# The Complexity of Computing the Behaviour of Lattice Automata on Infinite Trees

Karsten Lehmann[a], Rafael Peñaloza[b]

[a] *Optimisation Research Group, NICTA*
*Artificial Intelligence Group, Australian National University*
[b] *Theoretical Computer Science, TU Dresden, Germany*
*Center for Advancing Electronics Dresden*

**Abstract**

Several logic-based decision problems have been shown to be reducible to the emptiness problem of automata. In a similar way, non-standard reasoning problems can be reduced to the computation of the behaviour of weighted automata. In this paper, we consider a variant of weighted Büchi automata working on (unlabeled) infinite trees, where the weights belong to a lattice. We analyse the complexity of computing the behaviour of this kind of automata if the underlying lattice is not distributive.

We show that the decision version of this problem is in ExpTime and PSpace-hard in general, assuming that the lattice operations are polynomial-time computable. If the lattice is what we call linear-space-computable-encoded, then the upper bound can be reduced to PSpace, but the lower bound also decreases to NP-hard and co-NP-hard. We conjecture that the upper bounds provided are in fact tight.

*Keywords:* Weighted Automata, Behaviour Computation, Lattices, Complexity

## 1. Introduction

Automata have long been recognized as tools for solving logic-based reasoning tasks. Beyond well-known results on the equivalence of recognizable and MSOL-definable languages [10, 20], automata working on infinite inputs have been successfully applied to decide satisfiability of Linear Temporal Logic (LTL) [31, 22] formulas, and reason with Description Logic (DL) [2] knowledge bases, to name just two well-known examples. The main idea in both cases is to construct an automaton $\mathcal{A}$ that accepts all the (well-structured) models of the input, and perform an emptiness test on $\mathcal{A}$. The constructed automaton is a generalized Büchi automaton on infinite words in the case of LTL [38], and a

*Email addresses:* karsten.lehmann@nicta.com.au (Karsten Lehmann), penaloza@tcs.inf.tu-dresden.de (Rafael Peñaloza)

looping automaton (that is, a Büchi automaton where all states are accepting) on infinite trees for DL reasoning [3, 7, 11, 28].[1]

In most of these constructions, it is possible to use a simplified alphabet having only one symbol. Additional alphabet symbols can be encoded within the set of states of the automaton, and in this case the relevant models are described by the *accepting runs* of the automaton, rather than by the recognized language.

Automata-based decision procedures have been generalized to weighted automata over lattices as a means to deal with non-standard reasoning problems, such as axiom-pinpointing [6, 24], access control [4, 25], or context-based reasoning [5], as well as with non-standard semantics like fuzzy [9, 34, 35] and possibilistic semantics [30, 32]. The idea behind these constructions is that every model can be associated to a "weight" corresponding to the non-standard task. For example, in the axiom-pinpointing scenario, where one is interested in finding the causes of an inconsistency, this weight will be the set of axioms violated by the model.[2] We are then interested in finding the supremum of the weights of all these models, which in the case of axiom-pinpointing will be the set of all sets of axioms that prevent the existence of a model.

Suppose that we can associate every transition of the constructed automaton with a weight in such a way that the infimum of the weights of all transitions appearing in a successful run (that is, the weight of this successful run) corresponds exactly to the weight of the model it represents, as described before. Then, this kind of non-standard reasoning reduces to a computation of the behaviour of the weighted automaton. To fully understand the complexity of non-standard reasoning tasks, we need to study how hard it is to compute the behaviour of lattice automata. Thus, we are interested in the complexity of computing the behaviour of Büchi automata on infinite trees, whose weights belong to a lattice.

For distributive lattices, it is known that the behaviour of generalized Büchi automata can be computed in polynomial time [6, 16], matching the complexity of deciding emptiness of (unweighted) Büchi automata [33, 37]. This result provides tight upper bounds for the complexity of axiom-pinpointing in expressive DLs, and of reasoning in special kinds of fuzzy and possibilistic DLs and LTL. Unfortunately, distributivity of the lattice is not always a valid assumption. For example, in access control the underlying access lattice is often provided by the security manager, or automatically generated from a compact description of the access rights needed [14], and it can take any shape. In this paper we study the complexity of computing the behaviour in case the lattice is not distributive. We notice that without distributivity, the automata are not any more *weighted automata* in the standard sense, as defined in [26, 27, 17]. Variants of weighted automata on finite [18] and infinite [19] trees, in which the distribu-

[1] Other automata models have been considered in the literature, e.g. [12]. For the sake of brevity and clarity, in this paper we focus only on those based on Büchi automata.

[2] The actual weights used are slightly more complex than described here. For the full details see [6].

tivity assumption is dropped have been recently studied; in fact, the underlying algebra has been generalized to more complex valuation monoids [15]. However, those papers focus mostly on the expressivity of the automata, and their relation with weighted logics. To the best of our knowledge, there has been no systematic study of the complexity of computing the behaviour of automata over non-distributive structures.

We show that the behaviour of automata over arbitrary lattices can be computed by a simple "black-box" mechanism that tests emptiness of exponentially many unweighted Büchi automata. This yields an exponential time upper bound for the computation of the behaviour, assuming that lattice operations can be performed in polynomial time. Unfortunately, the best-case running time of this algorithm is also exponential on the number of different weights appearing in the automation. If the lattice can be represented in such a way that its operations do not increase the space requirements (a condition we call *linear-space-computable-encoded*), then this upper bound can be improved to polynomial space. The exponential upper bound for general lattices is not new; in fact, it is a simple consequence of the results from [19], where it was shown that every recognizable tree language over bi-locally finite strong bimonoids can be expressed as a recognizable step-function; i.e., the behaviour of every automaton over such strong bimonoids can be described as a finite weighted sum of languages accepted by (unweighted) automata. The tighter upper bound for the class of linear-space-computable-encoded lattices, on the other hand, was previously unknown.

Regarding lower bounds, we provide a linear-space-computable-encoded lattice $L_{\mathsf{sat}}$ and show that computing the behaviour of automata over this lattice is hard for the classes NP and co-NP. We further improve the lower bound for general lattices by providing a lattice $L_{\mathsf{qbf}}$ for which computing the behaviour is PSPACE-hard. This second lattice, however, is not linear-space-computable-encoded. The best previously known lower bound for the complexity of computing the behaviour was the polynomial-time lower bound obtained for distributive lattices [6, 16]. Our results show that dropping the distributivity does increase the complexity of the problem. To the best of our efforts, we were unable to close the gap between the lower and upper bounds found; however, we conjecture that the upper bounds are tight.

The paper is divided as follows. We first recall basic concepts from lattice and automata theory, and formally define the decision problem we study. Then, in Section 3 we provide upper bounds for the complexity of this problem by means of an algorithm. The lower bounds are provided in Section 4, before concluding the paper.

## 2. Lattice Tree Automata

We study a simple class of weighted automata that receive as input infinite unlabeled trees of a fixed arity $k$, and use elements of a possibly infinite lattice as weights. For a positive integer $k$, we denote the set $\{1, \ldots, k\}$ by $[k]$. We identify the nodes of an infinite tree by words from $[k]^*$ in the usual way: the

root node is represented by the empty word $\varepsilon$, and the $i$-th successor of a node $u$ is represented by $ui$ for $i, 1 \leq i \leq k$. In the case of labeled trees, we refer to the labeling of the node $u \in [k]^*$ in the tree $r$ by $r(u)$. An infinite tree $r$ with labels from a set $Q$ can be described as a function $r : [k]^* \to Q$.

As previously said, we consider only unlabeled trees as inputs for our automata. Given an arity $k$, there is exactly one such tree, that we simply call $[k]^*$. We will often refer to paths in this tree. A *path* is a subset $\mathsf{p} \subseteq [k]^*$ that contains the empty word ($\varepsilon \in \mathsf{p}$), is closed under prefixes (i.e., if $ui \in \mathsf{p}$, then $u \in p$ for every $u \in [k]^*, i \in [k]$), and every node has exactly one successor (that is, if $u \in \mathsf{p}$, then there is exactly one $i \in [k]$ with $ui \in \mathsf{p}$).

We call the unary tree, where $k = 1$, an *infinite word*. Notice that an infinite word has exactly one path that is equivalent to the word itself. As usual, we will often represent an infinite word with labels from a set $Q$ as an infinite sequence of elements from $Q$.

A *lattice* [23] is an algebraic structure $(L, \vee, \wedge)$ over a *carrier set* $L$ with the two binary operations *join* $\vee$ and *meet* $\wedge$ that are idempotent, associative, and commutative and satisfy the absorption laws $\ell_1 \vee (\ell_1 \wedge \ell_2) = \ell_1 = \ell_1 \wedge (\ell_1 \vee \ell_2)$ for all $\ell_1, \ell_2 \in L$. The lattice is called *distributive* if meets and joins distribute over each other; i.e. $(\ell_1 \vee \ell_2) \wedge \ell_3 = (\ell_1 \wedge \ell_3) \vee (\ell_2 \wedge \ell_3)$ holds for every $\ell_i \in L, 1 \leq i \leq 3$, and dually interchanging the $\vee$ and $\wedge$ operators.

The lattice operations induce a partial order $\leq$ on $L$, defined by $\ell_1 \leq \ell_2$ iff $\ell_1 \wedge \ell_2 = \ell_1$ for all $\ell_1, \ell_2 \in L$. As usual, we write $\ell_1 < \ell_2$ if $\ell_1 \leq \ell_2$ and $\ell_1 \neq \ell_2$. A subset $T \subseteq L$ is called an *antichain (in $L$)* if there are no two elements $\ell_1, \ell_2 \in T$ with $\ell_1 < \ell_2$. When it is clear from the context, we will often use the carrier set $L$ to denote the lattice $(L, \vee, \wedge)$.

We consider only lattices that are *bounded*; that is, where there are two elements $\mathbf{0}$ and $\mathbf{1}$ such that $\mathbf{0} \leq \ell \leq \mathbf{1}$ holds for every $\ell \in L$. Notice that any lattice $(L, \vee, \wedge)$ can be extended to a bounded lattice $(L', \vee, \wedge)$ by simply setting $L' = L \cup \{\mathbf{0}, \mathbf{1}\}$ and $\mathbf{0} \vee \ell = \ell = \mathbf{1} \wedge \ell$, $\mathbf{0} \wedge \ell = \mathbf{0}$ and $\mathbf{1} \vee \ell = \mathbf{1}$, for all $\ell \in L'$.

For the investigation of the complexity of problems, it is necessary to consider the size of a lattice element. To do that we assume that the lattice elements are represented as finite strings over some finite alphabet $\Sigma$, so $L \subseteq \Sigma^*$. We call this representation "encoding". The *size* of an element $\ell \in L$, denoted as $|\ell|$, is defined as the length of its encoding. Two lattices are *isomorphic* if they only differ in their encoding. Whenever the details of the encoding are not relevant, as in Section 3, we will simply assume that an appropriate encoding is being used. The results from Section 4 depend on a specific choice of the encoding of the elements, which we then describe in detail.

Before formally defining lattice Büchi tree automata, we recall the notion of (unweighted) Büchi tree automata.

**Definition 2.1** (tree automata)**.** A *Büchi tree automaton (BTA)* for arity $k$ is a tuple $\mathcal{A} = (Q, I, \Delta, F)$ where

- $Q$ is a finite set of *states*,

- $I \subseteq Q$ is the set of *initial states*,
- $\Delta \subseteq Q^{k+1}$ is the set of *transitions*, and
- $F \subseteq Q$ is the set of *accepting states*.

A *run* of the BTA $\mathcal{A}$ is a labeled tree $r : [k]^* \to Q$ such that for every $u \in [k]^*$, $(r(u), r(u1), \ldots, r(uk)) \in \Delta$. The run $r$ is *successful* if, for every path $\mathsf{p}$, there are infinitely many nodes $u \in \mathsf{p}$ with $r(u) \in F$. We say that $\mathcal{A}$ is *not empty*, in symbols $\mathcal{L}(\mathcal{A}) \neq \emptyset$, if there is at least one successful run $r$ of $\mathcal{A}$ with $r(\varepsilon) \in I$.

It is well known that emptiness of an BTA $\mathcal{A}$ can be decided in polynomial time on the number of states of $\mathcal{A}$ [33, 37]. Lattice tree automata are a generalization of tree automata, in which transitions are associated to a *weight* from a given lattice $L$. Rather than deciding the existence of a successful run, we are interested in computing the so-called behaviour, which accumulates the weights of all existing successful runs.

**Definition 2.2** (lattice tree automata)**.** Let $L$ be a lattice. A *lattice Büchi tree automaton (LBTA)* over $L$ for arity $k$ is a tuple $\mathcal{A} = (Q, \mathsf{in}, \mathsf{wt}, F)$ where

- $Q$ is a finite set of *states*,
- $\mathsf{in} : Q \to L$ is the *initial distribution*,
- $\mathsf{wt} : Q^{k+1} \to L$ assigns a *weight* to every transition, and
- $F \subseteq Q$ is the set of *accepting states*.

A *run* of the LBTA $\mathcal{A}$ is a labeled tree $r : [k]^* \to Q$. The *weight* of this run is $\mathsf{wt}(r) := \bigwedge_{u \in [k]^*} \mathsf{wt}(r(u), r(u1), \ldots, r(uk))$. A run is *successful* if, for every path $\mathsf{p}$, there are infinitely many nodes $u \in \mathsf{p}$ with $r(u) \in F$.

Let $\mathsf{succ}_{\mathcal{A}}$ denote the set of all successful runs of $\mathcal{A}$. The *behaviour* of the automaton $\mathcal{A}$ is

$$\|\mathcal{A}\| := \bigvee_{r \in \mathsf{succ}_{\mathcal{A}}} \mathsf{in}(r(\varepsilon)) \wedge \mathsf{wt}(r),$$

where as usual, we define $\bigvee_{\ell \in \emptyset} \ell = \mathbf{0}$.

Büchi tree automata are special cases of lattice Büchi tree automata, where the underlying lattice is the the Boolean lattice $(\{0, 1\}, \vee, \wedge)$, where $\vee$ and $\wedge$ stand for the logical disjunction and conjunction, respectively. In that case, the functions $\mathsf{in}$ and $\mathsf{wt}$ can be seen as the characteristic functions of the sets $I$ and $\Delta$, respectively.

Notice that even if the lattice $L$ is infinite, the weights of runs and the behaviour of an LBTA are well defined. The definition of $\mathsf{wt}(r)$ requires computing the meet over an infinite set of indices; however, since $Q$ is finite and $\wedge$ is idempotent, this meet is computed over finitely many different lattice elements. Additionally, although there may exist infinitely many successful runs, there are only finitely many values that may appear as their weights. Thus, the join

that defines $\|\mathcal{A}\|$ is computed over finitely many lattice elements too. For more details, see Section 3, where we exploit these facts to develop an algorithm that computes the behaviour.

**Remark.** We are interested in finding the complexity of computing the behaviour of LBTA. To abstract from the complexity of performing lattice operations, we parameterize the problem in terms of the lattice used. To ensure that the complexity problem we study is not influenced by the lattice operations, we only consider lattices where there is a Turing-Machine that computes the join and the meet of any two given elements in time *polynomial* in the size of the given elements.

In particular, as we restrict our attention only to lattices where joins and meets are polynomially computable, we have that the size of infima and suprema is also polynomial on the size of their arguments. However, iterative computation of suprema may yield an element of exponential size. For that reason, when dealing with space complexity classes it is sometimes worth looking at a subclass of lattices where the operations do not increase the space requirements.

**Definition 2.3** (linear-space-computable-encoded)**.** A lattice $(L, \vee, \wedge)$ is called *linear-space-computable-encoded* if for every two elements $\ell_1, \ell_2 \in L$ it holds that

$$|\ell_1 \wedge \ell_2| \leq \max\{|\ell_1|, |\ell_2|\} \quad \text{and} \quad |\ell_1 \vee \ell_2| \leq \max\{|\ell_1|, |\ell_2|\}.$$

Intuitively, in a linear-space-computable-encoded lattice, the representation of the meet (respectively, join) of two elements $\ell, \ell'$ consumes at most as much space as the representation of each of these elements. This restriction will allow us to bound the space requirement throughout the computation of the behaviour, as described in the following section.

Before studying the complexity of computing the behaviour of LBTA, we briefly describe some properties of linear-space-computable-encoded lattices. Every total order is linear-space-computable-encoded since the meet and join of two elements is always one of them; in symbols, $\{\ell_1 \wedge \ell_2, \ell_2 \vee \ell_2\} = \{\ell_1, \ell_2\}$. This fact holds true independently from the choice of encoding of the elements of the lattice. Additionally, every finite lattice $L$ is isomorphic to a linear-space-computable-encoded lattice: we need only to encode the elements of the lattice in such a way that they all consume the same space. For instance, if $L$ has cardinality $n$, we can simply enumerate all the elements, and describe them using a binary string of length $\log_2(n)$ bits for each of its elements. Moreover, every countable distributive lattice can be encoded to be linear-space-computable-encoded, even if it is infinite.

**Theorem 2.4.** *Every countable distributive lattice is isomorphic to a linear-space-computable-encoded lattice.*

*Proof.* Let $L$ be a countable distributive lattice, and let $\ell_1, \ell_2, \ldots$ be an arbitrary enumeration of the elements of $L$. Let $L_n, n \in \mathbb{N}$ denote the sublattice of $L$

generated by $\{\ell_1, \ell_2, \ldots, \ell_n\}$. By distributivity, we know that $L_n \subseteq L$ has at most $2^{2^n}$ elements, for every $n \in \mathbb{N}$. We encode the elements of $L$ using binary strings as follows: $\ell_1$ is encoded by the string 1. For every $n > 1$ we encode every element in $L_n \setminus L_{n-1}$ using a unique binary string of length $2^n$, which is always possible. We now show that under this encoding, the lattice $L$ is linear-space-computable-encoded.

Let $\ell, \ell' \in L$ such that $|\ell| = 2^n, |\ell'| = 2^m$ and assume without loss of generality that $n \leq m$. Then $\ell, \ell' \in L_m$, which is the lattice generated by the first $m$ elements in the enumeration. But then $\ell \wedge \ell' \in L_m$ and $\ell \vee \ell' \in L_m$, which means that $|\ell \wedge \ell'| \leq 2^m = \max\{|\ell|, |\ell'|\}$, and $|\ell \vee \ell'| \leq 2^m = \max\{|\ell|, |\ell'|\}$, satisfying the condition for linear-space-computable-encoded. □

Notice that this construction does not work for non-distributive lattices, since the sublattice generated by a finite subset of a non-distributive lattice may be infinite; to encode all these elements, strings of larger lengths would be required, violating the conditions from Definition 2.3.

If we want to formally consider the complexity of finding the behaviour of automata, we need to reformulate the task in terms of a decision problem. We call this reformulation the *behaviour verification* problem.

**Definition 2.5.** Let $L$ be a lattice. The *behaviour verification problem* consists in deciding, given an LBTA $\mathcal{A}$ over $L$ and $\ell \in L$, whether $\|\mathcal{A}\| = \ell$.

Other related decision problems, such as e.g. deciding whether $\|\mathcal{A}\| \leq \ell$, can also be studied. The precise behaviour of an automaton can also be computed from the answers of this problem over several instances $\ell$. As these problems have the same complexity, we focus exclusively on deciding the equality.

Notice that the lattice (especially its size) is not part of the input of the problem from Definition 2.5, but is given as a fixed parameter. This allows us to study the complexity of behaviour verification abstracting from the cost of the lattice operations performed. However, the weights appearing in the functions $\mathsf{in}$ and $\mathsf{wt}$ are considered in the size of the input LBTA $\mathcal{A}$, as they need to be explicitly expressed in the description of $\mathcal{A}$. Formally, let $\gamma$ be the largest element in the range of $\mathsf{in}$ and $\mathsf{wt}$; i.e.

$$\gamma := \max(\{|\mathsf{in}(q)| \mid q \in Q\} \cup \{|\mathsf{wt}(q, q_1, \ldots, q_k)| \mid (q, q_1, \ldots, q_k) \in Q^{k+1}\}).$$

We define the *size* $|\mathcal{A}|$ of the LBTA $\mathcal{A} = (Q, \mathsf{in}, \mathsf{wt}, F)$ for arity $k$ as

$$|\mathcal{A}| := |Q|^{k+1} \cdot \gamma.$$

As described before, we restrict our attention to lattices where infima and suprema can be computed in polynomial time by a deterministic Turing machine. Without this assumption, it is easy to show that the behaviour verification problem is highly undecidable. In fact, every decision problem $\mathcal{P}$ can be simulated by lattice operations, as described next.

Let $P_1, P_2, \ldots$ be all the positive instances and $N_1, N_2, \ldots$ all the negative instances of $\mathcal{P}$. We construct the lattice $L_\mathcal{P}$ whose Hasse diagram is depicted in

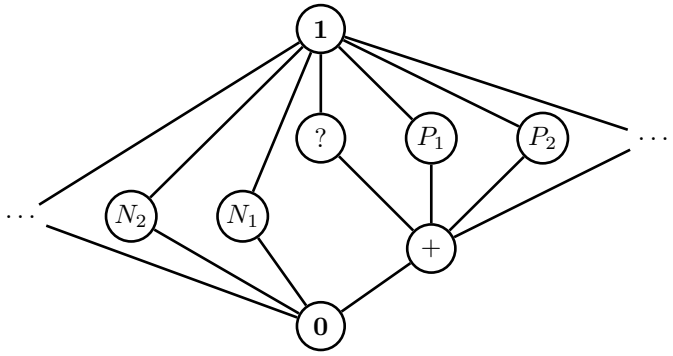


Figure 1: The lattice $L_{\mathcal{P}}$

Figure 1, where ? and + are two designated symbols. In particular, the meet of a positive instance with the designated element ? yields +, while the meet of ? a negative instance yields the lower bound $\mathbf{0}$. Thus, the problem $\mathcal{P}$ can be decided through operations on the lattice $L_{\mathcal{P}}$. Given an instance $\mathcal{I}$ of $\mathcal{P}$, we can build in linear time the LBTA $\mathcal{A}_{\mathcal{I}} = (\{q, q'\}, \mathsf{in}, \mathsf{wt}, \{q'\})$, where $\mathsf{in}(q) =?, \mathsf{in}(q') = \mathbf{0}$, and $\mathsf{wt}(q, q') = \mathcal{I}$, $\mathsf{wt}(q', q') = \mathbf{1}$, and $\mathsf{wt}(q, q) = \mathsf{wt}(q', q) = \mathbf{0}$. It is easy to see that $\|\mathcal{A}_{\mathcal{I}}\| = +$ if and only if $\mathcal{I}$ is a positive instance of $\mathcal{P}$. As this is true for any problem $\mathcal{P}$, in particular for highly undecidable problems (see e.g. [21]). Clearly, this high complexity arises not from the behaviour computation itself, but from the complexity of computing meets in the lattice $L_{\mathcal{P}}$. To abstract from this problem, we will restrict our attention to lattices whose operations are computable in polynomial time.

In the next section we show that the behaviour verification problem is in ExpTime, and show that this upper bound can be reduced to PSpace if the underlying lattice is linear-space-computable-encoded. Later on, we provide lower bounds for the complexity of this problem.

## 3. A Behaviour Verification Algorithm

In this section we describe a simple algorithm deciding the behaviour verification problem. This algorithm requires exponential time in general, but if the lattice is linear-space-computable-encoded, then it requires only polynomial space.

The exponential time algorithm follows conceptually the same ideas used in [19] to show that weighted automata over lattices recognize step functions. We describe it in detail, as the polynomial space algorithm will be a variant of this one. Let $\mathcal{A}$ be an LBTA over some lattice $(L, \vee, \wedge)$. We define the subset $L_{\mathcal{A}} \subseteq L$ of all weights appearing in $\mathcal{A}$ as follows:

$$L_{\mathcal{A}} := \{\mathsf{in}(q) \mid q \in Q\} \cup \{\mathsf{wt}(q, q_0, \ldots, q_k) \mid (q, q_0, \ldots, q_k) \in Q^{k+1}\}.$$

Notice that, since $Q$ is finite, the set $L_{\mathcal{A}}$ is also finite, even if $L$ is infinite. Every subset $M \subseteq L_{\mathcal{A}}$ defines the (unweighted) BTA $\mathcal{A}_M = (Q, \Delta_M, I_M, F)$ where:

- $(q, q_0, \ldots, q_k) \in \Delta_M$ iff $\mathsf{wt}(q, q_0, \ldots, q_k) \in M$, and
- $q \in I_M$ iff $\mathsf{in}(q) \in M$.

Intuitively, the automaton $\mathcal{A}_M$ accepts all successful runs of $\mathcal{A}$ that only use weights appearing in $M$. We will use this, together with the polynomial-time emptiness test of BTA to compute the behaviour of $\mathcal{A}$.

Let $\mathcal{M}(\mathcal{A}) := \{M \subseteq L_{\mathcal{A}} \mid \mathcal{L}(\mathcal{A}_M) \neq \emptyset\}$ be the set of all subsets of weights from $L_{\mathcal{A}}$ for which the automaton $\mathcal{A}_M$ has a successful run. Then we have the following equivalence.

**Theorem 3.1.** *For every LBTA $\mathcal{A}$ it holds that*

$$\|\mathcal{A}\| = \bigvee_{M \in \mathcal{M}(\mathcal{A})} \bigwedge_{\ell \in M} \ell.$$

*Proof.* For every successful run $r$ of $\mathcal{A}$ define

$$M_r := \{\mathsf{in}(r(\varepsilon))\} \cup \{\mathsf{wt}(r(u), r(u1), \ldots, r(uk)) \mid u \in [k]^*\};$$

that is, $M_r$ is the set of all weights appearing in the run $r$. Clearly, $M_r \subseteq L_{\mathcal{A}}$, and $\mathsf{in}(r(\varepsilon)) \wedge \mathsf{wt}(r) = \bigwedge_{\ell \in M_r} \ell$. Moreover, since $r$ is a successful run, it holds that $\mathcal{L}(\mathcal{A}_{M_r}) \neq \emptyset$. Then it follows that

$$\begin{aligned} \|\mathcal{A}\| = \bigvee_{r \in \mathsf{succ}_{\mathcal{A}}} \mathsf{in}(r(\varepsilon)) \wedge \mathsf{wt}(r) &= \bigvee_{r \in \mathsf{succ}_{\mathcal{A}}} \bigwedge_{\ell \in M_r} \ell \\ &\leq \bigvee_{M \in \mathcal{M}(\mathcal{A})} \bigwedge_{\ell \in M} \ell. \end{aligned}$$

For the opposite direction, for every $M \in \mathcal{M}(\mathcal{A})$ there exists a successful run $r_M$ of $\mathcal{A}_M$. By definition, if we consider $r_M$ as a run of $\mathcal{A}$, it follows that $\mathsf{in}(r_M(\varepsilon)) \in M$ and $\mathsf{wt}(r_M(u), r_M(u1), \ldots, r_M(uk)) \in M$ for every $u \in [k]^*$. It thus follows that $\bigwedge_{\ell \in M} \ell \leq \mathsf{in}(r_M(\varepsilon)) \wedge \mathsf{wt}(r_M)$. This implies that

$$\bigvee_{M \in \mathcal{M}(\mathcal{A})} \bigwedge_{\ell \in M} \ell \leq \bigvee_{r \in \mathsf{succ}_{\mathcal{A}}} \mathsf{in}(r(\varepsilon)) \wedge \mathsf{wt}(r) = \|\mathcal{A}\|.$$

□

To decide the behaviour verification problem for an LBTA $\mathcal{A}$ and an $\ell \in L$, we simply need to find which sets of weights belong to $\mathcal{M}(\mathcal{A})$, and compute the supremum of the infima of their values, as described in Algorithm 1. This yields the behaviour of $\mathcal{A}$ that is then compared to $\ell$. The algorithm executes the **for** loop an exponential number of times, measured on the size of $\mathcal{A}$; namely, once for every subset of $L_{\mathcal{A}}$, where the size of $L_{\mathcal{A}}$ is bounded by $|Q|^{k+1} + |Q|$. For each of these subsets, the algorithm performs an emptiness test that requires polynomial time [33, 37]. Thus, in total the algorithm requires exponential running time on $|\mathcal{A}|$.

**Algorithm 1** A behaviour computation algorithm.

**Input:** LBTA $\mathcal{A}$, $\ell \in L$
**Output:** $\|\mathcal{A}\| = \ell$

```
1: m := 0
2: for M ⊆ L_A do
3:    if L(A_M) ≠ ∅ then
4:       m := m ∨ ⋀_{ℓ∈M} ℓ
5:    end if
6: end for
7: if ℓ = m then
8:    return yes
9: else
10:   return no
11: end if
```

**Theorem 3.2.** *Behaviour verification is in* ExpTime.

In general, Algorithm 1 may also need exponential space; for instance, if the behaviour of $\mathcal{A}$ is exponential on both, $\mathcal{A}$ and the given input value $\ell$. Example 4.12 in Section 4.2 shows an exponential-space execution of this algorithm. However, although the algorithm may require exponential space, this does not imply that the behaviour verification problem is necessarily ExpTime-hard. Other algorithms may be able to compute the answer within a better complexity bound. In fact, if the lattice is distributive, then the behaviour can be computed in polynomial time [6], and hence behaviour verification is in P.

If the lattice from which the weights are taken is linear-space-computable-encoded, then Algorithm 1 can be executed using only polynomial space measured on $|\mathcal{A}|$: at every step of the execution, the algorithm needs to store in memory, along with the input automaton $\mathcal{A}$, the set $L_{\mathcal{A}}$, a subset $M \subseteq L_{\mathcal{A}}$, and the current computed value for $m$. All this information needs only polynomial space on the size of the input automaton, since all the elements of $L_{\mathcal{A}}$ are readily represented in $\mathcal{A}$, and the space required to encode $m$ is bounded by the largest representation of an element in $L_{\mathcal{A}}$, due to the linear-space-computable-encoded assumption. Thus, we have the following improved upper bound.

**Theorem 3.3.** *Behaviour computation of LBTA over linear-space-computable-encoded lattices is in* PSpace.

In the following section we provide some lower bounds for the complexity of this problem.

## 4. Lower Bounds

In this section we will show that the behaviour computation of LBTA is (i) NP-hard and co-NP-hard, if we consider only linear-space-computable-encoded

| $\oplus_1$ | $N$ | 0 | 1 | $X$ |
|---|---|---|---|---|
| $N$ | $N$ | 0 | 1 | $X$ |
| 0 | 0 | 0 | $X$ | $X$ |
| 1 | 1 | $X$ | 1 | $X$ |
| $X$ | $X$ | $X$ | $X$ | $X$ |

| $\otimes_1$ | $N$ | 0 | 1 | $X$ |
|---|---|---|---|---|
| $N$ | $N$ | $N$ | $N$ | $N$ |
| 0 | $N$ | 0 | $N$ | 0 |
| 1 | $N$ | $N$ | 1 | 1 |
| $X$ | $N$ | 0 | 1 | $X$ |

Table 1: $\oplus_1$ and $\otimes_1$

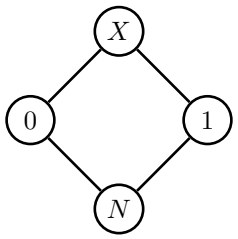


Figure 2: The lattice $(L_1, \oplus_1, \otimes_1)$

lattices, and (ii) PSPACE-hard in general. We show the former by providing reductions from satisfiability (SAT) and unsatisfiability (UNSAT) of propositional formulas, and the latter through a reduction from validity of quantified Boolean formulas (QBF). In the following, we assume that the reader is familiar with the basic notions of propositional logic and quantified boolean formulas, and in particular with the problems of satisfiability and validity of these formulas. For an introduction on these topics, we refer the reader to [8].

*4.1. Hardness for Linear-Space-Computable-Encoded Lattices*

We now show a reduction from satisfiability and unsatisfiability of propositional formulas to the behaviour of LBTA over a linear-space-computable-encoded infinite lattice $L_{\mathsf{sat}}$. We will define this lattice with the help of a family of finite lattices $L_n$ for $n \geq 1$, as follows.

Let $(L_1, \oplus_1, \otimes_1)$ be the lattice with $L_1 = \{1, 0, N, X\}$ and functions $\oplus_1, \otimes_1$ defined in Table 1. This lattice is depicted as a Hasse diagram in Figure 2. One can think of this lattice as the base for Belnap's four-valued relevance logic [1], in which 0 represents *false*, 1 means *true*, $X$ represents that a statement is both, true and false, and $N$ expresses that a statement is neither true, nor false.

For every $n \geq 1$, $(L_n, \oplus_n, \otimes_n)$ is the lattice obtained by restricting $(L_1)^n$ to the elements in $\{0, 1, X\}^n \cup \{N\}^n$.[3] We further extend this lattice to $L'_n$ by adding an element $Y_n$, i.e. $L'_n = L_n \cup \{Y_n\}$, with the operators

$$Y_n \oplus_n \ell = \begin{cases} Y_n & \text{if } \ell = N^n \\ X^n & \text{otherwise,} \end{cases} \qquad Y_n \otimes_n \ell = \begin{cases} Y_n & \text{if } \ell = X^n \\ N^n & \text{otherwise.} \end{cases}$$

[3] We represent the elements of $L_n$ as words of length $n$ over the alphabet $L_1$.

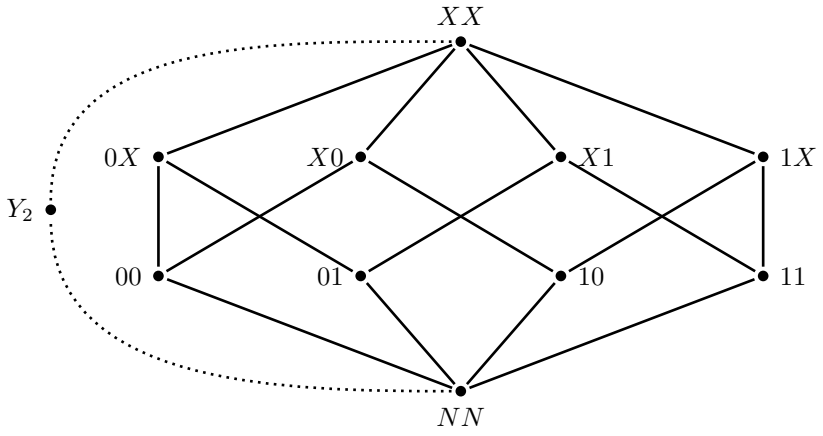


Figure 3: The lattice $L_2$ and its extension $L_2'$ with dotted lines

For every natural number $n$, $(L_n', \oplus_n, \otimes_n)$ defines a lattice. For instance, the lattices $L_2$ and its extension $L_2'$ are depicted in Figure 3, where dotted lines are used to differentiate the new relations added by the inclusion of $Y_2$.

The lattice $(L_{\mathsf{sat}}, \oplus_{\mathsf{sat}}, \otimes_{\mathsf{sat}})$ has $L_{\mathsf{sat}} := \bigcup_{n\geq 1} L_n \cup \{\top, \bot\}$ as its carrier set, and its operators are defined by:

$$\ell_1 \oplus_{\mathsf{sat}} \ell_2 = \begin{cases} \ell_1 \oplus_n \ell_2 & \text{if } \ell_1, \ell_2 \in L_n \\ \ell_2 & \text{if } \ell_1 = \bot \\ \ell_1 & \text{if } \ell_2 = \bot \\ \top & \text{otherwise,} \end{cases} \qquad \ell_1 \otimes_{\mathsf{sat}} \ell_2 = \begin{cases} \ell_1 \otimes_n \ell_2 & \text{if } \ell_1, \ell_2 \in L_n \\ \ell_2 & \text{if } \ell_1 = \top \\ \ell_1 & \text{if } \ell_2 = \top \\ \bot & \text{otherwise.} \end{cases}$$

It is easy to see that the operators $\oplus_{\mathsf{sat}}$ and $\otimes_{\mathsf{sat}}$ are idempotent, associative and commutative. To show that $(L_{\mathsf{sat}}, \oplus_{\mathsf{sat}}, \otimes_{\mathsf{sat}})$ is a lattice, we need only to verify that the absorption laws are satisfied. Let $\ell_1, \ell_2$ be two elements of $L_{\mathsf{sat}}$. If $\ell_1, \ell_2 \in L_n$ for some $n \geq 1$, then $\ell_1 \oplus_{\mathsf{sat}} (\ell_1 \otimes_{\mathsf{sat}} \ell_2) = \ell_1 = \ell_1 \otimes_{\mathsf{sat}} (\ell_1 \oplus_{\mathsf{sat}} \ell_2)$ follows from the fact that $L_n$ is a lattice, and that $\oplus_{\mathsf{sat}}$ and $\otimes_{\mathsf{sat}}$ behave as $\oplus_n$ and $\otimes_n$, respectively, over these elements. If $\ell_1 \in L_n$ and $\ell_2 \in L_m$ for $n \neq m$, then $\ell_1 \oplus_{\mathsf{sat}} (\ell_1 \otimes_{\mathsf{sat}} \ell_2) = \ell_1 \oplus_{\mathsf{sat}} \bot = \ell_1 = \ell_1 \otimes_{\mathsf{sat}} \top = \ell_1 \otimes_{\mathsf{sat}} (\ell_1 \oplus_{\mathsf{sat}} \ell_2)$. The remaining case, where $\{\ell_1, \ell_2\} \cap \{\top, \bot\} \neq \emptyset$, can be easily verified by a direct computation, using the definition; e.g., if $\ell_2 = \bot$, then $\ell_1 \oplus_{\mathsf{sat}} (\ell_1 \otimes_{\mathsf{sat}} \ell_2) = \ell_1 \oplus_{\mathsf{sat}} \ell_1 = \ell_1$.

The meets and joins of elements from $L_{\mathsf{sat}}$ can be computed in polynomial time: one needs only to test first whether the two elements have the same lenght (i.e., they belong to the same lattice $L_n$), which can be done in linear time. Afterwards, the operations $\otimes_n$ (respectively, $\oplus_n$), which need constant time each, need to be applied component-wise throughout the length of the words. This yields, in total, quadratically many steps for computing these operations.

The lattice $L_{\mathsf{sat}}$ is not distributive. To verify this, notice that on the one hand $X \oplus_{\mathsf{sat}} (XX \otimes_{\mathsf{sat}} XXX) = X \oplus_{\mathsf{sat}} \bot = X$, but on the other hand we have

that $(X \oplus_{\mathsf{sat}} XX) \otimes_{\mathsf{sat}} (X \oplus_{\mathsf{sat}} XXX) = \top \otimes_{\mathsf{sat}} \top = \top$. However, the lattice is linear-space-computable-encoded. To see this recall that the size of the elements of $L_{\mathsf{sat}}$ is given by:

$$|\ell| = \begin{cases} n & \text{if } \ell \in L_n \\ 1 & \text{otherwise.} \end{cases}$$

It is a direct consequence of the definition of $\otimes_{\mathsf{sat}}$ and $\oplus_{\mathsf{sat}}$ that

$$|\ell_1 \oplus_{\mathsf{sat}} \ell_2| \leq |\ell_1| \leq \max\{|\ell_1|, |\ell_2|\}, \text{ and}$$
$$|\ell_1 \otimes_{\mathsf{sat}} \ell_2| \leq |\ell_1| \leq \max\{|\ell_1|, |\ell_2|\}.$$

To recall, $L_{\mathsf{sat}}$ is an infinite, linear-space-computable-encoded lattice that is not distributive. We will now show how to reduce satisfiability and unsatisfiability of propositional formulas to verifying the behaviour of automata with weights taken from this lattice.

Given a propositional formula $\phi$, we will construct an LBTA $\mathcal{A}_\phi$ over the lattice $L_{\mathsf{sat}}$ for arity 1 such that its behaviour $\|\mathcal{A}_\phi\|$ expresses whether $\phi$ is satisfiable or unsatisfiable.

Let $\phi$ be a propositional formula. We may assume that it is in *conjunctive normal form (CNF)*; that is, $\phi$ is of the form

$$(\chi_{11} \vee \ldots \vee \chi_{1m_1}) \wedge (\chi_{21} \vee \ldots \vee \chi_{2m_2}) \wedge \ldots \wedge (\chi_{k1} \vee \ldots \vee \chi_{km_k}),$$

where each $\chi_{ij}, 1 \leq i \leq k, 1 \leq j \leq m_i$ is a literal.[4] Every propositional formula can be transformed in linear time into a satisfiable-equivalent formula in CNF [36]. As usual in the literature, we call a disjunction of literals a *clause*; hence, a formula is in CNF iff it is a conjunction of clauses.

We produce one state for every occurrence of a literal in the formula $\phi$ plus two auxiliary states $q_0$ and $q_e$; formally, the set of states is

$$Q_\phi := \{q_{ij} \mid 1 \leq i \leq k, 1 \leq j \leq m_i\} \cup \{q_0, q_e\}.$$

Intuitively, the states $q_{ij}$ will be used to identify which literal $\chi_{ij}$ is used to satisfy the $i$-th clause. The automaton will try to satisfy each clause sequentially, and the state $q_e$ expresses that all clauses have been already verified.

Let now $n$ be the number of propositional variables appearing in the formula $\phi$. The automaton $\mathcal{A}_\phi$ will use only weights taken from the sub-lattice $L'_n$ of $L_{\mathsf{sat}}$, as described next.

**Definition 4.1** ($\mathcal{A}_\phi$)**.** Given the propositional formula

$$\phi = (\chi_{11} \vee \ldots \vee \chi_{1m_1}) \wedge \ldots \wedge (\chi_{k1} \vee \ldots \vee \chi_{km_k})$$

in CNF, using the variables $x_1, \ldots, x_n$, the automaton $\mathcal{A}_\phi = (Q_\phi, \mathsf{in}_\phi, \mathsf{wt}_\phi, Q_\phi)$ over the lattice $L_{\mathsf{sat}}$ is defined by:

[4] Recall that a literal is a propositional variable, or a negated propositional variable.

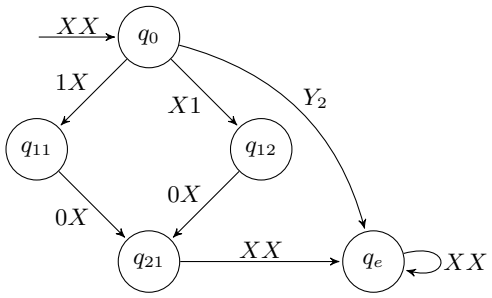


Figure 4: The automaton $\mathcal{A}_\phi$ for $\phi = (x_1 \vee x_2) \wedge \neg x_1$. Not-drawn edges that have weight $NN$.

- $Q_\phi := \{q_{ij} \mid 1 \leq i \leq k, 1 \leq j \leq m_i\} \cup \{q_0, q_e\}$,
- $\mathsf{in}_\phi(q) = \begin{cases} X^n & \text{if } q = q_0, \\ N^n & \text{otherwise,} \end{cases}$
- $\mathsf{wt}_\phi(q_0, q_{1j}) = \begin{cases} X^{\kappa-1}1X^{n-\kappa} & \text{if } \chi_{1j} = x_\kappa \\ X^{\kappa-1}0X^{n-\kappa} & \text{if } \chi_{1j} = \neg x_\kappa \end{cases}$
- $\mathsf{wt}_\phi(q_{ij'}, q_{(i+1)j}) = \begin{cases} X^{\kappa-1}1X^{n-\kappa} & \text{if } \chi_{(i+1)j} = x_\kappa \\ X^{\kappa-1}0X^{n-\kappa} & \text{if } \chi_{(i+1)j} = \neg x_\kappa \end{cases}$
- $\mathsf{wt}_\phi(q, q_e) = \begin{cases} X^n & \text{if } q = q_{kj} \text{ for some } j, 1 \leq j \leq m_k \\ X^n & \text{if } q = q_e \\ Y_n & \text{if } q = q_0, \end{cases}$
- all other transitions have weight $N^n$.

Intuitively, the weights ensure that a literal is chosen for satisfying every clause in the formula $\phi$, and that these choices form a valid valuation of the propositional variables; i.e., that there is no propositional variable $\chi$ such that both $\chi$ and $\neg\chi$ are chosen.

**Example 4.2.** Consider the formula $\phi := (x_1 \vee x_2) \wedge \neg x_1$. The automaton $\mathcal{A}_\phi$ obtained from this formula through the previous construction is depicted in Figure 4. Notice that the states $q_{12}$ and $q_{21}$ both correspond to the variable $x_1$, but represent the appearances of this variable in different clauses of $\phi$. Let $r$ be the run given by the word $q_0 q_{12} q_{21} q_e q_e^\omega$. This run expresses the choice in which the first clause is satisfied by its second literal, in this case $x_2$, and the second clause is satisfied by its first literal, here $\neg x_1$. The weight of this run is

$$\begin{aligned} \mathsf{wt}(r) &= \mathsf{wt}(q_0, q_{12}) \otimes_{\mathsf{sat}} \mathsf{wt}(q_{12}, q_{21}) \otimes_{\mathsf{sat}} \mathsf{wt}(q_{21}, q_e) \otimes_{\mathsf{sat}} \mathsf{wt}(q_e, q_e) \\ &= X1 \otimes_{\mathsf{sat}} 0X \otimes_{\mathsf{sat}} XX \otimes_{\mathsf{sat}} XX = 01. \end{aligned}$$

This weight can be interpreted as the satisfying valuation setting the variable $x_1$ to "false" and $x_2$ to "true".

Let now $r'$ be the run $q_0 q_{11} q_{21} q_e q_e^\omega$. Intuitively, this run chooses to satisfy the first clause through $x_1$ and the second clause through $\neg x_1$. We now have

$$\begin{aligned}\mathsf{wt}(r') &= \mathsf{wt}(q_0, q_{11}) \otimes_{\mathsf{sat}} \mathsf{wt}(q_{11}, q_{21}) \otimes_{\mathsf{sat}} \mathsf{wt}(q_{21}, q_e) \otimes_{\mathsf{sat}} \mathsf{wt}(q_e, q_e) \\ &= 1X \otimes_{\mathsf{sat}} 0X \otimes_{\mathsf{sat}} XX \otimes_{\mathsf{sat}} XX = NN.\end{aligned}$$

The value $NN$ expresses that the choices described by the run $r'$ do not correspond to a valid valuation of the variables appearing in $\phi$.

Since all the states are accepting states, every run on this automaton is successful. Moreover, the run $r_e = q_0 q_e^\omega$ has weight $Y_n$ since $\mathsf{wt}(q_0, q_e) = Y_n$, $\mathsf{wt}(q_e, q_e) = X^n$ and $Y_n \otimes_n X^n = Y_n$. Thus,

$$\|\mathcal{A}_\phi\| \geq \mathsf{in}(r_e(\varepsilon)) \otimes_{\mathsf{sat}} \mathsf{wt}(r_e) = X^n \otimes_n Y_n = Y_n.$$

This implies that $\|\mathcal{A}_\phi\| \in \{Y_n, X^n\}$. We will show that $\|\mathcal{A}_\phi\| = Y_n$ iff $\phi$ is unsatisfiable, and thus $\|\mathcal{A}_\phi\| = X^n$ iff $\phi$ is satisfiable.

**Lemma 4.3.** *If $\phi$ is satisfiable, then $\|\mathcal{A}_\phi\| = X^n$.*

*Proof.* If $\phi$ is satisfiable, then there exists a valuation $V$ that satisfies all its clauses; that is, in every clause $i$ there is a literal $\chi_{ij_i}$ that $V$ makes true. Let $r$ be the run defined by $r = q_0 q_{1j_1} q_{2j_2} \dots q_{kj_k} q_e q_e^\omega$; then $\mathsf{in}_\phi(r(\varepsilon)) = X^n$. Additionally, since no transition in this run has weight $Y_n$, it follows that $\mathsf{wt}_\phi(r) \neq Y_n$. We show that $\mathsf{wt}_\phi(r) > N^n$.

Suppose that $\mathsf{wt}_\phi(r) = N^n$, then since all transitions from $r$ have weight greater than $N^n$, this implies that there are $i, j \in \mathbb{N}$ and $\kappa, 1 \leq \kappa \leq n$ such that $\mathsf{wt}_\phi(r(1^i), r(1^{i+1})) = X^{\kappa-1}0X^{n-\kappa}$ and $\mathsf{wt}_\phi(r(1^j), r(1^{j+1})) = X^{\kappa-1}1X^{n-\kappa}$. This means that $V$ satisfies $\neg x_\kappa$ in clause $i$ and $x_\kappa$ in clause $j$, contradicting the assumption that $V$ is a valuation. Thus, $\mathsf{wt}_\phi(r) > N^n$, and $\|\mathcal{A}_\phi\| = X^n$. □

**Lemma 4.4.** *If $\|\mathcal{A}_\phi\| = X^n$, then $\phi$ is satisfiable.*

*Proof.* If $\|\mathcal{A}_\phi\| = X^n$, then there exists a run $r$ with $\mathsf{wt}_\phi(r) \notin \{N^n, Y_n\}$; that is, $\mathsf{wt}_\phi(r) \in \{0, 1, X\}^n$. This run must be of the form $q_0 q_{1j_1} q_{2j_2} \dots q_{kj_k} q_e q_e^\omega$, since otherwise, it would have a transition with weight in $\{N^n, Y_n\}$, contradicting the previous statement. Define the valuation $V$ that maps the variable $x_i$ to "true" if the $i$-th component of $\mathsf{wt}_\phi(r)$ is 1 and to "false" otherwise; i.e. if the $i$-th component of $\mathsf{wt}_\phi(r)$ is either 0 or $X$. We show that this valuation satisfies the formula $\phi$.

Suppose $V$ does not satisfy $\phi$; then there must be an $i, 1 \leq i \leq k$ such that $V$ violates $\chi_{ij}$ for all $j, 1 \leq j \leq m_i$; i.e, $V$ violates the $i$-th clause from $\phi$. In particular, it must violate $\chi_{ij_0}$ where $q_{ij_0} = r(1^i)$. If $\chi_{ij_0} = x_\kappa$, then $V$ evaluates $x_\kappa$ to "false", but $\mathsf{wt}_\phi(r) \leq \mathsf{wt}_\phi(r(1^{i-1}), r(1^i)) = X^{\kappa-1}1X^{n-\kappa}$, contradicting the construction of $V$. An analogous argument can be used if $\chi_{ij_0} = \neg x_\kappa$. Hence $V$ must satisfy $\phi$. □

The automaton $\mathcal{A}_\phi$ has linearly many states measured on the size of the input formula $\phi$,[5] and only uses weights from $L'_n$; thus, it can be constructed in polynomial time. This, together with the previous lemmas, yields our hardness result.

**Theorem 4.5.** *Behaviour verification over linear-space-computable-encoded lattices is* NP*-hard and co-*NP*-hard.*

*Proof.* Let $\phi$ be a propositional formula in CNF. From Lemmas 4.3 and 4.4 it follows that $\phi$ is satisfiable iff behaviour verification answers "yes" on input $\mathcal{A}_\phi, X^n$, and $\phi$ is unsatisfiable iff behaviour verification answers "yes" on input $\mathcal{A}_\phi, Y_n$. Since (un)satisfiability of propositional formulas is (co-)NP-hard [13], the former shows NP-hardness, and the latter implies co-NP-hardness of the behaviour verification problem. □

### *4.2. Hardness for General Lattices*

If we allow lattices that are not linear-space-computable-encoded, then the lower bound presented in the previous section can be increased to PSpace. We show this through a reduction from the problem of validity of quantified Boolean formulas (QBF). Recall that satisfiability and unsatisfiability of propositional formulas can be seen as special cases of QBF: a propositional formula $\phi$ is satisfiable if and only if $\exists x_1 \cdots x_n.\phi$ is valid, and analogously for unsatisfiability using only universal quantifiers; i.e., $\phi$ is unsatisfiable if $\forall x_1 \cdots x_n.\neg\phi$ is valid. Based on this fact, the reduction presented in this section will follow the same basic idea used before for proving the NP and co-NP lower bounds. However, it will require additional technical details for dealing with the nesting and changes of quantifiers. Intuitively, the weights in a run will ensure that all relevant valuations are considered.

The weights of our automaton will be sets of tuples from $L_n$. To encode a set as strings, one could introduce a new symbol that is not in the current alphabet and write it as a list where the elements are separated by the new symbol. For reasons of readability, we are going to use the set representation in our notations. Each weight intuitively expresses a set of valuations that satisfy the propositional formula $\phi$, obtained from ignoring the quantifiers in the input QBF. The order in which variables are quantified will correspond to the order in which they are represented in the tuple. Finally, since existentially and universally quantified variables are treated differently, we specify also a partition of the variables, as described next.

**Definition 4.6** ($L_\mathcal{S}$)**.** Let $\mathcal{S} = (n, \mathcal{I}_\forall)$, where $n \geq 1$ and $\mathcal{I}_\forall \subseteq \{1, \ldots, n\}$. Given a set $S \subseteq L_n$, a word $w \in S$ is $\mathcal{I}_\forall$*-incompatible* with respect to $S$ if any of the following conditions hold:

[5] As usual, we measure the size of a formula by the number of propositional variables it contains.

1. $w$ is of the form $u0v$ for some $u \in L_{i-1}, i \in \mathcal{I}_\forall$ and there is no $u'1v'$ or $u'Xv'$ in $S$ such that $u' \in L_{i-1}$, and $u \otimes_{i-1} u' \neq N^{i-1}$, or
2. $w$ is of the form $u1v$ for some $u \in L_{i-1}, i \in \mathcal{I}_\forall$ and there is no $u'0v'$ or $u'Xv'$ in $S$ such that $u' \in L_{i-1}$, and $u \otimes_{i-1} u' \neq N^{i-1}$.[6]

We denote as $\mathsf{inc}_{\mathcal{I}_\forall}(S)$ the set of all $\mathcal{I}_\forall$-incompatible elements in $S$.

The lattice $(L_\mathcal{S}, \oplus_\mathcal{S}, \otimes_\mathcal{S})$ uses as carrier set $L_\mathcal{S}$ the set of all antichains $S \subseteq \wp(L_n)$ such that $\mathsf{inc}_{\mathcal{I}_\forall}(S) = \emptyset$, where $\wp(L)$ denotes the power set of $L$. Given $S_1, S_2 \in L_\mathcal{S}$, let $\leq$ be the ordering from $L_n$. We extend the operators $\oplus_n$ and $\otimes_n$ to sets of words as follows:

$$S_1 \oplus_n S_2 := \{w \in S_1 \cup S_2 \mid \text{ for every } u \in S_1 \cup S_2,\ w \not\leq u\},$$
$$S_1 \otimes_n S_2 := \{w \in L_n \mid \text{ there exist } u \in S_1, v \in S_2 \text{ such that } w = u \otimes_n v \text{ and for every } u' \in S_1, v' \in S_2, w \not\leq u' \otimes_n v'\}.$$

The operators $\oplus_\mathcal{S}$ and $\otimes_\mathcal{S}$ are defined as follows:

$$S_1 \oplus_\mathcal{S} S_2 := S_1 \oplus_n S_2$$
$$S_1 \otimes_\mathcal{S} S_2 := (S_1 \otimes_n S_2) \setminus \mathsf{inc}_{\mathcal{I}_\forall}(S_1 \otimes_n S_2).$$

The idea is that the set $\mathcal{I}_\forall$ expresses the indices of the variables that are universally quantified. For each of these variables $x$, given a choice of valuation of the variables in which it depends, we need to ensure that both choices for evaluating $x$ still satisfy the formula $\phi$. The set $S$ stores the set of valuations that do not violate the formula. The incompatibility condition then detects which of these valuations violate the universal quantification. For example, if $\mathcal{S} = (1, \{1\})$, it means that the formula has only one variable that is universally quantified. Then $L_\mathcal{S} = \{\{X\}, \{0, 1\}, \{N\}\}$. Notice that $\{1\}$ is also an antichain of $L_1$, but $1 \in \mathsf{inc}_{\mathcal{I}_\forall}(\{1\})$, since we can see 1 as the word $\varepsilon 1 \varepsilon$ with $\varepsilon \in L_0$, and there is no word $\varepsilon 0 v$ or $\varepsilon X v$ in $\{1\}$. Thus, 1 satisfies the Condition 2 from Definiton 4.6. This means that $\{1\}$ does not belong to $L_\mathcal{S}$. The following example gives the intuition of these notions in further detail.

**Example 4.7.** The tuple $(2, \{1\})$ is used to express that the formula uses two variables, with the first one being universally quantified; that is, we are dealing with a quantified Boolean formula of the form $\forall x_1.\exists x_2.\phi(x_1, x_2)$. This formula is valid if, for every possible valuation of $x_1$, we can find a valuation of $x_2$ that makes $\phi(x_1, x_2)$ true. One possibility is to use the valuations 00 and 11, given a chosen valuation of $x_1$, $x_2$ is evaluated to the same truth value. Indeed, $\{00, 11\}$ is an antichain of $L_2$, and none of its elements is $\{1\}$-incompatible, which means that $\{00, 11\} \in L_{(2,\{1\})}$. Intuitively, this means that this set represents a valid choice of valuations for verifying the validity of the input formula.

Consider now the tuple $(2, \{2\})$, which will be used to deal with formulas of the form $\exists x_1.\forall x_2.\phi(x_1, x_2)$. This means that we have to choose one value for

[6] For the case where $i = 1$, we set $L_0 = \{\varepsilon\}$, and $N^0 = \varepsilon$.

$x_1$ in such a way that both valuations of $x_2$ make $\phi$ true. Clearly, the previous choice of valuations 00 and 11 is not adequate, since it depends on choosing a different value for $x_1$. In fact, $\{00, 11\}$ does not belong to $L_{(2,\{2\})}$ since both elements in this set are $\{2\}$-incompatible with respect to $\{00, 11\}$.

We will now show that the structure $(L_{\mathcal{S}}, \oplus_{\mathcal{S}}, \otimes_{\mathcal{S}})$ is indeed a lattice, as claimed before, and then describe how we can use it to decide validity of quantified Boolean formulas.

**Lemma 4.8.** *Let $S = (n, \mathcal{I}_\forall)$. Then $(L_{\mathcal{S}}, \oplus_{\mathcal{S}}, \otimes_{\mathcal{S}})$ is a lattice.*

*Proof.* It is easy to see that the operators $\oplus_{\mathcal{S}}$ and $\otimes_{\mathcal{S}}$ are idempotent, associative and commutative. We now prove that they satisfy the absorption laws. Let $S_1, S_2 \in L_{\mathcal{S}}$. Consider some $w \in S_1$. We show that $w \in S_1 \oplus_{\mathcal{S}} (S_1 \otimes_{\mathcal{S}} S_2)$. By definition, it suffices to show that there is no $u \in S_1 \otimes_{\mathcal{S}} S_2$ such that $w < u$. For every $u \in S_1 \otimes_{\mathcal{S}} S_2$ there exist $u_1 \in S_1, u_2 \in S_2$ such that $u = u_1 \otimes_n u_2$. In particular, $u \leq u_1$. Thus, if there exists some $u$ with $w < u$ then $w < u_1$, contradicting the fact that $S_1$ is an antichain from $L_n$. We thus have that $S_1 \subseteq S_1 \oplus_{\mathcal{S}} (S_1 \otimes_{\mathcal{S}} S_2)$.

Conversely, let $w \in S_1 \oplus_{\mathcal{S}} (S_1 \otimes_{\mathcal{S}} S_2)$, and suppose that $w \notin S_1$. It must then hold that (i) $w \in S_1 \otimes_{\mathcal{S}} S_2 \subseteq S_1 \otimes_n S_2$ and (ii) there is no $u \in S_1$ such that $w \leq u$. From (i) it follows that there exist $u_1 \in S_1, u_2 \in S_2$ with $w = u_1 \otimes_n u_2$. In particular $w \leq u_1$, which violates (ii). Hence $S_1 \oplus_{\mathcal{S}} (S_1 \otimes_{\mathcal{S}} S_2) \subseteq S_1$. The other absorption law can be proven in an analogous way. □

As done for the case of satisfiability of propositional formulas, we construct an infinite lattice that contains as sub-lattices each of the lattices $L_{\mathcal{S}}$. We assume that each of these sub-lattices are disjoint; that is, we make disjoint copies of them for different pairs $\mathcal{S}, \mathcal{S}'$. The lattice $(L_{\mathsf{qbf}}, \oplus_{\mathsf{qbf}}, \otimes_{\mathsf{qbf}})$ has

$$L_{\mathsf{qbf}} := \biguplus_{n \geq 1, \mathcal{I} \subseteq \{1,\ldots,n\}} L_{(n,\mathcal{I})} \cup \{\top, \bot\}$$

as carrier set, and its operators are defined by:

$$\ell_1 \oplus_{\mathsf{qbf}} \ell_2 = \begin{cases} \ell_1 \oplus_{\mathcal{S}} \ell_2 & \text{if } \ell_1, \ell_2 \in L_{\mathcal{S}} \\ \ell_2 & \text{if } \ell_1 = \bot \\ \ell_1 & \text{if } \ell_2 = \bot \\ \top & \text{otherwise,} \end{cases} \qquad \ell_1 \otimes_{\mathsf{qbf}} \ell_2 = \begin{cases} \ell_1 \otimes_{\mathcal{S}} \ell_2 & \text{if } \ell_1, \ell_2 \in L_{\mathcal{S}} \\ \ell_2 & \text{if } \ell_1 = \top \\ \ell_1 & \text{if } \ell_2 = \top \\ \bot & \text{otherwise.} \end{cases}$$

The proof that these operators define a lattice is analogous to the one presented before for the lattice $L_{\mathsf{sat}}$. It is also easy to verify that this lattice is not distributive. Recall that every element of $L_{\mathsf{qbf}} \setminus \{\top, \bot\}$ is a set of words of length $n$, for some $n$. We use the obvious encoding of these elements, explicitly stating the words that appear in the set. The size of such a set is the number of elements it has, times their length. Formally, we obtain that the size of an element $\ell$ of

$L_{\mathsf{qbf}}$ is given by:

$$|\ell| = \begin{cases} n \cdot m & \text{if } \ell \subseteq L_n, \text{ and } \ell \text{ has } m \text{ elements} \\ 1 & \text{otherwise.} \end{cases} \tag{1}$$

The join and meet of two sets of words $\ell, \ell' \in L_{\mathsf{qbf}}$ can be computed in polynomial time, as it corresponds to the pairwise application of the operator $\otimes_n$ (respectively, $\oplus_n$), which produces $m \cdot m'$ words, where $m, m'$ are the number of words in $\ell$ and $\ell'$, respectively. Afterwards, it is only necessary to compare all these words to remove all redundant and incompatible elements, which can also be done in polynomial time.

Under this encoding, this lattice is *not* linear-space-computable-encoded, as shown by the following example.

**Example 4.9.** Let $\mathcal{S} = (n, \emptyset)$ and consider the antichains $\{X^n\}$ and for every $i, 1 \leq i \leq n$ $\{X^{i-1}1X^{n-i}, X^{i-1}0X^{n-i}\}$. These are all elements of $L_{\mathcal{S}}$, which contain at most two words of length exactly $n$ each; i.e., every one of these elements has size $2n$. However,

$$\{X^n\} \otimes_{\mathcal{S}} \bigotimes_{1 \leq i \leq n} \{X^{i-1}1X^{n-i}, X^{i-1}0X^{n-i}\} = \{0,1\}^n,$$

is an antichain of $L_{\mathcal{S}}$ with $2^n$ elements. In other words, the meet of these $n$ elements has size larger than each of the individual elements, violating the definition of linear-space-computable-encoded.

This example does not imply that there is no encoding under which $L_{\mathsf{qbf}}$ is linear-space-computable-encoded. Indeed, we now describe an encoding that make this lattice linear-space-computable-encoded.[7] For every $n \in \mathbb{N}$, there are at most $4^n$ words of length $n$ using the symbols $X, 0, 1, N$. This implies that each lattice $L_{(n,\mathcal{I})}$ has at most $2^{4^n}$ elements. We can represent each of these elements using a binary string of length $4^n$ that describes which elements belong to the set and which do not. Under this encoding, every element of $L_{(n,\mathcal{I})}$ has size $4^n$, yielding a linear-space-computable-encoded lattice. However, as discussed at the end of this section, such an encoding would yield an exponential reduction, which would not be useful for obtaining the desired complexity lower bounds.

Let now $\nabla_1 x_1 \ldots \nabla_n x_n.\phi$ with $\nabla_i \in \{\exists, \forall\}$ be a quantified Boolean formula. As done for satisfiability of propositional formulas, we can assume that $\phi$ is in CNF.

Given the QBF $\psi := \nabla_1 x_1 \ldots \nabla_n x_n.\phi$, let $\mathcal{I}_\forall = \{i \mid \nabla_i = \forall\}$ be the set of indices of all universally quantified variables in $\psi$. We will construct an automaton $\mathcal{A}_\psi$ with weights belonging to the lattice $L_{(n,\mathcal{I}_\forall)}$, whose behaviour characterises validity of $\psi$.

We introduce one state $q_i$ for each variable $x_i$ appearing in the formula $\psi$ and one state $q_{n+j}$ for each clause $(\chi_{j1} \vee \ldots \vee \chi_{jm_j})$, along with the distinguished

[7] We are grateful to an anonymous reviewer for providing the idea of this construction.

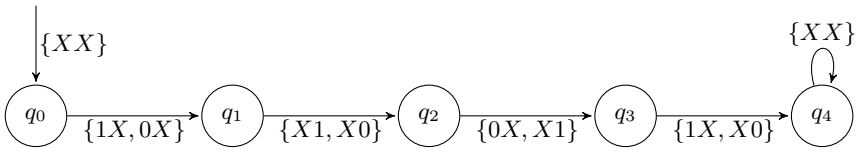


Figure 5: The automaton $\mathcal{A}_\psi$ for $\psi = \forall x_1 \exists x_2.(\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$.

initial state $q_0$. Notice that, contrary to the construction used in Section 4.1, here we do not consider one state for each *occurrence* of a variable, but rather one for each variable appearing in $\psi$.

**Definition 4.10** ($\mathcal{A}_\psi$)**.** Given a QBF $\psi = \nabla_1 x_1 \dots \nabla_n x_n.\phi$ with

$$\phi = (\chi_{11} \vee \dots \vee \chi_{1m_1}) \wedge \dots \wedge (\chi_{k1} \vee \dots \vee \chi_{km_k}),$$

let $Q_\psi := \{q_i \mid 0 \leq i \leq n+k\}$ and for every $i, 1 \leq i \leq k$ let

$$\begin{aligned} C_i :=& \{X^{\kappa-1}1X^{n-\kappa} \mid \chi_{ij} = x_\kappa, 1 \leq j \leq m_i\} \cup \\ & \{X^{\kappa-1}0X^{n-\kappa} \mid \chi_{ij} = \neg x_\kappa, 1 \leq j \leq m_i\}. \end{aligned}$$

The automaton $\mathcal{A}_\psi = (Q_\psi, \mathsf{in}_\psi, \mathsf{wt}_\psi, Q_\psi)$ is defined by:

$$\mathsf{in}_\psi(q) = \begin{cases} \{X^n\} & \text{if } q = q_0 \\ \{N^n\} & \text{otherwise.} \end{cases}$$

$$\mathsf{wt}_\psi(q_i, q_j) = \begin{cases} \{X^{j-1}1X^{n-j}, X^{j-1}0X^{n-j}\} & \text{if } j = i+1, 1 \leq j \leq n \\ C_{j-n} & \text{if } j = i+1, n < j \leq n+k \\ \{X^n\} & \text{if } i = j = n+k \\ \{N^n\} & \text{otherwise.} \end{cases}$$

All weights are elements of the sub-lattice $L_{(n,\mathcal{I}_\forall)}$ with $\mathcal{I}_\forall := \{i \mid \nabla_i = \forall\}$.

**Example 4.11.** Consider the QBF $\psi := \forall x_1 \exists x_2.(\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$, which is valid. Figure 5 depicts the automaton $\mathcal{A}_\psi$. All the transitions belong to $L_\mathcal{S}$ with $\mathcal{S} = (2, \{1\})$. The transitions that are not depicted have weight $\{NN\}$. We compute now the weight of the run $r = q_0 q_1 q_2 q_3 q_4 q_4^\omega$:

$$\begin{aligned} \mathsf{wt}_\psi(r) &= \{1X, 0X\} \otimes_\mathcal{S} \{X1, X0\} \otimes_\mathcal{S} \{0X, X1\} \otimes_\mathcal{S} \{1X, X0\} \\ &= \{00, 01, 10, 11\} \otimes_\mathcal{S} \{0X, X1\} \otimes_\mathcal{S} \{1X, X0\} \\ &= \{00, 01, 11\} \otimes_\mathcal{S} \{1X, X0\} = \{00, 11\}. \end{aligned}$$

Intuitively, the words in $\mathsf{wt}_\psi(r)$ describe the valuations that make $\psi$ valid: if $x_1$ is evaluated to "false", then $x_2$ must also be evaluated to "false", and if $x_1$ evaluates to "true" so must also $x_2$. The condition of having only compatible elements in these sets ensures that both choices of universally quantified variables

are verified, for every choice of their preceding existential variables. Indeed, if a word $u0v$ appears in this set, where 0 corresponds to a valuation of a universally quantified variable, the compatibility condition ensures that there is a $v'$ such that $u1v'$ belongs also to this set, and analogously for words of the form $u1v$.

Consider now the formula $\psi' := \exists x_1 \forall x_2.(\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$, which is not valid. The automaton $\mathcal{A}_{\psi'}$ is the same depicted in Figure 5, but with the weights belonging to $L_{\mathcal{S}'}$ with $\mathcal{S}' = (2, \{2\})$. Notice that

$$\{00, 01, 10, 11\} \otimes_2 \{0X, X1\} = \{00, 01, 11\}$$

but 11 is $\{2\}$-incompatible. Thus $\{00, 01, 10, 11\} \otimes_{\mathcal{S}'} \{0X, X1\} = \{00, 01\}$ and

$$\begin{aligned} \mathsf{wt}_{\psi'}(r) &= \{1X, 0X\} \otimes_{\mathcal{S}'} \{X1, X0\} \otimes_{\mathcal{S}'} \{0X, X1\} \otimes_{\mathcal{S}'} \{1X, X0\} \\ &= \{00, 01, 10, 11\} \otimes_{\mathcal{S}'} \{0X, X1\} \otimes_{\mathcal{S}'} \{1X, X0\} \\ &= \{00, 01\} \otimes_{\mathcal{S}'} \{1X, X0\} = \emptyset. \end{aligned}$$

Once we choose a valuation for the variable $x_1$, there is only one valuation of $x_2$ that makes the formula $(\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$ true. These valuations are removed by the incompatibility condition, since $x_2$ is universally quantified. Intuitively, having $\mathsf{wt}_{\psi'}(r) = \emptyset$ means that the formula $\phi'$ is not valid.

Before showing that these automata can be used to decide validity of QBF, we show that Algorithm 1 may require exponential space to verify their behaviour, as claimed in Section 3.

**Example 4.12.** Given $n \in \mathbb{N}$, let $\psi_n := \exists x_1 \exists x_2 \ldots \exists x_n.(x_1 \vee \neg x_1)$. Notice that $\mathcal{A}_{\psi_n}$ has $n+1$ states, and all its weights are sets of at most two words of length $n$, thus the size of $\mathcal{A}_{\psi_n}$ is polynomial on $n$. Moreover,

$$L_{\mathcal{A}_{\psi_n}} = \{\{X^{i-1}1X^{n-i}, X^{i-1}0X^{n-i}\} \mid 1 \leq i \leq n\}.$$

Suppose now that we call Algorithm 1 with input $\mathcal{A}_{\psi_n}$ and $\{X^n\}$, which is also polynomial on $n$. When the **for** loop terminates, the algorithm stores the behaviour of $\mathcal{A}_{\psi_n}$ in the variable $m$. It is a simple computation to verify that this behaviour is

$$m := \bigotimes_{\ell \in L_{\mathcal{A}_{\psi_n}}} \ell = \{0, 1\}^n;$$

that is, $m$ stores $2^n$ words of length $n$. This set needs space exponential on $n$ to be stored.

All runs of the automaton $\mathcal{A}_\psi$ are accepting. Additionally, all the weights used are greater or equal to $\{N^n\}$. Thus, the behaviour of this automaton is always greater or equal to $\{N^n\}$. We will show that $\|\mathcal{A}_\psi\| = \{N^n\}$ iff $\psi$ is not valid.

To do this, notice that we can understand every $\mathcal{S} \subseteq \{0, 1\}^n$ as a set of valuations of the variables $x_1, \ldots, x_n$. As described in Example 4.9, the first transitions of the automaton $\mathcal{A}_\psi$ simply produce the set of all valuations $\{0, 1\}^n$. The subsequent transitions remove those valuations that do not satisfy each of the clauses, as described by the following lemma.

**Lemma 4.13.** *Given a set $\mathcal{V} \subseteq \{0,1\}^n$ and a clause $c_i := (\chi_{i1} \vee \ldots \vee \chi_{im_i})$, the set $(\mathcal{V} \otimes_n C_i) \setminus \{N^n\}$ contains exactly those valuations from $\mathcal{V}$ that satisfy $c_i$.*

*Proof.* Notice first that by definition of $\otimes_n$, for every $w \in \{0,1\}^n$ and every $w' \in \{0,1,X\}^n$, $w \otimes_n w' \in \{w, N^n\}$. Thus $(\mathcal{V} \otimes_n C_i) \setminus \{N^n\} \subseteq \mathcal{S}$. Let $w \in \mathcal{V}$. If $w \in (\mathcal{V} \otimes_n C_i) \setminus \{N^n\}$, then there exists a $w' \in C_i$ such that $w \otimes_n w' = w$. If $w'$ is of the form $X^{\kappa-1}1X^{n-\kappa}$, then this means that $w$ is of the form $u1v$ for some $u \in \{0,1\}^{\kappa-1}$ and $v \in \{0,1\}^{n-\kappa}$, and $x_\kappa$ appears positively in the clause $c_i$. Thus, $w$ satisfies $c_i$. An analogous argument can be used if $w'$ is of the form $X^{\kappa-1}0X^{n-\kappa}$.

Conversely, if $w \notin (\mathcal{V} \otimes_n C_i) \setminus \{N^n\}$, then it must be the case that, for every $w' \in C_i$, $w \otimes_n w' = N^n$ holds. Let $\chi_{ij} = x_\kappa$ with $1 \leq j \leq m_i$. Then $w_{ij} := X^{\kappa-1}1X^{n-\kappa} \in C_i$. As $w \otimes_n w_{ij} = N^n$, this means that $w$ is of the form $u0v$ for some $u \in \{0,1\}^{\kappa-1}$ and $v \in \{0,1\}^{n-\kappa}$. Thus $w$ does not satisfy $\chi_{ij}$. An analogous argument can be used for all variables appearing negatively in $c_i$. Thus, $w$ does not satisfy any literal in $c_i$. □

With the help of this lemma, we can show that the behaviour of the automaton $\mathcal{A}_\psi$ characterises validity of the QBF $\psi$.

**Lemma 4.14.** *If $\|\mathcal{A}_\psi\| \neq \{N^n\}$, then $\psi$ is valid.*

*Proof.* If $\|\mathcal{A}_\psi\| \neq \{N^n\}$, then there exists a run $r$ such that $\mathsf{in}_\psi(r(\varepsilon)) \neq \{N^n\}$ and $\mathsf{wt}_\psi(r) \neq \{N^n\}$. In particular, for every $i, 0 \leq i \leq n+k$, $r(1^i) = q_i$ and $r(1^j) = q_{n+k}$ for every $j > n+k$; in symbols, $r = q_0q_1q_2 \ldots q_{n+k-1}q_{n+k}q_{n+k}^\omega$. Otherwise, there would be a transition with weight $\{N^n\}$, and $\mathsf{wt}_\psi(r) = \{N^n\}$. As explained before,

$$\mathsf{in}_\psi(r(\varepsilon)) \otimes_{\mathsf{qbf}} \bigotimes_{0 \leq i < n} \mathsf{wt}_\psi(q_i, q_{i+1}) = \{0,1\}^n.$$

Moreover, since $\mathsf{in}_\psi(r(\varepsilon)) \otimes_{\mathsf{qbf}} \mathsf{wt}_\psi(r) \neq \{N^n\}$, from Lemma 4.13 it follows that $\mathsf{in}_\psi(r(\varepsilon)) \otimes_{\mathsf{qbf}} \mathsf{wt}_\psi(r)$ is the set of all valuations that satisfy $\phi$. Since this value is in $L_{(n,\{i \mid \nabla_i = \forall\})}$, it follows from the compatibility conditions that for every $i$ with $\nabla_i = \forall$, and $u \in \{0,1\}^{i-1}$, a valuation $u1v$ satisfies $\phi$ iff a valuation $u0v'$ also satisfies $\phi$. Thus, all universal and existential restrictions are satisfied, and $\psi$ is valid. □

**Lemma 4.15.** *If $\psi$ is valid, then $\|\mathcal{A}_\psi\| \neq \{N^n\}$.*

*Proof.* Let $\mathcal{V}$ be the set of all valuations that satisfy $\phi$ and $\mathcal{V}' := \mathcal{V} \setminus \mathsf{inc}_{\mathcal{I}_\forall}(\mathcal{V})$. Since $\psi$ is valid, it holds that $\mathcal{V}'$ is in $L_{(n,\mathcal{I}_\forall)}$ and $\mathcal{V}' \notin \{\emptyset, \{N^n\}\}$. Let now $r$ be the run given by

$$r(1^i) = \begin{cases} q_i & 0 \leq i \leq n+k \\ q_{n+k} & i > n+k, \end{cases}$$

i.e, $r = q_0q_1q_2 \ldots q_{n+k-1}q_{n+k}q_{n+k}^\omega$. Using Lemma 4.13, we obtain that

$$\mathsf{in}_\psi(r(\varepsilon)) \otimes_{\mathsf{qbf}} \mathsf{wt}_\psi(r) = \mathcal{V}' \not\leq \{N^n\}$$

and hence $\|\mathcal{A}_\psi\| \neq \{N^n\}$. □

PSPACE-hardness of the behaviour verification problem is a direct consequence of these lemmas.

**Theorem 4.16.** *Behaviour verification is* PSPACE*-hard.*

*Proof.* As a direct consequence from Lemmas 4.14 and 4.15, a QBF $\psi$ is *not* valid iff behaviour verification answers "yes" on input $\mathcal{A}_\psi, \{N^n\}$. The number of states of $\mathcal{A}_\psi$ is linear on the number of variables and clauses appearing in $\psi$. Moreover, each of the weights used is at most a pair of words of size $n$, where $n$ is the number of variables in $\psi$. Thus, this automaton can be constructed in polynomial time, yielding a polynomial reduction from validity of QBF to the behaviour verification problem of LBTA. Since validity of quantified Boolean formulas is PSPACE-hard and PSPACE =co-PSPACE [29], we have that behaviour verification is PSPACE-hard. □

We emphasize that the encoding of the lattice $L_{\mathsf{qbf}}$ as sets of words, in which every element of this lattice has size described by Equation (1) is fundamental for this lower bound to hold. Indeed, as describe before, one could also measure the size of every element of $L_{\mathcal{S}}$, for $\mathcal{S} = (n, \mathcal{I}_\forall)$, as having size $4^n$. This would yield a linear-space-computable-encoded lattice. However, the constructed automaton would have size exponential on $n$, the number of variables in $\psi$.

## 5. Conclusions

We have studied the complexity of computing the behaviour of lattice Büchi automata over infinite trees. Our results provide different bounds for this problem, depending on the properties of the representation of the elements of the lattice. Mainly, we have studied the problem with respect to arbitrary lattices with the restriction that the meet and join can be computed in polynomial time but without assuming additional properties, such as finiteness or distributivity. For this general setting, we have provided a simple algorithm with exponential runtime, which provides an EXPTIME upper bound. This matches the upper bound previously obtained for bi-locally finite strong bimonoids in [19], a class of algebraic structures that properly contain all bounded lattices. We also provided a reduction from the validity problem of quantified Boolean formulas to the behaviour verification problem of lattice automata, thus obtaining a PSPACE lower bound for the complexity of this problem. Although these lower bounds were shown for infinite trees, very similar arguments can be used for proving hardness also for automata over finite words; the successful runs of our automata are in fact finite words followed by an infinite chain of the distinguished accepting state. Using that state as the only final state for a finite automaton would yield the claimed lower bounds.

Additionally, we studied the complexity of the problem under the assumption that lattice operations do not increase the space required to represent lattice elements; what we call linear-space-computable-encoded lattices. Under this

assumption, we were able to tighten the upper bound to PSPACE. However, the reduction we used to show PSPACE-hardness does not apply to this setting, as it uses an encoding of the lattice elements that is not linear-space-computable-encoded. For this restricted setting we were able to produce NP and co-NP lower bounds. The class of linear-space-computable-encoded lattices is interesting, as it covers the class of all total orders, and many other lattices, such as e.g. finite and distributive lattices, have encodings that make them linear-space-computable-encoded.

Despite the best of our efforts, we were not able to close the gaps between these complexity bounds. However, we conjecture that the ideas used for proving the NP lower bound with respect to linear-space-computable-encoded lattices and the PSPACE-hardness for general lattices can be combined to prove PSPACE-hardness also for linear-space-computable-encoded lattices. As future work, we plan to continue studying techniques for closing these complexity gaps.

It is worth recalling that our complexity results are parameterized on the lattice used. We restricted our investigation to lattices where the meet and the join can be computed in polynomial time, to be able to focus only on the behaviour of the automaton. Also, our hardness results show only that there *exist* some lattices for which behaviour verification is a hard problem. There may well be other lattices for which the problem is easier to solve. In fact, it is already known that for (infinite) distributive lattices, the behaviour can be computed in polynomial time. We will continue studying other restrictions on lattices that might lower the complexity of the problem.

**Acknowledgements**

R. Peñaloza is partially supported by DFG under grant BA 1122/17-1 and within the Cluster of Excellence 'cfAED'.

[1] Anderson, A. R., Belnap, N. D., 1975. Entailment: The Logic of Relevance and Necessity. Vol. 1. Princeton University Press.

[2] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. F. (Eds.), 2003. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press.

[3] Baader, F., Hladik, J., Peñaloza, R., 2008. Automata can show PSPACE results for description logics. Information and Computation 206 (9,10), 1045–1056, special Issue: First International Conference on Language and Automata Theory and Applications (LATA'07).

[4] Baader, F., Knechtel, M., Peñaloza, R., 2009. A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms. In: et al., A. B. (Ed.), Proceedings of the 8th International Semantic Web Conference (ISWC 2009). Vol. 5823 of Lecture Notes in Computer Science. Washington, DC, pp. 49–64.

[5] Baader, F., Knechtel, M., Peñaloza, R., April 2012. Context-dependent views to axioms and consequences of semantic web ontologies. Journal of Web Semantics 12–13, 22–40, available at http://dx.doi.org/10.1016/j.websem.2011.11.006.

[6] Baader, F., Peñaloza, R., August 2010. Automata-based axiom pinpointing. Journal of Automated Reasoning 45 (2), 91–129, Special Issue: Selected Papers from IJCAR 2008.

[7] Baader, F., Tobies, S., 2001. The inverse method implements the automata approach for modal satisfiability. In: Goré, R., Leitsch, A., Nipkow, T. (Eds.), Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001). Vol. 2083 of Lecture Notes in Artificial Intelligence. Springer-Verlag, Siena, Italy, pp. 92–106.

[8] Biere, A., Heule, M. J. H., van Maaren, H., Walsh, T. (Eds.), February 2009. Handbook of Satisfiability. Vol. 185 of Frontiers in Artificial Intelligence and Applications. IOS Press.

[9] Borgwardt, S., Peñaloza, R., 2011. Description logics over lattices with multi-valued ontologies. In: Walsh, T. (Ed.), Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11). Barcelona, Spain, pp. 768–773.

[10] Büchi, J. R., 1960. On a decision method in restricted second order arithmetic. In: Nagel, E., et al. (Eds.), Proceedings of the International Congress on Logic, Methodology and Philosophy of Science. Stanford University Press, pp. 1–11.

[11] Calvanese, D., De Giacomo, G., Lenzerini, M., 1999. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In: Proceedings of the 16th Interntional Joint Conference on Artificial Intelligence (IJCAI'99). pp. 84–89.

[12] Calvanese, D., De Giacomo, G., Lenzerini, M., 2002. 2ATAs make DLs easy. In: Proceedings of the 2002 Description Logic Workshop (DL 2002). pp. 107–118.

[13] Cook, S. A., 1971. The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on Theory of computing. STOC '71. ACM, New York, NY, USA, pp. 151–158.
URL `http://doi.acm.org/10.1145/800157.805047`

[14] Dau, F., Knechtel, M., 2009. Access policy design supported by FCA methods. In: Dau, F., Rudolph, S. (Eds.), Proceedings of the 17th International Conference on Conceptual Structures, (ICCS 2009). Vol. 5662 of Lecture Notes in Computer Science. pp. 141–154.

[15] Droste, M., Götze, D., Märcker, S., Meinecke, I., 2011. Weighted tree automata over valuation monoids and their characterization by weighted logics. In: Kuich, W., Rahonis, G. (Eds.), Algebraic Foundations in Computer Science. Vol. 7020 of Lecture Notes in Computer Science. Springer, pp. 30–55.

[16] Droste, M., Kuich, W., Rahonis, G., 2008. Multi-valued MSO logics over words and trees. Fundamenta Informaticae 84 (3,4), 305–327.

[17] Droste, M., Kuich, W., Vogler, H., 2009. Handbook of Weighted Automata, 1st Edition. EATCS. Springer Publishing Company, Incorporated.

[18] Droste, M., Stüber, T., Vogler, H., 2010. Weighted finite automata over strong bimonoids. Information Sciences 180 (1), 156–166.

[19] Droste, M., Vogler, H., 2012. Weighted automata and multi-valued logics over arbitrary bounded lattices. Theoretical Computer Science 418, 14–36.

[20] Elgot, C. C., 1961. Decision problems of finite automata design and related arithmetics. Transactions of the American Mathematical Society 98, 21–52.

[21] Finkel, O., 2009. Highly undecidable problems about recognizability by tiling systems. Fundamenta Informaticae 91 (2), 305–323.

[22] Gabbay, D. M., Pnueli, A., Shelah, S., Stavi, J., 1980. On the temporal analysis of fairness. In: Proceedings of the 7th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'80). pp. 163–173.

[23] Grätzer, G., 2003. General Lattice Theory, Second Edition. Birkhäuser Verlag.

[24] Horridge, M., Parsia, B., Sattler, U., 2009. Explaining inconsistencies in owl ontologies. In: Godo, L., Pugliese, A. (Eds.), Proceedings of the Third International Conference on Scalable Uncertainty Management (SUM 2009). Vol. 5785 of Lecture Notes in Computer Science. Springer, pp. 124–137.

[25] Knechtel, M., 2010. Access restrictions to and with description logic web ontologies. Ph.D. thesis, Dresden University of Technology, Germany.

[26] Kuich, W., Salomaa, A., 1985. Semirings, Automata and Languages. Vol. 6 of EATCS. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[27] Kupferman, O., Lustig, Y., 2007. Lattice automata. In: Cook, B., Podelski, A. (Eds.), Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2007). Vol. 4349 of Lecture Notes in Computer Science. Springer, pp. 199–213.

[28] Lutz, C., Sattler, U., 2000. The complexity of reasoning with boolean modal logic. In: Proceedings of Advances in Modal Logic 2000 (AiML 2000).

[29] Papadimitriou, C. H., 1994. Computational complexity. Addison-Wesley.

[30] Peñaloza, R., 2010. Using sums-of-products for non-standard reasoning. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (Eds.), Proceedings of the 4th International Conference on Language and Automata Theory and Applications (LATA 2010). Vol. 6031 of Lecture Notes in Computer Science. Springer-Verlag, pp. 488–499.

[31] Pnueli, A., 1977. The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on the Foundations of Computer Science (FOCS'77). pp. 46–57.

[32] Qi, G., Ji, Q., Pan, J. Z., Du, J., Apr. 2011. Extending description logics with uncertainty reasoning in possibilistic logic. International Journal on Intelligent Systems 26 (4), 353–381.
URL http://dx.doi.org/10.1002/int.20470

[33] Rabin, M. O., 1970. Weakly definable relations and special automata. In: Bar-Hillel, Y. (Ed.), Proceedings of Symposium on Mathematical Logic and Foundations of Set Theory. North-Holland Publ. Co., Amsterdam, pp. 1–23.

[34] Stoilos, G., Stamou, G. B., Pan, J. Z., Tzouvaras, V., Horrocks, I., 2007. Reasoning with very expressive fuzzy description logics. Journal of Artificial Intelligence Research (JAIR) 30, 273–320.

[35] Straccia, U., 2001. Reasoning within fuzzy description logics. Journal of Artificial Intelligence Research (JAIR) 14, 137–166.

[36] Tseitin, G. S., 1983. On the complexity of derivation in propositional calculus. In: Siekmann, J., Wrightson, G. (Eds.), Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970. Springer, Berlin, Heidelberg, pp. 466–483.

[37] Vardi, M. Y., Wolper, P., 1986. Automata-theoretic techniques for modal logics of programs. Journal of Computer and System Sciences 32 (2), 183–221.

[38] Wolper, P., Vardi, M. Y., Sistla, A. P., 1983. Reasoning about infinite computation paths. In: Proceedings of the 24th Annual Symposium of Foundations of Computer Science (SFCS'83). IEEE Computer Society, Washington, DC, USA, pp. 185–194.