula become unconstrained and the formula can be further simplified.

To formalize the simplification principle, we define when a term is *unconstrained due to* a set of variables U, which means that a term can evaluate to an arbitrary value by changing only values of variables in U. Further, we define when a term is unconstrained in a formula φ, which means that it is unconstrained due to a set of variables that are unconstrained in φ.

Definition 1. *Let t be a term and $U \subseteq vars(t)$ be a set of variables. We say that the term t is* unconstrained due to U *if, for each valuation μ of variables in $(vars(t) \setminus U)$ and every value b of the same sort as the term t, there exists a valuation ν of variables in U such that $[\![t]\!]_{\mu\cup\nu} = b$.*

Example 1. In the bit-vector theory, the following terms are unconstrained due to $\{x\}$ for any term t' not containing x:

- $x + t'$ and $t' + x$,
- $c^{[n]} \cdot x$ and $x \cdot c^{[n]}$ if c is an odd constant,
- $\texttt{bvnot}(x)$, $\texttt{bvxor}(x, t')$ and $\texttt{bvxor}(t', x)$,
- $x <_u c^{[n]}$ if $c \neq 0$,
- $c^{[n]} <_u x$ if $c \neq 2^n - 1$,
- $x = t'$ and $x \neq t'$.

Note that the last two terms are unconstrained due to $\{x\}$ because each sort of the bit-vector theory contains at least two elements. Further, the terms $x \cdot y$, $\texttt{bvand}(x, y)$, $\texttt{bvor}(x, y)$ are unconstrained due to $\{x, y\}$. A comprehensive list of unconstrained terms can be found for example in Franzén's doctoral thesis [10].

On the contrary, multiplication by an even constant is not an unconstrained term. For example, the term $2 \cdot x$ over the theory of bit-vectors never evaluates to 3 as the number 3 does not have a multiplicative inverse in the ring of integers modulo 2^{32}. As a consequence, the term $x \cdot y$ is neither unconstrained due to $\{x\}$, nor unconstrained due to $\{y\}$.

Definition 2. *A subterm t of a formula φ is called* unconstrained in the formula φ *if it is unconstrained due to a set of variables that are unconstrained in φ.*

The following theorem states the correctness of simplification based on unconstrained terms.

Theorem 1 ([8, 10]). *Let φ be a quantifier-free formula and t its subterm unconstrained in φ. Then φ is equisatisfiable with the formula $\varphi[t \leftarrow v]$, where v is a fresh variable of the same sort as t.*

Note that our definition of unconstrained terms and the statement of Theorem 1 are slightly more general than the ones given by Brummayer and Bruttomesso, which consider unconstrained terms containing only a single unconstrained variable. The definition of unconstrained term used in this paper is due to Franzén [10].

The approach where subformulas are identified with terms of sort *Boolean* brings some additional benefits. In particular, a subformula can be an unconstrained term even if it consists of terms that are not unconstrained. For example, let us consider the formula $\varphi \equiv (3x + 3y = 0 \ \wedge \ y > 0)$ over the theory of integers. The term $3x + 3y$ is not unconstrained as its value is always a multiple of 3. However, term $3x + 3y = 0$ of sort *Boolean* is unconstrained due to $\{x\}$. As x is unconstrained in φ, we can simplify the formula to the equisatisfiable form $(v \ \wedge \ y > 0)$. Elimination of pure literals can then further reduce the formula to the form $(y > 0)$. As both $\top$ and $\bot$ can be obtained by suitable choices of the value of the variable y, the term $y > 0$ is unconstrained due to $\{y\}$, and thus the formula can be simplified to v', where v' is a *Boolean* variable.

Note on models The simplified formulas are in general equisatisfiable to the original ones, but not equivalent. For example, the formulas $(x+3y = 0 \wedge y > 0)$ and $(v = 0 \wedge y > 0)$ mentioned above are both satisfiable, but they use different sets of variables and thus they have different models. In this case, a model of the original formula can be easily computed from the model μ of the simplified formula: it assigns to y the value $[\![y]\!]_\mu$ and to x the value $[\![-3y]\!]_\mu$. However, in some cases, the computation of a model for the original formula can be harder. For example, assume that we have replaced the unconstrained term $180423^{[32]} \cdot x$ over the bit-vector theory by a fresh variable y. To get the value of x such that the term $180423^{[32]} \cdot x$ evaluates to a given value of y then means to find the multiplicative inverse of 180423 in the ring of integers modulo 2^{32} and multiply it by the value of y. Although this inverse can be still computed using an extended Euclidean algorithm, it is computationally not trivial.

Note that algorithms for effective retrieval of models for the original formulas from models of the simplified formulas are beyond the scope of this paper.

4 Partially Constrained Terms in Quantifier-Free Formulas

The key property of the simplification presented in the previous section is that the possible values of an unconstrained term are precisely the same as the possible values of a fresh variable of the same sort. This approach can be generalized even to terms that are *partially constrained*: a complex term can be replaced by a simpler one representing the same values. For example, the value of the term

$6 \cdot x$ over the bit-vector theory can be any number divisible by 2. Therefore, if $6 \cdot x$ is a subterm of a formula where x is unconstrained, then the subterm $6 \cdot x$ can be replaced by $2 \cdot y$ where y is a fresh variable of the same sort as x.

The following definition formalizes the notion that two terms represent the same set of possible values for any fixed valuation of variables in C. Intuitively, in applications of this definition, the set C will contain all constrained variables.

Definition 3. *Let C be a set of variables and t, s be terms of the same sort. Further, let $U = (vars(t) \cup vars(s)) \smallsetminus C$. Terms t and s are called C-*interchangeable*, written $t \overset{C}{\rightleftharpoons} s$, if for every valuation μ of variables in C it holds that*

$$\{[\![t]\!]_{\mu\cup\nu} \mid \nu \textit{ is a valuation of } U\} = \{[\![s]\!]_{\mu\cup\rho} \mid \rho \textit{ is a valuation of } U\}.$$

Now we formulate the simplification principle for partially constrained terms and prove its correctness.

Theorem 2. *Let φ be a quantifier-free formula and C be the set of its constrained variables. For any subterm t of φ and any term s such that $t \overset{C}{\rightleftharpoons} s$ and $vars(s) \cap vars(\varphi) \subseteq C$, the formula φ is equisatisfiable with the formula $\varphi[t \leftarrow s]$.*

Proof. All variables of φ and $\varphi[t \leftarrow s]$ can be divided into three disjoint sets:

1. the set C of all constrained variables in φ,
2. the set $U = (vars(t) \cup vars(s)) \smallsetminus C$ of all variables in t or s that are not in C,
3. the set U' containing all variables that are neither in C, nor in U.

The precondition $vars(s) \cap vars(\varphi) \subseteq C$ formulated in the theorem implies that every variable of U appears either only in t or only in s and not in any other part of the formula. Moreover, variables of U' appear neither in t, nor in s.

Suppose that φ is satisfiable. Hence, there exists a valuation μ of variables in C, a valuation ν of variables in U, and a valuation ν' of variables in U' such that $\mu \cup \nu \cup \nu'$ is a satisfying assignment of φ. As t and s are C-interchangeable and do not contain any variable from U', there exists a valuation ρ of variables in U such that $[\![t]\!]_{\mu\cup\nu} = [\![s]\!]_{\mu\cup\rho}$. As valuations ν and ρ concern only variables of U that do not appear outside t and s, we get that the assignment $\mu \cup \rho \cup \nu'$ satisfies $\varphi[t \leftarrow s]$.

It remains to show that satisfiability of $\varphi[t \leftarrow s]$ implies satisfiability of φ. However, the arguments are completely symmetric. □

Note that the definition of C-interchangeability generalizes Definition 1 in the sense that a term t is unconstrained due to U if and only if it is C-interchangeable with a fresh variable u of the same sort, where $C = vars(t) \smallsetminus U$. Theorem 1 can then be seen as a corollary of Theorem 2.

Applications Now we show some applications of the previous theorem. We start with an example from the theory of non-linear real arithmetic and then focus on terms over the bit-vector theory. In particular, we focus on simplification of partially constrained terms with multiplication as this operation is very expensive for some SMT solvers, especially these based on BDDs.

Example 2. Consider the term $t \cdot u$ in the theory of non-linear real arithmetic, where u is an unconstrained variable and t is an arbitrary term not containing the variable u. The term $t \cdot u$ can be replaced by $\texttt{ite}(t = 0, 0, v)$, where v is a fresh variable, as the terms $t \cdot u$ and $\texttt{ite}(t = 0, 0, v)$ are $\mathit{vars}(t)$-interchangeable. In general, this simplification can be performed in any theory in which addition and multiplication form a field.

Example 3. In the bit-vector theory, the term $4 \cdot u$ can be evaluated to any bit-vector where the two least significant bits are zeroes. The same holds for the term $12 \cdot u$. In general, the term $c^{[n]} \cdot u$ with a constant $c^{[n]}$ can represent any bit-vector ending with i zeroes, where i is the highest integer such that 2^i divides c. This follows from the fact that c can be expressed as $2^i \cdot m$ for some odd number m and every odd number has a multiplicative inverse m^{-1} in the bit-vector theory. All bit-vectors with i zeroes at the end can be also represented by the term $v \ll i$. Hence, the terms $c^{[n]} \cdot u$ and $v \ll i$ are $\emptyset$-interchangeable. Finally, Theorem 2 implies that a formula φ with an unconstrained variable u and a term $c^{[n]} \cdot u$ is equisatisfiable with the formula $\varphi[c^{[n]} \cdot u \ \leftarrow \ v \ll i]$ where v is a fresh variable and i is the constant described above. Note that the term $v \ll i$ is easier to compute and express as a circuit than the original multiplication by a potentially large constant $c^{[n]}$.

Example 4. More interestingly, we can simplify also the term $t \cdot u$ where u is unconstrained, even if t is a term with a non-constant value. As an example, consider the term $t \cdot u^{[3]}$ for a 3-bit variable u. We write $t[i]$ as a shortcut for the extraction of the i-th least significant bit of the term t where $0 \leq i \leq 2$, i.e. $t[i] \equiv \texttt{extract}^1_{2-i}(t)$. Then $t \cdot u^{[3]}$ is $\mathit{vars}(t)$-interchangeable with the term

$$\texttt{ite}\Big(t[0] = 1^{[1]},\ v_0,\ \texttt{ite}\big(t[1] = 1^{[1]},\ v_1 \ll 1^{[3]},\ \texttt{ite}(t[2] = 1^{[1]},\ v_2 \ll 2^{[3]},\ 0^{[3]})\big)\Big).$$

In general, the term $t \cdot u^{[k]}$ is $\mathit{vars}(t)$-interchangeable with the term defined as

$$\begin{aligned}\texttt{ite}\Big(t[0] = 1^{[1]}, v_0,& \\ &\texttt{ite}\big(t[1] = 1^{[1]}, v_1 \ll 1^{[k]}, \\ &\qquad \dots \\ &\qquad\qquad \texttt{ite}(t[k-1] = 1^{[1]}, v_{k-1} \ll (k-1)^{[k]}, 0^{[k]}) \dots\big)\Big).\end{aligned}$$

Therefore, in a formula φ with an unconstrained variable $u^{[k]}$, a term $t \cdot u^{[k]}$ can be replaced by the term above with fresh variables $v_0, v_1, \dots, v_{k-1}$ and the resulting formula is equisatisfiable with φ.

In the previous two examples, the multiplications has been replaced by operations like bit-equality and bit-shift by a constant, which are very cheap for BDD-based SMT solvers.

Example 5. Now we discuss some simplification rules mentioned by Bruttomesso without a proof of correctness. For example, consider the simplification rule that

Table 1: Each line presents a pair of $\mathit{vars}(t)$-interchangeable terms, assuming that $b, u \notin \mathit{vars}(t)$. Terms on the right are considered simpler for SMT solvers than these on the left.

$t <_u u$	$b \wedge t \neq 2^k - 1$
$t <_s u$	$b \wedge t \neq 2^{k-1} - 1$
$u <_u t$	$b \wedge t \neq 0$
$u <_s t$	$b \wedge t \neq -2^{k-1}$
$t \leq_u u$	$b \vee t = 0$
$t \leq_s u$	$b \vee t = -2^{k-1}$
$u \leq_u t$	$b \vee t = 2^k - 1$
$u \leq_s t$	$b \vee t = 2^{k-1} - 1$

rewrites the term $t >_u u$ containing an unconstrained bit-vector variable u by the term $b \wedge t \neq 0$, where b is a fresh *Boolean* variable. Intuitively, the rule is correct as $t >_u u$ can be evaluated to both $\top$ and $\bot$ unless t is evaluated to 0. If the value of t is 0, $t >_u u$ evaluates to $\bot$. Correctness of this rule follows directly from Theorem 2 and the fact that the term $t >_u u$ is $\mathit{vars}(t)$-interchangeable with the term $b \wedge t \neq 0$ assuming that $u, b \notin \mathit{vars}(t)$. Similar simplification rules can be derived from pairs of $\mathit{vars}(t)$-interchangeable terms presented in Table 1.

5 Unconstrained Terms in Quantified Formulas

In this section, we extend the treatment of unconstrained variables to formulas containing quantifiers. To simplify the presentation, we suppose that all formulas are in the prenex normal form and do not contain any free variables. That is, $\varphi = Q_1 B_1 Q_2 B_2 \ldots Q_n B_n \psi$, where ψ is a quantifier-free formula, $Q_i \in \{\forall, \exists\}$ for all $1 \leq i \leq n$, and all B_i are pairwise disjoint sets of variables. Sequences $Q_i B_i$ are called *quantifier blocks*. Quantifier blocks are supposed to be maximal, that is $Q_i \neq Q_{i+1}$. A quantifier block $Q_i B_i$ is *existential* if $Q_i = \exists$ and *universal* otherwise. The *level of a variable* x is i such that $x \in B_i$. For a variable x, we denote as $\mathit{level}(x)$ its level and for a set of variables X we define $\mathit{levels}(X) = \{\mathit{level}(x) \mid x \in X\}$. If the set X contains only variables of the same level, we denote as $\mathit{level}(X)$ the level of all variables in that set. We say that an occurrence of a Boolean variable has the *positive polarity* if the occurrence is under an even number of negations and that it has the *negative polarity* otherwise. A variable is called *unconstrained* in the quantified formula φ if it is unconstrained in its quantifier-free part ψ.

It is easy to see that Theorem 1 can not be directly applied to quantified formulas. As an example, consider the formula $\varphi \equiv \exists x \forall y\, (x + y = 0)$. Although the variable x is unconstrained in the formula φ and the term $x + y$ is unconstrained due to $\{x\}$, the conclusion of Theorem 1 is not true regardless of the position of the quantifier for the fresh variable v: the formula φ is equisatisfiable neither with $\exists v \forall y\,(v = 0)$ nor with $\forall y \exists v\,(v = 0)$. The following modified definition of the unconstrained term solves this problem.

Definition 4. *Let φ be a quantified formula, t its subterm, and $U \subseteq \mathit{vars}(t)$ a set of variables such that $|\mathit{levels}(U)| = 1$. We say that the term t is* unconstrained due to U *if, for each valuation μ of variables in $(\mathit{vars}(t) \setminus U)$ and every value b of the same sort as the term t, there exists a valuation ν of variables in U such that $[\![t]\!]_{\mu\cup\nu} = b$ and, furthermore,*

$$\mathit{level}(U) \;\geq\; \max\Big(\mathit{levels}\big(\mathit{vars}(t) \setminus U\big)\Big).$$

For example, in the formula $\exists x \forall y\,(x + y = 0)$ mentioned above, the subterm $x + y$ is not unconstrained due to $\{x\}$, since $\mathit{level}(x) < \mathit{level}(y)$. On the other hand, it is unconstrained due to $\{y\}$.

The following theorem shows that a subterm that are unconstrained due to a set of unconstrained variables can be simplified even in quantified formulas.

Theorem 3. *Let φ be a formula, t be a term, U be a subset of $\mathit{vars}(t)$, and v be a variable not occurring in φ. If t is unconstrained due to the set of variables U and all variables in U are unconstrained in φ, then φ is equivalent to the formula φ in which the term t is replaced by v and the variables of U are replaced in their quantifier block by the variable v.*

Proof. The definition of unconstrained subterm implies that all variables in U have the same level. Let $k = \mathit{level}(U)$ and let $\varphi \equiv Q_1 B_1 \ldots Q_k B_k\, \psi$, where the formula ψ can contain quantifiers. We show that the formula $Q_k B_k\, \psi$ is equivalent to the formula $Q_k((B_k \setminus U) \cup \{v\})\,(\psi[t \leftarrow v])$.

Let $V = \bigcup_{1 \leq i < k} B_i$. Observe that $U \subseteq B_k$ and the last line of the definition of an unconstrained term implies $(\mathit{vars}(t) \setminus U) \subseteq V \cup (B_k \setminus U)$. Let μ be an assignment of values to all variables in V. We distinguish two cases according to the quantifier Q_k.

– Suppose that $Q_k = \exists$. If $[\![\exists B_k \psi]\!]_\mu = \top$, then there is a valuation ν of variables in B_k such that $[\![\psi]\!]_{\mu\cup\nu} = \top$. Note that the function $\mu \cup \nu$ assigns values to a superset of $\mathit{vars}(t)$ and therefore can be used to evaluate the term t. Let b_v be the value $[\![t]\!]_{\mu\cup\nu}$. For this value, we have $[\![\psi[t \leftarrow v]]\!]_{\mu\cup\nu\cup\{v\mapsto b_v\}} = \top$ and therefore also $[\![\exists(B_k \cup \{v\})\psi[t \leftarrow v]]\!]_\mu = \top$. Since all variables in U are unconstrained, the formula $\psi[t \leftarrow v]$ does not contain any variable from U and therefore $[\![\exists((B_k \setminus U) \cup \{v\})\,\psi[t \leftarrow v]]\!]_\mu = \top$.

  Conversely, if $[\![\exists((B_k \setminus U) \cup \{v\})\,\psi[t \leftarrow v]]\!]_\mu = \top$, there is a valuation ν of variables in $(B_k \setminus U) \cup \{v\}$ such that $[\![\psi[t \leftarrow v]]\!]_{\mu\cup\nu} = \top$. As t is unconstrained due to the set U, there is a valuation ν_U of variables in U such that $[\![t]\!]_{\mu\cup\nu\cup\nu_U} = \nu(v)$. Therefore $[\![\psi]\!]_{\mu\cup\nu\cup\nu_U} = \top$ and in turn $[\![\exists(B_k \cup \{v\})\,\psi]\!]_\mu = \top$, because $\nu \cup \nu_U$ is an assignment to variables from $B_k \cup \{v\}$. Finally, because the formula ψ does not contain the variable v, we know that $[\![\exists B_k\,\psi]\!]_\mu = \top$.

– If $Q_k = \forall$, the proof is dual to the $\exists$ case, but each existential quantifier is replaced by the universal quantifier and each $\top$ is replaced by $\bot$. $\square$

As an example, consider again the formula $\exists x \forall y\,(x + y = 0)$. According to the previous theorem, it is equivalent with $\exists x \forall v\,(v = 0)$, because the term $x + y$ is unconstrained due to $\{y\}$. Moreover, as the term $v = 0$ is unconstrained due to $\{v\}$, it is equisatisfiable with $\exists x \forall p\, p$, where p is a *Boolean* variable. This formula is trivially equivalent to $\bot$.

6 Partially Constrained Terms in Quantified Formulas

Both described extensions of unconstrained terms – i.e. partially constrained terms and unconstrained terms in quantified formulas – can be combined together in a fairly obvious way. The next theorem precisely describes this combination. The proof of this theorem is a straightforward extension of already presented proofs.

Theorem 4. *Let φ be a quantified formula and t be its subterm such that the set U of unconstrained variables appearing in t satisfies $|levels(U)| = 1$ and*

$$level(U) \;\geq\; \max\big(levels(C)\big),$$

where $C = vars(t) \setminus U$. Further, let s be an arbitrary term such that $t \overset{C}{\rightleftharpoons} s$ and $vars(s) \cap vars(\varphi) \subseteq C$. Then the formula φ is equivalent with the formula φ where the term t is replaced by the term s and the variables of U are replaced in their quantifier block by the set of variables $vars(s) \setminus vars(\varphi)$.

Note that due to this theorem, we can easily transfer simplification rules mentioned by Bruttomesso to quantified formulas, because they can be reformulated using the notion of interchangeable terms, as was described in Section 4.

Moreover, such simplifications can be combined with additional quantifier-specific simplification rules. The key observation is that the simplifications using unconstrained and partially constrained terms often introduce fresh – and therefore unconstrained – *Boolean* variables (see Table 1). For example, if b is an existentially quantified *Boolean* variable that is unconstrained in a formula φ, it can be replaced by $\top$ if it occurs in φ with the positive polarity and by $\bot$ if it occurs with the negative polarity and the resulting formula will be equivalent to the original one. Similarly, an unconstrained universally quantified *Boolean* variable can be replaced by $\bot$ if it has the positive polarity and by $\top$ if it has the negative polarity. Combining those simplifications with simplifications using unconstrained and partially constrained terms therefore yields more straightforward simplification rules, which are shown in Table 2. Although this table shows only rules for terms with positive polarity, the dual versions for terms with negative polarity are straightforward.

Table 2: Derived simplification rules for partially constrained terms (in the left column) with positive polarity in quantified formulas. We assume that u is an unconstrained variable and $\mathit{level}(u) \geq \max\big(\mathit{levels}(\mathit{vars}(t))\big)$.

	Quantifier type of u	
Term	Existential	Universal
$u = t$ or $t = u$	$\top$	$\bot$
$u \neq t$ or $t \neq u$	$\top$	$\bot$
$t <_u u$	$t \neq 2^k - 1$	$\bot$
$t <_s u$	$t \neq 2^{k-1} - 1$	$\bot$
$u <_u t$	$t \neq 0$	$\bot$
$u <_s t$	$t \neq -2^{k-1}$	$\bot$
$t \leq_u u$	$\top$	$t = 0$
$t \leq_s u$	$\top$	$t = -2^{k-1}$
$u \leq_u t$	$\top$	$t = 2^k - 1$
$u \leq_s t$	$\top$	$t = 2^{k-1} - 1$

7 Experimental Evaluation

We have implemented all mentioned simplifications of quantified bit-vector formulas containing partially constrained terms – including all rules mentioned by Franzén and rules from Examples 3 and 4, and Table 2. The algorithm iteratively simplifies the input formula up to the fixed point. The implementation is written in C++, uses Z3 API to parse the input formula, and is freely available at `https://gitlab.fi.muni.cz/xjonas/BVExprSimplifier`. We have evaluated the effect of implemented simplifications on the performance of the solvers Z3 [9], Boolector [13], and the BDD-based solver Q3B [12] on two sets of benchmarks.

The first set of benchmarks contains all 191 formulas from the BV[1] category of the SMT-LIB benchmark repository [2]. The second set of benchmarks consists of 5 461 formulas generated by the model checker SymDivine [4] when run on verification tasks from SV-COMP [5]. The generated quantified formulas arise from an equivalence check of two symbolic states, both of which are represented by path conditions. Since the input for the tool SymDivine is an LLVM bit-code, the resulting formulas are expected to contain a large number of unconstrained variables.

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 8 GB of RAM and 15 minutes of CPU time. All measured times are CPU times and include both the time of formula preprocessing and the time of the actual SMT solving, unless explicitly stated otherwise. For reliable benchmarking

[1] BV is a category of quantified bit-vector formulas without arrays and uninterpreted functions.

Table 3: For each benchmark set, solver, and configuration, the table provides the numbers of formulas decided as satisfiable (*sat*), unsatisfiable (*unsat*), or undecided (*other*) because of an error, a timeout or a memory out. The table also shows the time necessary to decide benchmarks in the benchmark set in the form of *simplification time + solving time*, or just *solving time* if the simplification was not performed. Only benchmarks that were decided by both configurations of the given solver are counted into solving times.

	SMT-LIB				SymDivine			
	sat	unsat	other	time (s)	sat	unsat	other	time (s)
`Z3`	70	92	29	426	1137	3999	325	3006
`Z3-s`	72	92	27	16 + 430	1137	4194	130	169 + 381
`Boolector`	78	95	18	1 513	1137	3296	1028	20 269
`Boolector-s`	86	95	10	16 + 1 257	1137	3610	714	169 + 1 173
`Q3B`	94	94	3	2 986	1137	4202	122	9 046
`Q3B-s`	94	94	3	16 + 2 233	1137	4202	122	169 + 3 082

we employed BenchExec [6], a tool that allocates specified resources for a program execution and measures their use precisely. We used the solver Z3 in the latest stable version 4.5.0, Boolector in the version attached to the paper [13], and the solver Q3B in the latest development version (commit `6830168` on GitHub).

For all three SMT solvers, we compare the performance of two different configurations:

– In configurations `Z3`, `Boolector`, and `Q3B`, SMT solvers are run on the input formula, after performing cheap local simplifications done by Z3.
– In configuration `Z3-s`, `Boolector-s`, and `Q3B-s`, the input file is simplified using the implemented simplifications based on unconstrained variables and the same cheap local simplifications performed by Z3 up to the fixed point and the result is fed to SMT solvers.

Table 3 summarizes the obtained results. Thanks to the simplifications, the solver Z3 was able to decide 2 more benchmarks from the SMT-LIB benchmark set and 195 more benchmarks from the SymDivine benchmark set. The solver Boolector can decide 8 more SMT-LIB benchmarks and 314 more SymDivine benchmarks thanks to the simplifications. On the other hand, the number of benchmarks decided by Q3B does not change after performing simplifications. This is not surprising, as the remaining 122 formulas in the SymDivine benchmark set, which Q3B did not solve, do not contain any unconstrained variables.

We have also examined the time needed to solve benchmarks. Even when the time of performing simplifications is counted, simplifications helped to reduce solving time of Z3 on SymDivine benchmarks to 18 %, the solving time of Boolector on SMT-LIB benchmarks to 84 %, the solving time of Boolector on

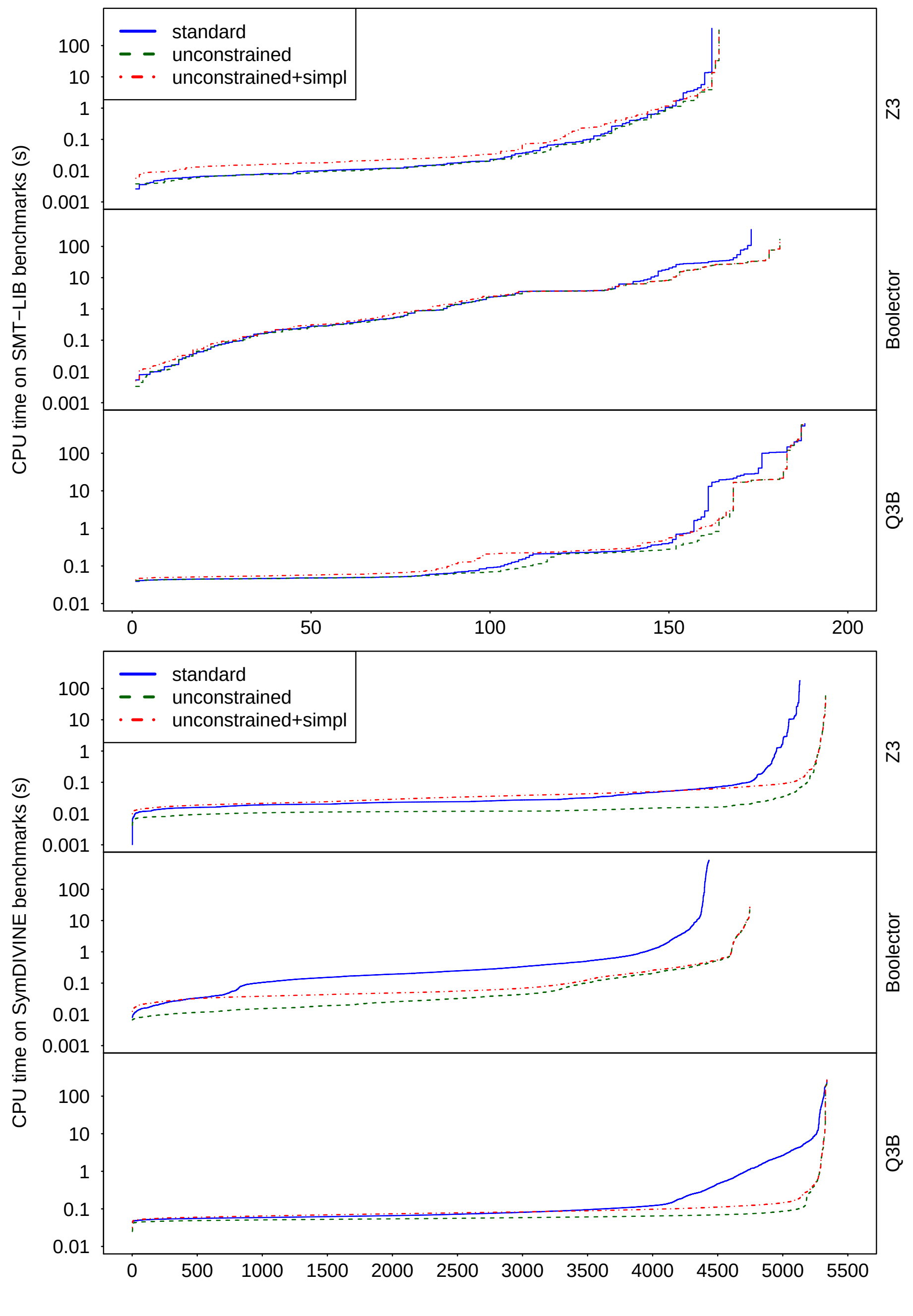

Fig. 1: The figure shows quantile plots for three configurations of all solvers run on both benchmark sets. The configuration *standard* runs on original formulas and the other two run on simplified formulas. The times of *unconstrained* do not include simplification time, the times of *unconstrained+simpl* do.

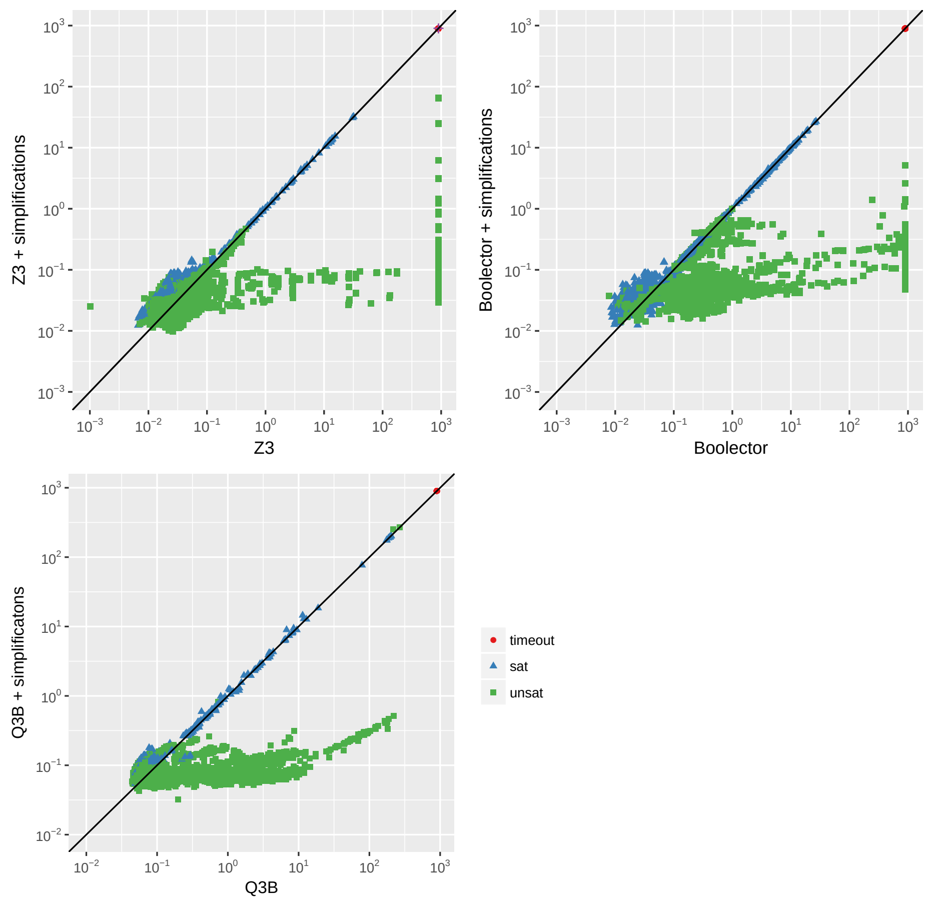

Fig. 2: The figure shows comparison of solving times with and without simplifications for all three SMT solvers on benchmarks from the SymDivine benchmark set. The times are in seconds and include the time of performing simplifications.

SymDivine benchmarks to 7 %, the solving time of Q3B on SMT-LIB benchmarks to 75 %, and the solving time of Q3B on SymDivine benchmarks to 35 % of the original CPU time. On the other hand, simplifications increased the solving time of Z3 on SMT-LIB benchmarks to 105 %, which is in part due to the time needed to perform simplifications and in part due to the fact that after the simplification, Z3's quantifier instantiation heuristics needed more instantiations to solve some of the formulas. Overall, the time needed to perform simplifications is usually negligible when compared to the actual solving time – each formula was simplified in less than 1.3 seconds and the average simplification time is 0.03 seconds. Quantile plots on Figure 1 show comparison of benchmarks solved within a given time for configurations with and without simplifications.

Additionally, scatter plots on Figure 2 show comparison of time needed to solve benchmarks for formulas from the SymDivine benchmark set. For this benchmark set, the proposed simplifications are clearly beneficial, as was expected. Interestingly, the benchmarks from SymDivine set for which the simplifications improved the solving time are not the same for all three SMT solvers – the performance of Z3 improved mainly on benchmarks originating in the SV-COMP category ECA; the performance of Boolector and Q3B improved on benchmarks from all SV-COMP categories. Scatter plots for SMT-LIB benchmarks are not presented in the paper, because the differences are not so significant. However, they can be found along with detailed results of all experiments on the address `http://www.fi.muni.cz/~xstrejc/sat2017/evaluation.html`.

In general, the experimental results clearly show that the proposed simplifications are beneficial for all the considered SMT solvers.

8 Conclusion

We have extended known simplifications of quantifier-free first-order formulas containing unconstrained terms to a more general notion of a partially constrained terms and to quantified formulas. We have further implemented the proposed simplifications for the theory of bit-vectors and shown the beneficial effect of such simplifications for benchmarks arising in verification of software and hardware.

References

1. 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`.
2. Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.
3. Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.
4. 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.
5. 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.
6. 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.
7. Robert Brummayer. *Efficient SMT solving for bit vectors and the extensional theory of arrays.* PhD thesis, Johannes Kepler University of Linz, 2010.
8. Roberto Bruttomesso. *RTL Verification: From SAT to SMT(BV).* PhD thesis, University of Trento, 2008.

9. Leonardo Mendonça de Moura and Nikolaj 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*, pages 337–340, 2008.
10. Anders Franzén. *Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT.* PhD thesis, University of Trento, 2010.
11. Liana Hadarean. *An Efficient and Trustworthy Theory Solver for Bit-vectors in Satisfiability Modulo Theories.* PhD thesis, New York University, 2015.
12. Martin Jonáš and Jan 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*, pages 267–283, 2016.
13. Mathias Preiner, Aina Niemetz, and Armin Biere. Counterexample-Guided Model Synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, pages 264–280, 2017.