Masarykova univerzita
fakulta informatiky

Complexity of Word-Level Model Checking with Arrays

bachelor's thesis

Lea Petřivalská

Brno, 2024

Declaration

Hereby, I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during the elaboration of this work are properly cited and listed in complete reference to the due source. In the preparation of this thesis, I used the following AI tools:

- Grammarly for syntax and grammar checking,
- ChatGPT for writing style improvement.

I declare that I used these tools according to the principles of academic integrity and that I checked the content and took full responsibility for it.

Advisor: RNDr. Martin Jonáš, Ph.D.

Acknowledgements

I would like to thank my thesis supervisor, Martin Jonáš, for his patient and diligent guidance, his valuable advice, and the support and encouragement I received from him over the months of our working together.

I remain grateful to my mother, who inspired me to pursue a degree in computer science and who didn't live long enough to see me complete it. I am also forever obliged to my grandmother, without whom I would hardly be able to carry on with my studies as I did.

Last but not least, I am thankful to my brothers, my friends, and my fellows – all the wonderful, inspiring humans who continue to fill my life with joy, care, and purpose and who have made this journey infinitely more bearable.

These acknowledgments wouldn't feel complete without also acknowledging that the pursuit of a college degree remains a privilege in today's world. As I approach the end of my undergraduate studies and continue to shape my future, my thoughts are with those who are currently facing different forms of oppression, violence, and discrimination, which make it impossible for them to shape theirs. May we strive for a world where the struggle to write a good thesis is more common than the other, more serious forms of struggle.

Abstract

This work delivers the complexity results for verification over *quantifier-free bit-vector logic with arrays* (QF_ABV). *Verification modulo theory* (VMT) is the problem of model checking symbolic transition systems described with formulas over some theory of first-order logic. As such, it forms a key part of formal verification. The notion of the complexity of various VMT fragments is useful for the development of software model-checking tools. As word-level variables in computer systems are typically modeled with fixed-size bit-vectors, the complexity of VMT over bit-vector logics is of most practical interest. Introducing arrays into bit-vector logics enables modeling memories and array-like data structures, making it possible to describe virtually all practically used computer systems. In the past, the complexity of VMT over various bit-vector logics has been studied by Kovásznai et al. [1]. However, the complexity of word-level model checking with arrays has remained an open problem. In this work, we show the VMT(QF_ABV) problem to be ExpSpace-complete when using unary encoding and 2ExpSpace-complete when using binary encoding of the bit-vector bit-widths and scalars in formulas.

Keywords

formal verification, model checking, complexity, bit-vector logic with arrays

Contents

Introduction **1**

1 First-order Logic and Theories **2**
1.1 Syntax 2
1.2 Semantics 3
1.3 Many-Sorted Logic 3
1.4 The Theory of Fixed-Size Bit-Vectors 4
1.5 The Theory of Arrays 4

2 Verification Modulo Theories **6**
2.1 Transition systems 6
2.2 Satisfiability Modulo Theories (SMT) 7
2.3 Verification Modulo Theories (VMT) 8

3 Computational Complexity **9**
3.1 Turing Machines 9
3.2 Complexity of Decision Problems 10
3.3 Classification of complexity 11
3.4 Hardness and Completeness 11
3.5 Problem encoding 12

4 Results **16**
4.1 Results for unary encodings 16
4.2 Results for binary encodings 21
4.3 Case for QF_ABV without constant arrays 24
4.4 Complexity of verifying safety properties 25

Conclusion **26**

Bibliography **28**

Introduction

Formal verification is a critical area of computer science that ensures the correctness and reliability of hardware and software systems using formal methods. It provides a level of certitude about the system properties that typically cannot be achieved simply by testing. Such a level of certitude is required, for example, in the design of microprocessors, compilers, operating systems, or safety-critical software such as aerospace systems or cryptography protocols.

If a system can be represented as a state-transition model, its properties can be checked by systematically exploring the state space. This approach to formal verification is called *model checking*. A *transition system* is a well-established state-transition model that can be described symbolically in first-order logic. In such a case, all components of the system, as well as the verified properties, are specified by formulas in a fixed first-order theory. The problem of model checking symbolic transition systems is thus known as *verification modulo theory* (VMT) [2].

The computational complexity of VMT varies per theory. Its notion is important for the development of software model-checking tools, as it provides theoretical limits of how efficiently the fragments can be solved. Previously, Kovásznai et al. [1] gave the complexity results for model checking over the quantifier-free logic of fixed-size bit-vectors. They have shown the problem to be PSpace-complete when bit-widths and scalars in formulas are unary-encoded and ExpSpace-complete when binary-encoded. In practice, computer systems contain not only registers that are modeled with fixed-size bit-vectors but also memories that are modeled with arrays. As far as we know, the complexity of model checking with arrays remained an open problem.

This thesis aims to analyze the complexity of word-level model checking with arrays. As the theory of arrays is broad, and only some of its fragments are significant from a practical point of view, this work considers a fragment of the theory that contains only arrays with bit-vector indices and bit-vector elements. More precisely, it explores the complexity of model checking symbolic transition systems expressed over quantifier-free bit-vector logic with arrays (QF_ABV). The QF_ABV logic enables the modeling of bounded memories and array-like data structures, which makes it sufficient to describe nearly all practically used computer systems.

The work is structured in the following way: Chapter 1 gives a concise overview of the syntax and semantics of first-order logic, introduces sorts, and defines the theory of fixed-size bit-vectors and the theory of arrays. Chapter 2 introduces the formalisms of transition systems and verification modulo theory with examples. Chapter 3 defines the computational complexity of decision problems using Turing machines and discusses the encoding of model-checking problems. Finally, Chapter 4 presents the complexity results for VMT(QF_ABV) for unary and binary encodings.

1 First-order Logic and Theories

Before proceeding to the topic of verification modulo theories, let us review the standard definition of the syntax and the semantics of first-order logic that can be found, for example, in Marker's book [3]. Subsequently, two first-order theories relevant to this work are introduced: the theory of fixed-size bit-vectors and the theory of arrays.

1.1 Syntax

The syntax of first-order logic is defined relative to a fixed *signature*:

Definition 1.1.1 (Signature)**.** A signature is a system of predicate and function symbols, with each symbol having a non-negative arity.

Besides signature symbols, the syntax contains variables (from an infinite set of variables), logical symbols ($\neg$, $\wedge$, $\vee$, ...), quantifiers ($\exists$, $\forall$), equality ($=$), and punctuation.

Definition 1.1.2 (Term)**.** Given a signature Σ:

- A variable is a Σ-term.
- If f is an n-ary function symbol in Σ and $t_1, \ldots, t_n$ are Σ-terms then $f(t_1, \ldots, t_n)$ is a Σ-term.

Definition 1.1.3 (Formula)**.** Given a signature Σ:

- If P is an n-ary predicate symbol in Σ and $t_1, \ldots, t_n$ are Σ-terms then $P(t_1, \ldots, t_n)$ is a Σ-formula.
- If t_1, t_2 are Σ-terms then $t_1 = t_2$ is a Σ-formula.
- A logical symbol applied to Σ-formulas is a Σ-formula.
- A quantifier applied to a Σ-formula is a Σ-formula.

A variable that appears in a formula and is not bound by quantifiers (i.e., depends on the variable assignment) is called a *free* variable. Free variables are commonly denoted in parentheses after the formula name, e.g., the formula $\phi(X)$ can contain free variables of the variable set X. A formula is said to be *closed* if it does not contain free variables. Note that although the syntax of first-order formulas contains quantifiers, this work mostly deals with quantifier-free fragments of the logic (see Section 1.5).

1.2 Semantics

Definition 1.2.1 (Structure)**.** Given a signature Σ, a Σ-structure $\mathcal{A}$ consists of a nonempty set A (the *universe*) and an assignment that maps:

- each variable to an element of A,
- each n-ary predicate symbol in Σ to a relation $P^{\mathcal{A}} \subseteq A^n$,
- each n-ary function symbol in Σ to a function $f^{\mathcal{A}} : A^n \to A$.

A *first-order theory* is defined as a class of structures (the *models* of that theory) describing the feasible interpretations of its elements and signature symbols. Formally:

Definition 1.2.2 (Theory)**.** A Σ-theory is a class of Σ-structures closed under isomorphism and variable reassignment.

Alternatively, a theory can be described as a set of closed Σ-formulas (the *axioms* of that theory). The two definitions are equivalent.

Given an assignment of variables and an interpretation of symbols in a signature Σ, the truth value of each Σ-formula is defined. For a complete inductive definition, see, for example, Shoenfield's book [4]. A Σ-structure $\mathcal{A}$ *satisfies* a Σ-formula ϕ, in symbols $\mathcal{A} \models \phi$, if ϕ is true under the assignment and symbol interpretation in $\mathcal{A}$. From this, the model-theoretic definition of $\mathcal{T}$-satisfiability follows.

Definition 1.2.3 ($\mathcal{T}$-satisfiability)**.** Given a Σ-theory $\mathcal{T}$, a Σ-formula ϕ is $\mathcal{T}$-satisfiable if there is a Σ-structure $\mathcal{A} \in \mathcal{T}$ such that $\mathcal{A} \models \phi$.

Note that if a theory has a single interpretation up to isomorphism, i.e., its structures differ solely by the variable assignment v, the notation $\mathcal{A} \models \phi$ can be simplified to $v \models \phi$.

1.3 Many-Sorted Logic

In the previous section, we only considered *unsorted* logic, where all objects belong to a single universe. However, for many theories, it is convenient to distinguish between different sorts (i.e., types) of objects. For that purpose, many-sorted logic is introduced, having slightly modified syntax.

In many-sorted logic, each variable is associated with a sort. A many-sorted signature Σ additionally assigns an $(n+1)$-tuple of sort symbols to each n-ary function symbol and an n-tuple of sort symbols to each n-ary predicate symbol. These are the *arities* of the symbols, and they specify the sorts of the input (resp. output) elements. The syntax of terms and formulas is defined as usual but must now comply with the sort assignment. The sorts of t_1 and t_2 in $t_1 = t_2$ must be the same. Finally, a structure $\mathcal{A}$ of a many-sorted signature Σ assigns a non-empty set (a *domain*) to each sort appearing in Σ and maps each variable to an object from the corresponding domain and each function and predicate symbol in Σ to a function or relation that respects the sorts specified by the associated tuples.

For a detailed definition of many-sorted logic, see Manzano and Aranda's entry [5] in *Stanford Encyclopedia of Philosophy.*

1.4 The Theory of Fixed-Size Bit-Vectors

The many-sorted theory of fixed-size bit-vectors is particularly useful in modeling computer systems. As the name suggests, it deals with bit-vectors of fixed size, and it defines basic operations interpreting the elements either as vectors (of bits) or as the corresponding binary encoded integers.

Formally, the theory of fixed-size bit-vectors is a theory over an infinite signature with infinitely many sorts. Each sort $[n]$, where $n \in \mathbb{N}^+$, corresponds to a set of all bit-vectors of length (or *bit-width*) n. Each function and predicate symbol has multiple versions in the signature that differ in the bit-widths of the input and output sorts. The theory consists of all structures isomorphic to a set of bit-vectors with the following signature and interpretation:

- predicate symbols $\leq_s^{[n]}, \leq_u^{[n]}$ of arity $([n], [n])$ are interpreted as inequalities of signed and unsigned binary-encoded integers, respectively,
- function symbols $+^{[n]}$, $\times^{[n]}$, $\div^{[n]}$ of arity $([n], [n], [n])$ are interpreted as addition, multiplication, and unsigned division, respectively,
- function symbols $\&^{[n]}$, $|^{[n]}$, $\oplus^{[n]}$, $\ll^{[n]}$, $\gg^{[n]}$ of arity $([n], [n], [n])$ are interpreted as bit-wise logical and/or/xor, logical left-shift, and logical right-shift respectively,
- a function symbol $\circ^{[n][m]}$ of arity $([n], [m], [n+m])$ is interpreted as concatenation,
- a function symbol $[i : j]^{[n]}$ of arity $([n], [j - i + 1])$, for each $0 \leq i \leq j < n$, is interpreted as extraction of bits at positions i to j inclusive,

for each $n, m \in \mathbb{N}^+$. In the rest of the work, superscripts indicating the symbol arity are omitted, as it can be deduced from the context.

In addition to the symbols listed above, the theory contains infinitely many constant (i.e., nullary function) symbols $k^{[n]}$, where $n \in \mathbb{N}^+$ and $k \in \{0, \ldots, 2^n - 1\}$, interpreted as the unsigned value of k binary encoded into a bit-vector of length n. The theory definition is taken from the PhD theses of Jonáš [6] and Hadarean [7].

1.5 The Theory of Arrays

For practicality, the theory of arrays, as we define it, is parametrized by two other theories – one for array indices and one for array elements. Later, this allows us to specify a fragment of the theory selected for this work. Intuitively, the theory of arrays defines operations on arrays that contain elements of a single sort. The operations include reading a value on a specific array index, creating a copy of an array that differs on a specific index, and, optionally, initializing an array filled with a constant value.

Given a theory $\mathcal{T}_I$ for array indices and a theory $\mathcal{T}_E$ for array elements, let I be a class of sorts from the theory $\mathcal{T}_I$, and let E be a class of sorts from the theory $\mathcal{T}_E$. The sorts in both classes are interpreted as in the respective theories. Then, the theory of arrays contains all sorts in I, all sorts in E, and for each $\sigma_I \in I$, $\sigma_E \in E$, it contains an array sort $[\sigma_I \to \sigma_E]$ interpreted as a set of arrays that map each index of the sort σ_I to

a value of sort σ_E. The signature of the theory of arrays contains the following function symbols with the following interpretation:

- a function symbol $\text{read}_{\sigma_I}^{\sigma_E}$ of arity $([\sigma_I \to \sigma_E], \sigma_I, \sigma_E)$ such that the expression $\text{read}_{\sigma_I}^{\sigma_E}(a, i)$ is interpreted as the value on index i in the array a,
- a function symbol $\text{write}_{\sigma_I}^{\sigma_E}$ of arity $([\sigma_I \to \sigma_E], \sigma_I, \sigma_E, [\sigma_I \to \sigma_E])$ such that the expression $\text{write}_{\sigma_I}^{\sigma_E}(a, i, v)$ is interpreted as an array containing the same elements as a on all indices except i, where the new value is v,

for each $\sigma_I \in I$, $\sigma_E \in E$.

Optionally, the theory also contains a unary function symbol $\text{constarr}_{\sigma_I}^{\sigma_E}$ of arity $(\sigma_E, [\sigma_I \to \sigma_E])$, for each $\sigma_I \in I$, $\sigma_E \in E$. The expression $\text{constarr}_{\sigma_I}^{\sigma_E}(v)$ is interpreted as an array of sort $[\sigma_I \to \sigma_E]$ containing the value v on all indices. The addition of this symbol does not affect the complexity of the VMT problem in the studied logic, as proven in Section 4.3. Axioms of the theory of arrays can be found, for example, in Barrett et al.'s chapter [8] in the *Handbook of Satisfiability.* In cases where symbol arity is apparent from the argument sorts, the subscripts and superscripts of the symbols can be omitted.

The theory of arrays is useful for modeling memories or array-like data structures. Note that while vectors in the theory of fixed-size bit-vectors are, by definition, bounded in size, the boundedness of arrays depends on the boundedness of the index sort. The theory can, therefore, contain both bounded and unbounded array sorts.

In this work, we explore an instance of the theory of arrays where the theory of fixed-size bit-vectors serves as both $\mathcal{T}_I$ and $\mathcal{T}_E$. Together with the signature and interpretation of the theory of fixed-size bit-vectors, this gives rise to the *bit-vector logic with arrays.* Moreover, this thesis will only consider formulas that do not contain quantifiers. The studied fragment of the logic is thus labeled QF_ABV (*quantifier-free bit-vector logic with arrays*) using the notation of the SMT-LIB [9]. The fragment is practically significant for model checking, as it is sufficient to model virtually all low-level and high-level computer systems. To increase readability, the array sort $[[n] \to [m]]$ is referenced as $[m \to n]$ in the rest of the work. Other instances of the theory of arrays, such as those where array indices are of array sorts, can be the subject of further research.

Examples of formulas in QF_ABV include:

- $a_1^{[8\to10]} = \text{constarr}_{[8]}^{[10]}(v_1^{[10]}) \vee \text{read}(a_1^{[8\to10]}, 0^{[8]}) + 5^{[10]} = 17^{[10]}$, which is satisfied by any assignment where the array a_1 contains the value of v_1 on all indices or it contains the value $12^{[10]}$ on the index $0^{[8]}$,
- $\neg(v_1^{[94]} = v_2^{[94]}) \wedge \text{read}(\text{write}(a_1^{[125\to94]}, 0^{[125]}, v_1^{[94]}), 0^{[125]}) = v_2^{[94]}$, which is unsatisfiable, since after writing the value v_1 to index $0^{[125]}$, this index cannot contain a value different from v_1.

2 Verification Modulo Theories

2.1 Transition systems

A *symbolic transition system* is a state-transition model of computation that can be used to model both sequential and parallel systems. The symbolic representation can then be used to verify the properties of the modeled systems. Note that in the rest of the text, by a *transition system* (TS), we always mean a symbolic transition system, as it is the only kind of transition system discussed in this work.

Definition 2.1.1. A transition system $\mathcal{S}$ is a triplet of the form $\mathcal{S} := (X, I, T)$, where $X := \{x_1, \ldots, x_n\}$ is a set of variables and I, T are formulas such that

- $I(x_1, \ldots, x_n)$ describes the initial configuration of the system,
- $T(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$ describes the transition relation. Here, $x_1, \ldots, x_n$ are the variables of the old state while $x'_1, \ldots, x'_n$ are the variables of the new state.

In many-sorted logic, each $x_i \in X$ is also assigned a sort.

Definition 2.1.2 (State)**.** A state s of a transition system $\mathcal{S} := (X, I, T)$ is a mapping that to each variable in X assigns a value of the corresponding sort. Let $D_1, \ldots, D_n$ be the domains of variables in X. Then $Q := D_1 \times \cdots \times D_n$ is the set of all states of $\mathcal{S}$.

Thus the formula I describes a subset of Q and the formula T describes a binary relation on Q. If a formula ϕ is satisfied under the variable assignment of a state s, we say that s *satisfies* ϕ, in symbols $s \models \phi$.

Definition 2.1.3 (Trace)**.** A trace of a transition system $\mathcal{S} := (X, I, T)$ is a possibly infinite sequence of states $s_0 s_1 s_2 \ldots$ such that

- $s_0 \models I$,
- all pairs of consecutive states s_i, s_{i+1} satisfy the formula $T(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$, where $x_1, \ldots, x_n$ are the variables of s_i and $x'_1, \ldots, x'_n$ are the variables of s_{i+1}.

Definition 2.1.4 (State reachability)**.** A state s is reachable in a system $\mathcal{S}$ if there is a trace $s_0 \ldots s_n$ for some $n \in \mathbb{N}_0$ in $\mathcal{S}$ such that $s_n = s$.

An example of a transition system below models a simple system controlling the traffic signals at a pedestrian crossing.

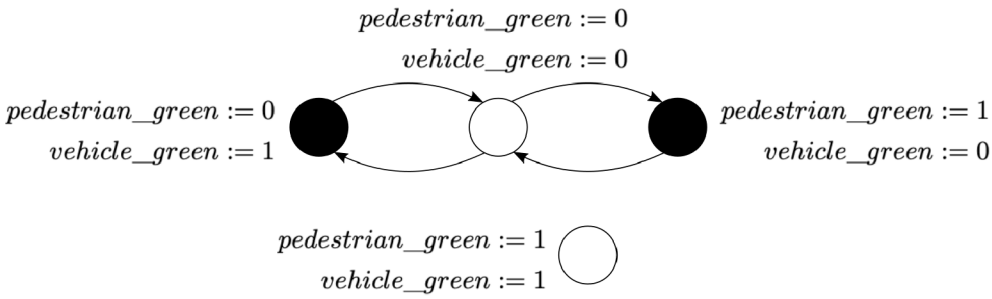

Figure 2.1.1: A visualization of the state space of the TS $\mathcal{S}$ from Example 2.1.1. The initial states are filled in black.

Example 2.1.1. Consider a transition system $\mathcal{S} := (X, I, T)$. The set X contains two variables,

$$X := \{\, \mathit{pedestrian_green} : \{0,1\},\ \mathit{vehicle_green} : \{0,1\} \,\}\,.$$

Both variables are of the same sort $\{0,1\}$ and evaluate to 1 if the respective traffic signal is green and to 0 if the respective traffic signal is red. For simplicity, other colors are not considered. The state space of $\mathcal{S}$ thus contains four states.

The formula I describes the initial configuration of $\mathcal{S}$. In this example, the system can start in any state where the color of the pedestrian traffic signal and the color of the vehicle traffic signal differ, i.e.

$$I := (\, \mathit{pedestrian_green} \neq \mathit{vehicle_green}\,)\,.$$

Finally, the formula T specifies the transition relation. The relation obeys the following principles: if either of the traffic signals is green, it must be red in the next state, and the signals alternate between having the same color and having different colors. Formally,

$$\begin{aligned} T := (&(\, \mathit{pedestrian_green} = 1 \ \rightarrow\ \mathit{pedestrian_green}' = 0\,) \wedge \\ &(\, \mathit{vehicle_green} = 1 \ \rightarrow\ \mathit{vehicle_green}' = 0\,) \wedge \\ &(\, \mathit{pedestrian_green} = \mathit{vehicle_green} \ \leftrightarrow \\ &\quad \mathit{pedestrian_green}' \neq \mathit{vehicle_green}'\,))\,. \end{aligned}$$

The state space of $\mathcal{S}$ with transitions is visualized in Figure 2.1.1. Note that the non-determinism of a transition system does not necessarily mean that the transitions are non-deterministic in the real world. It can, for example, model an external input, such as a pedestrian pushing a button, or other behavior that we simply do not want to model, such as timers. Notice that the state $(1,1)$ is not reachable in the system.

2.2 Satisfiability Modulo Theories (SMT)

Before introducing the problem of *verification modulo theories*, it is helpful to reintroduce $\mathcal{T}$-satisfiability as a decision problem. Satisfiability modulo theories (SMT) is the

problem of deciding whether a first-order formula ϕ is $\mathcal{T}$-satisfiable given a first-order theory $\mathcal{T}$. It is a generalization of the Boolean satisfiability problem (SAT) that only deals with the satisfiability of Boolean formulas. A procedure that checks the satisfiability of first-order formulas is called an *SMT solver*. The notion of SMT is crucial to formal verification.

2.3 Verification Modulo Theories (VMT)

Verification modulo theories (VMT) is the problem of model checking transition systems expressed symbolically in the framework of SMT, i.e., verifying that a transition system described with formulas over a particular first-order theory satisfies specific properties.

This work considers two main types of tested properties:

- **reachability properties** describe states that should be reachable in the system (e.g., a process eventually terminates);
- **safety properties** are conditions that must hold in all reachable states of the system (e.g., a memory never overflows).

For instance, given the system from Example 2.1.1, we might want to know if the system can allow pedestrians to cross the road eventually. In that case, the system should be checked for the reachability property $P_r := (\mathit{pedestrian_green} = 1)$. As Figure 2.1.1 shows, a state satisfying P_r is reachable. Hence, $\mathcal{S}$ satisfies the reachability property. Secondarily, we might want to check that green does not occur on both traffic signals simultaneously, which could result in an accident. Then, the system should be checked for the safety property $P_s := (\mathit{pedestrian_green} = 0 \lor \mathit{vehicle_green} = 0)$. Clearly, the only state violating P_s is not reachable, so $\mathcal{S}$ is P_s-safe.

A safety property P holds in $\mathcal{S}$ whenever a reachability property $\hat{P} := \neg P$ does **not** hold in $\mathcal{S}$. A reachability property P holds in $\mathcal{S}$ whenever a safety property $\hat{P} := \neg P$ does **not** hold in $\mathcal{S}$. Therefore, the problem of verifying reachability properties and the problem of verifying safety properties are dual. In this work, we thus give the complexity results for verification of reachability properties and then discuss the complexity of verifying safety properties implied by these results (see Section 4.4).

Definition 2.3.1. A system $\mathcal{S} := (X, I, T)$ satisfies a reachability property $P(X)$ if a trace $s_0, \dots s_n$ exists such that $s_n \models P$.

A procedure that solves the VMT problem is called a *model checker*. The approach to solving differs per theory. Often, bit-vector variables are broken down into individual bits and analyzed at the bit level using SAT solvers. However, for arrays, it is more efficient to perform model checking directly at the word level, as argued by Biere in [10]. The development of symbolic model checkers is thus what motivates research on the computational complexity of various VMT fragments.

This thesis explores the computational complexity of model checking transition systems described over QF_ABV. In the rest of the work, the labels VMT(QF_ABV$_1$) and VMT(QF_ABV$_2$) specifying the problem's logic and encoding will be used (for encoding, see Section 3.5).

3 Computational Complexity

This chapter introduces the time and space complexity of decision problems and defines several complexity classes that will be referred to later. The computational complexity of a problem depends on the chosen *model of computation*. This work employs computational complexity defined using Turing machines.

3.1 Turing Machines

A *Turing machine* (TM) is a computational model of an automatic system that can alter between a finite number of states and manage an unbounded memory called *tape*. Computation begins with an input word written on the tape and either finishes by accepting or rejecting the word or runs infinitely long. Formally:

Definition 3.1.1. A Turing machine is a 9-tuple $\mathcal{M} := (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, q_{acc}, q_{rej})$, where

- Q is the set of *states*,
- Σ is the *input alphabet*,
- $\Gamma \supset \Sigma$ is the *tape alphabet*,
- $\rhd \in \Gamma$ marks the beginning of the tape,
- $\sqcup \in \Gamma$ marks a blank tape square,
- $\delta \colon Q \setminus \{q_{acc}, q_{rej}\} \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the *transition function*,
- q_0 is the *initial state*,
- q_{acc} is the *accepting state*,
- q_{rej} is the *rejecting state*.

All sets Q, Σ, and Γ are finite. The transition function δ is total. Note that δ can also be *non-deterministic*, i.e., allow for multiple possible transitions. This leads to non-deterministic TMs. In such a case, $\delta \colon Q \setminus \{q_{acc}, q_{rej}\} \times \Gamma \to 2^{Q \times \Gamma \times \{\mathrm{L},\mathrm{R}\}}$.

The machine is initialized so that the first tape square contains the symbol $\rhd$, the characters of the input word $w \in \Sigma^*$ follow, and the rest of the tape is filled with $\sqcup$. The head is initially positioned on the first tape square, while q_0 is the initial machine state.

In each step, the machine state and the content of the active tape square are updated, and the head moves one square left (L) or right (R) of its current position. This transition depends on the contents of the active tape square and the current machine state, and it is described by δ. The machine halts once it reaches a final state, i.e., either q_{acc} or q_{rej}. A word $w \in \Sigma^*$ is accepted by $\mathcal{M}$ if given the input w, $\mathcal{M}$ halts in q_{acc} and it is rejected if $\mathcal{M}$ halts in q_{rej}.

It is required that $\triangleright$ is never overwritten and that the head does not move left of $\triangleright$, formally,

$$\forall q, p \in Q \ \forall c \in \Gamma \ \forall d \in \{\mathrm{L}, \mathrm{R}\} : \ \delta(q, \triangleright) = (p, c, d) \ \rightarrow \ (c = \triangleright \wedge d = \mathrm{R}) \, .$$

Definition 3.1.2. A configuration of a TM at any point of computation is given by the active state, the position of the head, and the entire content of the tape. Let c, c' be configurations of a TM $\mathcal{M}$. We write $c \vdash_{\mathcal{M}} c'$ if $\mathcal{M}$ transitions from c to c' in the next step defined by δ.

Definition 3.1.3. The language $\mathcal{L}(\mathcal{M})$ of a Turing machine $\mathcal{M}$ is the set of all words over Σ accepted by $\mathcal{M}$.

For more details on Turing machines and languages, see, e.g., Sipser's book [11].

3.2 Complexity of Decision Problems

A *decision problem* in computational theory is a problem that can be posed as a yes-no question. Model-checking problems belong to decision problems, as they answer the question: "Does the system S satisfy the property P?"

Decision problems are represented as sets of inputs for which the answer is *yes*. A TM $\mathcal{M}$ solves a problem $\mathcal{P}$ if $\mathcal{L}(\mathcal{M}) = \mathcal{P}$. If $\mathcal{M}$ also halts on every input, i.e., rejects every word $w \notin \mathcal{L}(\mathcal{M})$, we say, it *decides* the problem $\mathcal{P}$. A decision problem is said to be *decidable* if there is such a TM that decides it. Model checking in QF_ABV is decidable. The main intuition for a proof would be that all sorts in QF_ABV are bounded; hence, the state space is also bounded, and state reachability can be decided in finite time by exhaustively exploring the state space.

The complexity of decision problems is defined using the TMs that decide them. The complexity measures are defined as functions of the input size, and they place an *upper bound* on the amount of resources used by the TM in computation. For time complexity, this resource is the number of steps needed to reach the answer, while for space complexity, it is the number of distinct tape squares the TM visits in the computation. Note that $\mathbb{N}$ stands for $\mathbb{N}_0$ in the rest of the work. For $\mathbb{N} \setminus \{0\}$, we use the label $\mathbb{N}^+$. Formally, the function of time complexity $T_{\mathcal{M}} \colon \mathbb{N} \rightarrow \mathbb{N}$ of a TM $\mathcal{M}$ is defined as

$$T_{\mathcal{M}}(n) := max\{t_{\mathcal{M}}(w) \mid w \in \Sigma^n\} \, ,$$

where $t_{\mathcal{M}}(w)$ is the number of steps $\mathcal{M}$ takes given the input w before halting. Similarly, the function of space complexity $S_{\mathcal{M}} \colon \mathbb{N} \rightarrow \mathbb{N}$ is defined as

$$S_{\mathcal{M}}(n) := max\{s_{\mathcal{M}}(w) \mid w \in \Sigma^n\} \, ,$$

where $s_{\mathcal{M}}(w)$ is the number of distinct tape squares $\mathcal{M}$ visits before halting. If $\mathcal{M}$ does not halt on some input w, the respective complexity measures are undefined.

3.3 Classification of complexity

The same problem can be decided by various TMs with different complexities. Moreover, it can be shown that both the time and the space complexities of a TM can be decreased by a constant factor (see, for example, Bovet and Crascenzi's book [12]). Therefore, in order to obtain a meaningful classification of complexity, it is useful to group complexity functions based on their asymptotic upper bound. This gives rise to the $\mathcal{O}$-notation.

Definition 3.3.1. Let $f, g\colon \mathbb{N} \to \mathbb{N}$ be total functions. Then $f(n) \in \mathcal{O}(g(n))$ if there are constants $c, n_0 \in \mathbb{N}$ such that for all $n \geq n_0$

$$f(n) \leq c \cdot g(n)\,.$$

Using the $\mathcal{O}$-notation, we can now describe commonly used complexity classes that group problems solvable by machines of related complexities. In this work, notably, the ExpSpace and 2ExpSpace classes will be useful.

Definition 3.3.2. A problem $\mathcal{P} \in$ ExpSpace if it is decided by a deterministic TM $\mathcal{M}$ in exponential space, i.e., $S_{\mathcal{M}}(n) \in \mathcal{O}(2^{p(n)})$, for some polynomial p of n.

It is worth noting that ExpSpace = NExpSpace (by Savitch's theorem [13]), where NExpSpace is the class solvable by *non-deterministic* TMs in exponential space.

Definition 3.3.3. A problem $\mathcal{P} \in$ 2ExpSpace if it is decided by a deterministic TM $\mathcal{M}$ in double-exponential space, i.e., $S_{\mathcal{M}}(n) \in \mathcal{O}(2^{2^{p(n)}})$, for some polynomial p of n.

Notice that a problem $\mathcal{P}$ that lies in a complexity class C also lies in all classes $C' \supseteq C$ and some $C' \subseteq C$. Consequently, the notion that $\mathcal{P} \in C$ alone is not entirely telling of the problem's difficulty but rather gives an *upper estimate* of the amount of resources necessary to solve it. Thus, when analyzing a problem's complexity, it is useful to explore its so-called *hardness* as well.

3.4 Hardness and Completeness

The *hardness* of a decision problem indicates the *lower estimate* of resources needed to solve it. A problem that is *C-hard* can be used to solve all problems in the complexity class C. It is thus at least as difficult as all the problems in that class. If a problem $\mathcal{P}$ lies in a complexity class C and at the same time is C-hard, it is called *C-complete*.

A problem's hardness is formally defined using *polynomial-time reduction*. To define that, the polynomial-time function complexity needs to be introduced first. Notice that the complexity of a function differs from the complexity of a decision problem in that the contents of the tape at the time of halting matter.

Definition 3.4.1. A total function $f\colon \Sigma^* \to \Sigma'^*$ is computable in polynomial time if there is a deterministic TM $\mathcal{M}$ such that $T_{\mathcal{M}}(n) \in \mathcal{O}(p(n))$, where $p(n)$ is a polynomial of n, and given an input $w \in \Sigma^*$, the computation ends with the word $f(w)$ on tape.

Definition 3.4.2 (Polynomial-time reduction)**.** Let $\mathcal{P} \subseteq \Sigma^*$, $\mathcal{P}' \subseteq \Sigma'^*$ be decision problems. The problem $\mathcal{P}$ reduces to $\mathcal{P}'$ in polynomial time, in symbols $\mathcal{P} \leq_p \mathcal{P}'$, if there is a total function $f\colon \Sigma^* \to \Sigma'^*$ that is computable in polynomial time and

$$w \in \mathcal{P} \iff f(w) \in \mathcal{P}' .$$

Finally, the formal definitions of hardness and completeness are introduced:

Definition 3.4.3 (Hardness)**.** Consider a desision problem $\mathcal{P}$ and a complexity class C. The problem $\mathcal{P}$ is said to be C-hard if $\mathcal{P}' \leq_p \mathcal{P}$ for all $\mathcal{P}' \in C$.

Definition 3.4.4 (Completeness)**.** Consider a decision problem $\mathcal{P}$ and a complexity class C. The problem $\mathcal{P}$ is said to be C-complete if $\mathcal{P}$ is C-hard and $\mathcal{P} \in C$.

Note that although this work defines computational complexity using TMs, according to the *invariance thesis*, all reasonable models of computation decide the same fundamental complexity classes (see, for example, [14]). Thus, the polynomial-time computability of a function can be proven by showing an algorithm that computes it in polynomial time. Similarly, we can prove that a problem belongs to a particular complexity class by showing a deciding algorithm for it in that complexity class. This notion becomes useful later in Chapter 4.

3.5 Problem encoding

As was established in Section 3.2, a decision problem can be viewed as a language over some alphabet Σ. This section discusses how an instance of VMT, i.e., a pair of a transition system and a property, are encoded into strings so that the problem can be fully expressed in the formalism of Turing machines. This work only considers instances in QF_ABV logic. The definitions for encoding of terms and formulas are adapted from Kovásznai et al. [15]. Since the complexity of decision problems is defined relative to the input size, the problem encoding directly influences complexity as it determines the size of the input words.

When analyzing the complexity of model checking in bit-vector logics, two types of encodings are considered: *unary* and *binary*. The difference is in the encoding of scalars appearing in a formula, such as the bit-vector bit-widths. Note that binary encoding is equivalent to any other logarithmic encoding in asymptotic complexity. In unary encodings, the size of an encoded scalar depends linearly on its value. In binary encodings, the size of an encoded scalar is logarithmic with respect to its value. To make the notation more compact, given $n \in \mathbb{N}^+$, let

$$\mathrm{L}(n) := \lceil \log_2(n) \rceil .$$

Furthermore, let $s_{enc}(n)$ denote the size of an encoded scalar n in the rest of the section,

$$s_{enc}(n) := \begin{cases} 1 + n & \text{if the encoding is unary,} \\ 1 + \mathrm{L}(n+1) & \text{if the encoding is binary.} \end{cases}$$

To define the size of an arbitrary formula $\phi \in$ QF_ABV, first, we need to define the size of its terms. Since all variables that appear in quantifier-free formulas are free and thus already associated with a sort, a variable appearing in ϕ has a constant size in both unary and binary encodings. The size of a bit-vector constant $k^{[n]}$ depends on the size of the encoded scalar n and the value k. However, note that the size of k is logarithmic even in unary encoding, as it represents a bit vector. The size of $\mathrm{constarr}^{[m]}_{[n]}(t)$ depends not only on the size of the term t but also on the size of the encoded scalar n as the index sort cannot be deduced from the argument of the operation and thus the subscript $[n]$ cannot be omitted. The size of other function symbols applied to terms in QF_ABV depends solely on the size of the arguments, except for the extraction operation $t[i:j]$ that also depends on the size of the encoded scalars i and j. The following table defines the size $|t|$ of an arbitrary term t in QF_ABV:

	term	size
variable	x_i	1
BV constant	$k^{[n]}$	$1 + \mathrm{L}(k+1) + s_{enc}(n)$
constant array	$\mathrm{constarr}^{[m]}_{[n]}(t)$	$1 + s_{enc}(n) + \lvert t\rvert$
extraction	$t[i:j]$	$1+\lvert t\rvert+s_{enc}(i)+s_{enc}(j)$
other operation	$f(t_1, \ldots, t_n)$	$1 + \lvert t_1\rvert + \cdots + \lvert t_n\rvert$

Recall the inductive definition of formulas from Section 1.1 and recall that QF_ABV does not contain quantifiers. Note that $t_1 \neq t_2$ is encoded as $\neg(t_1 = t_2)$. The size $|\phi|$ of an arbitrary formula $\phi \in$ QF_ABV is defined as follows:

	formula	size
negated formula	$\neg\phi$	$1 + \lvert\phi\rvert$
compound formula	$\phi \vee \psi$, $\phi \wedge \psi$, ...	$1 + \lvert\phi\rvert + \lvert\psi\rvert$
equality	$t_1 = t_2$	$1 + \lvert t_1\rvert + \lvert t_2\rvert$
predicate application	$P(t_1, \ldots, t_n)$	$1 + \lvert t_1\rvert + \cdots + \lvert t_n\rvert$

Having defined the size of encoded terms and formulas in QF_ABV, we can now define the size of an encoded transition system over QF_ABV. Let $s_{enc}(\sigma_i)$, where σ_i is a sort in QF_ABV, be the size of a sort declaration,

$$s_{enc}(\sigma_i) := \begin{cases} 1 + s_{enc}(n) & \text{if } \sigma_i = [n]\,, \\ 1 + s_{enc}(n) + s_{enc}(m) & \text{if } \sigma_i = [n \to m]\,. \end{cases}$$

Then, the size of an encoded transition system $\mathcal{S} = (X, I, T)$ in QF_ABV is equal to $s_{enc}(X) + |I| + |T|$, where

$$s_{enc}(X) := \sum_{x_i\,:\,\sigma_i \in X} \Big(1 + s_{enc}(\sigma_i)\Big)\,.$$

The complete size of an encoded VMT problem instance is the sum of the size of the encoded TS and the size of the checked property. Given that the input size depends on the encoding type, we analyze the complexity of model checking in QF_ABV separately for unary and binary encodings. Hence, VMT(QF_ABV$_1$) and VMT(QF_ABV$_2$) stand for the problems of model checking symbolic transition systems in QF_ABV encoded with unary and binary encodings, respectively. For an overview of the complexity of the SMT and VMT problems for different bit-vector logics and encoding types, including the results proven in this thesis, see Table 3.1.

Example 3.5.1. This example illustrates how the size of an encoded transition system over QF_ABV is computed, and it demonstrates that the difference of sizes generated by unary and binary encodings can be exponential.

Consider a transition system $\mathcal{S} := (X, I, T)$, where

$$\begin{aligned} X &:= \{\, x_1 : [20], \\ &\qquad x_2 : [12 \to 85] \,\}\,, \\ I &:= (x_2 \neq \mathrm{constarr}_{[12]}(0^{[85]}))\,, \\ T &:= (\, x_1 \leqslant_u x_1' \;\wedge\; x_2' = \mathrm{write}(x_2, x_1[0:11], 15^{[85]})\,)\,. \end{aligned}$$

First, consider the size of the individual components for a general encoding:

$$\begin{aligned} s_{enc}(X) &= 2 + s_{enc}([20]) + s_{enc}([12 \to 85]) \\ &= 2 + (1 + s_{enc}(20)) + (1 + s_{enc}(12) + s_{enc}(85)) \\ &= 4 + s_{enc}(20) + s_{enc}(12) + s_{enc}(85)\,, \end{aligned}$$

$$\begin{aligned} |I| &= |\neg(x_2 = \mathrm{constarr}_{[12]}(0^{[85]}))| \\ &= 1 + |x_2 = \mathrm{constarr}_{[12]}(0^{[85]})| \\ &= 1 + 1 + |x_2| + |\mathrm{constarr}_{[12]}(0^{[85]})| \\ &= 1 + 1 + 1 + (1 + s_{enc}(12) + 1 + \mathrm{L}(1) + s_{enc}(85)) \\ &= 5 + s_{enc}(12) + s_{enc}(85)\,, \end{aligned}$$

$$\begin{aligned} |T| &= |x_1 \leqslant_u x_1' \;\wedge\; x_2' = \mathrm{write}(x_2, x_1[0:11], 15^{[85]})| \\ &= 1 + |x_1 \leqslant_u x_1'| + |x_2' = \mathrm{write}(x_2, x_1[0:11], 15^{[85]})| \\ &= 1 + 3 + (1 + |x_2'| + |\,\mathrm{write}(x_2, x_1[0:11], 15^{[85]})|) \\ &= 1 + 3 + (1 + 1 + (1 + |x_2| + |x_1[0:11]| + |15^{[85]}|)) \\ &= 7 + |x_2| + |x_1[0:11]| + |15^{[85]}| \\ &= 7 + 1 + (1 + 1 + s_{enc}(0) + s_{enc}(11)) + (1 + \mathrm{L}(16) + s_{enc}(85)) \\ &= 15 + s_{enc}(0) + s_{enc}(11) + s_{enc}(85)\,. \end{aligned}$$

	SMT	**VMT**
QF_BV$_1$	NP	PSPACE
QF_BV$_2$	NEXPTIME	EXPSPACE
QF_ABV$_1$	NP	EXPSPACE
QF_ABV$_2$	NEXPTIME	2-EXPSPACE

Table 3.1: Completeness classes for the SMT and VMT problems given a specific logic and encoding. The results for SMT are taken from [15]. The results for VMT in QF_BV are taken from [1]. The complexity of VMT(QF_ABV$_1$) and VMT(QF_ABV$_2$) is proven in Chapter 4.

Next, the exact sizes for unary and binary encodings are computed, respectively. Clearly, unary encodings yield the following sizes:

$$\begin{aligned} s_{enc}(X) &= 4 + (1 + 20) + (1 + 12) + (1 + 85) = 124\,, \\ |I| &= 5 + (1 + 12) + (1 + 85) = 104\,, \\ |T| &= 15 + (1 + 0) + (1 + 11) + (1 + 85) = 114\,. \end{aligned}$$

By contrast, binary encodings yield the sizes:

$$\begin{aligned} s_{enc}(X) &= 4 + (1 + \mathrm{L}(21)) + (1 + \mathrm{L}(13)) + (1 + \mathrm{L}(86)) = 23\,, \\ |I| &= 5 + (1 + \mathrm{L}(13)) + (1 + \mathrm{L}(86)) = 18\,, \\ |T| &= 15 + (1 + \mathrm{L}(1)) + (1 + \mathrm{L}(12)) + (1 + \mathrm{L}(86)) = 29\,. \end{aligned}$$

Hence, the size of $\mathcal{S}$ in unary encodings is 342, while in binary encodings it is 70.

4 Results

Equipped with the necessary foundation, we finally get to the bottom of this work, where we present the complexity results for the VMT(QF_ABV$_1$) and VMT(QF_ABV$_2$) problems. First, we prove the complexity results for unary encodings. In Section 4.2, we show modifications of the proofs, which yield the complexity results for binary encodings. Lastly, we show that the complexity of these problems does not change if the theory of arrays is defined without constant arrays, as was already hinted in Chapter 1.

4.1 Results for unary encodings

Theorem 1. VMT(QF_ABV$_1$) *is* ExpSpace-*complete.*

Proof. In Lemma 1, we prove that VMT(QF_ABV$_1$) is ExpSpace-hard by giving a polynomial reduction from each problem in ExpSpace to VMT(QF_ABV$_1$). In Lemma 2, we prove that VMT(QF_ABV$_1$) $\in$ NExpSpace thus, by Savitch's theorem [13], also VMT(QF_ABV$_1$) $\in$ ExpSpace. Lemmas 1 and 2 together imply VMT(QF_ABV$_1$) is ExpSpace-complete. □

Lemma 1. VMT(QF_ABV$_1$) *is* ExpSpace-*hard.*

Proof. Let $\mathcal{P} \subseteq \Sigma^*$ be an arbitrary decision problem in ExpSpace. Then, by definition, such deterministic TM exists that runs in exponential space and decides the problem $\mathcal{P}$. Let $\mathcal{M} := (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0, q_{acc}, q_{rej})$ be such a TM. Recall that $\mathcal{M}$ decides $\mathcal{P}$ if $\mathcal{L}(\mathcal{M}) = \mathcal{P}$ and $\mathcal{M}$ halts on every input. Let w be an arbitrary word over Σ. We show a function $f_{\mathcal{M}}$, that for any choice of w returns a transition system $\mathcal{S} = (X, I, T)$ over QF_ABV$_1$ and a reachability property $R(X)$, such that

$$w \in \mathcal{P} \iff R \text{ is reachable in } \mathcal{S}\,.$$

Next, we prove $f_{\mathcal{M}}$ is computable in polynomial time, hence $\mathcal{P} \leq_p$ VMT(QF_ABV$_1$).

Since $\mathcal{M}$ runs in exponential space, a polynomial p exists such that $S_{\mathcal{M}}(n) \in \mathcal{O}(2^{p(n)})$. Then there is such $c \in \mathbb{N}$ that $S_{\mathcal{M}}(n) \leq c \cdot 2^{p(n)}$ for each $n \in \mathbb{N}$. Let p, c be fixed and consider a function

$$f_{\mathcal{M}}(w) := (\mathcal{S}, R)\,,$$

where $\mathcal{S} = (X, I, T)$ and R are obtained as described in the following paragraphs.

The variable set X contains three variables,

$$X := \{\, \mathit{tape} : [m \to k], \quad \mathit{pos} : [m], \quad \mathit{state} : [n] \,\}\,,$$

where $m, n, k \in \mathbb{N}^+$, $m = p(|w|) + c$, $n = \lceil \log_2 |Q| \rceil$, and $k = \lceil \log_2 |\Gamma| \rceil$. The variables represent a finite prefix of $\mathcal{M}$'s tape, the position of $\mathcal{M}$'s head, and the current state of $\mathcal{M}$, respectively. Recall that by definition of space complexity, $\mathcal{M}$ visits at most $c \cdot 2^{p(|w|)}$ tape squares given the input w. Then, *tape* is large enough to contain all squares visited in the computation since the index sort $[m]$ contains $2^{p(|w|)+c} = 2^c \cdot 2^{p(|w|)} \geq c \cdot 2^{p(|w|)}$ different indices. Analogically, *pos* is large enough to store the position of $\mathcal{M}$'s head at any step. The element sort $[k]$ contains $2^{\lceil \log_2 |\Gamma| \rceil} \geq |\Gamma|$ elements, which makes it large enough to represent the symbols of the tape alphabet Γ. Similarly, the variable *state* can take on $2^{\lceil \log_2 |Q| \rceil}$ different values, $2^{\lceil \log_2 |Q| \rceil} \geq |Q|$.

In the rest of the proof, let $\texttt{enc}$ be an injective mapping of $\Gamma \to [k]$ and $Q \to [n]$. We will say a state s of $\mathcal{S}$ *corresponds to a configuration* c of $\mathcal{M}$, in symbols $s \approx c$, if all of the following conditions are satisfied:

- for all $i \in \{0, \ldots, 2^m - 1\}$, it holds that $\mathrm{read}(\mathit{tape}, i^{[m]}) = \texttt{enc}(a)$, where $a \in \Gamma$ is the symbol written in the tape square with index m in c,
- $\mathit{pos} = k^{[m]}$ and $\mathcal{M}$'s head is positioned on the tape square with index k in c,
- $\mathit{state} = \texttt{enc}(q)$, where q is the current state in c.

Note that tape squares are indexed starting from zero in the above definition.

Fix $l = |w|$ and for $0 \leq i < l$, let $w_i \in \Sigma$ denote the character on position i in w. Let $\texttt{clr_tape}$ represent a tape with all squares marked as empty except the first one, which is marked with $\rhd$, i.e., $\texttt{clr_tape} = \mathrm{write}(\mathrm{constarr}_{[m]}(\texttt{enc}(\sqcup)), 0^{[m]}, \texttt{enc}(\rhd))$. Then,

$$\begin{aligned} I := (\, & \mathit{tape} = \mathrm{write}(\cdots \mathrm{write}(\texttt{clr_tape}, 1^{[m]}, \texttt{enc}(w_0)), \cdots l^{[m]}, \texttt{enc}(w_{l-1})) \wedge \\ & \mathit{pos} = 0^{[m]} \wedge \\ & \mathit{state} = \texttt{enc}(q_0)\,)\,. \end{aligned}$$

A state s satisfies I, if *tape* contains $\texttt{enc}(\rhd), \texttt{enc}(w_0), \ldots, \texttt{enc}(w_{l-1})$ on the first $l+1$ indices and $\texttt{enc}(\sqcup)$ on all remaining indices, *pos* is $0^{[m]}$, and *state* is $\texttt{enc}(q_0)$. Therefore, exactly one state satisfies I, and this state corresponds to the initial configuration of $\mathcal{M}$ given the input w.

Finally, let $\texttt{shift}(\mathit{pos}, d)$ stand for $\mathit{pos} - 1^{[m]}$ and $\mathit{pos} + 1^{[m]}$ for $d = \mathrm{L}$ and $d = \mathrm{R}$, respectively, and let $(q', a', d) = \delta(q, a)$, in the formula

$$\begin{aligned} \texttt{trans}(q, a) := \Big(& (\mathit{state} = \texttt{enc}(q) \wedge \mathrm{read}(\mathit{tape}, \mathit{pos}) = \texttt{enc}(a)) \rightarrow \\ & (\mathit{state}' = \texttt{enc}(q') \wedge \mathit{tape}' = \mathrm{write}(\mathit{tape}, \mathit{pos}, \texttt{enc}(a')) \wedge \mathit{pos}' = \texttt{shift}(\mathit{pos}, d))\Big)\,. \end{aligned}$$

Then,

$$T := \Big(\mathit{state} \neq \texttt{enc}(q_{acc}) \wedge \mathit{state} \neq \texttt{enc}(q_{rej}) \wedge \bigwedge_{(q,a) \in Q \setminus \{q_{acc}, q_{rej}\} \times \Gamma} \texttt{trans}(q, a) \Big)\,.$$

As $\mathcal{M}$ is a deterministic TM, δ is total and deterministic. Thus, there is exactly one possible transition from each non-final state for each tape symbol. Once $\mathcal{M}$ reaches a final state, no transition is possible. Hence, the transitions of $\mathcal{S}$ mimic the steps of $\mathcal{M}$; for all states s of $\mathcal{S}$ such that $s \approx c$ for some configuration c of $\mathcal{M}$, the transition relation described by T contains a transition from s to s' if and only if $s' \approx c'$, where c' is a configuration of $\mathcal{M}$ such that $c \vdash_{\mathcal{M}} c'$.

In addition to $\mathcal{S}$, the function $f_{\mathcal{M}}$ returns the reachability property

$$R := (\mathit{state} = \texttt{enc}(q_{acc}))\,.$$

In the following two paragraphs, we prove that $w \in \mathcal{P} \iff R$ is reachable in $\mathcal{S}$:

If $w \in \mathcal{P}$, it is accepted by $\mathcal{M}$. Then, given the input w, $\mathcal{M}$ reaches an accepting state in a finite number of steps. Then for each $w \in \mathcal{L}(\mathcal{M})$, there is a sequence $c_0 \dots c_n$, $n \in \mathbb{N}$, of configurations such that c_0 is the initial configuration of $\mathcal{M}$ given the input w, q_{acc} is the active state in c_n, and $c_i \vdash_{\mathcal{M}} c_{i+1}$ for each $i \in \{0, \dots, n-1\}$. Since the initial configuration of $\mathcal{S}$ corresponds to the initial configuration of $\mathcal{M}$ given the input w and the transitions of $\mathcal{S}$ mimic the steps of $\mathcal{M}$ for states that correspond to configurations in $\mathcal{M}$, there is a trace $s_0 \dots s_n$ in $\mathcal{S}$ such that $s_i \approx c_i$ for $0 \leq i \leq n$. Then $s_0 \models I$ and $s_n \models R$. Thus R is reachable in $\mathcal{S}$.

If $w \notin \mathcal{P}$, it is rejected by $\mathcal{M}$, since $\mathcal{M}$ halts on every input. Analogically to the previous case, there is thus a sequence $c_0 \dots c_n$, $n \in \mathbb{N}$, of configurations such that c_0 is the initial configuration of $\mathcal{M}$ given the input w, q_{rej} is the active state in c_n, and $c_i \vdash_{\mathcal{M}} c_{i+1}$ for each $i \in \{0, \dots, n-1\}$. Since $\mathcal{M}$ is deterministic, this is also the only possible sequence of configurations the machine can pass before it halts. Recall that $\mathcal{S}$ is initialized in a state $s_0 \approx c_0$. Since transitions from states corresponding to configurations comply with δ, the only transition from $s_i \approx c_i$, where $0 \leq i < n$, is to $s_{i+1} \approx c_{i+1}$. No transition is possible from $s_n \approx c_n$ as $\mathit{state} = \texttt{enc}(q_{rej})$. Therefore, the only reachable states in $\mathcal{S}$ are $s_0 \approx c_0, \dots, s_n \approx c_n$. Surely, the active state in all configurations $c_0, \dots, c_{n-1}$ is different from q_{acc} else w would be accepted by $\mathcal{M}$. Thus for all $i \in \{0, \dots, n\}$, $s_i \not\models R$. Hence, R is not reachable in $\mathcal{S}$.

Next, let us show that $f_{\mathcal{M}}$ is computable in polynomial time. Given an arbitrary word $w \in \Sigma^*$ on input, consider the size of the unary-encoded output $(\mathcal{S}, R) = f_{\mathcal{M}}(w)$ with respect to the input size. First, recall that $m = p(|w|) + c$ is a polynomial in $|w|$ and $n = \lceil \log_2 |Q| \rceil$, $k = \lceil \log_2 |\Gamma| \rceil$ are fixed constants. Then consider the unary-encoded size of the variable set X,

$$\begin{aligned} s_{enc}(X) &= |X| + s_{enc}([m \to k]) + s_{enc}([m]) + s_{enc}([n]) \\ &= 3 + (1 + (m+1) + (k+1)) + (1 + (m+1)) + (1 + (n+1)) \\ &= 10 + 2m + n + k\,. \end{aligned}$$

Clearly, $s_{enc}(X)$ is polynomial with respect to $|w|$.

Second, recall that `enc` maps the symbols in Γ to constants of sort $[k]$, the unary-encoded size of which is at most $1 + \mathrm{L}(2^k) + (k+1) = 2 + 2k$, and the states in Q to constants of sort $[n]$, the unary-encoded size of which is at most $1 + \mathrm{L}(2^n) + (n+1) = 2 + 2n$. Then, consider the sizes of the unary-encoded formulas I, T and R expressed bellow.

First, consider the size of the unary-encoded formula I,

$$\begin{aligned}|I| = &3 + |\texttt{clr_tape}| + \sum_{i=0}^{|w|-1} \Big(1 + \mathrm{L}(i+2) + (m+1) + |\texttt{enc}(w_i)|\Big) \\ &+ 4 + \mathrm{L}(1) + (m+1) \\ &+ 2 + |\texttt{enc}(q_0)|\,.\end{aligned}$$

Since

$$\begin{aligned}|\texttt{clr_tape}| &= 1 + (1 + (m+1) + |\texttt{enc}(\sqcup)|) + (1 + \mathrm{L}(1) + (m+1)) + |\texttt{enc}(\triangleright)| \\ &\leq 9 + 2m + 4k\,,\end{aligned}$$

we get that

$$\begin{aligned}|I| \leq &12 + 2m + 4k + \sum_{i=0}^{|w|-1} \Big(1 + \mathrm{L}(i+2) + (m+1) + (2+2k)\Big) \\ &+ 6 + (m+1) + (2+2n)\,.\end{aligned}$$

Hence, $|I|$ is polynomial with respect to $|w|$.

Next, consider the size of the unary-encoded formula T,

$$|T| \;=\; 7 + |\texttt{enc}(q_{acc})| + |\texttt{enc}(q_{rej})| + \sum_{(q,a)\,\in\, Q\smallsetminus\{q_{acc},q_{rej}\}\times\Gamma} \Big(1 + |\texttt{trans}(q,a)|\Big)\,.$$

As $|\texttt{shift}(pos, d)| = |pos' = pos \pm 1^{[m]}| = 5 + \mathrm{L}(2) + (m+1) = 7 + m$, it follows that

$$\begin{aligned}|\texttt{trans}(q,a)| &= 1 + (7 + |\texttt{enc}(q)| + |\texttt{enc}(a)|) + (11 + |\texttt{enc}(q')| + |\texttt{enc}(a')| + 7 + m) \\ &\leq 34 + m + 4n + 4k\,.\end{aligned}$$

Then

$$|T| \leq 11 + 4n + (|Q| - 2) \cdot |\Gamma| \cdot (34 + m + 4n + 4k)\,.$$

Hence, $|T|$ is polynomial with respect to $|w|$.

Finally, consider the size of the unary-encoded formula R,

$$|R| = 2 + |\texttt{enc}(q_{acc})| \leq 4 + 2n\,.$$

Clearly, $|R|$ is also polynomial with respect to $|w|$.

Since the size of all the output components is polynomial with respect to $|w|$ and all subexpressions that depend on w can be evaluated in polynomial time, there is a deterministic algorithm that computes $f_{\mathcal{M}}$ in polynomial time. Then, $\mathcal{P} \leq_p$ VMT(QF_ABV$_1$) for an arbitrary $\mathcal{P}$ in ExpSpace. Hence, VMT(QF_ABV$_1$) is ExpSpace-hard. □

Proposition 1. *Let $\mathcal{S}$ be a transition system and let s be a state reachable in $\mathcal{S}$. Then a trace $s_0 \ldots s_n$ exists in $\mathcal{S}$ such that $s_n = s$ and all states $s_0, \ldots, s_n$ are distinct.*

Proof. Since s is reachable is $\mathcal{S}$, there is a trace $s_0 \dots s_n$ in $\mathcal{S}$ such that $s_n = s$ for some $n \in \mathbb{N}$. Consider the shortest of such traces. If there are multiple shortest traces, pick any one of them. If not all states $s_0, \dots, s_n$ are distinct, then $s_i = s_j$, for some $0 \leq i < j \leq n$. Thus $s_0 \dots s_i s_{j+1} \dots s_n$ is a shorter trace such that $s_n = s$, which contradicts the assumption that $s_0 \dots s_n$ is the shortest of such traces. Hence, all states $s_0, \dots, s_n$ are distinct. □

Lemma 2. VMT(QF_ABV$_1$) $\in$ NExpSpace .

Proof. Given a transition system $\mathcal{S} := (X, I, T)$ over QF_ABV and a reachability property $R(X)$, let $|Q|$ be the size of the state-space of $\mathcal{S}$ and consider the algorithm bellow. The algorithm accepts the input $(\mathcal{S}, R)$ if some non-deterministic run ends by accepting.

Algorithm 1 A deciding algorithm for VMT over QF_ABV

Input: $\mathcal{S}$, R

Output: accepts if R is reachable in $\mathcal{S}$; rejects otherwise

1. Check that $|Q| > 0$; if not, reject.
2. Pick a state s non-deterministically.
3. Check that $s \models I$; if not, reject.
4. Set *counter* to 1.
5. If $s \models R$, accept.
6. If *counter* $= |Q|$, reject.
7. Pick a state s' non-deterministically.
8. Check that the assignment of variables in s, s', respectively, satisfies T; if not, reject.
9. Let $s := s'$.
10. Increase *counter* by one.
11. Repeat steps 5.–11.

If R is reachable in $\mathcal{S}$, then a trace in $\mathcal{S}$ ends in a state s_n such that $s_n \models R$. From Proposition 1, it follows that there is a trace $s_0 \dots s_n$ in $\mathcal{S}$ such that $s_n \models R$ and all states $s_0, \dots, s_n$ are distinct. Then $n < |Q|$, so there is a choice of states such that the run accepts. Thus, the algorithm accepts the input $(\mathcal{S}, R)$ if R is reachable in $\mathcal{S}$. Clearly, the algorithm does not accept unless R is reachable in $\mathcal{S}$. If a state satisfying R

is not reached in $|Q| - 1$ iterations of the loop, the non-deterministic run rejects. Thus, the algorithm halts on every input. Hence, the non-deterministic algorithm *decides* the VMT(QF_ABV$_1$) problem.

Next, we show that the algorithm can be realized in exponential space given the unary encoding of the input. The size $|Q|$ of the state-space of $\mathcal{S}$ is obtained from the sort specification of variables in X in the following way. Given a fixed variable set $X = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$, the state space Q corresponds to the set $\sigma_1 \times \cdots \times \sigma_n$, thus $|Q| = \prod_{i=1}^{n} |\sigma_i|$, where

$$|\sigma_i| = \begin{cases} 2^m & \text{if } \sigma_i = [m]\,, \\ 2^{l \cdot 2^k} & \text{if } \sigma_i = [k \to l]\,. \end{cases}$$

Therefore, $|Q|$ is potentially double-exponential with respect to the index-sort bit-widths of array variables contained in X. Assume $|Q|$ is represented with a binary-encoded integer in the algorithm. Then, the space needed to store $|Q|$ is not larger than exponential with respect to the index-sort bit-widths. Given the unary encoding of the input, this space is at most exponential with respect to the input size. Hence, $|Q|$ and consequently also *counter* can be stored in exponential space.

The size of variables s and s' in memory equals to the space needed to store an assignment of the variables in X. A value of a bit-vector variable can be stored in a space equal to its bit-width. A value of sort $[k \to l]$ can be represented by 2^k bit-vectors of bit-width l. Thus, the space needed to store an array value is $2^k \cdot l$. Since an assignment can be represented as a string of values, its size is not larger than exponential with respect to the unary-encoded bit-widths. Hence, the states s, s' can be stored in exponential space.

All operations on integers performed by the algorithm can be done in exponential space. Lastly, consider the space needed to check if states satisfy the formulas I, R, and T. The size of the formulas' syntactic trees is linear with respect to their encoded size, and each node represents either a bit vector or an array. Then, each node is at most exponential with respect to the encoded input size. Consequently, checking that states satisfy the formulas requires, at most, exponential space.

Thus, the non-deterministic algorithm deciding VMT(QF_ABV$_1$) is realizable in exponential space. Hence, VMT(QF_ABV$_1$) $\in$ NExpSpace. □

4.2 Results for binary encodings

Theorem 2. VMT(QF_ABV$_2$) *is* 2ExpSpace-*complete.*

Proof. In Lemma 3, we prove that VMT(QF_ABV$_2$) is 2ExpSpace-hard with a suitable modification of the polynomial reduction from Lemma 1. In Lemma 4, we prove that VMT(QF_ABV$_2$) $\in$ 2ExpSpace with a revision of the algorithm proposed in Lemma 2. Thus, Lemma 3 and Lemma 4 together imply that the VMT(QF_ABV$_2$) problem is 2ExpSpace-complete. □

Lemma 3. VMT(QF_ABV$_2$) *is* 2ExpSpace-*hard.*

Proof. Let $\mathcal{P} \subseteq \Sigma^*$ be an arbitrary decision problem in 2ExpSpace. Therefore, such deterministic TM exists that runs in 2ExpSpace and decides the problem $\mathcal{P}$. Let $\mathcal{M} := (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, q_{acc}, q_{rej})$ be such a TM. Then, consider the reduction $f_{\mathcal{M}}$ proposed in Lemma 1. Let us show a modification $f'_{\mathcal{M}}$ of $f_{\mathcal{M}}$ that for any word w over Σ, returns a transition system $\mathcal{S} = (X, I, T)$ over QF_ABV$_2$ and a reachability property $R(X)$, such that

$$w \in \mathcal{P} \iff R \text{ is reachable in } \mathcal{S}\,.$$

Next, we will show that the modified function $f'_{\mathcal{M}}$ is also computable in polynomial time, hence $\mathcal{P} \leq_p$ VMT(QF_ABV$_2$).

Since $\mathcal{M}$ runs in 2ExpSpace, a polynomial p exists such that $S_{\mathcal{M}}(n) \in \mathcal{O}(2^{2^{p(n)}})$. Then $S_{\mathcal{M}}(n) \leq c \cdot 2^{2^{p(n)}}$ for some $c \in \mathbb{N}$. Let p, c be fixed. Then, consider a function $f'_{\mathcal{M}}(w) = (\mathcal{S}, R)$, where all parts of the output are obtained as in $f_{\mathcal{M}}(w)$ with two changes:

- The constant m that specifies the bit-widths of the variable *pos* and the index sort of *tape* is now $2^{p(|w|)} + c$ instead of $p(|w|) + c$.
- The output is binary instead of unary encoded.

Recall that $\mathcal{M}$ does not visit more than $c \cdot 2^{2^{p(|w|)}}$ tape squares given the input w, since $S_{\mathcal{M}}(|w|) \leq c \cdot 2^{2^{p(|w|)}}$. Then, the array *tape* of sort $[m \to k]$ is large enough to contain all tape squares visited in computation since $[m]$ contains $2^{c+2^{p(|w|)}} = 2^c \cdot 2^{2^{p(|w|)}} \geq c \cdot 2^{2^{p(|w|)}}$ different indices. Then, of course, the bit-vector *pos*: $[m]$ is large enough to capture the position of $\mathcal{M}$'s head at any step. The constants n, k do not depend on the TM's complexity and thus work the same as in the original reduction.

Then, by the arguments made in Lemma 1, we can see that the function $f'_{\mathcal{M}}$ reduces the problems in 2ExpSpace to VMT over QF_ABV. Let us show that $f'_{\mathcal{M}}$ is computable in polynomial time if the output is binary-encoded. Recall that $m = 2^{p(|w|)} + c$ and that $n = \lceil \log_2 |Q| \rceil$, $k = \lceil \log_2 |\Gamma| \rceil$ are fixed constants. Given an input word $w \in \Sigma^*$, consider the size of the binary-encoded output $(\mathcal{S}, R) = f'_{\mathcal{M}}(w)$ with respect to $|w|$.

The size of the binary-encoded variable set X is

$$\begin{aligned} s_{enc}(X) &= |X| + s_{enc}([m \to k]) + s_{enc}([m]) + s_{enc}([n]) \\ &= 3 + (3 + \mathrm{L}(m+1) + \mathrm{L}(k+1)) + (2 + \mathrm{L}(m+1)) + (2 + \mathrm{L}(n+1)) \\ &= 10 + 2 \cdot \mathrm{L}(m+1) + \mathrm{L}(n+1) + \mathrm{L}(k+1)\,. \end{aligned}$$

Since $\mathrm{L}(m+1) = \mathrm{L}(2^{p(|w|)} + c + 1) \leq \mathrm{L}(2^{p(|w|)} \cdot (2+c)) \leq p(|w|) + \mathrm{L}(2+c)$, clearly, $s_{enc}(X)$ is polynomial with respect to $|w|$.

Recall that `enc` maps the symbols in Γ to constants of sort $[k]$ and the states in Q to constants of sort $[n]$. The size of an arbitrary constant of sort $[k]$ in binary encoding is at most $1 + \mathrm{L}(2^k) + 1 + \mathrm{L}(k+1) = 2 + k + L(k+1)$. Analogically, the size of a binary-encoded constant of sort $[n]$ is at most $2 + n + \mathrm{L}(n+1)$. Then, consider the sizes of formulas I, T, and R expressed below.

For the size of the binary-encoded formula I, it holds that

$$\begin{aligned}|I| \;\leq\; & 3 + |\texttt{clr_tape}| + \sum_{i=0}^{|w|-1} \Big(4 + \mathrm{L}(i+2) + \mathrm{L}(m+1) + k + L(k+1)\Big) \\ & + 5 + \mathrm{L}(1) + \mathrm{L}(m+1) \\ & + 4 + n + \mathrm{L}(n+1)\,.\end{aligned}$$

Here, the size of `clr_tape` is

$$\begin{aligned}|\texttt{clr_tape}| &= 1 + (2 + \mathrm{L}(m+1) + |\texttt{enc}(\sqcup)|) + (2 + \mathrm{L}(1) + \mathrm{L}(m+1)) + |\texttt{enc}(\triangleright)| \\ &\leq 9 + 2 \cdot \mathrm{L}(m+1) + 2k + 2 \cdot \mathrm{L}(k+1)\,.\end{aligned}$$

Thus, $|I|$ is polynomial with respect to $|w|$.

For the size of the binary-encoded formula T, it holds that

$$|T| \;\leq\; 9 + 2n + 2 \cdot \mathrm{L}(n+1) + \sum_{(q,a)\,\in\, Q \setminus \{q_{acc}, q_{rej}\} \times \Gamma} \Big(1 + |\texttt{trans}(q,a)|\Big)\,.$$

In binary encodings, $|\texttt{shift}(pos, d)| = 6 + \mathrm{L}(2) + \mathrm{L}(m+1) = 7 + \mathrm{L}(m+1)$, thus

$$|\texttt{trans}(q,a)| \leq 34 + 2n + 2 \cdot \mathrm{L}(n+1) + 2k + 2 \cdot \mathrm{L}(k+1) + \mathrm{L}(m+1)\,.$$

Then, $|T|$ is polynomial with respect to $|w|$.

Finally, for the size of the binary-encoded formula R it holds that,

$$|R| = 2 + |\texttt{enc}(q_{acc})| \leq 4 + n + \mathrm{L}(n+1)\,.$$

Clearly, $|R|$ is also polynomial with respect to $|w|$.

Since the size of all the output components is polynomial with respect to $|w|$ and all subexpressions depending on w can be evaluated in polynomial time, there is a deterministic algorithm deciding $f'_{\mathcal{M}}$ in polynomial time. Then, $\mathcal{P} \leq_p \mathrm{VMT}(\mathrm{QF_ABV}_2)$ for an arbitrary $\mathcal{P}$ in 2ExpSpace. Hence, $\mathrm{VMT}(\mathrm{QF_ABV}_2)$ is 2ExpSpace-hard. □

Lemma 4. $\mathrm{VMT}(\mathrm{QF_ABV}_2) \in$ 2ExpSpace.

Proof. Consider Algorithm 1 proposed in Lemma 2. We have shown that the algorithm decides the VMT problem given an arbitrary transition system $\mathcal{S} := (X, I, T)$ over $\mathrm{QF_ABV}_1$ and a reachability property $R(X)$. Let us show that the algorithm is realizable in 2ExpSpace given a binary-encoded input.

In Lemma 2, it has been shown that the value of $|Q|$ is at most double exponential with respect to the index-sort bit-widths of array variables contained in X. In binary encodings, the encoded size of bit-widths is logarithmic with respect to their value. Then, the value of $|Q|$ is potentially triple exponential with respect to the binary-encoded bit-widths and consequently to the input size. However, both $|Q|$ and *counter* can be stored in double exponential space if represented as binary-encoded integers.

As shown in Lemma 2, the variables s and s' can be stored in a space that is exponential with respect to the index-sort bit-widths of array variables in X. Since these bit-widths are exponential with respect to their binary-encoded size, the variables s, s' can be stored in double exponential space.

Clearly, all used operations on integers can be performed in double exponential space. Checking that states satisfy formulas I, R, and T can be done in double exponential space, as the size of the formulas' syntactic trees is still linear with respect to their encoded length and the size of nodes representing bit vectors or arrays is now at most double exponential with respect to the encoded bit-widths.

There is thus a non-deterministic algorithm deciding VMT(QF_ABV$_2$) in double exponential space. Then, by Savitch's theorem, there is also a deterministic algorithm deciding VMT(QF_ABV$_2$) in double exponential space. Hence, we have shown that VMT(QF_ABV$_2$) $\in$ 2ExpSpace. $\square$

4.3 Case for QF_ABV without constant arrays

In Section 1.5, we stated that the presence of the symbol constarr in the theory of arrays does not affect the complexity of verification over QF_ABV. Let us prove this statement by showing a modification of the reductions from Lemma 1 and Lemma 3 that eliminates the constarr symbol while preserving the reducing property and polynomial-time computability of the functions. The algorithm given in Lemma 2 clearly works for both versions of the theory of arrays.

Consider the functions f, f' described in Lemmas 1 and 3. Notice that the symbol constarr appears exactly once in their outputs – that is, in the formula I, resp. `clr_tape`. There, it serves the purpose of initializing an array filled with $\texttt{enc}(\sqcup)$ representing a tape of empty squares. We propose a modification of $\mathcal{S}$ such that I only defines the first $|w| + 1$ squares of *tape*, and T ensures that a state corresponding to the complete initial configuration of $\mathcal{M}$ is reached before the simulation of the TM's run begins. In the following definitions, we use the constants and subformulas as defined in Lemma 1 (resp. Lemma 3) unless specified otherwise.

First, we introduce a fourth variable *running* of sort [1] into the variable set X. While *running* is set to 0, the system executes the steps needed to reach a state corresponding to the initial configuration of $\mathcal{M}$. Once *running* is set to 1, the simulation of the TM's run itself begins.

Second, we modify the formula I as follows:

$$I := \Big(\operatorname{read}(\mathit{tape}, 0^{[m]}) = \texttt{enc}(\triangleright) \wedge \bigwedge_{i=1}^{l} \operatorname{read}(\mathit{tape}, i^{[m]}) = \texttt{enc}(w_{i-1}) \wedge$$
$$\mathit{pos} = (l+1)^{[m]} \wedge \mathit{state} = \texttt{enc}(q_0) \wedge \mathit{running} = 0^{[1]} \Big) .$$

Notice that $\mathcal{S}$ now has multiple initial states, as the content of *tape* past the index $l^{[m]}$ can be arbitrary. Notice also that the initial value of *pos* is modified.

Next, let

$$
\begin{aligned}
T := \Big(\Big(& \mathit{running} = 0^{[1]} \wedge \mathit{tape}' = \mathrm{write}(\mathit{tape}, \mathit{pos}, \mathtt{enc}(\sqcup)) \wedge \mathit{pos}' = \mathit{pos} + 1^{[m]} \wedge \\
& \mathit{state}' = \mathit{state} \wedge (\mathit{pos} = 0^{[m]} - 1^{[m]} \leftrightarrow \mathit{running}' = 1^{[1]})\Big) \vee \\
& \Big(\mathit{running} = 1^{[1]} \wedge \mathit{running}' = 1^{[1]} \wedge \\
& \mathit{state} \neq \mathtt{enc}(q_{acc}) \wedge \mathit{state} \neq \mathtt{enc}(q_{rej}) \wedge \bigwedge_{(q,c) \in Q \setminus \{q_{acc}, q_{rej}\} \times \Gamma} \mathtt{trans}(q, c)\Big)\Big) .
\end{aligned}
$$

Starting in a state $s \models I$, with each transition, the content of *tape* at *pos* is changed to $\mathtt{enc}(\sqcup)$, and *pos* is incremented by one. The rest of the state remains untouched. This is repeated until *pos* reaches its maximal value. Then, the last tape square is rewritten with $\mathtt{enc}(\sqcup)$, *running* changes to 1, and *pos* is incremented once more, which sets it to 0. By that point, the state of $\mathcal{S}$ corresponds to the initial configuration of $\mathcal{M}$, and the simulation continues as in the original TS described in Lemma 1.

Finally, the reachability property R is modified to:

$$P := (\mathit{state} = \mathtt{enc}(q_{acc}) \wedge \mathit{running} = 1^{[1]}) .$$

Note that the length of traces needed to reach a state satisfying R increases exponentially in the modified TS because $2^m - 1 - |w|$ transitions are required first to reach a state where $\mathit{running} = 1^{[1]}$. However, that does not affect the complexity result as the encoded size of $\mathcal{S}$ clearly remains polynomial with respect to the input size in both encodings and the respective reductions.

In the algorithm introduced in Lemma 2, the symbol constarr can only appear in the input. The proposed algorithm works for all inputs $(\mathcal{S}, R)$ over QF_ABV. Thus, it also works for inputs that do not contain constant arrays. Hence, the complexity of model checking over QF_ABV for a version of the theory of arrays without constant arrays is the same as for the version with constant arrays.

4.4 Complexity of verifying safety properties

As was mentioned in Chapter 2, model checking of symbolic transition systems involves the verification of safety properties as well as the verification of reachability properties. It was also established that the two problems are dual, i.e., a safety property P holds in a system $\mathcal{S}$, whenever a reachability property $R = \neg P$ does not hold in $\mathcal{S}$. Here, we show that the complexity of safety-property verification is the same as the complexity of reachability-property verification in the respective encodings.

In order to prove the ExpSpace-hardness resp. 2ExpSpace-hardness of safety property verification over QF_ABV, we exchange the reachability property R for the safety property $P := \mathit{state} \neq q_{rej}$ in the reductions proposed in Lemmas 1 and 3. Recall the safety property has to be satisfied in all reachable states of the system. Since $\mathcal{M}$ is a deciding TM, it reaches a rejecting state if and only if $w \notin \mathcal{L}(\mathcal{M})$. Previously, we have shown that the transition systems described by the reductions simulate the run of $\mathcal{M}$

for all words $w \in \Sigma^*$. Thus, given an arbitrary input word, the system $\mathcal{S}$ satisfies P if and only if $w \in \mathcal{L}(\mathcal{M})$. Clearly, the reductions remain polynomial, which means verification of safety properties over QF_ABV is ExpSpace-hard given a unary encoding and 2ExpSpace-hard given a binary encoding.

Given a TS $\mathcal{S} = (X, I, T)$ over QF_ABV and safety propery $P(X)$, consider a deterministic algorithm that checks the system $\mathcal{S}$ for the reachability property $R = \neg P$ and **rejects** if R is reachable in $\mathcal{S}$; otherwise, it **accepts**. Since this algorithm differs from a reachability-property verification algorithm only by the negation of the property and flipping of the output, it clearly exists and is realizable in exponential resp. double exponential space depending on the encoding. Then, the problem of model-checking transition systems for safety properties lies in ExpSpace given a unary encoding and in 2ExpSpace given a binary encoding of the input.

Hence, the verification of safety properties over QF_ABV$_1$ is ExpSpace-complete, and the verification of safety properties over QF_ABV$_2$ is 2ExpSpace-complete.

Conclusion

This thesis has analyzed the complexity of word-level model checking over quantifier-free bit-vector logic with arrays. We have shown the problem to be ExpSpace-complete when its instances are unary-encoded and 2ExpSpace-complete when its instances are binary-encoded. Furthermore, we established that these results hold for both common versions of the theory of arrays: the version with constant arrays and the version without them. Additionally, we showed that the complexity of verifying reachability properties matches that of verifying safety properties.

These findings expand the understanding of the complexity of symbolic verification over bit-vector logics and contribute to the theoretical foundation for developing efficient software model-checking tools. Notably, they address a large class of practically relevant VMT instances for which the complexity was previously unknown, as the problem fragments studied earlier did not allow word-level modeling of memories and array-like data structures.

Future Work

While this work addressed the complexity of arguably the most practically significant fragment of word-level model checking with arrays, in future research, we would like to focus on other fragments of the logic to provide a more complete understanding. In particular, we aim to explore logics involving multi-dimensional arrays, i.e., arrays indexed with arrays where we expect the complexity to be even higher.

Bibliography

[1] Gergely Kovásznai, Helmut Veith, Andreas Fröhlich, and Armin Biere. "On the complexity of symbolic verification and decision problems in bit-vector logic". In: *International Symposium on Mathematical Foundations of Computer Science*. Springer. 2014, pp. 481–492.

[2] Alessandro Cimatti, Alberto Griggio, Sergio Mover, Marco Roveri, and Stefano Tonetta. "Verification modulo theories". In: *Formal Methods in System Design* 60.3 (2022), pp. 452–481.

[3] David Marker. "Model Theory: An Introduction". In: 217 (Jan. 2002). DOI: 10.1007/b98860.

[4] J.R. Shoenfield. *Mathematical Logic*. Taylor & Francis, 2001. ISBN: 9781568811352.

[5] María Manzano and Víctor Aranda. "Many-Sorted Logic". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta and Uri Nodelman. Winter 2022. Metaphysics Research Lab, Stanford University, 2022.

[6] Martin Jonáš. "Satisfiability of Quantified Bit-Vector Formulas: Theory and Practice". PhD thesis. Masarykova univerzita, Fakulta informatiky, 2019.

[7] Liana Hadarean. "An efficient and trustworthy theory solver for bit-vectors in satisfiability modulo theories". PhD thesis. New York University, 2015.

[8] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Satisfiability, Second Edition*. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, Feb. 2021. Chap. 33, pp. 825–885.

[9] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. www.SMT-LIB.org. 2016.

[10] Armin Biere. "Tutorial on World-Level Model Checking". In: *2020 Formal Methods in Computer Aided Design (FMCAD)*. 2020. DOI: 10.34727/2020/isbn.978-3-85448-042-6_3.

[11] Michael Sipser. *Introduction to the Theory of Computation*. Third. Boston, MA: Course Technology, 2013. ISBN: 113318779X.

[12] D.P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice-Hall international series in civil engineering and engineering mechanics. Prentice Hall, 1994. ISBN: 9780139153808.

[13] Walter J Savitch. "Relationships between nondeterministic and deterministic tape complexities". In: *Journal of computer and system sciences* 4.2 (1970), pp. 177–192.

[14] P. van Emde Boas and Facultaire Vakgroep Informatica (Amsterdam). *Machine Models and Simulations.* University of Amsterdam, Computer Science Department, 1987. URL: `https://books.google.cz/books?id=TS5CswEACAAJ`.

[15] Gergely Kovásznai, Andreas Fröhlich, and Armin Biere. "Complexity of fixed-size bit-vector logics". In: *Theory of Computing Systems* 59 (2016), pp. 323–376.