INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION INTERNE

N° 1767

ABSTRACT INTERPRETATION OF FIFO CHANNELS

BERTRAND JEANNET, THIERRY JÉRON & TRISTAN LE GALL

ISSN 1166-8687

IRISA

CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

Institut de Recherche en Informatique et Systèmes Aléatoires
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Abstract Interpretation of FIFO channels

Bertrand Jeannet, Thierry Jéron & Tristan Le Gall

Systèmes communicants
Projet VerTeCs

Publication interne n ° 1767 — December 2005 — 25 pages

Abstract: We address the analysis and the verification of communicating systems, which are systems built from sequential processes communicating via unbounded FIFO channels. We adopt the Abstract Interpretation approach to this problem, by defining approximate representations of sets of configuration of FIFO channels. In this paper we restrict our attention to the case where processes are finite-state processes and the alphabet of exchanged messages is finite. We first focus on systems with only one queue, for which we propose an abstract lattice based on regular languages, and we then generalize our proposal to systems with several queues. In particular, we define for these systems two abstract lattices, which are resp. non-relational and relational abstract lattices. We use those lattices for computing an over-approximation of the reachability set of a CFSM. Our experimental evaluation shows that, for some protocols, we obtain results that are as good as those obtained by exact methods founded on acceleration techniques.

Key-words: Communicating Finite State Machines, Reachability Analysis, FIFO channels systems, Abstract Interpretation

(Résumé : tsvp)

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Interprétation Abstraite de files de communication

Résumé : Nous nous intéressons à l'analyse et à la vérification de systèmes communiquants, qui sont des systèmes formés de processus séquentiels communiquant par des files de communication non bornées. Nous proposons de suivre l'approche de l'interprétation abstraite, en définissant des représentations approchées pour les ensembles de configuration de files de communication. Dans le cadre de cet article, nous nous restreignons au cas où les processus sont d'état fini et l'alphabet des messages échangés est également fini. Nous étudions d'abord les systèmes avec une seule file de communication, pour lesquels nous proposons un treillis abstrait fondé sur les langages réguliers, puis généralisons notre proposition aux systèmes avec plusieurs files. En particulier nous définissons pour ces derniers deux treillis abstraits, le premier non-relationel et le second relationel, c'est-à-dire capable de représenter des propriétés liant deux files de communication différentes. Nous utiliserons ces treillis pour calculer une sur-approximation de l'ensemble d'atteignabilité d'un CFSM. Notre évaluation expérimentale montre que nous obtenons, sur certains protocoles, des résultats aussi bons que ceux obtenus par des méthodes exactes fondées sur des techniques d'accélération.

Mots clés : Automates communicants, Analyse d'atteignabilité, Systèmes FIFO, Interprétation Abstraite

1 Introduction

Communicating Finite State Machines(CFSM) [11, 4] is a simple model to specify and verify communication protocols. This model consists of finite-state processes that exchange messages via unbounded FIFO queues. It is also used to define the semantics of standardized protocol specification languages such as SDL and Estelle (e.g., see [22]). Indeed, unbounded queues provide a useful abstraction that simplifies the semantics of specification languages, and frees the protocol designer from implementation details related to buffering policies and limitations. The CFSM model is a simple one, but it can be extended if we want to represent more complicated protocols. Moreover, even this simple model cannot be easily verified: reachability is undecidable for this class of systems [11], since unbounded queues can be used to simulate the tape of a Turing Machine. This fundamental result implies that the verification of even simple properties may be impossible in the general case.

Analysis of systems with unbounded FIFO channels. Some systems with FIFO channels can however be analyzed exactly. *Lossy channels systems* are very similar to CFSM, except that the channels can loose messages at any time. Those systems are easier to verify than perfect channels systems [14]: the reachability problem is decidable, but there is no effective algorithm to compute the reachability set. However, an on-the-fly analysis algorithm based on simple regular expressions is given in [2]. This algorithm can compute the effect of any *meta transition* (loops in the control transition systems), but may not terminate, as the potential number of loops to consider is infinite. The same approach can be applied to classical FIFO systems (*cf.* section 9.1).

Use of the Abstract Interpretation. Instead of restricting the class of considered systems, or designing exact but maybe non-terminating analysis algorithms, an alternative approach consists in resorting to approximations to solve the undecidability problem and to use the Abstract Interpretation theory [12]. In our specific case this consists in replacing in dataflow equations sets of FIFO channel configurations by abstract properties belonging to an abstract lattice. Such transformation results in conservative approximations: we will be able to prove a safety property, or the non-reachability of a state, but not to prove that a property is false or that a state is effectively reachable. The abstract lattices we propose in this paper are all based on regular languages, which exhibit among nice properties the closure under all Boolean operations, and a canonical representation with deterministic and minimized finite automata.

Outline We first introduce section 2 the model of communicating finite-state machines, and the analysis problem we address, namely reachability analysis. We recall the Abstract Interpretation framework in section 3 and explain how we can use it for the reachability analysis of CFSM. Then we define an abstract lattice for the analysis of systems with only one FIFO chanel (section 4). When there are several queues, we may use either a non-relational abstract lattice (section 7) or a relational one (section 6). We implemented our method and we present in section 8 a few case studies on which we experimented it. Eventually we discuss the related work and compare it with ours (section 9).

2 Communicating finite-state machines

2.1 Model

We consider a system of finite-state machines that communicate with each other by sending and receiving messages via n unbounded FIFO queues, modeling communication channels. We adopt a global point of view, by considering the global control structure resulting from the asynchronous product of all the machines.

Definition 1 (CFSM) *A communicating finite-state machine is given by a tuple* (C, Σ, c_0, Δ) *where:*

- *C is a finite set of locations (control states)*
- *$\Sigma = \Sigma_1 \uplus \Sigma_2 \uplus \cdots \uplus \Sigma_n$ is a finite alphabet of messages, where Σ_i denotes the alphabet of messages that can be stored in queue i; we assume that those alphabets are pairwise disjoint;*
- *$c_0 \in C$ is the initial location;*
- *$\Delta \subseteq C \times A \times C$ is a finite set of transitions, where $A = \bigcup_i \{i\} \times \{!, ?\} \times \Sigma_i$ is the set of actions. A transition is given by (c_1, a, c_2) where c_1 and c_2 are in C and a is an action, which can be*
  - *an output (output) $i!m$ with $m \in \Sigma_i$;*
  - *an input (input) $i?m$ with $m \in \Sigma_i$.*

$i!m$ means "the message m is sent through the channel modeled by the queue i" and $i?m$ means "the message m is received from the channel modeled by the queue i".

In the examples of this paper, the finite-state machines(FSMs) and the queues are depicted, but not the result of the global asynchronous product. Indeed, the reader understands easily the presentation of the FSMs whereas the asynchronous product model is needed to perform any global analysis.

Example: The connexion/deconnexion protocol between two machines is the following: the first machine can open a session (and send the message a to the other machine). Once a session is open, the first machine may close it on its own (and send the information message b) or on the demand of the other machine (if it receives the message c). The second one can read the information messages a and b, and ask for a session closure. Fig. 1 models this protocol with two automata, while Fig. 2 models the same protocol using the formalism given in Def. 1.

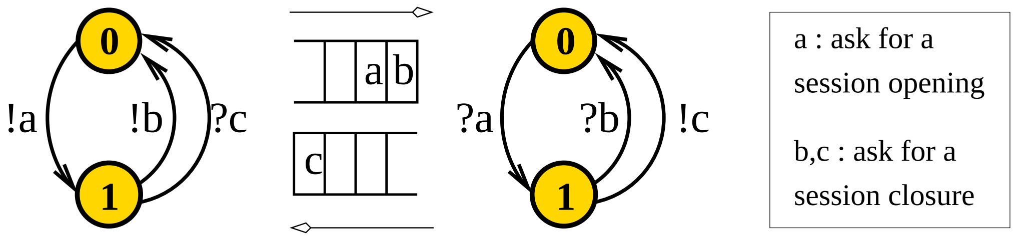

Figure 1: The connexion/deconnexion protocol

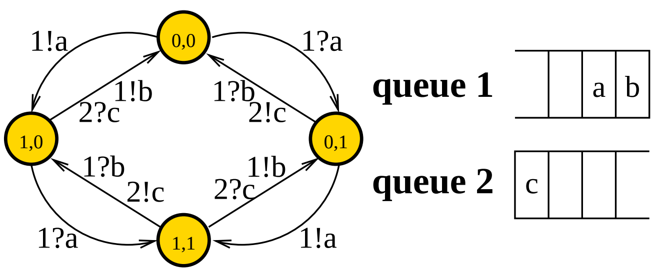

Figure 2: Same protocol with the Def. 1 formalism

2.2 Standard Operational Semantics

The standard operational semantics of a CFSM is given in term of a labelled transition system (LTS).

Definition 2 (LTS) *A labelled transition systems is defined by a tuple $\langle Q, Q_0, \Lambda, \rightarrow\rangle$ where Q is a countable set of states, $Q_0 \subseteq Q$ is the set of initial states, Σ is a finite alphabet and $\rightarrow \subseteq Q \times \Lambda \times Q$ is the transition relation.*

The semantics of a CFSM is an LTS $\langle Q, Q_0, A, \rightarrow\rangle$ where

- $Q = C \times \Sigma_1^* \times \cdots \times \Sigma_n^*$ is the set of states;
- $Q_0 = \{\langle c_0, \varepsilon, \ldots, \varepsilon\rangle\}$ is the set of the initial states;
- A is the alphabet of actions (*cf.* Def. 1).
- $\rightarrow$ is defined by the two rules:

$$\frac{(c_1, i!m, c_2) \in \Delta\ ,\ m \in \Sigma_i}{\langle c_1, w_1, \ldots, w_i, \ldots, w_n\rangle \stackrel{i!m}{\rightarrow} \langle c_2, w_1, \ldots, w_i.m, \ldots, w_n\rangle}$$

$$\frac{(c_1, i?m, c_2) \in \Delta\ ,\ m \in \Sigma_i\ ,\ w_i = m.w_i'}{\langle c_1, w_1, \ldots, w_i, \ldots, w_n\rangle \stackrel{i?m}{\rightarrow} \langle c_2, w_1, \ldots, w_i', \ldots, w_n\rangle}$$

A global state of a CFSM is thus a tuple $\langle c, w_1, \ldots, w_n\rangle \in C \times \Sigma_1^* \times \cdots \times \Sigma_n^*$ where c is the curent location and w_i is a finite word on Σ_i representing the content of queue i. At the beginning, all queues are empty, so the *initial state* is $\langle c_0, \varepsilon, \ldots, \varepsilon\rangle$.

The reflexive transitive closure $\rightarrow^*$ is defined as usual. A state $\langle c, w_1, \ldots, w_n\rangle$ is *reachable* if $\langle c_0, \varepsilon, \ldots, \varepsilon\rangle \rightarrow^* \langle c, w_1, \ldots, w_n\rangle$. The *reachability set* is the set of all states that are reachable. Computing this set is the purpose of the reachability analysis. We can achieve this computation by solving a fix-point equation, as shown in next section.

2.3 Forward collecting semantics and reachability analysis

According to the previous paragraph, the set of global states is $Q = C \times \Sigma_1^* \times \cdots \times \Sigma_n^*$, and the reachability set an element of $\wp(Q)$. We define in this section the forward collecting semantics of a CFSM, which defines the reachability set as the smallest element of the lattice $\wp(Q)$ ordered by inclusion that satisfies a fix-point equation.

Instead of considering unstructured sets of states in $\wp(Q)$, we prefer to associate sets of queue contents to control location. For doing this, we just observe that $\wp(Q) = \wp(C \times \Sigma_1^* \times \cdots \times \Sigma_n^*)$ is isomorphic to $C \rightarrow \wp(\Sigma_1^* \times \cdots \times \Sigma_n^*)$. Thus a set of states $X \in \wp(Q)$ can also be viewed as a map $X : C \rightarrow \wp(\Sigma_1^* \times \cdots \times \Sigma_n^*)$ associating a control state c with a langage $X(c)$ representing all possible contents of all the queues when being in the control state c.

We now define a function $\text{Post} : \wp(Q) \rightarrow \wp(Q)$ that associates to a set of states the set of its immediate successors by the transition relation.
For an output $i!m$ of the CFSM, and a language $L \in \wp(\Sigma_1^* \times \cdots \times \Sigma_n^*)$, we define

$$\text{Send}(L, i, m) = \{\langle w_1, \ldots, w_i.m, \ldots, w_n\rangle | \langle w_1, \ldots, w_i, \ldots, w_n\rangle \in L\}$$

that associates to a set of queues contents the possible queues contents after the output of the message m on the queue i.
For an input $i?m$ of the CFSM, and a language $L \in \wp(\Sigma_1^* \times \cdots \times \Sigma_n^*)$, we define

$$\text{Receive}(L, i, m) = \{\langle w_1, \ldots, w_i, \ldots, w_n\rangle | \langle w_1, \ldots, m.w_i, \ldots, w_n\rangle \in L\}$$

that associates to a set of queues contents the possible queues contents after the input of a message m on the queue i.

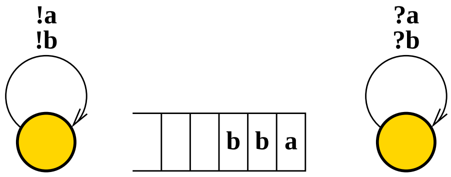

Figure 3: The infinite buffer model

Notice that if L is regular, $\text{Send}^i(L, m)$ and $\text{Receive}^i(L, m)$ are also regular.
We can now define two functions $\overrightarrow{\text{Send}} : \wp(Q) \to \wp(Q)$ and $\overrightarrow{\text{Receive}} : \wp(Q) \to \wp(Q)$:

$$\overrightarrow{\text{Send}}(X) = \lambda c. \bigcup_{(c',i!m,c)\in\Delta} \text{Send}(X(c'), i, m)$$

$$\overrightarrow{\text{Receive}}(X) = \lambda c. \bigcup_{(c',i?m,c)\in\Delta} \text{Receive}(X(c'), i, m)$$

The reader can check that these definitions agree with the operational semantics. For a set of states X, $\overrightarrow{\text{Send}}(X)$ represents the set of states that can be reached from a state of x after one output transition and $\overrightarrow{\text{Receive}}(X)$ represents the set of states that can be reached from a state of X after one input transition. Thus the function $\text{Post} : X \to X$ is defined by $\text{Post}(x) = \overrightarrow{\text{Send}}(x) \cup \overrightarrow{\text{Receive}}(x)$.

The recursive definition of the reachability set RS says that a state is reachable either if it is an initial state or if it is the immediate successor of a reachable state. Let Q_0 be the set of initial states, then $RS \in \wp(Q)$ is the least fix-point of the transfer function $F : \wp(Q) \to \wp(Q)$ defined as $F(X) = Q_0 \cup \text{Post}(X)$.

Example: The CFSM depicted in Fig. 3 models the "infinite buffer" protocol : a *sender process* can store a message in the queue, and a *receiver process* can read messages. In this example, there are only two kinds of messages: a and b.

This CFSM has one control state, called $(0, 0)$, and 4 transitions $(0, 0) \xrightarrow{!a} (0, 0)$, $(0, 0) \xrightarrow{!b} (0, 0)$, $(0, 0) \xrightarrow{?a} (0, 0)$ and $(0, 0) \xrightarrow{?b} (0, 0)$. At the begining, the queue is empty: the initial state is represented by the language $\{\varepsilon\}$. Thus the reachability set is of the form $\{(0, 0)\} \times L_{rs}$, where L_{rs} is the least fix-point of the function F defined, for a language L:

$$\begin{aligned} F(L) = &\quad \{\varepsilon\} \cup \text{Post}(L) \\ \text{i.e.} & \\ F(L) = &\quad \{\varepsilon\} \cup \text{Send}(L, a) \cup \text{Send}(L, b) \cup \text{Receive}(L, a) \cup \text{Receive}(L, b) \end{aligned}$$

However one cannot compute this *reachability set* in general [11]. Although there is no general algorithm that can compute the reachability set, there may be however some semi-algorithms that compute the reachability set in some cases, that are described in section 9.

3 Abstract Interpretation

3.1 Principles

Abstract Interpretation is a theory for computing approximate solutions of fix-point equations, in the context of program analysis. Most analysis problems, among others reachability analysis, come down to solving a fix-point equation of the form $x = F(x), x \in \wp(S)$, where S is the state space of the system, $\wp(S)$ is the complete lattice of sets of states, ordered by inclusion, and F is roughly the "successor set of states" function. The fundamental principles of Abstract Interpretation are [12]:

1. to substitute to the concrete domain $\wp(S)$, a simpler abstract domain A (static approximation). The concrete lattice $(\wp(S), \subseteq)$ and abstract lattice $(A, \sqsubseteq)$ are linked by a Galois connection $\wp(S) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ which ensures the soundness of the method.

2. to transpose the fix-point equation into the abstract domain, so that one has to solve an equation

$$y = F^\sharp(y), y \in A, \text{ with } F^\sharp \sqsupseteq \alpha \circ F \circ \gamma$$

3. to use a widening operator (dynamic approximation) to make the iterative computation of the least fix-point of $F^\sharp$ converge after a finite number of steps to some upper-approximation $\bar{y} \in A$ of the least fixpoint of $F^\sharp$ (more precisely, a post-fix-point of $F^\sharp$).

If the conditions above are satisfied, $\bar{y} \in A$ satisfies $\gamma(\bar{y}) \supseteq \mathrm{lfp}(F)$, which means that $\bar{y}$ is an upper-approximation of the least fixpoint of F.

3.2 Representations framework

Abstract Interpretation as described in the previous section assumes that the abstract lattice is complete. We are however interested in the lattice of regular languages $(\mathcal{R}(\Sigma), \subseteq)$ as an abstract domain, which is not a complete lattice (like the convex polyhedra lattice). In particular, one cannot define a correct abstraction function. For instance, the (most precise) abstraction of the language $\{a^n b^n \mid n \geq 0\}$ in $\mathcal{R}(\{a, b\})$ does not exist. One thus need to consider the slightly weaker *representations framework* of [10] instead of a classical Galois connection to formalize our abstraction. We recall the main result and invite the reader to see [10] for more details.

Let $(A, \bot, \sqsubseteq)$ be a partially ordered set with a smallest element $\bot$, and $\gamma : A \to \wp(S)$ a concretization function (*ie.*, γ is monotone and $\gamma(\bot) = \bot$).

Let $\nabla : A \times A \to A$ be an operator verifying:

1. $\forall a, a' \in A,\ a \sqsubseteq a \nabla a'$ and $a' \sqsubseteq a \nabla a'$ (∇ is an upper bound operator)

2. $\forall (a_i)_{i \in \mathbb{N}} \in A^{\mathbb{N}}$ such that $\forall i \geq 0, a_i \sqsubseteq a_{i+1}$, the sequence $(a_i)_{i \in \mathbb{N}}$ defined as $a'_0 = a_0$ and $a'_{i+1} = a'_i \nabla a_{i+1}$ has a greatest element a_∞ (thus obtained after a finite number of steps).

Here ∇ replaces both the least upper-bound and the widening operators. The main preservation theorem is the following one:

Theorem 1 *Let $F : \wp(S) \to \wp(S)$ be a monotonous function and $F^\sharp : A \to A$ a monotonous function such that $\gamma \circ F^\sharp \supseteq F \circ \gamma$. The sequence $(a_i)_{i \in \mathbb{N}}$ defined by:*

$$\begin{cases} a_0 &= \bot \\ a_{i+1} &= a_i & \textit{if } F^\sharp(a_i) \sqsubseteq a_i \\ a_{i+1} &= a_i \nabla F^\sharp(a_i) & \textit{otherwise} \end{cases}$$

has a greatest element a_∞, which is a post fix-point of $F^\sharp$. Morevoer, $\gamma(a_\infty)$ is a post fix-point of F.

Proof: We first prove that the sequence $(a_i)_{i \in \mathbb{N}}$ has a greatest element. If for some i, $F^\sharp(a_i) \sqsubseteq a_i$, then a_i is the limit of the sequence and a post fix-point of $F^\sharp$.

Assume by contradiction that $(a_i)_{i \in \mathbb{N}}$ has no greatest element. Then we have $\forall i : a_{i+1} = a_i \nabla F^\sharp(a_i) \sqsupseteq a_i$, and $(a_i)_{i \in \mathbb{N}}$ is an increasing sequence. Moreover, $a_0 = \bot \sqsubseteq F^\sharp(a_0)$ and the monotonicity of $F^\sharp$ implies that:

$$\begin{aligned} a'_0 &= a_0 \\ a'_{i+1} &= F^\sharp(a_i) \end{aligned}$$

is an increasing chain. The definition of the widening operator ensures that :

$$\begin{array}{rcl} a''_0 & = & a_0 \\ a''_{i+1} & = & a''_i \nabla a'_{i+1} \end{array}$$

has a greatest element a_∞. Then we show by induction on i that $\forall i \geq 0,\ a''_i = a_i$: this is obvious for $i = 0$ and

$$\begin{array}{rcll} a''_{i+1} & = & a''_i \nabla a'_{i+1} & \\ & = & a_i \nabla a'_{i+1} & \text{(induction hypothesis)} \\ & = & a_i \nabla F^\sharp(a_i) & \text{(definition of } a'_{i+1}) \\ & = & a_{i+1} & \end{array}$$

Consequently, the sequence $(a_i)_{i\in\mathbb{N}}$ has a greatest element. This is a contradiction of the first assumption.

We now check that $\gamma(a_\infty)$ is a post fix-point of F.

$$\begin{array}{rcll} F(\gamma(a_\infty)) & \subseteq & \gamma(F^\sharp(a_\infty)) & \text{(assumption on } F^\sharp) \\ & \subseteq & \gamma(a_\infty) & \text{by } F^\sharp(a_\infty) \sqsubseteq a_\infty \text{ and monotonicity of } \gamma \end{array}$$

□

This theorem offers a way to compute an overapproximation of the least fix-point of a function F under rather weak assumption on the abstract domain A, which may not be a lattice. In the sequel of the paper however, the abstract domains that we will define will have a lattice structure. The main point forbidding the use of the classical presentation of Abstract Interpretation is the incompleteness of the abstract lattice A and the related impossibility of defining an abstraction function α.

4 Systems with only one queue

If we consider a CSFM with only one queue, the concrete state-space has the structure $C \to \wp(\Sigma^*)$, and it will be abstracted by the set $C \to A$, where A is an abstract lattice for $\wp(\Sigma^*)$. In this section we will consider for A the set of regular languages.

4.1 The abstract lattice of regular languages

Let Σ be a finite alphabet and $\mathcal{R}(\Sigma)$ be the set of regular languages on Σ. $(\mathcal{R}(\Sigma), \bot, \top, \subseteq)$ is a lattice with $\bot = \emptyset$ and $\top = \Sigma^*$. It can be connected to the concrete lattice $\wp(\Sigma^*)$ by the identity concretization function $\gamma = id$.

Lemma 1 $(\mathcal{R}(\Sigma), \subseteq)$ *is of infinite height, and is not complete.*

Proof: Consider the familly of regular languages $(L_n)_{n\in\mathbb{N}}$:

$$\forall n \geq 0,\ L_n \ = \ \bigcup_{k\leq n} \{a^k.b^k\}$$

Since $\forall n \geq 0,\ L_n \subseteq L_{n+1}$, this family is an infinite increasing chain. Thus $(\mathcal{R}(\Sigma), \subseteq)$ has an infinite height. Moreover,

$$\bigcup_{n\in\mathbb{N}} L_n = \{a^i.b^i \mid i \in \mathbb{N}\}$$

is not a regular language; so $(\mathcal{R}(\Sigma), \subseteq)$ is not complete. □

From a computational point of view, a regular language L will be represented and manipulated using the (unique up to isomorphism) minimal deterministic automaton recognizing them. We call it the canonical automaton of L.

The operations we need for analyzing our system are the union $\cup$, the intersection $\cap$, the inclusion test $\subseteq$, and the send and receive operations, $\overrightarrow{\text{Send}}$ and $\overrightarrow{\text{Receive}}$. These operations, defined on $\wp(\Sigma^*)$, can be exactly restricted on $\mathcal{R}(\Sigma)$, as regular languages are stable by these operations. Their implementation using finite automata is classical and can be found in textbooks.

We mainly have to define a suitable widening operator ∇. Its choice is important for the analysis, as all approximations performed by the analysis will result of its application. The widening operator must be adapted to the FIFO structure, thus the abstraction should remain precise for both the begining and the end of the queue. In [16], a widening operator for regular languages was mentioned. We will adapt this operator to regular languages regognizing the content of a FIFO channel.

4.2 Widening operator

The widening operator we suggest will be derived from the notion of *auto-bisimulation*. In this section we define a family operators on finite automata $(\rho_k)_{k\geq 0}$ from which we be derived a family of widening operators $(\nabla_k)_{k\geq 0}$ working on regular languages. The operator ρ_k is defined by quotienting the canonical automaton of its argument by an equivalence relation $\approx_k$ on its states.

We first remind basic definitions.

Definition 3 (Finite automaton) *A finite automaton $\mathcal{M} = (Q, \Sigma, Q_0, Q_f, \rightarrow)$ is a finite labelled transition system ($|Q| < +\infty$) with a set of final states $Q_f \subseteq Q$. It is deterministic if $Q_0 = \{q_0\}$ and if $\forall q, q_1, q_2 \in Q, \forall a \in \Sigma,\ (q, a, q_1) \in\rightarrow \wedge (q, a, q_2) \in\rightarrow \Rightarrow q_1 = q_2$. In this case, we define the* transition function *$\delta\ : Q \times \Sigma^* \rightarrow Q$:*

$$\forall q \in Q,\ \delta(q, \varepsilon) = q$$

and

$$\frac{w' = w.a \quad \delta(q, w) = q'' \quad (q'', a, q') \in\rightarrow}{\delta(q, w') = q'}$$

A word $w \in \Sigma^*$ is accepted by *$\mathcal{M}$ if $\delta(q_0, w) \in Q_f$. The language recognized by a non-deterministic automaton is defined in a similar way; we define a transition function $\delta : \wp(Q) \times \sigma^* \rightarrow \wp(Q)$ and a word w is recognized by the automaton if and only if $\delta(Q_0, w) \cap Q_f \neq \emptyset$.*

If we have a finite automaton and an equivalence relation between states, we can define a quotient automaton as follows.

Definition 4 (Quotient automaton) *Let $\mathcal{M} = (Q, \Sigma, Q_0, Q_f, \rightarrow)$ be a finite automaton and $\approx \subseteq Q \times Q$ an equivence relation bewteen states. The equivalence class of a state $q \in Q$ is denoted by $\tilde{q} \in Q/\approx$. The* quotient automaton *$(\tilde{Q}, \Sigma, \tilde{Q_0}, \tilde{Q_f}, \tilde{\rightarrow})$ is defined by:*

- $\tilde{Q} \triangleq Q/\approx$
- $\tilde{Q_0} \triangleq \{\overline{q} \in \tilde{Q} \,|\, \overline{q} \cap Q_0 \neq \emptyset\}$
- $\tilde{Q_f} \triangleq \{\overline{q} \in \tilde{Q} \,|\, \overline{q} \cap Q_f \neq \emptyset\}$
- $(\overline{q_1}, a, \overline{q_2}) \in \tilde{\rightarrow}$ *if and only if* $\exists q_1 \in \overline{q_1},\ \exists q_2 \in \overline{q_2}\ :\ (q_1, a, q_2) \in\rightarrow$

In term of recognized language, the quotient automaton is "an overapproximation" of the initial automaton:

Lemma 2 *Let L be a regular langage recognized by a finite automaton $\mathcal{M} = (Q, \Sigma, Q_0, Q_f, \Delta)$, $\approx$ an equivalence relation on Q, and $\tilde{L}$ be the language recognized by the quotient automaton $\tilde{\mathcal{M}} = (\tilde{Q}, \Sigma, \tilde{Q_0}, \tilde{Q_f}, \tilde{\rightarrow})$. Then $L \subseteq \tilde{L}$.*

Proof: Let $w = a_1 \dots a_n$ be a word recognized by $\mathcal{M}$. This implies that there is a sequence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$ with $q_0 \in Q_0$ and $q_n \in Q_f$. By definition of the quotient automaton, there is a sequence $\tilde{q_0} \xrightarrow{a_1} \tilde{q_1} \xrightarrow{a_2} \dots \xrightarrow{a_n} \tilde{q_n}$ in $\tilde{\mathcal{M}}$ with $\tilde{q_0} \in \tilde{Q_0}$ and $\tilde{q_f} \in \tilde{Q_f}$ so w is a word recognized by $\tilde{\mathcal{M}}$. □

The operator ρ_k. We now define a operator ρ_k on regular languages by quotienting their canonical automaton by an equivalence relation on states $\approx_k$. We use as an equivalence relation the smallest bisimulation relation of order k that is finer than a basic equivalence relation.

Definition 5 (Bisimulation of order k) *Let $(Q, \to, Q_0, Q_f, \to)$ be a finite automaton, and a coloring function* col $: Q \to [1..N]$ *defining an equivalence relation* $q_1 \approx_{\mathrm{col}} q_2 \Leftrightarrow \mathrm{col}(q_1) = \mathrm{col}(q_2)$. *For $k \geq 0$, the smallest bisimulation of order k finer than $\approx_{\mathrm{col}}$ is defined inductively by:*

$$q_1 \approx_0 q_2 \quad \textit{iff} \quad q_1 \approx_c ol q_2 \tag{1}$$

$$q_1 \approx_{k+1} q_2 \quad \textit{iff} \quad \begin{cases} q_1 \approx_k q_2 \\ \forall a \in \Sigma, \forall q_1' \in Q,\ (q_1, a, q_1') \in \to \Longrightarrow \exists q_2' \in Q \ :\ (q_2, a, q_2') \in \to \wedge\, q_1' \approx_k q_2' \\ \forall a \in \Sigma, \forall q_2' \in Q,\ (q_2, a, q_2') \in \to \Longrightarrow \exists q_1' \in Q \ :\ (q_1, a, q_1') \in \to \wedge\, q_1' \approx_k q_2' \end{cases} \tag{2}$$

The classical bisimulation relation in the litterature is just the limit of the sequence $\bigcap_{k \geq 0} \approx_k$ in the lattice $\wp(Q \times Q)$ ordered by inclusion. One may also characterize $\approx_k$ in the following way.

Proposition 1 *Given a deterministic finite automaton $(Q, \to, Q_0, Q_f, \to)$, a coloring function* col : $Q \to [1..N]$*, and $\approx_k$ the bisimulation of order k built on* col*, we have*

$$q_1 \approx_k q_2 \ \Leftrightarrow\ \forall w \in \Sigma^*, |w| \leq k \Rightarrow \mathrm{col}(\delta(q, w)) = col(\delta(q', w))$$

with the convention that $\delta(q, w)$ is equal to $\perp$ when not defined.

Proof: The equivalence is obvious for $k = 0$. Assume that the property is true up to a given k.

- Let $q_1, q_2 \in Q$ be two states such that $\forall w \in \Sigma^*, |w| \leq k + 1 \Rightarrow \mathrm{col}(\delta(q_1, w)) = col(\delta(q_2, w))$. We deduce that $\forall a \in \Sigma, \exists q_1' \in Q, \exists q_2' \in Q \ :\ q_1' = \delta(q_1, a) \wedge q_2' = \delta(q_2, a) \wedge \forall w \in \Sigma^*, |w| \leq k \Rightarrow \mathrm{col}(\delta(q_1', w)) = col(\delta(q_2', w))$. The states q_1' and q_2' as chosen in the formula satisfies $q_1' \approx_k q_2'$ by the induction hypothesis, and it is easy to see that the formula implies the equation (2).
- Let $q_1, q_2 \in Q$ 2 states such that $q_1 \approx_{k+1} q_2$ and $w \in \Sigma^*$ a word of length $k + 1$. So $w = a.w'$ with $|w'| = k$. We define $q_1' = \delta(q_1, a)$ and $q_2' = \delta(q_2, a)$. We have $q_1' \approx_k q_2'$, and by the induction hypothesis $\mathrm{col}(\delta(q_1', w')) = \mathrm{col}(\delta(q_2', w'))$. So $\mathrm{col}(\delta(q_1, w)) = \mathrm{col}(\delta(q_1', w')) = \mathrm{col}(\delta(q_2', w')) = \mathrm{col}(\delta(q_2, w))$.

□

We now define the operator ρ_k. It uses the bisimulation $\approx_k$ on states built on a coloring function that basically separates initial states, final states, and other states.

Definition 6 (Unary operator ρ_k) *Let $L \in \mathcal{R}(\Sigma)$ a regular language, and $\mathcal{M} = (Q, \Sigma, Q_0, Q_f, \to)$ the minimal* [1] *deterministic automaton recognizing it. We define a coloring function* col : $Q \to \{0, 1, 2, 3\}$ *as:*

$$\mathrm{col}(q) = \begin{cases} 0 & \textit{if} \quad q \in Q_0 \cap Q_f \\ 1 & \textit{if} \quad q \in Q_f \setminus Q_0 \\ 2 & \textit{if} \quad q \in Q_0 \setminus Q_f \\ 3 & \textit{otherwise} \end{cases} \tag{3}$$

$\rho_k(L)$ is the language recognized by the quotient automaton $\tilde{\mathcal{M}}$ defined by the equivalence relation $\approx_k$ built on col.

[1] A deterministic automaton is minimal if any other deterministic automaton recognizing the same language has more states. For a regular languague L, there is a (unique to isomorphism) minimal automaton recognizing L.

Remark 1 *ρ_k is extensive ($\rho_k(L) \supseteq L$ and idempotent $\rho_k \circ \rho_k = \rho_k$, but it is not monotomnous. Thus it does not fit in the mathematical definition of a upper closure operator.*

Definition of the widening operator. Our widening operator consists in applying this unary operator ρ_k to the union of its arguments.

Definition 7 (Widening operator ∇) *Let k be a fixed integer and L a regular language recognized by the minimal deterministic automaton $\mathcal{M}$. We define a binary operator $\nabla : \mathcal{R}(\Sigma) \times \mathcal{R}(\Sigma) \to \mathcal{R}(\Sigma)$:*

$$L_1 \nabla L_2 \triangleq \rho_k(L_1 \cup L_2)$$

Theorem 2 *∇ is a widening operator for the abstract lattice of regular languages $\mathcal{R}(\Sigma)$, i.e.*

1. *$\forall L_1, L_2,\ L_1 \subseteq L_1 \nabla L_2 \wedge L_2 \subseteq L_1 \nabla L_2$.*
2. *For any increasing chain ($L_0 \subseteq L_1 \subseteq \dots$), the increasing chain defined by $L'_0 = L_0,\ L'_{i+1} = L'_i \nabla L_{i+1}$ is not strictly increasing (it stabilizes after a finite number of steps).*

Proof:

1. According to Lemma 2, we have $L_1 \cup L_2 \subseteq L_1 \nabla L_2$
2. We prove that $|Q/\approx_k|$ is less or equal to a constant depending on k and $|\Sigma|$. Thus the set $\{\rho_k L \,|\, L \in \mathcal{R}(\Sigma)\}$ is finite.

   For a state q, we consider the "execution tree of depth k":

   (a) each node is labelled with a triple (a, x, i), with $a \in \Sigma \cup \{-\}$, $x \in Q$ and $i \geq 0$

   (b) the root is labelled with $(-, q, k)$

   (c) if there is a node labelled with (x, i), with $i > 0$ and a transition $(x, a, y) \in \to$, we create a son labelled by $(a, y, i-1)$

   One can color a node labelled by (a, x, i) with $\mathrm{col}(x)$. According to the first definition of $\approx_k$, two states q_1 and q_2 are in the same equivalence class if they have the same colored execution tree. So there are as many equivalence classes as colored execution tree.

   Since the maximum branching degree of this tree is $|\Sigma|$, there are at most $2^{|\Sigma|^k}$ uncolored trees (proof by induction on k), each tree has at most $|\Sigma|^{k+1}$ nodes, so each tree can be colored in at most $4^{|\Sigma|^{k+1}}$ ways. Thus, we have $|Q/\approx_k| \leq 4^{|\Sigma|^{k+1}} \times 2^{|\Sigma|^k}$.

□

Remark 2 *When the notion of widening operator was introduced in the 70ies, the idea was to guess the limit of a post-fix-point computation. However, our "widening operator" ∇ doesn't try to guess the limit of a sequence of languages. Even if ∇ verifies the mathematical properties of a widening operator, one may argue that it must be considered as an upper bound operator rather than a genuine widening operator.*

4.3 Effects of the widening operator

With the formal definition given above, the reader may have some difficulties to fancy the effects of the widening operator. Let L be a regular language describing a queue content, k a given integer. We want to give an intution of the shape of the language $\rho_k(L)$. We illustrate by some examples the approximations we make.

Let us consider $\mathcal{M}_L$ the minimal deterministic automaton that recognizes the language L. For each example, we depict:

1. $\mathcal{M}_L$
2. the partition of states gererated by $\approx_k$
3. the quotient automaton $\tilde{\mathcal{M}_L}$ recognizing $\rho_k(L)$

Number of messages The multiple repetition of the same message $aa \ldots a$ may be overapproximated by an infinite repetition of the same message a^*. If the systems allows arbitrary-long channel contents, this approximation can guess the limit of the fix-point computation.

Example: The language $L = \{aaaa\}$ is recognized by the automaton $\mathcal{M}_L$ depicted in Fig. 4. The two states marked with an X (Fig. 5) are in the same equivalence class (with $k = 1$), so they are merged. Here, the effect of the widening operator is to create a loop: $\nabla_1 L = \{a^i | i \geq 3\}$ (Fig. 6). In this example, the information we loose is the number of messages 'a'.

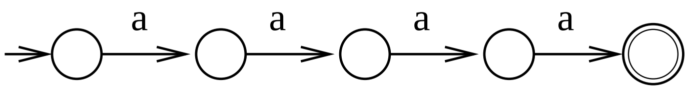

Figure 4: Automaton recognizing $L = \{aaaa\}$

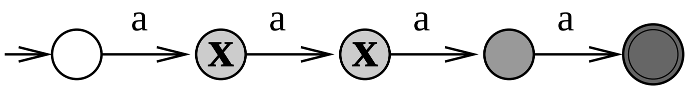

Figure 5: Equivalence classes of $\approx_1$

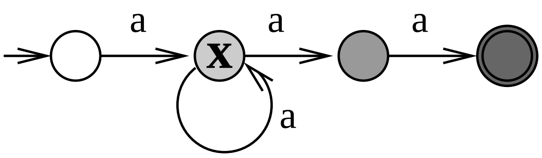

Figure 6: Automaton recognizing $\rho_1(L) = \{aaaa^*\}$

Sum of languages If $L = L_1 + L_2$, the widening operator may merge some word of L_1 with words of L_2. This approximation may simplify the automaton.

Example: The language $L = \{aac + bad\}$ is recognized by the automaton $\mathcal{M}_L$ depicted in Fig. 7. The two states marked with an X (Fig. 8) are in the same equivalence class (with $k = 1$), so they are merged. The result is the language $\nabla_1 L = \{(a + b)a(c + d)\}$ (Fig. 9. In this example, we loose the property "if we have an 'a' at the begining of the queue, then we have a 'c' at the end, else a 'd' ".

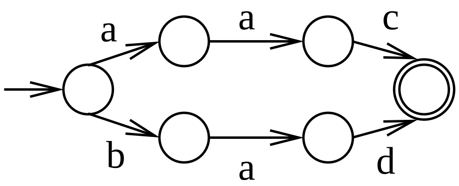

Figure 7: Automaton recognizing $L = \{aac + bad\}$

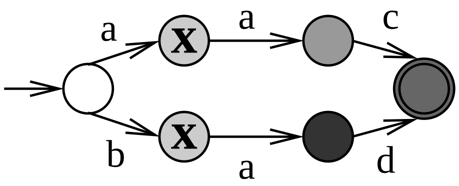

Figure 8: Equivalence classes of $\approx_1$

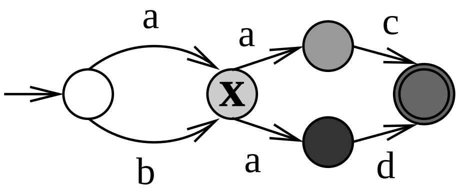

Figure 9: Automaton recognizing $\rho_1(L) = \{(a+b)a(c+d)\}$

Suffixes and prefixes Due to the FIFO structure of the communication channels, our approximation must be more precise at the beginning and the end of the words representing the queue-contents.

Theorem 3 *Let L be a regular language and k a fixed integer. Then L and $\rho_k(L)$ have the same set of prefixes of length 1 and the same set of suffixes of length less or equal to k.*

Proof: Let $\mathcal{M} = (Q, Q_0, Q_f, \rightarrow)$ be the minimal deterministic automaton recognizing L and $(\tilde{Q}, \Sigma, \tilde{Q_0}, \tilde{Q}_f, \tilde{\rightarrow})$ the quotient automaton. Since $L \subseteq \rho_k(L)$, the prefixes (and suffixes) of L are also prefixes (and suffixes) of $\rho_k L$. We must prove the converse.

If $a \in \Sigma$ is a prefix of length 1 of a word of $\rho_k(L)$, then there is $\overline{q_0} \in \tilde{Q_0}$ $\overline{q} \in \tilde{Q}$ such that $(\overline{q_0}, a, \overline{q}) \in \tilde{\rightarrow}$. The definition of the color function ensures that q_0 is also an initial state of $\mathcal{M}$, so a is a prefix of length 1 of L.

To end the proof, we prove by induction that, if there is a sequence $\overline{q_1} \xrightarrow{a_1} \overline{q_2} \xrightarrow{a_2} \ldots \xrightarrow{a_l} \overline{q_f}$ allowed by the quotient automaton, with $\overline{q_f} \in \tilde{Q}_f$ and $l \leq k$, then there are $l+1$ states $q_1 \in \overline{q_1}, q_2 \in \overline{q_2}, \ldots, q_f \overline{q_f}$ such that: $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \ldots \xrightarrow{a_k} q_f$ is allowed by $\mathcal{M}$ (obvious for $k = 0$).

Let $\overline{q_1} \xrightarrow{a_1} \overline{q_2} \xrightarrow{a_2} \ldots \xrightarrow{a_l} \overline{q_f}$ be a sequence verifying the conditions. Then there are l states $q_2 \in \overline{q_2}, \ldots, q_f \in \overline{q_f}$ such that: $q_2 \xrightarrow{a_2} \ldots \xrightarrow{a_l} q_f$ is allowed by $\mathcal{M}$ (induction hypothesis).

Moreover, $\exists q'_1 \in \overline{q_1}, q'_2 \in \overline{q_2},\ q'_1 \xrightarrow{a_1} q'_2$ (defnition of the quotient automaton). So we have two states $q_2 \approx_k q'_2$ According to the first definition of $\approx_k$, since $|a_2 \ldots a_l| \leq k$, $\exists q'_3 \in \overline{q_3} \ldots q'_f \in \overline{q_f}$ such that $q'_2 \xrightarrow{a_2} \ldots \xrightarrow{a_l} q'_f$ is allowed by $\mathcal{M}$. So $q'_1 \xrightarrow{a_1} q'_2 \xrightarrow{a_2} \ldots \xrightarrow{a_l} q'_f$.

□

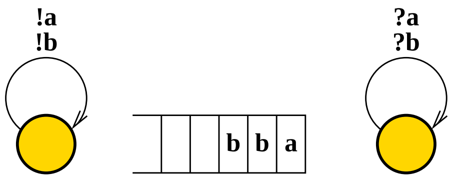

Figure 10: The infinite buffer model

4.4 Example of reachability analysis

Let us come back to the "infinite buffer" example (Example 2.3). This protocol is modeled by CFSM depicted in Fig. 10. We recall that the reachability analysis issue is to find the least fix-point of a function F:

$$\begin{aligned} F(L) = & \quad \{\varepsilon\} \cup \mathrm{Send}(L,a) \cup \mathrm{Send}(L,b) \cup \mathrm{Receive}(L,a) \cup \mathrm{Receive}(L,b) \\ F(L) = & \quad \{\varepsilon\} \cup \mathrm{Post}(L) \end{aligned}$$

According to theorem 1, we can compute an over-approximation of this least fix-point by finding out the limit of $(x_n)_{n\in\mathbb{N}}$:

$$\begin{array}{rcll} x_0 & = & \{\varepsilon\} & \\ x_{n+1} & = & x_n & \text{if } \mathrm{Post}(x_n) \subseteq x_n \\ x_{n+1} & = & x_n \nabla \mathrm{Post}(x_n) & \text{otherwise} \end{array}$$

Thanks to the properties of the widening, this computation always terminates and leads to a post-fixed-point. We choose $k = 1$ for the widening definition (*cf.* section 4.2), and the results of the different steps of computations are :

L_0	ε
L_1	$\varepsilon + (a + b)$
L_2	$\varepsilon + (a + b) + (a + b)(a + b)$

$\mathrm{Post}(L_2) = \varepsilon + (a + b) + (a + b)(a + b) + (a + b)(a + b)$ is recognized by the automataton depicted in Fig. 11; the states 1 and 2 are in the same equivalence class, so $L_3 = L_2 \nabla \mathrm{Post}(L_2)$ is recognized by the non-deteministic automaton depicted in Fig. 12. We have $L_3 = (a + b)^*$ and $\mathrm{Post}(L_3) \subseteq L_3$, so L_3 is the limit of the sequence (in fact, it is the least fix-point of the equation).

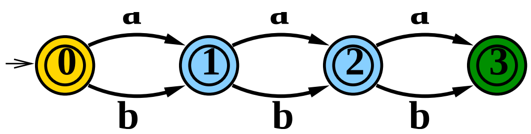

Figure 11: $\mathrm{Post}(L_2) = \varepsilon + (a + b) + (a + b)(a + b) + (a + b)(a + b)$

Here we have everything we need to perform a reachability analysis of system with only one queue. However, some specific issues must be taken into account when we deal with CFSM systems with several queues.

5 Systems with several queues: representation issues

Representations In section 4, we dealt with systems with only one queue. In this case, there is no representation issue; a content of a queue is represented by a finite word, and a set of contents

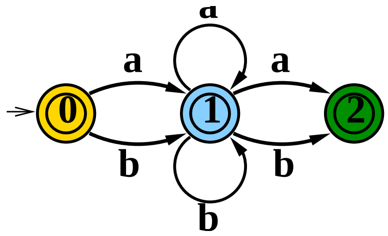

Figure 12: $L_3 = L_2 \nabla \mathrm{Post}(L_2)$

by a language. When there are several queues, we must choose a way to represent the diiferent queue-contents by a single representation.

We may choose to represent n words $w_1, \ldots, w_n$ by:

1. a vector of words $\langle w_1, \ldots, w_n \rangle$
2. a concatenation of words $w_1 \ldots w_n$ (we recall that the alphabets are pairwise disjoint)
3. an "interlaced" word formed by the sequence first letter of w_1, first letter of w_2 ..., first letter of w_n, second letter of w_1, second letter of w_2, etc. Il the n words have different length, we use a special letter $-$. For example, il we have three words $w_1 = aba$, $w_2 = c$ and $w_3 = deef$, we will represent them by the word $w = acdb - ea - e - -f$

The third representation may be used for the representation of sets of integer vectors (e.g. within the NDD framework [23]), but does not fit with the FIFO operations : the operations Send(L, m) and Receive(L, m) modifiy the size of only one queue, so the computation of the effects on "interlaced" words may be costly. We discard it and we will use the two first representations in order to define two kinds of analysis.

Relational and non-relational lattices At present time, we defined an abstract lattice for the representation a single queue-content. When there are n queues, we must choose whereas we analyse each queue independently or we analyse all the queues together. In the first case, we may re-use n times the previous lattice. In the second case, we must find a proper abstract lattice. This is a general issue when using Abstract Interpretation techniques. We only need a *non-relational lattice* (i.e. a lattice without any relation between the different queue-contents) in the first case, wheras a *relational lattice*(i.e. a lattice that keep some relation between the different queue-contents) is needed in the second case. For example, for the analysis of a program with several integer variables, one can use a non-relational lattice is based on intervals [12], or a relational one based on polyedra [13].

The non-relational analysis is easier, but less precise. In our case, we use n "small" automata to represent the queue contents independantly. For the relational analysis, we use one "big" automaton to represent all the queue contents. Thus the analysis will preserve some relationships between the different queue-contents. The reason why the non-relational analysis will be faster is that the operations we use are not linear (they may be exponential) so it is faster to do n times those operations on small automata rather than one time on a big automaton.

6 A non-relational lattice

Considering a system with n queues, we can represent the contents of all queues by a vector of finite words $\langle w_1, \ldots, w_n \rangle$. Generally speaking, a set of vectors $X \subseteq E^n$ can be approximated by a single vector $\langle X_1, \ldots, X_n \rangle \in \wp(E)^n$, where X_i is the projection of X on the dimension i. Moreover, a language $L \in \mathcal{L}(\Sigma)$ is approximated by a vector of languages $\alpha(L) = \langle L_1, \ldots L_n \rangle$ with $L_i = L_{/\Sigma_i}$.

Example: Considering a set of 3 vectors of words $v_1 = \langle aa, cd\rangle$, $v_2 = \langle ab, cd\rangle$ and $v_2 = \langle aa, cc\rangle$, we over-approximate it by the vector of languages $\langle L_1, L_2\rangle$ with $L_1 = \{aa + ab\}$ and $L_2 = \{cd + cc\}$.

Thus we first the abstract the lattice $\mathcal{L}(\Sigma)$ by $\mathcal{L}(\Sigma_1) \times \cdots \times \mathcal{L}(\Sigma_n)$, loosing all relations between the contents of the differents queues. Then we can reuse abstract lattice defined in section 4 and we obtain the following connexion:

$$\wp(\Sigma_1^* \times \ldots \times \Sigma_n^*) \underset{\alpha_c}{\overset{\gamma_c}{\leftrightarrows}} \mathcal{L}(\Sigma_1) \times \cdots \times \mathcal{L}(\Sigma_n) \xleftarrow{\gamma} \mathcal{R}(\Sigma_1) \times \cdots \times \mathcal{R}(\Sigma_n)$$

The operations in this lattice are the same as the operations defined in section 4. For the widening operator, given a fixed integer k, we may extend the former definition to:

$$\langle L_1, \ldots, L_n\rangle \overrightarrow{\nabla} \langle L'_1, \ldots, L'_n\rangle \triangleq \langle L_1 \nabla L'_1, \ldots, L_n \nabla L'_n\rangle$$

If needed, we can also define $\overrightarrow{\nabla}$ with different values of k (one value for each single queue content).

Remark 3 *In this lattice, the upper bound ("the union") is no longer exact, beacuse of the cartesian product. For example, the upper bound of the languages $\langle a, c\rangle$ and $\langle b, d\rangle$ is the language $\langle a + b, c + d\rangle$*

7 A relational lattice

7.1 The QDD representation

We can also represent the contents of all queues by a concatenation of words, assumning that the queues alphabets are pairwise disjoint.

Example: Considering three queue contents represented by the words $abaa \in \Sigma_1^*$, $cdec \in \Sigma_2^*$ and $fgg \in \Sigma_3^*$, we represent all the contents by a single word $abaacdecfgg \in \Sigma^*$.

The notion of QDD was first introduced by B. Boigelot and P. Godefroid [5]. A QDD is a finite automaton recognizing words that can be decomposed in the following way : $w = w_1 w_2 \ldots w_n$ with $\forall i, w_i \in \Sigma_i^*$. This allows to encode some relations between the contents of several queues.

Example: Considering a system with two queues with alphabets $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{c, d\}$. The QDD depicted on Fig. 13 defines the language $L = a^*\sharp(c + d)^* + a^*b(a + b)^*\sharp c^*d(c + d)^*$ and encodes the relation "if there is at least one b in the first queue, then there is at least one d in the second queue".

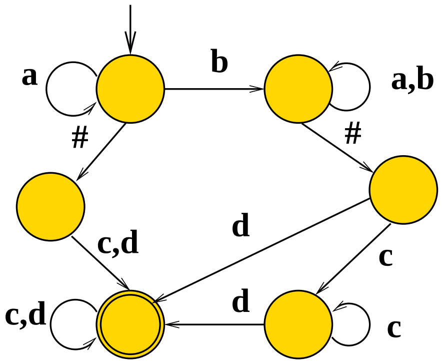

Figure 13: Example of QDD

Since QDDs are finite automata, the abstract lattice $(\mathcal{R}(\Sigma), \sqsubseteq^\sharp)$ can be defined as before, except that the widening operator ∇ must avoid merging the different queue contents.

7.2 Effective representation and manipulation of QDDs

A QDD is a finite automaton (as defined in definition 3) with a special character $\sharp \notin \Sigma$. This character marks the separation between two queue-contents. Let $\mathcal{M} = (Q, \Sigma \cup \{\sharp\}, Q_0, Q_f)$ be a QDD. A word $w = w_1 w_2 \ldots w_n$ is effectively accepted by a QDD if $\exists q_0 \in Q_0, q_f \in Q_f,\ \delta(q_0, w') = q_f)$ where $w' = w_1 \sharp w_2 \sharp \ldots \sharp w_n$.

Thus each state $q \in Q$ of the QDD can be associated to one (and only one) queue-content. This association is given by a function $a : Q \to [1 \ldots n]$. On the other hand, for an integer i, we can define $Q_i = \{q \in Q | a(q) = i\}$. The sub-automaton $\mathcal{M}_i = (Q_i, \Sigma_i, Q_{0,i}, Q_{f,i})$ recognizes the projection on the alphabet Σ_i of the language recognized by $\mathcal{M}$.

A state $q \in Q_i$ is *initial for a queue* i if a separation transition leads to this state (i.e. $\exists q' \in Q_{i-1}, q' \xrightarrow{\sharp} q$). The initial states for the first queue are the initial states of the automaton. A state $q \in Q_i$ is *final for a queue* i if a separation transition starts from this state (i.e. $\exists q' \in Q_{i-1}, q \xrightarrow{\sharp} q'$). The final states for the last queue are the final states of the automaton.

Concerning the standard operations ($\sqcup$, $\sqcap$, Send, Receive), and inclusion test $\subseteq$, they are natural extension of their counterpart for an automaton representing a single queue. In particular, they do not induce any approximation. This also implies that their result does no depend on the order on queues choosed for the representation.

7.3 The widening operator for QDDs

We will use the same widening operator as defined in section 4.2, but using $4n$ colors instead of 4. For a state q, we define its color $c(q)$:

- $c(q) = 4 * a(q)$ if q is both an initial and a final state for the queue $a(q)$
- $c(q) = 4 * a(q) + 1$ if q is a final (but not initial) state for the queue $a(q)$
- $c(q) = 4 * a(q) + 2$ if q is an initial (but not final) state for the queue $a(q)$
- $c(q) = 4 * a(q) + 3$ otherwise

A natural question arise: is such an operator invariant w.r.t. the ordering of queues ? The answer is unfortunately negative, as illustrated by the following example.

Example: Let us consider a CFSM with 2 channels, $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{c, d\}$, and the language: $L = aaaac + baaad$. This language L_1 is recognized by a QDD (Fig. 14). Choosing $k = 2$ for our widening operator, the two states marked with an "x" are merged, as well as the two states marked with an "y". Thus $\nabla_2 L_1 = (a+b)aa(c+d)$. If we change the ordering, we start from the language $L' = caaaa + dbaaa$ recognized by a second QDD (Fig. 15), which gives after widening $\nabla_2 L' = L'$. In this case, the second ordering preserve the information, unlike the first one.

Consequently, the precision of our analysis depends on the ordering. From a "philosophical" point of view, a widening operator which would be independent of the ordering would have been more satisfactory, but we did not find out yet such a widening operator, with good properties w.r.t. precision and efficiency. In a different setting however, where the aim is not to approximate but to compress the size of the representation, notice that the BDD representation for Boolean functions strongly depends on the ordering of variables.

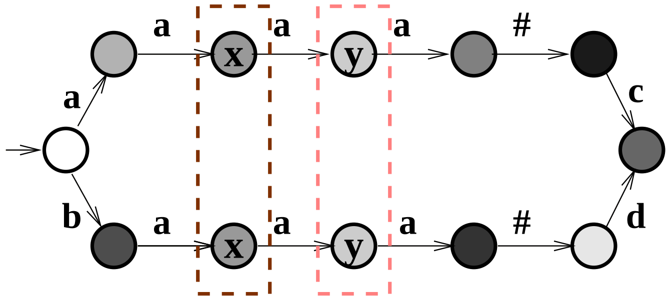

Figure 14: Equivalence classes for the first QDD

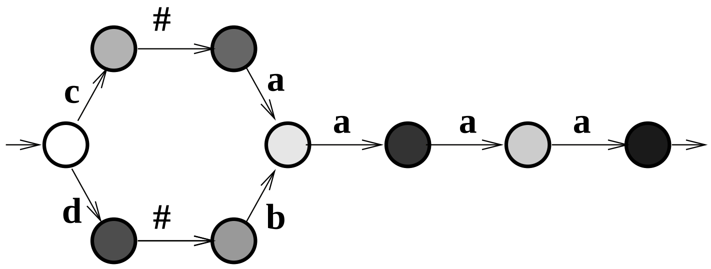

Figure 15: Equivalence classes for the second QDD

If we relax efficiency, we may obtain such a widening operator by considering all the permutations on queues. We would apply the widening on each representation, and then take the intersection of all the results. We conjecture that the obtained operator would have the property of a widening operator. Notice however that the number of permutation is exponential, and that such a solution would imply the ability of converting from the representation with one order to a representation with a different order. If we consider again the parallel with BDDs, such a reordering would be certainly more complex in our case than for BDDs, as we have cyclic graphs while BDDs are acyclic graphs.

8 Applications

In this section, we give some examples of rechability analysis based on the relational lattice (section 7) or the non-relational one (section 6). Both analysis were implemented using the language Objective CAML, so we have a tool for an automatic reachability analysis of CFSM. The precision analysis (as well as the cost of the computation) strongly depends on the choice of k used to define the operator ∇. The fix-point computation uses the two lattices defined in section 6 and section 7. The generic fix-point calculator also uses strategies we do not mention in this paper, the reader can refer to [10] for more details.

We analyse the two protocols we already described: the infinite buffer protocol (Example 2.3) and the connexion/deconnexion protocol. We also analyse some toy examples and the alernating bit protocol (ABP). For each analysis, we mention the choice of k and the cost

8.1 The infinite buffer

We recall the automaton-based description of the infinite buffer (Fig. 16).

This is a very simple protocol : the relational analysis and the non-relational one both find the exact reachability set $L_0 = (a+b)^*$. The result is obtained :

- in 4 iteration steps, if we choose $k = 0$ for the definition of the widening operator.

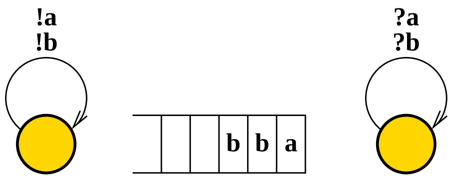

Figure 16: The infinite buffer model

- in 9 iteration steps, if we choose $k = 5$.

This example illustrates that the precision of the analysis may not be improved if we choose a greater value of k, whereas the complexity is growing. We guess that, for most CFSM, the analysis with $k = 1$ or $k = 2$ is precise enough.

8.2 The connexion/deconnexion protocol

The second example we used in previous sections is the connexion/deconnexion protocol (Fig. 17):

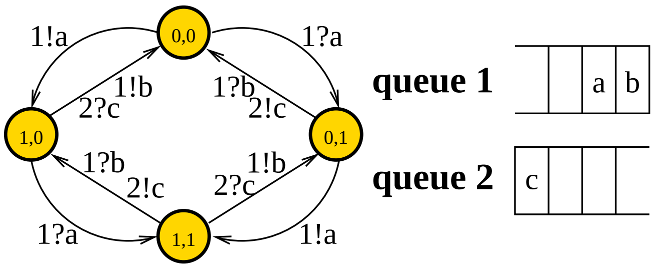

Figure 17: The connexion/deconnexion protocol

This protocol is a good example of what happens when the contents of the differents channels strongly depend on each other; in this case, the non-relational analysis is worse (both for precision and efficiency) than the relational one. We choose $k = 2$ to define the ∇ operaor in both cases.
Here is the result of the relational analysis (7 iteration steps):

0, 0		$(a.b)^*\#\varepsilon$
	$+$	$(a.b)^*\#\varepsilon$
	$+$	$(b.a)^*.a.b.(a.b)^*\#\varepsilon$
	$+$	$b.(a.b)^*\#c$
1, 0		$\varepsilon\#c$
	$+$	$a\#\varepsilon$
	$+$	$(a.b)^*.a\#\varepsilon$
	$+$	$(b.a)^*\#c$
	$+$	$(b.a)^*.a.(b.a)^*\#\varepsilon$
0, 1		$b.(a.b)^*\#\varepsilon$
1, 1		$\varepsilon\#\varepsilon$
	$+$	$(b.a)^*\#\varepsilon$

Here is the result of the non-relational analysis (8 iteration steps):

$0,0$	$a^* + a^*.b.(a^+.b)^*.a^*$	c^*
$1,0$	$a^* + a^*.b.(a^+.b)^*.a^+$	c^*
$0,1$	$a^* + a^*.b.(a^+.b)^*.a^*$	c^*
$1,1$	$a^+ + a^*.b.(a^+.b)^*.a^+$	c^*

The first analysis (but not the second one) shows that there are at most one c in the second queue, and that there are the same number of a and b (+/- 1) in the first queue[2].

The reason why the non-relational analysis is far worse and costly in this case is that it cannot detect that there is at most one 'c' in the second queue, thus many non-reachable states are explored by this analysis.

8.3 A toy example

The previous example shows that the non-relational analysis may be longer than the relational one. However, this is an exception. In general, this kind of analysis is faster. We provide an example where the non-relational analysis is much faster than the relational one. This toy example (Fig. 18) is a CFSM with 4 queues, where we can only add messages. A basic (human) analysis shows that the content of the two first queues is $(a.b)^*$, and the content of the two last ones is $(c.d)^*$. If we choose $k = 5$ (for the widening definition), the non-relational analysis stops in about one second and leads to the result (a^*, b^*, c^*, d^*).

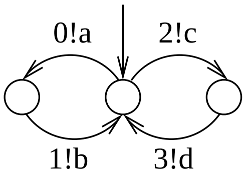

Figure 18: A toy example

The relational one takes several minutes for a similar result. In fact, the relational analysis will consider all the languages $L_{i,j} = \{a^i\#b^i\#c^j\#d^j\}$ with $i, j = 1..5$ before performing any widening operation.

8.4 The Alternating Bit Protocol

The Alternating Bit Protocol (ABP) is a data-transmission protocol, between a sender S and a receiver R. S transmits some data pakage m throught a FIFO channel C[3], and R and S exchange some information (one-bit messages) throught two channels K and L(Fig. 19).

[2]Indeed, this protocol is a faulty one, there is a correct version available in [18]. This correct version can be easily analyzed, since the length of the queue-content is bounded.

[3]This transmission is often modeled by two abstract actions SEND and RECEIVE instead of this third channel C.

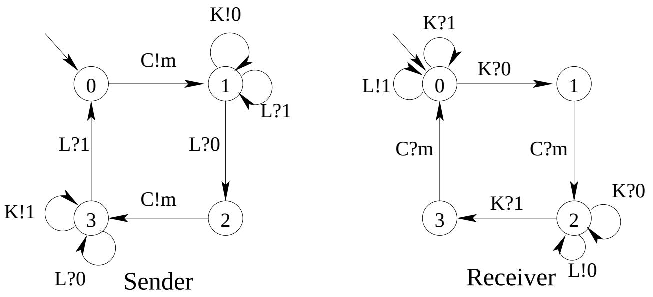

Figure 19: The Alternating Bit Protocol

We performed a relational analysis of the CFSM modeling this protocol:

Sender	Receiver	Contents K#L#C
0	0	$1^*\#1^*\#\varepsilon$
0	1	$\emptyset$
0	2	$\emptyset$
0	3	$\emptyset$
1	0	$1^*0^*\#1^*\#m$
1	1	$0^*\#1^*\#m$
1	2	$0^*\#1^*0^*\#\varepsilon$
1	3	$\emptyset$
2	0	$\emptyset$
2	1	$\emptyset$
2	2	$0^*\#0^*\#\varepsilon$
2	3	$\emptyset$
3	0	$1^*\#0^*1^*\#\varepsilon$
3	1	$\emptyset$
3	2	$0^*1^*\#0^*\#m$
3	3	$1^*\#0^*\#m$

This result was obtained with 8 iteration steps. It shows that the control states $(0,1)$, $(0,2)$, $(0,3)$, $(1,3)$, $(2,0)$, $(2,1)$, $(2,3)$ and $(3,1)$ are not reachable and that there is at most one message in data channel C. Comparing this result to the experimental results given in [5, 1] shows that we obtain the exact result.

9 Related works

The reachability analysis of FIFO channel systems has been studied for years, we must compare our method, based on Abstract Interpretation, with other ones, like the acceleration techniques.

9.1 Acceleration techniques

Those techniques aim at verifying automatically infinite-state systems, like unbounded FIFO channel systems. We give the example of analysis based on QDDs and acceleration to illustrate the acceleration techniques.

According to the collecting semantics(section 2), for a control state c we define a langage L_c representing all possible content of all the queues when in this control state. If L_c is regular, it can be represented by a QDD [5]. If there is a *loop* $\theta \triangleq c = c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} c_k = c$ in the control structure and a reachable set of states E representable by QDDs, we may compute in a single step the effect of the loop θ, i.e. finding the QDDs representing E_{θ^*}, the set of states that can be reached from a state of E following the loop θ an arbitrary number of times. For the exploration of the state space, we can remplace the entire loop by a single *meta-transition*. However, even if all loops may be accelerated, we still have to explore an infinite transition system (since there is an infinite number of loops). We may find some termination conditions [6] or use heuristics that lead to semi-algorithms: for example, we may "flat" the transition system and find a proper exploration order [3]. Instead of QDDs, we may also use a representation based on CQDDs [7] or *semi-linear regular expressions*(SLREs) [17], and use similar techniques. We may perform an analysis of lossy channel systems using a representation based on *linear regular expressions*(LREs)[2].

In the following table, we recall the acceleration techniques used for the analysis of systems with FIFO channels. "Queue type" can be a lossy channels (the messages can be lost) or perfect FIFO (no message loss). The "representation" means the way languages are represented (regular expressions or automata). The two other columns tell when a loop may be accelerated if the CFSM system has only one queue or if it has several queues.

queue type	representation	single queue	several queues	reference
lossy channel	LRE	always	always	[2]
perfect FIFO	QDD	always	sometimes	[5]
perfect FIFO	SLRE	always	sometimes	[17]
perfect FIFO	CQDD	always	always	[7]

Acceleration and widening techniques have the same purpose: guessing the limit of a sequence of sets of states. The difference between them is that the acceleration is exact, but is not always possible, whereas the widening is just an approximation that can be used at any time. Moreover, acceleration is strongly based on the transitions of the analyzed system, whereas widening, in its basic definition at least, considers only a sequence of abstract terms and does not exploit the structure of the system under analysis. Thus widening is more generic, but maybe less efficient, that acceleration.

9.2 Comparison with other works

Compared to acceleration techniques, the main advantage of our method based on Abstract Interpretation is that our analysis will always terminate.

But the guarantee of termination is useless if our result (an approximation of the exact result) is not enough precise. The example given in section 8 shows that we may obtain the exact result. In fact, with a relational analysis, our result is a QDD-representable language, so we may be as precise as all the acceleration methods (except an analysis based on the CQDD framework).

In other words, compared to methods based on semi-linear regular expressions or linear regular expressions, our analysis may be more precise (e.g. we can deal with nested loops), but more costly. Compared to methods based on QDD or CQDD, we cannot be more precise and we may not be as efficient, but we can deal with any kind of CFSM systems.

We may also be more precise than other approximate analysis of CFSM (e.g. [21]). These methode do not use the Abstract Interpretation framework and the result of the analysis is not a language describing the content of all queues but just a simple information, like "the queue is empty" or "the first message in the queue may be a, b or c".

Convergence of set of regular languages Our widening operator can be considered as a way of ensuring the convergence of a sequence of regular languages. It is a variant of a widening definition for regular languages mentionned in [16]. Other ways of finding the limit of a sequence of regular languages are proposed in [19, 9, 8]. However these methods do not aim at analysing specifically

CFSM. Hence, we think that they are less adapted to this model since they do not take into account the FIFO policy of the channels. That is why we did not use them for the definition of the widening operator.

10 Conclusion

In this paper, we explained how one can perform a reachability analysis of CFSM using Abstract Interpretation techniques. We first looked at systems with only one communication channel, defining an abstract lattice based on the set of regular languages. The widening operator of this abstract lattice is defined using an equivalence relation between the states of the automaton, similar to the k-order bisimulation relation. Concerning systems with several comunication channels, we highlighted the difference between a relational analysis, which preserv some relation between two different queue-contents, and a non-relational one. Then we proposed a non-relational lattice, based on tuple of regular languages, and a relational one, based on QDDs. Finally, we illustrated our method by performing a reachability analysis on examples of CFSM and we compaired our work with other methods, mostly based on acceleration techniques.

As we mentioned in section 9, other works aiming at analysing CFSM do not use the Abstract Interpretation framework. We may argue that using Abstract Interpretation for the analysis of CFSM systems has three advantages. The first one is that our analysis will always terminate and lead to a regular language describing all queue-contents. The use of the widening operator allows us to overcome the undecidability of the reachability problem. The second one is that this method is quite simple: we use the well-known regular languages as an abstract lattice, and all the convergence issues are solve by a widening operator based on the (also well-known) bisimulation. The third one is that the Abstract Interpretation framework is a good way to combine several lattices, if one wants to perform a special analysis. For example, if we want to have better information on the size of the queue, we would combine: the QDD-based lattice (section 7) and an abstract lattice counting the number of messages of each type on each queue.

We may also perform not only reachability analysis, but also verification of safety properties. If one wants to verify if a property p is satisfied by a system $\mathcal{M}$, one can use an observer $\mathcal{O}_p$ and perfom a reachability analysis of the system corresponding to the synchronus product $\mathcal{O}_p \times M$. At the present time, we must build this product manually, but we will soon using the methods defined here in the NBAC tool [20].

Other future works include the use of infinite alphabets. Here there was only a finite number of messages, but protocoles often used infinite number of messages. In fact, a lot of communication protocols use numbrer or "token" to identify the different frames. To take them into account, we must be able to deal with a maybe infinite number of messages. We may also defined a lattice based on CQDD-like objects (automata with counters), in order to get a better precision of the analysis. This lattice will be defined as a reduced product[4] of a regular languages lattice and another abstract lattice as defined in [15]. Another application of our work will be the supervisory control of asynchronous systems.

References

[1] Parosh Aziz Abdulla, Aurore Annichini, and Ahmed Bouajjani. Symbolic verification of lossy channel systems: Application to the bounded retransmission protocol. In *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 208–222. Springer-Verlag, 1999.

[4] the reduced product of two abstract lattices A and A' is the smallest lattice being as precise as both A and A'

[2] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 305–318, London, UK, 1998. Springer-Verlag.

[3] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST: Fast Acceleration of Symbolic Transition systems. In Warren A. Hunt, Jr and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 118–121, Boulder, Colorado, USA, July 2003. Springer.

[4] Gregor V. Bochmann. *Finite state description of communication protocols.* IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[5] Bernard Boigelot and Patrice Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. *Form. Methods Syst. Des.*, 14(3):237–255, 1997.

[6] Bernard Boigelot, Patrice Godefroid, Bernard Willems, and Pierre Wolper. The power of qdds (extended abstract). In *SAS '97: Proceedings of the 4th International Symposium on Static Analysis*, pages 172–186, London, UK, 1997. Springer-Verlag.

[7] Ahmed Bouajjani and Peter Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. *Theor. Comput. Sci.*, 221(1-2):211–250, 1999.

[8] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In *CAV*, pages 372–386, 2004.

[9] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *Computer Aided Verification*, pages 403–418, 2000.

[10] François Bourdoncle. *Sémantiques des Langages Impératifs d'Ordre Supérieur et Interprétation Abstraite.* PhD thesis, Ecole Polytechnique, 1992.

[11] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[12] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[13] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96, New York, NY, USA, 1978. ACM Press.

[14] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

[15] Alain Deutsch. Interprocedural may-alias analysis for pointers: beyond k-limiting. In *PLDI '94: Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*, pages 230–241. ACM Press, 1994.

[16] Jérôme Feret. Abstract interpretation-based static analysis of mobile ambients. In *Eighth International Static Analysis Symposium (SAS'01)*, number 2126 in LNCS. Springer-Verlag, 2001. © Springer-Verlag.

[17] Alain Finkel, S. Purushothaman Iyer, and Grégoire Sutre. Well-abstracted transition systems: application to fifo automata. *Information and Computation*, 181(1):1–31, 2003.

[18] C. Jard and M. Raynal. Specification of properties is required to verify distributed algorithms. In *Colloque AFCET sur les langages de spécification*, février 1987.

[19] David Lesens, Nicolas Halbwachs, and Pascal Raymond. Automatic verification of parameterized linear networks of processes. In *POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 346–357. ACM Press, 1997.

[20] The NBAC verification/slicing tool. http://www.irisa.fr/prive/bjeannet/nbac/nbac.html.

[21] Wuxu Peng and S. Puroshothaman. Data flow analysis of communicating finite state machines. *ACM Trans. Program. Lang. Syst.*, 13(3):399–442, 1991.

[22] Kenneth J. Turner. *Using Formal Description Techniques: An Introduction to Estelle, Lotos, and SDL*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[23] Pierre Wolper and Bernard Boigelot. An automata-theoretic approach to presburger arithmetic constraints (extended abstract). In *SAS '95: Proceedings of the Second International Symposium on Static Analysis*, pages 21–32, London, UK, 1995. Springer-Verlag.