Theoretical Computer Science 411 (2010) 3444–3459

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Fixed point guided abstraction refinement for alternating automata☆

Pierre Ganty[a], Nicolas Maquet[b,*], Jean-François Raskin[b]

[a] Imdea Software, Madrid, Spain

[b] Université Libre de Bruxelles (ULB), Belgium

ARTICLE INFO

Keywords:
Alternating automata
Abstract interpretation
Antichains

ABSTRACT

In this paper, we develop and evaluate two new algorithms for checking emptiness of alternating automata. These algorithms build on previous works. First, they rely on antichains to efficiently manipulate the state-spaces underlying the analysis of alternating automata. Second, they are abstract algorithms with built-in refinement operators based on techniques that exploit information computed by abstract fixed points (and not counter-examples as it is usually the case). The efficiency of our new algorithms is illustrated by experimental results.

1. Introduction

Alternating automata are a generalization of both nondeterministic and universal automata. In an alternating automaton, the transition relation is defined using positive Boolean formulas: disjunctions allow for the expression of nondeterministic transitions and conjunctions allow for the expression of universal transitions. The emptiness problem for alternating automata being PSPACE-COMPLETE [3], several computationally hard automata-theoretic and model-checking problems can be reduced in polynomial time to the emptiness problem for those automata. Here are some illustrative examples: the emptiness problem for a product of n nondeterministic automata, the language inclusion between two nondeterministic automata, or the LTL model-checking problem, can all be reduced in linear time to the emptiness problem for alternating automata. It is thus very desirable to design efficient algorithms for checking emptiness of those automata. In this paper, we propose new algorithms for efficiently checking the emptiness problem for alternating automata over finite words. Those new algorithms combine two recent lines of research.

First, we use efficient techniques based on *antichains*, initially introduced in [8], to symbolically manipulate the state-spaces underlying the analysis of alternating automata. Antichain-based techniques have been applied to several problems in automata theory [8,10,11,1] and for solving games of imperfect information [15]. For example, in [11], we show how to efficiently solve the language inclusion problem between nondeterministic Büchi automata by exploiting the structures of the underlying automata-based constructions. Automata that were out of reach of existing algorithms can be treated with these new antichain algorithms; see also [12] for new developments on that problem. Those techniques have also been applied with success to the satisfiability and model-checking of LTL specifications [10]. Our team has implemented these algorithms in a tool called ALASKA [9], which is available for download.[1]

☆ Work supported by the projects: (i) Quasimodo: "Quantitative System Properties in Model-Driven-Design of Embedded System", http://www.quasimodo.aau.dk/, (ii) Gasics: "Games for Analysis and Synthesis of Interactive Computational Systems", http://www.ulb.ac.be/di/gasics/, (iii) Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", http://moves.ulb.ac.be, a PAI program funded by the Federal Belgian Government, (iv) CFV (Federated Center in Verification) funded by the FNRS http://www.ulb.ac.be/di/ssd/cfv/, (v) EU People/COFUND program 229599 AMAROUT, (vi) EU IST FET Project 231620 HATS, and (vii) Madrid Regional Government project S2009TIC-1465 PROMETIDOS.

* Corresponding author. Tel.: +32 26505595; fax: +32 26505609.
E-mail addresses: pierre.ganty@imdea.org (P. Ganty), nmaquet@ulb.ac.be (N. Maquet), jfraskin@ulb.ac.be (J.-F. Raskin).

[1] See http://www.antichains.be.

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.05.037

Second, to apply this antichain technique to even larger instances of alternating automata, we instantiate a generic abstract refinement method that we have proposed in [7] and further developed in [13,14]. This abstract refinement method does not use counter-examples to refine inconclusive abstractions contrary to most of the methods presented and implemented in the literature; see for example [4]. Instead, our algorithm uses the entire information computed by the abstract analysis and combines it with information obtained by one application of a concrete predicate transformer. The algorithm presented in [7] is a generic solution that does not lead directly to efficient implementations. In particular, as shown in [13], in order to obtain an efficient implementation of this algorithm, we need to define a family of abstract domains on which abstract analysis can be effectively computed, as well as practical operators to refine the elements of this family of abstract domains. In this paper, we use the set of *partitions* of the locations of an alternating automaton to define the family of abstract domains. Those abstract domains and their refinement operators can be used both in *forward* and *backward* algorithms for checking emptiness of alternating automata.

To show the practical interest of these new algorithms, we have implemented them into the ALASKA tool. We illustrate the efficiency of our new algorithms on examples of alternating automata constructed from LTL specifications interpreted over finite words. With the help of these examples, we show that our algorithms are able to concentrate the analysis on the important parts of the state-space and abstract away the less interesting parts *automatically*. This allows us to treat much larger instances than with the concrete forward or backward algorithms. We are confident that those new algorithms will allow us to solve problems of practical relevance that are currently out of reach of automatic methods.

Structure of the paper. In Section 2, we recall some important notions about alternating automata and about the lattice of partitions. In Section 3, we recall the basis for antichain algorithms and their application to the emptiness of alternating automata. In Section 4, we develop an adequate family of abstract domains based on the lattice of partitions along with the tools to refine elements of this family. In Section 5, we present our abstract forward and backward algorithms with refinement. In Section 6, we report on experiments that illustrate the efficiency of our algorithms. Finally, we draw some conclusions and evaluate future directions in Section 7.

2. Preliminaries

Alternating Automata. Let S be a set. We denote by $\mathcal{B}^+(S)$ the set of *positive Boolean formulas* over S, which are the formulas of the form $\phi ::= s \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2$, where $s \in S$. A *valuation* for a set of propositions S is encoded as a subset of S. For each formula $\phi \in \mathcal{B}^+(S)$, we use $[\![\phi]\!] \subseteq 2^S$ to denote the set of valuations that satisfy ϕ; as usual, $c \in [\![\phi]\!]$ is interpreted as the valuation that assigns "true" precisely to the variables in c. Let Σ be a finite alphabet. A finite word w is a finite sequence $w = \sigma_1\sigma_2 \ldots \sigma_n$ of letters from Σ. We write Σ^* the set of finite words over Σ. We now recall the definition of *alternating finite automata over finite words*.

Definition 1. An *alternating finite automaton* (AFA for short) is a tuple $\langle \mathsf{Loc}, \Sigma, l_0, \delta, F\rangle$ where: $\mathsf{Loc} = \{l_0, \ldots, l_n\}$ is the set of locations; $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ is the set of alphabet symbols; $l_0 \in \mathsf{Loc}$ is the initial location; $\delta\colon \mathsf{Loc} \times \Sigma \to B^+(\mathsf{Loc})$ is the transition function; and $F \subseteq \mathsf{Loc}$ is the set of accepting locations.

As we will often manipulate sets of sets of locations in what follows, we will refer to the inner sets as *cells*. Let $\mathsf{Cells}(S) = 2^S$. A *cell* of an AFA with locations Loc is an element of $\mathsf{Cells}(\mathsf{Loc})$. For any set $X \subseteq \mathsf{Cells}(\mathsf{Loc})$, $\overline{X} \equiv \mathsf{Cells}(\mathsf{Loc}) \setminus X$ is the *complement* of X. Instead of defining the traditional notion of runs for AFA, we define their semantics as a *directed graph*, the nodes of which are cells. Each edge in the cell graph is labeled by an alphabet symbol.

Remark. In what follows, we denote cells by lowercase letters (usually c) and sets of cells by upper-case letters (usually X or Y).

Definition 2. Let $A = \langle \mathsf{Loc}, \Sigma, l_0, \delta, F\rangle$. Then $[\![A]\!] = \langle V, E\rangle$ where: $V = \mathsf{Cells}(\mathsf{Loc})$ and $\langle c, \sigma, c'\rangle \in E$ iff $c' \in [\![\bigwedge_{l\in c} \delta(l, \sigma)]\!]$. A word $w = \sigma_1\sigma_2 \ldots \sigma_n$ is *accepted* by the automaton A iff there exists a path $c_0, c_1, \ldots, c_n$ of cells of V such that $l_0 \in c_0$, $c_n \in \mathsf{Cells}(F)$ (the set of *accepting cells*), and $\forall i \in \{1, \ldots, n\}\colon \langle c_{i-1}, \sigma_i, c_i\rangle \in E$. Define $L(A)$ to be the set of words accepted by A.

In what follows, we will consider $[\![A]\!]$ simply as the set of edges E of the cell graph and leave the set of vertices V implicit.

Predicate Transformers. We have defined the semantics of alternating automata as a directed graph of cells. To explore this graph, we use *predicate transformers* of type $2^{\mathsf{Cells}(\mathsf{Loc})} \to 2^{\mathsf{Cells}(\mathsf{Loc})}$.

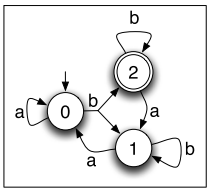
Definition 3. We consider the following *predicate transformers* (A is an AFA):

$$post_\sigma[A](X) = \{c_2 \mid \exists\langle c_1, \sigma, c_2\rangle \in [\![A]\!]\colon c_1 \in X\}\ post[A](X) = \bigcup_{\sigma\in\Sigma} post_\sigma[A](X)$$

$$\widetilde{post}_\sigma[A](X) = \{c_2 \mid \forall\langle c_1, \sigma, c_2\rangle \in [\![A]\!]\colon c_1 \in X\}\ \widetilde{post}[A](X) = \bigcap_{\sigma\in\Sigma} \widetilde{post}_\sigma[A](X)$$

$$pre_\sigma[A](X) = \{c_1 \mid \exists\langle c_1, \sigma, c_2\rangle \in [\![A]\!]\colon c_2 \in X\}\ pre[A](X) = \bigcup_{\sigma\in\Sigma} pre_\sigma[A](X)$$

$$\widetilde{pre}_\sigma[A](X) = \{c_1 \mid \forall\langle c_1, \sigma, c_2\rangle \in [\![A]\!]\colon c_2 \in X\}\ \widetilde{pre}[A](X) = \bigcap_{\sigma\in\Sigma} \widetilde{pre}_\sigma[A](X).$$



	$\{\}$	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$
$\{\}$	a, b	a, b	a, b	a, b	a, b	a, b	a, b	a, b
$\{0\}$	-	a	-	-	a	a	b	a, b
$\{1\}$	-	a	b	-	a, b	a	b	a, b
$\{2\}$	-	-	a	b	a	b	a, b	a, b
$\{0,1\}$	-	a	-	-	a	a	b	a, b
$\{0,2\}$	-	-	-	-	a	-	b	a, b
$\{1,2\}$	-	-	-	-	a	-	b	a, b
$\{0,1,2\}$	-	-	-	-	a	-	b	a, b

Fig. 1. An example AFA A (top), and the adjacency matrix for $[\![A]\!]$ (bottom).

Example 4. Since $\widetilde{pre}$ and $\widetilde{post}$ are somewhat less common that *pre* and *post*, we illustrate them on the example AFA of Fig. 1. The $\widetilde{pre}[A](X)$ operation computes the set of cells which have a set of successors that is included in X. For instance, $\widetilde{pre}[A](\{\{0,1\},\{1,2\},\{0,1,2\}\}) = \{\{0,2\},\{1,2\},\{0,1,2\}\}$. Similarly, $\widetilde{post}[A](X)$ is the set of cells which have a set of predecessors that is included in X. For instance, $\widetilde{post}[A](\{\{1,2\},\{1\},\{2\},\{\}\}) = \{\{1\},\{2\},\{\}\}$.

These predicate transformers are actually two pairs which are *dual of each other*, as expressed in the following lemma.

Lemma 5 (*From [6]*). *For any* AFA A *with locations* Loc*, for any* $X \subseteq \mathsf{Cells}(\mathsf{Loc})$*, we have that* $\widetilde{post}[A](X) = \overline{post[A](\overline{X})}$ *and* $\widetilde{pre}[A](X) = \overline{pre[A](\overline{X})}$.

Let f be a function over a poset $(L, \sqsubseteq)$. A *fixpoint* of f is an element $l \in L$ such that $f(l) = l$. We denote by $\mu X \cdot f(X)$ and $\nu X \cdot f(X)$, respectively, the *least* and the *greatest fixpoint*, when they exist, of f. The well-known Knaster–Tarski's theorem states that each monotone function $f \in L \mapsto L$ over a complete lattice $\langle L, \sqsubseteq, \sqcup, \sqcap, \top, \bot\rangle$ admits a least fixpoint and the following characterization holds:

$$\mu X \cdot f(X) = \bigsqcap\{X \in L \mid f(X) \sqsubseteq X\}. \tag{1}$$

Dually, f also admits a greatest fixpoint and the following characterization holds:

$$\nu X \cdot f(X) = \bigsqcup\{X \in L \mid X \sqsubseteq f(X)\}. \tag{2}$$

It is well known that predicate transformers can be used to solve automata emptiness using *fixed point expressions*, which operate either in a *backward* or *forward* fashion. This is summarized in the following theorem:

Theorem 6. *Let* $A = \langle \mathsf{Loc}, \Sigma, l_0, \delta, F\rangle$ *an* AFA, $\mathcal{F} = \mathsf{Cells}(F)$.
$L(A) = \emptyset$ *iff* $(\mu X \cdot post[A](X) \cup [\![l_0]\!]) \subseteq \overline{\mathcal{F}}$ *iff* $(\mu X \cdot pre[A](X) \cup \mathcal{F}) \subseteq \overline{[\![l_0]\!]}$.

Lemma 5 shows that our four predicate transformers are actually two pair of dual functions. The fixed point expressions of Theorem 6 exhibit a similar duality, as shown by the following lemma.

Lemma 7 (*From [6]*). *For every* AFA $A = \langle \mathsf{Loc}, \Sigma, l_0, \delta, F\rangle$ *we have that:*

- $(\nu X \cdot \widetilde{post}[A](x) \cap \overline{[\![l_0]\!]}) = \overline{(\mu X \cdot post[A](X) \cup [\![l_0]\!])}$;
- $(\nu X \cdot \widetilde{pre}[A](x) \cap \overline{\mathsf{Cells}(F)}) = \overline{(\mu X \cdot pre[A](X) \cup \mathsf{Cells}(F))}$.

Note that this lemma is of practical importance because least fixed points based on *pre* and *post* can be evaluated efficiently with the use of a *frontier*. Since we have that $post[A](X \cup Y) = post[A](X) \cup post[A](Y)$ and $pre[A](X \cup Y) = pre[A](X) \cup pre[A](Y)$, for any X, Y, we can safely avoid to recompute *pre* and *post* over previously encountered cells.

The lattice of partitions. The heart of our abstraction scheme is to *partition* the set of locations Loc of an AFA in order to build a smaller (hopefully more tractable) automaton. We recall the notion of partitions and some of their properties. Let $\mathcal{P}$ be a partition of the set $S = \{l_0, \ldots, l_n\}$ into k *classes* (called *blocks* in the sequel) $\mathcal{P} = \{b_1, \ldots, b_k\}$. Partitions are classically ordered as follows: $\mathcal{P}_1 \preceq \mathcal{P}_2$ (read $\mathcal{P}_1$ *refines* $\mathcal{P}_2$) iff $\forall\, b_1 \in \mathcal{P}_1, \exists\, b_2 \in \mathcal{P}_2 : b_1 \subseteq b_2$. It is well known, see [2], that the set of partitions together with $\preceq$ form a complete lattice where $\{\{l_0\}, \ldots, \{l_n\}\}$ is the $\preceq$-minimal element, $\{\{l_0, \ldots, l_n\}\}$ is the $\preceq$-maximal element and the greatest lower bound of two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$, noted $\mathcal{P}_1 \curlywedge \mathcal{P}_2$, is the partition given by $\{b \neq \emptyset \mid \exists\, b_1 \in \mathcal{P}_1, \exists\, b_2 \in \mathcal{P}_2 : b = b_1 \cap b_2\}$. The least upper bound of two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$, noted $\mathcal{P}_1 \curlyvee \mathcal{P}_2$, is the finest partition such that given $b \in \mathcal{P}_1 \cup \mathcal{P}_2$, for all $l_i \neq l_j : l_i \in b$ and $l_j \in b$ we have: $\exists\, b' \in \mathcal{P}_1 \curlyvee \mathcal{P}_2 : l_i \in b'$ and $l_j \in b'$. Note that by the above definitions we have that $\mathcal{P}_1 \preceq \mathcal{P}_2$ iff there exists a partition $\mathcal{P}$ such that $\mathcal{P}_1 \curlyvee \mathcal{P} = \mathcal{P}_2$. Also, we shall use $\mathcal{P}$ as a function such that $\mathcal{P}(l)$ simply returns the block b to which l belongs in $\mathcal{P}$.

Example 8. Given the set $S = \{a, b, c\}$ and two partitions $A_1 = \{\{a, b\}, \{c\}\}$ and $A_2 = \{\{a, c\}, \{b\}\}$. We have that $A_1 \curlywedge A_2 = \{\{a\}, \{b\}, \{c\}\}$, $A_1 \curlyvee A_2 = \{\{a, b, c\}\}$, and $A_2(a) = \{a, c\}$.

3. Deciding AFA emptiness using antichains

A fundamental problem regarding AFA is the *emptiness problem*; i.e., to decide if there exists at least one word accepted by an AFA. It is well known that this decision problem is PSPACE-COMPLETE [3]. Since nondeterministic automata (NFA, for short) emptiness can be solved in linear time, a natural solution is to first perform an AFA $\rightarrow$ NFA translation and then check for emptiness. The translation is simple (albeit computationally difficult), as it amounts to a subset construction, similar to that of NFA determinization. Notice that the cell-graph semantics of AFA defined in the previous section is essentially an NFA obtained by subset construction.

In earlier works [8,10,11], we have designed new efficient algorithms for AFA emptiness. Those algorithms are based on efficient manipulations of $\subseteq$-upward- or downward-closed sets of cells using *antichains*. In our context, an antichain is a set of $\subseteq$-incomparable cells which uniquely represents both its upward- and downward-closures. Formally, for any finite set S, an antichain over S is a set $A \subseteq \mathsf{Cells}(S)$ such that $\forall c_1, c_2 \in A : c_1 \subseteq c_2 \Rightarrow c_1 = c_2$. The crucial properties of antichains are that (i) they are canonical representations of $\subseteq$-closed sets of cells, (ii) the predicate transformers on AFA evaluate to $\subseteq$-closed sets (they can thus be canonically represented with antichains) and, (iii) evaluating a predicate transformer on any set of cells is equivalent to evaluating it on the $\subseteq$-closure of that set (we can thus evaluate predicate transformers *directly* on antichains, without losing any information). In the remainder of this section, we shall recall the framework of antichains, applied to AFA emptiness.

Order relation on $\mathsf{Cells}(\cdot)$ *and antichains.* Let D be some finite domain. We define the *upward-closure* of $X \subseteq \mathsf{Cells}(D)$ as $\uparrow X = \{c \in \mathsf{Cells}(D) \mid \exists\, c' \in X : c \supseteq c'\}$. A set $X \subseteq \mathsf{Cells}(D)$ is *upward-closed* iff $X = \uparrow X$. The *downward-closure* of X is $\downarrow X = \{c \in \mathsf{Cells}(D) \mid \exists\, c' \in X : c \subseteq c'\}$. The set X is *downward-closed* iff $X = \downarrow X$. For every set $X \subseteq \mathsf{Cells}(D)$, there exists a unique set of *minimal elements* $\lfloor X \rfloor = \{c \in X \mid \nexists\, c' \in X : c' \subset c\}$. Likewise, there exists a unique set of *maximal elements* $\lceil X \rceil = \{c \in X \mid \nexists\, c' \in X : c' \supset c\}$. Both sets $\lfloor X \rfloor$ and $\lceil X \rceil$ are *antichains* and they *canonically represent* their upward- and downward-closure, respectively. In fact, $\uparrow X = \uparrow \lfloor X \rfloor$ and $\downarrow X = \downarrow \lceil X \rceil$. Also, it is well known that if X is upward-closed, then $\overline{X}$ is downward-closed, and vice versa.

Antichain and predicate transformers. Lemmas 9 and 10 below formalize the strong affinity that each of the four previously defined predicate transformers have with $\subseteq$-closed sets. First, their output is always $\subseteq$-closed, for all inputs; and second, evaluating them on any set of cells is equivalent to evaluating them on the appropriate $\subseteq$-closure of that set.

Remark. In this work, we do not provide implementation-level details on how the predicate transformers are computed or how to compute the antichain representing a closed set of cells. Such information can be found in [10,9].

Lemma 9. *Let A be an* AFA *with locations* Loc*, and let $X \subseteq \mathsf{Cells}(\mathsf{Loc})$ we have that $post[A](X)$ and $\widetilde{pre}[A](X)$ are upward-closed, and that $pre[A](X)$ and $\widetilde{post}[A](X)$ are downward-closed.*

Proof. We prove each property in turn. First, let $c' \in post[A](X)$. We know that there exists some $c \in X$ and $\sigma \in \Sigma$ such that $\langle c, \sigma, c' \rangle \in [\![A]\!]$ with $c' \in [\![\bigwedge_{l \in c} \delta(l, \sigma)]\!]$. Clearly, any other element of $\mathsf{Cells}(\mathsf{Loc})$ which includes c' will also be a member of $[\![\bigwedge_{l \in c} \delta(l, \sigma)]\!]$, thus $post[A](X)$ is upward-closed. Second, let $c \in pre[A](X)$. Just like before, we know that there exists some $c' \in X$ and $\sigma \in \Sigma$ such that $\langle c, \sigma, c' \rangle \in [\![A]\!]$ with $c' \in [\![\bigwedge_{l \in c} \delta(l, \sigma)]\!]$. Clearly, for any other $c'' \subseteq c$ we have that $c' \in [\![\bigwedge_{l \in c''} \delta(l, \sigma)]\!]$, thus $pre[A](X)$ is downward-closed. Finally, because $\widetilde{post}$ is the complement of some *post* and $\widetilde{pre}$ is the complement of some *pre*, they are clearly downward- and upward-closed, respectively. □

Lemma 10. *Let A be an* AFA *with locations* Loc*, and let $X \subseteq \mathsf{Cells}(\mathsf{Loc})$. We have that $post[A](X) = post[A](\uparrow X)$, $\widetilde{post}[A](X) = \widetilde{post}[A](\downarrow X)$, $pre[A](X) = pre[A](\downarrow X)$, and $\widetilde{pre}[A](X) = \widetilde{pre}[A](\uparrow X)$.*

Proof. Let $c, c' \in \mathsf{Cells}(\mathsf{Loc})$ such that $c \subseteq c'$. We conclude from $\delta : \mathsf{Loc} \times \Sigma \rightarrow B^+(\mathsf{Loc})$ that $\forall \sigma \in \Sigma : [\![\bigwedge_{l' \in c'} \delta(l', \sigma)]\!] \subseteq [\![\bigwedge_{l \in c} \delta(l, \sigma)]\!]$, hence that $post[A](\{c'\}) \subseteq post[A](\{c\})$ by definition of *post*, and, finally, that $post[A](\uparrow X) \subseteq post[A](X)$ since $X \subseteq \uparrow X$. The reverse inclusion follows by monotonicity. Let $c, c' \in \mathsf{Cells}(\mathsf{Loc})$ such that $c \subseteq c'$, we conclude from $\delta : \mathsf{Loc} \times \Sigma \rightarrow B^+(\mathsf{Loc})$ that:

$$\left\{ c_1 \mid \forall \sigma \in \Sigma : \left[\!\!\left[\bigwedge_{l \in c_1} \delta(l, \sigma) \right]\!\!\right] \subseteq c' \right\} \subseteq \left\{ c_2 \mid \forall \sigma \in \Sigma : \left[\!\!\left[\bigwedge_{l \in c_2} \delta(l, \sigma) \right]\!\!\right] \subseteq c \right\}$$

hence that $\widetilde{pre}[A](\{c'\}) \subseteq \widetilde{pre}[A](\{c\})$ by definition of $\widetilde{pre}$ and finally that $\widetilde{pre}[A](\uparrow X) \subseteq \widetilde{pre}[A](X)$ since $X \subseteq \uparrow X$. The reverse inclusion follows by monotonicity. Similar arguments hold for $\widetilde{post}$ and *pre*, which we omit here. □

The following corollary is a formal summary of the relationships between antichains, $\subseteq$-closed sets and predicate transformers.

Corollary 11. *Let A be an* AFA *with locations* Loc*, and let $X \in \mathsf{Cells}(\mathsf{Loc})$, we have*

$$post[A](X) = \uparrow \lfloor post[A](\lfloor X \rfloor) \rfloor \quad \text{and} \quad \widetilde{pre}[A](X) = \uparrow \lfloor \widetilde{pre}[A](\lfloor X \rfloor) \rfloor$$
$$pre[A](X) = \downarrow \lceil preA(\lceil X \rceil) \rceil \quad \text{and} \quad \widetilde{post}[A](X) = \downarrow \lceil \widetilde{post}[A](\lceil X \rceil) \rceil \,.$$

Proof. We show this for *post* and leave the remaining cases to the reader. First, by Lemma 10 we have that $post[A](X) = post[A](\uparrow X)$ which implies that $post[A](X) = post[A](\lfloor X \rfloor)$, since $\uparrow\lfloor X\rfloor = \uparrow X$ for any set X. Second, by Lemma 9 we know that $post[A](X)$ is an upward-closed set, so we obtain $post[A](X) = \uparrow\lfloor post[A](\lfloor X\rfloor)\rfloor$. □

Efficient computation on antichains representation. We now give the algorithms to efficiently perform set-theoretic operations directly over antichains representations. Let D be some finite domain. Let $X, Y \subseteq \mathsf{Cells}(D)$ be two antichains of *minimal elements* representing the upward-closed sets $\uparrow X$ and $\uparrow Y$. We denote by $X \sqcup Y$ and $X \sqcap Y$ the (unique) antichain such that $\uparrow(X \sqcup Y) = \uparrow X \cup \uparrow Y$ and $\uparrow(X \sqcap Y) = \uparrow X \cap \uparrow Y$, respectively. Those antichains can be efficiently computed as follows: $X \sqcup Y = \lfloor X \cup Y \rfloor$, and $X \sqcap Y = \lfloor \{x \cup y \mid x \in X \wedge y \in Y\}\rfloor$. We write $X \sqsubseteq Y$ when $\uparrow X \subseteq \uparrow Y$, which holds iff $\forall x \in X : \exists y \in Y : y \subseteq x$ giving a way to efficiently check inclusion between upward-closed sets represented by their antichains. Symmetrically, if X and Y are two antichains of *maximal elements* representing sets $\downarrow X$ and $\downarrow Y$, then we denote by $X \sqcup Y$ and $X \sqcap Y$ the unique antichain such that $\downarrow(X \sqcup Y) = \downarrow X \cup \downarrow Y$ and $\downarrow(X \sqcap Y) = \downarrow X \cap \downarrow Y$, respectively. Those antichains can be efficiently computed as follows: $X \sqcup Y = \lceil X \cup Y\rceil$, and $X \sqcap Y = \lceil \{x \cap y \mid x \in X \wedge y \in Y\}\rceil$. For maximal antichains, $X \sqsubseteq Y$ holds when $\downarrow X \subseteq \downarrow Y$, which is equivalent to $\forall x \in X : \exists y \in Y : x \subseteq y$. Note that all these operations have linear or quadratic complexity in the size of their input while the sets that are presented are usually exponential in the size of those inputs.

The definitions above show that we can use the *proxy operators* $\sqcup$ and $\sqcap$ and *proxy predicate* $\sqsubseteq$ instead of $\cup$, $\cap$ and $\subseteq$ in all the computations over antichains, provided that we do not mix minimal- and maximal antichains. This enables us to use antichains to evaluate more efficiently the fixed point expressions of Theorem 6. Notice that $[\![l_0]\!]$ and $\mathsf{Cells}(F)$ are respectively upward- and downward-closed sets of cells. Also, $\lceil\mathsf{Cells}(F)\rceil = \{\{F\}\}$, $\lfloor\overline{\mathsf{Cells}(F)}\rfloor = \{\{l\} \mid l \notin F\}$, $\lfloor[\![l_0]\!]\rfloor = \{\{l_0\}\}$, and $\lceil\overline{[\![l_0]\!]}\rceil = \{\mathsf{Loc} \setminus \{l_0\}\}$, all of which are antichains of linear size w.r.t. the AFA. We can now rewrite the fixed point expressions of Theorem 6 to exploit the properties of antichains.

Theorem 12 (*From [8]*). *Let $A = \langle \mathsf{Loc}, \Sigma, l_0, \delta, F\rangle$ be an* AFA. *The following statements are equivalent:*

- $L(A) = \emptyset$
- $(\mu X \cdot \lfloor post[A](X)\rfloor \sqcup \lfloor[\![l_0]\!]\rfloor) \sqsubseteq \lfloor\overline{\mathsf{Cells}(F)}\rfloor$ *(forward fixed point)*
- $(\mu X \cdot \lceil pre[A](X)\rceil \sqcup \lceil\mathsf{Cells}(F)\rceil) \sqsubseteq \lceil\overline{[\![l_0]\!]}\rceil$ *(backward fixed point).*

This theorem provides the basis of efficient antichain-based algorithms to decide AFA emptiness. In what follows, we will refer to them respectively as the *concrete forward* and *concrete backward* algorithms, as they operate on the concrete semantics of AFA.

4. Abstraction of alternating automata

This section introduces an original abstraction framework for AFA and is structured as follows. In Section 4.1 we formally define a pair of concrete and abstract domains, along with a pair of abstraction and concretization functions and show that they form a Galois connection. We also show that this Galois connection has useful algebraic properties with respect to $\subseteq$-closed sets, which we need for subsequent proofs. Computing abstractions and concretizations is costly, so Section 4.2 defines a *syntactic abstraction* for AFA, computable in linear time, which saves the need of explicitly evaluating abstractions and concretizations. Finally, Section 4.3 provides precision and representability results of our Galois connection; essentially, let X be a set of cells, there exists a unique coarsest abstract domain, which is easily computable, and such that when X is abstracted and then concretized, the resulting set is still X. In other words, we can easily compute the coarsest abstract domain to represent any set of cells without loss of information.

4.1. Abstract domain

Given an AFA with locations Loc, our algorithm will use a family of abstract domains defined by the set of partitions of Loc. The concrete domain is the complete lattice $2^{\mathsf{Cells}(\mathsf{Loc})}$ (ordered by set-inclusion), and each partition $\mathcal{P}$ defines the abstract domain as $2^{\mathsf{Cells}(\mathcal{P})}$. We refer to elements of $\mathsf{Cells}(\mathsf{Loc})$ as *concrete cells* and elements of $\mathsf{Cells}(\mathcal{P})$ as *abstract cells*. An abstract cell is thus a set of blocks of the partition $\mathcal{P}$ and it represents all the concrete cells which can be constructed by choosing at least one location from each block. To capture this representation role of abstract cells, we define the following predicate:

Definition 13. The predicate $\mathsf{Covers} : \mathsf{Cells}(\mathcal{P}) \times \mathsf{Cells}(\mathsf{Loc}) \to \{\top, \bot\}$ is defined as follows: $\mathsf{Covers}(c^\alpha, c)$ iff $c^\alpha = \{\mathcal{P}(l) \mid l \in c\}$.

Note that for every concrete cell c there exists *exactly one* abstract cell c^α such that $\mathsf{Covers}(c^\alpha, c)$. Similarly, we have that for every abstract cell c^α there exists *at least one* concrete cell c such that $\mathsf{Covers}(c^\alpha, c)$.

Example 14. Let $\mathsf{Loc} = \{1, \ldots, 5\}$, $\mathcal{P} = \{a : \{1\}, b : \{2, 3\}, c : \{4, 5\}\}$. We have $\mathsf{Covers}(\{a, c\}, \{1, 3\}) = \bot$, $\mathsf{Covers}(\{a, c\}, \{1, 4\}) = \top$, $\mathsf{Covers}(\{a, c\}, \{1\}) = \bot$.

To make proper use of the theory of abstract interpretation, we define an *abstraction function* and a *concretization function*, and show, in Lemma 16, that they form a *Galois connection* between the concrete domain and each of our abstract domains.

Definition 15. Let $\mathcal{P}$ be a partition of the set Loc. We define the functions $\alpha_{\mathcal{P}} : 2^{\mathsf{Cells(Loc)}} \to 2^{\mathsf{Cells}(\mathcal{P})}$ and $\gamma_{\mathcal{P}} : 2^{\mathsf{Cells}(\mathcal{P})} \to 2^{\mathsf{Cells(Loc)}}$ as follows: $\alpha_{\mathcal{P}}(X) = \{c^\alpha \mid \exists\, c \in X\colon \mathsf{Covers}(c^\alpha, c)\}$, $\gamma_{\mathcal{P}}(X) = \{c \mid \exists\, c^\alpha \in X\colon \mathsf{Covers}(c^\alpha, c)\}$.

Let $\mathcal{P}$ be a partition of the set Loc. The pair of functions $(\alpha_{\mathcal{P}}, \gamma_{\mathcal{P}})$ forms a *Galois connection* [5] iff the following holds

$$\forall X \subseteq \mathsf{Cells(Loc)}\ \forall Y \subseteq \mathsf{Cells}(\mathcal{P})\colon \alpha_{\mathcal{P}}(X) \subseteq Y \Leftrightarrow X \subseteq \gamma_{\mathcal{P}}(Y).$$

We briefly denote this fact as $(2^{\mathsf{Cells(Loc)}}, \subseteq) \underset{\alpha_{\mathcal{P}}}{\overset{\gamma_{\mathcal{P}}}{\leftrightarrows}} (2^{\mathsf{Cells}(\mathcal{P})}, \subseteq)$, or simply $\underset{\alpha_{\mathcal{P}}}{\overset{\gamma_{\mathcal{P}}}{\leftrightarrows}}$, when both the *concrete* and *abstract domains* are clear from the context.

The Galois connection relates the concrete and abstract notions of precision: an abstract value $Y \subseteq \mathsf{Cells}(\mathcal{P})$ approximates a concrete value $X \subseteq \mathsf{Cells(Loc)}$ when $\alpha_{\mathcal{P}}(X) \subseteq Y$, or equivalently (by definition of the Galois connection), $X \subseteq \gamma_{\mathcal{P}}(Y)$.

Incidentally, $(\alpha_{\mathcal{P}}, \gamma_{\mathcal{P}})$ is a *Galois insertion* if $\forall Y \subseteq \mathsf{Cells}(\mathcal{P})\colon \alpha_{\mathcal{P}} \circ \gamma_{\mathcal{P}}(Y) = Y$.

Finally, let us recall basic properties [5] of Galois connection. Let $\mathcal{P}$ be a partition of Loc we have that:

- $\alpha_{\mathcal{P}}$ and $\gamma_{\mathcal{P}}$ are monotonous mappings;
- $\gamma_{\mathcal{P}} \circ \alpha_{\mathcal{P}}(X) \supseteq X$;
- $\gamma_{\mathcal{P}}(X \cap Y) = \gamma_{\mathcal{P}}(X) \cap \gamma_{\mathcal{P}}(Y)$ where $X, Y \subseteq \mathsf{Cells}(\mathcal{P})$;
- $\alpha_{\mathcal{P}}(X \cup Y) = \alpha_{\mathcal{P}}(X) \cup \gamma_{\mathcal{P}}(Y)$ where $X, Y \subseteq \mathsf{Cells(Loc)}$.

Lemma 16. *Let $\mathcal{P}$ be a partition of* Loc. *The pair $(\alpha_{\mathcal{P}}, \gamma_{\mathcal{P}})$ forms a Galois insertion.*

Proof. We first show that $\alpha(X) \subseteq Y$ iff $X \subseteq \gamma(Y)$:

$$\begin{aligned}
&\alpha(X) \subseteq Y\\
&\text{iff } \{c^\alpha \mid \exists\, c \in X\colon \mathsf{Covers}(c^\alpha, c)\} \subseteq Y\\
&\text{iff } \forall\, c \in X\colon \exists\, c^\alpha \in Y\colon \mathsf{Covers}(c^\alpha, c)\\
&\text{iff } X \subseteq \{c \mid \exists\, c^\alpha \in Y\colon \mathsf{Covers}(c^\alpha, c)\}\\
&\text{iff } X \subseteq \gamma(Y).
\end{aligned}$$

To show that it is a Galois insertion, we prove that $\{c^\alpha\} \subseteq \alpha \circ \gamma(c^\alpha)$.

$$\begin{aligned}
\alpha \circ \gamma(c^\alpha) &= \alpha\big(\{c \mid \mathsf{Covers}(c^\alpha, c)\}\big)\\
&= \big\{c'^\alpha \mid \exists c\colon \mathsf{Covers}(c^\alpha, c) \wedge \mathsf{Covers}(c'^\alpha, c)\big\}\\
&\supseteq \{c^\alpha\}. \quad \square
\end{aligned}$$

Remark. In what follows, we alleviate the notations by omitting (i) the $\mathcal{P}$ subscript of α and γ when the partition is clear from the context, and (ii) the extra brackets when $\alpha(\cdot)$, $\gamma(\cdot)$, $\uparrow\cdot$ and $\downarrow\cdot$ are applied to singleton sets. Additionally, we define $\mu_{\mathcal{P}} = \gamma_{\mathcal{P}} \circ \alpha_{\mathcal{P}}$.

Lemmas 17 and 18 below provide a couple of algebraic properties of α and γ which we will need later to prove precision results of our abstract operators.

Lemma 17. *Let $\mathcal{P}$ be a partition of* Loc, *$c \in \mathsf{Cells(Loc)}$ and $c^\alpha \in \mathsf{Cells}(\mathcal{P})$. We have that $\alpha(\uparrow c) = \uparrow\alpha(c)$, $\alpha(\downarrow c) = \downarrow\alpha(c)$, $\gamma(\uparrow c^\alpha) = \uparrow\gamma(c^\alpha)$, and $\gamma(\downarrow c^\alpha) = \downarrow\gamma(c^\alpha)$.*

Proof. For the first equality, let $c \in \mathsf{Cells(Loc)}$, we show that $\uparrow\alpha(c) = \alpha(\uparrow c)$. We conclude from the definition of α that $\alpha(c) = \{c_\alpha \mid \mathsf{Covers}(c_\alpha, c)\}$, hence that $\alpha(c) = \{c_\alpha \mid c_\alpha = \{\mathcal{P}(l) \mid l \in c\}\}$ by definition of Covers, and finally that $\alpha(c) = \{\mathcal{P}(l) \mid l \in c\}$.

$$\begin{aligned}
\alpha(\uparrow c) &= \big\{\alpha(c') \mid c \subseteq c'\big\}\\
&= \big\{\big\{\mathcal{P}(l) \mid l \in c'\big\} \mid c \subseteq c'\big\}. \qquad (3)
\end{aligned}$$

We now prove each inclusion in turn.

$$\begin{aligned}
&\uparrow\alpha(c) \subseteq \alpha(\uparrow c)\\
&\text{iff } \forall c'_\alpha\colon \alpha(c) \subseteq c'_\alpha \to c'_\alpha \in \alpha(\uparrow c)\\
&\text{iff } \forall c'_\alpha\colon \{\mathcal{P}(l) \mid l \in c\} \subseteq c'_\alpha \to c'_\alpha \in \big\{\big\{\mathcal{P}(l) \mid l \in c'\big\} \mid c \subseteq c'\big\} \qquad \text{by (3)}\\
&\text{iff } \forall c'_\alpha\colon \{\mathcal{P}(l) \mid l \in c\} \subseteq c'_\alpha \to \exists c'\colon c \subseteq c' \wedge c'_\alpha = \big\{\mathcal{P}(l) \mid l \in c'\big\}\\
&\text{iff true} \qquad (\text{set } c' = c \cup \{l \in b \mid b \in c'_\alpha\}).
\end{aligned}$$

For the reverse direction, we have

$$\begin{aligned}
&\alpha(\uparrow c) \subseteq \uparrow\alpha(c)\\
&\text{iff } \forall c' \colon c \subseteq c' \to \alpha(c') \in \uparrow\alpha(c)\\
&\text{iff } \forall c' \colon c \subseteq c' \to \left\{\mathcal{P}(l) \mid l \in c'\right\} \in \uparrow\{\mathcal{P}(l) \mid l \in c\} && \text{by } \alpha(c) \text{ above}\\
&\text{iff true.}
\end{aligned}$$

For the second equality, let $c_\alpha \in \mathsf{Cells}(\mathcal{P})$, we show that $\uparrow\gamma(c_\alpha) = \gamma(\uparrow c_\alpha)$.

$$\begin{aligned}
\gamma(\uparrow c_\alpha) &= \left\{c \mid \exists c'_\alpha \colon c_\alpha \subseteq c'_\alpha \wedge \mathsf{Covers}(c'_\alpha, c)\right\} && \text{definition of } \gamma\\
&= \left\{c \mid \exists c'_\alpha \colon c_\alpha \subseteq c'_\alpha \wedge c'_\alpha = \{\mathcal{P}(l) \mid l \in c\}\right\} && \text{definition of Covers}\\
&= \{c \mid c_\alpha \subseteq \{\mathcal{P}(l) \mid l \in c\}\} && \text{elim. } c'_\alpha
\end{aligned}$$

If c is such that $c_\alpha \subseteq \{\mathcal{P}(l) \mid l \in c\}$, so is every $c \subseteq c'$

$$\begin{aligned}
&= \uparrow\{c \mid c_\alpha \subseteq \{\mathcal{P}(l) \mid l \in c\}\}\\
&= \uparrow\{c \mid c_\alpha = \{\mathcal{P}(l) \mid l \in c\}\} && \text{keep the minimum}\\
&= \uparrow\{c \mid c_\alpha = \mathsf{Covers}(c_\alpha, c)\} && \text{definition of Covers}\\
&= \uparrow\gamma(c_\alpha) && \text{definition. of } \gamma.
\end{aligned}$$

The remaining cases are proved similarly. □

Lemma 18. *Let $X, Y \subseteq \mathsf{Cells}(\mathsf{Loc})$ such that both are either upward- or downward-closed. We have that $\alpha(X \cup Y) = \alpha(X) \cup \alpha(Y)$, $\alpha(X \cap Y) = \alpha(X) \cap \alpha(Y)$. Moreover, we have $\gamma(X \cap Y) = \gamma(X) \cap \gamma(Y)$, and $\gamma(X \cup Y) = \gamma(X) \cup \gamma(Y)$ for $X, Y \subseteq \mathsf{Cells}(\mathcal{P})$.*

Proof. The equality $\alpha(X \cup Y) = \alpha(X) \cup \alpha(Y)$ follows from the fact that (α, γ) is a Gallois connection. Let us prove each direction of $\alpha(X \cap Y) = \alpha(X) \cap \alpha(Y)$ in turn. The inclusion $\subseteq$ follows trivially by the (α, γ) Galois connection. For the other direction, we have

$$\begin{aligned}
&c^\alpha \in \alpha(X) \cap \alpha(Y)\\
&\text{iff } \exists c \in X \colon \mathsf{Covers}(c^\alpha, c) \wedge \exists c' \in Y \colon \mathsf{Covers}(c^\alpha, c') && \text{def. of } \alpha\\
&\text{iff } \exists c \in X \colon c^\alpha = \{\mathcal{P}(l) \mid l \in c\} \wedge \exists c' \in Y \colon c^\alpha = \left\{\mathcal{P}(l) \mid l \in c'\right\} && \text{def. of Covers}\\
&\text{only if } \exists c \in X\, \exists c' \in Y \colon c^\alpha = \left\{\mathcal{P}(l) \mid l \in c \cup c'\right\} && (*)
\end{aligned}$$

For each $c \in X$, $c' \in Y$, let c'' denote $c \cup c'$. We have $c'' \in X \cap Y$ since $X = \uparrow X$ and $Y = \uparrow Y$, so we find that

$$\begin{aligned}
&\exists c'' \in X \cap Y \colon c^\alpha = \left\{\mathcal{P}(l) \mid l \in c''\right\} && \text{equiv. to } (*)\\
&\text{iff } \exists c'' \in X \cap Y \colon \mathsf{Covers}(c^\alpha, c'') && \text{def. of Covers}\\
&\text{iff } c^\alpha \in \alpha(X \cap Y).
\end{aligned}$$

The equality $\gamma(X \cap Y) = \gamma(X) \cap \gamma(Y)$ is a property of Galois connections. Finally,

$$\begin{aligned}
\gamma(X \cup Y) &= \{c \mid \exists c_\alpha \in X \cup Y \colon \mathsf{Covers}(c_\alpha, c)\} && \text{definition of } \gamma\\
&= \{c \mid \exists c_\alpha \in X \colon \mathsf{Covers}(c_\alpha, c)\} \cup \{c \mid \exists c_\alpha \in Y \colon \mathsf{Covers}(c_\alpha, c)\}\\
&= \gamma(X) \cup \gamma(Y) && \text{definition of } \gamma. \quad \square
\end{aligned}$$

Corollary 19. *Let $\mathcal{P}$ be a partition of* Loc, *$X \subseteq \mathsf{Cells}(\mathsf{Loc})$ and $Y \subseteq \mathsf{Cells}(\mathcal{P})$. We have $\alpha(\uparrow X) = \uparrow\alpha(X)$, $\alpha(\downarrow X) = \downarrow\alpha(X)$, $\gamma(\uparrow Y) = \uparrow\gamma(Y)$, and $\gamma(Y) = \downarrow\gamma(Y)$.*

4.2. Efficient abstract analysis

In what follows, we will need to evaluate fixed point expressions over the abstract domain. In theory, we could simply surround every predicate transformer occurring in the fixed point expressions by $\alpha \circ \cdot \circ \gamma$ to obtain an abstract fixed point. However, for obvious performance concerns, we want to avoid as many explicit evaluations of γ and α as possible, and ideally make all the computations *directly over the abstract domain*. Furthermore, we would like that these *abstract predicate transformers* enjoy the same useful properties w.r.t. antichains so that we can reuse the results of the previous section. To achieve this goal, we proceed as follows. Given a partition $\mathcal{P}$ of the set of locations of an alternating automaton, we use a *syntactic transformation* θ that builds an *abstract* AFA which over-approximates the behavior of the original automaton. Later we will show that the *pre* and *post* predicate transformers can be directly evaluated on this abstract automaton to obtain the same result as the $\alpha \circ \cdot \circ \gamma$ computation on the original automaton. Doing so results in avoiding almost all explicit evaluations of α and γ. To express this syntactic transformation, we define *syntactic variants* of the abstraction and concretization functions.

Definition 20. Let $\mathcal{P}$ be a partition of the set Loc. We define the following *syntactic* abstraction and concretization functions over positive Boolean formulas.

$$\hat{\alpha} : \mathcal{B}^+(\mathsf{Loc}) \to \mathcal{B}^+(\mathcal{P}) \qquad \hat{\gamma} : \mathcal{B}^+(\mathcal{P}) \to \mathcal{B}^+(\mathsf{Loc})$$

$$\hat{\alpha}(l) = \mathcal{P}(l) \qquad \hat{\gamma}(b) = \bigvee_{l \in b} l$$

$$\hat{\alpha}(\phi_1 \vee \phi_2) = \hat{\alpha}(\phi_1) \vee \hat{\alpha}(\phi_2) \qquad \hat{\gamma}(\phi_1 \vee \phi_2) = \hat{\gamma}(\phi_1) \vee \hat{\gamma}(\phi_2)$$

$$\hat{\alpha}(\phi_1 \wedge \phi_2) = \hat{\alpha}(\phi_1) \wedge \hat{\alpha}(\phi_2) \qquad \hat{\gamma}(\phi_1 \wedge \phi_2) = \hat{\gamma}(\phi_1) \wedge \hat{\gamma}(\phi_2).$$

Remark. It is easy to see that (i) the set of models of a positive Boolean formula is necessarily upward-closed, and (ii) that every upward-closed set can be represented by a positive Boolean formula.

We formalize the link between the two variants of α and γ as follows:

Lemma 21. *For every $\phi \in \mathcal{B}^+(\mathsf{Loc})$, we have that $[\![\hat{\alpha}(\phi)]\!] = \alpha([\![\phi]\!])$, and for every $\phi \in \mathcal{B}^+(\mathcal{P})$, we have that $[\![\hat{\gamma}(\phi)]\!] = \gamma([\![\phi]\!])$.*

Proof. We start by $[\![\hat{\alpha}(\phi)]\!] = \alpha([\![\phi]\!])$, and proceed by induction over the structure of ϕ. For the base case, Lemma 17 and the definition of $[\![\cdot]\!]$ show that $\alpha([\![l]\!]) = \alpha(\uparrow\{\{l\}\}) = \uparrow\alpha(\{\{l\}\}) = \uparrow\{\{\mathcal{P}(l)\}\} = [\![\mathcal{P}(l)]\!] = [\![\hat{\alpha}(l)]\!]$. The inductive cases are immediate by Lemma 18. We follow with $[\![\hat{\gamma}(\phi)]\!] = \gamma([\![\phi]\!])$ and proceed again by induction on the structure of ϕ. For the base case, again Lemma 17 and definition of $[\![\cdot]\!]$ show that $\gamma([\![b]\!]) = \gamma(\uparrow\{\{b\}\}) = \uparrow\gamma(\{\{b\}\}) = \Uparrow\bigcup_{l\in b}\{\{l\}\} = [\![\bigvee_{l\in b} l]\!] = [\![\hat{\gamma}(b)]\!]$. Finally, the inductive cases follow from Lemma 18. □

We can now define the θ transformation.

Definition 22. Let $A = \langle \mathsf{Loc}, \Sigma, l_0, \delta, F\rangle$ be an AFA and $\mathcal{P}$ a partition of Loc. We define $\theta(A, \mathcal{P}) = \langle \mathsf{Loc}^\alpha, \Sigma, b_0, \delta^\alpha, F^\alpha\rangle$ to be an AFA where: $\mathsf{Loc}^\alpha = \mathcal{P}$, $b_0 = \mathcal{P}(l_0)$, $\delta^\alpha(b, \sigma) = \hat{\alpha}(\bigvee_{l\in b}\delta(l, \sigma))$, and $F^\alpha = \{b \in \mathcal{P} \mid b \cap F \neq \emptyset\}$.

We formalize the way the abstract automata approximate the behavior of the concrete automaton with the following lemma.

Lemma 23. *Let A be an* AFA *with locations* Loc *and let $\mathcal{P}$ be a partition of* Loc*, we have $L(A) \subseteq L(\theta(A, \mathcal{P}))$.*

Proof. We begin by showing that every cell $c \in \mathsf{Cells}(\mathsf{Loc})$ is simulated by its corresponding abstract cell $c^\alpha \in \mathsf{Cells}(\mathsf{Loc}^\alpha)$. Definition 2 of $[\![A]\!]$ shows that $\langle c_1, \sigma, c_2\rangle$ iff $c_2 \in [\![\bigwedge_{l\in c_1}\delta(l, \sigma)]\!]$. Let $\alpha(\{c_1\}) = \{c_1^\alpha\}$ and $\alpha(\{c_2\}) = \{c_2^\alpha\}$. We have the following equivalences $\langle c_1^\alpha, \sigma, c_2^\alpha\rangle \in [\![\theta(A, \mathcal{P})]\!]$ iff $c_2^\alpha \in [\![\bigwedge_{b\in c_1^\alpha}\delta^\alpha(b, \sigma)]\!]$ iff $c_2^\alpha \in [\![\bigwedge_{b\in c_1^\alpha}\hat{\alpha}(\bigvee_{l\in b}\delta(l, \sigma))]\!]$ (by Definition 22) iff $c_2^\alpha \in \alpha([\![\bigwedge_{b\in c_1^\alpha}\bigvee_{l\in b}\delta(l, \sigma)]\!])$ (by Lemma 21 and Definition 20). It is easy to see that we have $\langle c_1^\alpha, \sigma, c_2^\alpha\rangle \in [\![\theta(A, \mathcal{P})]\!]$ because $c_2 \in [\![\bigwedge_{b\in c_1^\alpha}\bigvee_{l\in b}\delta(l, \sigma)]\!]$ holds since $c_2 \in [\![\bigwedge_{l\in c_1}\delta(l, \sigma)]\!]$. Now let us show that for every accepting path $c_0, c_1, \ldots, c_n$ in $[\![A]\!]$, the corresponding path $c_0^\alpha, c_1^\alpha, \ldots, c_n^\alpha$ in $[\![\theta(A, \mathcal{P})]\!]$ is also accepting. We know that $l_0 \in c_0$ and we can easily show that $b_0 \in c_0^\alpha$ since we know that $\mathcal{P}(l_0) \in c_0^\alpha$ and $\mathcal{P}(l_0) = b_0$. Finally, we must show that if $c_n \in \mathsf{Cells}(F)$, then $c_n^\alpha \in \mathsf{Cells}(F^\alpha)$. This is easy to see because each block in Loc^α is accepting as soon as it contains at least one accepting location; thus every accepting cell in A is abstracted by an accepting cell in $\theta(A, \mathcal{P})$. □

Lemma 24 provides algebraic properties of *post* that will be needed in what follows when we characterize the precision of our abstract operators.

Lemma 24. *Let A be an* AFA *with locations* Loc *and alphabet Σ.*
For any $\sigma \in \Sigma$, for any $X, Y \subseteq \mathsf{Cells}(\mathsf{Loc})$ such that $\uparrow X = X$ and $\uparrow Y = Y$, we have:

$$post_\sigma[A](X \cup Y) = post_\sigma[A](X) \cup post_\sigma[A](Y) \quad \text{and} \quad post_\sigma[A](X \cap Y) = post_\sigma[A](X) \cap post_\sigma[A](Y).$$

Proof. The equality $post_\sigma[A](X \cup Y) = post_\sigma[A](X) \cup post_\sigma[A](Y)$ follows by property of the Galois connection $\overleftarrow{\underset{post}{\overrightarrow{pre}}}$ (see [6] for a detailed proof). For the second equality, we prove each direction in turn. The inclusion "$\supseteq$" follows easily from monotonicity of $post_\sigma$. For the case $post_\sigma[A](X \cap Y) \subseteq post_\sigma[A](X) \cap post_\sigma[A](Y)$, we prove that:

$$\forall c_x \in X,\ c_y \in Y, \quad c' : c' \in post_\sigma[A](c_x) \cap post_\sigma[A](c_y) \to c' \in post_\sigma[A](\uparrow c_x \cap \Uparrow c_y).$$

We first observe that $c_x \cup c_y \in \uparrow c_x \cap \Uparrow c_y \subseteq X \cap Y$. Then,

$$\begin{aligned} post_\sigma[A](c_x \cup c_y) &= \bigwedge_{l\in c_x\cup c_y}\delta(l, \sigma) \\ &= \bigwedge_{l\in c_x}\delta(l, \sigma) \wedge \bigwedge_{l\in c_y}\delta(l, \sigma) \\ &= \left[\!\!\left[\bigwedge_{l\in c_x}\delta(l, \sigma)\right]\!\!\right] \cap \left[\!\!\left[\bigwedge_{l\in c_y}\delta(l, \sigma)\right]\!\!\right] \\ &= post_\sigma[A](c_x) \cap post_\sigma[A](c_y). \end{aligned}$$

We conclude from $c' \in post_\sigma[A](c_x) \cap post_\sigma[A](c_y)$ that $c' \in post_\sigma[A](c_x \cup c_y)$, and finally that $c' \in post_\sigma[A](\uparrow c_x \cap \uparrow c_y)$ since $c_x \cup c_y \in \uparrow c_x \cap \uparrow c_y$. □

The fact that *post* distributes over the union and intersection allows us to evaluate $post_\sigma[A](\llbracket\phi\rrbracket)$ with $\phi \in \mathcal{B}^+(\mathsf{Loc})$ in a natural fashion, as stated below.

Lemma 25. *Let $\phi \in \mathcal{B}^+(\mathsf{Loc})$, $post_\sigma[A](\llbracket\phi\rrbracket) = \llbracket\phi'\rrbracket$ where ϕ' is the formula ϕ where each occurrence of every variable $l \in \mathsf{Loc}$ is replaced by $\delta(l, \sigma)$.*

Proof. The result is a consequence of the definition of $\llbracket A\rrbracket$, Lemma 24, and the fact that $\llbracket\phi_1 \vee \phi_2\rrbracket = \llbracket\phi_1\rrbracket \cup \llbracket\phi_2\rrbracket$ and $\llbracket\phi_1 \wedge \phi_2\rrbracket = \llbracket\phi_1\rrbracket \cap \llbracket\phi_2\rrbracket$. □

The following Theorem is one of the main results of this paper and its proof makes use of most of the results stated previously. We will show that for any AFA A with locations Loc and for any partition $\mathcal{P}$ of Loc, $post[\theta(A, \mathcal{P})]$ and $pre[\theta(A, \mathcal{P})]$ are the best abstract counterparts for $\mathcal{P}$ of $post[A]$ and $pre[A]$, respectively. This will be crucial for the practical efficiency of our algorithm, as this enables us to avoid expensive computations of α and γ. Moreover, the $\theta(A, \mathcal{P})$ abstraction is computable in linear time and is easy to implement.

Theorem 26. *Let A be an* AFA, *$\mathcal{P}$ a partition of its locations and $A^\alpha = \theta(A, \mathcal{P})$. Then*

$$\alpha \circ post[A] \circ \gamma = post[A^\alpha] \quad \text{and} \quad \alpha \circ pre[A] \circ \gamma = pre[A^\alpha].$$

Proof. We prove each equality in turn.

For $\alpha \circ post[A] \circ \gamma = post[A^\alpha]$, we will show the following:

$$\begin{aligned}
&c^\alpha \in (\alpha \circ post[A] \circ \gamma)(X) \quad \text{iff } c^\alpha \in post[A^\alpha](X).\\
&c^\alpha \in post[A^\alpha](X)\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} post_\sigma[A^\alpha](X) && \text{definition of } post\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \{c_2 \mid \exists\, \langle c_1, \sigma, c_2\rangle \in \llbracket A^\alpha\rrbracket, c_1 \in X\} && \text{definition of } post_\sigma\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \left\{c_2 \mid c_2 \in \left[\!\!\left[\bigwedge_{b\in c_1} \delta^\alpha(b,\sigma)\right]\!\!\right], c_1 \in X\right\} && \text{definition of } \llbracket A^\alpha\rrbracket\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \left\{c_2 \mid c_2 \in \left[\!\!\left[\bigwedge_{b\in c_1} \hat{\alpha}\left(\bigvee_{l\in b} \delta(l,\sigma)\right)\right]\!\!\right], c_1 \in X\right\} && \text{definition of } \delta^\alpha\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \left\{c_2 \mid c_2 \in \left[\!\!\left[\hat{\alpha}\left(\bigwedge_{b\in c_1}\bigvee_{l\in b} \delta(l,\sigma)\right)\right]\!\!\right], c_1 \in X\right\} && \text{Definition 20}\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \left\{c_2 \mid c_2 \in \alpha\left(\left[\!\!\left[\bigwedge_{b\in c_1}\bigvee_{l\in b} \delta(l,\sigma)\right]\!\!\right]\right), c_1 \in X\right\} && \text{Lemma 21}\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \left\{c_2 \mid c_2 \in \alpha \circ post_\sigma[A]\left(\left[\!\!\left[\bigwedge_{b\in c_1}\bigvee_{l\in b} l\right]\!\!\right]\right), c_1 \in X\right\} && \text{Lemma 25}\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \left\{c_2 \mid c_2 \in \alpha \circ post_\sigma[A]\left(\left[\!\!\left[\hat{\gamma}\left(\bigwedge_{b\in c_1} b\right)\right]\!\!\right]\right), c_1 \in X\right\} && \text{Definition 20}\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \{c_2 \mid c_2 \in \alpha \circ post_\sigma[A] \circ \gamma(\uparrow\{c_1\}), c_1 \in X\} && \text{Lemma 21}\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \{c_2 \mid c_2 \in \alpha \circ post_\sigma[A] \circ \gamma(\uparrow X)\}\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \{c_2 \mid c_2 \in \alpha \circ post_\sigma[A] \circ \uparrow \circ \gamma(X)\} && \text{Corollary 19}\\
&\text{iff } c^\alpha \in \bigcup_{\sigma\in\Sigma} \alpha \circ post_\sigma[A] \circ \gamma(X) && \text{Lemma 10}\\
&\text{iff } c^\alpha \in \alpha \circ post[A] \circ \gamma(X) && \text{definition of } post.
\end{aligned}$$

For $\alpha \circ pre[A] \circ \gamma = pre[A^\alpha]$, we will show the following:

$$\exists \sigma \in \Sigma : c^\alpha \in (\alpha \circ pre_\sigma[A] \circ \gamma)(X) \quad \text{iff } c^\alpha \in pre_\sigma[A^\alpha](X).$$

$$\begin{aligned}
&c^\alpha \in pre_\sigma[A^\alpha](X) \\
&\text{iff } c^\alpha \in \{c_1 \mid \exists c_2 \in X : \langle c_1, \sigma, c_2\rangle \in [\![A^\alpha]\!]\} && \text{def. of } pre_\sigma \\
&\text{iff } c^\alpha \in \left\{c_1 \mid \exists c_2 \in X : c_2 \in \left[\!\!\left[\bigwedge_{b\in c_1} \delta^\alpha(b,\sigma)\right]\!\!\right]\right\} && \text{def. of } [\![A^\alpha]\!] \\
&\text{iff } c^\alpha \in \left\{c_1 \mid \exists c_2 \in X : c_2 \in \left[\!\!\left[\bigwedge_{b\in c_1} \hat{\alpha}\left(\bigvee_{l\in b} \delta(l,\sigma)\right)\right]\!\!\right]\right\} && \text{def. of } \delta^\alpha \\
&\text{iff } c^\alpha \in \left\{c_1 \mid \exists c_2 \in X : c_2 \in \left[\!\!\left[\hat{\alpha}\left(\bigwedge_{b\in c_1}\bigvee_{l\in b} \delta(l,\sigma)\right)\right]\!\!\right]\right\} && \text{Definition 20} \\
&\text{iff } c^\alpha \in \left\{c_1 \mid \exists c_2 \in X : c_2 \in \alpha\left(\left[\!\!\left[\bigwedge_{b\in c_1}\bigvee_{l\in b} \delta(l,\sigma)\right]\!\!\right]\right)\right\} && \text{Lemma 21} \\
&\text{iff } c^\alpha \in \left\{c_1 \mid \exists c_2 \in X\ \exists c : c_2 = \alpha(c) \wedge c \in \left[\!\!\left[\bigwedge_{b\in c_1}\bigvee_{l\in b} \delta(l,\sigma)\right]\!\!\right]\right\} \\
&\text{iff } c^\alpha \in \left\{c_1 \mid \exists c_2 \in X\ \exists c,c' : c_2 = \alpha(c) \wedge c' \in \gamma(c_1) \wedge c \in \left[\!\!\left[\bigwedge_{l\in c'} \delta(l,\sigma)\right]\!\!\right]\right\} && \text{def. of } \gamma \\
&\text{iff } c^\alpha \in \{c_1 \mid \exists c_2 \in X\ \exists c,c' : c_2 = \alpha(c) \wedge c' \in \gamma(c_1) \wedge \langle c, \sigma, c'\rangle \in [\![A]\!]\} && \text{def. of } [\![A]\!] \\
&\text{iff } c^\alpha \in \{c_1 \mid \exists c_2 \in X\ \exists c,c' : c \in \gamma(c_2) \wedge \alpha(c') = c_1 \wedge \langle c, \sigma, c'\rangle \in [\![A]\!]\} && \text{def. of } \alpha, \gamma \\
&\text{iff } c^\alpha \in \{c_1 \mid \exists c_2 \in X\ \exists c,c' : c \in \gamma(c_2) \wedge \alpha(c') = c_1 \wedge c' \in pre_\sigma[A](c)\} && \text{def. of } pre \\
&\text{iff } c^\alpha \in \{c_1 \mid c_1 \in \alpha \circ pre_\sigma[A] \circ \gamma(X)\} \\
&\text{iff } c^\alpha \in \alpha \circ pre_\sigma[A] \circ \gamma(X). \quad \square
\end{aligned}$$

4.3. Precision of the abstract domain

We now present some results about precision and representability in our family of abstract domains. In particular, for the automatic refinement of abstract domains, we will need an effective way of computing the *coarsest partition* which can represent an upward- or downward-closed set of cells without loss of precision.

Definition 27. A set of cells $X \subseteq \mathsf{Cells(Loc)}$ is *representable* in the abstract domain $2^{\mathsf{Cells}(\mathcal{P})}$ iff $\mu_\mathcal{P}(X) = X$ (recall that $\mu_\mathcal{P} = \gamma_\mathcal{P} \circ \alpha_\mathcal{P}$).

In what follows, when a set of concrete cells X is representable in the abstract domain $2^{\mathsf{Cells}(\mathcal{P})}$, we shall simply say that $\mathcal{P}$ can represent X.

Definition 28. Let $c, c' \in \mathsf{Cells(Loc)}$ and $\mathcal{P}$ a partition of Loc.
We write $c \equiv_\mathcal{P} c'$ iff $\{\mathcal{P}(l) \mid l \in c\} = \{\mathcal{P}(l) \mid l \in c'\}$.

Lemma 29. *Let $X \subseteq \mathsf{Cells(Loc)}$, $\mathcal{P}$ a partition of* Loc*, and $c \in X$. The following equality holds: $\mu_\mathcal{P}(X) = \left\{c' \mid \exists c \in X : c \equiv_\mathcal{P} c'\right\}$.*

Proof. $\boxed{\supseteq}$ Let $c \in X$, $c \equiv_\mathcal{P} c'$, and c^α be the unique abstract cell such that $\mathsf{Covers}(c^\alpha, c)$. From the definition of $\equiv_\mathcal{P}$, it is clear that we also have $\mathsf{Covers}(c^\alpha, c')$. Thus we have $c' \in \gamma(\{c^\alpha\})$ with $c^\alpha \in \alpha(\{c\})$ and $c \in X$ which, by monotonicity of α and γ, implies that $c' \in \mu_\mathcal{P}(X)$. $\boxed{\subseteq}$ Let $c' \in \mu_\mathcal{P}(X)$ and c^α be the unique abstract cell such that $\mathsf{Covers}(c^\alpha, c')$. Since $c' \in \gamma(\alpha(X))$, we know that $c^\alpha \in \alpha(X)$. By definition of α, we know that there exists $c \in X$ with $\mathsf{Covers}(c^\alpha, c)$, which implies that $c \equiv_\mathcal{P} c'$. $\square$

The following lemma states that if a set is representable in two partitions, then it is representable in their least upper bound. This will immediately imply that there exists a unique coarsest partition which represents any set.

Lemma 30. *Let $X \subseteq \mathsf{Cells(Loc)}$, $\mathcal{P}_1, \mathcal{P}_2$ two partitions of* Loc *and $\mathcal{P} = \mathcal{P}_1 \curlyvee \mathcal{P}_2$. If $\mu_{\mathcal{P}_1}(X) = X$ and $\mu_{\mathcal{P}_2}(X) = X$, then $\mu_\mathcal{P}(X) = X$.*

Proof. By hypothesis, we have that $\mathcal{P} = \mathcal{P}_1 \curlyvee \mathcal{P}_2$. By properties of the least upper bound of two partitions (see [2]), we have that for every $l, l' \in \mathsf{Loc}$: $\{l\} \equiv_\mathcal{P} \left\{l'\right\}$ iff

$$\exists \ell_0, \ldots, \ell_n : l = \ell_0 \wedge l' = \ell_n \wedge \{\ell_0\} \equiv_1 \{\ell_1\} \equiv_2 \{\ell_2\} \cdots \{\ell_{n-1}\} \equiv_n \{\ell_n\}$$

where $\equiv_i \in \left\{\equiv_{\mathcal{P}_1}, \equiv_{\mathcal{P}_2}\right\}$. Now we generalize the above results from Loc to Cells(Loc):

$$
\begin{aligned}
& c \equiv_{\mathcal{P}} c' \\
& \text{iff } \{\mathcal{P}(l) \mid l \in c\} = \left\{\mathcal{P}(l') \mid l' \in c'\right\} && \text{definition of of } \equiv_{\mathcal{P}} \\
& \text{iff } \forall l \in c\, \exists l' \in c' \colon \mathcal{P}(l) = \mathcal{P}(l') \\
& \quad \wedge \forall l' \in c'\, \exists l \in c \colon \mathcal{P}(l) = \mathcal{P}(l') \\
& \text{iff } \forall l \in c\, \exists l' \in c' \colon \{l\} \equiv_{\mathcal{P}} \left\{l'\right\} \\
& \quad \wedge \forall l' \in c'\, \exists l \in c \colon \{l\} \equiv_{\mathcal{P}} \left\{l'\right\} && \text{definition of of } \equiv_{\mathcal{P}}.
\end{aligned}
$$

Note that $c \equiv_{\mathcal{P}} c'$ is equivalent to $\alpha_{\mathcal{P}}(c) = \alpha_{\mathcal{P}}(c')$. Hence, by using the above result, we obtain that for every $c, c' \in$ Cells(Loc):

$$c \equiv_{\mathcal{P}} c' \quad \text{iff } \exists c_0, \ldots, c_n \colon c = c_0 \wedge c' = c_n \wedge c_0 \equiv_1 c_1 \equiv_2 c_2 \cdots c_{n-1} \equiv_n c_n$$

where $\equiv_i \in \left\{\equiv_{\mathcal{P}_1}, \equiv_{\mathcal{P}_2}\right\}$. We now prove that $\mu_{\mathcal{P}_2}(X) \subseteq X$ and $\mu_{\mathcal{P}_1}(X) \subseteq X$ implies $\mu_{\mathcal{P}}(X) \subseteq X$. Note that the reverse inclusions follow directly from properties of Galois insertions. From $\mu_{\mathcal{P}_1}(X) \subseteq X$ and $\mu_{\mathcal{P}_2}(X) \subseteq X$, we conclude that $\forall c \in X\, \forall c' \colon (c \equiv_{\mathcal{P}_1} c') \to c' \in X$ and $\forall c \in X\, \forall c' \colon (c \equiv_{\mathcal{P}_2} c') \to c' \in X$, respectively, hence that $\forall c \in X\, \forall c' \colon (c \equiv_{\mathcal{P}} c') \to c' \in X$ by the above, and finally that $\mu_{\mathcal{P}}(X) \subseteq X$. □

As the lattice of partitions is a complete lattice, we have the following corollary.

Corollary 31. *For all $X \subseteq$ Cells(Loc), $\mathcal{P} = \curlyvee\{\mathcal{P}' \mid \mu_{\mathcal{P}'}(X) = X\}$ is the coarsest partition such that $\mu_{\mathcal{P}}(X) = X$.*

For upward- and downward-closed sets, we have an efficient way to compute this coarsest partition. We start with upward-closed sets. To obtain an algorithm, we use the notion of *neighbour list*. The neighbour list of a location l with respect to an upward-closed set X, denoted $\mathcal{N}_X(l)$, is the set of cells containing l in $\lfloor X \rfloor$, from which l has been removed.

Definition 32. Let $X \subseteq$ Cells(Loc) be an upward-closed set. The *neighbour list* of a location $l \in$ Loc w.r.t. X is the set $\mathcal{N}_X(l) = \{c \setminus \{l\} \mid c \in \lfloor X \rfloor, l \in c\}$.

The following lemma states that if two locations share the same neighbour lists w.r.t. an upward-closed set X, then they can be put in the same partition block and preserve the representability of X. Conversely, X cannot be represented by any partition which puts into the same block two locations that have different neighbour lists.

Lemma 33. *Let $\mathcal{P}$ be a partition of* Loc *and X be an upward-closed set, $\mu_{\mathcal{P}}(X) = X$ iff $\forall\, l, l' \in$ Loc: $\mathcal{P}(l) = \mathcal{P}(l') \to \mathcal{N}_X(l) = \mathcal{N}_X(l')$.*

Proof. $\boxed{\to}$ By contradiction, we have that $\mu_{\mathcal{P}}(X) = X$, and for some l, l', we have $\mathcal{P}(l) = \mathcal{P}(l')$ but $\mathcal{N}_X(l) \neq \mathcal{N}_X(l')$. Without loss of generality, we consider that $s \in \mathcal{N}_X(l)$ and $s \notin \mathcal{N}_X(l')$. By the definition of $\mathcal{N}_X$, we know that $s \cup \{l\} \in \lfloor X \rfloor$, $s \cup \left\{l'\right\} \notin \lfloor X \rfloor$. By Lemma 29 and as $l \equiv_{\mathcal{P}} l'$, we also know that $\left(s \cup \left\{l'\right\}\right) \equiv_{\mathcal{P}} (s \cup \{l\})$, and so $s \cup \left\{l'\right\} \in \mu_{\mathcal{P}}(X)$. If $s \cup \left\{l'\right\} \notin X$, then $X \neq \mu_{\mathcal{P}}(X)$ and we have a contradiction. Otherwise, because $s \cup \left\{l'\right\} \in X$ but $s \cup \left\{l'\right\} \notin \lfloor X \rfloor$, there must exist a smaller cell $c \subset s \cup \left\{l'\right\}$ with $c \in \lfloor X \rfloor$. Two cases remain, both of which lead to a contradiction. First, if $l' \notin c$, then $c \subseteq s$, but then $s \cup \{l\}$ could not have appeared in $\lfloor X \rfloor$. Finally, if $l' \in c$, then $c = s' \cup \left\{l'\right\}$ with $s' \subset s$. As $l \equiv_{\mathcal{P}} l'$, we know that $s' \cup \{l\} \in \mu_{\mathcal{P}}(X)$, but $s' \cup \{l\}$ cannot be in X because $s \cup \{l\} \in \lfloor X \rfloor$.
$\boxed{\leftarrow}$ First, let us consider the particular case of partition $\mathcal{P}_{l,l'}$ where only l and l' share a block, i.e. $\mathcal{P}_{l,l'} = \left\{\left\{l, l'\right\}\right\} \cup \left\{\left\{l''\right\} \mid l'' \notin \left\{l, l'\right\}\right\}$. Assume that $\mathcal{N}_X(l) = \mathcal{N}_X(l')$. We will show that X is representable in that partition. By definition of abstraction, and concretization functions, and their algebraic properties, $\mu_{\mathcal{P}_{l,l'}}(X) = \uparrow \cup_{x \in \lfloor X \rfloor} \mu_{\mathcal{P}_{l,l'}}(x)$. Let us consider $\mu_{\mathcal{P}_{l,l'}}(x)$. There are two cases, either (i) $x \cap \{l, l'\} = \emptyset$, then clearly $\mu_{\mathcal{P}_{l,l'}}(\{x\}) = \{\{x\}\}$, or (ii) $x \cap \{l, l'\} \neq \emptyset$, then there are a priori three cases to consider: (a) $l \in x$, $l' \in x$, (b) $l \in x$, $l' \notin x$, (c) $l \notin x$, $l' \in x$. Case (a) is not possible as $\mathcal{N}_X(l) = \mathcal{N}_X(l')$, and so l and l' cannot appear together in a set in $\lfloor X \rfloor$. Case (b), $\mu_{\mathcal{P}_{l,l'}}(\{x\}) = \{(x \setminus \{l\}) \cup \{l'\}, x\}$, and let us show that $(x \setminus \{l\}) \cup \{l'\}$ belongs to $\lfloor X \rfloor$. Assume it is not the case, so $x \setminus \{l\} \notin \mathcal{N}_X(l')$, but as $x \setminus \{l\} \in \mathcal{N}_X(l)$, we obtain a contradiction with our hypothesis that $\mathcal{N}_X(l) = \mathcal{N}_X(l')$. Case (c) is symmetrical to case (b). And so, we have established that X is representable in $\mathcal{P}_{l,l'}$. Let us now turn to the general case. Let $\mathcal{P}$ be a partition such that $\forall l, l' \in$ Loc: $\mathcal{P}(l) = \mathcal{P}(l') \to \mathcal{N}_X(l) = \mathcal{N}_X(l')$. One can easily check that $\mathcal{P} = \curlyvee_{\{l,l' \mid \mathcal{P}(l) = \mathcal{P}(l')\}} \mathcal{P}_{l,l'}$, because the least upper bound on partitions has the effect of merging blocks which overlap by at least one element. By the previous result and by Lemma 30, we have that X is representable in $\mathcal{P}$. □

Lemmas 30 and 33 yield a natural algorithm for computing the *coarsest partition which can represent an upward-closed set X*. In fact, computing the neighbour list w.r.t. X for each element of Loc suffices to compute the coarsest partition which can represent X.

Corollary 34. *Let $X \subseteq$ Cells(Loc) be an upward-closed set. Then the partition $\mathcal{P}$ induced by the equivalence relation $l \sim l'$ iff $\mathcal{N}_X(l) = \mathcal{N}_X(l')$ is the coarsest partition such that $\mu_{\mathcal{P}}(X) = X$.*

Input: $A = \langle \mathsf{Loc}, \Sigma, l_0, \delta, F\rangle$
Output: True iff $L(A) = \emptyset$

1 $\mathcal{P}_0 \leftarrow \{F, \mathsf{Loc} \setminus F\}$
2 $Z_0 \leftarrow \overline{\mathsf{Cells}(F)}$
3 **for** i **in** 0, 1, 2, ... **do**
4 $\quad A_i^\alpha \leftarrow \theta(A, \mathcal{P}_i)$
5 $\quad A_i^\alpha = \langle \mathsf{Loc}^\alpha, \Sigma, b_0, \delta^\alpha, F^\alpha\rangle$
6 $\quad I \leftarrow [\![b_0]\!]$
7 $\quad R_i \leftarrow \mu x \cdot (I \cup post[A_i^\alpha](x)) \cap \alpha_{\mathcal{P}_i}(Z_i)$
8 $\quad$ **if** $post[A_i^\alpha](R_i) \subseteq \alpha_{\mathcal{P}_i}(Z_i)$ **then**
9 $\quad\quad$ **return** True
10 $\quad$ **if** $[\![l_0]\!] \not\subseteq Z_i$ **then**
11 $\quad\quad$ **return** False
12 $\quad Z_{i+1} \leftarrow \gamma_{\mathcal{P}_i}(R_i) \cap \widetilde{pre}[A](\gamma_{\mathcal{P}_i}(R_i))$
13 $\quad \mathcal{P}_{i+1} \leftarrow \curlyvee\{\mathcal{P} \mid \mu_{\mathcal{P}}(Z_{i+1}) = Z_{i+1}\}$

Input: $A = \langle \mathsf{Loc}, \Sigma, l_0, \delta, F\rangle$
Output: True iff $L(A) = \emptyset$

1 $\mathcal{P}_0 \leftarrow \{\{l_0\}, \mathsf{Loc} \setminus \{l_0\}\}$
2 $Z_0 \leftarrow \overline{[\![l_0]\!]}$
3 **for** i **in** 0, 1, 2, ... **do**
4 $\quad A_i^\alpha \leftarrow \theta(A, \mathcal{P}_i)$
5 $\quad A_i^\alpha = \langle \mathsf{Loc}^\alpha, \Sigma, b_0, \delta^\alpha, F^\alpha\rangle$
6 $\quad B \leftarrow \mathsf{Cells}(F^\alpha)$
7 $\quad R_i \leftarrow \mu x \cdot (B \cup pre[A_i^\alpha](x)) \cap \alpha_{\mathcal{P}_i}(Z_i)$
8 $\quad$ **if** $pre[A_i^\alpha](R_i) \subseteq \alpha_{\mathcal{P}_i}(Z_i)$ **then**
9 $\quad\quad$ **return** True
10 $\quad$ **if** $\mathsf{Cells}(F) \not\subseteq Z_i$ **then**
11 $\quad\quad$ **return** False
12 $\quad Z_{i+1} \leftarrow \gamma_{\mathcal{P}_i}(R_i) \cap \widetilde{post}[A](\gamma_{\mathcal{P}_i}(R_i))$
13 $\quad \mathcal{P}_{i+1} \leftarrow \curlyvee\{\mathcal{P} \mid \mu_{\mathcal{P}}(Z_{i+1}) = Z_{i+1}\}$

Fig. 2. The *abstract forward* (left) and *abstract backward* (right) algorithms.

Example 35. Let $\mathsf{Loc} = \{1, \ldots, 7\}$, $\lfloor X \rfloor = \{\{1, 2, 4\}, \{1, 3, 4\}, \{2, 4, 5\}, \{3, 4, 5\}\}$. $\mathcal{N}_X(1) = \mathcal{N}_X(5) = \{\{2, 4\}, \{3, 4\}\}$, $\mathcal{N}_X(2) = \mathcal{N}_X(3) = \{\{1, 4\}, \{4, 5\}\}$, and $\mathcal{N}_X(6) = \mathcal{N}_X(7) = \{\}$. The coarsest partition which can represent X is: $\mathcal{P} = \{a : \{1, 5\}, b : \{2, 3\}, c : \{4\}, d : \{6, 7\}\}$, with $\lfloor \alpha_{\mathcal{P}}(X) \rfloor = \{\{a, b, c\}\}$.

The representability of downward-closed sets follows immediately from Lemma 36. In practice, we simply compute the coarsest partition for the complementary upward-closed set.

Lemma 36. *Let $X \subseteq \mathsf{Cells}(\mathsf{Loc})$, $\mathcal{P}$ a partition of* Loc. *$\mu_P(X) = X$ iff $\mu_P(\overline{X}) = \overline{X}$.*

Proof. Because for every $Y \subseteq \mathsf{Cells}(\mathsf{Loc})$, we have $\mu_P(Y) \supseteq Y$ (by property of Galois connections) it is enough to show that for every $X \subseteq \mathsf{Cells}(\mathsf{Loc})$, $\mu_P(X) \subseteq X$ implies $\mu_P(\overline{X}) \subseteq \overline{X}$. We conclude from Lemma 29 that $\mu_P(X) \subseteq X$ iff

$$\forall c \in X\, \forall c' : c \equiv_{\mathcal{P}} c' \rightarrow c' \in X. \tag{4}$$

On the other hand, $\mu_{\mathcal{P}}(\overline{X}) \subseteq \overline{X}$ is equivalent to $\mu_{\mathcal{P}}(\overline{X}) \cap X = \emptyset$. Assuming that this equality does not hold, we find that there is $c \notin X$ and $c' \in X$ such that $c \equiv_{\mathcal{P}} c'$ by definition of $\mu_{\mathcal{P}}$, which contradicts (4), hence the result. □

5. Abstraction refinement algorithm

This section presents two fixed point guided abstraction refinement algorithms for AFA (Fig. 2). These algorithms share several ideas with the generic algorithm presented in [7] but they are formally different because they do not strictly refine the abstract domain at each refinement step, but always use the most abstract domain instead. We thus provide arguments showing their correctness.

To make the algorithms more readable, we have chosen not to include the antichain-specific notations in the pseudo-code. From the results of Section 3, it is easy to see that the forward abstract algorithm only manipulates upward-closed sets while the backward abstract algorithm only manipulates downward-closed sets; all these sets can thus be represented using antichains, which is what we implemented.

We concentrate here on explanations related to the abstract forward algorithm. The abstract backward algorithm is the dual of this algorithm, hence proofs are similar to the forward case. We first give an informal presentation of the ideas underlying the algorithm and then we expose formal arguments for its soundness and completeness.

Description of the forward abstract algorithm. The most important information computed in the algorithm is Z_i, which is an over-approximation of the set of reachable cells which cannot reach an accepting cell in i steps or less. In other words, all the cells outside Z_i are either unreachable, or can lead to an accepting cell in i steps or less (or both). Our algorithm always uses the coarsest partition $\mathcal{P}_i$ that allows Z_i to be represented in the corresponding abstract domain. The algorithm begins by initializing Z_0 with the set of non-accepting cells and by initializing $\mathcal{P}_0$ accordingly (lines 1 and 2). The main loop proceeds as follows. First, we compute the abstract reachable cells R_i which are within Z_i, which is done by applying the θ transformation using $\mathcal{P}_i$ (line 5), and by computing a forward abstract fixed point (line 7). If R_i does not contain a cell which can leave Z_i, we know (as we will formally prove later in this section) that the automaton is empty (line 8). If on the other hand, an initial cell (i.e., a cell containing l_0) is no longer in Z_i then we know that it can lead to an accepting cell in i steps or less (as it is obviously reachable) and we conclude that the automaton is non-empty (line 11). In the case where both tests failed, we *refine* the information contained in Z_i by removing all the cells which can leave R_i in one step, as we know that these cells are either surely unreachable or can lead to an accepting cell in $i + 1$ steps or less. Finally, the current abstract domain is changed to be able to represent the new Z_i (line 13), using the neighbour list algorithm of Corollary 34. It is important to note that this refinement operation is not the traditional refinement used in counter-example guided abstraction refinement. Note also that our algorithm does not necessarily choose a new abstract domain that is strictly more precise than the previous one

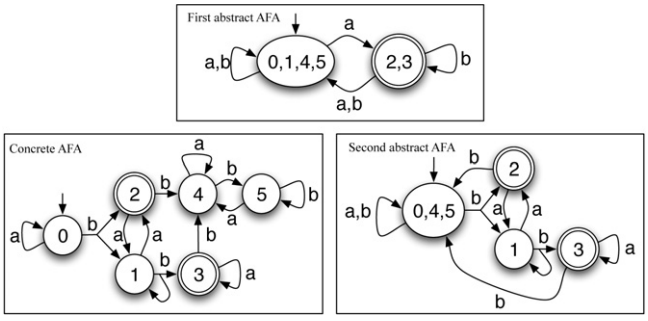


Fig. 3. Illustration of the *abstract forward* algorithm. One refinement step suffices to show emptiness.

as in [7]. Instead, the algorithm uses the most abstract domain possible at all times. As we cannot rely on the termination proof from [7], we provide a new one in this section.

Example 37. An illustration of the abstract forward algorithm is depicted at Fig. 3. The algorithm computes $\mathcal{P}_0 = \{[0, 1, 4, 5], [2, 3]\}$; $Z_0 = \uparrow\{\{0\}, \{1\}, \{4\}, \{5\}\}$; $R_0 = \uparrow\{\{[0, 1, 4, 5]\}\}$; $Z_1 = \uparrow\{\{0\}, \{1, 2\}, \{4\}, \{5\}\}$; $\mathcal{P}_1 = \{[0, 4, 5], [1], [2], [3]\}$; $R_1 = \uparrow\{\{[0, 4, 5]\}, \{[1], [2]\}\}$. Since $post[A_1^\alpha](R_1) \subseteq \alpha_{\mathcal{P}_1}(Z_1)$, the AFA is empty.

Completeness and correctness of the forward abstract algorithm. Correctness and completeness rely on the properties formalized in the following lemma.

Lemma 38. *Let* $\mathsf{Reach} = \mu X \cdot \left([\![l_0]\!] \cup post[A](X)\right)$ *be the reachable cells of A, let* $\mathsf{Bad}^k = \cup_{j=0}^k pre^j[A](\mathsf{Cells}(F))$ *be the cells that can reach an accepting cell in k steps or less. The following four properties hold:*

1. $\forall i \geq 0$: $\mu_{\mathcal{P}_i}(Z_i) = Z_i$, *i.e.* Z_i *is representable in* $\mathcal{P}_i$;
2. $\forall i \geq 0$: $Z_{i+1} \subseteq Z_i$, *i.e. the sets* Z_i *are decreasing;*
3. $\forall i \geq 0$: $\mathsf{Reach} \setminus \mathsf{Bad}^i \subseteq Z_i$, *i.e.* Z_i *over-approximates the reachable cells that cannot reach an accepting cell in i steps or less;*
4. $\forall i \geq 0$ *if* $Z_i = Z_{i+1}$, *then* $post[A_i^\alpha](R_i) \subseteq \alpha_{\mathcal{P}_i}(Z_i)$.

Proof. We prove each point in turn. Point 1 is straightforward as $\mathcal{P}_0$ is chosen in line 1 to be able to represent Z_0, and $\mathcal{P}_{i+1}$ is chosen in line 13 to be able to represent Z_{i+1}. Point 2 follows directly from the fact that $R_i \subseteq \alpha_{\mathcal{P}_i}(Z_i)$, Z_i is representable in $\mathcal{P}_i$ by the previous point, and the definition of Z_{i+1} in line 12. Point 3 is established by induction. The property is clearly true for Z_0. Let us establish it for Z_{i+1} using the induction hypothesis that it is true for Z_i. By soundness of the theory of abstract interpretation, we know that in line 7 we compute a set R_i which over-approximates the set $\mathsf{Reach} \setminus \mathsf{Bad}^i$. In line 12, we remove cells that can leave this set in one step, so Z_{i+1} is an over-approximation of the reachable cells that cannot reach an accepting cell in $i + 1$ steps or less, i.e. $\mathsf{Reach} \setminus \mathsf{Bad}^{i+1} \subseteq Z_{i+1}$, which concludes the proof. Point 4 is established as follows. If $Z_i = Z_{i+1}$, then clearly $post[A](\gamma_{\mathcal{P}_i}(R_i)) \subseteq \gamma_{\mathcal{P}_i}(R_i)$ as no cell can leave $\gamma_{\mathcal{P}_i}(R_i)$ in one step (from line 12). Then $\gamma_{\mathcal{P}_i}(R_i) \subseteq Z_i$ shows that $post[A](\gamma_{\mathcal{P}_i}(R_i)) \subseteq Z_i$. Finally, we conclude from monotonicity of $\alpha_{\mathcal{P}_i}$ (itself a consequence of the Galois connection, see Lemma 16) that $\alpha_{\mathcal{P}_i}(post[A](\gamma_{\mathcal{P}_i}(R_i))) \subseteq \alpha_{\mathcal{P}_i}(Z_i)$. This is equivalent to $post[A_i^\alpha](R_i) \subseteq \alpha_{\mathcal{P}_i}(Z_i)$ by Theorem 26. □

Let us now establish the soundness and completeness of our forward algorithm.

Theorem 39. *The forward abstract algorithm with refinement is sound and complete to decide the emptiness of* AFA.

Proof. Let A be the AFA on which the algorithm is executed. First, let us show that the algorithm is sound. Assume that the algorithm returns "True". In this case, the test of line 8 evaluates to true which implies that $post[A_i^\alpha](R_i) \subseteq \alpha_{\mathcal{P}_i}(Z_i)$, hence $\alpha_{\mathcal{P}_i} \circ post[A] \circ \gamma_{\mathcal{P}_i}(R_i) \subseteq \alpha_{\mathcal{P}_i}(Z_i)$ by Theorem 26 and then that $post[A](\gamma_{\mathcal{P}_i}(R_i)) \subseteq \gamma_{\mathcal{P}_i} \circ \alpha_{\mathcal{P}_i}(Z_i)$ by Galois connection and finally that $post[A](\gamma_{\mathcal{P}_i}(R_i)) \subseteq Z_i$ by Point 1 of Lemma 38. Because $\gamma_{\mathcal{P}_i}(R_i)$ is an over-approximation of the concrete reachable cells and as $post[A](\gamma_{\mathcal{P}_i}(R_i)) \subseteq Z_i \subseteq \overline{\mathsf{Cells}(F)}$, we know that all the accepting cells are unreachable, hence $L(A) = \emptyset$. Now, assume that the algorithm returns "False". Then $[\![l_0]\!] \not\subseteq Z_i$ which means that l_0 is able to reach an accepting cell in i steps or less. Since l_0 is obviously reachable, we can conclude that $L(A) \neq \emptyset$. To prove the completeness of the algorithm, we only need to establish its termination. Suppose it does not terminate, point 2 and 4 of Lemma 38 show that the chain of Z_i is strictly descending, namely $Z_{i+1} \subset Z_i$. So, if the algorithm does not terminate, it means that we have an infinite strictly decreasing chain of values in $\mathsf{Cells}(\mathsf{Loc})$ which contradicts that Loc is a finite set. Hence the algorithm terminates. □

Let us turn to the backward algorithm which is the dual of the forward algorithm. Lemma 40 which is the dual of Lemma 38 is the central argument to prove termination and correctness of the algorithm. We state the result without providing a proof which can be obtained easily by dualizing the proof of Lemma 38.

Lemma 40. *Let* $\mathsf{Success} = \mu X \cdot \left(\mathsf{Cells}(F) \cup pre[A](X)\right)$ *be the cells that reach a cell of* $\mathsf{Cells}(F)$, *let* $\mathsf{Reach}^k = \cup_{j=0}^k post^j[A]([\![l_0]\!])$ *be the cells reachable in k steps or less. The following four properties hold:*

1. $\forall i \geq 0$: $\mu_{\mathcal{P}_i}(Z_i) = Z_i$, *i.e.* Z_i *is representable in* $\mathcal{P}_i$;

2. $\forall i \geq 0$: $Z_{i+1} \subseteq Z_i$, *i.e. the sets* Z_i *are decreasing;*
3. $\forall i \geq 0$: $\mathsf{Success} \setminus \mathsf{Reach}^i \subseteq Z_i$, *i.e.* Z_i *over-approximates the cells that can reach an accepting cell and that are unreachable from* $[\![l_0]\!]$ *in* i *steps or less;*
4. $\forall i \geq 0$ *if* $Z_i = Z_{i+1}$, *then* $pre[A_i^\alpha](R_i) \subseteq \alpha_{\mathscr{P}_i}(Z_i)$.

Let us now turn to the soundness and completeness of our backward algorithm.

Theorem 41. *The backward abstract algorithm with refinement is sound and complete to decide the emptiness of* AFA.

Proof. Let A be the AFA on which the algorithm is executed. First, let us show that the algorithm is sound. Assume that the algorithm returns "True". In this case, the test of line 8 evaluates to true which implies that $pre[A_i^\alpha](R_i) \subseteq \alpha_{\mathscr{P}_i}(Z_i)$ and so $pre[A](\gamma_{\mathscr{P}_i}(R_i)) \subseteq Z_i$ by Point 1 of Lemma 40 and Theorem 26. Because $\gamma_{\mathscr{P}_i}(R_i)$ is an over-approximation of the concrete cells that reach an accepting cell and as $pre[A](\gamma_{\mathscr{P}_i}(R_i)) \subseteq Z_i \subseteq \overline{[\![l_0]\!]}$, we know that all the cells that can reach an accepting cell are unreachable from $[\![l_0]\!]$. Hence $L(A) = \emptyset$. Now, assume that the algorithm returns "False". Then $\mathsf{Cells}(F) \not\subseteq Z_i$ which means that a reachable cell is able to reach an accepting cell in i steps or less. Hence we conclude that $L(A) \neq \emptyset$. To prove the completeness of the algorithm, we only need to establish its termination. This part of the proof is exactly the same as for the forward abstract algorithm. □

6. Experimental evaluation

In this section, we evaluate the practical performance of our techniques by applying them to the satisfiability problem of LTL over finite words.

Definition 42. Let Prop be a finite set of propositions. An LTL formula ϕ over Prop is of the form: $\phi ::= p \in \mathsf{Prop} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid X\phi \mid \phi_1 U \phi_2$. Let $\Sigma = 2^{\mathsf{Prop}}$. The semantics of a finite-word LTL formula ϕ, which we denote $[\![\phi]\!]$, is a subset of Σ^* as defined by the following semantic rules. Let $w \in \Sigma^*$. We use the following notations: w_i is the letter in w at the position i, starting at zero; $|w|$ is the length of w; and $w_{i\rightarrow}$ is the suffix of w starting at position i.

$w \in [\![p]\!]$ iff $p \in w_0$
$w \in [\![\neg\phi]\!]$ iff $w \notin [\![\phi]\!]$
$w \in [\![\phi_1 \wedge \phi_2]\!]$ iff $w \in [\![\phi_1]\!]$ and $w \in [\![\phi_2]\!]$
$w \in [\![\phi_1 \vee \phi_2]\!]$ iff $w \in [\![\phi_1]\!]$ or $w \in [\![\phi_2]\!]$
$w \in [\![X\phi]\!]$ if $|w| \geq 1$ and $w_{1\rightarrow} \in [\![\phi]\!]$; else $w \notin [\![X\phi]\!]$
$w \in [\![\phi_1 U \phi_2]\!]$ iff $\exists\, i, 0 \leq i < |w| : w_{i\rightarrow} \in [\![\phi_2]\!]$ and $\forall\, j, 0 \leq j \leq i : w_{j\rightarrow} \in [\![\phi_1]\!]$.

We define the syntactic shortcuts *true* and *false* in the usual way, as well as $F\phi \equiv true\, U \phi$ and $G\phi \equiv \neg F \neg \phi$. Notice that no word of length 0 or 1 can satisfy $Xtrue$, which is convenient to detect the end of the word. The formula $F\neg Xtrue$ is thus valid in finite-word LTL, and $GXtrue$ is not satisfiable.

We have run our algorithms on three series of benchmarks. Each benchmark is composed of a pair of LTL formulas $\langle \psi, \phi \rangle$ interpreted on finite words, and for which we want to know whether ϕ is a logical consequence of ψ, i.e. whether $\psi \models \phi$ holds. To solve this problem, we translate the formula $\psi \wedge \neg\phi$ into an AFA and check that the language of the AFA is empty. This translation is linear in the size of the formula and creates a location in the AFA for each subformula. As we will see, our ψ formulas are constructed as large conjunctions of constraints and model the behavior of finite-state systems, while the ϕ formulas model properties of those systems. We defined properties with varying degrees of *locality*. Intuitively, a property ϕ is local when only a small number of subformulas of ψ are needed to establish $\psi \models \phi$. This is not a formal notion but it will be clear from the examples. We will show in this section that our abstract algorithms are able to automatically identify subformulas which are not needed to establish the property. We only report results where $\psi \models \phi$ holds; these are arguably the most difficult instances, as the entire fixed point must be computed. We now present each benchmark in turn.

Benchmark 1. The first benchmark takes 2 parameters $n > 0$ and $0 < k \leq n$: $\mathsf{Bench1}(n, k) = \langle \psi \equiv \bigwedge_{i=0}^{n-1} G(p_i \rightarrow (F(\neg p_i) \wedge F(p_{i+1}))), \phi \equiv Fp_0 \rightarrow Fp_k \rangle$. Clearly we have that $\psi \models \phi$ holds for all values of k and also that the subformulas of ψ for $i > k$ are not needed to establish $\psi \models \phi$.

Benchmark 2. This second benchmark is used to demonstrate how our algorithms can automatically detect less obvious versions of locality than for Bench1. It uses 2 parameters k and n with $0 < k \leq n$ and is built using the following recursive nesting definition: $\mathsf{Sub}(n, 1) = Fp_n$; for odd values of $k > 1$ $\mathsf{Sub}(n, k) = F(p_n \wedge X(\mathsf{Sub}(n, k-1)))$; and for even values of $k > 1$ $\mathsf{Sub}(n, k) = F(\neg p_n \wedge X(\mathsf{Sub}(n, k-1)))$. We have : $\mathsf{Bench2}(n, k) = \langle \psi \equiv \bigwedge_{i=0}^{n-1} G(p_i \rightarrow \mathsf{Sub}(i+1, k)), \phi \equiv Fp_0 \rightarrow Fp_n \rangle$. It is relatively easy to see that $\psi \models \phi$ holds for any value of k, and that for odd values of k, the nested subformulas beyond the first level are not needed to establish the property.

Benchmark 3. This third and final benchmark aims at demonstrating the usefulness of our abstraction algorithms in a more realistic setting. We specified the behavior of a lift with n floors with a parametric LTL formula. For n floors, $\mathsf{Prop} = \{f_1, \ldots, f_n, b_1, \ldots, b_n, open\}$. The f_i propositions represent the current floor. Only one of the f_i's can be true at any time, which is initially f_1. The b_i propositions represent the state (lit or unlit) of the call-buttons of each floor and there is only one button per floor. The additional *open* proposition is true when the doors of the lift are open. The constraints on the dynamics of this system are as follows: (i) initially the lift is at the first floor and the doors are open, (ii) the lift must close its doors

					concrete forward			*abstract backward*					
	n	k	|Loc|	|Prop|	time	ATC	iters	time	ATC^{α}	ATC^{γ}	iters	steps	$\|\mathcal{P}\|$
Bench1	11	5	50	12	0.10	6	3	0.23	55	2	5	3	27
	15	5	66	16	1.60	6	3	0.56	55	2	5	3	31
	19	5	82	20	76.62	6	3	8.64	55	2	5	3	35
	11	7	50	12	0.13	8	3	0.87	201	2	5	3	31
	15	7	66	16	2.04	8	3	1.21	201	2	5	3	35
	19	7	82	20	95.79	8	3	9.99	201	2	5	3	39
	11	9	50	12	0.16	10	3	12.60	779	2	5	3	35
	15	9	66	16	2.69	10	3	13.42	779	2	5	3	39
	19	9	82	20	125.85	10	3	46.47	779	2	5	3	43
Bench2	7	1	19	8	0.06	8	2	0.10	11	2	4	3	14
	10	1	25	11	0.06	10	2	0.10	14	2	4	3	17
	13	1	31	14	0.08	14	2	0.12	17	2	4	3	20
	7	3	33	8	0.78	201	14	0.13	11	2	4	3	26
	10	3	45	11	802.17	4339	20	0.30	14	2	4	3	35
	13	3	57	14	>1000	–	–	1.26	17	2	4	3	44
	7	5	47	8	88.15	2122	26	0.14	11	2	4	3	26
	10	5	65	11	>1000	–	–	0.37	14	2	4	3	35
	13	5	83	14	>1000	–	–	1.47	17	2	4	3	44
Lift : Spec1	8	3	84	17	0.30	10	17	0.51	23	40	7	4	21
	12	3	116	25	17.45	10	25	1.63	23	40	7	4	21
	16	3	148	33	498.65	10	33	26.65	23	40	7	4	21
	8	4	84	17	0.26	10	17	1.29	37	72	10	6	24
	12	4	116	25	17.81	10	25	5.02	37	72	10	6	24
	16	4	148	33	555.44	10	33	78.75	37	72	10	6	24
	8	5	84	17	0.32	10	17	3.70	42	141	12	8	27
	12	5	116	25	20.24	10	25	47.45	42	141	12	8	27
	16	5	148	33	543.27	10	33	>1000	–	–	–	–	–
Lift : Spec2	8	3	84	17	0.46	10	17	1.18	58	72	8	4	22
	12	3	116	25	17.98	10	25	3.64	58	72	8	4	22
	16	3	148	33	557.75	10	33	48.90	58	72	8	4	22
	8	4	84	17	0.29	10	17	3.04	124	126	11	6	25
	12	4	116	25	19.29	10	25	10.63	124	126	11	6	25
	16	4	148	33	576.56	10	33	128.40	124	126	11	6	25
	8	5	84	17	0.31	10	17	15.88	131	266	14	8	28
	12	5	116	25	19.47	10	25	283.90	131	266	14	8	28
	16	5	148	33	568.83	10	33	>1000	–	–	–	–	–

Fig. 4. Experimental results. Times are in seconds.

when changing floors, (iii) the lift must go through floors in the correct order, (iv) when a button is lit, the lift eventually reaches the corresponding floor and opens its doors, and finally (v) when the lift reaches a floor, the corresponding button becomes unlit. Let n be the number of floors. We apply our algorithms to check two properties which depend on a parameter k with $1 < k \leq n$, namely $\mathsf{Spec1}(k) = G((f_1 \wedge b_k) \rightarrow (\neg f_k U f_{k-1}))$, and $\mathsf{Spec2}(k) = G((f_1 \wedge b_k \wedge b_{k-1}) \rightarrow (b_k U \neg b_{k-1}))$.

Experimental results. All the results of our experiments are found in Fig. 4, and were performed on a quad-core 3,2 GHz Intel CPU with 12 GB of memory. We only report results for the concrete forward and abstract backward algorithms which were the fastest (by a large factor) in all our experiments. The columns of the table are as follows. *ATC* is the size of the largest antichain encountered, *iters* is the number of iterations of the fixed point in the concrete case and the maximal number of iterations of all the abstract fixed points in the abstract case, ATC^{α} and ATC^{γ} are respectively the sizes of the largest abstract and concrete antichains encountered, *steps* is the number of execution of the refinement steps and $|\mathcal{P}|$ is the maximum number of blocks in the partitions.

Benchmark 1. The partition sizes of the first benchmark illustrate how our algorithm exploits the locality of the property to abstract away the irrelevant parts of the system. For local properties, i.e. for small values of k, $|\mathcal{P}|$ is small compared to |Loc| meaning that the algorithm automatically ignores many subformulas which are irrelevant to the property. For larger values

of k, the overhead induced by the successive abstraction computations becomes larger, but it becomes less important as the system grows.

Benchmark 2. On the second benchmark, our abstract algorithm largely outperforms the concrete algorithm. Notice how for $k \geq 3$ the partition sizes do not continue to grow (it also holds for values of k beyond 5). This means that contrary to the concrete algorithm, the abstract algorithm does not get trapped in the intricate nesting of the F modalities (which are not necessary to prove the property) and abstracts it completely with a constant number of partition blocks. The speed improvement is considerable.

Benchmark 3. On this final benchmark, the abstract algorithm outperforms the concrete algorithm when the locality of the property spans less than 5 floors. Beyond that value, the abstract algorithm starts to take longer than the concrete version. From the *ATC* column, the antichain sizes remain constant in the concrete algorithm, when the number of floors increases. This indicates that the difficulty of this benchmark comes mainly from the exponential size of the alphabet rather than the state-space itself. As our algorithms only abstract the locations and not the alphabet, these results are not surprising.

7. Discussion

We have proposed in this paper two new abstract algorithms with refinement for deciding language emptiness for AFA. Our algorithm is based on an abstraction refinement scheme inspired from [7], which is different from the usual refinement techniques based on counter-example elimination [4]. Our algorithm also builds on the successful technique of antichains, that we have introduced in [8], to symbolically manipulate closed sets of cells (sets of locations). We have demonstrated with a set of benchmarks, that our algorithm is able to find coarse abstractions for complex automata constructed from large LTL formulas. For a large number of instances of those benchmarks, the abstract algorithms outperform by several orders of magnitude the concrete algorithms. We believe that this clearly shows the interest of our new algorithms and their potential future developments.

Several lines of future works can be envisioned. First, we should try to design a version of our algorithms where refinements are based on counter-examples and compare the relative performance of the two methods. Second, we have developed our technique for automata on finite words. We need to develop more theory to be able to apply our ideas to automata on infinite words. The fixed points involved in deciding emptiness for the infinite word case are more complicated (usually nested fixed points) and our theory must be extended to handle this case. Finally, it would be interesting to enrich our abstraction framework to deal with very large alphabets, possibly by partitioning the set of alphabet symbols.

Acknowledgements

The authors would like to thank Gilles Geeraerts for some fruitful discussions on the abstraction scheme.
The second author is supported by an FNRS-FRIA grant.

References

[1] A. Bouajjani, P. Habermehl, L. Holík, T. Touili, T. Vojnar, Antichain-based universality and inclusion testing over nondeterministic finite tree automata, in: CIAA, in: LNCS, vol. 5148, 2008, pp. 57–67.
[2] S. Burris, H.P. Sankappanavar, A Course in Universal Algebra, Springer, 1981.
[3] A.K. Chandra, D. Kozen, L.J. Stockmeyer, Alternation, J. ACM 28 (1) (1981) 114–133.
[4] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement for symbolic model checking, J. ACM 50 (5) (2003) 752–794.
[5] P. Cousot, Méthodes Itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes, Thèse d'état ès sciences mathématiques, Université scientifique et médicale de Grenoble, March 1978 (in French).
[6] P. Cousot, Partial completeness of abstract fixpoint checking, invited paper, in: SARA:'00: Proc. 4th Int. Symp. on Abstraction, Reformulations and Approximation, in: LNAI, vol. 1864, Springer, 2000, pp. 1–25.
[7] P. Cousot, P. Ganty, J.-F. Raskin, Fixpoint-guided abstraction refinements, in: SAS, in: LNCS, vol. 4634, Springer, 2007, pp. 333–348.
[8] M. De Wulf, L. Doyen, T.A. Henzinger, J.-F. Raskin, Antichains: a new algorithm for checking universality of finite automata, in: CAV, in: LNCS, vol. 4144, 2006, pp. 17–30.
[9] M. De Wulf, L. Doyen, N. Maquet, J.-F. Raskin, Alaska, in: ATVA, in: LNCS, vol. 5311, 2008, pp. 240–245.
[10] M. De Wulf, L. Doyen, N. Maquet, J.-F. Raskin, Antichains: alternative algorithms for LTL satisfiability and model-checking, in: TACAS, in: LNCS, vol. 4963, 2008.
[11] L. Doyen, J.-F. Raskin, Improved algorithms for the automata-based approach to model-checking, in: TACAS, in: LNCS, vol. 4424, Springer, 2007, pp. 451–465.
[12] S. Fogarty, M. Vardi, Buechi complementation and size-change termination, in: TACAS, in: LNCS, vol. 5505, 2009, pp. 16–30.
[13] P. Ganty, The fixpoint checking problem: an abstraction refinement perspective, Ph.D. Thesis, Université Libre de Bruxelles, 2007.
[14] P. Ganty, J.-F. Raskin, L. Van Begin, From many places to few: automatic abstraction refinement for petri nets, Fund. Inform. 88 (3) (2008) 275–305.
[15] J.-F. Raskin, K. Chatterjee, L. Doyen, T.A. Henzinger, Algorithms for omega-regular games with imperfect information, Logical Methods Comput. Sci. 3 (3) (2007).