Minimizing Deterministic Lattice Automata

Shulamit Halamish and Orna Kupferman

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

Abstract. Traditional automata accept or reject their input, and are therefore Boolean. In contrast, weighted automata map each word to a value from a semiring over a large domain. The special case of *lattice automata*, in which the semiring is a finite lattice, has interesting theoretical properties as well as applications in formal methods. A *minimal deterministic automaton* captures the combinatoric nature and complexity of a formal language. Deterministic automata are used in run-time monitoring, pattern recognition, and modeling systems. Thus, the minimization problem for deterministic automata is of great interest, both theoretically and in practice.

For traditional automata on finite words, a minimization algorithm, based on the Myhill-Nerode right congruence on the set of words, generates in polynomial time a canonical minimal deterministic automaton. A polynomial algorithm is known also for weighted automata over the tropical semiring. For general deterministic weighted automata, the problem of minimization is open. In this paper we study minimization of lattice automata. We show that it is impossible to define a right congruence in the context of lattices, and that no canonical minimal automaton exists. Consequently, the minimization problem is much more complicated, and we prove that it is NP-complete. As good news, we show that while right congruence fails already for finite lattices that are fully ordered, for this setting we are able to combine a finite number of right congruences and generate a minimal deterministic automaton in polynomial time.

1 Introduction

Automata theory is one of the longest established areas in Computer Science. Standard applications of automata theory include pattern matching, syntax analysis, and formal verification. In recent years, novel applications of automata-theoretic concepts have emerged from numerous sciences, like biology, physics, cognitive sciences, control, and linguistics. These novel applications require significant advances in fundamental aspects of automata theory [2]. One such advance is a transition from a Boolean to a multi-valued setting: while traditional automata accept or reject their input, and are therefore Boolean, novel applications, for example in speech recognition and image processing [18], are based on weighted automata, which map an input word to a value from a semiring over a large domain [7].

Focusing on applications in formal verification, the multi-valued setting arises directly in *quantitative verification* [10], 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 [9], *query checking* [5], which can be reduced to model checking over multi-valued systems, and verification of systems from *inconsistent viewpoints* [11], in which the value of the atomic propositions is the composition of their values in the different viewpoints.

Recall that 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). In many of the applications of the multi-valued setting described above, the values are taken from finite lattices. For example (see Figure 2), in the abstraction application, researchers use the lattice $\mathcal{L}_3$ of three fully ordered values [3], as well as its generalization to $\mathcal{L}_n$ [6]. In query checking, the lattice elements are sets of formulas, ordered by the inclusion order [4]. When reasoning about inconsistent viewpoints, each viewpoint is Boolean, and their composition gives rise to products of the Boolean lattice, as in $\mathcal{L}_{2,2}$ [8]. Finally, when specifying prioritized properties of system, one uses lattices in order to specify the priorities [1].

In [13], the authors study lattice automata, their theoretical properties, and decision problems for them. In a nondeterministic lattice automaton on finite words (LNFA, for short), each transition is associated with a *transition value*, which is a lattice element (intuitively indicating the truth of the statement “the transition exists”), and each state is associated with an *initial value* and an *acceptance value*, indicating the truth of the statements “the state is initial/accepting”, respectively. Each run r of an LNFA $\mathcal{A}$ has a value, which is the *meet* of the values of all the components of r: the initial value of the first state, the transition value 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 LNFA over an alphabet Σ and lattice $\mathcal{L}$ induces an $\mathcal{L}$-language $L : \Sigma^* \to \mathcal{L}$. Note that traditional finite automata (NFAs) can be viewed as a special case of LNFAs over the lattice $\mathcal{L}_2$. In a deterministic lattice automaton on finite words (LDFA, for short), at most 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 on σ to q' is not $\bot$. Thus, an LDFA $\mathcal{A}$ 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 $\mathcal{A}$.

For example, the LDFA $\mathcal{A}$ in Figure 1 below is over the alphabet $\Sigma = \{0, 1, 2\}$ and the lattice $\mathcal{L} = \langle 2^{\{a,b,c,d\}}, \subseteq \rangle$. All states have acceptance value $\{a, b, c, d\}$, and this is also the initial value of the single initial state. The $\mathcal{L}$-language of $\mathcal{A}$ is $L : \Sigma^* \to \mathcal{L}$ such that $L(\epsilon) = \{a, b, c, d\}$, $L(0) = \{c, d\}$, $L(0 \cdot 0) = \{d\}$, $L(1) = \{a, b\}$, $L(1 \cdot 0) = \{a\}$, $L(2) = \{c, d\}$, $L(2 \cdot 0) = \{c\}$, and $L(x) = \emptyset$ for all other $x \in \Sigma^*$.

A *minimal deterministic automaton* captures the combinatoric nature and complexity of a formal language. Deterministic automata are used in run-time monitoring, pattern recognition, and modeling systems. Thus, the minimization problem for deterministic automata is of great interest, both theoretically and in practice. For traditional automata on finite words, a minimization algorithm, based on the Myhill-Nerode right congruence on the set of words, generates in polynomial time a canonical minimal deterministic automaton [20,21]. In more detail, given a regular language L over Σ, then the relation $\sim_L \subseteq \Sigma^* \times \Sigma^*$, where $x \sim_L y$ iff for all $z \in \Sigma^*$ we have that $x \cdot z \in L$ iff

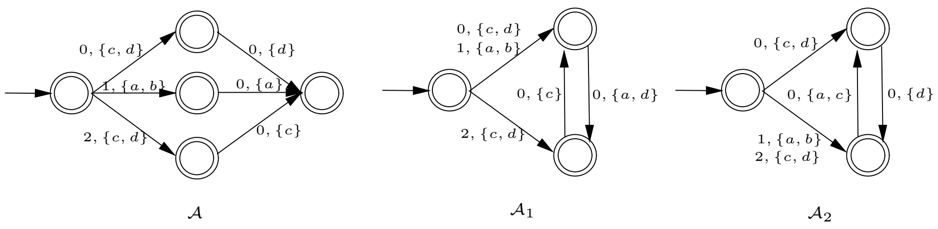


Fig. 1. An LDFA with two different minimal LDFAs

$y \cdot z \in L$, is an equivalence relation, its equivalence classes correspond to the states of a minimal automaton for $\mathcal{A}$, and they also uniquely induce the transitions of such an automaton. Further, given a deterministic automaton for L, it is possible to use the relation $\sim_L$ in order to minimize it in polynomial time.

A polynomial algorithm is known also for deterministic weighted automata over the *tropical semiring* [18]. In such automata, each transition has a value in $\mathbb{R}$, each state has an initial and acceptance values in $\mathbb{R}$, and the value of a run is the sum of the values of its components. Unlike the case of DFAs, in the case of weighted automata there may be several different minimal automata. They all, however, have the same graph topology, and only differ by the values assigned to the transitions and states. In other words, there is a canonical minimal topology, but no canonical minimal automaton. For semirings that are not the tropical semiring, and in particular, for lattice automata, the minimization problem is open.

In this work we study the minimization problem for lattice automata. An indication that the problem is not easy is the fact that in the latticed setting, the "canonical topology" property does not hold. To see this, consider again the LDFA $\mathcal{A}$ in Figure 1. Note that an automaton for $L(\mathcal{A})$ must have at least three states. Indeed, if it has at most two then the word $w = 000$ would not get the value $\perp$, whereas $L(000) = \perp$. Hence, the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ presented on its right are two minimal automata for L. Their topologies differ by the transition leaving the initial state with the letter 1[1].

The absence of the "canonical topology" property suggests that efforts to construct the minimal LDFA by means of a right congruence are hopeless. The main difficulty in minimizing LDFAs is that the "configuration" of an automaton consists not only of the current state, but also of the run that leads to the state, and the value accumulated reaching it[2]. Attempts to somehow allow the definition of the right congruence to refer to lattice values (as is the case in the successful minimization of weighted automata over the tropical semiring [18]) do not succeed. To see this, consider even a simpler

[1] Note that the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are not simple, in the sense that the transitions are associated with values from the lattice that are not $\perp$ or $\top$. The special case of simple lattice automata, where the value of a run is determined only by the value associated with the last state of the run is simpler, and has been solved in the context of fuzzy automata [17,22]. We will get back to it in Section 2.2.

[2] It is interesting to note that also in the context of deterministic Büchi automata, there is no single minimal topology for a minimal automaton. As with LDFAs, this has to do with the fact that the outcome of a run depends on its on-going behavior, rather than its last state only.

model of LDFA, in which all acceptance values are $\top$ (the greatest value in the lattice, and thus, $L(x \cdot z) \leq L(x)$ for all $x, z \in \Sigma^*$). A natural candidate for a right congruence for an $\mathcal{L}$-language L is the relation $\sim_L \subseteq \Sigma^* \times \Sigma^*$ such that $x \sim_L x'$ iff for all $z \in \Sigma^*$ there exists $l \in \mathcal{L}$ such that $L(x \cdot z) = L(x) \wedge l$ and $L(x' \cdot z) = L(x') \wedge l$. Unfortunately, the relation is not even transitive. For example, for the language L of the LDFA $\mathcal{A}$ in Figure 1, we have $0 \sim_L 1$ and $1 \sim_L 2$, but $0 \not\sim_L 2$.

We formalize these discouraging indications by showing that the problem of LDFA minimization is NP-complete. This is a quite surprising result, as this is the first parameter in which lattice automata are more complex than weighted automata over the tropical semiring. In particular, lattice automata can always be determinized [13,15], which is not the case for weighted automata over the tropical semiring [18]. Also, language containment between nondeterministic lattice automata can be solve in PSPACE [13,14], whereas the containment problem for weighted automata on the tropical semiring is undecidable [16]. In addition, lattices have some appealing properties that general semirings do not, which make them seem simpler. 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 $a \wedge (a \vee b) = a$), do not hold in a general semiring, and do hold for lattices. Nevertheless, as mentioned, we are able to prove that their minimization is NP-complete.

Our NP-hardness proof is by a reduction from the vertex cover problem [12]. The lattice used in the reduction is $\mathcal{L} \subset 2^E$, for the set E of edges of the graph, with the usual set-inclusion order. The reduction strongly utilizes on the fact that the elements of 2^E are not fully ordered. The most challenging part of the reduction is to come up with a lattice that, on the one hand, strongly uses the fact $\mathcal{L}$ is not fully ordered, yet on the other hand is of size polynomial in E (and still satisfies the conditions of closure under meet and join).

As pointed above, the NP-hardness proof involved a partially ordered lattice, embodied in the "subset lattice", and strongly utilizes on the order being partial. This suggests that for fully ordered lattices, we may still be able to find a polynomial minimization algorithm. On the other hand, as we shall show, the property of no canonical minimal LDFA is valid already in the case of fully ordered lattice, which suggests that no polynomial algorithm exists. As good news, we show that minimization of LDFAs over fully ordered lattices can nevertheless be done in polynomial time. The idea of the algorithm is to base the minimization on linearly many minimal DFAs that correspond to the different lattice values. The fact the values are fully ordered enables us to combine these minimal automata into one minimal LDFA.

Due to lack of space, some proofs are omitted in this version and can be found in the full version, on the authors' home pages.

2 Preliminaries

This section introduces the definitions and notations related to lattices and lattice automata, as well as some background about the minimization problem.

2.1 Lattices and Lattice Automata

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 $\langle A, \leq \rangle$ 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 $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.

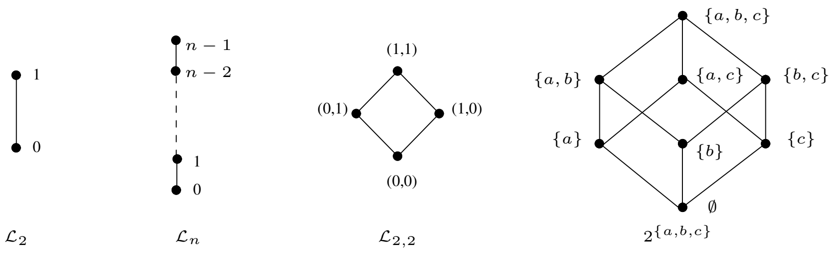


Fig. 2. Some lattices

In Figure 2 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 $0, 1 ... n-1$, with the order $0 \leq 1 \leq ... \leq n-1$. 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 $b \leq b'$. 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 $\{a, c\} \wedge \{b\} = \bot$.

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 $S : X \rightarrow \mathcal{L}$ 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* is an $\mathcal{L}$-set over Σ^*. Thus, an $\mathcal{L}$-language $L : \Sigma^* \rightarrow \mathcal{L}$ assigns a value in $\mathcal{L}$ to each word over Σ.

A *deterministic lattice automaton* on finite words (LDFA, for short) is a 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 $\mathcal{L}$-transition-relation, and $F \in \mathcal{L}^Q$ is an $\mathcal{L}$-set of accepting states. The fact $\mathcal{A}$ is deterministic is reflected in two conditions on Q_0 and δ. First, there is at most one state $q \in Q$, called the *initial state* of $\mathcal{A}$, such that $Q_0(q) \neq \bot$. In addition, for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at most one state $q' \in Q$, called the *σ-destination* of q, such that $\delta(q, \sigma, q') \neq \bot$. The *run* of an LDFA 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, where q_0 is the initial state of $\mathcal{A}$, and for all $1 \leq i \leq n$ it holds that q_i is the *σ_i-destination* of q_{i-1}. Note that $\mathcal{A}$ may not have a run on w. The *value* of w is $val(w) = Q_0(q_0) \wedge \bigwedge_{i=1}^{n} \delta(q_{i-1}, \sigma_i, q_i) \wedge F(q_n)$. Intuitively, $Q_0(q_0)$ is the value of q_0 being initial, $\delta((q_{i-1}, \sigma_i, q_i))$ is 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. The *traversal value* of w is $tr\_val(w) = Q_0(q_0) \wedge \bigwedge_{i=1}^{n} \delta(q_{i-1}, \sigma_i, q_i)$, and its *acceptance value* is $F(q_n)$. The $\mathcal{L}$-language of $\mathcal{A}$, denoted $L(\mathcal{A})$, maps each word w to the value of its run in $\mathcal{A}$. Note that since $\mathcal{A}$ is deterministic, it has at most one run on w whose value is not $\bot$. This is why we talk about the traversal and acceptance values of words rather than of runs.

Note that traditional deterministic automata over finite words (DFA, for short) correspond to LDFA over the lattice $\mathcal{L}_2$. Indeed, over $\mathcal{L}_2$, a word is mapped to the value $\top$ if the run on it uses only transitions with value $\top$ and its final state has value $\top$.

An LDFA is *simple* if Q_0 and δ are Boolean. Note that the traversal value of a run r of a simple LDFA is either $\bot$ or $\top$, thus the value of r is induced by F. Simple LDFAs have been studied in the context of *fuzzy* logic and automata [17,22].

Analyzing the size of $\mathcal{A}$, one can refer to $|\mathcal{L}|$, $|Q|$, and $|\delta|$. Since the emphasize in this paper is on the size of the state space, we use $|\mathcal{A}|$ to refer to the size of its state space. Our complexity results, however, refer to the size of the input, and thus take into an account the other components of $\mathcal{A}$ as well, and in particular the size of $\mathcal{L}$.

2.2 Minimizing LDFAs

We now turn to discuss the problem of minimizing LDFA. We describe the difficulties of the problem, and present some examples that are important for the full comprehension of the issue.

A first attempt to minimize lattice automata would be to follow the classical paradigm for minimizing DFA using the Myhill Nerode theorem [20,21]. In fact, for the case of simple LDFA, the plan proceeds smoothly (see [17,22], where the problem is discussed by means of fuzzy automata)[3]: Given an $\mathcal{L}$-language L, we extend the definition of $\sim_L$ to fit the nature of $\mathcal{L}$-languages. For all $x, x' \in \Sigma^*$, we say that $x \sim_L x'$ iff for all $z \in \Sigma^*$ it holds that $L(x \cdot z)$=$L(x' \cdot z)$. Clearly, $\sim_L$ is an equivalence relation, since it is based on the equality relation. As in the case of DFA, we can build a minimal simple LDFA $\mathcal{A}_{min}$ for L such that $|\mathcal{A}_{min}| = | \sim_L |$. We construct it in the same manner, only that here the acceptance values are defined such that $F([x]) = L(x)$. Also, we can show that every simple LDFA for L must have at least $| \sim_L |$ states. Indeed, if this is not the case then we have two words $x, x' \in \Sigma^*$ reaching the same state q, while $x \not\sim_L x'$. The contradiction is reached when we read the distinguishing tail z from q as in the case of DFA, due to the fact that in simple LDFA the value of the words is solely determined by the final state.

[3] Several variants of fuzzy automata are studied in the literature. The difficulties we cope with in the minimization of our lattice automata are not applied to them, and indeed they can be minimized by variants of the minimization construction described here [19].

So, simple lattice automata can be minimized in polynomial time, and the key for proving it was a generalization of the right congruence to require agreement on the values of the words. Encouraged by this, we now turn to examine the case of general LDFAs. Unfortunately, the generalization does not seem to work here. To see the difficulties in the latticed setting, consider an $\mathcal{L}$-language L over $\Sigma = \mathcal{L}$ such that $L(l_1 \cdot l_2 \cdots l_n) = \bigwedge_{i=1}^{n} l_i$. The language L can be recognized by an LDFA $\mathcal{A}$ with a single state q. The initial and acceptance values of q are $\top$, and for every $l \in \Sigma$, there is an l-transition with value l from q to itself. Thus, the single run of $\mathcal{A}$ on an input word maps it to the meet of all letters. Clearly, there exist $x, x' \in \Sigma^*$ such that $L(x) \neq L(x')$, and still $\mathcal{A}$ has only one state. Thus, despite being mapped to different values, x and x' reach the same state in $\mathcal{A}$. This observation shows a crucial difference between the setting of DFAs or simple LDFAs and the one of general LDFA. It is only in the latter, that a "configuration" of a word is not only the state it reaches but also the traversal value accumulated when reading it. In our example, the words x and x' have a different traversal value, which is implicit in the LDFA, and an attempt to distinguish between words according to the values they have accumulated results in LDFAs with needlessly more states. Accordingly, a right congruence that may be helpful for minimization should take into an account the value accumulated by the words, and in particular, in the case of L above, should have only one equivalence class.

Following the above discussion, we now try to define an equivalence relation for LDFA that does take into an account the accumulated traversal values. Let us first consider a simpler model of LDFAs in which all acceptance values are $\top$. Note that in this model, for all $x, z \in \Sigma^*$ we have that $L(x \cdot z) \leq L(x)$. Let L be an $\mathcal{L}$-language in the model above. We define a relation $\sim_L \subseteq \Sigma^* \times \Sigma^*$ such that $x \sim_L x'$ iff for all $z \in \Sigma^*$ there exists $l \in \mathcal{L}$ such that $L(x \cdot z) = L(x) \wedge l$ and $L(x' \cdot z) = L(x') \wedge l$. Note that the relation $\sim_L$ indeed takes into a consideration the values accumulated when x and x' are read. Indeed, x and x' are equivalent iff for all tails $z \in \Sigma^*$ there exists some $l \in \mathcal{L}$ such that z can be read with the value l after reading either x or x'. Unfortunately, the relation is not even transitive. For example, for the language L of the LDFA $\mathcal{A}$ in Figure 1, we have $0 \sim_L 1$ and $1 \sim_L 2$, but $0 \not\sim_L 2$.

We have seen some evidences that minimization of LDFAs cannot follow the minimization paradigm for DFAs and even not the one for deterministic weighted automata over the tropical semiring. In the rest of the paper we formalize these evidences by showing that the problem is NP-complete. We also challenge them by describing a polynomial algorithm for minimization of LDFAs over fully ordered lattices – a special case for which the evidences apply.

3 Minimizing General LDFA

In this section we study the problem of minimizing LDFAs and show that unlike the case of DFAs, and even the case of weighted DFAs over the tropical semiring, which can be minimized in polynomial time, here the problem is NP-complete. We consider the corresponding decision problem MINLDFA=$\{\langle \mathcal{A}, k\rangle : \mathcal{A}$ is an LDFA and there exists an LDFA $\mathcal{A}'$ with at most k states such that $L(\mathcal{A}') = L(\mathcal{A})\}$.

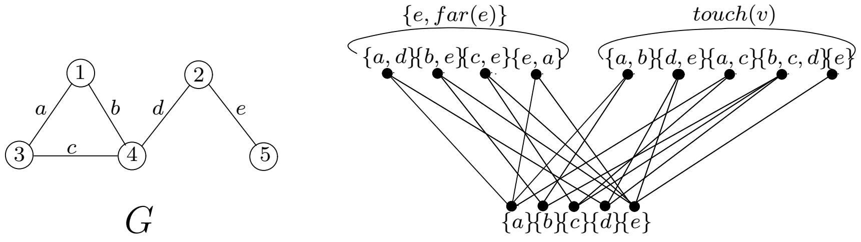
Theorem 1. *MINLDFA is NP-complete.*

Proof. We start with membership in NP. Given $\mathcal{A}$ and k, a witness to their membership in MINLDFA is an LDFA $\mathcal{A}'$ as required. Since $|\mathcal{A}'| = k$, its size is linear in the input. Deciding language equivalence between LDFAs is NLOGSPACE-complete [13], thus we can verify that $L(\mathcal{A}') = L(\mathcal{A})$ in polynomial time.

For the lower bound, we show a polynomial time reduction from the Vertex Cover problem (VC, for short), proved to be NP-complete in [12]. Recall that VC=$\{\langle G, k\rangle : G$ is an undirected graph with a vertex cover of size $k\}$, where a *vertex cover* of a graph $G = \langle V, E\rangle$ is a set $C \subseteq V$ such that for all edges $(u, v) \in E$ we have $u \in C$ or $v \in C$.

Before we describe the reduction, we need some definitions. Consider an undirected graph $G = \langle V, E\rangle$. Let $n = |V|$ and $m = |E|$. For simplicity, we assume that V and E are ordered, thus we can refer to the minimum element in a set of vertices or edges. For $v \in V$, let $touch(v) = \{e : e = (v, u)$ for some $u\}$. For $e = (v_1, v_2) \in E$, let $far(e) = \min\{e' : e' \notin touch(v_1) \cup touch(v_2)\}$. That is, $far(e)$ is the minimal edge that is not adjacent to e. Note that if $\{e' : e' \notin touch(v_1) \cup touch(v_2)\} = \emptyset$, then $\{v_1, v_2\}$ is a VC of size two, so we can assume that $far(e)$ is well defined.

Example 1. In the graph G below, we have, for example, $touch(1) = \{a, b\}$, $touch(2) = \{d, e\}$, $far(a) = d$, $far(b) = e$, and $far(c) = e$.



1
2
3
4
5
a
b
c
d
e
G
$\{e, far(e)\}$
$touch(v)$
$\{a,d\}$ $\{b,e\}$ $\{c,e\}$ $\{e,a\}$
$\{a,b\}$ $\{d,e\}$ $\{a,c\}$ $\{b,c,d\}$ $\{e\}$
$\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$

Fig. 3. A graph and its corresponding lattice

We now turn to describe the reduction. Given an input $G = \langle V, E\rangle$, we construct an LDFA $\mathcal{A} = \langle \mathcal{L}, Q, \Sigma, \delta, Q_0, F\rangle$ as follows:

– $\mathcal{L} \subseteq 2^E$ contains the following elements: $\{\emptyset, E\} \cup \{\{e\} : e \in E\} \cup \{\{e, far(e)\} : e \in E\} \cup \{touch(v) : v \in V\}$, with the usual set-inclusion relation. In particular, $\bot = \emptyset$ and $\top = E$. Note that $\mathcal{L}$ contains at most $2 + n + 2m$ elements ("at most" since $\{e, far(e)\}$ may be identical to $\{far(e), far(far(e))\}$). For example, the graph in Example 1 induces the lattice shown on its right (for clarity, we omit the elements $\top$ and $\bot$ in the figure). Note that in this example we have $\{a, far(a)\} = \{far(a), far(far(a))\}$, so we omit the unnecessary element.

We claim that though $\mathcal{L}$ does not contain all the elements in 2^E, the operators $join$ and $meet$ are well defined for all $l_1, l_2 \in \mathcal{L}$.[4] In the case l_1 and l_2 are ordered,

[4] We note that this point has been the most challenging part of the reduction, as on the one hand, we have to strongly use the fact $\mathcal{L}$ is not fully ordered (as we show in Section 4, polynomial minimization is possible for LDFAs over fully ordered lattice), yet on the other hand the reduction has to be polynomial and thus use only polynomially many values of the subset lattice.

the closure for both $join$ and $meet$ is obvious. Otherwise, we handle the operators separately as follows. We start with the case of $meet$. Closing to $meet$ is easy since $l_1 \wedge l_2$ never contains more than one edge. Indeed, if l_1, l_2 are of the form $touch(v_1)$, $touch(v_2)$ then their $meet$ is the single edge (v_1, v_2). In all other possibilities for l_1 and l_2 that are not ordered, one of them contains at most two edges, so the fact they are not ordered implies that they have at most one edge in their $meet$. As for $join$, given l_1 and l_2 let $S = \{l : l \geq l_1 \text{ and } l \geq l_2\}$. We claim that all the elements in S are ordered, thus we can define $l_1 \vee l_2$ to be the minimal element in S. Assume by way of contradiction that S contains two elements l and l' that are not ordered. On the one hand, $l \wedge l' \geq l_1 \vee l_2$. Since l_1 and l_2 are not ordered, this implies that $l \wedge l'$ is of size at least two. On the other hand, as we argued above, the $meet$ of two elements that are not ordered contains at most one edge, and we have reached a contradiction.

– $Q = \{q_{init}, q_0, ..., q_{m-1}\}$.
– $\Sigma = \{e_0, ..., e_{m-1}\} \cup \{\#\}$.
– For $0 \leq i < m$, we define $\delta(q_{init}, e_i, q_i) = \{e_i, far(e_i)\}$ and $\delta(q_i, \#, q_{(i+1) mod m}) = \{e_i\}$. For all other $q \in Q$ and $\sigma \in \Sigma$, we define $\delta(q, \sigma, q) = \bot$.
– $Q_0(q_{init}) = \top$, and $Q_0(q) = \bot$ for all other $q \in Q$.
– $F(q) = \top$ for all $q \in Q$.

For example, the graph G in Example 1 induces the LDFA $\mathcal{A}_G$ below.

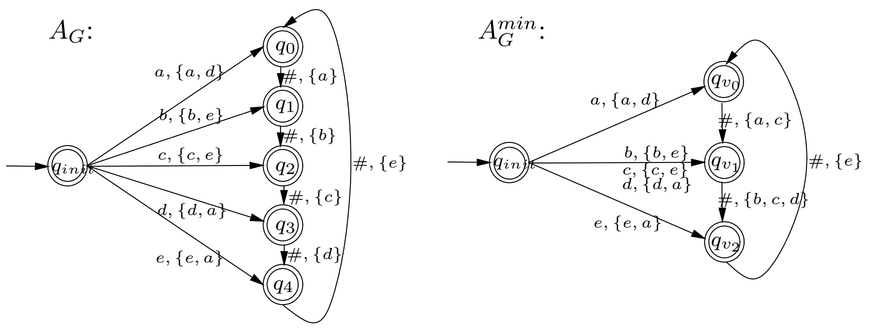


Fig. 4. The LDFA induced by G, and the minimal LDFA that corresponds to the 3-cover $\{3, 4, 5\}$

It is not hard to see that the $\mathcal{L}$-language induced by $\mathcal{A}$, denoted L, is such that for all $e \in \Sigma$, we have that $L(e) = \{e, far(e)\}$ and $L(e \cdot \#) = \{e\}$. In addition, $L(\epsilon) = \top$, and for all other $w \in \Sigma^*$, we have that $L(w) = \bot$. Also, $\mathcal{A}$ is indeed deterministic, and has $m + 1$ states. Finally, since the components of $\mathcal{A}$ are all of size polynomial in the input graph, the reduction is polynomial.

In the full version we proved that G has a k-VC iff there is an LDFA with $k + 1$ states for L.

4 Minimizing an LDFA over a Fully Ordered Lattice

In Section 3, we saw that the problem of minimizing LDFAs is NP-complete. The hardness proof involved a partially ordered lattice, embodied in the "subset lattice", and strongly utilized on the order being partial. This suggests that for fully ordered lattices, we may still be able to find a polynomial minimization algorithm. On the other hand, as we show below, the property of no canonical minimal LDFA is valid already in the case of fully ordered lattice, and there is a tight connection between this and the fact we could not come up with a polynomial algorithm in the general case.

Example 2. Let $\mathcal{L} = \langle\{0,1,2,3\},\leq\rangle$, and let L be the $\mathcal{L}$-language over $\Sigma = \{0,1,2\}$, where $L(\epsilon) = 3, L(0) = 3, L(0{\cdot}0) = 1, L(1) = 1, L(1{\cdot}0) = 1, L(2) = 3, L(2{\cdot}0) = 2$, and $L(x) = 0$ for all other $x \in \Sigma^*$.

Note that L is monotonic, in the sense that for all $x, z \in \Sigma^*$, we have that $L(x \cdot z) \leq L(x)$. For monotonic $\mathcal{L}$-languages, it is tempting to consider the relation $\sim_L \subseteq \Sigma^* \times \Sigma^*$ such that $x \sim_L x'$ iff for all $z \in \Sigma^*$ there exists $l \in \mathcal{L}$ such that $L(x \cdot z) = L(x) \wedge l$ and $L(x' \cdot z) = L(x') \wedge l$. It is not hard to see, however, that $0 \sim_L 1$ and $1 \sim_L 2$, but $0 \not\sim_L 2$. Thus, even for monotonic languages over a fully ordered lattice, a relation that takes the accumulated values into account is not transitive, and there are two minimal LDFAs with different topologies for L. In the first, the letters 0 and 1 lead to a state from which the letter 0 is read with the value 1, and in the second, the letters 1 and 2 lead to a state from which 0 is read with the value 2.

In this section we show that in spite of the non-canonicality, we are able to minimize them in polynomial time. We describe a polynomial time algorithm that is given an LDFA $\mathcal{A} = \langle\mathcal{L},Q,\Sigma,\delta,Q_0,F\rangle$ over a fully ordered lattice, and returns an LDFA $\mathcal{A}_{min}$ with a minimal number of states, such that $L(\mathcal{A}_{min}) = L(\mathcal{A})$.

Let $\mathcal{L} = \{0,1,...,n-1\}$ be the fully ordered lattice, let $L : \Sigma^* \rightarrow \mathcal{L}$ be the language of $\mathcal{A}$, and let $m = max_{w\in\Sigma^*}L(w)$; that is, m is the maximal value of a word in $L(\mathcal{A})$. Finally, let $q_0 \in Q$ be the single state with initial value that is not $\perp$. For each $1 \leq i \leq m$ we define a DFA $\mathcal{A}_i$ that accepts exactly all words w such that $L(w) \geq i$. Note that it is indeed enough to consider only the automata $\mathcal{A}_1, ..., \mathcal{A}_m$, as $\mathcal{A}_{m+1}, ..., \mathcal{A}_{n-1}$ are always empty and hence not needed, and $\mathcal{A}_0$ is not needed as well, as $L(\mathcal{A}_0) = \Sigma^*$.

For $1 \leq i \leq m$, we define $\mathcal{A}_i = \langle Q_i,\Sigma,\delta_i,q_0,F_i\rangle$ as follows:

- $Q_i \subseteq Q$ contains exactly all states that are both reachable from q_0 using transitions with value at least i, and also have some state with acceptance value at least i that is reachable from them using zero or more transitions with value at least i. Note that $q_0 \in Q_i$ for all i.
- δ_i contains all transitions that their value in $\mathcal{A}$ is at least i and that both their source and destination are in Q_i.
- $F_i \subseteq Q_i$ contains all states their acceptance value in $\mathcal{A}$ is at least i.

For readers that wonder why we do not define δ_i first, as these transitions with value at least i, and then define Q_i and F_i according to reachability along δ_i, note that such a definition would result in different automata that are not *trim*, as it may involve transitions that never lead to an accepting state in $\mathcal{A}_i$, and states that are equivalent to a rejecting sink. As we will see later, the fact that all the components in our $\mathcal{A}_i$ are essential is going to be important.

Note that for all $1 < i \leq m$, we have that $Q_i \subseteq Q_{i-1}$, $\delta_i \subseteq \delta_{i-1}$, and $F_i \subseteq F_{i-1}$. Also, it is not hard to see that $\mathcal{A}_i$ indeed accepts exactly all words w such that $L(w) \geq i$.

We now turn back to the given LDFA $\mathcal{A}$ and describe how it can be minimized using $\mathcal{A}_1, ..., \mathcal{A}_m$. First, we apply a pre-processing on $\mathcal{A}$ that reduces the values appearing in $\mathcal{A}$ to be the minimal possible values (without changing the language). Formally, we define $\mathcal{A}' = \langle \mathcal{L}, Q, \Sigma, \delta', Q_0, F' \rangle$, where

- For all $q, q' \in Q$ and $\sigma \in \Sigma$, we have that $\delta'(q, \sigma, q') = max\{i : (q, \sigma, q') \in \delta_i\}$.
- For all $q \in Q$, we have that $F'(q) = max\{i : q \in F_i\}$.

Note that since for all $1 < i \leq m$, we have that $\delta_i \subseteq \delta_{i-1}$ and $F_i \subseteq F_{i-1}$, then for all $1 \leq i \leq m$, we also have that $\delta'(q, \sigma, q') \geq i$ iff $(q, \sigma, q') \in \delta_i$ and $F'(q) \geq i$ iff $q \in F_i$.

Lemma 1. $L(\mathcal{A}) = L(\mathcal{A}')$.

By Lemma 1, it is enough to minimize $\mathcal{A}'$. We start with applying the algorithm for minimizing DFA on $\mathcal{A}_1, ..., \mathcal{A}_m$. Each such application generates a partition of the states of $\mathcal{A}_i$ into equivalence classes.[5] Let us denote the equivalence classes produced for $\mathcal{A}_i$ by $\mathcal{H}_i = \{S_1^i, S_2^i, ..., S_{n_i}^i\}$.

Now, we construct from $\mathcal{A}'$ a minimal automaton $\mathcal{A}_{min} = \langle \mathcal{L}, Q_{min}, \Sigma, \delta_{min}, Q_{0_{min}}, F_{min} \rangle$ as follows.

- We obtain the set Q_{min} by partitioning the states of $\mathcal{A}'$ into sets, each inducing a state in Q_{min}. The partitioning process is iterative: we maintain a disjoint partition $\mathcal{P}_i$ of the states Q, starting with one set containing all states, and refining it along the iterations. The refinement at the i-th iteration is based on $\mathcal{H}_i$, and guarantees that the new partition $\mathcal{P}_i$ agrees with $\mathcal{H}_i$, meaning that states that are separated in $\mathcal{H}_i$ are separated in $\mathcal{P}_i$ as well. At the end of this process, the sets of the final partition constitute Q_{min}.

  More specifically, the algorithm has $m + 1$ iterations, starting with $i = 0$, ending with $i = m$. Let us denote the partition obtained at the i-th iteration by $\mathcal{P}_i = \{T_1^i, ..., T_{d_i}^i\}$. At the first iteration, for $i = 0$, we have that $d_0 = 1$, and $T_1^0 = Q$. At the i-th iteration, for $i > 0$, we are given the partition $\mathcal{P}_{i-1} = \{T_1^{i-1}, ..., T_{d_{i-1}}^{i-1}\}$, and generate $\mathcal{P}_i = \{T_1^i, ..., T_{d_i}^i\}$ as follows. For each $1 \leq j \leq d_{i-1}$, we examine T_j^{i-1} and partition it further. We do it in two stages. First, we examine $S_1^i, S_2^i, ..., S_{n_i}^i$ and for each $1 \leq k \leq n_i$ we compute the set $U_{j,k}^i = T_j^{i-1} \cap S_k^i$, and if $U_{j,k}^i \neq \emptyset$, then we add $U_{j,k}^i$ to $\mathcal{P}_i$. Thus, we indeed separate the states that are separated in $\mathcal{H}_i$. At the second stage, we consider the states in T_j^{i-1} that do not belong to $U_{j,k}^i$ for all k. Note that these states do not belong to Q_i, so $\mathcal{A}_i$ is indifferent about them. This is the stage where we have a choice in the algorithm. We choose an arbitrary k for which $U_{j,k}^i \neq \emptyset$, and add these states to $U_{j,k}^i$. If no such k exists, we know that no state in T_j^{i-1} appears in Q_i, so we have no reason to refine T_j^{i-1}, and we can add T_j^{i-1} to $\mathcal{P}_i$. Finally, we define Q_{min} to be the sets of $\mathcal{P}_m$.

[5] Note that, by definition, all the states in Q_i have some accepting state reachable from them, so the fact we do not have a rejecting state is not problematic, as such a state would have constitute a singleton state in all the partitions we are going to consider.

- The transition relation δ_{min} is defined as follows. Consider a state $T \in Q_{min}$. We choose one state $q_{rep}^T \in T$ to be a *representative* of T, as follows. Let $i_{max}^T = max\{i$: there is $q \in T$ s.t. $q \in Q_i\}$, and let q_{rep}^T be a state in $T \cap Q_{i_{max}^T}$. Note that $T \cap Q_{i_{max}^T}$ may contain more than one state, in which case we can assume Q is ordered and take the minimal. We now define the transitions leaving T according to the original transitions of q_{rep}^T in $\mathcal{A}'$. For $\sigma \in \Sigma$, let $q_{dest} \in Q$ be the σ-destination of q_{rep}^T in $\mathcal{A}'$. For all $T' \in Q_{min}$, if $q_{dest} \in T'$ we define $\delta_{min}(T, \sigma, T') = \delta'(q_{rep}^T, \sigma, q_{dest})$; otherwise, $\delta_{min}(T, \sigma, T') = 0$.
- For all $T \in Q_{min}$, if $q_0 \in T$, where q_0 is the initial state of $\mathcal{A}'$, we define $Q_{0_{min}}(T) = Q_0(q_0)$; otherwise, $Q_{0_{min}}(T) = 0$.
- For all $T \in Q$, we define $F_{min}(T) = F'(q_{rep}^T)$.

An example illustrating an execution of the algorithm can be found in the full version.

Let $L_{min} = L(\mathcal{A}_{min})$ and $L' = L(\mathcal{A}')$. We prove that the construction is correct. Thus, $L_{min} = L'$, $|\mathcal{A}_{min}|$ is minimal, and the time complexity of the construction of $\mathcal{A}_{min}$ is polynomial.

We first prove that $L_{min} = L'$. For $q, q' \in Q$, we say that $q \sim_i q'$ iff there exists some class $S \in \mathcal{H}_i$ such that $q, q' \in S$. Also, we say that $q \equiv_i q'$ iff for all $j \leq i$ it holds that $q \sim_j q'$. Note that although $\sim_i$ and $\equiv_i$ are equivalence relations over $Q_i \times Q_i$, they are not equivalence relations over $Q \times Q$, as they are not reflexive. Indeed, for $q \in Q \setminus Q_i$, there is no class $S \in \mathcal{H}_i$ such that $q \in S$, so $q \not\sim_i q$ and of course $q \not\equiv_i q$. However, it is easy to see that $\sim_i$ and $\equiv_i$ are both symmetric and transitive over $Q \times Q$.

Lemma 2 below explains the essence of the relation $\equiv_i$. As we shall prove, if $q \equiv_i q'$ then q and q' agree on the transition and acceptance values in $\mathcal{A}'$, if these values are less than i.

Lemma 2. *For $q, q' \in Q$, if $q \equiv_i q'$ then for all $j < i$, the following hold.*

- *For all $\sigma \in \Sigma$, we have $\delta'(q, \sigma, s) = j$ iff $\delta'(q', \sigma, s') = j$, where $s, s' \in Q$ are the σ-destinations of q, q' in $\mathcal{A}'$, respectively. .*
- *$F'(q) = j$ iff $F'(q') = j$.*

In the case of DFA, we know that each state of the minimal automaton for L corresponds to an equivalence class of $\sim_L$, and the minimization algorithm merges all the states of the DFA that correspond to each class into a single state. Consequently, the transition function of the minimal automaton can be defined according to one of the merged states, and the definition is independent of the state being chosen. In the case of our $\mathcal{A}_{min}$, things are more complicated, as states that are merged in $\mathcal{A}_{min}$ do not correspond to equivalence classes. We still were able, in the definition of the transitions and acceptance values, to chose a state q_{rep}^T, for each state T. Lemma 3 below explains why working with the chosen state is sound.

The lemma considers a word $w \in \Sigma^*$, and examines the connection between a state q_i in the run of $\mathcal{A}'$ on w and the corresponding state T_i in the run of $\mathcal{A}_{min}$ on w. It shows that if $L'(w) \geq l$ for some $l \in \mathcal{L}$, then $q_i \equiv_l q_{rep}^{T_i}$ for all i. Thus, the states along the run of $\mathcal{A}_{min}$ behave the same as the corresponding states in $\mathcal{A}'$ on values that are less than l, and may be different on values that are at least l, as long as they are both at least l. Intuitively, this is valid since after reaching a value l, we can replace all subsequent values $l' \geq l$ along the original run with any other value $l'' \geq l$.

Lemma 3. *Let $w = \sigma_1\sigma_2...\sigma_k$ be a word in Σ^*, and let $q_0, q_1, ..., q_k$ and $T_0, T_1, ..., T_k$ be the runs of $\mathcal{A}'$ and $\mathcal{A}_{min}$ on w respectively. For $l \in \mathcal{L}$, if $L'(w) \geq l$ then for all $0 \leq i \leq k$ it holds that $q_i \equiv_l q_{rep}^{T_i}$.*

Based on the above, we now turn to prove that $L_{min} = L'$. Let $w \in \Sigma^*$, and let $l = L'(w)$. We show that $L_{min}(w) = l$. Let $r' = q_0, q_1, ..., q_k$ and $r_{min} = T_0, T_1, ..., T_k$ be the runs of $\mathcal{A}'$ and $\mathcal{A}_{min}$ on w respectively.

We first show that $L_{min}(w) \geq l$. Consider the values read along r', which are $Q_0(q_0)$, $\delta'(q_0, \sigma_1, q_1)$, $\ldots$, $\delta'(q_{k-1}, \sigma_k, q_k)$, and $F'(q_k)$. Since $L'(w) = l$ we know that all these values are at least l. By Lemma 3 we get that for all $0 \leq i \leq k$ it holds that $q_i \equiv_l q_{rep}^{T_i}$. Then by applying the first part of Lemma 2 on $q_0, ..., q_{k-1}$ and the second part on q_k, we get that the values $\delta'(q_{rep}^{T_0}, \sigma_1, s_0)$, $\ldots$, $\delta_{min}(q_{rep}^{T_{k-1}}, \sigma_k, s_{k-1})$ and $F_{min}(q_{rep}^{T_k})$ are all at least l, where s_i is the σ_i-destination of $q_{rep}^{T_i}$ for all $0 \leq i < k$. Thus, we get that $\delta_{min}(T_0, \sigma_1, T_1), \ldots, \delta_{min}(T_{k-1}, \sigma_k, T_k)$ and $F_{min}(T_k)$ are all at least l as well, since these values are defined according to $q_{rep}^{T_0}, ..., q_{rep}^{T_k}$. Together with the fact that the initial value remains the same in $\mathcal{A}_{min}$, we get that $L_{min}(w) \geq l$.

In order to prove that $L_{min}(w) \leq l$, we show that at least one of the values read along r_{min} is l. Since $L'(w) = l$, at least one of the values read along r' is l. If this value is $Q_0(q_0)$ then we are done, since by definition $Q_0(T_0) = Q_0(q_0)$. Otherwise, it must be one of the values $\delta'(q_0, \sigma_1, q_1), \ldots, \delta'(q_{k-1}, \sigma_k, q_k)$ or $F'(q_k)$. Let q_d be the state from which the value l is read for the first time along r', either as a transition value ($d < k$) or as an acceptance value ($d = k$). We claim that $q_d \in Q_{l+1}$ (note that if $l = m$, then clearly $L_{min}(w) \leq l$, thus, we assume that $l < m$, so Q_{l+1} is well defined). If $d = 0$, then we are done, since $q_0 \in Q_{l+1}$ by definition. Otherwise, we look at the transition (q_{d-1}, σ_d, q_d). By the definition of q_d, we know that $\delta'(q_{d-1}, \sigma_d, q_d) \geq l+1$, and by the definition of δ' it then follows that $(q_{d-1}, \sigma_d, q_d) \in \delta_{l+1}$. Thus, we get that $q_d \in Q_{l+1}$. Now, by the definition of Q_{l+1}, there exists some state $q_{acc}^{l+1} \in Q$ with acceptance value at least $l + 1$ that is reachable from q_d in $\mathcal{A}$ using zero or more transitions with value at least $l + 1$. Let w' be the word read along these transitions from q_d to q_{acc}^{l+1}, and let $w'' = \sigma_1, ..., \sigma_d \cdot w'$. It is easy to see that $L'(w'') \geq l+1$. Thus, we can apply Lemma 3, and get that $q_d \equiv_{l+1} q_{rep}^{T_d}$. Then, by applying Lemma 2 on q_d and $q_{rep}^{T_d}$ we conclude that the value l is read from T_d along r_{min}, and we are done.

We now turn to prove that $|\mathcal{A}_{min}|$ is minimal. Let $N = |\mathcal{A}_{min}|$. We describe below N different words $w_1, ..., w_N \in \Sigma^*$, and prove that for all $i \neq j$ the words w_i and w_j cannot reach the same state in an LDFA for L. Clearly, this implies that an LDFA for L must contain at least N states, so $|\mathcal{A}_{min}|$ is indeed minimal.

We define the words $w_1, ..., w_N$ as follows. Let $T_1, ..., T_N$ be the states of $\mathcal{A}_{min}$, and let $q_{rep}^{T_1}, ..., q_{rep}^{T_N}$ be their representatives respectively. We go back to the original automaton $\mathcal{A}$, and for each such representative q, we define the following:

- $reach(q) = \{w \in \Sigma^* : \delta(q_0, w, q) > 0\}$, where q_0 is the initial state of $\mathcal{A}$.
- $maxval(q) = max\{\delta(q_0, w, q) : w \in reach(q)\}$. Note that $maxval(q)$ considers only the traversal values of the words reaching q.
- $maxw(q)$ is $w \in \Sigma^*$ for which $\delta(q_0, w, q) = maxval(q)$. Note that there may be several such words, so we can take the lowest one by lexicographic order, to make it well defined.

For all $1 \leq i \leq N$, we now define $w_i = maxw(q^{T_i}_{rep})$. Note that these words are indeed different, as they reach different states in the deterministic automaton $\mathcal{A}$. Consider two different indices i and j. We prove that the words $maxw(q^{T_i}_{rep})$ and $maxw(q^{T_j}_{rep})$ cannot reach the same state in an LDFA for L. Consider the states $q^{T_i}_{rep}$ and $q^{T_j}_{rep}$. These states belong to different sets in Q_{min}. Let $1 \leq l \leq m$ be the index of the iteration in which they were first separated.

We use the following lemmas:

Lemma 4. *The states $q^{T_i}_{rep}$ and $q^{T_j}_{rep}$ belong to different classes in $\mathcal{H}_l$.*

Lemma 5. *$tr\_val(maxw(q^{T_i}_{rep})) \geq l$ and $tr\_val(maxw(q^{T_j}_{rep})) \geq l$ in all LDFA for L.*

Based on the above, we prove that the words $maxw(q^{T_i}_{rep})$ and $maxw(q^{T_j}_{rep})$ cannot reach the same state in an LDFA for L. By Lemma 4, there is a distinguishing tail $z \in \Sigma^*$. That is, without loss of generality, z is read in $\mathcal{A}$ from $q^{T_i}_{rep}$ with value at least l, and is read from $q^{T_j}_{rep}$ with value less than l. Let us examine the words $maxw(q^{T_i}_{rep}) \cdot z$ and $maxw(q^{T_j}_{rep}) \cdot z$. By applying Lemma 5 on $\mathcal{A}$, we get that $L(\mathcal{A})(maxw(q^{T_i}_{rep}) \cdot z) \geq l$ and that $L(\mathcal{A})(maxw(q^{T_j}_{rep}) \cdot z) < l$. Now, let $\mathcal{U}$ be an LDFA for L, and assume by way of contradiction that $maxw(q^{T_i}_{rep})$ and $maxw(q^{T_j}_{rep})$ are reaching the same state in $\mathcal{U}$. Let q be that state. Applying Lemma 5 on $\mathcal{U}$, we get that both $maxw(q^{T_i}_{rep})$ and $maxw(q^{T_j}_{rep})$ are reaching q with traversal value at least l. Now, let us examine the value v_z with which z is read from q. If $v_z \geq l$, then $L(\mathcal{U})(maxw(q^{T_j}_{rep}) \cdot z) \geq l$, which contradicts the fact that $L(\mathcal{A})(maxw(q^{T_j}_{rep}) \cdot z) < l$. On the other hand, if $v_z < l$, then $L(\mathcal{U})(maxw(q^{T_i}_{rep}) \cdot z) < l$, which contradicts the fact that $L(\mathcal{A})(maxw(q^{T_i}_{rep}) \cdot z) \geq l$.

Thus, we conclude that $|\mathcal{A}_{min}|$ is minimal.

It is not hard to see that each of the three stages of the algorithm (constructing the automata $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$ and minimizing them, generating $\mathcal{A}'$ from $\mathcal{A}$, and constructing $\mathcal{A}_{min}$ from $\mathcal{A}'$) can be implemented in polynomial time. In the full version we analyse the complexity in detail, and show that the overall complexity is $O(|\mathcal{L}|(|Q| \log |Q| + |\delta|))$.

We can now conclude with the following.

Theorem 2. *An LDFA over a fully ordered lattice can be minimized in polynomial time.*

References

1. Alur, R., Kanade, A., Weiss, G.: Ranking automata and games for prioritized requirements. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 240–253. Springer, Heidelberg (2008)
2. ESF Network programme. Automata: from mathematics to applications (AutoMathA) (2010), `http://www.esf.org/index.php?id=1789`
3. Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 274–287. Springer, Heidelberg (1999)

4. Bruns, G., Godefroid, P.: Temporal logic query checking. In: Proc. 16th LICS, pp. 409–420. IEEE Computer Society, Los Alamitos (2001)
5. Chan, W.: Temporal-logic queries. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 450–463. Springer, Heidelberg (2000)
6. Chechik, M., Devereux, B., Gurfinkel, A.: Model-checking infinite state-space systems with fine-grained abstractions using SPIN. In: Dwyer, M.B. (ed.) SPIN 2001. LNCS, vol. 2057, pp. 16–36. Springer, Heidelberg (2001)
7. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. Springer, Heidelberg (2009)
8. Easterbrook, S., Chechik, M.: A framework for multi-valued reasoning over inconsistent viewpoints. In: Proc. 23rd Int. Conf. on Software Engineering, pp. 411–420. IEEE Computer Society Press, Los Alamitos (2001)
9. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
10. Henzinger, T.A.: From boolean to quantitative notions of correctness. In: Proc. 37th POPL, pp. 157–158 (2010)
11. Hussain, A., Huth, M.: On model checking multiple hybrid views. Technical Report TR-2004-6, University of Cyprus (2004)
12. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103 (1972)
13. Kupferman, O., Lustig, Y.: Lattice automata. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 199–213. Springer, Heidelberg (2007)
14. Kupferman, O., Lustig, Y.: Latticed simulation relations and games. International Journal on the Foundations of Computer Science 21(2), 167–189 (2010)
15. Kirsten, D., Mäurer, I.: On the determinization of weighted automata. Journal of Automata, Languages and Combinatorics 10(2/3), 287–312 (2005)
16. Krob, D.: The equality problem for rational series with multiplicities in the tropical emiring is undecidable. Journal of Algebra and Computation 4, 405–425 (1994)
17. Li, Y., Pedrycz, W.: Minimization of lattice finite automata and its application to the decomposition of lattice languages. Fuzzy Sets and Systems 158(13), 1423–1436 (2007)
18. Mohri, M.: Finite-state transducers in language and speech processing. Computational Linguistics 23(2), 269–311 (1997)
19. Malik, D.S., Mordeson, J.N., Sen, M.K.: Minimization of fuzzy finite automata. Information Sciences 113, 323–330 (1999)
20. Myhill, J.: Finite automata and the representation of events. Technical Report WADD TR-57-624, pp. 112–137, Wright Patterson AFB, Ohio (1957)
21. Nerode, A.: Linear automaton transformations. Proceedings of the American Mathematical Society 9(4), 541–544 (1958)
22. Zekai, L., Lan, S.: Minimization of lattice automata. In: Proc. 2nd ICFIE, pp. 194–205 (2007)