# Lattice Automata

Orna Kupferman and Yoad Lustig

Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel
{orna,yoadl}@cs.huji.ac.il

**Abstract.** Several verification methods involve reasoning about multi-valued systems, in which an atomic proposition is interpreted at a state as a lattice element, rather than a Boolean value. The automata-theoretic approach for reasoning about Boolean-valued systems has proven to be very useful and powerful. We develop an automata-theoretic framework for reasoning about multi-valued objects, and describe its application. The basis to our framework are *lattice automata* on finite and infinite words, which assign to each input word a lattice element. We study the expressive power of lattice automata, their closure properties, the blow-up involved in related constructions, and decision problems for them. Our framework and results are different and stronger then those known for semi-ring and weighted automata. Lattice automata exhibit interesting features from a theoretical point of view. In particular, we study the complexity of constructions and decision problems for lattice automata in terms of the size of both the automaton and the underlying lattice. For example, we show that while determinization of lattice automata involves a blow up that depends on the size of the lattice, such a blow up can be avoided when we complement lattice automata. Thus, complementation is easier than determinization. In addition to studying the theoretical aspects of lattice automata, we describe how they can be used for an efficient reasoning about a multi-valued extension of LTL.

## 1 Introduction

Several recent verification methods involve reasoning about *multi-valued Kripke structures* in which an atomic proposition is interpreted at a state as a *lattice* element[1], rather than a Boolean value. The multi-valued setting arises directly in systems in which the designer can give to the atomic propositions rich values like "uninitialized", "unknown", "high impedance", "don't care", "logic 1", "logic 0", and more (c.f., the IEEE Standard Multivalue Logic System for VHDL Model Interoperability [IEEE93]), and arise 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 [GS97, BG99], query checking [Cha00], which can be reduced to model checking over multi-valued Kripke structures, and verification of systems from inconsistent viewpoints [HH04], in which the value of the atomic propositions is the composition of their values in the different viewpoints. The various applications use various types of lattices (see Figure 1). For example, in the abstraction application, researchers have used three

[1] 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 $b$).

values ordered as in $\mathcal{L}_3$ [BG99], as well as its generalization to linear orders [CDG01]. In query checking, the lattice elements are sets of formulas, ordered by the inclusion order [BG01]. When reasoning about inconsistent viewpoints, each viewpoint is Boolean, and their composition gives rise to products of the Boolean lattice, as in $\mathcal{L}_{2,2}$ [EC01]. Finally, in systems with rich values of the atomic propositions, several orders may be used with respect to the various values, which in fact do not always induce a lattice.

The *automata-theoretic* approach uses the theory of automata as a unifying paradigm for system specification, verification, and synthesis [Kur94, VW94, KVW00]. Automata enable the separation of the logical and the algorithmic aspects of reasoning about systems, yielding clean and asymptotically optimal algorithms. The automata-theoretic framework for reasoning about Boolean-valued systems has proven to be very versatile. Automata are the key to techniques such as on-the-fly verification, and they are useful also for modular verification, partial-order verification, verification of real-time and hybrid systems, open systems, and infinite-state systems. Many decision and synthesis problems have automata-based solutions and no other solution for them is known. Automata-based methods have been implemented in both academic and industrial automated-verification tools (c.f., COSPAN and SPIN).

In this work, we describe an automata-theoretic framework for reasoning about multi-valued objects. Consider a lattice $\mathcal{L}$. For a set $X$ of elements, an *$\mathcal{L}$-set over $X$* is a function $S : X \rightarrow \mathcal{L}$ assigning to each element of $X$ a value in $\mathcal{L}$. For an alphabet $\Sigma$, an $\mathcal{L}$-language is a function $L : \Sigma^* \rightarrow \mathcal{L}$ that gives a value in $\mathcal{L}$ to each word over $\Sigma$. A *nondeterministic lattice automaton on finite words* (*LNFW*, for short) gets as input words over $\Sigma$ and assigns to each word a value in $\mathcal{L}$. Thus, each LNFW defines an $\mathcal{L}$-language. Technically, in an LNFW $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F \rangle$, the sets of initial and final states are $\mathcal{L}$-sets over $Q$ (i.e., $Q_0, F \in \mathcal{L}^Q$ describe the "initial value" and the "acceptance value" of each state), and $\delta$ is an $\mathcal{L}$-set over $Q \times \Sigma \times Q$ (i.e., $\delta \in \mathcal{L}^{Q \times \Sigma \times Q}$ describes the "traversal value" of each labeled transition). Then, the value of a run of $\mathcal{A}$ is the *meet* of values of the components of the run (that is, the initial value of the first state, the traversal values of the transitions that have been taken, and the acceptance value of the last state), and the value that $\mathcal{A}$ assigns to a word is the *join* of the values of the runs of $\mathcal{A}$ on $w$.

The definition of LNFW is not too surprising, and, as we mention in the sequel, it is similar to previous definitions of "weighted automata". Things, however, become very interesting when one starts to study properties of LNFWs. Essentially, in the Boolean setting, the only important piece of information about a run is the membership of its last state in the set of accepting states. In the lattice setting, on the other hand, all the components of the run are important. To see the computational challenges that the lattice setting involves, consider for example the simple property of closure under join for deterministic lattice automata (LDFW, for short, where only a single initial/successor state is possible (has a value different from $\perp$)). Stating that LDFW are closed under join, one has to construct, given two LDFWs $\mathcal{A}_1$ and $\mathcal{A}_2$, an LDFW $\mathcal{A}$ such that for every word $w$, the value of $\mathcal{A}$ on $w$ is the join of the values of $\mathcal{A}_1$ and $\mathcal{A}_2$ on $w$. In the traditional Boolean setting, join corresponds to union, and it is easy to construct $\mathcal{A}$ as the product of $\mathcal{A}_1$ and $\mathcal{A}_2$. In the lattice setting, however, it is not clear how to define the traversal value of the transitions of $\mathcal{A}$ based on the traversal value of the transitions of $\mathcal{A}_1$ and $\mathcal{A}_2$. We show that, indeed, the product construction cannot work, and the LDFW $\mathcal{A}$ must contain in its state space a component that depends on $\mathcal{L}$. Dependency in $\mathcal{L}$ cannot be avoided also when we determinize LNFWs: every LNFW $\mathcal{A}$ has an equivalent LDFW $\mathcal{A}'$. Nevertheless, while in the traditional Boolean case the construction of $\mathcal{A}'$ involves the subset construction [RS59] and for $\mathcal{A}$ with $n$ states we get $\mathcal{A}'$ with $2^n$ states, here the subset construction looses information such as the traversal value with which each state in the set has been reached, and we show a tight $m^n$ bound on the size of $\mathcal{A}'$, where $m = |\mathcal{L}|$.

Of special interest is the complementation problem[2] for LNFW. In the Boolean setting, it is easy to complement deterministic automata, and complementation of nondeterministic automata involves determinization. In the lattice setting, determinization involves an $m^n$ blow up, and moreover, complementation involves an $nm$ blow up even if we start with a deterministic automaton. Interestingly, by adopting ideas from the theory of automata on infinite words [KV01][3], we are able to avoid determinization, avoid the dependency in $m$, and complement LNFW with a $2^n$ blow up only. For this purpose we define universal lattice automata (LUFW, for short), which dualize LNFW, show that complementation can be done by dualization, and that LUFW can be translated to LNFW with a $2^n$ blow up[4].

Once we prove closure properties, we proceed to study the fundamental decision problems for the new framework: the *emptiness-value* and the *universality-value* problems, which corresponds to the emptiness and universality problems in the Boolean setting and decide, given $\mathcal{A}$, how likely it is (formalized by means of values in $\mathcal{L}$) for $\mathcal{A}$ to accept some word or all words; and the *implication-value problem*, which corresponds to the language-inclusion problem and decides, given two LNFWs $\mathcal{A}_1$ and $\mathcal{A}_2$, how likely it is that membership in the language of $\mathcal{A}_1$ implies membership in the language of $\mathcal{A}_2$. We show that, using the tight constructions described earlier, the problems have the same complexities as the corresponding problems in the Boolean setting.

We then turn to applications of LNFW for reasoning about multi-valued temporal logics and systems. We define the logic *Lattice LTL* (*LLTL*, for short), where the constants can take lattice values, and whose semantics is defined with respect to multi-valued Kripke-structures. We extend LNFW to the framework of automata on infinite words, define *nondeterministic lattice Büchi word automata* (*LNBW*, for short), and show that known translations of LTL to nondeterministic Büchi word automata [VW94] can be lifted to the lattice setting. Then, we use LNBW to solve the satisfiability and model-checking problems for LLTL, and show that both problems are PSPACE–complete — not harder than in the Boolean setting. In addition, we study some basic theory of lattice automata on infinite words. In particular, we show that the comple-

[2] Discussing complementation, we restrict attention to *De Morgan lattices*, where complementation inside the lattice is well defined (See Section 2.1).

[3] As we discuss in the paper, there are several common computational aspects of LNFW and automata on infinite words, as reasoning in both theories has to cope with the fact that the outcome of a run depends on its on-going behavior, rather than its last state only.

[4] We note that the latter construction is not trivial; it has the flavor of the construction in [MH84] for the case of infinite words, but unlike [MH84] (or the much simpler Boolean case), the result LNFW is nondeterministic; if one seeks an equivalent LDFW, a dependency in $m$ cannot be avoided.

mentation construction of [KV01] can be combined with the ideas we use in the case of LNFW complementation, thus LNBW complementation involves a $2^{O(n \log n)}$ blow up and is independent of $m$.

*Related Work.* We are aware of two previous definitions of automata over lattices and their applications in verification. Our framework, however, is the first to study the theoretical aspects of lattice automata, rather than only use them. Also, the applications we suggest go beyond these that are known. Below we discuss the two known definitions and compare them with our contribution. In [BG01], Bruns and Godefroid introduce *Extended Alternating Automata* (EAA, for short). EAA extend the automata-theoretic approach to branching-time model checking [KVW00], they run on trees, and map each input tree to a lattice value. EAA have been used for query checking [BG01] and model checking multi-valued $\mu$-calculus [BG04]. EAA are incomparable with the model we study here. On the one hand, EAA are more general, as they run on trees and are alternating. On the other hand, they are not making full use of the lattice framework, as their "lattice aspect" is limited to the transition function having lattice values in its range.

Also, the application of reasoning about LLTL properties, which we describe here, cannot be achieved with EAA, as it involves a doubly-exponential translation of LLTL to $\mu$-calculus, which we avoid. In [CDG01], Chechik, Devereux, and Gurfinkel define *multiple-valued Büchi automata* ($\mathcal{X}$Büchi automata, for short) and use them for model checking multiple-valued LTL. Like LNFW, each transition in a $\mathcal{X}$Büchi automata has a traversal value and the automata define $\mathcal{L}$-languages. Unlike LNFW, $\mathcal{X}$Büchi automata (and the multiple-valued LTL that correspond to them) are restricted to lattices that are finite linear orders. Thus, the setting and its potential applications is weaker.

In addition to lattice-based multi-valued logics, other related concepts were investigated. Lattice-based automata (for distributive lattices) can be seen as a special case of *weighted automata* [Moh97], which are in turn a special case of *semiring automata* [KS86]. Semiring automata is a very general algebraic notion of automata in which computations get values from some semiring. However, the model of semiring automata is algebraic in nature and is relatively far from the standard notion of finite automata. Weighted automata is another notion in which computations get values from a semiring, one that closely resembles the standard model of finite automata. In fact, since a distributive lattice is a semiring in which $\oplus$ is a join and $\otimes$ is a meet, the definitions of lattice automata are a special case of the definitions of weighted automata. However, while (distributive) lattices are semirings, lattices share some properties that general semirings do not. Specifically, the idempotent laws (i.e., $a \vee a = a$ and $a \wedge a = a$) as well as the absorption laws (i.e., $a \vee (a \wedge b) = a$ and $a \wedge (a \vee b) = a$), which are very intuitive in a logical context, do not hold in a general semiring, and do hold for lattices. Furthermore, the complementation operand that is essential for choosing lattices as a framework for multi-valued reasoning, has no natural interpretation in a general semiring. Finally, our results here go beyond these that are known for semiring automata. In particular, we consider also automata on infinite words, both nondeterministic and universal automata, and we study the computational aspects of constructions and decision problems.

Due to space limitations, proofs are omitted and can be found in the full version at the authors web pages.

## 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 *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 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 $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.

The traditional disjunction and conjunction logic operators correspond to the join and meet lattice operators. In a general lattice, however, there is no natural counterpart to negation. A *De Morgan* (or *quasi-Boolean*) lattice is a lattice in which every element $a$ has a unique complement element $\neg a$ such that $\neg\neg a = a$, De Morgan rules hold, and $a \leq b$ implies $\neg b \leq \neg a$. In the rest of the paper we consider only finite[5] distributive De Morgan lattices.

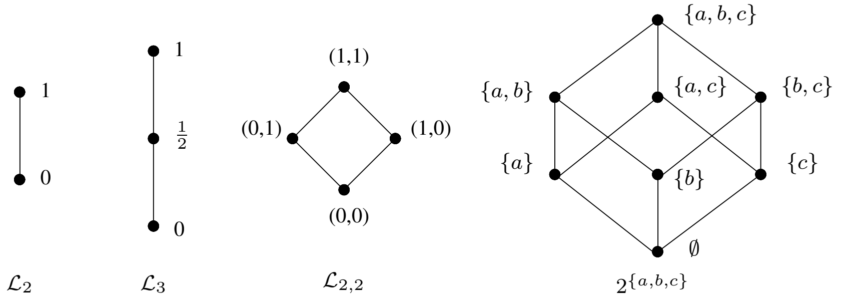

**Fig. 1.** Some lattices

In Figure 1 we describe some (finite distributive De Morgan) 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 $\mathcal{L}_3$ contains in addition the value $\frac{1}{2}$, with the order $0 \leq \frac{1}{2} \leq 1$, and with negation defined by $\neg 0 = 1$ and $\neg\frac{1}{2} = \frac{1}{2}$. 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 $b \leq b'$. Also, $\neg(a, b) = (\neg a, \neg b)$. Finally, the lattice $2^{\{a,b,c\}}$ is the power set of $\{a, b, c\}$ with the set-inclusion

[5] Note that focusing on finite lattices is not as restrictive as may first seem. Indeed, even when the lattice is infinite, the problems we consider involve only finite Kripke structures, formulas, and automata. Therefore, only a finite number of lattice elements appear in a problem, and since the lattice is distributive, the logical operations closure of these values is still finite.

order. Complementation is interpreted as set complementation relative to $\{a, b, c\}$. In this lattice, for example, $\{a\} \vee \{b\} = \{a, b\}$, $\{a\} \wedge \{b\} = \bot$, $\{a, c\} \vee \{b\} = \top$, and $\{a, c\} \wedge \{b\} = \bot$.

A *join irreducible* element $l \in \mathcal{L}$ is a value, other then $\bot$, for which if $l_1 \vee l_2 \geq l$ then either $l_1 \geq l$ or $l_2 \geq l$. By Birkhoff's representation theorem for finite distributive lattices in order to prove that $l_1 = l_2$ it is sufficient if to prove that for every join irreducible element $l$ it holds that $l_1 \geq l$ iff $l_2 \geq l$. We denote the set of join irreducible elements of $\mathcal{L}$ by $JI(\mathcal{L})$. A *meet irreducible* element $l \in \mathcal{L}$ is a value for which if $l_1 \wedge l_2 \leq l$ then either $l_1 \leq l$ or $l_2 \leq l$. Note that in a De Morgan lattice an element is meet irreducible iff its complement is join irreducible. We denote the set of meet irreducible elements of $\mathcal{L}$ by $MI(\mathcal{L})$.

Consider a lattice $\mathcal{L}$ (we abuse notation and refer to $\mathcal{L}$ also as a set of elements, rather than a pair of a set with an order on it). 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 $x \in X$. The usual set operators can be lifted to $\mathcal{L}$-sets as expected. Given two $\mathcal{L}$-sets $S_1$ and $S_2$ over $X$, we define join, meet, and complementation so that for every element $x \in X$, we have [6] $S_1 \vee S_2(x) = S_1(x) \vee S_2(x)$, $S_1 \wedge S_2(x) = S_1(x) \wedge S_2(x)$, and $comp(S_1)(x) = \neg S_1(x)$.

## 2.2 Lattice Automata

Consider a lattice $\mathcal{L}$ and an alphabet $\Sigma$. An *$\mathcal{L}$-language* is an $\mathcal{L}$-set over $\Sigma^*$. Thus an $\mathcal{L}$-language $L : \Sigma^* \to \mathcal{L}$ assigns a value in $\mathcal{L}$ to each word over $\Sigma$. A *nondeterministic lattice automaton* on finite words (LNFW, for short) is a six-tuple $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F \rangle$, where $\mathcal{L}$ is a lattice, $\Sigma$ 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 $\mathcal{L}$-transition-relation, and $F \in \mathcal{L}^Q$ is an $\mathcal{L}$-set of accepting states.

A *run* of an LNFW on a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$ is a sequence $r = q_0, \ldots, q_n$ of $n+1$ states. The *value* of $r$ on $w$ is $val(r, w) = Q_0(q_0) \wedge \bigwedge_{i=0}^{n-1} \delta(q_i, \sigma_{i+1}, q_{i+1}) \wedge F(q_n)$. Intuitively, $Q_0(q_0)$ is the value of $q_0$ being initial, $\delta((q_i, \sigma_{i+1}, q_{i+1}))$ is the value of $q_{i+1}$ being a successor of $q_i$ when $\sigma_{i+1}$ 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, with $0 \leq i \leq n-1$. We refer to $Q_0(q_0) \wedge \bigwedge_{i=0}^{n-1} \delta(q_i, \sigma_{i+1}, q_{i+1})$ 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, $val(\mathcal{A}, w) = \bigvee \{val(r, w) :$ $r$ is a run of $\mathcal{A}$ 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, $L(\mathcal{A})(w) = val(\mathcal{A}, w)$.

An LNFW is a *deterministic* lattice automaton on finite words (LDFW, for short) if there is exactly one state $q \in Q$ such that $Q_0(q) \neq \bot$, and for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is exactly one state $q' \in Q$ such that $\delta(q, \sigma, q') \neq \bot$. An LNFW is *simple* if $Q_0$ and $\delta$ are Boolean. Note that the traversal value of a run $r$ of a simple LNFW is either $\bot$ or $\top$, thus the value of $r$ is induced by $F$.

[6] If $S_1$ and $S_2$ are over different domains $X_1$ and $X_2$, we can view them as having the same domain $X_1 \cup X_2$ and let $S_1(x) = \bot$ for $x \in X_2 \setminus X_1$ and $S_2(x) = \bot$ for $x \in X_1 \setminus X_2$.

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 $r$ on a word $w$ 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.

**Example 1.** Figure 2 depicts three LNFWs. When we draw an LNFW, we denote the fact that $\delta(q, \sigma, q') = l$ by an edge attributed by $(\sigma, l)$ from $q$ to $q'$. For simplicity, we sometimes label an edge with a set $S \subseteq \Sigma \times L$. In particular, when $\Sigma = \mathcal{L}$, we use $(l, \top)$ to denote the set $\{(l, \top) : l \in \mathcal{L}\}$ and we use $(l, l)$ to denote the set $\{(l, l) : l \in \mathcal{L}\}$. For states $q$ with $Q_0(q) = l \neq \bot$, we draw into $q$ an edge labeled $l$, and for states $q$ with $F(q) = l \neq \bot$, we draw $q$ as a double circle labeled $l$. For example, the LNFW $\mathcal{A}_2 = \langle \mathcal{L}, \mathcal{L}, \{q_1, q_2\}, Q_0, \delta, F\rangle$ is such that $Q_0(q_1) = \top$ and $Q_0(q_2) = \bot$. Also, for every $l \in \mathcal{L}$, we have $\delta(q_1, l, q_1) = \delta(q_2, l, q_2) = \top$, and $\delta(q_1, l, q_2) = l$. All other triplets $\langle q, l, q\rangle \in Q \times \mathcal{L} \times Q$ are mapped by $\delta$ to $\bot$. Finally, $F(q_1) = \bot$ and $F(q_2) = \top$.

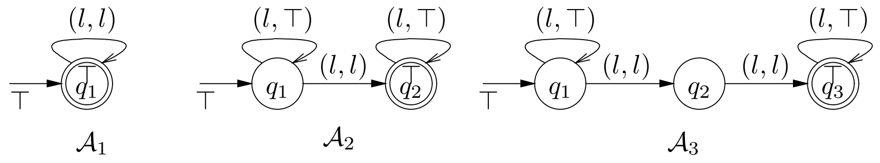

**Fig. 2.** Three LNFWs

Let us consider the $\mathcal{L}$-languages of the LNFWs in Figure 2. The LNFW $\mathcal{A}_1$ is deterministic. Its single run $r$ a word $w = l_1 \cdot l_2 \cdots l_n$ starts in $q_1$ with value $\top$ and whenever the letter $l_i$ is read, the traversal value so far is met with $l_i$. The acceptance value of $r$ is $\top$, thus the value of $r$ on $w$ is $\bigwedge_{i=1}^{n} l_i$. Hence, the language $L_1$ of $\mathcal{A}_1$ is such that $L_1(l_1 \cdot l_2 \cdots l_n) = \bigwedge_{i=1}^{n} l_i$. The LNFW $\mathcal{A}_2$ is nondeterministic. Reading a word $w = l_1 \cdot l_2 \cdots l_n$, it guesses a letter $l_i$ with which the transition from $q_1$ to $q_2$ is made. Since the values of the self loops in $q_1$ and $q_2$ are $\top$ and so are the initial and acceptance values, the value of such a run on $w$ is $l_i$. Taking the join on all runs, we get that the language $L_2$ of $\mathcal{A}_2$ is such that $L_2(l_1 \cdot l_2 \cdots l_n) = \bigvee_{i=1}^{n} l_i$. Finally, the LNFW $\mathcal{A}_3$ is also nondeterministic. Here, going from $q_1$ to $q_3$ two successive letters are read, each contributing its value to the traversal value of the run. Hence the language $L_3$ of $\mathcal{A}_3$ is such that $L_3(l_1 \cdot l_2 \cdots l_n) = \bigvee_{i=1}^{n-1} (l_i \wedge l_{i+1})$. $\square$

In the traditional Boolean setting, a *universal* automaton (UFW, for short) accepts a word $w$ if all its runs on $w$ are accepting. Lifting this definition to the lattice framework, a universal lattice automaton (LUFW, for short) has the same components as an LNFW, only that the value of a run $r = q_0 \ldots q_n$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$ is $val(r, w) = comp(Q_0(q_0)) \vee \bigvee_{i=0}^{n-1} comp(\delta(q_i, \sigma_{i+1}, q_{i+1})) \vee comp(F(q_n))$, and the value of $\mathcal{A}$ on $w$ is $val(\mathcal{A}, w) = \bigwedge\{val(r, w) :\ r \text{ is a run of } \mathcal{A} \text{ on } w\}$. Thus, LUFW dualize LNFW in the three elements that determine the value of an automaton on a run:

first, the way we refer to the components of a single run is disjunctive (rather than conjunctive). Second, the way we refer to the collection of runs is conjunctive (rather than disjunctive). Finally, the initial values, transition values, and acceptance values are all complemented.

**Example 2.** Consider the three LNFWs discussed in Example 1. When we view them as LUFW, their languages $\tilde{L_1}, \tilde{L_2}$, and $\tilde{L_3}$ are such that $\tilde{L_1}(l_1 \cdot l_2 \cdots l_n) = \bigvee_{i=1}^{n} comp(l_i)$, $\tilde{L_2}(l_1 \cdot l_2 \cdots l_n) = \bigwedge_{i=1}^{n} comp(l_i)$, and $\tilde{L_3}(l_1 \cdot l_2 \cdots l_n) = \bigwedge_{i=1}^{n-1}(comp(l_i) \vee comp(l_{i+1}))$. □

**Remark 3.** In many applications, the input words to the LNFW are generated by a graph in which each vertex is labeled by a letter in $\Sigma$. In some applications, the transition relation of the graph is an $\mathcal{L}$-set, thus each edge has a value in $\mathcal{L}$. Accordingly, in a more general framework, each letter in $\Sigma$ has a weight — a value in $\mathcal{L}$ that corresponds to the value of the edge between the current and next vertices. Then, the value of a run of the automaton over a weighted word $w = \langle \sigma_1, l_1 \rangle \cdot \langle \sigma_2, l_2 \rangle \cdots \langle \sigma_n, l_n \rangle$ takes the weights of the letters into account: when we are in state $q_i$, read a letter $\langle \sigma_{i+1}, l_{i+1} \rangle$, and move to state $q_{i+1}$, the contribution to the value of the run is $l_{i+1} \wedge \delta(q_i, \sigma_{i+1}, q_{i+1})$ (rather than $\delta(q_i, \sigma_{i+1}, q_{i+1})$ only). Since the lattice is distributive, it is easy to see that the value of such an LNFW over the word $w$ is equal to the meet of its value on $\langle \sigma_1, \top \rangle \cdot \langle \sigma_2, \top \rangle \cdots \langle \sigma_n, \top \rangle$ with $\bigwedge_{1 \leq i \leq n} l_i$. Thanks to this decompositionality, it is easy to adjust our framework to automata that read words with weighted letters. For technical simplicity, we assume no weights. □

**Remark 4.** It is interesting to compare LNFW's to EAA's as defined in [BG04]. (Formally, EAA are defined only for infinite trees but it is easy to accommodate them to finite words). In EAA, there is no explicit concept of transition value. Since, however, EAA are alternating, it is possible to model a transition into state $q$ with value $l$ by the formula $q \wedge l$. By taking the meet of a transition with a lattice value, it is possible to ensure that in all runs, the value attached to the *source* vertex of the transition is at most $l$. Intuitively, the value of an EAA run flows "upwards" while the value of an LNFW run flows "downwards". An interesting outcome of this observation is that while it is natural to define the value of a prefix of a run of an LNFW, an LNFW run, it does not seem possible to define the value of a prefix of an EAA run. We find the ability to refer to this value helpful both in understanding the intuition behind the runs of automata and in reasoning about them — as we will demonstrate in Section 3. □

## 3 Closure Properties

In this section we study closure properties of LNFW and LDFW. We show that LNFW and LDFW are closed under join, meet, and complementation, show that LNFW can be determinized and simplified, and analyze the blow-up that the various constructions operators involve. In addition to the dependency in the size $n$ of the original automaton (or automata, in case of the join and meet operators), our analysis refers to the size $m$ of the lattice over which the automata are defined. The dependence on both $n$ and $m$ is tight and the proofs in full version provide both upper bounds and lower bounds.

## 3.1 Nondeterministic Automata on Finite Words

**Theorem 5 [closure under join and meet].** *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be LNFW over $\mathcal{L}$, with $n_1$ and $n_2$ states, respectively. There are LNFW $\mathcal{A}_\vee$ and $\mathcal{A}_\wedge$, with $n_1 + n_2$ and $n_1 \cdot n_2$ states, respectively, such that $L(\mathcal{A}_\vee) = L(\mathcal{A}_1) \vee L(\mathcal{A}_2)$ and $L(\mathcal{A}_\wedge) = L(\mathcal{A}_1) \wedge L(\mathcal{A}_2)$.*

The constructions are slight variants of the standard Boolean case constructions.

**Theorem 6 [simplification].** *Let $\mathcal{A}$ be an LNFW (LDFW) with $n$ states, over a lattice $\mathcal{L}$ with $m$ elements. There is a simple LNFW (resp. LDFW) $\mathcal{A}'$, with $n \cdot m$ states, such that $L(\mathcal{A}') = L(\mathcal{A})$.*

Intuitively, the state space of $\mathcal{A}'$ is $Q \times \mathcal{L}$, where $\langle q, l \rangle$ stands for state $q$ with value $l$.

We now turn to consider determinization of LNFW. For simple LNFW, determinization can proceed using the subset construction as in the Boolean case [RS59]. If we start with a general LNFW $\mathcal{A}$ with state space $Q$, this results in an LDFW $\mathcal{A}'$ with state space $2^{Q \times \mathcal{L}}$. As Theorem 7 below shows, LNFW determinization does depend on $\mathcal{L}$, but we can do better than maintaining subsets of $Q \times \mathcal{L}$. The idea is to maintain, instead, functions in $\mathcal{L}^Q$, where each state $q$ of $\mathcal{A}$ is mapped to the join of the values with which $\mathcal{A}$ might have reached $q$. Note that the resulting automaton is a simple LDFW.

**Theorem 7 [determinization].** *Let $\mathcal{A}$ be an LNFW with $n$ states, over a lattice $\mathcal{L}$ with $m$ elements. There is a simple LDFW $\mathcal{A}'$, with $m^n$ states, such that $L(\mathcal{A}') = L(\mathcal{A})$.*

We now turn to study complementation on LNFW. As with traditional automata, it is possible to complement an automaton through determinization. Starting with an LNFW with $n$ states over a lattice with $m$ elements, we can construct, by Theorem 7, a simple LDFW which can be easily complemented to LNFW with $m^n$ states. We now show that by using universal automata, it is possible to circumvent determinization and avoid the dependency on $m$. We first observe that viewing an LNFW as an LUFW complements its language. The proof is easy and is based on applying De Morgan rules on $val(\mathcal{A}, w)$.

**Lemma 1.** *Let $\mathcal{A}$ be an LNFW and let $\tilde{\mathcal{A}}$ be $\mathcal{A}$ when viewed as an LUFW. Then, $L(\tilde{\mathcal{A}}) = comp(L(\mathcal{A}))$.*

**Theorem 8.** *Let $\mathcal{A}$ be an LUFW, with $n$ states. There is an LNFW $\mathcal{A}'$, with $2^n$ states, such that $L(\mathcal{A}') = L(\mathcal{A})$.*

The intuition being the proof of Theorem 8 is as follows. Let $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F \rangle$. Consider a word $w = \sigma_1 \cdots \sigma_n$. The runs of $\mathcal{A}$ on $w$ can be arranged in a directed acyclic graph $G = \langle Q \times \{0, \ldots, n\}, E \rangle$, where $E(\langle q, i-1 \rangle, \langle q', i \rangle)$ for all $q, q' \in Q$ and $1 \leq i \leq n$. Each edge $\langle \langle q, i-1 \rangle, \langle q', i \rangle \rangle$ in $G$ has a value in $\mathcal{L}$, namely $comp(\delta(q, \sigma_i, q'))$. Also, vertices in $Q \times \{0\}$ and $Q \times \{n\}$ have an initial and an acceptance value, respectively, induced by $comp(Q_0)$ and $comp(F)$. The value of $\mathcal{A}$ on $w$ is the meet of the values of the paths of $G$, where a value of a path is the join of the values of its components. In order for $\mathcal{A}'$ to map $w$ to this value, we let $\mathcal{A}'$ keep track of paths that still have to contribute to a component value, and let the traversal value of the runs of $\mathcal{A}'$ maintain the value contributed so far. Thus, as in the subset construction,

$\mathcal{A}'$ follows all runs of $\mathcal{A}$ (that is, all the paths of $G$). However, at any time during the run, $\mathcal{A}'$ may decide nondeterministically to take into account the current component value of some of the paths. Two things happen in a transition in which $\mathcal{A}'$ decides to take into account paths that go through a vertex whose state component belongs to a set $P \subseteq Q$. First, the traversal value of the transition is the meet of the traversal value of transitions that enter $P$. Second, in its subset construction, $\mathcal{A}'$ release the set $P$, as there is no further need to follow paths that visit $P$.

In Section 3.3, we present a general paradigm for decomposing lattice automata to Boolean automata, each associated with a join-irreducible element of the lattice. The paradigm can be used for proving Theorem 8 too. In the full version we describe a direct construction, which applies the paradigm, but hides the intermediate Boolean automata.

We can now complement an LNFW $\mathcal{A}$ by transforming the LUFW with the same structure as $\mathcal{A}$ to an LNFW. Hence, by Lemma 1 and Theorem 8, we have the following:

**Theorem 9 [closure under complementation].** *Let $\mathcal{A}$ be an LNFW with $n$ states. There is an LNFW $\mathcal{A}'$, with $2^n$ states, such that $L(\mathcal{A}') = comp(L(\mathcal{A}))$.*

## 3.2 Deterministic Automata on Finite Words

**Theorem 10 [closure under join and meet].** *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be LDFW over $\mathcal{L}$. There are LDFW $\mathcal{A}_\vee$ and $\mathcal{A}_\wedge$ such that $L(\mathcal{A}_\vee) = L(\mathcal{A}_1) \vee L(\mathcal{A}_2)$ and $L(\mathcal{A}_\wedge) = L(\mathcal{A}_1) \wedge L(\mathcal{A}_2)$. If $\mathcal{A}_1$ has $n_1$ states, $\mathcal{A}_2$ has $n_2$ states, and $\mathcal{L}$ has $m$ elements, then $\mathcal{A}_\vee$ has at most $n_1 \cdot n_2 \cdot m^2$ and at least $n_1 \cdot n_2 \cdot m$ states, and $\mathcal{A}_\wedge$ has $n_1 \cdot n_2$ states.*

The meet construction coincides with the one for LNFW. For the join construction, we first simplify $\mathcal{A}_1$ and $\mathcal{A}_2$ using Theorem 6 and only then apply the construction for LNFW[7].

We now turn to study complementation of LDFW. In the Boolean setting, complementation of deterministic automata is easy, and involves dualization. In the lattice setting dualization does not work, and should be combined with simplification. Therefore, we have the following.

**Theorem 11 [closure under complementation].** *Let $\mathcal{A}$ be an LDFW, with $n$ states, over $\mathcal{L}$. There is an LDFW $\mathcal{A}'$, with $n \cdot m$ states, such that $L(\mathcal{A}') = comp(L(\mathcal{A}))$.*

## 3.3 Lattice Automata on Infinite Words

Lattice automata can run on infinite words and define $\mathcal{L}$-languages of words in $\Sigma^\omega$. A *nondeterministic Büchi lattice automaton* on infinite words (LNBW, for short) has the

[7] The gap between the upper and the lower bound in Theorem 10 follows from the fact that the exact dependency in $m$ depends on the type of the lattice $\mathcal{L}$. For all types, the join construction requires at most an $m^2$ blow-up, and at least an $m$ blow-up. By considering the types individually, it is possible to tighten the bound. In particular, for a lattice that is a full order, the tight bound is $n_1 \cdot n_2 \cdot m$, and for the powerset lattice, the tight bound is $n_1 \cdot n_2 \cdot m^{\log_2 3}$. Essentially, the different types of lattices induce different ways to partition the $m^2$ pairs of lattice values between the state space of the joint automaton and the value accumulated by the run in the form of traversal value.

same components as an LNFW, thus $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, Q_0, \delta, F \rangle$, only that it runs on infinite words. A *run* of $\mathcal{A}$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots$ is an infinite sequence $r = q_0, q_1, \ldots$ of states. The *traversal value* of $r$ on $w$ is $trval(r, w) = Q_0(q_0) \wedge \bigwedge_{i \geq 0} \delta(q_i, \sigma_{i+1}, q_{i+1})$. The *acceptance value* of $r$ on $w$ is $aval(r, w) = \bigwedge_{i \geq 0} \bigvee_{j \geq i} F(q_j)$. The *value* of $r$ on $w$ is $val(r, w) = trval(r, w) \wedge aval(r, w)$.

Note that the acceptance value of a run corresponds to the Büchi condition in the Boolean case. There, $F$ should be visited infinitely often, thus all suffixes should visit $F$. Accordingly, here, the meet of all suffixes is taken, where each suffix contribute the join of its members.

**Theorem 12 [LNBW closure properties].** *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be LNBWs with $n_1$ and $n_2$ states, respectively.*

1. *There is an LNBW $\mathcal{A}_\vee$ with $n_1 + n_2$ states such that $L(\mathcal{A}_\vee) = L(\mathcal{A}_1) \vee L(\mathcal{A}_2)$.*
2. *There is an LNBW $\mathcal{A}_\wedge$ with $3 \cdot n_1 \cdot n_2$ states such that $L(\mathcal{A}_\wedge) = L(\mathcal{A}_1) \wedge L(\mathcal{A}_2)$.*
3. *There is an LNBW $\tilde{\mathcal{A}}_1$ with $2^{O(n_1 \log(n_1))}$ states such that $L(\tilde{\mathcal{A}}_1) = comp(L(\mathcal{A}_1))$.*

The proof of Theorem 12 follows from a general paradigm for transformation between lattice automata. The key observation is that a lattice-automaton over lattice $\mathcal{L}$ can be decomposed to a family Boolean automata where each Boolean automaton in the family corresponds to a join-irreducible (or meet irreducible) element of $\mathcal{L}$. A transformation on the lattice automaton can then be obtained by applying the transformation on the underlying Boolean automata, which can then be composed back to a lattice automaton. For the paradigm to work, we need to ensure some consistency requirements that have to do with maintaining the order of the lattice. In the following NBW stands for Nondeterministic Büchi automata on Words. We proceed with the details.

For an underlying set of states $Q$, we introduce an ordering on NBWs whose state space is $Q$. For $i \in \{1, 2\}$, let $\mathcal{A}_i = \langle \Sigma, Q, Q_i^0, \delta_i, F_i \rangle$ be an NBW. Let $\mathcal{A}_1 \leq \mathcal{A}_2$ when $Q_2^0 \subseteq Q_1^0$, $\delta_2 \subseteq \delta_1$, and $F_2 \subseteq F_1$. Intuitively, "smaller automata have more accepting runs". Formally, it is easy to see that $\mathcal{A}_1 \leq \mathcal{A}_2$ implies $L(\mathcal{A}_2) \subseteq L(\mathcal{A}_1)$.

A family $\{\mathcal{A}_l\}_{l \in \mathcal{L}}$ of NBWs that share a state space and are indexed by lattice elements is *$\mathcal{L}$-consistent* if $l_1 \leq l_2$ implies $\mathcal{A}_{l_1} \leq \mathcal{A}_{l_2}$. Similarly, a family is *$\mathcal{L}$-reverse-consistent* if $l_1 \leq l_2$ implies $\mathcal{A}_{l_1} \geq \mathcal{A}_{l_2}$.

**Lemma 2 [decomposition].** *For an LNBW $\mathcal{A}$ it is possible to construct, in logarithmic space, the following $\mathcal{L}$-consistent families:*

1. *A family $\{\mathcal{A}_l\}_{l \in JI(\mathcal{L})}$ of NBWs such that for all $w \in \Sigma^\omega$, we have $w \in L(\mathcal{A}_l)$ iff $\mathcal{A}(w) \geq l$.*
2. *A family $\{\mathcal{A}_l\}_{l \in MI(\mathcal{L})}$ of NBWs such that for all $w \in \Sigma^\omega$, we have $w \notin L(\mathcal{A}_l)$ iff $\mathcal{A}(w) \leq l$.*

The proof for the join irreducible case is based on a construction of the NBWs $\mathcal{A}_l$ according to criteria like $Q_l^0 = \{q \in Q \mid Q_0(q) \geq l\}$. The proof of the meet irreducible case is based on a construction according to criteria like $Q_l^0 = \{q \in Q \mid Q_0(q) \not\leq l\}$.

For tuples of NBWs, we say that $\langle \mathcal{A}_1, \ldots, \mathcal{A}_k \rangle \leq \langle \mathcal{B}_1, \ldots, \mathcal{B}_k \rangle$ iff $\mathcal{A}_i \leq \mathcal{B}_i$ for every $i \in \{1, \ldots, k\}$. We say that a construction $\varphi : \text{NBW}^k \to \text{NBW}$ is *monotone* if

$\langle \mathcal{A}_1, \ldots, \mathcal{A}_k \rangle \leq \langle \mathcal{B}_1, \ldots, \mathcal{B}_k \rangle$ implies $\varphi(\langle \mathcal{A}_1, \ldots, \mathcal{A}_k \rangle) \leq \varphi(\langle \mathcal{B}_1, \ldots, \mathcal{B}_k \rangle)$. A construction is *antitone* if $\langle \mathcal{A}_1, \ldots, \mathcal{A}_k \rangle \leq \langle \mathcal{B}_1, \ldots, \mathcal{B}_k \rangle$ implies $\varphi(\langle \mathcal{A}_1, \ldots, \mathcal{A}_k \rangle) \geq \varphi(\langle \mathcal{B}_1, \ldots, \mathcal{B}_k \rangle)$.

**Lemma 3.** *Let $k \geq 0$ be an integer. For every $i \leq k$, let $\{\mathcal{A}_l^i\}_{l \in \mathcal{L}}$ be an $\mathcal{L}$-consistent family. If $\varphi : \mathrm{NBW}^k \to \mathrm{NBW}$ is a monotone construction, then $\{\varphi(\mathcal{A}_l^1, \ldots \mathcal{A}_l^k)\}_{l \in \mathcal{L}}$ is an $\mathcal{L}$-consistent family. Similarly, if $\varphi$ is antitone then $\{\varphi(\mathcal{A}_l^1, \ldots \mathcal{A}_l^k)\}_{l \in \mathcal{L}}$ is an $\mathcal{L}$-reverse-consistent family.*

**Lemma 4 [composition].** *Let $\{\mathcal{A}_l\}_{l \in JI(\mathcal{L})}$ be an $\mathcal{L}$-consistent family of NBWs, parameterized by the join irreducible elements of $\mathcal{L}$. There is an LNBW $\mathcal{A}$, sharing the state space of the family, such that for every $w \in \Sigma^\omega$ and $l \in JI(\mathcal{L})$, it holds that $w \in L(\mathcal{A}_l)$ iff $L(\mathcal{A})(w) \geq l$. Furthermore, the construction of $\mathcal{A}$ can be made in logarithmic space.*

The proof is based on the construction of $\mathcal{A}$ from $\{A_l\}_{l \in JI(\mathcal{L})}$ according to criteria like $Q_0(q) = \bigvee \{l \in JI(\mathcal{L}) \mid q \in Q_l^0\}$.

We now have the basic building blocks needed to apply the paradigm of reducing lattice automata constructions to Boolean ones. Below we show how to apply this paradigm in the case of LNBW complementation. The other cases are simpler and are left to the reader. As a first step, we need a Boolean construction for NBW complementation that is an antitone.

**Lemma 5.** *There exists an antitone construction $\varphi$ : NBW $\to$ NBW such that for every NBW $\mathcal{A}$, we have $L(\varphi(\mathcal{A})) = comp(L(\mathcal{A}))$. Furthermore, if $\mathcal{A}$ has $n$ states, then $\varphi(\mathcal{A})$ has at most $2^{O(n \log(n))}$ states, and the construction can be made using space polynomial in $n$.*

In the full version, we prove the lemma by proving that (a small variant of) the [KV01] construction for NBW complementation is antitone. To prove the results for join and meet of languages, we need similar constructions of monotone (rather than antitone) constructions of union and intersection. The standard construction for union is already monotone. For the meet case, a small variant of the usual [Cho74] construction for intersection is needed, and is discussed in the full version.

We can now complete the construction for LNBW complementation. Given an LNBW $\mathcal{A}$, we use the decomposition lemma to construct a consistent family $\{\mathcal{A}_l\}_{l \in MI(\mathcal{L})}$ of NBWs such that $\mathcal{A}(w) \leq l$ iff $w \notin L(\mathcal{A}_l)$ for all $w \in \Sigma$. By applying the construction from Lemma 5, we get a reverse-consistent family $\{\mathcal{A}'_l\}_{l \in MI(\mathcal{L})}$ of NBWs such that $\mathcal{A}(w) \leq l$ iff $w \in L(\mathcal{A}'_l)$ for all $w \in \Sigma$.

Next, we re-index the family by identifying $\mathcal{A}'_l$ with $\mathcal{A}''_{comp(l)}$. Since an element is meet irreducible iff its complement is join irreducible, the resulting family $\{\mathcal{A}''_{comp(l)}\}_{l \in MI(\mathcal{L})}$ is indexed by the join irreducible elements of $\mathcal{L}$ and can be seen as $\{\mathcal{A}''_l\}_{l \in JI(\mathcal{L})}$. Furthermore, for $l_1, l_2 \in JI(\mathcal{L})$, if $l_1 \leq l_2$, then $comp(l_2) \geq comp(l_1)$. Therefore, since $\{\mathcal{A}'_l\}$ is a reverse-consistent family, we get that $\mathcal{A}'_{comp(l_1)} \leq \mathcal{A}'_{comp(l_2)}$; i.e., $\mathcal{A}''_{l_1} \leq \mathcal{A}''_{l_2}$. Thus, $\{\mathcal{A}''_l\}_{l \in JI(\mathcal{L})}$ is a consistent family.

Finally, we apply the composition lemma on $\{\mathcal{A}''_l\}_{l \in JI(\mathcal{L})}$ to get a single LNBW $\tilde{\mathcal{A}}$. To prove that $\tilde{\mathcal{A}}$ is indeed $comp(\mathcal{A})$ fix a word $w \in \Sigma^\omega$ and a join irreducible element $l \in JI(\mathcal{L})$. The following equivalences hold: $\tilde{\mathcal{A}}(w) \geq l$ iff $w \in L(\mathcal{A}''_l)$ iff

$w \in L(\mathcal{A}'_{comp(l)})$ iff $w \notin L(\mathcal{A}_{comp(l)})$ iff $\mathcal{A}(w) \leq comp(l)$ iff $comp(\mathcal{A}(w)) \geq l$. The result follows from Birkhoff's representation theorem.

## 4 Applications

In this section we apply our framework to the satisfiability and model-checking problems of multi-valued LTL. We first discuss decision problems for LNFW and LNBW.

### 4.1 Decision Problems

Consider an LNFW (or LNBW) $\mathcal{A}$ over a lattice $\mathcal{L}$. The *range* of $\mathcal{A}$ is the set of lattice values $l$ for which there is a word $w$ that $\mathcal{A}$ accepts with value $l$. Thus, $range(\mathcal{A}) = \bigcup_{w \in \Sigma^*} val(\mathcal{A}, w)$. The *emptiness value* of $\mathcal{A}$, denoted $e\_val(\mathcal{A})$, is then the join of all the values in its range; i.e., $e\_val(\mathcal{A}) = \bigvee range(\mathcal{A})$. Intuitively, $e\_val(\mathcal{A})$ describes how likely it is for $\mathcal{A}$ to accept a word. In particular, if $e\_val(\mathcal{A}) = \bot$, then $\mathcal{A}$ gives value $\bot$ to all the words in $\Sigma^*$. Over Boolean lattice, $e\_val(\mathcal{A}) = \bot$ if $\mathcal{A}$ is empty and $e\_val(\mathcal{A}) = \top$ if $\mathcal{A}$ is not empty. Note, however, that for a general (finite distributive De Morgan) lattice, $e\_val(\mathcal{A}) \neq \bot$ does not imply that there is a word that is accepted with value $e\_val(\mathcal{A})$. The *emptiness-value problem* is to decide, given an LNFW (or LNBW) $\mathcal{A}$, a value $l \in \mathcal{L}$, and an order relation $\sim \in \{<, \leq, =, \geq, >\}$, whether $e\_val(\mathcal{A}) \sim l$.

**Theorem 13.** *The emptiness-value problem for LNFW (or LNBW) is NLOGSPACE-complete.*

In the full version we discuss the *universality-value* and the *implication-value* problems, which corresponds to the universality and the language inclusion problems in the Boolean setting.

### 4.2 LLTL Model Checking and Satisfiability

As discussed in Section 1, the multi-valued setting appears in practice either directly, with multi-valued systems and specifications, or indirectly, as various methods are reduced to reasoning in a multi-valued setting. In this section we show how lattice automata provide a unifying automata-theoretic framework for reasoning about multi-valued systems and specifications,

A *multi-valued Kripke structure* is a six-tuple $K = \langle AP, \mathcal{L}, W, W_0, R, L \rangle$, where $AP$ is a set of atomic propositions, $\mathcal{L}$ is a lattice, $W$ is a finite set of states, $W_0 \in \mathcal{L}^W$ is an $\mathcal{L}$-set of initial states, $R \in L^{W \times W}$ is an $\mathcal{L}$-transitions relation, and $L : W \to \mathcal{L}^{AP}$ maps each state to an $\mathcal{L}$-set of atomic propositions. We require $R$ to be total in its first element, thus for every $w \in W$ there is at least one $w' \in w$ such that $R(w, w') \neq \bot$. A path of $K$ is an infinite sequence $w_1, w_2, \ldots$ of states. For technical simplicity, we assume that $W_0$ and $R$ are Boolean. As discussed in Remark 3, it is easy to adjust our framework to handle weighted input letters, and hence, weighted initial states and transitions. In the Boolean setting, a path of $K$ is one that has value $\top$, thus $w_1 \in w_0$ and $R(w_i, w_{i+1})$ for all $i \geq 1$.

The logic *LTL* is a linear temporal logic. Formulas of LTL are constructed from a set $AP$ of atomic propositions using the usual Boolean operators and the temporal operators $X$ ("next time") and $U$ ("until"). The semantics of LTL is traditionally defined with respect to computations of Kripke structures in which each state is labeled by a set of atomic propositions true in this state and each two states are either connected or not connected by an edge. Note that traditional Kripke structures correspond to multi-valued Kripke structures over the lattice $\mathcal{L}_2$. We define the logic *Latticed-LTL* (LLTL, for short), which is the expected extension of LTL to multi-valued Kripke structures. The syntax of LLTL is similar to the one of LTL, except that the logic is parameterized by a lattice $\mathcal{L}$ and its constants are elements of $\mathcal{L}$. Let $\pi = w_1, w_2, \ldots$ be a path of a multi-valued Kripke structure. The value of an LLTL formula $\psi$ on the path $\pi$ in position $i$, denoted $val(\pi, i, \psi)$ is inductively defined as follows:

- For a lattice element $l \in L$, we have $val(\pi, i, l) = l$ for all $\pi$ and $i$.
- For an atomic proposition $p \in AP$, we have $val(\pi, i, p) = w_i(p)$ for all $\pi$ and $i$.
- $val(\pi, i, \neg\psi) = \neg val(\pi, i, \psi)$.
- $val(\pi, i, \psi \wedge \theta) = val(\pi, i, \psi) \wedge val(\pi, i, \theta)$.
- $val(\pi, i, X\psi) = val(\pi, i+1, \psi)$.
- $val(\pi, i, \psi U \theta) = \bigvee_{k \geq i}(val(\pi, k, \theta) \wedge \bigwedge_{i \leq j < k} val(\pi, j, \psi))$.

For an LLTL formula $\psi$, the *satisfiability value* of $\psi$, denoted $sat(\psi)$, is $\bigvee\{val(\pi, 1, \psi) : \pi \in (\mathcal{L}^{AP})^\omega\}$. Thus, the satisfiability value describes how likely it is for some path to satisfy $\psi$. The LLTL satisfiability problem is to determine, given an LLTL formula $\psi$, a value $l \in \mathcal{L}$, and an order relation $\sim \in \{<, \leq, =, \geq, >\}$, whether $sat(\psi) \sim l$. For a multi-valued Kripke structure $K$ and an LLTL formula $\psi$, the *satisfaction value* of $\psi$ in $K$, denoted $sat(K, \psi)$, is $\bigwedge\{val(\pi, 1, \psi) : \pi \text{ is a path of } K\}$. Thus, the satisfaction value describes how likely it is for all paths of $K$ to satisfy $\psi$. The LLTL model-checking problem is to determine, given a multi-valued Kripke structure $K$, an LLTL formula $\psi$, a value $l \in \mathcal{L}$, and an order relation $\sim \in \{<, \leq, =, \geq, >\}$, whether $sat(K, \psi) \sim l$.

**Theorem 14.** *Given an LLTL formula $\psi$, there is an LNBW $\mathcal{A}_\psi$ such that for every word $w \in (\mathcal{L}^{AP})^\omega$, we have $\mathcal{A}_\psi(w) = val(w, 1, \psi)$.*

We can now use the automata-theoretic approach in order to solve the satisfiability and model checking problems for LLTL.

**Theorem 15 [LLTL satisfiability and model checking].** *The LLTL satisfiability-value and satisfaction-value problems are PSPACE-complete.*

The proof is similar to the standard automata-theoretic approach to verification proofs. The full proof can be found in the full version.

Note that Theorem 15 also follows from reduction to several Boolean problems as presented in [BG04]. The advantage of the approach presented here, is solving LLTL model checking and satisfiability using direct lattice methods. The advantages of such direct methods were argued in [BG04], which solved the model checking for $\mu$L (the lattice extension of $\mu$-calculus) directly, using EAA. Theorem 15, however, does not follow from the latter due to the doubly-exponential blow up of translating LTL formulas to $mu$-calculus.

## References

[BG99] G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proc. 11th CAV*, LNCS 1633, pages 274–287, 1999.

[BG01] G. Bruns and P. Godefroid. Temporal logic query checking. In *Proc. 16th LICS*, pages 409–420, 2001.

[BG04] G. Bruns and P. Godefroid. Model checking with 3-valued temporal logics. In *31st ICALP*, LNCS 3142, pages 281–293, 2004.

[CDG01] M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using SPIN. In *SPIN Workshop in model-checking software*, LNCS 2057, pages 16-36, 2001.

[Cha00] W. Chan. Temporal-logic queries. In *Proc. 12th CAV*, LNCS 1855, pages 450–463, 2000.

[Cho74] Y. Choueka. Theories of automata on $\omega$-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.

[EC01] S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Proc. 23rd ICSE*, pages 411–420, 2001.

[GS97] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Proc. 9th CAV*, LNCS 1254, pages 72–83, 1997.

[HH04] A. Hussain and M. Huth. On model checking multiple hybrid views. Technical Report TR-2004-6, University of Cyprus, 2004.

[IEEE93] IEEE standard multivalue logic system for VHDL model interoperability (std_logic_1164), 1993.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

[KS86] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1986.

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

[KV01] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM TOCL*, 2(2):408–429, July 2001.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on $\omega$-words. *Theoretical Computer Science*, 32:321–330, 1984.

[Moh97] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[RS59] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[Saf88] S. Safra. On the complexity of $\omega$-automata. In *Proc. 29th FOCS*, pages 319–327, 1988.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.