Information Processing Letters 135 (2018) 57–61

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On the complexity of the quantified bit-vector arithmetic with binary encoding

M. Jonáš *, J. Strejček

Faculty of Informatics, Masaryk University, Botanická 68a, 602 00, Brno, Czech Republic

ARTICLE INFO

Article history:
Received 5 September 2017
Received in revised form 21 February 2018
Accepted 23 February 2018
Communicated by Krishnendu Chatterjee

Keywords:
Computational complexity
Satisfiability modulo theories
Fixed-size bit-vectors

ABSTRACT

We study the precise computational complexity of deciding satisfiability of first-order quantified formulas over the theory of fixed-size bit-vectors with binary-encoded bit-widths and constants. This problem is known to be in **EXPSPACE** and to be **NEXPTIME**-hard. We show that this problem is complete for the complexity class **AEXP**(poly) – the class of problems decidable by an alternating Turing machine using exponential time, but only a polynomial number of alternations between existential and universal states.

1. Introduction

The first-order theory of fixed-size bit-vectors is widely used for describing properties of software and hardware. Although most current applications use only the quantifier-free fragment of this logic, there are several use cases that benefit from using bit-vector formulas containing quantifiers [1–5]. Consequently, computational complexity of quantified bit-vector logic has been investigated in recent years. It has been shown that deciding satisfiability of quantified bit-vector formulas is **PSPACE**-complete and it becomes **NEXPTIME**-complete when uninterpreted functions are allowed in addition to quantifiers [6].

However, these results suppose that all scalars in the formula are represented in the unary encoding, which is not the case in practice, because in most of real-world applications, bit-widths and constants are encoded logarithmically. For example, the format SMT-LIB [7], which is an input format for most of the state-of-the-art SMT solvers, represents all scalar values as decimal numbers. Such representation can be exponentially more succinct than the representation using unary-encoded scalars. The satisfiability problem for bit-vector formulas with binary-encoded scalars has been recently investigated by Kovásznai et al. [8]. They have shown that the satisfiability of quantified bit-vector formulas with binary-encoded scalars and with uninterpreted functions is **2-NEXPTIME**-complete. The situation for the same problem without uninterpreted functions is not so clear: deciding satisfiability of quantified bit-vector formulas with binary encoded scalars and without uninterpreted functions (we denote this problem as BV2 satisfiability) is known to be in **EXPSPACE** and to be **NEXPTIME**-hard, but its precise complexity has remained unknown [8].

In this paper, we solve this open problem by identifying the complexity class for which BV2 satisfiability is complete. We use the notion of an alternating Turing machine introduced by Chandra et al. [9] and show that the BV2 satisfiability problem is complete for the class **AEXP**(poly) of problems solvable by an alternating Turing machine using exponential time, but only a polynomial number of alternations.

* Corresponding author.
E-mail addresses: martin.jonas@mail.muni.cz (M. Jonáš), strejcek@fi.muni.cz (J. Strejček).

https://doi.org/10.1016/j.ipl.2018.02.018
0020-0190/

Table 1
Recursive definition of the formula size. Operations include logical connectives, function symbols, and predicate symbols. Each t_i denotes a subterm or a subformula, each i_j denotes a scalar argument of an operation, and $Q \in \{\exists, \forall\}$ [8].

	Expression	Size
Constant	$\lvert c^{[n]}\rvert$	$L(c) + L(n)$
Variable	$\lvert x^{[n]}\rvert$	$1 + L(n)$
Operation	$\lvert o(t_1, \ldots, t_k, i_1, \ldots, i_p)\rvert$	$1 + \sum_{1 \le i \le k} \lvert t_i\rvert + \sum_{1 \le j \le p} L(i_j)$
Quantifier	$\lvert Q\, x^{[n]} \varphi\rvert$	$\lvert x^{[n]}\rvert + \lvert\varphi\rvert$

2. Quantified bit-vector formulas

The *theory of fixed-size bit-vectors* (BV or *bit-vector theory* for short) is a many-sorted first-order theory with infinitely many sorts corresponding to bit-vectors of various lengths. Each bit-vector variable has an explicitly assigned sort, e.g. $x^{[3]}$ is a bit-vector variable of bit-width 3. The BV theory uses only three predicates, namely equality ($=$), unsigned inequality of binary-encoded non-negative integers ($\le_u$), and signed inequality of integers in 2's complement representation ($\le_s$). The signature also contains constants $c^{[n]}$ for each $n \ge 1$ and $0 \le c \le 2^n - 1$, and various interpreted functions, namely addition ($+$), multiplication ($*$), unsigned division ($\div$), bitwise negation ($\sim$), bitwise and ($\&$), bitwise or ($|$), bitwise exclusive or ($\oplus$), left-shift ($\ll$), right-shift ($\gg$), concatenation ($\cdot$), and extraction of a subword starting at the position i and ending at the position j ($extract(\_, i, j)$). Although various sources define the full bit-vector theory with different sets of functions, all such definitions can be polynomially reduced to each other [8]. All numbers occurring in the formula, i.e. values of constants, bit-widths and bounds i, j of extraction, are called *scalars*.

There are more ways to encode scalars occurring in the bit-vector formula: in the *unary encoding* or in a *logarithmic encoding*. In this paper, we focus only on formulas using the *binary encoding*. This covers all logarithmic encodings, since all of them are polynomially reducible to each other. In the binary encoding, $L(n)$ bits are needed to express the number n, where $L(0) = 1$ and $L(n) = \lfloor \log_2 n \rfloor + 1$ for all $n > 0$. The entire formula is encoded in the following way: each constant $c^{[n]}$ has both its value c and bit-width n encoded in binary, each variable $x^{[n]}$ has its bit-width n encoded in binary, and all scalar arguments of functions are encoded in binary. The size of the formula φ is denoted $|\varphi|$. The recursive definition of $|\varphi|$ is given in Table 1. For quantified formulas with binary-encoded scalars, we define the corresponding satisfiability problem:

Definition 1 *([8]).* The BV2 *satisfiability problem* is to decide satisfiability of a given closed quantified bit-vector formula with all scalars encoded in binary.

Similarly to Kovásznai et al. [8], we use an *indexing* operation, which is a special case of the extraction operation that produces only a single bit. In particular, for a term $t^{[n]}$ and a number $0 \le i < n$, the indexing operation $t^{[n]}[i]$ is defined as $extract(t^{[n]}, i, i)$. We assume that bits of bit-vectors are indexed from the least significant. For example, given a bit-vector variable $x^{[6]} = x_5 x_4 x_3 x_2 x_1 x_0$, the value of $x^{[6]}[1]$ refers to x_1. In the following, we use a more general version of the indexing operation, in which the index can be an arbitrary bit-vector term, not only a fixed scalar. This operation can be defined using the indexing operation and the bit-shift operation with only a linear increase in the size of the term:

$$t^{[n]}[s^{[n]}] \stackrel{\text{df}}{\equiv} (t^{[n]} \gg s^{[n]})[0].$$

3. Alternation complexity

We assume a basic familiarity with an *alternating Turing machine* (ATM) introduced by Chandra, Kozen, and Stockmeyer [9], and basic concepts from the complexity theory, which can be found for example in Kozen [10]. We recall that each state of an ATM is either *existential* or *universal*. Existential states behave like states of a non-deterministic Turing machine: a run passing through an existential state continues with one of the possible successors. In contrast to this, a run entering a universal state forks and continues into all possible successors. Hence, runs of an ATM are trees. Such a run is accepting if each branch of the run ends in an accepting state.

This section recalls some complexity classes related to alternating Turing machines. Computations in such complexity classes are bounded not only by time and memory, but also by the number of alternations between existential and universal states during the computation. Although bounding both time and memory is useful in some applications, in this paper we need only complexity classes related to ATMs that are bounded in time and the number of alternations. Therefore, the following definition introduces a family of complexity classes parameterized by the number of steps and alternations used by corresponding ATMs.

Definition 2. Let $t, g\colon \mathbb{N} \to \mathbb{N}$ be functions such that $g(n) \ge 1$. We define the complexity class $\mathbf{ATIME}(t, g)$ as the class of all problems A for which there is an alternating Turing machine that decides A and, for each input of length n, it needs at most $t(n)$ steps and $g(n) - 1$ alternations along every branch of every run. If T and G are classes of functions, let $\mathbf{ATIME}(T, G) = \bigcup_{t \in T, g \in G} \mathbf{ATIME}(t, g)$.

Chandra et al. have observed several relationships between classical complexity classes related to time and memory and the complexity classes defined by ATMs [9]. We recall relationships between alternating complexity classes and the classes **NEXPTIME** and **EXPSPACE**, which are important for this paper. It can easily be seen that the class **NEXPTIME** corresponds to all problems solvable by an alternating Turing machine that starts in an existential state and can use exponential time and no alternations: this yields an inclusion $\mathbf{NEXPTIME} \subseteq \mathbf{ATIME}(2^{\mathcal{O}(n)}, 1)$. On the other hand, results of Chandra et al. imply that **EXPSPACE** is precisely the complexity class $\mathbf{ATIME}(2^{n^{\mathcal{O}(1)}}, 2^{n^{\mathcal{O}(1)}})$ of problems solvable in exponential time and with exponential number of alternations. An interesting class that lies in between those two complexity