Lattice Automata : a Representation for Languages on Infinite Alphabets, and some Applications to Verification

Tristan Le Gall[1] and Bertrand Jeannet[2]

[1] INRIA Rennes, tlegall@irisa.fr
[2] INRIA Rhônes-Alpes, bertrand.jeannet@inrialpes.fr

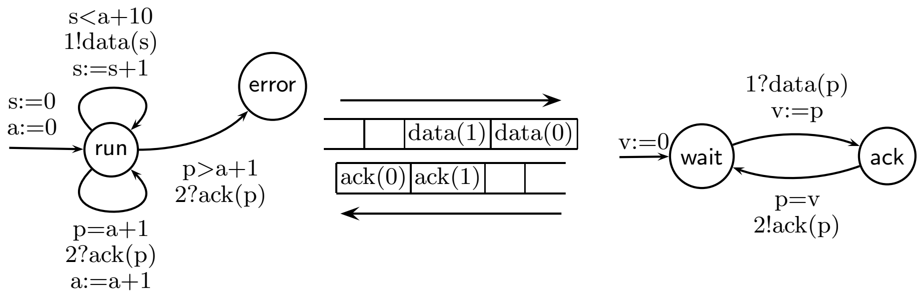
Abstract. This paper proposes a new abstract domain for languages on infinite alphabets, which acts as a functor taking an abstract domain for a concrete alphabet and lift it to an abstract domain for words on this alphabet. The abstract representation is based on lattice automata, which are finite automata labeled by elements of an atomic lattice. We define a normal form, standard language operations and a widening operator for these automata. We apply this abstract lattice for the verification of symbolic communicating machines, and we discuss its usefulness for interprocedural analysis.

1 Introduction

This paper proposes a new abstract domain for languages on infinite alphabets, which acts as a functor taking an abstract domain for a concrete alphabet and lift it to an abstract domain for words on this alphabet. This abstract domain can be used not only for abstracting words, but also for abstracting the *queue* or *stack* datatypes.

The initial motivation of this work was the analysis of communicating machines exchanging messages via FIFO queues, which typically models communication protocols. In [1], we presented a reachability analysis based on abstract interpretation for protocols with a finite alphabet of messages. The set of the queue contents was abstracted by regular languages equipped with a widening operator. We want to generalize this approach to protocols with an infinite number of messages. The need for infinite alphabets comes typically from the modeling of systems sending messages with integer parameters to FIFO channels.

This is the case for the very simplified version of a sliding window protocol depicted on Fig. 1. The *sender* process tries to send data (identified by an integer) to the *receiver* process. The receiver process sends an acknowledgement messages identifying the data received. The sender has two variables : s is the index of the next data to send, and a is the index of the last acknowledgement message received. The protocol ensures that the sender waits for acknowledgment if $s = a+10$. If the sender gets a message $\mathrm{ack}(p)$ with $p > a+1$, it means that at least one message has been lost and the protocol terminates with an error. There



(a) Sender (b) Queues (c) Receiver

Fig. 1. A simple sliding window protocol

are two queues: one from the sender to the receiver containing data messages, the other containing acknowledgement message. Notice that we do not model here possible loss of messages.

In order to provide an abstract domain for such FIFO queues, we introduce *lattice automata*, which are finite automata in which letters belonging to a finite alphabet are replaced by elements of an atomic lattice Λ. The idea is that a language like

$$Y = \sum_{n \geq 0} \mathit{data}(0) \cdot \ldots \cdot \mathit{data}(n)$$

can be abstracted by an interval lattice automaton recognizing

$$X = \mathit{data}(0) + \mathit{data}(0) \cdot \mathit{data}(1) + \mathit{data}(0) \cdot \mathit{data}(1) \cdot \big(\mathit{data}([2, +\infty])\big)^*$$

which represents all words $\mathit{data}(p_0) \ldots \mathit{data}(p_n)$ such that $p_0 = 0$, $p_1 = 1$, and $p_k \in [2, +\infty]$ for $k \geq 2$. This idea is rather simple, but making it work properly requires a detailed study.

Contributions. The first contribution of this paper is to define with lattice automata an effective and canonical representation of languages on infinite alphabets, equipped with well-defined operations (union, intersection, concatenation, ...). Although the normal form we propose induces in general some approximations, it is a robust notion in the sense that the normalization operator is an upper closure operator which returns the best upper-approximation of a language in the set of normalized languages. The resulting abstract domain allows to lift any *atomic* abstract domain A for $\wp(S)$ to an abstract domain $\mathrm{Reg}(A)$ for $\wp(S^*)$, the set of finite words defined on the alphabet S.

The second contribution of the paper is to demonstrate the use of this representation for the analysis of symbolic communicating machines (SCM). It appears indeed that this representation needs to be exploited in a clever way in order to be able to prove even simple properties, like in the example that messages in the queue 1 are always indexed by numbers smaller than the variable s. Our analysis shows that $a \leq s \leq a + 10$ (complete results are in section 5).

A third contribution is to show that lattice automata are also adapted to the abstraction of call-stacks in imperative programs, and allows to design potentially very precise interprocedural analysis.

Outline. Sect. 3 recalls some definitions about lattice and finite automata, and gives the definition of a widening operator for the regular languages lattice. The core contribution of the paper is Sect. 4 where we define lattice automata and their operations, which allows manipulating languages on infinite alphabet. In Sect. 5 we exploit this representation for the abstract interpretation of Symbolic Communicating Machines. Sect. 6 discuss the use of lattice automata for interprocedural analysis, as they allow abstracting call-stacks of imperative programs. Proofs are omitted but can be found in the companion report [2].

2 Related work

Many techniques have been devoted to the analysis of Communicating Finite-State Machines (CFSM), where both the machines and the alphabet of messages are *finite*. Reachability of CFSM is undecidable [3]. Most approaches for the verification of CFSM are based on exact but semi-decidable acceleration techniques [4, 6, 8]. A few attack the problem with approximate techniques [9, 10]. None of these works deals with a potentially infinite alphabet of messages.

More generally, there are works aiming at extending classical finite automata (resp. tree automata) representations for regular sets of words (resp. trees) for verification purposes. Mauborgne has proposed efficient representations for a class of sets of trees which strictly includes regular trees [11]. A recent work [**?**] introduces a different concept of "lattice automata", defined as finite (resp. Büchi) automata mapping finite (resp. infinite) words to elements of a finite lattice. Those automata do not represent finite words over an infinite alphabet, and do not apply to the verification of FIFO channels systems or interprocedural analysis. Another work [**?**] considers regular expressions over an infinite alphabet, which may be used instead of the lattice automata of this paper. A widening operator for regular expression is given, but does not work in the case of FIFO channels systems, because of the semantics of the send operation. Another approach is to focus on the decidability of some logic like the first order logic or the monadic second order logic when the model is a word with data or a tree with data (model of a XML document) [13]. New kind of automata were introduced, like register automata [14], pebble automata [15] or data automata [16], with the idea that a word with data satisfies the logical formula if it is recognized by the corresponding automata. This approach cannot be applied as is to our problem, as such a logical approach does not take into account any specific logical interpretation for data (in other words, the data domain is unspecified) and there is no notion of approximation.

They have been recently a lot of works devoted to shape analysis which can be relevant to the analysis of queues or stacks. A FIFO queue or a stack can indeed be represented by a list, which is most often the easiest data type to abstract in shape analysis techniques. However, most shape analysis focus on the structure ("the shape") of the memory and ignores data values hold by the memory cells [17–20]. It is sometimes possible to handle finite data values, but mainly by brute force enumeration, which is algorithmically expensive in such a context, certainly more than the above-cited approaches based on finite automata. [21] is a pioneering work for taking into account data values in memory cells. It uses a global polyhedron to relate the numeric contents of each abstract memory cells in an abstract shape graph. The resulting abstraction is incomparable to our proposal, as it is based on very different principles. In particular, the information used for abstraction in shape graphs is mostly attached to nodes instead of edges as in automata. The involved algorithms are also probably more expensive than in our solution. Conversely, it should be noted that our abstract domain could be applied to shape analysis, although this deserves yet a full study.

3 Preliminaries

A *finite automaton* is a quintuple $\mathcal{A} = (Q, \Sigma, Q_0, Q_f, \delta)$ where Q is a finite set of states, Σ a finite alphabet, $Q_0 \subseteq Q$ (resp. $Q_f \subseteq Q$) the set of initial (resp. final) states, and $\delta \subseteq Q \times \Sigma \times Q$ the transition relation. The set of words recognized by $\mathcal{A}$ is a regular language denoted by $L(\mathcal{A})$. $\mathrm{Reg}(\Sigma)$ is the set of regular languages over the finite alphabet Σ. Let $\approx$ be an equivalence relation on Q. The equivalence class of $q \in Q$ w.r.t. $\approx$ is denoted by $\widetilde{q}$. The *quotient automaton* $\mathcal{A}/\approx = \langle \widetilde{Q}, \Sigma, \widetilde{Q_0}, \widetilde{Q_f}, \widetilde{\delta} \rangle$ is defined by $\widetilde{Q} \triangleq Q/\approx$, $\widetilde{Q_0} \triangleq \{\widetilde{q} | q \in Q_0\}$, $\widetilde{Q_f} \triangleq \{\widetilde{q} | q \in Q_f\}$ and $(\widetilde{q}, a, \widetilde{q'}) \in \widetilde{\delta} \triangleq \exists q_0 \in \widetilde{q}, \exists q_0' \in \widetilde{q'} : (q_0, a, q_0') \in \delta$. Given an equivalence relation $\simeq$ on Q and $k \geq 0$, the *k-depth bisimulation* equivalence relation $\approx_k$ based on $\simeq$ is defined by $\approx_0 \triangleq \simeq$ and

$$q \approx_{k+1} q' \triangleq \begin{cases} \forall (\sigma, q_1) \in \Sigma \times Q : \delta(q, \sigma, q_1) \Rightarrow \exists q_1' : \delta(q', \sigma, q_1') \wedge q_1 \approx_k q_1' \\ \forall (\sigma, q_1') \in \Sigma \times Q : \delta(q', \sigma, q_1') \Rightarrow \exists q_1 : \delta(q, \sigma, q_1) \wedge q_1 \approx_k q_1' \end{cases}$$

$\mathcal{A}$ is *deterministic* if there is an unique initial state and if $\delta(q, \sigma, q') \wedge \delta(q, \sigma, q'') \Rightarrow q' = q''$. Any finite automaton can be transformed into a minimal deterministic automaton (MDA) recognizing the same language, using the subset construction and quotienting the result w.r.t. the largest bisimulation relation separating final states from non final states. This provides a normal form for regular languages.

Widening on finite automata. $(\mathrm{Reg}(\Sigma), \subseteq)$ is a lattice which can be equipped with the following widening operator [22, 1]. One first defines the extensive and idempotent operator ρ_k, which associates to a MDA $\mathcal{A}$ its quotient by the k-depth bisimulation relation based on the partition of the Q which separates exactly initial, final and ordinary states. The widening ∇_k is then defined as $X_1 \nabla_k X_2 \triangleq \rho_k(X_1 \cup X_2)$. This operator is a widening as its codomain is finite.

Lattices. An element $x \in \Lambda$ of a lattice Λ is an *atom* if $\bot \sqsubset x \wedge \forall y \in \Lambda : \bot \sqsubset y \sqsubseteq x \implies y = x$. The set of atoms of Λ is denoted by $At(\Lambda)$. A lattice Λ is *atomic* if any element $x \in \Lambda$ is the least upper bound of atoms smaller than it: $x = \bigsqcup\{a \mid a \in At(\Lambda) \wedge a \sqsubseteq x\}$. A finite partition of a lattice is a finite set $(\lambda_i)_{0 \leq i < n}$

of elements such that $\forall i \neq j : \lambda_i \sqcap \lambda_j = \bot$ and $\forall \lambda \in \Lambda : \lambda = \bigsqcup_{i=0}^{n-1}(\lambda \sqcap \lambda_i)$. If the lattice is atomic, there is an isomorphism between an element $\lambda \in \Lambda$ and its projection $\langle \lambda \sqcap \lambda_0, \ldots, \lambda \sqcap \lambda_{n-1} \rangle$ on the partition. A (finite) *partitioning function* $\pi : \Sigma \to \Lambda$ is a function such that $(\pi(\sigma)_{\sigma \in \Sigma})$ is a (finite) partition of Λ.

4 Lattice automata

In this section we define with *lattice automata* an abstract representation for languages on infinite alphabets. The principle of lattice automata is to use elements of an atomic lattice for labelling the transitions of a finite automaton, and to use a partition of this lattice in order to define a projected finite automaton which acts as a guide for defining extensions of the classical finite automata operations. We motivate our choices and show that they lead to a robust notion of approximation, in the sense that normalization is an upper-closure operation and can be seen as a best upper-approximation in the lattice of normalized lattice automata. Proofs are omitted but can be found in the companion report [2].

Lattice automata are finite automata, the transitions of which are labelled by elements of an atomic lattice $(\Lambda, \sqsubseteq)$ instead of elements of a finite and unstructured alphabet. They recognize languages on atomic elements of this lattice. For instance, the interval automaton of Fig. 2 recognizes all sequences of rational numbers $x_0 \ldots x_{n-1}$ where n is odd, $x_{2i} \in [0, 1]$ and $x_{2i+1} \in [1, 2]$.

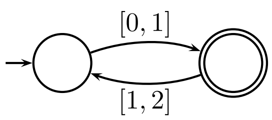


Fig. 2. an interval lattice automaton

Definition 1 (Lattice automaton). *A lattice automaton $\mathcal{A}$ is defined by a tuple $\langle \Lambda, Q, Q_0, Q_f, \delta \rangle$ where $(\Lambda, \sqsubseteq)$ is an atomic lattice, Q a finite set of states, $Q_0 \subseteq Q$ and $Q_f \subseteq Q$ the sets of initial and final states and $\delta \subseteq Q \times (\Lambda \setminus \{\bot\}) \times Q$ a finite transition relation.*

A word $w = a_0 \ldots a_n \in At(\Lambda)^*$ is accepted by $\mathcal{A}$ if there exists a sequence $q_0, q_1, \ldots, q_{n+1}$ such that $q_0 \in Q_0$, $q_{n+1} \in Q_f$, and $\forall i \leq n$, $\exists (q_i, \lambda_i, q_{i+1}) \in \delta : a_i \sqsubseteq \lambda_i$. The set of words recognized by $\mathcal{A}$ is denoted by $L_{\mathcal{A}}$. We denote by $\mathrm{Reg}(\Lambda)$ the set of languages recognized by a lattice automaton. The inclusion relation between languages induces a partial order on lattice automata: $\mathcal{A} \sqsubseteq \mathcal{A}'$ iff $L_{\mathcal{A}} \subseteq L_{\mathcal{A}'}$.

Discussion. Def. 1 raises some issues that we discuss before refining this definition. We need indeed to provide a normal form for lattice automata, in order to use them as a well-defined abstract domain with a robust notion of approximation. The first issue is related to the bounded branching property: in a deterministic finite automaton, there are at most $|\Sigma|$ transitions outgoing from a state, whereas the branching degree of lattice automata is not bounded, *cf.* Fig. 3. The second issue is the notion of determinism. The natural definition of determinism would require that $\mathcal{A}$ is deterministic *iff* it has a unique initial state

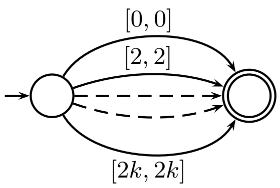
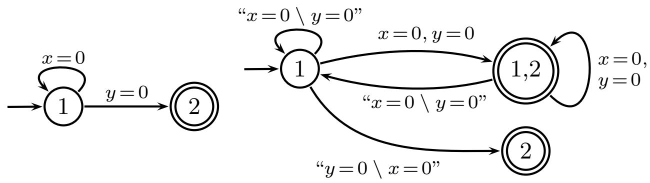


Fig. 3. A family of interval automata $\mathcal{A}_k$ with unbounded branching degree



(a) non-deterministic automaton

(b) deterministic automaton ?

Fig. 4. Attempt to determinize a lattice automaton on the lattice of affine equalities

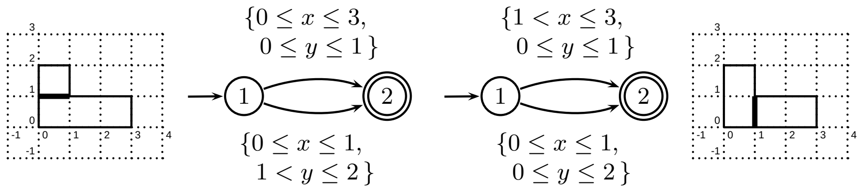


Fig. 5. Two deterministic convex polyhedra automata that are equivalent

and if $(q, \lambda_1, q_1) \in \delta \wedge (q, \lambda_2, q_2) \in \delta \Longrightarrow \lambda_1 \sqcap \lambda_2 = \bot$. However the automaton of Fig. 4 based on the lattice of affine equalities[3] shows that one cannot always find a deterministic automaton recognizing the same language as a given non-deterministic automaton, because an element of a lattice does not necessarily have a complement. Now, determinism is more or less a requirement for defining a normal form. Fig. 5 illustrates a third annoying issue : as there is no canonical representation for unions of convex polyhedra [**?**], how do we decide which one among the two minimal automata of the figure should be considered canonical ?

The solution we propose to fix these problems is to use a finite partition of the lattice Λ, and to decide when two transitions should be merged using the least upper bound operator. The fusion of transitions will induce in general an over-approximation, controlled by the fineness of the partition. The gain is that the projection of labels onto their equivalence class produces a finite automaton on which we can reuse classical notions.

Definition 2 (Partitioned lattice automaton (PLA)). *A partitioned lattice automaton $\mathcal{A}$ is a lattice automaton $\mathcal{A} = \langle \Lambda, \pi, Q, Q_0, Q_f, \delta \rangle$ equipped with a finite partitioning function $\pi : \Sigma \to \Lambda$ such that the transition relation satisfies: $\forall (q, \lambda, q') \in \delta,\ \exists \sigma \in \Sigma : \lambda \sqsubseteq \pi(\sigma)$.*
A PLA is merged *if: $(q, \lambda_1, q') \in \delta \wedge (q, \lambda_2, q') \in \delta \Longrightarrow \pi^{-1}(\lambda_1) \cap \pi^{-1}(\lambda_2) = \emptyset$*

Definition 3 (Shape automaton). *Given a PLA $\mathcal{A} = \langle \Lambda, \pi, Q, Q_0, Q_f, \delta \rangle$, its shape automaton* shape$(\mathcal{A})$ *is a finite automaton $(\Sigma, Q, Q_0, Q_f, \to)$ obtained by*

[3] The lattice of affine equalities (or affine subspaces) [23] is the lattice formed by the conjunctions of affine equalities on the space $\mathbb{R}^n$

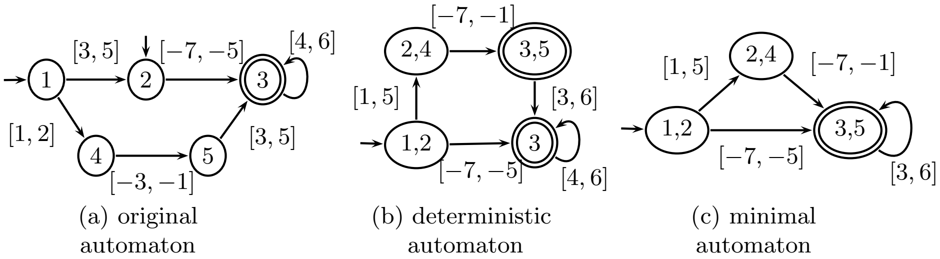


Fig. 6. Determinization and minimization of an interval automaton with the partition $\{]-\infty, 0], [0, +\infty[\}$

projecting the transition relation δ onto the equivalence classes:

$$(q, \lambda, q') \in \delta \implies q \overset{\pi^{-1}(\lambda)}{\to} q'$$

Two transitions of a PLA labelled by elements belonging to different equivalence classes cannot be merged and are always kept separate, whereas they might be merged in the opposite case. Deterministic merged PLA have the finite branching degree property: its states can have at most $|\Sigma|$ outgoing transitions. Notice that non-merged PLA are as expressive as lattice automata.

Definition 4. *A PLA $\mathcal{A}$ is deterministic if* shape($\mathcal{A}$) *is deterministic.*

4.1 Normalization of PLAs

We use the shape automaton as a guideline for the determinization and minimization of lattice automata.

Determinization. The determinization of a PLA, which is illustrated on Fig. 6, mimics the determinization of its shape automaton using the subset construction on states. The difference is that the transitions are merged in the course of the algorithm when they are labelled by values belonging to the same equivalence class. Consequently, the resulting automaton recognizes a larger language. The precise determinization algorithm can be found in [2]. This algorithm gives the best approximation in term of inclusion of languages.

Proposition 1 (Determinising PLA is a best upper-approximation). *Let $\mathcal{A}$ be a PLA and $\mathcal{A}'$ the PLA obtained with the determinization algorithm sketched above. Then $\mathcal{A}'$ is the best upper-approximation of $\mathcal{A}$ as a merged and deterministic PLA, ie., $\mathcal{A} \sqsubseteq \mathcal{A}'$ and for any merged and deterministic PLA $\mathcal{A}''$ based on the same partition, $\mathcal{A} \sqsubseteq \mathcal{A}'' \implies \mathcal{A}' \sqsubseteq \mathcal{A}''$.*

Corollary 1 (The determinization operation is an upper-closure operation). *The operation det* : PLA $\to$ PLA *is an upper-closure operation: it is extensive ($det(\mathcal{A}) \sqsupseteq \mathcal{A}$), monotonic and idempotent ($det \circ det = det$).*

Minimization. We use for PLA a notion of minimization based on its shape automaton, in the same spirit as for the notion of determinism. A PLA is *minimal* or *normalized* if it is merged and if its shape automaton is minimal and deterministic. A normalized PLA will be also called a NLA (normalized lattice automaton). The algorithm to minimize a PLA consists in to remove its unconnected states, to determinize it according to the previous algorithm and to quotient it according to the equivalence bisimulation relation as defined on the states of its shape automaton (*cf.* Sect. 3). However, when quotienting the states of a PLA, transitions labelled by elements belonging to the same equivalence class are merged, which may generate some approximations, *cf.* Fig. 6.

Theorem 1 (Minimizing PLA is a best upper-approximation). *For any PLA $\mathcal{A}$, there is a unique (up to isomorphism) NLA $\mathcal{A}'$ based on the same partition π such that $\mathcal{A} \sqsubseteq \mathcal{A}'$ and for any NLA $\mathcal{A}''$ based on the partition π, $\mathcal{A} \sqsubseteq \mathcal{A}'' \Longrightarrow \mathcal{A}' \sqsubseteq \mathcal{A}''$.*

Corollary 2 (The normalization operation is an upper-closure operation). *The normalization function* $\widehat{\cdot} : \mathrm{PLA} \to \mathrm{NLA}$ *is an upper-closure operator: it is extensive, monotonic (given a fixed partition), and idempotent.*

Thm. 1 defines a normalization for languages recognized by PLA. For any language $L \in \mathrm{Reg}(\Lambda)$ recognized by a PLA $\mathcal{A}$, $\widehat{L}$ will denote the language recognized by the unique NLA verifying the properties of Thm. 1. The set of NLA defined on Λ with the partition π will be denoted by $\mathrm{Reg}(\Lambda, \pi)$, which denotes also the corresponding set of recognized languages.

Influence of the partitioning functions. The precision of the approximations made during the merging, determinization and minimization operations depends on the fineness of the partitioning function. For example, all outgoing transitions from a given state would be merged during the determinization algorithm employed with the trivial partition of size 1.

Refining the partition π makes the class of normalized languages $\mathrm{Reg}(\Lambda, \pi)$ more expressive. A partitioning function $\pi_2 : \Sigma_2 \to \Lambda$ *refines* $\pi_1 : \Sigma_1 \to \Lambda$ if $\forall \sigma_2 \in \Sigma_2, \exists \sigma_1 \in \Sigma_1 : \pi_2(\sigma_2) \sqsubseteq \pi_1(\sigma_1)$.

Proposition 2. *Let π_1 and π_2 two partitioning functions for Λ, with π_2 refining π_1. Then* $\mathrm{Reg}(\Lambda, \pi_1) \subseteq \mathrm{Reg}(\Lambda, \pi_2)$.

One can also refine PLAs before normalizing them. The automaton $\mathcal{A}_2 = \langle \Lambda, \pi_2, Q, Q_0, Q_f, \delta_2 \rangle$ *refines* $\mathcal{A}_1 = \langle \Lambda, \pi_1, Q, Q_0, Q_f, \delta_1 \rangle$ if π_2 refines π_1 and if δ_2 is obtained with:

$$\frac{(q, \lambda_1, q') \in \delta_1 \qquad \sigma_2 \in \Sigma_2 \qquad \lambda_2 = \lambda_1 \sqcap \pi(\sigma_2)}{(q, \lambda_2, q') \in \delta_2}$$

This refinement does not change the recognized language, but may increase the precision of the merging, determinization and minimization algorithms :

Proposition 3. *Let $\mathcal{A}_1$ be a PLA and $\mathcal{A}_2$ a PLA refining $\mathcal{A}_1$. Then $det(\mathcal{A}_1) \sqsupseteq det(\mathcal{A}_2)$ and $\widehat{\mathcal{A}_1} \sqsupseteq \widehat{\mathcal{A}_2}$.*

Choosing an adequate partitioning function is thus important. For the analysis of SCM, where data messages are usually composed of a message type and some parameters, the type of the message defines a natural partition. When this standard partition is not sufficient for the analysis, it can be refined to a more adequate partition. In this sense, the abstraction refinement techniques based on partitioning (see for instance [24, 25]) are applicable to lattice automata.

4.2 Operations on PLAs and their recognized languages

Set operations. Two NLAs can be compared for language by first testing for inclusion their shape automata, and in case of inclusion, by comparing the labels of matching transitions. The exact union of two NLAs can be computed by considering their disjoint union; this produces a non-deterministic PLA with two initial states. Normalizing it transforms the exact union operation in an upper bound operator $\sqcup$ defined as follows: $\mathcal{A}_1 \sqcup \mathcal{A}_2 \triangleq \widehat{\mathcal{A}_1 \cup \mathcal{A}_2}$. As a corollary of Theorem 1, this upper bound operator is actually a *least upper bound operator* on the set of NLA ordered by language inclusion. The exact intersection of two NLAs can be computed by considering their product. The result is a merged and deterministic PLA, which is not necessarily minimal. Minimizing it may induce an upper-approximation; thus, NLAs are not closed under intersection. These operations allow us to equip the set of NLA with a join semilattice structure:

Proposition 4. *The set of NLA defined on an atomic lattice Λ with a fixed partition π, ordered by language inclusion, is a join semilattice: it has bottom and top elements, and a least upper bound operator. If the standard lattice operations on Λ are computable, so are the operations on the join semilattice of NLA.*

Other language operations. Operation such as language concatenations or (the generalisation of) left derivation of NLAs mimics their counterparts on finite automata, with the difference that the normalization following them induces approximations in term of languages. Another useful operation for the analysis of communicating machines is the *first* operation. Left derivation and *first* operations are defined on languages as: $L/\lambda \triangleq \{\omega \mid \exists a \in At : a \cdot \omega \in L \wedge a \sqsubseteq \lambda\}$ and $first(L) = \{a \in At(\Lambda) \mid a \cdot \omega \in L\}$.

Widening on NLAs. The widening on NLA we define here combines the widening on finite automata with the standard widening operator $\nabla_\Lambda : \Lambda \times \Lambda \to \Lambda$ that we assume for the atomic lattice Λ. If a widening operator is not strictly required for Λ (because it satisfies the ascending chain condition), then one takes $\nabla_\Lambda = \sqcup_\Lambda$.

We first extend the ρ_k operator defined on finite automata in Sect 3. If $\mathcal{A}$ is a NLA, $k \geq 0$ an integer and $\approx_k$ the k-depth bisimulation relation defined on shape($\mathcal{A}$), then $\rho_k(\mathcal{A})$ is defined as the quotient PLA $\mathcal{A}/\approx_k$. The widening

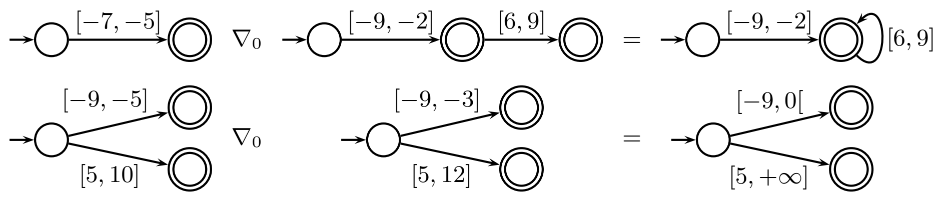


Fig. 7. Widening on interval PLA, with a partition $[-\infty, 0[\sqcup[0, +\infty]$ and $k = 0$

operator we suggest consists in applying the operator ρ_k when the two argument automata have a different shape automaton, and to apply the widening operator of the lattice Λ on their matching transitions when the two argument automata have the same shape automaton, as illustrated by Fig. 7.

Definition 5. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two NLA defined on the same partition π with $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$. The widening operator ∇_k is defined as :*

$$\mathcal{A}_1 \nabla_k \mathcal{A}_2 \triangleq \begin{cases} \widehat{\rho_k(\mathcal{A}_2)} & \textit{if } \mathrm{shape}(\mathcal{A}_1) \neq \mathrm{shape}(\rho_k(\mathcal{A}_2)) \\ \mathcal{A}_1 \nearrow \mathcal{A}_2 & \textit{otherwise (which implies } \mathrm{shape}(\mathcal{A}_1) = \mathrm{shape}(\mathcal{A}_2)\textit{)} \end{cases}$$

where $\mathcal{A}_1 \nearrow \mathcal{A}_2$ is the NLA $\mathcal{A}$ which has the same set of states as $\mathcal{A}_1$ and $\mathcal{A}_2$ and the set of transitions δ defined by the rule:

$$\frac{\sigma \in \Sigma \quad (q, \lambda_1, q') \in \delta_1 \quad (q, \lambda_2, q') \in \delta_2 \quad \lambda_1, \lambda_2 \sqsubseteq \pi(\sigma)}{(q, (\lambda_1 \nabla_\Lambda \lambda_2) \sqcap \pi(\sigma), q') \in \delta}$$

If $\mathcal{A}_1 \not\sqsubseteq \mathcal{A}_2$, then $\mathcal{A}_1 \nabla_k \mathcal{A}_2 \triangleq \mathcal{A}_1 \nabla_k (\mathcal{A}_1 \sqcup \mathcal{A}_2)$.

Theorem 2. *∇_k is a proper widening operator:*

1. *for any NLA $\mathcal{A}1, \mathcal{A}_2$ defined on the same partition such that $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$, $\mathcal{A}_2 \sqsubseteq \mathcal{A}_1 \nabla_k \mathcal{A}_2$;*
2. *If there is an increasing chain of NLA $\mathcal{A}_0 \sqsubseteq \mathcal{A}_1 \sqsubseteq \ldots \sqsubseteq \mathcal{A}_n \sqsubseteq \ldots$, the chain $\mathcal{A}'_0 \sqsubseteq \mathcal{A}'_1 \sqsubseteq \ldots \sqsubseteq \mathcal{A}'_n \sqsubseteq \ldots$ defined as $\mathcal{A}'_0 = \mathcal{A}_0$ and $\mathcal{A}'_{i+1} = \mathcal{A}'_i \nabla_k (\mathcal{A}'_i \sqcup \mathcal{A}_{i+1})$ is not strictly increasing.*

4.3 NLA as an abstract domain for languages, stacks and queues.

Normalized lattice automata allow us to define an *abstract domain functor* which lifts abstract domains for some set to abstract domains for languages on this set. More precisely, given an *atomic* abstract lattice $A \xrightarrow{\gamma_A} \wp(S)$ for some set S with the concretization function γ_A, and a partitioning function π for A, $\mathrm{Reg}(A, \pi)$ can be viewed as an abstract domain for $\mathcal{L}(S) = \wp(S^*)$, the languages on elements of S, with the concretization function

$$\begin{aligned} \gamma :\ & \mathrm{Reg}(A, \pi) \to \wp(S^*) \\ & \mathcal{A} \quad\ \mapsto \{s_0 \ldots s_n \in S^* \mid a_0 \ldots a_n \in L_{\mathcal{A}} \wedge \forall i : s_i \in \gamma_A(a_i)\} \end{aligned}$$

and the widening operator of Def. 5. $\mathrm{Reg}(A, \pi)$ is a non-complete join semilattice of infinite height.

The "abstract domain functor" $\mathrm{Reg}(A, \pi)$ is in addition monotonic in the following sense. Given two abstractions $\gamma_i : A_i \to C, i = 1, 2$, for the same concrete domain C, A_2 *refines* A_1 if there exists a (concretization) function $\gamma_{12} : A_1 \to A_2$ such that $\gamma_1 = \gamma_2 \circ \gamma_{12}$. This means that any concrete property $c \in C$ representable by A_1 is representable by A_2.

Theorem 3. *Let $\gamma_i : A_i \to \wp(S), i = 1, 2$, two abstractions for $\wp(S)$ such that A_2 refines A_1, with $\gamma_{12} : A_1 \to A_2$ such that $\gamma_1 = \gamma_2 \circ \gamma_{12}$. Let π_1 a partitioning function for A_1, and π_2 a partitioning function for A_2 refining $\gamma_{12} \circ \pi_1$. The abstract domain* $\mathrm{Reg}(A_2, \pi_2)$ *refines* $\mathrm{Reg}(A_1, \pi_1)$.

Hence an abstraction of $\wp(S^*)$ based on lattice automata is parametrised first by the underlying alphabet lattice A, then by a partition π of A, and last by the parameter k of the widening operator ∇_k.

Most language operations can be efficiently abstracted in $\mathrm{Reg}(A, \pi)$. This is in particular the case of language operations corresponding to the operations offered by the *FIFO queue* or *stack* abstract datatypes. Hence, $\mathrm{Reg}(A, \pi)$ is a suitable abstract domain for FIFO queues or stacks on elements of S.

5 Application to the abstract interpretation of SCMs

We illustrate in this section the application of the abstract lattice defined in the previous section to the analysis of Symbolic Communicating Machines (SCMs).

Symbolic Communicating Machines. Symbolic Communicating Machines (SCM) are Communicating Finite-State Machine extended with a finite set of variables V, the values of which can be sent into FIFO queues, *cf.* Fig. 1. This model is similar to other models like Extended Communicating Finite-State Machines [26] or Parametrized Communicating Finite-State Machines [27].

Syntax. A SCM with N queues is defined by a tuple $\langle C, V, c_0, \Theta_0, P, \Delta\rangle$ where C is a nonempty finite set of locations; $V = \{v_1, \dots, v_n\}$ is a nonempty, finite set of *state* variables; $c_0 \in C$ is the initial control state; $\Theta_0(\boldsymbol{v})$ is the *initial condition*; $P = \{p_1, \dots, p_n\}$ is a nonempty, finite set of formal parameters that are used to push/pop values on/from FIFO queues; and Δ is a finite set of *transitions*.
A transition δ is either an *input* $\langle c_1, G, i?\boldsymbol{p}, A, c_2\rangle$ or an *output* $\langle c_1, G, i!\boldsymbol{p}, A, c_2\rangle$ where c_1 and c_2 are resp. the origin and destination locations; $i \in [1..N]$ is a queue number; $\boldsymbol{p}$ is the vector of formal parameters, which holds the values pushed or popped on/from the queue i; $G(\boldsymbol{v}, \boldsymbol{p})$ is a predicate on the variables and the formal parameters (also called guard); and A is an assignment of the form $\boldsymbol{v'} := A(\boldsymbol{v}, \boldsymbol{p})$ defining the values of the state variables after the transition.

Semantics. In the sequel, a variable v takes its value in a set denoted by $\mathcal{D}_v$, and the set of valuations of all variables in V is denoted by D_V. A global state of a SCM is a tuple $\langle c, \boldsymbol{v}, w_1, \ldots, w_N\rangle \in C \times \mathcal{D}_V \times ((\mathcal{D}_P)^*)^N$ where c is a control state, $\boldsymbol{v}$ is the current value of the variables and w_i is a finite word on $\mathcal{D}_P$ representing the content of queue i. The operational semantics of a SCM $\langle C, V, \Sigma, c_0, \Theta_0, P, \Delta\rangle$ is given as an infinite transition system $\langle Q, Q_0, \rightarrow\rangle$ where $Q = C \times \mathcal{D}_V \times ((\mathcal{D}_P)^*)^N$ is the set of states; $Q_0 = \{\langle c_0, \boldsymbol{v}, \varepsilon, \ldots, \varepsilon\rangle \mid \boldsymbol{v} \in \Theta_0\}$ is the set of the initial states; and $\rightarrow$ is defined by the two rules:

$$\frac{(c_1, G, i!\boldsymbol{p}, A, c_2) \in \Delta \qquad w_i' = w_i \cdot \boldsymbol{p} \qquad G(\boldsymbol{v}, \boldsymbol{p}) \qquad \boldsymbol{v'} = A(\boldsymbol{v}, \boldsymbol{p})}{\langle c_1, \boldsymbol{v}, w_1, \ldots, w_i, \ldots, w_N\rangle \rightarrow \langle c_2, \boldsymbol{v'}, w_1, \ldots, w_i', \ldots, w_N\rangle}$$

$$\frac{(c_1, G, i?\boldsymbol{p}, A, c_2) \in \Delta \qquad w_i = \boldsymbol{p}.w_i' \qquad G(\boldsymbol{v}, \boldsymbol{p}) \qquad \boldsymbol{v'} = A(\boldsymbol{v}, \boldsymbol{p})}{\langle c_1, \boldsymbol{v}, w_1, \ldots, w_i, \ldots, w_N\rangle \rightarrow \langle c_2, \boldsymbol{v'}, w_1, \ldots, w_i', \ldots, w_N\rangle}$$

The main issue for the reachability analysis of SCM is the abstraction of the queue contents. [1] described and experimented the abstraction of the queue contents of a CFSM using the regular languages lattice equipped with the widening operator of Sect. 3. Lattice automata allow us to attack the analysis of SCM. For the sake of simplicity, we assume a single queue in the sequel. The running example which has two queues is analysed with a *non-relational, attribute-independent* method, in which to each queue is associated a lattice automaton,

Analysing SCM with a single queue: a simple approach. If there is a single queue, the concrete set of states associated to each control point $c \in C$ has the structure $\wp(\mathcal{D}_V \times (\mathcal{D}_P)^*)$: one associates to each control point the set of possible configurations for the state variables and the FIFO queue. For the sake of simplicity, we will assume in the sequel that all variables and parameters are of rational type, and that sets of valuations are abstracted using convex polyhedra. Let $L^{(n)} = \mathsf{Pol}(\mathbb{Q}^n)$ denotes the atomic lattice of convex polyhedra. We have the abstraction chain

$$\wp(\mathbb{Q}^n \times (\mathbb{Q}^p)^*) \leftrightarrows \wp(\mathbb{Q}^n) \times \wp((\mathbb{Q}^p)^*) \longleftarrow L^{(n)} \times \mathrm{Reg}(L^{(p)}) \stackrel{\triangle}{=} A^{(n)}$$

Tab. 1 gives the corresponding abstract semantics.

We experimented with this analysis on the sliding window protocol depicted in Fig. 1, using a generic fixpoint calculator and the APRON library [28], with no partitioning of the alphabet lattice. Tab. 2 depicts the obtained reachable set on the product of the two automata. This straightforward abstraction is not very accurate since there is no relation between the values of parameter variables in the queue and the value of the state variables.

SCM with a single queue: linking message and state variables. The idea to improve on the previous abstraction is to use an augmented semantics in which both the state variables and the message are pushed on queues. This allows not only to establish relations between messages in queues and the current environment, but also to indirectly establish relations between the messages contained in different queues. For instance, the abstract value

$$\big(s \in [8,9],\ \mathit{data}(\{s-p=3\}) \cdot \mathit{data}(\{s-p=2\}) \cdot \mathit{data}(\{s-p=1\})\big)$$

Standard abstract semantics			
guard	$G(\boldsymbol{v},\boldsymbol{p})$	$[\![G]\!]^\sharp : L^{(n)} \to L^{(n+p)}$	extended to $A^{(n)} \to A^{(n+p)}$
assignment	$A(\boldsymbol{v},\boldsymbol{p})$	$[\![A]\!]^\sharp : L^{(n+p)} \to L^{(n)}$	extended to $A^{(n+p)} \to A^{(n)}$
output	$1!\boldsymbol{p}$	$[\![1!\boldsymbol{p}]\!]^\sharp : A^{(n+p)} \to A^{(n+p)}$ $(Y,F) \mapsto (Y, F\cdot(\exists \boldsymbol{v} : Y))$	
input	$1?\boldsymbol{p}$	$[\![1?\boldsymbol{p}]\!]^\sharp : A^{(n+p)} \to A^{(n+p)}$ $(Y,F) \mapsto (Y \sqcap \mathit{first}(F),\ F/(\exists \boldsymbol{v} : Y))$	
transition	t	$[\![t]\!]^\sharp : A^{(n)} \to A^{(n)}$ $X \mapsto \begin{cases} [\![A]\!]^\sharp \circ [\![G]\!]^\sharp & \text{if } t=(G,--,A) \\ [\![A]\!]^\sharp \circ [\![1!\boldsymbol{p}]\!]^\sharp \circ [\![G]\!]^\sharp & \text{if } t=(G,1!\boldsymbol{p},A) \\ [\![A]\!]^\sharp \circ [\![1?\boldsymbol{p}]\!]^\sharp \circ [\![G]\!]^\sharp & \text{if } t=(G,1?\boldsymbol{p},A) \end{cases}$	

Table 1. Abstract semantics, with $L^{(n)} = \mathsf{Pol}(\mathbb{Q}^n)$ and $A^{(n)} = L^{(n)} \times \mathrm{Reg}(L^{(p)})$

C	Abstract Value
rw	$[\|v \geq 0; s \geq 0; a \geq 0\|]$ $(d, [\|p \geq 0\|])^*$ $(a, [\|p \geq 0\|])^*$
ra	$[\|v \geq 0; s \geq 0; a \geq 0\|]$ $(d, [\|p \geq 0\|])^*$ $(a, [\|p \geq 0\|])^*$
ew	$[\|v \geq 0; s \geq 0; a \geq 2\|]$ $(d, [\|p \geq 0\|])^*$ $(a, [\|p \geq 0\|])^*$
ea	$[\|v \geq 0; s \geq 0; a \geq 2\|]$ $(d, [\|p \geq 0\|])^*$ $(a, [\|p \geq 0\|])^*$

Table 2. Analysis with the "simple" abstraction

C	Abstract Value
rw	$[\|0 \leq a \leq s \leq a+10\|]$ $(d, [\|0 \leq a \leq s-1 \leq a+9; 0 \leq p \leq s-1\|])^*$ $(a, [\|0 \leq a \leq s \leq a+10; 0 \leq p \leq s-1; 0 \leq v \leq s-1\|])^*$
ra	$[\|0 \leq a \leq s \leq a+10; 0 \leq v \leq s-1\|]$ $(d, [\|0 \leq a \leq s-1 \leq a+9; 0 \leq p \leq s-1\|])^*$ $(a, [\|0 \leq a \leq s \leq a+10; 0 \leq p \leq s-1; 0 \leq v \leq s-1\|])^*$
ew	$[\|0 \leq a \leq s-2 \leq a+8; 0 \leq v \leq s-1\|]$ $(d, [\|0 \leq a \leq s-1 \leq a+9; 0 \leq p \leq s-1\|])^*$ $(a, [\|0 \leq a \leq s \leq a+10; 0 \leq p \leq s-1; 0 \leq v \leq s-1\|])^*$
ea	$[\|0 \leq a \leq s-2 \leq a+8; 0 \leq v \leq s-1\|]$ $(d, [\|0 \leq a \leq s-1 \leq a+10; 0 \leq p \leq s-1\|])^*$ $(a, [\|0 \leq a \leq s \leq a+10; 0 \leq p \leq s-1; 0 \leq v \leq s-1\|])^*$

Table 3. Analysis with the refined abstraction, using a widening "up to" for polyhedra

will represent the 2 concrete states $(s = 8, \mathit{data}(5) \cdot \mathit{data}(6) \cdot \mathit{data}(7))$ and $(s = 9, \mathit{data}(6) \cdot \mathit{data}(7) \cdot \mathit{data}(8))$.

Formally, the new abstraction is defined by

$$\begin{array}{rll} \gamma : & L^{(n)} \times \mathrm{Reg}(L^{(n+p)}) & \to \wp(Q^n \times (Q^p)^*) \\ & (Y,F) & \mapsto \{(\boldsymbol{v}, (\boldsymbol{v},\boldsymbol{p_0})\ldots(\boldsymbol{v},\boldsymbol{p_k})) \mid \boldsymbol{v} \in Y \wedge (\boldsymbol{v},\boldsymbol{p_0})\ldots(\boldsymbol{v},\boldsymbol{p_k})) \in L_F\} \end{array}$$

The main difference of the induced abstract semantics w.r.t. the previous one is that the operators $[\![G]\!]$ and $[\![A]\!]$ now modify the lattice automaton representing the queue content, because one must modify the transition labels each time the state variables are altered.

We obtained the results of Tab. 3 with this refined abstraction, using the widening ∇_k with $k = 0$. In particular, we proved that $0 \leq a \leq s \leq a + 10$, $0 \leq p \leq s-1$ and $0 \leq v \leq s$. The two control states ew[4] and ea are however still reachable, whereas they are not in the real system. An exact analysis should give

[4] Control states are represented by two letters xy, encoding the control states of the sender (r = run,e= error) and of the receiver (w=wait,a=ack).

the invariant $a \leq v \leq s \leq a+10$ with first queue $p = s-1, p = s-2, \ldots, p = v+1$ and second queue $p = v, p = v-1, \ldots, p = a+1$. Those relations are lost when merging transitions of the automata representing the queues. This abstraction is far better than the first one, but should still be improved.

6 Application to interprocedural analysis

We discuss here an interesting application of lattice automata to precise interprocedural analysis, based on the abstraction of call-stacks of imperative programs. This would allow simplifying the stack abstraction proposed in [10] and to extend to infinite state programs the approach of [29] which uses pushdown automata to model finite-state recursive programs and finite automata for representing (co)reachable sets of configurations.

The concrete semantics of imperative programs without global variables makes use of the domains depicted on the following table:

c	$\in C$	: control points
ϵ_i	$\in LEnv_i = LVar_i \to Value$	: local environments for procedure/function P_i
$\langle c, \epsilon \rangle$	$\in Act = C \times LEnv$	: activation record ($LEnv = \bigcup_i LEnv_i$)
Γ	$\in Act^+$	: stacks = program states

Given an abstraction $\wp(LEnv) \leftrightarrows L$ for environments, we can use the abstraction:

$$\wp(Act^+) \xleftarrow{\gamma} \mathrm{Reg}(C \times L, \pi)$$

with the partitioning function $\pi(c) = \{c\} \times \top_L$ based on control points. Lattice automata provide all the necessary operations to abstract the concrete semantics of such imperative programs (*ie.*, push and pop operations, modifications of the top of the stack).

There are some advantages in having an explicit representation of call-stacks in interprocedural analysis. As long as data variables are not abstracted, the more classical functional approach is as precise, but as soon as they are, such an explicit representation allows recovering some loss of information. Moreover, the stack abstraction approach allows a natural description of techniques such as polyvariant analysis, where separate analyses are performed on the same procedure depending on the call context. Last, some applications *require* information on the stack. This is the case for instance of analysis related to stack inspection mechanisms for ensuring security properties in JAVA and .NET architecture [30]. Another example is the test selection technique proposed in [31], in the context of conformance testing of reactive systems w.r.t. an interprocedural specification.

Interprocedural analysis by explicit representation of the call-stacks. The call-string approach of [32], generalised by tokens in [33] corresponds to a very strong abstraction of the call-stacks in which the value of variables is largely ignored. It is more suitable to compilation-oriented dataflow analysis than to program verification. A different line of work consists in modelling recursive programs using pushdown automata and exploiting the property that the set of (co)reachable stacks of pushdown automata is regular [34] and its computation has a polynomial complexity [35–37]. The use of lattice automata can be seen as an extension of this line of work to more expressive, non finite-state programs, where undecidability is overcome by resorting to approximations. The lattice automata abstraction may be seen as both an improvement and a simplification of [10], which derives and unifies two classical techniques for interprocedural analysis (namely the call-string and functional approaches identified by [32]) by abstract interpretation of the operational semantics of imperative programs. In the context of interprocedural shape analysis, [38] represents explicitly the call-stack using the same abstract representation than for the memory configurations.

7 Conclusion

We defined in this paper an abstract domain for languages on infinite alphabets, which are represented by *lattice automata*. Their principle is to use elements of an atomic lattice for labelling the transitions of a finite automaton, and to use a partition of this lattice in order to define a projected finite automaton which acts as a guide for defining determinization, minimization and widening operations. We motivate our choices and show that they lead to a robust notion of approximation, in the sense that normalization is an upper-closure operation and can be seen as a best upper-approximation in the join semilattice of normalized lattice automata.

The resulting abstract domain allows to lift any atomic abstract domain A for $\wp(S)$ to an abstract domain $\mathrm{Reg}(A)$ for $\wp(S^*)$. It is also *parametric*: it is parametrised first by the underlying alphabet lattice, then by a partition of the alphabet lattice, and last by the parameter of the widening operator. Its precision may be improved by adjusting these parameters.

We illustrate the use of lattice automata for the verification of symbolic communicating machines, and we show the need for a non-standard semantics to couple the abstraction of the state variables of the machines with the contents of the FIFO queues. We also explore the applicability of lattice automata to interprocedural analysis and compare this solution to related work. As a result, this work allows to extend both analysis techniques dedicated to communicating machines, and interprocedural analysis based on an explicit representation of the call-stacks.

Future work includes first a deeper study of the application of lattice automata to the analysis of SCM. The challenge we would like to take up is the verification of the SSCOP communication protocol, which is a sliding window protocol from which our running example is extracted. Previous verification attempts that we are aware of are either based on enumerated state-space exploration techniques [39], or on the partial use of theorem proving [40]. It would be interesting to study the application of lattice automata to shape analysis, in the spirit of [20]. A last direction which could be explored is the generalisation of lattice automata recognizing languages on infinite alphabets to tree automata recognizing trees on infinite sets of symbols.

References

1. Le Gall, T., Jeannet, B., Jéron, T.: Verification of communication protocols using abstract interpretation of FIFO queues. In: Algebraic Methodology and Software Technology, AMAST '06. Volume 4019 of LNCS. (2006)
2. Le Gall, T., Jeannet, B.: Analysis of communicating infinite state machines using lattice automata. Technical Report PI 1839, IRISA (2007)
3. Brand, D., Zafiropulo, P.: On communicating finite-state machines. Journal of ACM **30**(2) (1983)
4. Boigelot, B., Godefroid, P., Willems, B., Wolper, P.: The power of QDDs (extended abstract). In: Static Analysis Symposium, SAS'97. (1997)
5. Boigelot, B., Godefroid, P.: Symbolic verification of communication protocols with infinite state spaces using QDDs. Formal Methods in System Design **14**(3) (1997)
6. Bouajjani, A., Habermehl, P.: Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. Theoretical Computer Science **221**(1-2) (1999)
7. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Fast Acceleration of Symbolic Transition systems. In: Computer Aided Verification (CAV'03). Volume 2725 of LNCS. (2003)
8. Finkel, A., Iyer, S.P., Sutre, G.: Well-abstracted transition systems: application to FIFO automata. Information and Computation **181**(1) (2003)
9. Peng, W., Puroshothaman, S.: Data flow analysis of communicating finite state machines. ACM Trans. Program. Lang. Syst. **13**(3) (1991)
10. Jeannet, B., Serwe, W.: Abstracting call-stacks for interprocedural verification of imperative programs. In: Int. Conf. on Algebraic Methodology and Software Technology, AMAST'04. Volume 3116 of LNCS. (2004)
11. Mauborgne, L.: Tree schemata and fair termination. In: Static Analyis Symposium, SAS'00. Volume 1824 of LNCS. (2000)
12. Mauborgne, L.: Representation of sets of trees for abstract interpretation. PhD thesis, École Polytechnique (1999)
13. Neven, F., Schwentick, T., Vianu, V.: Towards regular languages over infinite alphabets. In: Mathematical Foundations of Computer Science, MFCS'01. Volume 2136 of LNCS. (2001)
14. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science **134**(2) (1994)
15. Milo, T., Suciu, D., Vianu, V.: Typechecking for XML transformers. In: Symposium on Principles of Database Systems. (2000)
16. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: Symposium on Logic in Computer Science,LICS '06. (2006)
17. Sagiv, M., Reps, T., Wilhelm, R.: Parametric shape analysis via 3-valued logic. In: Symposium on Principles of Programming Languages (POPL'99). (1999)
18. Yavuz-Kahveci, T., Bultan, T.: Automated verification of concurrent linked lists with counters. In: Static Analysis Symposium, SAS'02. Volume 2477 of LNCS. (2002)
19. Distefano, D., O'Hearn, P., Yang, H.: A local shape analysis based on separation logic. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS'06. Volume 3920 of LNCS. (2006)
20. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract tree regular model checking of complex dynamic data structures. In: Static Analysis Symposium, SAS'06. Volume 4218 of LNCS. (2006)

21. Gopan, D., DiMaio, F., Dor, N., Reps, T., Sagiv, M.: Numeric domains with summarized dimensions. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS'04. Volume 2988 of LNCS. (2004)
22. Feret, J.: Abstract interpretation-based static analysis of mobile ambients. In: Static Analysis Symposium (SAS'01). Volume 2126 of LNCS. (2001)
23. Karr, M.: Affine relationships among variables of a program. Acta Informatica **6** (1976)
24. Jeannet, B.: Dynamic partitioning in linear relation analysis. application to the verification of reactive systems. Formal Methods in System Design **23**(1) (2003)
25. Mauborgne, L., Rival, X.: Trace partitioning in abstract interpretation based static analyzers. In: European Symposium on Programming, ESOP'05. Volume 3444 of LNCS. (2005)
26. Higuchi, M., Shirakawa, O., Seki, H., Fujii, M., Kasami, T.: A verification procedure via invariant for extended communicating finite-state machines. In: Computer Aided Verification,CAV '92. Volume 663 of LNCS. (1993)
27. Lee, D., Ramakrishnan, K.K., Moh, W.M., Shankar, U.: Protocol specification using parameterized communicating extended finite stte machines - a case study of the atm abr rate control scheme. In: Int. Conf. on Network Protocols, ICNP'96. (1996)
28. Jeannet, B., Miné, A.: The APRON Numerical Abstract Domain Library. (http://apron.cri.ensmp.fr/library/)
29. Esparza, J., Schwoon, S.: A BDD-based model checker for recursive programs. In: Computer Aided Verification, CAV'01. Volume 2102 of LNCS. (2001)
30. Besson, F., Jensen, T., Métayer, D.L., Thorn, T.: Model checking security properties of control flow graphs. Journal of Computer Security **9** (2001)
31. Constant, C., Jeannet, B., Jéron, T.: Automatic test generation from interprocedural specifications. Technical Report PI 1835, IRISA (2007) Submitted to TESTCOM/FATES conference.
32. Sharir, M., Pnueli, A.: Semantic foundations of program analysis. In: Program Flow Analysis: Theory and Applications. (1981)
33. Jones, N.D., Muchnick, S.S.: A flexible approach to interprocedural data flow analysis and programs with recursive data structures. In: Symposium on Principles of Programming Languages (POPL'82). (1982)
34. Caucal, D.: On the regular structure of prefix rewriting. Theoretical Computer Science **106** (1992)
35. Esparza, J., Knoop, J.: An automata-theoretic approach to interprocedural dataflow analysis. In: Int. Conf. on Foundations of Software Science and Computation Structure (FoSSaCS '99). Volume 1578 of LNCS. (1999)
36. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model checking. In: Int. Conf. on Concurrency Theory, CONCUR'97. Volume 1243 of LNCS. (1997)
37. Finkel, A., Willems, B., Wolper, P.: A direct symbolic approach to model checking pushdown systems. Electronic Notes on Theoretical Computer Science **9** (1997)
38. Rinetzky, N., Sagiv, M.: Interprocedural shape analysis for recursive programs. In: Compiler Construction, CC'01. Volume 2027 of LNCS. (2001)
39. Bozga, M., Fernandez, J.C., Ghirvu, L., Jard, C., Jéron, T., Kerbrat, A., Morel, P., Mounier, L.: Verification and test generation for the SSCOP protocol. Scientific Computer Programming **36**(1) (2000)
40. Rusu, V.: Combining formal verification and conformance testing for validating reactive systems. Journal of Software Testing, Verification, and Reliability **13**(3) (2003)