Message-Passing Algorithms for the Verification of Distributed Protocols

Loïg Jezequel, Javier Esparza

To cite this version:

Loïg Jezequel, Javier Esparza. Message-Passing Algorithms for the Verification of Distributed Protocols. 15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2014), Jan 2014, San Diego, United States. hal-01699321

HAL Id: hal-01699321

https://hal.archives-ouvertes.fr/hal-01699321

Submitted on 2 Feb 2018

Message-Passing Algorithms for the Verification of Distributed Protocols

Loïg Jezequel and Javier Esparza

Institut für Informatik, Technische Universität München, Germany

Abstract. Message-passing algorithms (MPAs) are an algorithmic paradigm for the following generic problem: given a system consisting of several interacting components, compute a new version of each component representing its behaviour inside the system. MPAs avoid computing the full state space by propagating messages along the edges of the system interaction graph. We present an MPA for verifying local properties of distributed protocols with a tree communication structure. We report on an implementation, and validate it by means of two case studies, including an analysis of the PGM protocol.

Introduction

Message-passing algorithms (MPAs) are an algorithmic paradigm for problems (called reduction problems) that can be generically described as follows. The input to the problem is a system consisting of several components communicating in some way. When considered in isolation, each component has a set of behaviours. However, not all these behaviours are necessarily realizable within the system, since some actions may need the cooperation of other components. The problem consists of computing a new version of each component whose behaviours are those behaviours of the original component that are realizable within the system.

MPAs work by propagating messages containing information about the behaviour of parts of the system along the edges of its interaction graph. Before sending a message, a component can process it to remove redundant or useless information. This way MPAs avoid computing the full state space. MPAs can be applied on any system, however they ensure to solve the reduction problem only for systems whose interaction graph is a tree. A generic description of MPAs can be found in [1]. In particular, MPAs for distributed planning of [2, 3] have been developed in an automata theoretic setting. More precisely, to each component of the system is attached a (weighted) automaton, and interaction between components is modelled by means of automata-theoretic operations. The nice experimental results obtained with this approach on planning problems suggest to study the use of MPAs for solving more generic formal verification problems, and in this paper we explore this idea.

We present an MPA for verifying local properties of distributed protocols with a tree communication structure. Loosely speaking, “local” means that the property is defined from the point of view of one of the components of the protocol. For instance, a local property of a protocol for broadcast communication is that a receiver gets all messages sent by the source. On the other hand, the mutual exclusion property for a mutal exclusion protocol with N processes is an example of a global property (which may sometimes be equivalent to a local property, but not always).

We model components as labeled transition systems (LTSs) $\mathcal{L}_1, \ldots, \mathcal{L}_n$, communicating à la CSP by rendez-vous. We present a generic MPA parametrized by an equivalence relation $\equiv$, which is assumed to be a congruence with respect to parallel composition and hiding. The MPA computes LTSs $\mathcal{K}_1, \ldots, \mathcal{K}_n$ such that $\mathcal{K}_i \equiv (\mathcal{L}_1 || \cdots || \mathcal{L}_n) \setminus \overline{\Sigma}_i$, where $\overline{\Sigma}_i$ is the complement of the alphabet of $\mathcal{L}_i$ (that is, we hide all actions but those involving the i-th component). We call $\mathcal{K}_i$ the *update* of $\mathcal{L}_i$. For a system with n-components the MPA requires $2n - 2$ messages, which we show is optimal. We then present two different instances of the algorithm, suitable for checking safety and liveness properties, respectively. The first instance instantiates $\equiv$ with the standard trace semantics. The second one chooses for $\equiv$ the semantics consisting of the infinite traces of the LTS plus its set of divergences, i.e., the traces after which an infinite sequence of silent transitions can occur. For both semantics we report on an implementation of the MPA, and we describe the technique used to reduce the size of the messages.

We evaluate our two instances on two case studies: a mutual exclusion protocol for tree networks proposed in [4], and a version of the Pragmatic General Multicast Protocol (PGM) [5]. For the PGM we show how the result of the MPA allows us to identify potential problems of the protocol when the different parameters of the protocol are assigned unsuitable values.

Related work. The connection of our work to other work on MPAs has been described above.

Several compositional approaches to verification exist, aiming at avoiding the state-explosion problem in the verification of distributed systems by considering them component by component. In [6] the authors use an assume-guarantee way of reasoning in which they show that a component guarantees some property as soon as it is in a system satisfying some assumption, the hard part being to choose good assumptions, which is achieved by progressively learning them. In [7] the authors introduce thread-modular model checking where the states of a multithreaded software are enumerated thread by thread (a state taking into account the value of the program counter of a single thread instead of the value of the program counters of all threads), potentially leading to an exponential reduction of the size of the state space considered, at the price of doing incomplete verification.

Our work is based on the possibility to replace a component of a system by an equivalent one (with respect to a suitable notion of equivalence or preorder), but smaller in some way. This is also the starting point of works such as [8, 9] for example. This approach has been implemented for trace, failures, and bisimulation equivalence or related preorders in tools like FDR [10, 11], the Concurrency Workbench [12], CADP [13] and others, and most model-checkers using abstraction techniques use it in some way. However, these tools address the problem of computing an update of the whole system, instead of an update for each component, and leave the choice of which components to minimize or reduce with respect to the given congruence, and in which order, to the user. (In particular, this hinders a direct comparison with these tools, since we would have to compare a fully automatic and a partially manual procedure. On the other hand, our algorithm could also be implemented on top of these tools.) By focusing on this problem we are able to provide a simple algorithm with a minimal number of exchanged messages.

Protocols with tree communication structure have also been analyzed by means of regular model checking (see [14] for a recent survey). The goal of regular model checking is more ambitious than ours, since it aims at proving the protocol correct for an arbitrary number of processes. On the other hand, this reduces the range of protocols that can be verified. In particular, we do not know of any analysis of our two case studies using regular model checking.

The PGM protocol has been analysed in a number of papers [15–17]. This work is orthogonal to ours. In [15, 16] the focus is on timing aspects and relations between parameters, which are analysed for small instances of the protocol (below five processes), while we concentrate on analyzing simpler properties of larger instances with more than one hundred processes. Finally, the work in [17] is a manual proof providing cut-off bounds for parametric analysis.

1 Definitions and notations

LTS. A *labelled transition system with silent transitions* (LTS) is a tuple $\mathcal{L} = (\Sigma, S, T, s^0)$ where Σ is a finite set of *labels*, S is a finite set of *states*, $T = \{T^\sigma \ : \ \sigma \in \Sigma\} \cup \{T^\tau\}$ is a set of *transition relations*: $\forall \sigma \in \Sigma \cup \{\tau\}, T^\sigma \subseteq S \times S$ (the elements of T^σ are called *σ-transitions*), and $s^0 \in S$ is an *initial state*. A finite (resp. infinite) sequence of labels and τ, $tr = \sigma_1\sigma_2\ldots$ is a *trace* of $\mathcal{L}$ if there exits a finite (resp. infinite) alternating sequence of states and transitions (a *path*) $\pi = s_0t_1s_1t_2\ldots$ *realizing* it, that is, such that $s^0 = s_0$, and $t_i = (s_{i-1}, s_i) \in T^{\sigma_i}$ for all $i > 0$. A finite (resp. infinite) sequence of labels tr is an *observable trace* of $\mathcal{L}$ if there exists a trace tr' of $\mathcal{L}$ such that by removing all τ from tr' one gets tr: $tr'|_\Sigma = tr$. The set of finite observable traces of $\mathcal{L}$ is denoted by $\mathcal{T}^*_\mathcal{L}$ while its set of infinite observable traces is denoted by $\mathcal{T}^\omega_\mathcal{L}$, and the set of all its observable traces is $\mathcal{T}_\mathcal{L} = \mathcal{T}^*_\mathcal{L} \cup \mathcal{T}^\omega_\mathcal{L}$. We write $s_0 \overset{\tau}{\leadsto}_\mathcal{L} s_n$ if there exists a path $\pi = s_0t_1s_1t_2\ldots s_n$ such that $\forall 1 \leq i \leq n, t_i \in T^\tau$. Figure 1 gives some examples of LTSs, states are represented by circles and labelled transitions by labelled arrows between states. Initial states are distinguished with small arrows.

Definition 1. *The* hiding *of a set Σ' in an LTS $\mathcal{L} = (\Sigma, S, T, s^0)$ is the LTS $\mathcal{L} \setminus \Sigma' = (\Sigma \setminus \Sigma', S, T', s^0)$ with T' such that ${T'}^\sigma = T^\sigma$ for any $\sigma \in \Sigma \setminus \Sigma'$ and ${T'}^\tau = \cup_{\sigma \in \Sigma' \cup \{\tau\}} T^\sigma$.*

For $\mathcal{L} = (\Sigma, S, T, s^0)$ and some set of labels Σ' we sometimes write $\mathcal{L} \setminus \overline{\Sigma}'$ for $\mathcal{L} \setminus (\Sigma \setminus \Sigma')$.

Definition 2. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be LTSs where $\mathcal{L}_i = (\Sigma_i, S_i, T_i, s_i^0)$. Their* parallel composition *$\mathcal{L}_1 || \mathcal{L}_2$ is the LTS $\mathcal{L} = (\Sigma, S, T, s^0)$ such that: $\Sigma = \Sigma_1 \cup \Sigma_2$, $S = S_1 \times S_2$, if $\sigma \in \Sigma_1 \cap \Sigma_2$ then $T^\sigma = \{((s_1, s_2), (s_1', s_2')) : (s_1, s_1') \in T_1^\sigma \wedge (s_2, s_2') \in T_2^\sigma\}$, if $\sigma \in \Sigma_1 \setminus \Sigma_2$ then $T^\sigma = \{((s_1, s_2), (s_1', s_2)) : (s_1, s_1') \in T_1^\sigma \wedge s_2 \in S_2\}$, if $\sigma \in \Sigma_2 \setminus \Sigma_1$ then $T^\sigma = \{((s_1, s_2), (s_1, s_2')) : s_1 \in S_1 \wedge (s_2, s_2') \in T_2^\sigma\}$, and $T^\tau = \{((s_1, s_2), (s_1', s_2)) : (s_1, s_1') \in T_1^\tau \wedge s_2 \in S_2\} \cup \{((s_1, s_2), (s_1, s_2')) : s_1 \in S_1 \wedge (s_2, s_2') \in T_2^\tau\}$, and finally $s^0 = (s_1^0, s_2^0)$.*

Notice that this parallel composition is commutative and associative, so one can safely write $\mathcal{L} = \mathcal{L}_1 || \dots || \mathcal{L}_n$ for the parallel composition of more than two LTSs. The right LTS in Figure 3 is the parallel composition of the middle LTS in the same figure and the right LTS in Figure 1.

Definition 3. *An equivalence relation between LTSs is called a* congruence for LTSs*, denoted by $\equiv$, if for any LTSs $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}$ such that $\mathcal{L}_1 \equiv \mathcal{L}_2$ and any set of labels Σ one has $\mathcal{L}_1 || \mathcal{L} \equiv \mathcal{L}_2 || \mathcal{L}$ and $\mathcal{L}_1 \setminus \Sigma \equiv \mathcal{L}_2 \setminus \Sigma$.*

MLTS. A *marked labelled transition system* (MLTS), or LTS with marked states, is a tuple $\mathcal{ML} = (\mathcal{L}, F)$ where $\mathcal{L} = (\Sigma, S, T, s^0)$ is an LTS and $F \subseteq S$ is a set of *marked states*. According to the set of marked states one can define the set of *marked traces* $\mathcal{MT}_{\mathcal{ML}}$ of $\mathcal{ML}$ as the set of traces for which there exists a realization verifying some condition on the marked states (examples are given below: automata and Büchi automata).

Definition 4. *Given MLTSs $\mathcal{ML}_1, \mathcal{ML}_2$ where $\mathcal{ML}_i = (\mathcal{L}_i, F_i)$, their* parallel composition *$\mathcal{ML}_1 || \mathcal{ML}_2$ is the MLTS $\mathcal{ML} = (\mathcal{L}, F)$ such that $\mathcal{L} = \mathcal{L}_1 || \mathcal{L}_2$ and $F = F_1 \times F_2$.*

FSA. A *finite state automaton* (FSA) is an MLTS $\mathcal{A} = (\mathcal{L}, F)$ such that a trace tr of $\mathcal{A}$ is a marked trace if and only if it is finite and it has a realization $\pi = s_0 t_1 s_1 \dots s_k$ such that $s_k \in F$. The set $\mathcal{MT}_{\mathcal{A}}$ is usually called the language of $\mathcal{A}$.

NBA. A *Büchi automaton* (NBA) is an MLTS $\mathcal{B} = (\mathcal{L}, F)$ such that a trace tr of $\mathcal{B}$ is a marked trace if and only if it is infinite and it has a realization $\pi = s_0 t_1 s_1 t_2 \dots$ such that there exists an infinite number of $i \geq 0$ for which $s_i \in F$. As for FSAs, the set $\mathcal{MT}_{\mathcal{B}}$ is usually called the language of $\mathcal{B}$.

2 Message-passing algorithms

Before presenting a formal description of message-passing algorithms, we illustrate them on an example. Consider the three LTSs of Figure 1. They represent an abstract view of a small distributed system involving three processes: a sender (S), a capacity one channel (C), and a receiver (R). S has to accomplish some task. It initially does a choice between doing it alone (right transition) or doing it together with R (left transition) by exchanging some messages through C.

The behaviour of this system is captured by $\mathcal{L} = S||C||R$, and so – denoting by Σ_S (resp. Σ_C, Σ_R) the set of labels of S (resp. C, R) – the behaviour of S (resp. C, R) inside this system is captured by $\mathcal{L} \setminus \overline{\Sigma}_S$ (resp. $\mathcal{L} \setminus \overline{\Sigma}_C$, $\mathcal{L} \setminus \overline{\Sigma}_R$).

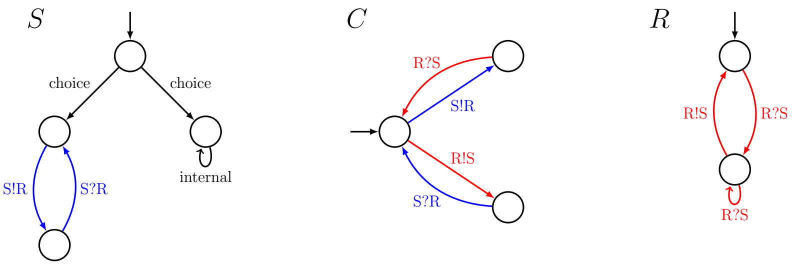

Fig. 1. A distributed system constituted of three interacting LTSs.

Definition 5. *The* interaction graph *of a system $\mathcal{L}_1 || \cdots || \mathcal{L}_n$, where Σ_i is the alphabet of $\mathcal{L}_i$, has $\mathcal{L}_1, \ldots, \mathcal{L}_n$ as nodes, and an edge $\{\mathcal{L}_i, \mathcal{L}_j\}$ when $\Sigma_i \cap \Sigma_j \neq \emptyset$.*

The MPAs can solve the reduction problems for systems whose interaction graph is a tree. They proceed by sending messages (which have the same type as the components, i.e. LTSs in our example) along the edges of the tree, i.e. each component sends a message to each of its neighbours. In the system of Figure 1, the interaction graph is a line (and so a tree) with C in the middle and S and R at the extremities (see Figure 2). Indeed S interacts (that is, shares labels) only with C and R also interacts only with C. So each of S and R will send a message to C and C will send a message to each of S and R.

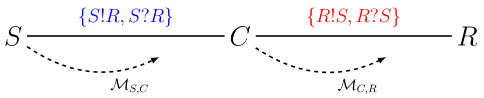

Fig. 2. The interaction graph of the system of Figure 1. Over each edge (plain line) the corresponding set of shared labels is indicated. Dashed lines represent the messages propagated from S to R.

The idea behind these messages is the following. In a tree shaped interaction graph, each edge separates the graph into two subtrees whose roots are the extremities of the edge (in our example, the edge (C, R) separates the graph into a tree containing only R and a tree with C as root and S as leaf). Using this fact each component at the extremity of the removed edge will send a message to the other component. This message describes the possible behaviours of the subtree from which its sender is the root as they can be seen by its receiver (for example C sends a message to R describing the behaviour of $S||C$ as R sees it, that is using only labels shared between C and R). This message is computed from the messages received from the neighbours of its sender in the subtree from which this sender is the root.

In the system of Figure 1 the message from S to C is then $\mathcal{M}_{S,C} \equiv S \setminus \overline{\Sigma_S \cap \Sigma_C}$ (see Figure 3, left for an example preserving observable traces) and it is then used to build the message from C to R: $\mathcal{M}_{C,R} \equiv (\mathcal{M}_{S,C}||C) \setminus \overline{\Sigma_C \cap \Sigma_R}$ (see Figure 3, middle). Similarly the remaining messages can be built: $\mathcal{M}_{R,C} \equiv R \setminus \overline{\Sigma_R \cap \Sigma_C}$ and $\mathcal{M}_{C,S} \equiv (\mathcal{M}_{R,C}||C) \setminus \overline{\Sigma_C \cap \Sigma_S}$. It can then be proved (it is a consequence of Theorem 1 below) using the separation property of the edges of a tree shaped interaction graph described above, that the composition of each component with all the messages it received describes the behaviour of this component in the full system. For example (see Figure 3, right) $R' = R||\mathcal{M}_{C,R}$ has the same set of observable traces than $\mathcal{L} \setminus \overline{\Sigma}_R$. Similarly $S' = S||\mathcal{M}_{C,S}$ and $C' = C||\mathcal{M}_{R,C}||\mathcal{M}_{S,C}$ have the same set of observable traces than $\mathcal{L} \setminus \overline{\Sigma}_S$ and $\mathcal{L} \setminus \overline{\Sigma}_C$ respectively.

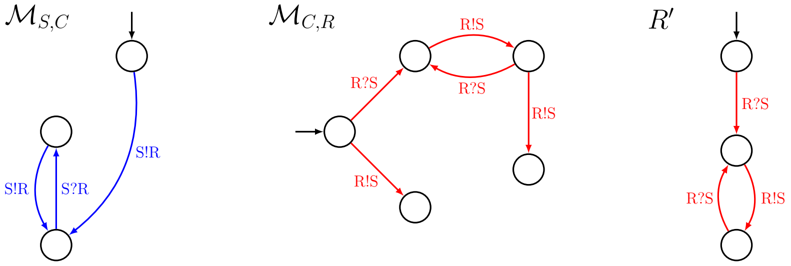

Fig. 3. Messages from S to R, updated component R'.

2.1 Formal description of an MPA for LTSs

Algorithm 1 below presents a formal description of an MPA for a system $\mathcal{L} = \mathcal{L}_1|| \dots ||\mathcal{L}_n$ whose interaction graph $\mathcal{G} = (V, E)$ is a tree. Any LTS $\mathcal{K}_i$ obtained at the end of Algorithm 1 is called the *update* of $\mathcal{L}_i$ (in $\mathcal{G}$).

For the presentation it is convenient to model an undirected edge $\{\mathcal{L}_i, \mathcal{L}_j\}$ as two directed edges $(\mathcal{L}_i, \mathcal{L}_j)$ and $(\mathcal{L}_j, \mathcal{L}_i)$. So the input to Algorithm 1 is a directed graph $\mathcal{G}$ derived from a tree in this way.

We denote by $x :\equiv \mathcal{L}$ that the variable x is assigned some LTS $\mathcal{L}'$ such that $\mathcal{L}' \equiv \mathcal{L}$. Notice that this is a nondeterministic assignment, and so Algorithm 1 is nondeterministic. In the next section we present several instances of the algorithm, and for each one we explain how the nondeterminism is resolved.

In order to formulate and prove the correctness of the algorithm we introduce some notations.

Algorithm 1 An MPA for LTSs

```
Input: an interaction graph $\mathcal{G} = (V, E)$ with $V = \{\mathcal{L}_1, \dots, \mathcal{L}_n\}$
1: $\mathsf{M} \leftarrow E$
2: while $\mathsf{M} \neq \emptyset$ do
3:     choose $(\mathcal{L}_i, \mathcal{L}_j) \in \mathsf{M}$ such that $(\mathcal{L}_k, \mathcal{L}_i) \notin \mathsf{M}$ for every $k \neq j$
4:     $\mathcal{M}_{i,j} :\equiv (\mathcal{L}_i \mid\mid (\mid\mid_{k \neq j,\ (\mathcal{L}_k, \mathcal{L}_i) \in E} \mathcal{M}_{k,i})) \setminus \overline{\Sigma}_j$
5:     remove $(\mathcal{L}_i, \mathcal{L}_j)$ from $\mathsf{M}$
6: end while
7: for all $i \in V$ do
8:     $\mathcal{K}_i :\equiv \mathcal{L}_i \mid\mid (\mid\mid_{(\mathcal{L}_j, \mathcal{L}_i) \in E} \mathcal{M}_{j,i})$
9: end for
```

Definition 6. *Given $\mathcal{G} = (V, E)$ with $V = \{\mathcal{L}_1, \dots, \mathcal{L}_n\}$, we denote the parallel composition $(\mathcal{L}_1 \mid\mid \cdots \mid\mid \mathcal{L}_n)$ by $\widehat{\mathcal{G}}$.*
Given a tree $\mathcal{G} = (V, E)$ and $(\mathcal{L}_i, \mathcal{L}_j) \in E$, we denote by $\mathcal{G}_{ij}$ the maximal subtree of $\mathcal{G}$ containing $\mathcal{L}_j$ but not $\mathcal{L}_i$.

Lemma 1. *Let $\mathcal{G} = (V, E)$ be a tree and $(\mathcal{L}_i, \mathcal{L}_j) \in E$. Let $\mathcal{M}^{\mathcal{G}}_{j,i}$ be the content of variable $\mathcal{M}_{j,i}$ after termination of Algorithm 1 on input $\mathcal{G}$. Then $\mathcal{M}^{\mathcal{G}}_{j,i} = \widehat{\mathcal{G}}_{ij} \backslash \overline{\Sigma}_i$.*

Proof. The proof is by induction on the depth of $\mathcal{G}^{ij}$. If $\mathcal{G}^{ij}$ has depth 1, then $\mathcal{G}_{ij}$ contains the vertex $\mathcal{L}_j$ and no arcs. By line 4 we get $\mathcal{M}^{\mathcal{G}}_{j,i} \equiv \mathcal{L}_i \setminus \overline{\Sigma}_i = \widehat{\mathcal{G}}_{ij} \setminus \overline{\Sigma}_j$ (recall that, since we represent undirected trees as directed graphs, we have $(\mathcal{L}_j, \mathcal{L}_i) \in E$).

Assume now that $\mathcal{G}^{ij}$ has depth larger than 1. Let $\mathcal{L}_{j_1}, \dots, \mathcal{L}_{j_m}$ be the neighbours of $\mathcal{L}_j$ in $\mathcal{G}_{ij}$. Then the trees $\mathcal{G}jj_1, \dots \mathcal{G}jj_m$ are proper subtrees of $\mathcal{G}ij$, and in particular have smaller depth. By induction hypothesis we have

$$\mathcal{M}^{\mathcal{G}_{ij}}_{j_1,j} \equiv \widehat{\mathcal{G}}_{jj_1} \setminus \overline{\Sigma}_j \quad \dots \quad \mathcal{M}^{\mathcal{G}_{ij}}_{j_m,j} \equiv \widehat{\mathcal{G}}_{jj_m} \setminus \overline{\Sigma}_j \tag{1}$$

By line 4 of the algorithm, and since $\equiv$ is a congruence, we get

$$\begin{aligned}
\mathcal{M}^{\mathcal{G}}_{j,i} &\equiv (\ \mathcal{L}_j \mid\mid (\mid\mid_{k \neq i,\ (\mathcal{L}_k, \mathcal{L}_j) \in E} \mathcal{M}^{\mathcal{G}}_{k,j})\ ) \setminus \overline{\Sigma}_i && (2)\\
&\equiv (\ \mathcal{L}_j \mid\mid \mathcal{M}^{\mathcal{G}}_{j_1,j} \mid\mid \cdots \mid\mid \mathcal{M}^{\mathcal{G}}_{j_m,j}\ ) \setminus \overline{\Sigma}_i && (3)\\
&\equiv (\ \mathcal{L}_j \mid\mid \mathcal{M}^{\mathcal{G}_{ij}}_{j_1,j} \mid\mid \cdots \mid\mid \mathcal{M}^{\mathcal{G}_{ij}}_{j_m,j}\ ) \setminus \overline{\Sigma}_i && (4)\\
&\equiv (\ \mathcal{L}_j \mid\mid (\widehat{\mathcal{G}}_{jj_1} \setminus \overline{\Sigma}_j) \mid\mid \cdots \mid\mid (\widehat{\mathcal{G}}_{jj_m} \setminus \overline{\Sigma}_j)\ ) \setminus \overline{\Sigma}_i && (5)
\end{aligned}$$

where (4) follows from (3) because $\mathcal{L}_k$ is a node of $\mathcal{G}_{ij}$ for every $\mathcal{M}_{k,j}$.

Since $\mathcal{G}$ is a tree, the sets of labels of $\widehat{\mathcal{G}}_{jj_1}, \dots, \widehat{\mathcal{G}}_{jj_m}$ are pairwise disjoint, and so

$$\mathcal{L}_j \mid\mid (\widehat{\mathcal{G}}_{jj_1} \setminus \overline{\Sigma}_j) \mid\mid \cdots \mid\mid (\widehat{\mathcal{G}}_{jj_m} \setminus \overline{\Sigma}_j) \equiv (\mathcal{L}_j \mid\mid \widehat{\mathcal{G}}_{jj_1} \mid\mid \cdots \mid\mid \widehat{\mathcal{G}}_{jj_m}) \setminus \overline{\Sigma}_j \tag{6}$$

Moreover, for the same reason, there are no edges between any of $\mathcal{L}_{j_1}, \dots, \mathcal{L}_{j_m}$ and $\mathcal{L}_i$. So $\Sigma_i \cap (\Sigma_{j_1} \cup \cdots \cup \Sigma_{j_m}) = \emptyset$, which implies

$$(\mathcal{L}_j \mid\mid \widehat{\mathcal{G}}_{jj_1} \mid\mid \cdots \mid\mid \widehat{\mathcal{G}}_{jj_m}) \setminus \overline{\Sigma}_j \setminus \overline{\Sigma}_i \equiv (\mathcal{L}_j \mid\mid \widehat{\mathcal{G}}_{jj_1} \mid\mid \cdots \mid\mid \widehat{\mathcal{G}}_{jj_m}) \setminus \overline{\Sigma}_i \tag{7}$$

Putting together (4)-(7), we obtain

$$\mathcal{M}_{j,i} \equiv (\mathcal{L}_j \mid\mid \widehat{\mathcal{G}}_{jj_1} \mid\mid \cdots \mid\mid \widehat{\mathcal{G}}_{jj_m}) \setminus \overline{\Sigma}_i \tag{8}$$

By definition of $\widehat{\mathcal{G}}_{ij}$, we have

$$\widehat{\mathcal{G}}_{ij} \equiv \mathcal{L}_j \mid\mid \widehat{\mathcal{G}}_{jj_1} \mid\mid \cdots \mid\mid \widehat{\mathcal{G}}_{jj_m} \tag{9}$$

which together with (8) yields

$$\mathcal{M}^{\mathcal{G}}_{j,i} \equiv \widehat{\mathcal{G}}_{ij} \setminus \overline{\Sigma}_i \tag{10}$$

as desired.

We can now prove correctness of Algorithm 1.

Theorem 1. *Let $\mathcal{G} = (V, E)$ with $V = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$ be a tree-shaped interaction graph. The result of running Algorithm 1 on $\mathcal{G}$ are LTSs $\mathcal{K}_1, \ldots, \mathcal{K}_n$ such that $\mathcal{K}_i \equiv \widehat{\mathcal{G}} \setminus \overline{\Sigma}_i$ for every $\mathcal{L}_i \in V$.*

Proof. We have

$$\begin{aligned}
\mathcal{K}_i &\equiv \mathcal{L}_i \mid\mid (\mid\mid_{(\mathcal{L}_j, \mathcal{L}_i) \in E}\ \mathcal{M}^{\mathcal{G}}_{i,j}) && (11)\\
&\equiv \mathcal{L}_i \mid\mid (\mid\mid_{(\mathcal{L}_j, \mathcal{L}_i) \in E}\ (\widehat{\mathcal{G}}_{ij} \setminus \overline{\Sigma}_i)) && (12)\\
&\equiv \mathcal{L}_i \mid\mid (\mid\mid_{(\mathcal{L}_j, \mathcal{L}_i) \in E}\ \widehat{\mathcal{G}}_{ij}) \setminus \overline{\Sigma}_i && (13)\\
&\equiv \widehat{\mathcal{G}} \setminus \Sigma_i && (14)
\end{aligned}$$

Here, (11) follows from line 8 of the algorithm; (12) follows from Lemma 1; (13) follows from the fact that no two neighbours of $\mathcal{L}_i$ are connected by an edge, and so their sets of labels are disjoint. Finally, (14) follows from the definitions of $\widehat{\mathcal{G}}$ and $\widehat{\mathcal{G}}_{ij}$.

If the messages of Algorithm 1 are large in the worst case (in theory they can have the size of the full system) their number is optimal. More precisely, no MPA using fewer messages can be correct, where the only assumption we make about MPAs is that the output $\mathcal{K}_i$ is a function of $\mathcal{L}_i$ and the messages $\mathcal{L}_i$ receives from its neighbours.

Theorem 2. *For every correct MPA algorithm and every $n \geq 1$ there is a graph $\mathcal{G}_n$ such that the algorithm requires at least $2n - 2$ messages on $\mathcal{G}_n$.*

Proof. Assume there is a correct MPA $\mathcal{A}$ which always requires less than $2n-2$ messages on graphs with n nodes. Consider the system $\mathcal{S}_1 = \mathcal{L}_1 || \cdots || \mathcal{L}_n$ where for every $1 \leq i \leq n$ the only maximal trace of $\mathcal{L}_i$ is $a_i a_{i+1} b_{i+1} b_i$ (its interaction graph is a line). Since the algorithm needs fewer than $2n-2$ messages, there is an index i such that either $\mathcal{L}_i$ sends no message to $\mathcal{L}_{i-1}$, or sends no message to $\mathcal{L}_{i+1}$.

In the first case, consider the system $\mathcal{S}_2$ which is identical to $\mathcal{S}_1$, except that the only maximal trace of $\mathcal{L}_i$ is $a_i b_{i+1} a_{i+1} b_i$. For $\mathcal{S}_1$ the only maximal trace of $\mathcal{K}_n$ is $a_n a_{n+1} b_{n+1} b_n$, while for $\mathcal{S}_2$ the only maximal trace of $\mathcal{K}_n$ is $a_n a_{n+1}$. Since, by our assumption on MPAs, $\mathcal{L}_{i-1}$ does not receive any message from $\mathcal{L}_i$, it returns the same result $\mathcal{K}_n$ in both cases, and so the algorithm is incorrect.

In the second case, consider the system $\mathcal{S}_2$ which is identical to $\mathcal{S}_1$, except that the only maximal trace of $\mathcal{L}_{i+1}$ is $a_i b_{i+1} a_{i+1} b_i$, and proceed analogously with $\mathcal{K}_1$ instead of $\mathcal{K}_n$.

Observations The restriction to tree-shaped interaction graphs can be weakened in several ways.

Communication graphs. We can replace the interaction graph by a potentially smaller communication graph, thus reducing the number of messages.

Definition 7. *Let $\mathcal{G}$ be any subgraph of an interaction graph. An edge $(\mathcal{L}_i, \mathcal{L}_j) \in E$ is* redundant *if there exists a sequence $(\mathcal{L}_i, \mathcal{L}_{k_1})(\mathcal{L}_{k_1}, \mathcal{L}_{k_2}) \ldots (\mathcal{L}_{k_\ell}, \mathcal{L}_j)$ of edges such that $i \neq k_m \neq j$ and $\Sigma_{\mathcal{L}_{k_m}} \supseteq \Sigma_{\mathcal{L}_i} \cap \Sigma_{\mathcal{L}_j}$ for every $1 \leq k_m \neq j$.*

A communication graph of a system is any subgraph of the interaction graph obtained by iteratively removing redundant edges.

If some communication graph of a system is a tree then all its communication graphs are trees [1]. We then say that the system *lives on a tree.* The following proposition shows that the MPA can be applied to any system that lives on a tree, even if its interaction graph is not a tree.

Proposition 1. *Let $\mathcal{G} = (V, E)$ with $V = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$ be a tree-shaped communication graph of $\mathcal{L} = \mathcal{L}_1 || \ldots || \mathcal{L}_n$. The result of running Algorithm 1 on $\mathcal{G}$ are LTSs $\mathcal{K}_1, \ldots, \mathcal{K}_n$ such that $\mathcal{K}_i \equiv \widehat{\mathcal{G}} \setminus \overline{\Sigma}_i$ for every $\mathcal{L}_i \in V$.*

Proof. The proof follows the lines of Theorem 2. We just have to adjust the arguments justifying Equations (6) and (7) in Lemma 1, and Equation (13) in Theorem 2. For (6) and (13) observe that, if the communication graph is a tree, then there are no edges between the neighbours of $\mathcal{L}_j$. Therefore, by the definition of communication graph, every common label of any two processes in (5) belongs to Σ_j which suffices to derive (6) and (13).

For (7) we observe that the communication graph also contains no edges between any of $\mathcal{L}_{j_1}, \ldots, \mathcal{L}_{j_m}$ and $\mathcal{L}_i$. So we have $(\Sigma_{j_1} \cup \ldots \cup \Sigma_{j_m}) \cap \mathcal{L}_i \subseteq \mathcal{L}_j$, and so

$$(\mathcal{L}_j \;||\; \widehat{\mathcal{G}}_{jj_1} || \cdots || \widehat{\mathcal{G}}_{jj_m}) \setminus \overline{\Sigma}_j \setminus \overline{\Sigma}_i \equiv (\mathcal{L}_j \;||\; \widehat{\mathcal{G}}_{jj_1} || \cdots || \widehat{\mathcal{G}}_{jj_m}) \setminus \overline{\Sigma}_i$$

Tree decompositions. Any system $\mathcal{L} = \mathcal{L}_1 || \dots || \mathcal{L}_n$ can be transformed into an equivalent one that lives on a tree. This is in itself trivial, since we can always choose this system as one single LTS equivalent to $\mathcal{L}$. However, this destroys the concurrency of the system. In order to preserve as much concurrency as possible we can compute a *tree decomposition* of a communication graph [18]. Every set $\{\mathcal{L}_{i_1}, \dots, \mathcal{L}_{i_k}\}$ of the decomposition is then replaced by any single LTS equivalent to the subsystem $\mathcal{L}_{i_1} || \dots || \mathcal{L}_{i_k}$. For instance, if the interaction graph of $\mathcal{L}_1 || \dots || \mathcal{L}_n$ is a ring with an even number of components, we can take the system $\mathcal{L}'_1 || \dots || \mathcal{L}'_{n/2}$, where $\mathcal{L}'_i \equiv \mathcal{L}_i || \mathcal{L}_{n-i+1}$ (Figure 4).

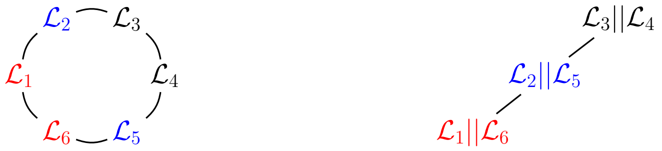
$\mathcal{L}_2$
$\mathcal{L}_3$
$\mathcal{L}_1$
$\mathcal{L}_4$
$\mathcal{L}_6$
$\mathcal{L}_5$
$\mathcal{L}_3||\mathcal{L}_4$
$\mathcal{L}_2||\mathcal{L}_5$
$\mathcal{L}_1||\mathcal{L}_6$

Fig. 4. A possible tree decomposition (right) of a ring-shaped interaction graph (left).

Computing one summary. Finally, we observe that computing one single update of component only requires to exchange $n - 1$ messages instead of $2n - 2$.

Proposition 2. *Let $\mathcal{L}_1, \dots, \mathcal{L}_n$ be LTSs such that the system $\mathcal{L} = \mathcal{L}_1 || \dots || \mathcal{L}_n$ lives on a tree, and let $1 \leq i \leq n$. The update $\mathcal{K}_i$ of $\mathcal{L}_i$ can be computed by an MPA that uses only $n - 1$ messages.*

Proof. Consider a communication graph $\mathcal{G} = (V, E)$ of $\mathcal{L} = \mathcal{L}_1, \dots, \mathcal{L}_n$ that is a tree. If $n = 1$ then $i = 1$ and $\mathcal{L}_1 = \mathcal{K}_1$ is computed using $n - 1 = 0$ messages. Assume the proposition is true up to $n = k$. Consider $n = k + 1$ and take $1 \leq i \leq n$. Remark that $\mathcal{K}_i$ is computed exactly from the messages of the form $\mathcal{M}_{\mathcal{L}_j, \mathcal{L}_i}$ for $(\mathcal{L}_j, \mathcal{L}_i) \in E$ (denote by E_i the set containing these edges). Given such a $\mathcal{L}_j$, consider the largest subtree of $\mathcal{G}$ rooted in $\mathcal{L}_j$ which does not contains $\mathcal{L}_i$. This subtree has $n_j < k + 1$ nodes, so the update of $\mathcal{L}_j$ in this subtree can be computed from $n_j - 1$ messages. Remark that the hiding of $\overline{\Sigma}_i$ in this update is exactly $\mathcal{M}_{\mathcal{L}_j, \mathcal{L}_i}$. From that $\mathcal{K}_i$ is computed using $\sum_{(\mathcal{L}_j, \mathcal{L}_i) \in E_i} (n_j - 1) + |E_i| = n - 1$ messages.

3 Local verification of distributed protocols on trees

In this section we describe our implementations of Algorithm 1, tailored for checking local linear-time safety and liveness properties, respectively. Each implementation requires to

- Choose a congruence $\equiv$ that preserves the properties of interest.
- Resolve the nondeterminism introduced by $:\equiv$ in lines 4 and 8

Furthermore, $\mathcal{L}' :\equiv \mathcal{L}'$ should be implemented so that $\mathcal{L}'$ is as small as possible.

3.1 Safety

A local safety property of a component $\mathcal{L}_i$ within a system $(\mathcal{L}_1||\dots||\mathcal{L}_n)$ is a property of the (observable) finite traces of $\mathcal{K}_i$, the update of $\mathcal{L}_i$. In order to preserve local safety properties we choose $\equiv$ as (observable) finite trace equivalence, i.e., $\mathcal{L} \equiv \mathcal{L}'$ if and only if $\mathcal{T}^*_{\mathcal{L}} = \mathcal{T}^*_{\mathcal{L}'}$. This equivalence is well known to be a congruence, and in the following we denote it by $\equiv_{\mathcal{T}}$. By Theorem 1, local safety properties of $\mathcal{L}_i$ can be decided by examining the traces of its update $\mathcal{K}_i$ returned by Algorithm 1.

We implement $\mathcal{L}' :\equiv_{\mathcal{T}} \mathcal{L}$ as follows: $\mathcal{L}'$ is the unique τ-free, minimal deterministic LTS equivalent to $\mathcal{L}$. More precisely,

$$\mathcal{L}' := MIN(DET(RED(\mathcal{L}))),$$

where RED, DET, MIN are algorithms for removing τ-transitions, determinizing, and minimizing LTSs, respectively. These algorithms are implemented using standard automata operations (see e.g. [19]).

This particular instantiation of Algorithm 1 closely corresponds to the MPA at the basis of the work presented in [2, 3].

3.2 Liveness

It is well known that defining a local liveness property of $\mathcal{L}_i$ as a property of the (observable) infinite traces of $\mathcal{K}_i$ is inadequate [20]. Consider two systems with sets of infinite traces $\{ab^\omega\}$ and $\{ab^\omega, ac^\omega\}$, respectively. If $\Sigma_i = \{a, b\}$, then in both cases the only infinite trace of $\mathcal{K}_i$ is ab^ω. However, in the first system $\mathcal{L}_i$ satisfies "after a eventually b", while in the second it does not. To solve this problem we keep information about the *divergences* of $\mathcal{L}_i$.

Definition 8. *Given $\mathcal{L} = (\Sigma, S, T, s^0)$ with transitions labelled by τ, a* divergence *of $\mathcal{L}$ is a finite observable trace tr of $\mathcal{L}$ such that there exists an infinite trace tr' of $\mathcal{L}$ verifying $tr'|_\Sigma = tr$. The set of divergences of $\mathcal{L}$ is denoted by $\mathcal{D}_{\mathcal{L}}$.*

Given $(\mathcal{L}_1||\dots||\mathcal{L}_n)$ without τ-transitions, a divergence *of $\mathcal{L}_i$ in $\mathcal{L}$ is a finite observable trace $tr_i \in \mathcal{T}^*_{\mathcal{L}_i}$ such that there exists an infinite observable trace $tr \in \mathcal{T}^\omega_{\mathcal{L}}$ satisfying $tr|_{\Sigma_i} = tr_i$.*

The hiding operation links these two definitions: Given $\mathcal{L} = \mathcal{L}_1||\dots||\mathcal{L}_n$ without τ-transitions, the set of divergences of $\mathcal{L}_i$ in $\mathcal{L}$ is equal to the set $\mathcal{D}_{\mathcal{L}\setminus\overline{\Sigma}_i}$ of divergences of $\mathcal{L} \setminus \overline{\Sigma}_i$.

We define a local liveness property of $\mathcal{L}_i$ as a property of $\mathcal{T}_{\mathcal{K}_i}$ and $\mathcal{D}_{\mathcal{L}\setminus\overline{\Sigma}_i}$, and so we choose: $\mathcal{L} \equiv \mathcal{L}'$ if and only if $\mathcal{T}_{\mathcal{L}} = \mathcal{T}_{\mathcal{L}'}$ and $\mathcal{D}_{\mathcal{L}} = \mathcal{D}_{\mathcal{L}'}$. This equivalence is known to be a congruence for LTSs (see [21] for example). In the following we denote it by $\equiv_{\mathcal{D}}$.

By Theorem 1, local liveness properties of $\mathcal{L}_i$ can be decided by examining the traces of the updated version $\mathcal{K}_i$ obtained by Algorithm 1.

In order to implement $\mathcal{L}' :\equiv_{\mathcal{D}} \mathcal{L}$, we profit from the following fact: for finite LTSs, $\mathcal{T}_{\mathcal{L}} = \mathcal{T}_{\mathcal{L}'}$ iff $\mathcal{T}^*_{\mathcal{L}} = \mathcal{T}^*_{\mathcal{L}'}$, and so $\mathcal{L} \equiv_{\mathcal{D}} \mathcal{L}$ iff $\mathcal{T}^*_{\mathcal{L}} = \mathcal{T}^*_{\mathcal{L}}$ and $\mathcal{D}_{\mathcal{L}} = \mathcal{D}_{\mathcal{L}'}$. This allows us to replace the nondeterministic assignment by a five-step procedure:

$$\mathcal{L}' := HID(MIN(DET(RED(DIV(\mathcal{L}))))),$$

where MIN, DET, and RED are defined as above. For $\mathcal{L} = (\Sigma, S, T, s^0)$, $DIV(\mathcal{L})$ is the LTS $(\Sigma_d, S_d, T_d, s^0_d)$ such that: $\Sigma_d = \Sigma \cup \{\tau'\}$ with $\tau' \notin \Sigma$, $S_d = S$, $T_d = T \cup T^{\tau'}_d$ with $T^{\tau'}_d = \{(s, s) \ : \ \exists s', s \overset{\tau}{\rightsquigarrow}_{\mathcal{L}} s' \overset{\tau}{\rightsquigarrow}_{\mathcal{L}} s'\}$, and $s^0_d = s^0$. Finally, $HID(\mathcal{L})$ is defined as $\mathcal{L} \setminus \{\tau'\}$, where τ' has to be the same as used in DIV. We have:

Theorem 3. *$\mathcal{L} \equiv_{\mathcal{D}} HID(MIN(DET(RED(DIV(\mathcal{L})))))$ for any LTS $\mathcal{L}$.*

Proof. Denote by Σ the set of labels of $\mathcal{L}$. First remark that a finite sequence tr of labels from Σ is a divergence of $\mathcal{L}$ if and only if $tr\tau'^{\omega}$ is an infinite observable trace of $DIV(\mathcal{L})$. This is because (1) tr is a divergence of $\mathcal{L}$ if and only if there exists a path realizing tr in $\mathcal{L}$ and reaching a state from which there exists an infinite path using only τ-transitions, and (2) $DIV(\mathcal{L})$ is an exact copy of $\mathcal{L}$ with the addition of τ'-transitions, all of the form (s, s) for $s \in S_d = S$, and there is a τ'-transition from a state s to itself if and only if there exists in $\mathcal{L}$ an infinite path using only τ-transitions and starting from s.

From this, and because MIN, DET, and RED preserve observable traces, one gets that a finite sequence tr of labels from Σ is a divergence of $\mathcal{L}$ if and only if $tr\tau'^{\omega}$ is an infinite observable trace of $MIN(DET(RED(DIV(\mathcal{L}))))$. Also remark that $MIN(DET(RED(DIV(\mathcal{L}))))$ does not contain any τ-transition.

The remark that HID only replaces τ'-transitions by τ-transitions then allows to conclude that a finite sequence tr of labels from Σ is a divergence of $\mathcal{L}$ if and only if $tr\tau^{\omega}$ is an infinite trace of $HID(MIN(DET(RED(DIV(\mathcal{L})))))$ if and only if tr is a divergence of $HID(MIN(DET(RED(DIV(\mathcal{L})))))$.

4 Experimental evaluation

The approaches described in the previous section have been implemented as an extension of the planner Distoplan [3]. In this section we report on an experimental evaluation of the performances of this implementation on two protocols: a mutual exclusion algorithm on trees [4] and the pragmatic general multicast protocol [5]. All experiments were performed using the same computer with an Intel Core i5 processor and a memory limit set to 4GB.

4.1 Raymond's mutual exclusion protocol.

In [4], Raymond presents a distributed protocol ensuring mutual exclusion for n processes organized as a tree. Processes communicate by rendez-vous, which allows us to model the protocol as a parallel composition of LTSs, one for each process. The unique communication graph is given by the tree. The scaling parameter is the number of processes, which fits well with our approach as the difference between two instances of the protocol is due to the number of LTSs needed to model them, rather than to their sizes.

The protocol can be roughly described as follows. A single token is passed between the processes, a process being allowed to access its critical section only if it owns the token. At any time, each process P_i not holding the token knows which of its neighbours in the tree is closest to the token. In other words P_i knows in which maximal subtree containing exactly one of its neighbours, but not P_i itself, the token currently is. Its requests for the token (and all requests by other processes that it may have to transmit) are sent to this particular neighbour. A more precise description of this protocol is given in Algorithm 2. It describes in a PROMELA-like manner an agent called x. $\mathcal{N}$ denotes the set of neighbours of x in the tree of agents on which the protocol is executed. We consider that x $\in \mathcal{N}$. $\mathcal{Q}(\mathcal{N})$ denotes the set of queues of elements from $\mathcal{N}$. Notice that the same element never appears twice in the queue requestQ. At each step one of three mutually exclusive guarded atomic instruction sequences is executed: (1) re-assignation of the token when x holds it, (2) request for the token, (3) non-deterministic choice between: asking for entering the critical section, receiving a request message from a neighbour, receiving the token from a neighbour, deciding to exit the critical section.

Algorithm 2 An agent for Raymond's mutual exclusion protocol

```
Agent x (holder: $\mathcal{N}$, using: $\{true, false\}$, requestQ: $\mathcal{Q}(\mathcal{N})$, asked: $\{true, false\}$)
holder=x $\wedge$ using=$false$
        $\wedge$ isnotempty(requestQ) $\rightarrow$ holder:=dequeue(requestQ)
                                     asked:=$false$
                                     holder=x $\rightarrow$ using:=$true$
                                     holder$\neq$x $\rightarrow$ !token to holder
holder$\neq$x $\wedge$ asked=$false$
        $\wedge$ isnotempty(requestQ) $\rightarrow$ !request(x) to holder
                                     asked:=$true$
else $\rightarrow$ isnotin(x,requestQ) $\rightarrow$ enqueue(x,requestQ)
       ?request(n) $\rightarrow$ enqueue(n,requestQ)
       ?token $\rightarrow$ holder:=x
       using=$true$ $\rightarrow$ using:=$false$
```

We consider instances of this protocol in which processes form a complete binary tree. The results obtained are presented in Table 1. The leftmost columns give the depth of the binary tree considered (Depth) and the corresponding number of processes (Processes). For each depth the column **Traces** reports the time (in seconds) needed for the following tasks: (1) run Algorithm 1 to obtain the updated versions of all the processes with respect to trace equivalence $\equiv_{\mathcal{T}}$ (subcolumn MPA); (2) same but computing only $n-1$ messages in order to obtain the updated version of the root only (subcolumn OneWay); (3) verify a simple local property from the updated root (subcolumn Verification). Column

Divergences gives the times required by analogous tasks, but with respect to divergence equivalence $\equiv_{\mathcal{D}}$.

Table 1. Results for the analysis of Raymond's mutual exclusion protocol on complete binary trees. Times are in seconds.

Depth	**Processes**	**Traces**			**Divergences**		
		MPA	**OneWay**	**Verification**	**MPA**	**OneWay**	**Verification**
2	3	0.12	0.15	<0.01	0.14	0.13	<0.01
3	7	1.41	1.23	<0.01	2.07	1.88	<0.01
4	15	2.36	2.20	<0.01	4.58	4.36	<0.01
5	31	5.29	4.67	<0.01	10.44	9.67	<0.01
6	63	10.62	9.63	<0.01	21.81	20.27	<0.01
7	127	21.94	19.70	<0.01	44.86	41.55	<0.01

We check a safety (in the case of $\equiv_{\mathcal{T}}$) and a liveness (in the case of $\equiv_{\mathcal{D}}$) property in order to compare our MPA with an algorithm that constructs the state space, for which we use Spin. The local safety property verified for the root processes is: "It is not possible to request the token twice without receiving it in between". Remark that, it is also expressible as the following global property: "It is not possible for the root process to request the token twice without receiving it in between". We verify this property by checking emptiness of the product of the automaton φ on the left of Figure 5 (representing the negation of the property) and the (projection onto the labels of φ of the) updated version of the root process obtained by Algorithm 1 (in which all states are considered accepting). The local liveness property verified for the root processes is: "The token is received in finite time after any request". Similarly, the property is verified by checking emptiness of the product of the Büchi automaton φ' on the right of Figure 5 (representing the negation of the property) with the (projection on the labels of φ' of the) updated version of the root process obtained by Algorithm 1 (in which all states are considered accepting).

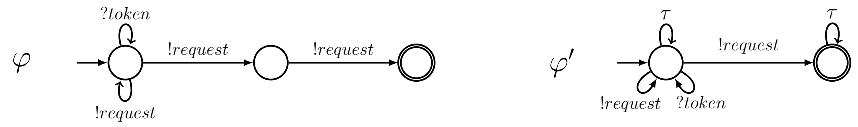

Fig. 5. Two properties to be checked on Raymond's mutual exclusion protocol. $!request$ is a general token request action and $?token$ is a general token reception action.

Analysis of the results. As expected, the approach scales very well with the number of components: at each depth the number of components almost doubles while the time spent for computing the updates of the components is slightly more than doubling. We compare it with a verification of the same properties using Spin [22]. Since the system is highly concurrent, we use Spin with the partial order reduction optimization. Spin outperforms our approach for the complete binary tree of depth 2 (it needs less than 0.01 seconds to verify the properties). For trees of depth 3 or greater, however, Spin runs out of memory (memory limit being 4GB). Notice that, since the properties we check are true, Spin needs a full state-space exploration to verify them, while our MPA prevents this. We also remark that the time needed by our MPA is almost entirely spent in computing the updates of the components, and so the additional cost of verifying other properties after the first one is small, since the previously computed updates can be re-used. Finally, observe that, in this example, the difference in time spent between the standard application of Algorithm 1 (computing all updates) and the case where only the update of the root is computed (so only half of the messages are constructed) is not significant. This can be explained by the fact that – being the initial owner of the token – the root imposes more constraints to the system than the other components. So the messages from the root are potentially much simpler to compute than the messages to the root.

4.2 The pragmatic general multicast protocol.

[5] describes the pragmatic general multicast protocol (PGM), a reliable distributed protocol for distributing information from multiple senders to multiple receivers in a network, designed to minimize the load of the network due to acknowledgement messages and retransmissions of lost messages. We consider the specific version of this very generic protocol given in Algorithm 3 (which is almost the one described in [17]): a unique source sends information to multiple receivers in a network organized as a tree. Each process is described in a Promela-like manner and consists of a single loop in which a non-deterministic choice is done at each step between several guarded atomic instruction sequences.

The source can receive a negative acknowledgement $nak(nr)$ for some data, in this case it sends back a confirmation $ncf(nr)$ and, if the data nr is still within range of its window it adds it to the set $recNak$ of data to be re-sent. If some data nr is in the set $recNak$ and still within range of the window this data can be re-sent as a message $rdata(nr, txWTr)$. And, if no data needs to be re-sent, a new data can be sent as a message $odata(nr, txWTr)$ and the window may be moved. Any network element can receive negative acknowledgements and propagate them above itself in the tree. It can also transmit data below itself in the tree. Finally, it may generate new negative acknowledgements while no confirmation have been received for them. A receiver can receive data, and, when it allows it to deduce that some data are missing (by looking at the previously received data and because of the fact that the data are consecutive integers) it can send negative acknowledgements for these data.

As before we represent each process by an LTS. However, communications are no longer by rendez-vous but use messages sent through bidirectional channels (which can lose messages). Each channel is thus also modelled as an LTS. The unique communication graph of such a system is a tree.

Algorithm 3 PGM: source, network elements, and receivers

```
Source (data: $\mathbb{N}$, winSize: $\mathbb{N}$, txWTr: $\mathbb{N}$, recNak: $2^{\mathbb{N}}$)
?nak(nr) $\rightarrow$ !ncf(nr)
           $txWTr < nr \rightarrow$ recNak := add(recNak,nr)
isin(nr,recNak) $\rightarrow nr > txWTr \rightarrow$ !rdata(nr,txWTr)
                recNak := remove(recNak,nr)
lenght(recNak) $= 0 \rightarrow$ !odata(data,txWTr)
                    data := data + 1
                    $odata > winSize + txWTr \rightarrow$ txWTr := txWTr + winSize

Network element (setRepair: $2^{\mathbb{N}}$)
?nak(nr) $\rightarrow$ setRepair := add(setRepair,nr)
           isnotin(setRepair,nr) $\rightarrow$ !nak(nr) upwards
           !ncf(nr) downwards
?rdata(nr,txWTr) $\rightarrow$ setRepair := remove(setRepair,nr)
                  !rdata(nr,txWTr) downwards
?ncf(nr) $\rightarrow$ setRepair := remove(setRepair,nr)
?odata(nr,txWTr) $\rightarrow$ !odata(nr,txWTr) downwards
isin(nr,setRepair) $\rightarrow$ !nak(nr) upwards

Receiver (rxWTr: $\mathbb{N}$, setNr: $2^{\mathbb{N}}$, setMissing: $2^{\mathbb{N}}$)
?odata(nr,txWTr) $\wedge$ $rxWTr < nr \rightarrow rxWTr < txWTr \rightarrow$ rxWTr := txWTr
                                setNr := add(setNr,nr)
                                for all ($rxWTr < i < nr$ $\wedge$ isnotin(i,setNr))
                                    setMissing := add(setMissing,i)
                                setMissing := remove(setMissing,nr)
?rdata(nr,txWTr) $\wedge$ $rxWTr < nr \rightarrow rxWTr < txWTr \rightarrow$ rxWTr := txWTr
                                setNr := add(setNr,nr)
                                setMissing := remove(setMissing,nr)
isin(nr,setMissing) $\rightarrow nr > rxWTr \rightarrow$ !nak(nr)
                     $nr \leq rxWTr$ setMissing := remove(setMissing,nr)
```

In our experiments we considered two possible topologies for the systems: lines and complete binary trees. In each of these cases we considered instances of the protocol with increasing numbers of processes. We also made other parameters vary: the number of different data to be sent by the source (two or three) and the capacity of the channels (one or two messages). The initial value of *data* for the source is set to 1 and the value of *winSize* is set to anything higher than the number of different data to be sent. All other integer parameters are initialized to 0 and all sets are initially empty.

Figure 6 presents the update of a leaf as obtained by Algorithm 1 (using $\equiv_{\mathcal{T}}$ as congruence) when ran on a complete binary tree of depth five with two data to be sent and channels of capacity one. It is interesting to notice that just by looking at this LTS, a behaviour (corresponding to the path with larger labels in the figure) that may not be directly anticipated from the description of the protocol can be remarked: it is possible for the leaf to deduce that the second (and last) data will never be received. This can in fact be explained by the possible losses of messages. For sure $ncf(1)$ has been sent (by the source after reception of $nak(1)$ from another leaf) after $odata(2,0)$. So, each channel between the source and the leaf represented in Figure 6 has contained $ncf(1)$ at some time, and if it has contained $odata(2,0)$ at some time it was before $ncf(1)$. The only explanation for receiving $ncf(1)$ before $odata(2,0)$ is thus a loss of $odata(2,0)$ at some channel.

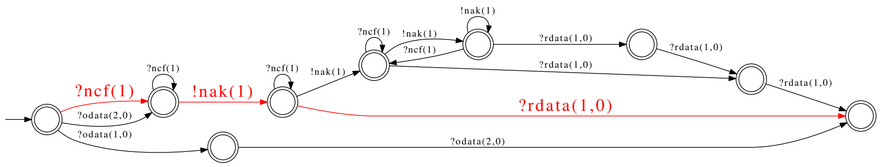

Fig. 6. PGM protocol: update of a leaf in a complete binary tree of depth five when the number of different data to be sent is two and the channels capacity is one.

Table 2 gives the results obtained for the PGM protocols on lines, in the case where the source can only send two different data and the channels are of capacity one. Results are organized as before. The only difference are in the Basic and MPA columns. The Basic column presents the times obtained by running Algorithm 1 while the MPA columns present the times obtained by running Algorithm 4. This algorithm is a variation of Algorithm 1 where messages that “cross” at some edge of the communication graph are not completely independent: the constraints imposed by the first to be computed are used to compute the second.

Algorithm 4 Variation of Algorithm 1

```
Input: an interaction graph $\mathcal{G} = (V, E)$ with $V = \{\mathcal{L}_1, \dots, \mathcal{L}_n\}$
M $\leftarrow E$
while M $\neq \emptyset$ do
    choose $(\mathcal{L}_i, \mathcal{L}_j) \in$ M such that $(\mathcal{L}_k, \mathcal{L}_i) \notin$ M for every $k \neq j$
    $\mathcal{L}_i :\equiv \mathcal{L}_i \mid\mid (\mid\mid_{\substack{k \neq j, \\ (\mathcal{L}_k, \mathcal{L}_i) \in E}} \mathcal{M}_{k,i})$
    $\mathcal{M}_{i,j} :\equiv \mathcal{L}_i \setminus \overline{\Sigma}_j$
    remove $(\mathcal{L}_i, \mathcal{L}_j)$ from M
end while
for all $i \in V$ do
    $\mathcal{K}_i :\equiv \mathcal{L}_i \mid\mid (\mid\mid_{(\mathcal{L}_j, \mathcal{L}_i) \in E} \mathcal{M}_{j,i})$
end for
```

The local safety and liveness properties we check at the source are the following: “The last data can only be sent once” (its negation is represented by the FSA φ on the left of Figure 7) and "The first data is always sent at least once" (its negation is represented by the NBA φ' on the right of Figure 7). The verification process is the same as above.

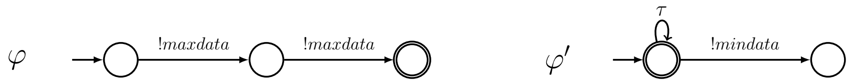

Fig. 7. Two properties to be checked on PGM. $!maxdata$ (respectively $!mindata$) represents any sending of the last data (respectively the first data) using an odata or an rdata message.

Table 3 gives the other results obtained for the PGM protocols on lines. Table 4 gives the results obtained for the PGM protocols on complete binary trees. They only report the running times of Algorithm 4 as the verification of the properties considered still always requires less than 0.01 seconds.

Analysis of the results. SPIN can deal with lines of length 5 within 20 seconds but cannot handle larger lines nor binary trees without running out of the 4GB of memory allowed to it. In addition to what has been noticed in the case of Raymond's mutual exclusion protocol, it appears that using Algorithm 4 instead of Algorithm 1 can significantly reduce running times. This is due to the fact that some constraints are taken into account earlier, and so some messages can be simplified. Using this version of our approach is not always that efficient however. In the case of Raymond's mutual exclusion protocol, for example, it almost does not reduce running times. The comparison of the different numbers of data and capacities of channels also shows that, if our approach scales well with the number of components of the system to analyse, it is more sensitive to increases of the sizes of these components.

Conclusion

We have presented message-passing algorithms for the verification of local properties of distributed protocols. The components of the protocol must have a tree-shaped communication structure. The MPAs compute for each component $\mathcal{L}_i$ an LTS equivalent to the result of hiding in the full LTS (the LTS of the full protocol) all actions not appearing in $\mathcal{L}_i$. We have shown that the MPAs can be instantiated with different equivalence notions, in particular trace equivalence and divergence equivalence. We have evaluated the algorithms on two well-known protocols, and shown that for several important properties they scale very well, in particular much better than a generic search-based model-checker like SPIN.

The properties we have used for our comparison with SPIN were true properties, i.e., properties that hold for the protocol. In fact, for false properties SPIN often outperforms our approach, possibly due to the existence of a relatively

Table 2. Analysis of PGM (with two different data and channels of capacity one) on lines. Times are in seconds.

Processes	**Traces**				**Divergences**		
	Basic	**MPA**	**OneWay**	**Verification**	**MPA**	**OneWay**	**Verification**
5	7.79	0.08	0.03	<0.01	0.11	0.08	<0.01
10	20.27	0.13	0.08	<0.01	0.16	0.13	<0.01
15	32.76	0.19	0.15	<0.01	0.22	0.20	<0.01
20	41.99	0.23	0.16	<0.01	0.26	0.20	<0.01
25	53.14	0.26	0.21	<0.01	0.31	0.24	<0.01
30	67.50	0.30	0.25	<0.01	0.37	0.27	<0.01
35	77.32	0.35	0.29	<0.01	0.43	0.34	<0.01
40	89.95	0.40	0.32	<0.01	0.49	0.36	<0.01
45	101.25	0.46	0.36	<0.01	0.57	0.40	<0.01
50	113.60	0.50	0.40	<0.01	0.60	0.44	<0.01

Table 3. Analysis of PGM on lines using Algorithm 4. Different numbers of data to be sent (d) and different sizes of channels (c) are considered. Times are in seconds.

Processes	**Traces**		**Divergences**	
	d=2, c=2	**d=3, c=1**	**d=2, c=2**	**d=3, c=1**
5	10.71	10.63	15.37	13.26
10	19.19	12.60	28.94	18.00
15	27.24	14.56	41.77	22.21
20	35.53	16.46	55.77	26.80
25	43.66	18.24	68.40	30.95
30	52.14	20.66	81.43	35.36
35	60.16	22.64	95.39	39.82
40	68.78	24.80	109.17	44.49
45	77.00	26.66	122.57	48.56
50	85.12	29.01	136.60	53.27

Table 4. Analysis of PGM on complete binary trees using Algorithm 4. Different numbers of data to be sent (d) and different sizes of channels (c) are considered. Times are in seconds.

Depth	**Proc.**	**Traces**			**Divergences**		
		d=2, c=1	**d=2, c=2**	**d=3, c=1**	**d=2, c=1**	**d=2, c=2**	**d=3, c=1**
3	7	0.85	26.26	59.30	1.41	33.96	93.02
4	15	1.58	56.10	114.05	1.60	72.89	156.93
5	31	2.48	113.82	235.32	2.93	153.47	316.63
6	63	5.06	231.27	472.19	5.73	310.28	641.13
7	127	10.24	474.57	979.85	12.10	625.61	1582.23

large number of counterexamples, which allows Spin to quickly find one. This suggests to run Spin and a suitable MPA in parallel.

Future work will explore how to instantiate the MPAs with equivalence relations sensitive to deadlocks or partial deadlocks.

References

1. E. Fabre. *Bayesian Networks of Dynamic Systems*. Habilitation à diriger des recherches, Université de Rennes1, 2007.
2. E. Fabre and L. Jezequel. Distributed optimal planning: an approach by weighted automata calculus. In *CDC*, pages 211–216, 2009.
3. E. Fabre, L. Jezequel, P. Haslum, and S. Thiébaux. Cost-optimal factored planning: Promises and pitfalls. In *ICAPS*, pages 65–72, 2010.
4. K. Raymond. A tree-based algorithm for distributed mutual exclusion. *TCS*, 7(1):61–77, 1989.
5. T. Speakman et al. PGM reliable transport protocol specification. RFC 3208 (Experimental) of the IETF, 2001.
6. J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu. Learning assumptions for compositional verification. In *TACAS*, pages 331–346, 2003.
7. C. Flanagan and S. Qadeer. Thread-modular model checking. In *SPIN*, pages 213–224, 2003.
8. S. Graf and B. Steffen. Compositional minimization of finite state systems. In *CAV*, pages 186–196, 1990.
9. O. Grumberg and D. E. Long. Model checking and modular verification. *TOPLAS*, 16(3):843–871, 1994.
10. A. W. Roscoe, P. H. B. Gardiner, M. H. Goldsmith, J. R. Hullance, D. M. Jackson, and J. B. Scattergood. Hierarchical compression for model-checking CSP or how to check 10^{20} dining philosophers for deadlock. In *TACAS*, pages 133–152, 1995.
11. FRD2 user manual, 2009.
12. R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *TOPLAS*, 15(1):36–72, 1993.
13. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *STTT*, 15(2):89–107, 2013.
14. P. A. Abdulla. Regular model checking. *STTT*, 14(2):109–118, 2012.
15. B. Bérard, P. Bouyer, and A. Petit. Analysing the PGM protocol with UPPAAL. *International journal of production research*, 42(14):2773–2791, 2004.
16. M. Boyer and M. Sighireanu. Synthesis and verification of constraints in the PGM protocol. In *FME*, pages 264–281. Springer, 2003.
17. J. Esparza and M. Maidl. Simple representative instantiations for multicast protocols. In *TACAS*, pages 128–143, 2003.
18. H. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *STC*, pages 226–234, 1993.
19. J. Sakarovitch. *Éléments de théorie des automates*. Vuibert, 2003.
20. S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Seminar on Concurrency*, pages 281–305, 1984.
21. A. Valmari. All linear-time congruences for finite LTSs and familiar operators. In *ACSD*, 2012.
22. G. Holzmann. *The SPIN model checker: primer and reference manual*. Addison-Wesley Professional, 2003.