Scientific Annals of Computer Science vol. 28 (2), 2018, pp. 269–288
doi: 10.7561/SACS.2018.2.269

SMT-Solvers in Action: Encoding and Solving Selected Problems in NP and EXPTIME

Artur NIEWIADOMSKI[1], Piotr SWITALSKI[1]
Teofil SIDORUK[2], Wojciech PENCZEK[1,2]

Abstract

We compare the efficiency of seven modern SMT-solvers for several decision and combinatorial problems: the bounded Post correspondence problem (BPCP), the extended string correction problem (ESCP), and the Towers of Hanoi (ToH) of exponential solutions. For this purpose, we define new original reductions to SMT for all the above problems, and show their complexity. Our extensive experimental results allow for drawing quite interesting conclusions on efficiency and applicability of SMT-solvers depending on the theory used in the encoding.

Keywords: SAT, SMT, SMT-solvers, SMT-Lib, complexity, post correspondence problem, string correction problem, Towers of Hanoi

1 Introduction

The Boolean satisfiability problem (SAT), as the first known NP-complete problem [22], is amongst the most significant theoretical issues of the last decades. Moreover, besides purely academic discourse, SAT-solving algorithms have important practical applications in a number of domains, like, e.g., planning [23], web service composition [31, 28], verification [1, 2], and model

[1]Siedlce University, Faculty of Science, Institute of Computer Science, 3-Maja 54, 08-110 Siedlce, Poland Email: artur.niewiadomski@uph.edu.pl; piotr.switalski@uph.edu.pl

[2] Institute of Computer Science, Polish Academy of Sciences, Jana Kazimierza 5, 01-248 Warsaw, Poland, Email: t.sidoruk@ipipan.waw.pl, penczek@ipipan.waw.pl

checking [9, 3, 13, 20, 34, 36, 27, 35]. Reductions to SAT are generally an efficient way of solving NP-complete problems, both theoretical and practical. However, for many classes of problems constraints in encodings cannot be easily expressed in terms of propositional formulas.

Often, out of necessity or convenience, the formula is expressed as a Boolean combination of predicates over some theory, like, e.g., theory of real and integer arithmetic [24]. Then, in many cases, using reasoning methods tailored to the underlying theory [6] is a more efficient way of satisfiability checking than a direct translation to SAT. The list of specialised decision procedures for (fragments of) theories with practical applications is long and still growing. It includes, amongst others, theories of arrays, strings, finite sets, lists, tuples, and bit-vectors of a fixed or arbitrary finite size. These procedures are implemented and incorporated in many modern SMT-solvers which have recently become more and more powerful and efficient tools. In this paper we compare their performance and draw interesting conclusions.

In our recent papers [29, 30] we investigated efficiency of several modern SAT-solvers, applying them to solving numerous problems of different computation complexity. One of the lessons learned is that while SAT-solvers behave quite efficiently for NP-complete and harder problems, they are by far inferior to tailored algorithms for P-complete problems. Thus, this paper deals with NP and harder problems only. We present seven modern SMT-solvers: CVC4 [16], MathSAT5 [12], Yices [17], Z3 [15], SMTinterpol [11], Boolector [10], and STP [18]. Their efficiency is compared for the following three problems: bounded Post correspondence problem (BPCP), extended string correction problem (ESCP), and the Towers of Hanoi (ToH) problem. We define new original encodings for these problems, which, to the best of our knowledge, have not been previously translated to SMT. For two of them we compare the efficiency of SMT-solvers using two different underlying theories: theory of arrays and bit-vectors. The new reductions for BPCP and ESCP serve not only our experimental studies, but could also be used in practice for solving these problems. ToH, while known to be always satisfiable, is an interesting benchmark because of its exponential complexity w.r.t. the formula size. Obviously, having a valuation satisfying an SMT formula, we can easily reconstruct a problem solution, i.e., we know how to achieve the goal. Each reduction is followed by two results stating the complexity of the reduction and its correctness, i.e., that the formula coding the problem is of a given size and it is satisfiable iff the problem has a solution. Our extensive experimental results allow for drawing quite interesting conclusions on efficiency and applicability of SMT-solvers depending on the theory used in the encoding. Additionally, our intention is to showcase the extent of improvements in the area of SMT solvers.

The rest of this paper is organized as follows. In the next section we present shortly SMT-solving algorithms and modern SMT-solvers. A reduction to SMT for BPCP is discussed in Section 3. A reduction to SMT for ESCP is presented in Section 4, while a reduction for ToH is given is Section 5. In Section 6 experimental results and comparisons are discussed. The final section contains conclusions.

2 Short Overview of SMT-solvers

In this section we briefly present SMT-solving algorithms and modern SMT-solvers.

Satisfiability Modulo Theories (SMT) can be viewed as a generalization of SAT. Both problems consist in determining whether a given formula is satisfiable. However, instances of SAT are purely Boolean formulas, whereas those of SMT are expressed in first-order logic, using predicates from a number of possible input theories. There are two main approaches to solving SMT instances. The first one, referred to as 'eager', involves encoding the input as a Boolean formula. It is then passed to a regular SAT solver, thereby leveraging continuous advancements in the area of SAT solving. Of course, the effectiveness of this method, often called bit-blasting [10], hinges on the initial translation to SAT. Depending on the specifics of the input theory, it could result in a significant increase in the size of the formula.

By contrast, the 'lazy' approach uses specific reasoning procedures for different input theories, allowing to choose the most efficient algorithms and data structures for each. In the lazy schema [7], a SAT-solver works with a Boolean abstraction of the input formula, a skeleton where all predicates over respective theories are replaced with fresh propositional variables. Then, if a satisfying valuation is found, specialized theory solvers are called to check the validity of a proposed solution over underlying theories and domains.

As is the case in the SAT-solving community, there are annual SMT Competitions [4]. Being open to everyone and providing full source code of participating programs, they play a major role in the continued development of more efficient SMT-solvers. SMT-Lib 2 [5] has emerged as the standardized language for SMT instances, similarly to DIMACS for Boolean formulas in CNF. While both formats use plain text for input files, the SMT-Lib 2 syntax is completely different and much more complex, given the increased expressive power of first order logic.

For example, assuming that i is a symbolic integer variable, the constraint expressing that $(i + 2 < 10)$ is the SMT-Lib command $assert(< (+\ i\ 2)\ 10)$. Moreover, if arr is a symbolic array of integers indexed by integers, the equality of the i-th element of the array with some symbolic integer x, i.e., $arr[i] = x$, is expressed as $assert(= (select\ arr\ i)\ x)$.

Apart from the core Boolean operators, the SMT-Lib 2 standard defines several theories, including these of integer and real numbers, arrays, fixed size bit-vectors, and others. Additionally, multiple sub-logics combining specific theories are available, making it possible to employ the most specialized, and thus efficient, reasoning procedures and satisfiability techniques for a given fragment of the full SMT-Lib 2 logic. The SMT encodings for benchmarks presented in this paper use two such sub-logics, namely QF_ALIA (quantifier-free arrays and linear integer arithmetic) and QF_BV (quantifier-free bit-vectors).

3 Reduction to SMT for BPCP

Below we discuss a translation of the bounded Post correspondence problem to an SMT formula exploiting arrays and linear integer arithmetic.

Post correspondence problem (PCP), introduced in [32], has been analysed in the literature numerous times, e.g., in [19], and has several equivalent formulations. One of them is as follows. Given a finite alphabet Σ containing at least two symbols, let Σ^+ be the set of all non-empty words over Σ. Let $W = (w_1, \ldots, w_n)$, $V = (v_1, \ldots, v_n)$, where $\forall_{i=1..n}(w_i, v_i) \in \Sigma^+ \times \Sigma^+$, be two non-empty, finite sequences of n words of Σ^+. The problem consists in finding a sequence of indices $(i_1, \ldots, i_k)$, such that $\overline{w} = w_{i_1} \cdot w_{i_2} \cdot \ldots \cdot w_{i_k}$, $\overline{v} = v_{i_1} \cdot v_{i_2} \cdot \ldots \cdot v_{i_k}$, and $\overline{w} = \overline{v}$. Intuitively, the problem is to find a sequence of indices for which the concatenation of the corresponding words of the lists W and V are equal. In general, when there is no upper bound on a solution length k, PCP is undecidable, as shown by Post in [32]. Thus, we deal with a bounded version of PCP (BPCP, for short), where k is bounded. BPCP is in NP [19] and was also studied in different contexts, for example, using DNA-based bio-computations [21]. An instance of PCP can be visualised using a list of tiles similar to domino bricks where the i-th tile contains the word w_i on the top, and the word v_i at the bottom. Each tile has assigned an id equal to the position of the tile in the input list and we assume that at least k copies of each tile is at our disposal. Thus, a solution is the sequence of tile's ids, and a single element of the sequence is also called a *solution step*. An example instance and a solution is depicted in Fig. 1.

Translation to SMT. Let $m \in \mathbb{N}_+$ be the length of the longest word of W and V, i.e., $m = max(|w|)$, for $w \in W \cup V$. To avoid ambiguity, we refer to the words of W and V as to *elements* or *segments*, while the concatenated word $\overline{w}$ and $\overline{v}$ is called the 'upper' and the 'bottom' word, respectively. Moreover, we treat the symbols of Σ as the consecutive natural numbers $\{0, \ldots, |\Sigma| - 1\}$.

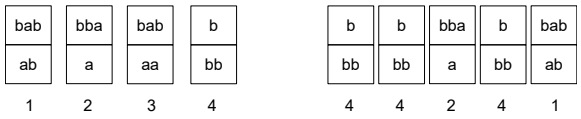

Figure 1: An example instance of PCP, for $n = 4$ and $\Sigma = \{a, b\}$, $W = (bab, bba, bab, b)$, $V = (ab, a, aa, bb)$. The solution $(4, 4, 2, 4, 1)$ corresponds to the word $\overline{w} = \overline{v} = bbbbabbab$.

We encode a BPCP instance as an SMT formula using theory of arrays and linear integer arithmetic. To this aim, we allocate a symbolic array $\mathbf{a}$ of integers, indexed also by integers. The subsequent elements of $\mathbf{a}$ are intended to store the consecutive symbols of the upper and the bottom word. Note that using the same array for both words implies their equality. Besides the array, we allocate the following integer variables[3] $\mathbf{s}_0, \ldots, \mathbf{s}_{k-1}$ to store solution sequences, $\mathbf{p}^w_0, \ldots, \mathbf{p}^w_k$, and $\mathbf{p}^v_0, \ldots, \mathbf{p}^v_k$ to store starting positions of consecutive segments inside the upper and the bottom word, respectively. Note that since $\mathbf{p}^w_0$ and $\mathbf{p}^v_0$ encode positions of the first upper and bottom segments, the variables $\mathbf{p}^w_k$ and $\mathbf{p}^v_k$ would point at start of the $k+1$-th segment, while there are only k of them. Actually, $\mathbf{p}^w_k$ and $\mathbf{p}^v_k$ are used to compute the total length of the resulting word.

Thus, to encode BPCP we need $3k + 2$ symbolic integers, and one symbolic array, where no more than $k * m$ integers are stored.

Now, we are at a position to show a reduction of BPCP to SMT in a top-down manner. Overall, the formula encoding BPCP is as follows.

$$bpcp(k, W, V) = (\mathbf{p}^w_0 = 0) \wedge (\mathbf{p}^v_0 = 0) \wedge (\mathbf{p}^w_k = \mathbf{p}^v_k) \wedge allWords(k, W, V) \quad (1)$$

where the first part of the formula states that the words $\overline{w}$ and $\overline{v}$ begin at the position 0 and their lengths are equal, whereas $allWords(k, W, V)$ encodes all possible combinations of k elements of W or V over the symbolic array $\mathbf{a}$:

$$allWords(k, W, V) = \bigwedge_{j=0}^{k-1} \Big(wrd(j, W, \mathbf{p}^w) \wedge wrd(j, V, \mathbf{p}^v) \Big). \quad (2)$$

The formula $wrd(j, A, \mathbf{p})$, encoding a choice between the elements of A as the j-th solution step, is given below, where a_i is the i-th element (word) of A, $|a_i|$ and $|A|$ denote the length of a_i, and the number of elements in A, respectively, whilst

[3] Actually, in the SMT-Lib 2 nomenclature, these are called *integer constants*, because solvers try to determine a constant integer value, such that all constraints involving them are met.

$at(a_i, x)$ is the natural number representing the x-th symbol of a_i.

$$wrd(j, A, \mathbf{p}) = \bigvee_{i=0}^{|A|-1} \Big((\mathbf{s}_j = i) \wedge (\mathbf{p}_{j+1} = \mathbf{p}_j \oplus |a_i|) \wedge \bigwedge_{x=0}^{|a_i|-1} (\mathbf{a}[\mathbf{p}_j \oplus x] = at(a_i, x)) \Big) \tag{3}$$

Thus, for a chosen a_i, the formula stores i as the value of the variable $\mathbf{s}_j$, it sets the value of $\mathbf{p}_{j+1}$ to the start position of the next segment of the word, and it encodes the subsequent symbols of a_i over $\mathbf{a}$ starting from the position indicated by $\mathbf{p}_j$. The $\oplus$ symbol denotes the addition involving symbolic integers.

Note that this formula is used twice in (2), for both words, the upper and the bottom one. Each instance refers to a different word sequence and position variables ($W, \mathbf{p}^w$ and $V, \mathbf{p}^v$, respectively). However, both the formulae make use of the same symbolic array $\mathbf{a}$, as well as the same solution variables. This enforces the equality of $\overline{w}$ and $\overline{v}$ and of the indices of the consecutive segments of $\overline{w}$ and $\overline{v}$.

In what follows, by the size of an SMT-formula we mean the number of the theory predicates used in the formula. Next we discuss the size of the formula $bpcp(k, W, V)$.

Lemma 1 *Given k, m, V, and W, where $|W| = |V| = n$, the formula $bpcp(k, W, V)$ is of size $O(knm)$.*

Proof: Notice that the formula (1) contains 3 integer expressions (equalities). Then, the formula (2) consists of $2k$ instances of the formula (3), and each such an instance contains no more than $n(3 + 3m)$ operations like integer additions and comparisons. Thus, the total size of the formula $bpcp(k, W, V)$ is $3 + 2k(3n + 3nm) = 3 + 6kn(1 + m)$ which is in $O(knm)$. □

Theorem 1 *Given k, V, and W. BPCP has a solution iff $bpcp(k, W, V)$ is satisfiable.*

Proof: The proof follows directly from the structure of the formula $bpcp(k, W, V)$. Notice that for any pair of words $\overline{w}, \overline{v}$ a necessary condition for being a solution of BPCP is that their lengths are equal, i.e., $|\overline{w}| = |\overline{v}|$. This is ensured by the conjunction of the first three constraints at the beginning of the formula (1) which states that both words start from the position 0 and the concatenated k segments end at the same position. Next, the formula (2) encodes all possible combinations of k segments of the upper and the lower word using a conjunction of respective instances of the formula (3), each of them dealing with the encoding of a single (upper or bottom) segment. Let us now analyse the structure of the formula $wrd(j, A, \mathbf{p})$.

The purpose of the formula is to encode an alternative of all possible input words (segments) at the j-th step. The start position of the j-th segment is encoded using the respective position variable, either $\mathbf{p}_j^w$ or $\mathbf{p}_j^v$. Moreover, the length of a chosen segment at some position determines also the start position of the next segment, which is encoded over $\mathbf{p}_{j+1}^w$ ($\mathbf{p}_{j+1}^v$, respectively). Such an encoding, together with the initial constraints from the formula (1) ensures that the k-upper and k-bottom segments constitute words of the same length. Note, however, that the consecutive symbols of the upper and the bottom word are encoded over the same symbolic array. Thus, the whole formula is satisfiable iff $\overline{w} = \overline{v}$. □

4 Reduction to SMT for SCP

String correction problem (SCP) is a well-studied class of problems originated from string edit distance. The edit distance defines a set of simple operators conducted on strings. SCP consists in finding out whether a string A can be transformed into a string B in a finite k edit operations. If three single-character operators: $insertion$, $deletion$ or $substitution$, are allowed, then the edit distance is called Levenshtein's one [25]. For example, the Levenshtein distance between "golden" and "older" is 2:

- golden -> olden (deletion of 'g'),
- olden -> older (substitution of 'n' to 'r').

Wagner and Fischer [33] presented the string-to-string correction problem, where Levenshtein distance was used, and gave an algorithm which solves this problem in time proportional to the product of the lengths of the two strings. A more interesting extension of edit distance is Damerau-Levenshtein distance [14], where the set of operators is extended with the operator exchanging the positions of any two symbols in a string. This problem is called the *extended string-to-string correction problem* (ESCP). Both mentioned edit distances are efficiently solvable if all operations have the same cost [8]. In this case the problem is of the same complexity as before [33]. However, if one limits the available operations to single character deletions and swapping the adjacent symbols only, and allows for using at most k edits, where $k \in \mathbb{N}$, then the problem becomes NP-complete [33].

Formally, the problem is formulated as follows. Given a finite alphabet Σ, two non-empty strings $A, B \in \Sigma^+$, and $k \in \mathbb{N}_+$. Assume that $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, where $a_i, b_j \in \Sigma$ for $i = 1..n$ and $j = 1..m$. The problem is whether one can transform A into B by a sequence of at most k edit operations

including *swap* and *deletion*. The swap operator occurs when two consecutive symbols switch their positions. Deletion is the removal of an individual instance of a symbol from a string, therefore shortening its length by 1. After deletion, the gap at the end of the string is completed with the special value ε denoting the empty character. Note that we have $n \geq m$ as otherwise the problem is unsolvable without an insert operator. Consider an example presented in Fig. 2. A transformation from

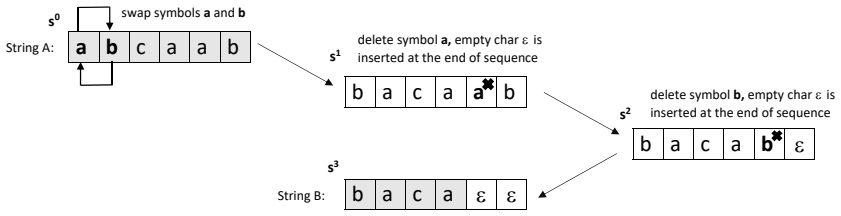

Figure 2: An example of the problem with inputs: A = abcaab, B = baca, and the parameter $k = 3$.

$A = abcaab$ to $B = baca$ is obtained by swapping the first two symbols (a and b) of A and then deleting the last two symbols (ab) of A.

Translation to SMT. The presented encoding of ESCP uses symbolic integers to encode the alphabet symbols and positions inside strings, and symbolic arrays of integers to represent strings. While we need to encode an initial string A and its k copies, the strings are organized in a symbolic arrays $\mathbf{s}^j$, for $j = 0..k$, each representing the possible evolutions of the string A after applying j edit operations. The reduction of ESCP to SMT is presented in a top-down fashion. The whole formula is as follows:

$$escp(k, A, B) = \bigwedge_{i=1}^{n} \Big(\mathbf{s}^0[i] = a_i\Big) \wedge \Big(\bigvee_{j=1}^{k} \bigwedge_{i=1}^{m} (\mathbf{s}^j[i] = b_i)\Big) \wedge \bigwedge_{j=1}^{k} step(j), \quad (4)$$

where a_i and b_i denote the i-th symbol of string A and B, respectively.

The first part of the formula encodes the input word A over the array $\mathbf{s}^0$. Next, the disjunctions encode the desired states of a transformation after j operations, that is, the situation when the representation of the word B is encoded by array $\mathbf{s}^j$. Finally, the last part of the formula encodes all possible changes introduced in the

consecutive steps, as explained below:

$$step(j) = (del(j) \veebar swap(j)) \tag{5}$$

The formula (5) expresses that in the j-th step either a symbol is deleted from the string or two adjacent symbols are swapped.

The formula $del(j)$ encoding the delete operation executed at step j is as follows:

$$\begin{gathered}(p_j \geq 0) \wedge (p_j < n) \wedge (\mathbf{s}^{j-1}[p_j] \neq \varepsilon) \wedge \bigwedge_{i=0}^{n-1} \Big((i < p_j) \wedge (\mathbf{s}^j[i] = \mathbf{s}^{j-1}[i]) \Big) \vee \\ \Big((i \geq p_j) \wedge (\mathbf{s}^j[i] = \mathbf{s}^{j-1}[i \oplus 1]) \Big) \vee \Big((i = n-1)(\mathbf{s}^j[n-1] = \varepsilon) \Big)\end{gathered} \tag{6}$$

The symbolic integer variable p_j denotes the position at which the deletion is applied in the j-th step. Thus, the first two conjuncts of the formula (6) bound the value of p_j to a correct position in the string.

The next conjunct ensures that there is a non-empty symbol at position p_j. Then, all symbols up to position p_j are 'copied' from the previous state. Next, starting from position p_j, the consecutive symbols are 'copied' from the previous state and shifted left by one position. At the very end the empty symbol is added.

Finally, the encoding of the swap operation at the j-th step is defined as:

$$\begin{gathered}swap(j) = (p_j < n-1) \wedge (p_j \geq 0) \wedge (\mathbf{s}^{j-1}[p_j \oplus 1] \neq \varepsilon) \wedge \\ \bigwedge_{i=0}^{n-2} \Big(\big((\mathbf{s}^j[i] = \mathbf{s}^{j-1}[i]) \wedge ((i < p_j) \vee (i > p_j + 1)) \big) \vee \\ \big((\mathbf{s}^j[p_j] = \mathbf{s}^{j-1}[p_j \oplus 1]) \wedge (i = p_j) \big) \vee \big((\mathbf{s}^j[p_j \oplus 1] = \mathbf{s}^{j-1}[p_j]) \wedge (i = p_j + 1) \big) \Big)\end{gathered} \tag{7}$$

where the first line is a precondition ensuring that p_j is a valid position in the string and a non-empty symbol is affected by the operation. This time the value of p_j should be less than $n-1$ because there has to exist a neighbouring symbol, at position $p_j \oplus 1$. The next conjunct 'copies' all unchanged symbols (i.e., all but those at the positions p_j and $p_j \oplus 1$), and afterwards encode swap of the two symbols at positions p_j and $p_j \oplus 1$.

Lemma 2 *The formula $escp(k, A, B)$ is of size $O(kn^2)$.*

Proof: The formula (4) consists of three conjuncts; the first two are of size n and kn, respectively. For the latter, we approximate the number of the inner conjunctions m with n; recall that $n \geq m$ as the problem is otherwise unsolvable without additional operators. Finally, the last subformula is the conjunction over k steps of the ESCP algorithm. Each of these contains the disjunction over n positions in the input string on which operations $swap$ or $delete$ are performed, and whose respective formulas are both of size n. As such, the size of the third conjunct is kn^2. The whole formula $escp(k, A, B)$ is thus of size $n + kn + kn^2$, and so is in $O(kn^2)$. □

Theorem 2 *Given A, B, and k. ESCP has a solution iff the formula $escp(k, A, B)$ is satisfiable.*

Proof: ($\Longrightarrow$) Assume there exists a solution, i.e., a sequence of k operations such that A is transformed into B. We will show that the formula $escp(k, A, B)$ is then satisfied. Any valid solution obviously originates from the input string A, represented in our encoding by the initial state array $\mathbf{s}^0$. Thus, the values stored in array $\mathbf{s}^0$ correspond to the respective symbols of A, as enforced in the first conjunct of the main formula $escp(k, A, B)$. At some point during the subsequent k operations, the desired state where A equals B is reached. As such, there is an operation $j \leq k$, after which all symbolic variables of $\mathbf{s}^j$ correspond to the respective symbols of B. This is represented by the second conjunct of $escp(k, A, B)$. Finally, at each step between these initial and final states, either $delete$ or $swap$ is performed, as permitted by the ESCP conditions. This constraint is enforced by the third and final conjunct of $escp(k, A, B)$. We will now analyse it from the bottom up, starting with the two subformulas encoding operations $delete$ and $swap$. Suppose the character at position p_j is deleted from A in the j-th operation. If so, then p_j must be within the non-empty part of the string, as the deletion of the empty character ε is not valid. Furthermore, following the operation, all characters before the deletion point, that is, at $n < p_j$, are left unchanged, while those after p_j only have their positions shifted by one to the left. These constraints are encoded by the formula $del(j)$. Following a similar pattern, suppose the j-th operation consists of swapping the character at position p_j. Recall that the $swap$ operation in ESCP is defined as exchanging the positions of two consecutive characters; since the last character has no subsequent, it is only applicable up to and including $p_j = n - 1$. All but the two swapped characters, that is, positions ranging over $\{0, \ldots, p_j - 1\}$ and $\{p_j + 2, \ldots, n - 1\}$, are left unchanged. These constraints are encoded in $swap(j)$. The (exclusive) disjunction of $del(j)$ and $swap(j)$ over the positions $p \in \{0, \ldots, n - 1\}$ (with the exception of $n - 1$ for the former) constitutes the subformula $step(j)$. Note that

although at any given step j only one of these can be performed, we use the XOR operator.
($\Longleftarrow$) Assuming $escp(k, A, B)$ evaluates to true, we can obtain its satisfying assignment, that is, the initial array $\mathbf{s}^0$ and the arrays $\mathbf{s}^1 ... \mathbf{s}^k$, corresponding to the state of the input in each of the k steps. They in turn, through encoding using a symbolic array, represent a valid solution of ESCP where the input string A (encoded by $\mathbf{s}^0$) is transformed into B (encoded by $\mathbf{s}^j$) using at most k operations. □

5 Reduction to SMT for ToH

This section discusses a translation to SMT for the ToH problem of exponential time complexity.

Towers of Hanoi (ToH) [26] is a well-known mathematical puzzle where n discs of different sizes are moved between three towers. Initially, all discs are on the first tower, arranged in order of size, ascending. The solution of ToH consists in finding a sequence of disc movements such that all are moved to another tower, preserving their original order. From any given tower t, only the top disc (i.e., the smallest present on t) can be moved to another, and the move is valid unless there is a smaller one on the destination tower t'.

Below, we present our original SMT encoding of the ToH problem. Let $i \in \{0, \ldots, (2^{n-1} - 1)\}$ denote a sequence of states that constitutes a valid ToH solution. The i-th state is then represented by bit-vectors $\mathbf{d}^{t,i}$, each of length n, where $t \in \{0, 1, 2\}$ corresponds to the state of the first, second, and third tower, respectively. By $D(j, i, t)$, we denote $\mathbf{d}^{t,i}_{j-1}$, i.e., that the j-th disc (in order of size, ascending) is placed on tower t. In other words, we set specific bits in $\mathbf{d}^{t,i}$ to 'true' if corresponding discs are indeed located on tower t, and conversely to 'false' if they are not.

The initial and final states are encoded as follows:

$$\mathcal{I} = \bigwedge_{j=1}^{n} D(j, 0, 0), \qquad \mathcal{F} = \bigwedge_{j=1}^{n-1} \Big(D(j, max, 1) \Big) \wedge D(n, max, 0), \tag{8}$$

where $max = 2^{n-1} - 1$ is the number of moves. For some disc j in the i-th state, the pre- and post-move conditions are that no smaller discs can be present on source tower t and destination tower t', respectively. As such, they can be succintly expressed in terms of bit-vector comparisons. Note that by $\mathbf{dmax}_j$ we denote a bit-vector whose bits up to and including the j-th are positive, while the remaining ones are negative.

$$pre(j, i, t) = \mathbf{d}^{t,i} \leq \mathbf{dmax}_j, \tag{9}$$

$$post(j,i,t,t') = \mathbf{d}^{t',i} \leq \mathbf{dmax}_j \wedge D(j,i,t') \wedge \neg D(j,i,t) \tag{10}$$

Any given disc j can potentially be moved to two other towers:

$$move(j,i,t) = pre(j,i,t) \wedge \Big(\bigvee_{t' \in \{0,1,2\} \setminus \{t\}} post(j,i+1,t,t') \Big), \tag{11}$$

while all possible moves in the i-th state are encoded as:

$$\mathcal{M}(i) = \bigvee_{t=0}^{2} \Big(\bigvee_{j=1}^{n} \big(move(j,i,t) \wedge \bigwedge_{k \in \{1,\ldots,n\} \setminus \{j\}} \mathbf{d}_k^{t,i} = \mathbf{d}_k^{t,i+1} \big) \Big), \tag{12}$$

Thus, the whole formula encoding the ToH problem is as follows:

$$ToH(n) = \mathcal{I} \wedge \mathcal{F} \wedge \bigwedge_{i=0}^{max-1} \mathcal{M}(i). \tag{13}$$

The subsequent two lemmas follow from the construction of the formula $ToH(n)$.

Lemma 3 *The formula $ToH(n)$ is of size $O(2^n)$.*

Proof: The solution of the ToH puzzle is a sequence of $max = 2^{n-1} - 1$ states, each of which is represented by three bit-vectors corresponding to towers. Thus, we need $3 \cdot 2^{n-1} - 3 = 2^n + 2^{n-1} - 3$ variables and the formula is of size $O(2^n)$. □

Theorem 3 *The ToH problem for n discs has a solution iff the formula $ToH(n)$ is satisfiable.*

Proof: The main formula $ToH(n)$ has three conjuncts: initial state $\mathcal{I}$, final state $\mathcal{F}$ and the conjuction of all possible moves $\mathcal{M}(i)$ over $max = 2^{n-1} - 1$ moves. Let us first examine the former. $\mathcal{I}$ enforces that all discs are initially on the first tower and as such is self-explanatory. Following the observation in [26], $\mathcal{F}$ does not actually correspond to the 'true' final state of a ToH puzzle, since from that point onwards, remaining moves can be trivially obtained by mirroring past steps. $\mathcal{M}(i)$ is a disjunction over towers $t \in \{0,1,2\}$ and then discs $j \in \{1,\ldots,n\}$ of possible moves $move(j,i,t)$, in conjunction with an additional constraint ensuring every disc but the selected j-th one remains in place, as per the rules of the ToH puzzle. Thus, $\mathcal{M}(i)$ encodes all possible moves in the i-th state. Each possible move $move(j,i,t)$ is a conjunction of pre- and post-move conditions, which again intuitively corresponds to the rules of the ToH puzzle: $pre(j,i,t)$ ensures that the j-th disc being moved is indeed on source tower t, and that there is no smaller one on top of it. Conversely, $post(j,i,t,t')$ verifies that the j-th disc is no longer on t, has been moved to t' and that the move was valid (i.e., there were no smaller discs present on t'). □

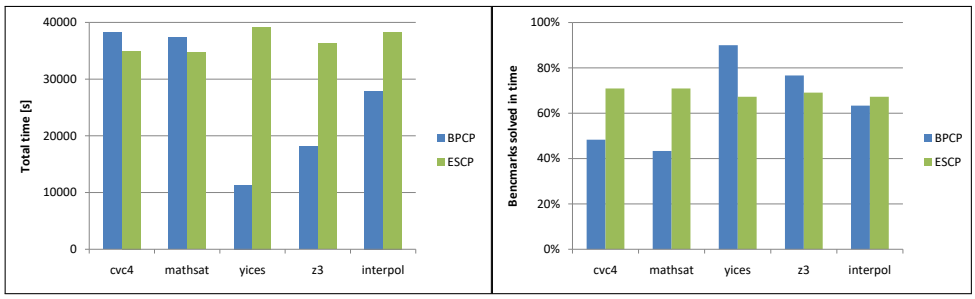

Figure 3: Experimental results for array encoding. Total computation time (left) and percentage of benchmarks solved in time (right).

6 Experimental results and comparisons

In this section we discuss experimental results and compare the efficiency of the SMT-solvers applied. The test platform is a virtual machine with 8 CPU cores and 64 GB of RAM, running Ubuntu 16.04 LTS.

6.1 Theory of arrays and linear integer arithmetic

For encodings using the theory of arrays and linear integer arithmetic (the QF_ALIA logic in SMT-Lib 2 nomenclature) we have a total of 60 benchmarks for BPCP and 110 for ESCP whose results are presented in Fig. 3. Yices was the clear winner in the BPCP test, being able to solve about 90 percent of the instances within the timeout period set at 1000 seconds. Its total running time was also by far the shortest. On the other hand, the performance and effectiveness of all SMT-solvers were remarkably similar when dealing with ESCP instances. While MathSAT and CVC4 proved marginally better both in terms of running time and percentage of solved instances, the latter was around 70 percent for all five tested solvers, and time differences were also minimal.

6.2 Bit-vector Theory

Experimental results for the same instances of BPCP and ESCP, but encoded using the bit-vector theory (the QF_BV logic in SMT-Lib 2 nomenclature), are presented in Fig. 4. Additionally, Fig. 5 summarizes results for the Towers of Hanoi (ToH) problem, which are also listed in Tab. 1. Note that there is no randomised input for ToH instances, hence the relatively small number of benchmarks (10). Boolector,

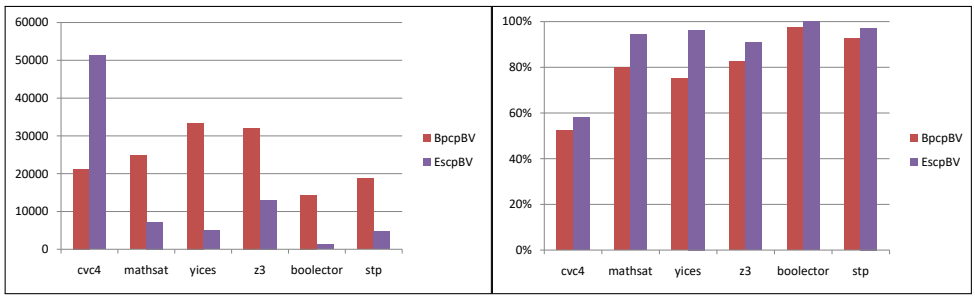

Figure 4: Experimental results for bit-vector encoding. Total computation time (left) and percentage of benchmarks solved in time (right).

N	**cvc4**	**mathsat**	**yices**	**z3**	**boolector**	**stp**
3	0.03	0.01	0.00	0.01	0.03	0.03
4	0.10	0.02	0.00	0.03	0.19	0.10
5	0.29	0.07	0.01	0.08	0.68	0.29
6	0.86	0.21	0.04	0.23	1.33	0.87
7	2.88	0.62	0.12	0.91	1.85	2.51
8	8.64	1.84	0.39	2.53	2.88	7.11
9	23.46	5.97	1.09	18.90	5.52	19.77
10	120.23	20.78	3.27	175.41	11.07	30.26
11	538.74	76.63	9.36	871.20	40.19	83.83
12	1000*	303.63	34.30	1000*	111.50	199.73
Total	1695.23	409.78	**48.58**	2069.30	175.24	344.50

Table 1: Results for the Towers of Hanoi (ToH) benchmark.

utilizing the Lingeling SAT-solver, was the fastest in both the BPCP and ESCP tests. Notably, it reached 100 percent instances solved in the latter. On the other hand, CVC4 was well behind the other solvers in terms of both the total running time and the percentage of the instances successfully solved within the timeout.

A general comparison of the ESCP and BPCP benchmarks is depicted in Fig. 6. The arrows indicate the solvers which were the fastest in processing the respective benchmark set. Clearly, the most efficient way to solve ESCP is to use bit-vector encoding and Boolector. Boolector is also the most efficient solver to deal with bit-vector-based encoding of BPCP. However, Yices is the fastest to solve BPCP using array-based encoding. The array-based encoding of ESCP involving multiple symbolic arrays constituted the most time-consuming benchmarks. All tested solvers supporting QF_ALIA solved them in a similar time, between 34000

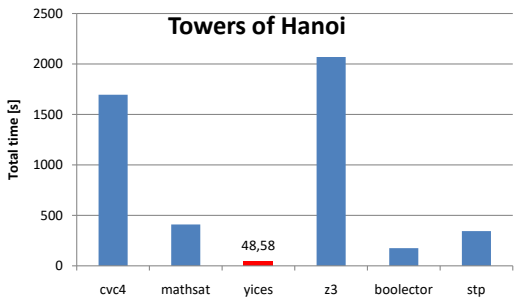

Figure 5: Total computation time for 10 Towers of Hanoi benchmarks using bit-vector encoding.

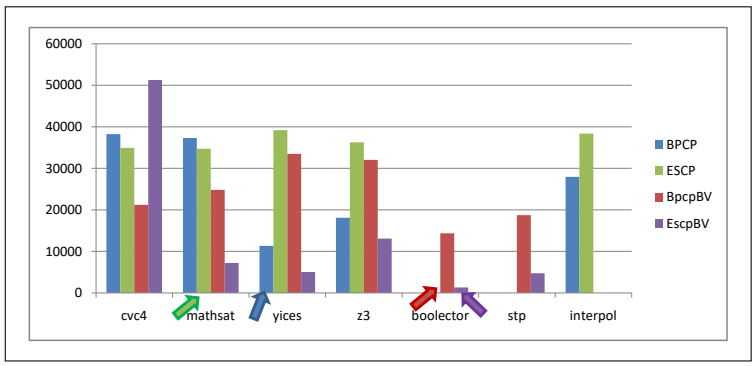

Figure 6: Overall comparison of ESCP and BPCP benchmarks

and 39000 sec., with only a slight advantage of MathSAT.

7 Conclusions

We have presented extensive experiments involving seven modern SMT-solvers and compared their efficiency for two NP-complete problems: bounded Post correspondence problem (BPCP) and extended string-to-string correction problem (ESCP), as well as for the Towers of Hanoi (ToH) puzzle, whose solutions are of exponential complexity. Our original SMT encodings are implemented using both the QF_ALIA and QF_BV logics of SMT-Lib 2 , i.e., the theories of arrays and bit-vectors.

It can be observed that certain SMT-solvers are better suited to one or the other of these two logics. MathSAT, for example, was much more efficient when dealing

with the bit-vector-based encodings of BPCP and ESCP than using the array-based encoding. Moreover, it appears that the performance of specialized solvers, i.e., those that only support one of the two theories, is superior to that of their more universal counterparts. For instance, Boolector, which only supports the theory of bit-vectors but not integer arrays, was by far the most efficient in bit-vector-based benchmarks for both BPCP and ESCP. Although Yices finished first in the ToH test, Boolector and STP, another solver supporting the QF_BV logic exclusively, finished second and third, respectively, both far ahead of remaining universal SMT-solvers.

Overall, as was the case with our prior SAT-solver comparisons [29, 30], the results indicate that solving SMT also remains a considerable challenge. No particular solver proved superior in all test instances, and indeed the opposite could be observed, that is, specialized solvers only supporting a single SMT-Lib 2 theory often demonstrating superior efficiency.

An interesting topic that remains to be investigated in possible future work is a comparison of pure SAT versus SMT encodings using different input theories for a broad range of computational problems of different complexity.

Acknowledgements. The authors are grateful for granting access to the computing infrastructure built in the projects No. POIG.02.03.00-00-028/08 "PLATON - Science Services Platform" and No. POIG.02.03.00-00-110/13 "Deploying high-availability, critical services in Metropolitan Area Networks (MAN-HA)".

References

[1] P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic Reachability Analysis Based on SAT-Solvers. In *Proc. of the 6th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 411–425. Springer-Verlag, 2000. doi:10.1007/3-540-46419-0_28.

[2] A. Armando and L. Compagna. An Optimized Intruder Model for SAT-Based Model-Checking of Security Protocols. In *Electronic Notes in Theoretical Computer Science*, volume 125, pages 91–108. Elsevier Science Publishers, March 2005. doi:10.1016/j.entcs.2004.05.021.

[3] A. Armando, J. Mantovani, and L. Platania. Bounded Model Checking of Software Using SMT Solvers Instead of SAT Solvers. *Int. Journal on Software Tools for Technology Transfer*, 11(1):69–83, 2009. doi:10.1007/11691617_9.

[4] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In K. Etessami and S. K. Rajamani, editors, *Computer Aided Verification*, pages 20–23, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11513988_4.

[5] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6. Technical Report, Department of Computer Science, The University of Iowa, 2017. Available at www.SMT-LIB.org.

[6] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Handbook of Satisfiability: volume 185 Frontiers in Artificial Intelligence and Applications, chapter Satisfiability Modulo Theories. *IOS Press*, pages 825–885, 2009. doi:10.3233/978-1-58603-929-5-825.

[7] C. Barrett and C. Tinelli. Satisfiability Modulo Theories. In E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors, *Handbook of Model Checking*, pages 305–343, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-10575-8_11.

[8] M. Berglund. Analyzing Edit Distance on Trees: Tree Swap Distance is Intractable. In *Stringology*, pages 59–72, 2011.

[9] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking Using SAT Procedures Instead of BDDs. In *Proc. of the ACM/IEEE Design Automation Conference (DAC)*, pages 317–320, 1999. doi:10.1109/dac.1999.781333.

[10] R. Brummayer and A. Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 174–177. Springer, 2009. doi:10.1007/978-3-642-00768-2_16.

[11] J. Christ, J. Hoenicke, and A. Nutz. SMTInterpol: An Interpolating SMT Solver. In *International SPIN Workshop on Model Checking of Software*, pages 248–254. Springer, 2012. doi:10.1007/978-3-642-31759-0_19.

[12] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT5 SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 93–107. Springer, 2013. doi:10.1007/978-3-642-36742-7_7.

[13] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, 2001. doi:10.1023/A:1011276507260.

[14] F. J. Damerau. A Technique for Computer Detection and Correction of Spelling Errors. *Commun. ACM*, 7(3):171—-176, 1964. doi:10.1145/363958.363994.

[15] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Proc. of the Theory and Practice of Software, 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-78800-3_24.

[16] M. Deters, A. Reynolds, T. King, C. W. Barrett, and C. Tinelli. A Tour of CVC4: How It Works, and How to Use It. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, page 7. IEEE, 2014. doi:10.1109/fmcad.2014.6987586.

[17] B. Dutertre. Yices 2.2. In *International Conference on Computer Aided Verification*, pages 737–744. Springer, 2014. doi:10.1007/978-3-319-08867-9_49.

[18] V. Ganesh and D. L. Dill. A Decision Procedure for Bit-Vectors and Arrays. In *International Conference on Computer-Aided Verification*, pages 519–531. Springer, 2007. doi:10.1007/978-3-540-73368-3_52.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[20] M. Kacprzak and W. Penczek. Unbounded Model Checking for Alternating-Time Temporal Logic. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 646–653, 2004.

[21] L. Kari, G. Gloor, and S. Yu. Using DNA to Solve the Bounded Post Correspondence Problem. *Theor. Comput. Sci.*, 231(2):193–203, 2000. doi:10.1016/s0304-3975(99)00100-0.

[22] R. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. doi:10.1007/978-1-4684-2001-2_9.

[23] H. Kautz and B. Selman. Planning as Satisfiability. In *ECAI 92: Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363, 1992.

[24] T. King. *Effective Algorithms for the Satisfiability of Quantifier-Free Formulas Over Linear Real and Integer Arithmetic*. PhD Thesis, New York University, 2014.

[25] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8):707—-710, 1966.

[26] R. Martins and I. Lynce. Effective CNF Encodings for the Towers of Hanoi. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, 2008.

[27] A. Męski, W. Penczek, M. Szreter, B. Woźna-Szcześniak, and A. Zbrzezny. BDD-Versus SAT-Based Bounded Model Checking for the Existential Fragment of Linear Temporal Logic with Knowledge: Algorithms and Their Performance. *Autonomous Agents and Multi-Agent Systems*, 28(4):558–604, 2014. doi:10.1007/s10458-013-9232-2.

[28] A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, and A. Zbrzezny. Towards Automatic Composition of Web Services: SAT-Based Concretisation of Abstract Scenarios. *Fundam. Inform.*, 120(2):181–203, 2012. doi:10.3233/FI-2012-756.

[29] A. Niewiadomski, W. Penczek, and T. Sidoruk. Comparing Efficiency of Modern SAT-Solvers for Selected Problems in P, NP, PSPACE, and EXPTIME. In *Proceedings of the 26th International Workshop on Concurrency, Specification and Programming, Warsaw, Poland, September 25-27, 2017*, pages 1–12, 2017.

[30] A. Niewiadomski, P. Switalski, W. Penczek, and T. Sidoruk. Applying Modern SAT-Solvers to Solving Hard Problems. *submitted to Fundamenta Informaticae*, 2018.

[31] W. Penczek, A. Półrola, and A. Zbrzezny. Towards Automatic Composition of Web Services: A SAT-Based Phase. In *Proc. of the 2nd Int. Workshop on Abstractions for Petri Nets and Other Models of Concurrency and of the Int. Workshop on Scalable and Usable Model Checking (APNOC'10 + SUMO'10)*, pages 76–96, 2010.

[32] E. L. Post. A Variant of a Recursively Unsolvable Problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946. doi:10.1090/s0002-9904-1946-08555-9.

[33] R. A. Wagner and M. J. Fischer. The String-to-String Correction Problem. *Journal of the ACM*, 21(1):168—173, 1974. doi:10.1145/321796.321811.

[34] B. Woźna, A. Zbrzezny, and W. Penczek. Checking Reachability Properties for Timed Automata Via SAT. *Fundamenta Informaticae*, 55(2):223–241, 2003.

[35] B. Woźna-Szcześniak. SAT-Based Bounded Model Checking for Weighted Deontic Interpreted Systems. *Fundam. Inform.*, 143(1-2):173–205, 2016. doi:10.3233/fi-2016-1310.

[36] A. Zbrzezny. SAT-Based Reachability Checking for Timed Automata with Diagonal Constraints. *Fundamenta Informaticae*, 67(1-3):303–322, 2005.