Inherent Vacuity in Lattice Automata*

Hila Gonen and Orna Kupferman

School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel.

Abstract. *Vacuity checking* is traditionally performed after model checking has terminated successfully. It ensures that all the elements of the specification have played a role in its satisfaction by the system. The need to check the quality of specifications is even more acute in *property-based design*, where the specification is the only input, serving as a basis to the development of the system. *Inherent vacuity* adapts the theory of vacuity in model checking to the setting of property-based design. Essentially, a specification is inherently vacuous if it can be mutated into a simpler equivalent specification, which is known, in the case of specifications in linear temporal logic, to coincide with the fact the specification is satisfied vacuously in all systems.

A recent development in formal methods is an extension of the Boolean setting to a *multi-valued* one. In particular, instead of Boolean automata, which either accept or reject their input, there is a growing interest in weighted automata, which map an input word to a value from a semiring over a large domain. A distributive finite lattice is a special case of a semiring, and *lattice automata* are used in several methods for reasoning about multi-valued objects. We study inherent vacuity in the setting of lattice automata, namely the ability to mutate the value of a transition in the automaton without changing its language. We define the concept of inherent vacuity in lattice automata, study the complexity of deciding different types of vacuity, and relate the setting to the one known for linear temporal logics.

1 Introduction

In recent years, we see a growing awareness to the importance of assessing the quality of (formal) specifications. In the context of model checking, the quality of the specification is assessed by analyzing the effect of applying mutations to the formulas. If the system satisfies the mutated specification, we know that some elements of the specification do not play a role in its satisfaction, thus the specification is satisfied in some *vacuous* way [5, 28]. Vacuity is successfully used in order to improve specifications and detect design errors [26] and has been a subject of extensive research [4, 5, 10, 18, 28, 31].

Property assurance is the activity of eliciting specifications that faithfully capture designer intent [7, 33]. Obvious quality checks one may perform for a given specification are *non-validity* and *satisfiability* [34]. More involved quality checks are studied in the PROSYD project [32]. As discussed in [33], checking vacuity of the specifications

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 278410, and from The Israel Science Foundation (grant no 1229/10).

in the context of property assurance would be of great importance. While early work on vacuity was done in the context of model checking, researchers have also developed the concept of "vacuity without design" [13], which is formalized for linear temporal logic (LTL) formulas in [17], by means of *inherent vacuity*.

Consider a system $\mathcal{S}$ and a formula φ. We say that a subformula ψ of φ *does not affect the satisfaction of* φ *in* $\mathcal{S}$ if $\mathcal{S}$ also satisfies the formula $\forall x.\varphi[\psi \leftarrow x]$, in which ψ is replaced by a universally quantified proposition. Then, a formula φ is *vacuously satisfied* in $\mathcal{S}$ if φ has a subformula that does not affect its satisfaction in $\mathcal{S}$ [4]. Now, as defined in [17], the formula φ is inherently vacuous if there exists a subformula ψ of φ such that $\varphi \equiv \forall x.\varphi[\psi \leftarrow x]$ or, equivalently, if for every system $\mathcal{S}$, if $\mathcal{S} \models \varphi$, then $\mathcal{S}$ satisfies φ vacuously.

The framework in [17] studies specifications given by LTL formulas. A recent development in formal methods is an extension of the Boolean setting to a multi-valued one. In particular, instead of Boolean automata, which either accept or reject their input, there is a growing interest in weighted automata, which map an input word to a value from a semiring over a large domain [15, 30]. Focusing on applications in formal verification, the multi-valued setting arises directly in *quantitative verification* [21] and in reasoning about quality of systems [1], and indirectly in applications like *abstraction methods*, in which it is useful to allow the abstract system to have unknown assignments to atomic propositions and transitions [35], *query checking* [11], which can be reduced to model checking over multi-valued systems, and verification of systems from *inconsistent viewpoints* [23], in which the value of the atomic propositions is the composition of their values in the different viewpoints.

As mentioned above, in the multi-valued setting, the automata map words to a value from a semiring over a large domain. A *distributive finite lattice* is a special case of a semiring. A lattice $\langle A, \leq \rangle$ is a partially ordered set in which every two elements $a, b \in A$ have a least upper bound (a *join* b) and a greatest lower bound (a *meet* b). Finite lattices are useful in many of the applications of the multi-valued setting described above. For example (see Figure 1), in the abstraction application, researchers use the lattice $\mathcal{L}_3$ of three fully ordered values [8], as well as its generalization to $\mathcal{L}_n$ [12]. In query checking, the lattice elements are sets of formulas, ordered by the inclusion order [9]. When reasoning about inconsistent viewpoints, each viewpoint is Boolean, and their composition gives rise to products of the Boolean lattice, as in $\mathcal{L}_{2,2}$ [16]. Finally, when specifying prioritized properties of system, one uses lattices in order to specify the priorities [3].

In a *nondeterministic lattice automaton* on finite words (LNFW, for short) [27], each transition is associated with a *transition value*, which is a lattice element. Intuitively, the value indicates the truth of the statement "the transition exists". Each state in the LNFW is associated with an *initial value* and an *acceptance value*, indicating the truth of the statements "the state is initial/accepting", respectively. The value of a run r of an LNFW $\mathcal{A}$ is the *meet* of the values of all the components of r: the initial value of the first state, the transition values of all the transitions taken along r, and the acceptance value of the last state. The value of a word w is then the *join* of the values of all the runs of $\mathcal{A}$ on w. Accordingly, an LNFW over an alphabet Σ and lattice $\mathcal{L}$ induces an $\mathcal{L}$-language $L : \Sigma^* \to \mathcal{L}$. Note that traditional finite automata (NFWs) correspond to LNFWs over

the lattice $\mathcal{L}_2$. In a *deterministic lattice automaton* on finite words (LDFW, for short), exactly one state has an initial value that is not $\bot$ (the least lattice element), and for every state q and letter σ, at most one state q' is such that the value of the transition from q to q' with σ is not $\bot$. Thus, an LDFW $\mathcal{A}$ has at most one run whose value is not $\bot$ on each input word, and the value of this run is the value of the word in the language of $\mathcal{A}$.

Since being introduced in [27], lattice automata have been used in different contexts. Fully-ordered lattices are sometimes useful as is (for example, when modeling priorities [3]), and sometimes thanks to the fact that real values can often be abstracted to finitely many linearly ordered classes. The power-set lattice models a wide range of partially-ordered values. For example, as mentioned above, in a setting with inconsistent viewpoints, we have a set of agents, each with a different viewpoint of the system, and the truth value of an atomic proposition or a formula indicates the set of agents according to whose viewpoint the atomic proposition or the formula are true. As another example, in [2] the authors study a model of incomplete information in the multi-valued setting using lattice automata. Researchers have also studied theoretical properties of lattice automata, like their minimization and approximation [19, 20], and a bisimulation relation for them [14].

We study vacuity and inherent vacuity in lattice automata. Essentially, the goal is to formalize the ability to mutate the value of a transition in the automaton without changing its language. Consider a transition τ in an LNFW. We say that τ is v-*tolerant*, for a value v of the lattice, if changing the value of τ to v does not change the language of $\mathcal{A}$. We say that a transition τ is *universally flexible* ($\forall$-*flexible*, for short) if τ is v-tolerant for every value v in $\mathcal{L}$. Likewise, τ is *existentially flexible* ($\exists$-*flexible*, for short) if τ is v-tolerant for some value v in $\mathcal{L}$ that is different from the value of τ.

Natural decision problems arise from the above definitions. Specifically, the $\forall$-FLEXIBILITY problem is to decide, given an LNFW and a transition τ in it, whether τ is $\forall$-flexible, and dually for the $\exists$-FLEXIBILITY problem. Solving the flexibility decision problems, we distinguish between four classes of LNFWs, induced by the branching structure of the LNFW (that is, whether it is deterministic or non-deterministic), and the lattice with respect to which it is defined (that is, whether the lattice is fully or partially ordered). Note that our definition of $\forall$-flexible is similar to the definition of "does not affect the satisfaction" for LTL formulas, in the sense that the mutated component is universally quantified. In the case of LTL, checking whether a sub formula ψ affect the the satisfaction of a specification φ, it is possible to check only the "most challenging" mutation – one that replaces ψ by *true* or by *false*, according to the polarity of ψ in φ [28]. Given a transition τ, in $\mathcal{A}$, deciding whether τ is universally or existentially flexible can be done by checking all the mutations of the value of τ. An intermediate question we study is whether it is sufficient to check a single "most challenging" mutation. We show that both universal and existential flexibility are NLOGSPACE-complete for LDFWs and PSPACE-complete for LNFWs, regardless of the type of the lattice. The difference between full-order LNFWs and partial-order LNFWs is reflected, however, in the time complexity of the problems.

As done in [17] for LTL formulas, we introduce and compare two definitions of inherent vacuity for lattice automata. Given two LNFWs $\mathcal{A}$ and $\mathcal{A}'$, we say that the

language of $\mathcal{A}'$ is contained in the language of $\mathcal{A}$, denoted $L(\mathcal{A}') \leq L(\mathcal{A})$, if for every word $w \in \Sigma^*$, we have $L(\mathcal{A}')(w) \leq L(\mathcal{A})(w)$. For two LNFWs $\mathcal{A}$ and $\mathcal{A}'$ such that $L(\mathcal{A}') \leq L(\mathcal{A})$, we say that a transition τ in $\mathcal{A}$ *does not affect the containment of* $L(\mathcal{A}')$ *in* $L(\mathcal{A})$, if for every $v \in \mathcal{L}$, the inequality $L(\mathcal{A}') \leq L(\mathcal{A})$ holds also when changing the value of τ in $\mathcal{A}$ to v. Also, $\mathcal{A}'$ is *vacuously contained* in $\mathcal{A}$ if there is a transition τ in $\mathcal{A}$ that does not affect the containment of $L(\mathcal{A}')$ in $L(\mathcal{A})$. Now, an LNFW $\mathcal{A}$ is *inherently vacuous* if there exists a $\forall$-flexible transition in $\mathcal{A}$, which we show to be equivalent to a definition according to which $\mathcal{A}$ is inherently vacuous if for every LNFW $\mathcal{A}'$, if $L(\mathcal{A}') \leq L(\mathcal{A})$, then $\mathcal{A}'$ is vacuously contained in $\mathcal{A}$. Thus, as in the case of LTL formulas, the two definitions coincide.

Due to the lack of space, some proofs are missing and can be found in a full version, in the authors' URLs.

2 Preliminaries

2.1 Lattices

Let $\langle A, \leq \rangle$ be a partially ordered set, and let P be a subset of A. An element $a \in A$ is an *upper bound* on P if $a \geq b$ for all $b \in P$. Dually, a is a *lower bound* on P if $a \leq b$ for all $b \in P$. An element $a \in A$ is the *least element of* P if $a \in P$ and a is a lower bound on P. Dually, $a \in A$ is the *greatest element of* P if $a \in P$ and a is an upper bound on P. A partially ordered set $\langle A, \leq \rangle$ is a *lattice* if for every two elements $a, b \in A$ both the least upper bound and the greatest lower bound of $\{a, b\}$ exist, in which case they are denoted $a \vee b$ (a *join* b) and $a \wedge b$ (a *meet* b), respectively. A lattice is *fully ordered* if every two elements in it are comparable. Note that w.l.o.g. every fully-ordered lattice corresponds to the lattice $\langle \{0, \ldots, n\}, \leq \rangle$ for some n. For ease of presentation, from now on we assume that every fully-ordered lattice is the lattice $\langle \{0, \ldots, n\}, \leq \rangle$ for some n. We use $a < b$ to indicate that $a \leq b$ and $a \neq b$. We say that a is a *child* of b, denoted $a \prec b$, if $a < b$ and there is no c such that $a < c < b$. A lattice is *complete* if for every subset $P \subseteq A$ both the least upper bound and the greatest lower bound of P exist, in which case they are denoted $\bigvee P$ and $\bigwedge P$, respectively. In particular, $\bigvee A$ and $\bigwedge A$ are denoted $\top$ (*top*) and $\bot$ (*bottom*), respectively. A lattice $\langle A, \leq \rangle$ is finite if A is finite. Note that every finite lattice is complete. A lattice $\langle A, \leq \rangle$ is *distributive* if for every $a, b, c \in A$, we have $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ and $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.

Consider a lattice $\mathcal{L} = \langle A, \leq \rangle$. We sometimes abuse notation and refer to $\mathcal{L}$ also as a set of elements, and thus talk about elements $l \in \mathcal{L}$ (rather than $l \in A$). A *join irreducible element* $l \in \mathcal{L}$ is a value, other than $\bot$, such that for all $a, b \in \mathcal{L}$, if $a \vee b \geq l$ then either $a \geq l$ or $b \geq l$. We denote the set of join irreducible elements of $\mathcal{L}$ by $JI(\mathcal{L})$. By Birkhoff's representation theorem for finite distributive lattices, in order to prove that $a = b$ it is sufficient to prove that for every join irreducible element l, it holds that $a \geq l$ iff $b \geq l$.

In Figure 1 we describe some finite lattices. The elements of the lattice $\mathcal{L}_2$ are the usual truth values 1 (*true*) and 0 (*false*) with the order $0 \leq 1$. The lattice $\mathcal{L}_n$ contains the values $0, 1, \ldots, n-1$, with the order $0 \leq 1 \leq, \ldots, \leq n-1$. The lattice $\mathcal{L}_{2,2}$ is the Cartesian product of two $\mathcal{L}_2$ lattices, thus $(a, b) \leq (a', b')$ if both $a \leq a'$ and $b \leq b'$.

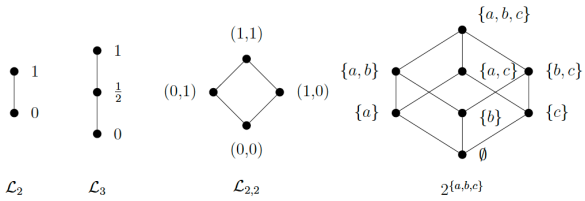

Fig. 1. Some lattices.

Finally, the lattice $2^{\{a,b,c\}}$ is the power set of $\{a, b, c\}$ with the set-inclusion order. In this lattice, for example, $\{a\} \vee \{b\} = \{a, b\}$, $\{a\} \wedge \{b\} = \bot$, $\{a, c\} \vee \{b\} = \top$, and $\{a, c\} \wedge \{b\} = \bot$. Note that the join irreducible elements of the lattice $\mathcal{L}_n$ are all the elements in the lattice except for $\bot$. In the case of the lattice $2^{\{a,b,c\}}$, the join irreducible elements are all the singletons, that is, $JI(\mathcal{L}) = \{\{a\}, \{b\}, \{c\}\}$.

We define the *graph of* $\mathcal{L}$ as the undirected graph $\langle A, E_{\prec}\rangle$ in which $E_{\prec}(v, v')$ iff $v \prec v'$ or $v' \prec v$. The *distance* between two elements $a, b \in \mathcal{L}$, denoted $dist(a, b)$, is the shortest path from a to b in the graph of $\mathcal{L}$. For example, in the fully-ordered lattice $\mathcal{L}$, we have $dist(i, j) = |i - j|$, and in the power-set lattice, the distance coincides with the Hamming distance, thus $dist(X_1, X_2) = |(X_1 \setminus X_2) \cup (X_2 \setminus X_1)|$. When $dist(a, b) = 1$, we say that a and b are neighbors. Note that a and b are neighbors iff $a \prec b$ or $b \prec a$. For two elements i and j in a fully-ordered lattice, we define $i + j$ as $min\{\top, i + j\}$ and $i - j$ as $max\{\bot, i - j\}$.

For a set X of elements, an *$\mathcal{L}$-set over X* is a function $S : X \to \mathcal{L}$ assigning to each element of X a value in $\mathcal{L}$. It is convenient to think about $S(x)$ as the truth value of the statement "x is in S". We say that an $\mathcal{L}$-set S is *Boolean* if $S(x) \in \{\top, \bot\}$ for all $x \in X$.

Consider a lattice $\mathcal{L}$ and an alphabet Σ. An *$\mathcal{L}$-language* over Σ is an $\mathcal{L}$-set over Σ^*. Thus, an $\mathcal{L}$-language $L : \Sigma^* \to \mathcal{L}$ assigns a value in $\mathcal{L}$ to each word over Σ. For two $\mathcal{L}$-languages L_1 and L_2, we say that L_1 is *contained* in L_2, denoted $L_1 \leq L_2$, if for every word $w \in \Sigma^*$ it holds that $L_1(w) \leq L_2(w)$. The meet of two languages L_1 and L_2, denoted $L_1 \wedge L_2$, is the language that maps each word $w \in \Sigma^*$ to the meet of the values of w in L_1 and in L_2; that is, for all w, we have that $(L_1 \wedge L_2)(w) = L_1(w) \wedge L_2(w)$. The join of L_1 and L_2, denoted $L_1 \vee L_2$, is defined dually, thus, for every w, we have $(L_1 \vee L_2)(w) = L_1(w) \vee L_2(w)$.

Below is a useful extension of Birkhoff's representation theorem [6] from equality to inequality.

Proposition 1. *Consider a lattice $\mathcal{L}$ and two elements $a, b \in \mathcal{L}$. If for every join irreducible element $l \in \mathcal{L}$ it holds that $a \geq l$ implies $b \geq l$, then $b \geq a$.*

2.2 Lattice Automata

A *nondeterministic lattice automaton* on finite words (LNFW, for short) [27] is a six-tuple $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F\rangle$, where $\mathcal{L}$ is a finite lattice, Σ is an alphabet, Q is a

finite set of states, $Q_0 \in \mathcal{L}^Q$ is an $\mathcal{L}$-set of initial states, $\delta \in \mathcal{L}^{Q\times\Sigma\times Q}$ is an $\mathcal{L}$-set of transitions, and $F \in \mathcal{L}^Q$ is an $\mathcal{L}$-set of accepting states. An LNFW is a full-order LNFW if $\mathcal{L}$ is a fully-ordered lattice. Otherwise, it is called a partial-order LNFW to emphasize that the lattice is not fully-ordered. We use $|\mathcal{A}|$ to refer to the size of $\mathcal{A}$, that is, $|\mathcal{A}| = |Q \times \Sigma \times Q|$.

A *run* of $\mathcal{A}$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$ is a sequence $r = \tau_1, \ldots, \tau_n$ of n successive transitions, where $\tau_i \in Q \times \Sigma \times Q$. Let $q_0, \ldots, q_n$ be such that $\tau_i = \langle q_{i-1}, \sigma_i, q_i \rangle$ for every $1 \leq i \leq n$. In particular, q_0 is the first state of the run, and q_n is the last state of the run. The *value* of r is $val(r) = Q_0(q_0) \wedge \bigwedge_{i=1}^{n} \delta(\tau_i) \wedge F(q_n)$. Intuitively, $Q_0(q_0)$ is the value of q_0 being initial, $\delta(\tau_i)$ is the value of taking the transition τ_i, namely, the value of q_i being a successor of q_{i-1} when σ_i is the input letter, $F(q_n)$ is the value of q_n being accepting, and the value of r is the meet of all these values.

We refer to $Q_0(q_0) \wedge \bigwedge_{i=1}^{n} \delta(\tau_i)$ as the *traversal value* of r and refer to $F(q_n)$ as its *acceptance value*. For a word w, the value of $\mathcal{A}$ on w, denoted $\mathcal{A}(w)$, is the join of the values of all the possible runs of $\mathcal{A}$ on w. That is, $val(\mathcal{A}, w) = \bigvee\{val(r) : r \text{ is a run of } \mathcal{A} \text{ on } w\}$. The $\mathcal{L}$-language of $\mathcal{A}$, denoted $L(\mathcal{A})$, maps each word w to its value in $\mathcal{A}$. That is, $L(\mathcal{A})(w) = val(\mathcal{A}, w)$.

Let $\mathcal{A}$ be an LNFW, and δ_1, δ_2 be $\mathcal{L}$-sets of transitions of $\mathcal{A}$. We say that $\delta_1 \leq \delta_2$ if for every transition $\tau \in Q \times \Sigma \times Q$, it holds that $\delta_1(\tau) \leq \delta_2(\tau)$.

An LNFW is *deterministic* (LDFW, for short) if there is exactly one state $q \in Q$, called the *initial state* of $\mathcal{A}$, such that $Q_0(q) \neq \bot$, and for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at most one state $q' \in Q$, called the *σ-successor* of q, such that $\delta(q, \sigma, q') \neq \bot$. Note that if $\mathcal{A}$ is deterministic, then it has at most one run on w whose value is not $\bot$.

Traditional nondeterministic automata over finite words (NFW, for short) correspond to LNFW over the lattice $\mathcal{L}_2$. Indeed, over $\mathcal{L}_2$, the value of a run r is either $\top$, in case the run uses only transitions with value $\top$ and its final state has value $\top$, or $\bot$ otherwise. Also, the value of $\mathcal{A}$ on w is $\top$ iff the value of some run on it is $\top$. This reflects the fact that a word w is accepted by an NFW if some legal run on w is accepting. Similarly, traditional deterministic automata over finite words (DFW, for short) correspond to LDFW over the lattice $\mathcal{L}_2$.

Below is a simple yet useful proposition about the relation between two LNFWs.

Proposition 2. *Let $\mathcal{A}_1 = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta_1, F \rangle$ and $\mathcal{A}_2 = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta_2, F \rangle$ be LNFWs such that $\delta_1 \leq \delta_2$. Then, for every word $w \in \Sigma^*$, it holds that $L(\mathcal{A}_1)(w) \leq L(\mathcal{A}_2)(w)$.*

3 Vacuity in Lattice Automata

The essence of vacuity is detection of components of the specification that play no role in its satisfaction. In this section we formalize and study this intuition in the setting of lattice automata. That is, we formalize and study the influence that the value of a single transition has or may not have on the language of a lattice automaton.

We start by defining tolerance and flexibility of transitions, which formalize and quantify the ability to mutate the value of transitions without changing the language of the automaton. We first need some definitions regarding runs of lattice automata.

Consider an LNFW $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F \rangle$. We say that a run r on a word w is *a critical run* in $\mathcal{A}$ if removing it from the set of runs of $\mathcal{A}$ on w changes the value of w in $\mathcal{A}$. Formally, $L(\mathcal{A})(w) \neq \bigvee\{val(r') : r' \neq r \text{ is a run of } \mathcal{A} \text{ on } w\}$. Note that for the case of a full-order LNFW, a run r on a word w is critical iff $L(\mathcal{A})(w) = val(r)$ and there is no run $r' \neq r$ on w such that $L(\mathcal{A})(w) = val(r')$.

Consider a run r, and let q_0 and q_n be the first and last states of r, respectively. For a transition τ taken in r, the value of r without τ, denoted $val_{-\tau}(r)$, is $Q_0(q_0) \wedge \bigwedge_{\tau' \in \{r \setminus \tau\}} \delta(\tau') \wedge F(q_n)$. We say that τ is *a bottleneck in* r if $val(r) \neq val_{-\tau}(r)$. That is, removing the effect of τ from the value of r changes it. Note that since the value of a run r on w is the meet of the values of all its components (transitions, initial state and accepting state), for the case of a fully-ordered lattice, the value of a run is actually determined by the minimal value throughout the run. Thus, in a full-order LNFW, a transition τ is a bottleneck in a run r iff $\delta(\tau)$ is the minimal value in r, and there is no other value v throughout the run r such that $\delta(\tau) = v$.

For a transition τ in $\mathcal{A}$, we use $\mathcal{A}_{\tau \leftarrow v}$ to denote $\mathcal{A}$ with the value $\delta(\tau)$ being changed to v. We say that τ is *v-tolerant* if changing $\delta(\tau)$ to v does not change the language of $\mathcal{A}$; that is, if $L(\mathcal{A}) = L(\mathcal{A}_{\tau \leftarrow v})$.

We say that a transition τ is *universally flexible with respect to* δ (*$\forall$-flexible*, for short, when δ is clear from the context) if τ is v-tolerant for every value v in $\mathcal{L}$. Likewise, τ is *existentially flexible with respect to* δ (*$\exists$-flexible*, for short) if τ is v-tolerant for some value $v \neq \delta(\tau)$ in $\mathcal{L}$.

Remark 1. Recall that an LNFW over the lattice $\mathcal{L}_2$ is a standard NFW. In this case, we get that a transition τ is $\forall$-flexible iff τ is $\exists$-flexible. Consider an NFW $\mathcal{A}' = \langle \Sigma, Q, Q_0, \delta, F \rangle$. For every $q, q' \in Q$ and $\sigma \in \Sigma$, the transition $\tau = \langle q, \sigma, q' \rangle$ is *flexible* if it exists in $\mathcal{A}'$, and removing it does not change the language of $\mathcal{A}'$, or if it does not exist in $\mathcal{A}'$, and adding it as a transition does not change the language of $\mathcal{A}'$. Note that a transition in an NFW is flexible iff its corresponding transition in the matching LNFW over the lattice $\mathcal{L}_2$ is $\forall$-flexible, or, equivalently, $\exists$-flexible. □

Two basic questions we would like to study consider the universal and existential flexibility of transitions, as formally specified below.

- $\forall$-FLEXIBILITY: Given an LNFW and a transition τ in it, decide whether τ is $\forall$-flexible.
- $\exists$-FLEXIBILITY: Given an LNFW and a transition τ in it, decide whether τ is $\exists$-flexible.

Remark 2. The definitions above refer to a single transition. That is, our study examines the influence of the value of a single transition on the language of the automaton. In the full version, we consider also sets of transitions. There, we define *$\forall$-uniform-flexibility*, which indicates that we can mutate the vector of values of the transitions in the set to any uniform vector of values without changing the language, and *$\forall$-mixed-flexibility*, which indicates that we can mutate the vector of values to any vector without changing the language of the automaton. We prove equivalence between these two definitions, study also the dual *$\exists$-uniform-flexibility* and *$\exists$-mixed-flexibility* notions, and study the complexity of the corresponding decision problems. □

4 Useful Observations on Tolerance and Flexibility

In this section we provide some useful observations towards the solution of the flexibility decision problems. We distinguish between four classes of LNFWs, induced by the branching structure of the LNFW (that is, whether it is deterministic or nondeterministic), and the lattice with respect to which it is defined (that is, whether the lattice is fully or partially ordered). Note that the four classes are partially ordered according to their generalization, with the deterministic linear class being a special case of the nondeterministic linear and the deterministic partially ordered classes. The latter two classes are not ordered, and are special cases of the most general class, namely the one of nondeterministic and partial-order LNFWs. Accordingly, we are going to present positive results on the most general class for which they apply, and present negative results on the most restricted ones. Throughout the section we refer to a lattice automaton $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F \rangle$.

In the context of vacuity in LTL, we say that a subformula ψ of a specification φ does not affect the satisfaction of φ in a system $\mathcal{S}$ that satisfies φ if $\mathcal{S}$ also satisfies the specification obtained from φ by replacing ψ by a universally quantified atomic proposition. Thus, the approach taken there is the universal one – all mutations of ψ should result in a formula that is satisfied in $\mathcal{S}$. It is shown in [28] that rather than checking φ with ψ being replaced by a universally quantified atomic proposition, it is sufficient to check a single "most challenging" mutation – one that replaces ψ by *true* or by *false*, according to the polarity of ψ in φ. Given a transition τ in $\mathcal{A}$, deciding whether τ is $\forall$-flexible or $\exists$-flexible can be done by checking all the replacements to $\delta(\tau)$. One of the questions we would like to answer is whether it is sufficient to change $\delta(\tau)$ to $\bot$, $\top$, or perhaps to another single value in order to answer the flexibility questions.

We first show that there is no single value $v \in \mathcal{L}$ such that for every transition τ in $\mathcal{A}$, the transition τ is $\forall$-flexible iff τ is v-tolerant. This holds already for full-order LDFWs.

Example 1. Consider the LDFW $\mathcal{A}$ with $\mathcal{L} = \{1, 2, 3\}$, described in Figure 2.

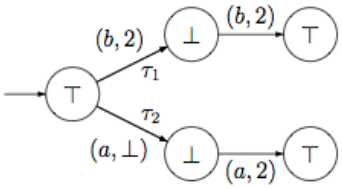

Fig. 2. No single value to check.

It is easy to see that $L(\mathcal{A})(bb) = 2$ and $L(\mathcal{A})(aa) = \bot$. The transitions τ_1 and τ_2 are not $\forall$-flexible. Indeed, if we change $\delta(\tau_1)$ to $\bot$, we get $L(\mathcal{A})(bb) = \bot$, and if we change $\delta(\tau_2)$ to $\top$ we get $L(\mathcal{A})(aa) = 2$. Assume by way of contradiction that there is a value $v \in \mathcal{L}$ that satisfies the requirement in the claim. If $v \geq 2$, then changing $\delta(\tau_1)$ to v does not change the language of $\mathcal{A}$. Thus, we get that τ_1 is not $\forall$-flexible, but

is v-tolerant. Otherwise, $v < 2$ and changing $\delta(\tau_2)$ to v does not change the language of $\mathcal{A}$. Thus, we get that τ_2 is not $\forall$-flexible, but is v-tolerant. Hence, there is no single value that enables us to determine the $\forall$-flexibility of all the transitions in $\mathcal{A}$. $\square$

Thus, we can not expect to check flexibility of all the transitions in a lattice automaton using a single value, in particular the values $\bot$ and $\top$ do not serve as a single replacement. In the following sections we check the situation for a single transition, and we consider universal and existential flexibility in the four classes of lattice automata.

4.1 Full-order LDFW

Universal flexibility Recall that a transition τ of $\mathcal{A}$ is $\forall$-flexible if τ is v-tolerant for every value v in $\mathcal{L}$. Since $\mathcal{L}$ is fully ordered, it is tempting to believe that we can check the tolerance of τ with respect to a single "most challenging" value. The transition τ_1 in the LDFA in Example 1 demonstrates that $\top$-tolerance does not imply $\forall$ flexibility. Indeed, τ_1 is $\top$-tolerant but is not $\forall$-flexible, as changing $\delta(\tau_1)$ to $\bot$ changes $L(\mathcal{A})(bb)$ to $\bot$. As we now show, however, a unique check is sufficient for checking universal flexibility. This is similar to the case of subformulas in LTL, where a unique (either *true* or *false*) mutation is sufficient, and depends on the polarity of the mutated subformula. Here, the original value plays the role of the polarity.

Proposition 3. *Consider a transition τ in a full-order LDFW $\mathcal{A}$. If $\delta(\tau) \neq \bot$, then τ is $\forall$-flexible iff τ is $\bot$-tolerant. If $\delta(\tau) = \bot$, then τ is $\forall$-flexible iff τ is $\top$-tolerant.*

Proof. We start with the case $\delta(\tau) \neq \bot$. First, if τ is $\forall$-flexible, then, by definition, τ is $\bot$-tolerant. Now, since $\mathcal{A}$ is deterministic, if changing $\delta(\tau)$ to $\bot$ does not change the language of $\mathcal{A}$, then the value of every run that traverses τ was $\bot$ before the change. Since $\mathcal{L}$ is fully ordered, this means that every run that traverses τ had value $\bot$ in it, either in a transition or in an initial or an accepting state. Thus, the value of every run that traverses this transition is $\bot$ regardless what $\delta(\tau)$ is, or in other words, τ is $\forall$-flexible.

We continue to the case $\delta(\tau) = \bot$. First, if τ is $\forall$-flexible, then, by definition, τ is $\top$-tolerant. Now, if τ is $\top$-tolerant, then we have $L(\mathcal{A}) = L(\mathcal{A}_{\tau \leftarrow \top})$. Let v be a value in $\mathcal{L}$. By Proposition 2, since $\bot \leq v \leq \top$ we have that $L(\mathcal{A}) \leq L(\mathcal{A}_{\tau \leftarrow v}) \leq L(\mathcal{A}_{\tau \leftarrow \top})$. Thus, we get that $L(\mathcal{A}) = L(\mathcal{A}_{\tau \leftarrow v})$ for every $v \in \mathcal{L}$, namely, τ is $\forall$-flexible, and we are done.

Existential Flexibility Recall that a transition τ of $\mathcal{A}$ is $\exists$-flexible if τ is v-tolerant for some value $v \neq \delta(\tau)$ in $\mathcal{L}$. The transition τ_1 in the LDFA in Example 1 demonstrates that $\exists$-flexibility does not imply $\bot$-tolerance. Indeed, while changing $\delta(\tau_1)$ to $\bot$ changes $L(\mathcal{A})(bb)$ to $\bot$, the transition τ_1 is $\top$-tolerant. As in the case of universal flexibility, however, a unique check is sufficient.

Lemma 1. *Consider a transition τ in an LDFW $\mathcal{A}$. If τ is a bottleneck in some run, then it is not $\exists$-flexible.*

Proof. Since $\mathcal{A}$ is over a fully ordered lattice, then τ being a bottleneck in some run implies that $\delta(\tau)$ is the meet of all the values throughout that run, and there is no value throughout that run that equals $\delta(\tau)$. Thus, since $\mathcal{A}$ is deterministic, changing $\delta(\tau)$ to a lower value decreases the value of some word in the language of $\mathcal{A}$, and changing $\delta(\tau)$ to a greater value increases the value of some word in the language of $\mathcal{A}$. Thus, the transition τ is not $\exists$-flexible.

Proposition 4. *Consider a transition τ in an LDFW $\mathcal{A}$. If $\delta(\tau) \neq \top$, then τ is $\exists$-flexible iff τ is $\top$-tolerant. If $\delta(\tau) = \top$, then τ is $\exists$-flexible iff τ is $(\top - 1)$-tolerant.*

Proof. We start with the case $\delta(\tau) \neq \top$. First, if τ is $\top$-tolerant, then, by definition, τ is $\exists$-flexible. Now, if τ is $\exists$-flexible, then by Lemma 1 we get that τ is not a bottleneck in any run. Thus, we can increase $\delta(\tau)$ without changing the language and τ is $\top$-tolerant.

We continue to the case $\delta(\tau) = \top$. First, if τ is $(\top - 1)$-tolerant, then, by definition, τ is $\exists$-flexible. Now, if τ is $\exists$-flexible, then τ is v-tolerant for some value $v \neq \top$ in $\mathcal{L}$. Thus, we have $L(\mathcal{A}_{\tau \leftarrow v}) = L(\mathcal{A})$. Since $v \leq (\top - 1) \leq \tau$, we get by Proposition 2 that $L(\mathcal{A}_{\tau \leftarrow v}) \leq L(\mathcal{A}_{\tau \leftarrow (\top - 1)}) \leq L(\mathcal{A})$, and so $L(\mathcal{A}_{\tau \leftarrow (\top - 1)}) = L(\mathcal{A})$. Namely, the transition τ is $(\top - 1)$-tolerant.

4.2 Full-order LNFW

In Propositions 3 and 4 we showed that in the case of full-order LDFW, if $\delta(\tau) \neq \bot$ then τ is $\forall$-flexible iff τ is $\bot$-tolerant, and that if $\delta(\tau) \neq \top$ then τ is $\exists$-flexible iff τ is $\top$-tolerant. As we now show in Example 2 below, This does not hold for LNFWs.

Example 2. with Let $\mathcal{L} = \{1, 2, 3\}$. Consider the LNFW $\mathcal{A}_1$ described in the left of Figure 3.

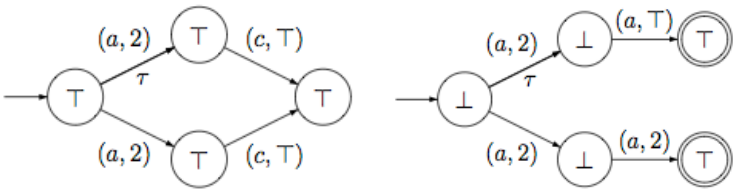

Fig. 3. Propositions 3 and 4 do not hold for partial-order LDFWs.

It is easy to see that $L(\mathcal{A}_1)(a) = L(\mathcal{A}_1)(ac) = 2$. Consider the upper-left transition τ. If we change $\delta(\tau)$ to $\bot$ we get an equivalent LNFW. However, τ is not $\forall$-flexible. Indeed, changing $\delta(\tau)$ to $\top$ changes $L(\mathcal{A}_1)(a)$ and $L(\mathcal{A}_1)(ac)$ to $\top$.

Consider now the LNFW $\mathcal{A}_2$ described in the right of the figure. It is easy to see that $L(\mathcal{A}_2)(aa) = 2$. Consider the upper-left transition τ. If we change $\delta(\tau)$ to $\bot$ we get an equivalent LNFW. However, τ is not $\top$-tolerant. Indeed, changing $\delta(\tau)$ to $\top$ changes $L(\mathcal{A}_2)(aa)$ to $\top$. □

Example 2 is a negative result for the class of full-order LNFWs. In Propositions 5 and 6 we will show a positive result for the more general partial-order LNFW.

4.3 Partial-order LDFW

In Example 2 we showed that Propositions 3 and 4, which apply to full-order LDFWs, do not hold for full-order LNFWs. Below we show that they do not hold for partial-order LDFWs either. Thus, we conclude that Propositions 3 and 4 are tight for full-order LDFWs and do not hold for partial-order LDFWs or for LNFWs.

Example 3. Let $\mathcal{L} = 2^{\{a,b\}}$. Consider the LDFW $\mathcal{A}$ described in Figure 4.

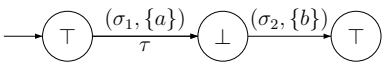

Fig. 4. Propositions 3 and 4 do not hold for full-order LNFWs.

It is easy to see that $L(\mathcal{A})(\sigma_1 \cdot \sigma_2) = \emptyset$. Consider the left transition τ. If we change $\delta(\tau)$ to $\bot$ we get an equivalent LDFW. However, τ is not $\forall$-flexible. Indeed, changing $\delta(\tau)$ to $\{a, b\}$ changes $L(\mathcal{A})(\sigma_1 \cdot \sigma_2)$ to $\{b\}$. Thus, there exists a partial-order LDFW with a transition τ in it, such that $\delta(\tau) \neq \bot$ and τ is $\bot$-tolerance but is not $\forall$-flexible.

Further observe that τ, which is $\bot$-tolerance and hence $\exists$-flexible, is not $\top$-tolerant. Indeed, changing $\delta(\tau)$ to $\top$ changes $L(\mathcal{A}_2)(\sigma_1 \cdot \sigma_2)$ to $\{b\}$. Thus, there exists a partial-order LDFW with a transition τ in it, such that $\delta(\tau) \neq \top$ and τ is $\exists$-flexible but is not $\top$-tolerant. In particular, $\exists$-flexibility does not imply $\top$-tolerance. □

Example 3 is a negative result for the class of partial-order LDFWs. In Propositions 5 we will show a positive result for the more general partial-order LNFW. The last negative result we are going to show concerns existential flexibility and shows that there, checking even both extreme values $\top$ and $\bot$ may not be of help. In Example 4 below we formalize this intuition.

Example 4. Consider the LDFW $\mathcal{A}$ with $\mathcal{L} = 2^{\{a,b,c\}}$, described in Figure 5.

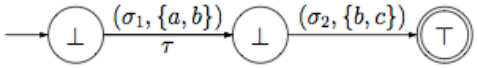

Fig. 5. τ is $\exists$-flexible, but is neither $\bot$-tolerant nor $\top$-tolerant.

It is easy to see that $L(\mathcal{A})(\sigma_1 \cdot \sigma_2) = \{b\}$. Consider the left transition τ. If we change $\delta(\tau)$ to $\{b\}$ we get an equivalent LDFW, thus, τ is $\exists$-flexible. However, changing $\delta(\tau)$ to $\bot$ changes $L(\mathcal{A})(\sigma_1 \cdot \sigma_2)$ to $\bot$, and changing $\delta(\tau)$ to $\top$ changes $L(\mathcal{A})(\sigma_1 \cdot \sigma_2)$ to $\{b, c\}$, thus, τ is neither $\bot$-tolerant nor $\top$-tolerant. □

4.4 Partial-order LNFW

Universal flexibility As shown in Examples 2 and 3, checking only $\bot$-tolerance or $\top$-tolerance is not sufficient in order to determine $\forall$-flexibility. As we show, however, in Proposition 5 below, checking both is sufficient, even in the most general model.

Proposition 5. *A transition τ in an LNFW $\mathcal{A}$ is $\forall$-flexible iff τ is both $\bot$-tolerant and $\top$-tolerant.*

Proof. If τ is $\forall$-flexible, then, by definition, τ is $\bot$-tolerant and $\top$-tolerant. Now, if τ is $\bot$-tolerant and $\top$-tolerant, we have $L(\mathcal{A}_{\tau\leftarrow\bot}) = L(\mathcal{A}) = L(\mathcal{A}_{\tau\leftarrow\top})$. Let v be a value in $\mathcal{L}$. By Proposition 2, since $\bot \leq v \leq \top$, we have that $L(\mathcal{A}_{\tau\leftarrow\bot}) \leq L(\mathcal{A}_{\tau\leftarrow v}) \leq L(\mathcal{A}_{\tau\leftarrow\top})$. Since $L(\mathcal{A}_{\tau\leftarrow\bot}) = L(\mathcal{A}_{\tau\leftarrow\top})$, it must be that $L(\mathcal{A}_{\tau\leftarrow\bot}) = L(\mathcal{A}_{\tau\leftarrow v}) = L(\mathcal{A}_{\tau\leftarrow\top})$. This holds for every $v \in \mathcal{L}$, thus, we get that τ is v-tolerant for every $v \in \mathcal{L}$, that is, τ is $\forall$-flexible, and we are done.

By Proposition 2, since $\bot \leq \delta(\tau) \leq \top$, we have $L(\mathcal{A}_{\tau\leftarrow\bot}) \leq L(\mathcal{A}) \leq L(\mathcal{A}_{\tau\leftarrow\top})$. Hence, the two tolerance checks from Proposition 5 can be performed in a single language-containment check:

Lemma 2. *Consider an LNFW $\mathcal{A}$. A transition τ in $\mathcal{A}$ is $\forall$-flexible iff $L(\mathcal{A}_{\tau\leftarrow\top}) \leq L(\mathcal{A}_{\tau\leftarrow\bot})$.*

Existential flexibility Unlike the case of $\forall$-flexibility, which amounts to tolerance of the two extreme values, namely $\top$-tolerance and $\bot$-tolerance, Example 4 shows that this is not true for $\exists$-flexibility, even in LDFW. As we prove below, we can still avoid checking all possible values and restrict attention to the neighbors of the original value of the mutated transition.

Lemma 3. *Let τ be a transition in an LNFW $\mathcal{A}$. If τ is v'-tolerant for some $v' \in \mathcal{L}$, then τ is v''-tolerant for every value $v'' \in \mathcal{L}$ such that $(\delta(\tau) \wedge v') \leq v'' \leq v'$, $v' \leq v'' \leq (\delta(\tau) \vee v')$, $(\delta(\tau) \wedge v') \leq v'' \leq \delta(\tau)$, or $\delta(\tau) \leq v'' \leq (\delta(\tau) \vee v')$.*

Lemma 3 implies that the search for a value with respect to which a transition is tolerant can consider only the neighbors of the current value. Formally, we have the following. The proof, which appears in the full version, analyzes all possible relations between the value of $\delta(\tau)$ and a value that witnesses its $\exists$-flexibility.

Proposition 6. *Consider an LNFW $\mathcal{A}$ and a transition τ in $\mathcal{A}$. The transition τ is $\exists$-flexible iff τ is v'-tolerant for some neighbor v' of $\delta(\tau)$ in the graph of the lattice $\mathcal{L}$.*

Remark 3. In order to justify the need to check all the neighbors of $\delta(\tau)$, consider the LDFW $\mathcal{A}$ with $\mathcal{L} = 2^{\{a,b,c,d\}}$, described in Figure 6.

Consider the right transition τ'. If $\delta(\tau') = \{a, b, d\}$, then the only value v for which τ is v-tolerant is $\{a, b\}$. If $\delta(\tau') = \{a, c, d\}$, then the only value v for which τ is v-tolerant is $\{a, c\}$. If $\delta(\tau') = \{b, c, d\}$, then the only value v for which τ is v-tolerant is $\{b, c\}$. If $\delta(\tau') = \{a, b, c\}$, then the only value v for which τ is v-tolerant is $\{a, b, c, d\}$. It is easy to see that in each case, the only value that can give an indication for the $\exists$-flexibility of τ is a neighbor of $\delta(\tau)$. Also, note that every neighbor of $\delta(\tau)$ is useful in one of the cases. Thus, it is required to check at least all the neighbors of $\delta(\tau)$ in order to determine $\exists$-flexibility. $\square$

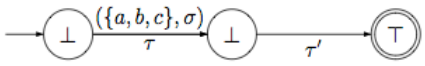

Fig. 6. checking tolerance for neighbors.

5 Complexity of the Decision Problems

In this section we use the observations from Section 4 in order to find the complexity of the flexibility decision problems.

We first prove a lower bound for the flexibility problem in DFWs and NFWs. As discussed in Remark 1, an NFW corresponds to an LNFW over the lattice $\mathcal{L}_2$, and similarly, a DFW corresponds to an LDFW over this lattice. Then, universal and existential flexibility coincide, and a transition in an NFW is flexible iff the corresponding transition in the matching LNFW over the lattice $\mathcal{L}_2$ is $\forall$-flexible and $\exists$-flexible. Accordingly, the FLEXIBILITY problem for NFW is to decide, given an NFW and a transition τ in it, whether τ is flexible.

Theorem 1. *The FLEXIBILITY problem is NLOGSPACE-hard for DFWs and is PSPACE-hard for NFWs.*

Proof. We start with DFWs and describe a reduction from the non-reachability problem, proven to be NLOGSPACE-hard in [24, 25]. Given a graph $G = \langle V, E\rangle$ and two vertices u, v, we construct a DFW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$ with a transition τ such that τ is flexible iff v is not reachable from u. The DFW $\mathcal{A}$ is similar to G, with an additional transition τ from v to a new state. We define $\mathcal{A}$ so that this new transition is flexible iff v is not reachable from u. Intuitively, v is not reachable from u off τ is not reachable from an initial state, which determines τ's flexibility.

Formally, $\mathcal{A} = \langle E \cup \{e_{new}\}, V \cup \{q\}, u, \delta, \{q\}\rangle$. For every edge $e \in E$ such that $e = (w, w')$, we add to $\mathcal{A}$ a transition $\tau' = \langle w, e, w'\rangle$. That is, all the edges of the graph are transitions in the automaton with different letters. We also add to $\mathcal{A}$ the transition $\tau = \langle v, e_{new}, q\rangle$, where $e_{new} \notin E$, and $q \notin V$. Note that $\mathcal{A}$ is a DFW, as required, and that this reduction is computable using logarithmic space. In the full version we prove that indeed τ is flexible iff v is not reachable from u.

For the nondeterministic setting, we show a reduction from the universality problem for NFWs, namely, the problem of deciding, given an NFW $\mathcal{A}$, whether $L(\mathcal{A}) = \Sigma^*$. The reduction is to the flexibility problem for NFWs. Since the universality problem is PSPACE-hard [29], hardness in PSPACE follows.

Given an NFW $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, F\rangle$, we define $\mathcal{A}' = \langle \Sigma', Q', Q'_0, \delta', F'\rangle$ to be similar to $\mathcal{A}$, with an additional component that includes, among others, a transition τ that is going to be flexible iff $L(\mathcal{A}) = \Sigma^*$. Intuitively, if $L(\mathcal{A}) = \Sigma^*$, then the additional component does not contribute to the language of $\mathcal{A}$, and τ is flexible. If, however, $L(\mathcal{A}) \neq \Sigma^*$, there are words that are accepted only using the new component, so τ is not flexible. We assume that if there is a word $w \notin L(\mathcal{A})$, then w is of length at least 1. This assumption does not affect the hardness of the universality problem.

Formally, $\mathcal{A}' = \langle \Sigma \cup \{\sigma_{new}\}, Q \cup \{s_0, s_f\}, Q_0 \cup \{s_0\}, \delta', F \cup \{s_f\}\rangle$. We obtain $\mathcal{A}'$ from $\mathcal{A}$ by adding to $\mathcal{A}$ a transition $\tau = \langle q, \sigma_{new}, q\rangle$, for every accepting state $q \in F$.

Next, we add to $\mathcal{A}'$ an additional component with two states: s_0 and s_f (see Figure 7). The state s_0 is added to the set of initial states, and the state s_f is added to the set of accepting states. For each $\sigma \in \Sigma$, we add a transition $\langle s_0, \sigma, s_0 \rangle$, and a transition $\langle s_0, \sigma, s_f \rangle$. Finally, we add a transition $\tau = \langle s_f, \sigma_{new}, s_f \rangle$. Note that the reduction is computable using logarithmic space. In the full version we prove that indeed $\tau = \langle s_f, \sigma_{new}, s_f \rangle$ is flexible iff $L(\mathcal{A}) = \Sigma^*$.

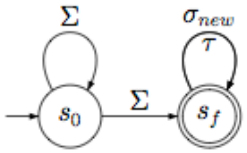

Fig. 7. The new component added to $\mathcal{A}'$.

Theorem 2. *The $\forall$-FLEXIBILITY problem is NLOGSPACE-complete for LDFWs and is PSPACE-complete for LNFWs.*

Proof. We start with the upper bounds. As shown in Lemma 2, in order to check whether a transition τ is $\forall$-flexible, it is sufficient to perform a single containment check: $L(\mathcal{A}_{\tau \leftarrow \top}) \leq L(\mathcal{A}_{\tau \leftarrow \bot})$. The language-containment problem is in NLOGSPACE and PSPACE, for LDFWs and LNFWs, respectively [27], implying the required upper bounds.

Now, since flexibility in DFWs and NFWs corresponds to universal flexibility in LDFWs and LNFWs, respectively, the lower bounds follow from Theorem 1.

Theorem 3. *The $\exists$-FLEXIBILITY problem is NLOGSPACE-complete for LDFWs and is PSPACE-complete for LNFWs.*

Proof. For the upper bounds, consider an LNFW $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F \rangle$, and let τ be a transition in $\mathcal{A}$ with $\delta(\tau) = v$. By Proposition 6, the transition τ is $\exists$-flexible iff τ is v'-tolerant for some neighbor v' of v in the graph of the lattice $\mathcal{L}$; that is, $L(\mathcal{A}) = L(\mathcal{A}_{\tau \leftarrow v'})$. By Proposition 2, for a value $v' \in \mathcal{L}$ such that $v' > v$ we have $L(\mathcal{A}) \leq L(\mathcal{A}_{\tau \leftarrow v'})$, and for a value $v' \in \mathcal{L}$ such that $v' < v$, we have $L(\mathcal{A}_{\tau \leftarrow v'}) \leq L(\mathcal{A})$. Hence, it is sufficient to check for every neighbor v' of v such that $v' > v$, if $L(\mathcal{A}_{\tau \leftarrow v'}) \leq L(\mathcal{A})$ holds, and for every neighbor v' of v such that $v' < v$ or if $L(\mathcal{A}) \leq L(\mathcal{A}_{\tau \leftarrow v'})$ holds. We get that τ is $\exists$-flexible iff there exists a neighbor of v for which the corresponding inequality holds. The upper bound now follows from the known NLOGSPACE and PSPACE complexities of the language-containment problem, for LDFWs and LNFWs, respectively.

Finally, since flexibility in DFWs and NFWs corresponds to existential flexibility in LDFWs and LNFWs, respectively, the lower bounds follow from Theorem 1.

By Theorems 2 and 3, the complexity of the flexibility problems coincide for full-order and partial-order lattice automata. The difference between the two settings is reflected in the time-complexity analysis of the algorithms we described. Given an LNFW

$\mathcal{A}$ over a lattice $\mathcal{L}$, let $n = |\mathcal{A}|, m = |\mathcal{L}|$, and $k = |JI|$. Precisely, we have the following.

Theorem 4. *The $\forall$-FLEXIBILITY problem can be decided in time $O(n(m+n))$ for full-order LDFWs, in time $O(kn(m+n))$ for partial-order LDFWs, and in time $O(k(nm+2^{O(n)}))$ for LNFWs.*

Theorem 5. *The $\exists$-FLEXIBILITY problem can be decided in time $O(n(m+n))$ for full-order LDFWs, in time $O(rkn(m+n))$ for partial-order LDFWs, in time $O(k(nm+2^{O(n)}))$ for full-order LNFWs, and in time $O(rk(nm+2^{O(n)}))$ for partial-order LNFWs, where r is the number of the neighbors of $\delta(\tau)$ in the graph of $\mathcal{L}$.*

Remark 4. In practice, systems and specifications are sometimes underspecified as the designer intentionally does not care about some values in some configurations [22]. Our algorithms can be easily changed to handle settings in which a transition can get the value $\varnothing$ (dont care) or get a set of possible values. In this case, flexibility gets additional significance, as we can assume that the value of transitions for which the designer did bother to specify a value is important. For example, if a transition has a value different than $\varnothing$, and it turns out to be $\forall$-flexible, we can assume that there is an error in the modeling of the specification, since this transition could have also gotten the value $\varnothing$. □

6 Inherent Vacuity with Analogy to Temporal Logic

In [17], the authors introduce two different definitions of inherent vacuity for LTL formulas and prove that they coincide. Consider an LTL formula φ. We say that a subformula ψ of φ does not affect the satisfaction of φ in $\mathcal{S}$ if $\mathcal{S}$ also satisfies the formula $\forall x.\varphi[\psi \leftarrow x]$. We refer to the formula $\forall x.\varphi[\psi \leftarrow x]$ as the ψ-strengthening of φ. Also, we say that a formula φ is vacuously satisfied in $\mathcal{S}$ if φ has a subformula that does not affect its satisfaction in $\mathcal{S}$ [4].

We can now describe the two different definitions of inherent vacuity for LTL formulas from [17]. According to the first definition, an LTL formula φ is inherently vacuous (by mutation) if there exists a subformula ψ of φ such that $\varphi \equiv \forall x.\varphi[\psi \leftarrow x]$. That is, φ is equivalent to its ψ-strengthening. As opposed to the first definition, the second one does not restrict attention to a single subformula. According to the second definition, an LTL formula φ is inherently vacuous (by reference) if for every system $\mathcal{S}$, if $\mathcal{S} \models \varphi$, then $\mathcal{S}$ satisfies φ vacuously. In this section we introduce two different definitions of inherent vacuity for lattice automata, analogous to the definitions in [17], and show that they coincide as well.

Given two LNFWs $\mathcal{A}$ and $\mathcal{A}'$ such that $L(\mathcal{A}') \leq L(\mathcal{A})$, we say that a transition τ in $\mathcal{A}$ *does not affect the containment of $L(\mathcal{A}')$ in $L(\mathcal{A})$*, if for every $v \in \mathcal{L}$ it holds that $L(\mathcal{A}') \leq L(\mathcal{A}_{\tau \leftarrow v})$. Note that this requirement applies to every value in $\mathcal{L}$. Also, $\mathcal{A}'$ is *vacuously contained* in $\mathcal{A}$ if there is a transition τ in $\mathcal{A}$ that does not affect the containment of $L(\mathcal{A}')$ in $L(\mathcal{A})$.

Definition 1. *An LNFW $\mathcal{A}$ is* inherently vacuous by mutation *if there exist a transition τ in $\mathcal{A}$ that is $\forall$-flexible. We then say that $\mathcal{A}$ is inherently vacuous by mutation with witness τ.*

Definition 2. *An LNFW $\mathcal{A}$ is* inherently vacuous by reference *if for every LNFW $\mathcal{A}'$, if $L(\mathcal{A}') \leq L(\mathcal{A})$, then $\mathcal{A}'$ is vacuously contained in $\mathcal{A}$.*

Theorem 6. *An LNFW $\mathcal{A}$ is inherently vacuous by mutation iff $\mathcal{A}$ is inherently vacuous by reference.*

Proof. For the first direction, assume that $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F \rangle$ is inherently vacuous by mutation. Then, there is a transition τ in $\mathcal{A}$ that is $\forall$-flexible, that is, for every $v \in \mathcal{L}$ it holds that $L(\mathcal{A}) = L(\mathcal{A}_{\tau \leftarrow v})$. Accordingly, for every LNFW $\mathcal{A}'$, if $L(\mathcal{A}') \leq L(\mathcal{A})$, then for every $v \in \mathcal{L}$ we have that $L(\mathcal{A}') \leq L(\mathcal{A}_{\tau \leftarrow v})$, and so $\mathcal{A}'$ is vacuously contained in $\mathcal{A}$. Thus, $\mathcal{A}$ is inherently vacuous by reference.

For the second direction, assume that $\mathcal{A}$ is inherently vacuous by reference, and assume, by way of contradiction, that $\mathcal{A}$ is not inherently vacuous by mutation. Then, there exist no transition τ in $\mathcal{A}$ that is $\forall$-flexible. It is not hard to prove that then, there is no transition τ in $\mathcal{A}$ such that for all LNFWs $\mathcal{A}'$ with $L(\mathcal{A}') \leq L(\mathcal{A})$, the transition τ does not affect the containment of $L(\mathcal{A}')$ in $L(\mathcal{A})$. Indeed, since $L(\mathcal{A}) \leq L(\mathcal{A})$, the existence of such a transition would have implied universal flexibility of τ.

Let k be the number of transitions in $\mathcal{A}$. By the assumption, for every candidate transition τ_i, with $1 \leq i \leq k$, there is an LNFW $\mathcal{A}_i = \langle \mathcal{L}, \Sigma, Q_i, Q_i^0, \delta_i, F_i \rangle$ such that $L(\mathcal{A}_i) \leq L(\mathcal{A})$ but there is a value $v \in \mathcal{L}$ such that $L(\mathcal{A}_i) \not\leq L(\mathcal{A}_{\tau_i \leftarrow v})$. Without loss of generality, we assume that the state spaces Q_i are pairwise disjoint. Let $\mathcal{A}'$ be the LNFW obtained by "putting all the LNFWs $\mathcal{A}_i$ next to each other". Formally, $\mathcal{A}' = \langle \mathcal{L}, \Sigma, \bigcup\{Q_i\}_{1 \leq i \leq k}, \bigcup\{Q_i^0\}_{1 \leq i \leq k}, \bigcup\{\delta_i\}_{1 \leq i \leq k}, \bigcup\{F_i\}_{1 \leq i \leq k} \rangle$. Note that, naturally, $L(\mathcal{A}') = \bigvee_{1 \leq i \leq k} L(\mathcal{A}_i)$. Since $L(\mathcal{A}_i) \leq L(\mathcal{A})$ for every $1 \leq i \leq k$ we get that $\bigvee_{1 \leq i \leq k} L(\mathcal{A}_i) \leq L(\mathcal{A})$ and thus $L(\mathcal{A}') \leq L(\mathcal{A})$. Now, since $\mathcal{A}$ is inherently vacuous by reference, then $\mathcal{A}'$ is vacuously contained in $\mathcal{A}$. Let τ_i be a transition that does not affect the containment of $L(\mathcal{A}')$ in $L(\mathcal{A})$. Then, for every $v \in \mathcal{L}$ it holds that $L(\mathcal{A}') \leq L(\mathcal{A}_{\tau_i \leftarrow v})$. Since $L(\mathcal{A}_i) \leq L(\mathcal{A}')$, we get that for every $v \in \mathcal{L}$ it holds that $L(\mathcal{A}_i) \leq L(\mathcal{A}_{\tau_i \leftarrow v})$, and so τ_i does not affect the containment of $L(\mathcal{A}_i)$ in $L(\mathcal{A})$, and we have reached a contradiction.

Thus, as in the case of LTL formulas, the two definitions of inherent vacuity coincide.

Remark 5. As discussed in Section 1, lattices and lattice automata have practical applications in formal methods. Some of the applications use the specification formalism *latticed LTL* (LLTL, for short), which extends LTL by mapping computations in which atomic propositions have values from a lattice into lattice values [12]. The translation of LTL into automata [36] has been extended to a translation of LLTL into latticed automata [27]. When applied to the lattice automata obtained from LLTL formulas, vacuity in the automata correspond to vacuity in the formulas. Since changes in subformulas induce changes in transitions from all states of the automaton that are associated with these subformulas, the relevant type of vacuity is the one discussed in Remark 2, namely when the value of a set of transitions is mutated. □

References

1. S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. In *Proc. 40th Int. Colloq. on Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 15 – 27. Springer, 2013.
2. S. Almagor and O. Kupferman. Latticed-ltl synthesis in the presence of noisy inputs. In *Proc. 17th Int. Conf. on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science. Springer, 2014.
3. R. Alur, A. Kanade, and G. Weiss. Ranking automata and games for prioritized requirements. In *Proc. 20th Int. Conf. on Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2008.
4. R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M.Y. Vardi. Enhanced vacuity detection for linear temporal logic. In *Proc. 15th Int. Conf. on Computer Aided Verification*. Springer, 2003.
5. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. *Formal Methods in System Design*, 18(2):141–162, 2001.
6. G. Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.
7. R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. RAT: A tool for the formal analysis of requirements. In *Proc. 19th Int. Conf. on Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 263–267. Springer, 2005.
8. G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proc. 11th Int. Conf. on Computer Aided Verification*, pages 274–287, 1999.
9. G. Bruns and P. Godefroid. Temporal logic query checking. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, pages 409–420. IEEE Computer Society, 2001.
10. D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M.Y. Vardi. Regular vacuity. In *Proc. 13th Conf. on Correct Hardware Design and Verification Methods*, volume 3725 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 2005.
11. W. Chan. Temporal-logic queries. In *Proc. 12th Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 450–463. Springer, 2000.
12. M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using SPIN. In *Proc. 8th Int. SPIN Workshop on Model Checking Software*, volume 2057 of *Lecture Notes in Computer Science*, pages 16–36. Springer, 2001.
13. H. Chockler and O. Strichman. Easier and more informative vacuity checks. In *Proc. 5th International Conference on Formal Methods and Models for Co-Design*, pages 189–198, 2007.
14. M. Ciric, J. Ignjatovic, N. Damljanovic, and M. Basic. Bisimulations for fuzzy automata. *Fuzzy Sets and Systems*, 186(1):100–139, 2012.
15. M. Droste, W. Kuich, and H. Vogler (eds.). *Handbook of Weighted Automata*. Springer, 2009.
16. S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Proc. 23rd Int. Conf. on Software Engineering*, pages 411–420. IEEE Computer Society Press, 2001.
17. D. Fisman, O. Kupferman, S. Seinvald, and M.Y. Vardi. A framework for inherent vacuity. In *4th International Haifa Verification Conference*, volume 5394 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2008.
18. A. Gurfinkel and M. Chechik. Extending extended vacuity. In *Proc. 5th Int. Conf. on Formal Methods in Computer-Aided Design*, volume 3312 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2004.

19. S. Halamish and O. Kupferman. Approximating deterministic lattice automata. In *10th Int. Symp. on Automated Technology for Verification and Analysis*, volume 7561 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2012.
20. S. Halamish and O. Kupferman. Minimizing deterministic lattice automata. *ACM Transactions on Computational Logic*, 16(1):1–21, 2015.
21. T.A. Henzinger. From Boolean to quantitative notions of correctness. In *Proc. 37th ACM Symp. on Principles of Programming Languages*, pages 157–158, 2010.
22. Y. Hoskote, T. Kam, P.-H Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *Proc. 36st Design Automation Conf.*, pages 300–305, 1999.
23. A. Hussain and M. Huth. On model checking multiple hybrid views. Technical Report TR-2004-6, University of Cyprus, 2004.
24. N. Immerman. Nondeterministic space is closed under complement. *Information and Computation*, 17:935–938, 1988.
25. N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and Systems Science*, 11:68–75, 1975.
26. O. Kupferman. Sanity checks in formal verification. In *Proc. 17th Int. Conf. on Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2006.
27. O. Kupferman and Y. Lustig. Lattice automata. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 199 – 213. Springer, 2007.
28. O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *Software Tools for Technology Transfer*, 4(2):224–233, 2003.
29. A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. 13th IEEE Symp. on Switching and Automata Theory*, pages 125–129, 1972.
30. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.
31. K.S. Namjoshi. An efficiently checkable, proof-based formulation of vacuity in model checking. In *Proc. 16th Int. Conf. on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 57–69. Springer, 2004.
32. PROSYD. The Prosyd project on property-based system design. http://www.prosyd.org, 2007.
33. M. Roveri. Novel techniques for property assurance. Technical report, PROSYD FP6-IST-507219, 2007.
34. K.Y. Rozier and M.Y. Vardi. LTL satisfiability checking. In *Proc. 14th International SPIN Workshop*, volume 4595 of *Lecture Notes in Computer Science*, pages 149–167. Springer, 2007.
35. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proc. 9th Int. Conf. on Computer Aided Verification*, volume 1254, pages 72–83. Springer, 1997.
36. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.