Verification of parameterized communicating automata via split-width

Marie Fortin, Paul Gastin

Verification of parameterized communicating automata via split-width

Marie Fortin and Paul Gastin

LSV, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France
mfortin@ens-cachan.fr, gastin@lsv.ens-cachan.fr

Abstract. We study verification problems for distributed systems communicating via unbounded FIFO channels. The number of processes of the system as well as the communication topology are not fixed a priori. Systems are given by parameterized communicating automata (PCAs) which can be run on any communication topology of bounded degree, with arbitrarily many processes. Such systems are Turing powerful so we concentrate on under-approximate verification. We extend the notion of split-width to behaviors of PCAs. We show that emptiness, reachability and model-checking problems of PCAs are decidable when restricted to behaviors of bounded split-width. Reachability and emptiness are EXPTIME-complete, but only polynomial in the size of the PCA. We also describe several concrete classes of bounded split-width, for which we prove similar results.

Keywords: Parameterized distributed systems, Model checking, Split-width, Message sequence charts

1 Introduction

We study verification problems for parameterized communicating automata (PCAs), which model distributed systems consisting of arbitrarily many identical processes, distributed on some communication topology. Each process runs a copy of the same finite automaton, that can send and receive messages from other processes through FIFO channels. Though the system may contain unboundedly many processes, we assume that each process may only communicate with a bounded number of other processes.

PCAs were introduced in [5] to study logical characterizations of parameterized systems. They extend communicating finite-state machines [8]. While the latter assume a fixed and known communication topology, a PCA can be run on *any* communication topology of bounded degree. Communicating finite-state machines are already Turing equivalent, and thus their verification is undecidable. The fact that PCAs can be run on arbitrarily large topologies induces other sources of undecidability. For instance, a Turing machine can be simulated on grid topologies by a PCA *performing a bounded number of actions on each process*. Thus, some restrictions on the topologies are necessary. Yet, even when

fixing a simple class of topologies such as pipelines, and imposing rendez-vous synchronization, reachability of PCAs is undecidable [6].

In order to regain decidability, we focus on under-approximate verification. The idea is to restrict the verification problems to meaningful classes $\mathcal{C}$ of behaviors. Typically, we are interested in the following problem: given a PCA $\mathcal{S}$, is there a topology $\mathcal{T}$ and a behavior in $\mathcal{C}$ over $\mathcal{T}$ on which $\mathcal{S}$ reaches some local/global state? Our aim is to study under-approximation classes $\mathcal{C}$ for which verification problems of PCAs becomes decidable. Even when we cannot cover all behaviors of a system with a decidable class, under-approximate verification is still very useful to detect bugs. Usually, the classes $\mathcal{C}_i$ are parameterized and cover more and more behaviors when the parameter i increases.

The behaviors of PCAs are described by message sequence charts (MSC) [17], that is, graphs describing the communications between the different processes. Each node of the graph corresponds to an action performed by some process (sending or receiving a message), and the dependencies between the different actions are indicated by the edges of the graph. For model-checking, the specifications of our systems are typically given as monadic second order logic (MSO) or propositional dynamic logic (PDL) formulas over MSCs.

It is known that for any MSO definable class of bounded degree graphs, having a decidable MSO theory is equivalent to having bounded tree-width [9]. This applies to classes of MSCs, and characterizes decidability of MSO model-checking. However, showing a bound on tree-width is in general difficult. In the case of MSCs over a *fixed* architecture, an alternative notion called split-width has been introduced [1, 2]. Split-width is equivalent to tree-width on MSCs, but easier to use. It provides means to define uniform decision procedures, applying to many well-studied restrictions of communicating finite-state machines [10].

Following this approach, we generalize the definition of split-width, and we extend existing results over fixed architectures to the parameterized case. The idea of split-width is to decompose an MSC into atomic pieces, that is, pairs of matching send and receive events. This is done using two operations: splitting some edges between consecutive events of a same process, and dividing the resulting graph into disjoint components. Intuitively, an MSC has bounded split-width when this can be done while splitting a bounded number of edges at each step, on a bounded number of processes.

We show that emptiness and reachability of PCAs restricted to MSCs of bounded split-width are decidable. These problems are EXPTIME-complete but only polynomial in the size of the PCA. Our decision procedures are based on an interpretation of MSCs of bounded split-width into binary trees, and reductions to tree automata verification problems.

In the extended version [**?**] we also prove that model-checking restricted to bounded split-width is decidable. For MSO specifications, it has non-elementary complexity. However, the under-approximate model-checking is respectively EXPTIME-complete and 2-EXPTIME complete for CPDL (PDL with converse) and ICPDL (PDL with converse and intersection). In all cases, the problem is still polynomial in the size of the PCA.

Further, we give several examples of concrete classes of MSCs with bounded split-width, for which we show similar decidability and complexity results. In particular, we consider existentially bounded or context bounded behaviors. But our approach based on split-width is generic since it can be easily adapted to other under-approximation classes, and it relaxes the assumptions of [6, 7] which only considered rendez-vous syncronization and pipeline, ring, or tree topologies. In the case of context bounds, we show that it is sufficient to assume that the communication topology has bounded tree-width. In the general case of bounded split-width, we do not assume any restriction on the topology (apart from being of bounded degree). But it should be noticed that a bound on split-width already implies a bound on the tree-width of the communication topology.

Related work. Various models of parameterized systems have been considered in the literature. In several cases, the assumptions of the model are sufficient to get decidability results without additional restrictions on the behaviors. Examples include token-passing systems [3, 15], models with a global store without locking [14], message-passing systems communicating synchronously via broadcast [12, 13], or communicating via rendez-vous [4]. An important distinction between the latter and PCAs is that PCAs use point-to-point communication: a process can distinguish between its neighbors, and specify the recipients of its messages. This makes the model more expressive, but also leads to undecidability.

The emptiness and model-checking problems for PCAs have been considered in [6] and [7], respectively. Both papers assume rendez-vous synchronization, and a fixed class of topologies: pipelines, rings, or trees. Several notions of *contexts* are introduced, and decision procedures are described for the corresponding classes of context-bounded behaviors.

Communicating finite-state machines (over fixed topologies) have been more extensively studied, and several restrictions are known to bring back decidability. Our work generalizes some of them to the parameterized setting, namely, context bounds (introduced in [18] for multi-stack concurrent systems), existential bounds [16], and bounded split-width [2].

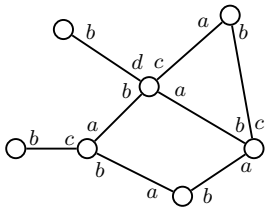
Split-width was first introduced for multi-pushdown systems [11], and then generalized to communicating multi-pushdown systems [2].

Outline. In Section 2, we define topologies, PCAs, and MSCs. In Section 3, we introduce split-width. In Section 4, we give several examples of classes of bounded split-width, and state our results for the reachability problems of those classes. In Section 5, we present in more details the decision procedures leading to these results. In Section 6, we briefly present how they can be extended to decide model-checking problems, and discuss possible extensions of our model.

2 Parameterized communicating automata

We describe our formal model for communicating systems: we introduce topologies, MSCs, and parameterized communicating automata. Our definitions are taken from [5], except that we abstract away idle processes.

Topologies. We model distributed systems consisting of an unbounded number of processes. Each process is equipped with a bounded number of interfaces, through which it can communicate with other processes. A topology is a graph, describing the connections between the different processes (each of which is represented by a node in the graph, see example on the right). Throughout the paper, we assume a *fixed* nonempty finite set $\mathcal{N} = \{a, b, \ldots\}$ of *interface names* (or, simply, *interfaces*).



Definition 1. *A* topology *over $\mathcal{N}$ is a pair $\mathcal{T} = (P, \longmapsto)$ where P is the nonempty finite set of* processes *and $\longmapsto \subseteq P \times \mathcal{N} \times \mathcal{N} \times P$ is the* edge relation. *We write $p \overset{a\ b}{\longmapsto} q$ for $(p, a, b, q) \in \longmapsto$, which means that the a-interface of p is connected the b-interface of q. We require that, whenever $p \overset{a\ b}{\longmapsto} q$, the following hold:*

(a) $p \neq q$ *(there are no self loops),*
(b) $q \overset{b\ a}{\longmapsto} p$ *(adjacent processes are mutually connected), and*
(c) *for all $a', b' \in \mathcal{N}$ and $q' \in P$ such that $p \overset{a'\ b'}{\longmapsto} q'$, we have $a = a'$ iff $q = q'$ (an interface is connected to at most one process, and two distinct interfaces are connected to distinct processes).*

Message sequence charts. The possible behaviors of our systems are depicted as message sequence charts. A message sequence chart consists of a set of processes, and, for each process, of a sequence of events. Each event corresponds to an action of the process (sending or receiving a message through a given interface), and matching send and receive events are connected by a message edge. Events are labeled with elements of $\Sigma \overset{\text{def}}{=} \{a? \mid a \in \mathcal{N}\} \cup \{a! \mid a \in \mathcal{N}\}$, according to the type of action they execute.

Definition 2. *A* pre-message sequence chart (pre-MSC) *over the set of interfaces $\mathcal{N}$ is a tuple $M = (P, E, \rightarrow, \lhd, \pi, \lambda)$, where*

- *P and E are nonempty finite sets of* processes *and* events*, respectively.*
- *$\pi : E \to P$ is a surjective map determining the location of an event. For $p \in P$, we let $E_p \overset{\text{def}}{=} \{e \in E \mid \pi(e) = p\}$.*
- *$\lambda : E \to \Sigma$ associates with each event the type of action that it executes. We let $E_? \overset{\text{def}}{=} \{e \in E \mid \exists a \in \mathcal{N}.\lambda(e) = a?\}$, and $E_! \overset{\text{def}}{=} \{e \in E \mid \exists a \in \mathcal{N}.\lambda(e) = a!\}$.*
- *$\rightarrow$ is a union $\bigcup_{p \in P} \rightarrow_p$, where each $\rightarrow_p \subseteq E_p \times E_p$ is the direct-successor relation of some total order on E_p.*
- *$\lhd \subseteq E_! \times E_?$ defines a bijection from $E_!$ to $E_?$. Moreover, for each $(e, f) \in \lhd$, $\pi(e) \neq \pi(f)$.*

Given such a pre-MSC, we define $\mathcal{T}_M \overset{\text{def}}{=} (P, \{(p, a, b, q) \in P \times \mathcal{N}^2 \times P \mid \exists (e, f) \in E_p \times E_q.\, (e \lhd f \wedge \lambda(e) = a! \wedge \lambda(f) = b?)$ or $(f \lhd e \wedge \lambda(e) = a? \wedge \lambda(f) = b!)\})$.

Not all pre-MSCs correspond to actual behaviors of systems. To define MSCs, we additionally require that the events are coherently ordered, and that communications are compatible with some topology and follow a FIFO policy.

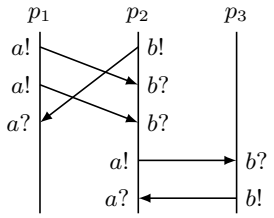


Fig. 1. An MSC with 3 processes

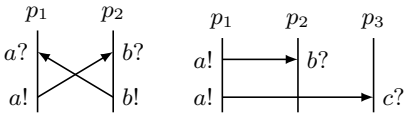


Fig. 2. pre-MSCs that are not MSCs

Definition 3. *A* message sequence chart (MSC) *over $\mathcal{N}$ is a pre-MSC $M = (P, E, \rightarrow, \lhd, \pi, \lambda)$ such that the relation $\leq \stackrel{\text{def}}{=} (\rightarrow \cup \lhd)^*$ is a partial order, and:*

- *$\mathcal{T}_M$ as defined above is a topology, called the* observable topology *of M.*
- *For all $e_1 \lhd e_2$, $f_1 \lhd f_2$ s.t. $\pi(e_i) = \pi(f_i)$, we have $e_1 \leq f_1$ iff $e_2 \leq f_2$ (FIFO).*

An MSC M is called *compatible* with a topology $\mathcal{T}$ when $\mathcal{T}_M$ is a subgraph of $\mathcal{T}$. Intuitively, an MSC is compatible with a topology $\mathcal{T}$ when it can be interpreted as a behavior over $\mathcal{T}$, in which some processes may be inactive or may not use all their interfaces. We denote by MSC the set of all MSCs over $\mathcal{N}$, and by $\mathsf{MSC}_{\mathcal{T}}$ the set of all MSCs compatible with a topology $\mathcal{T}$.

Example 4. An example MSC is depicted in Figure 1. The vertical lines represent the succession of events on a given process, and $\lhd$-edges are depicted by arrows.

Parameterized communicating automata. The idea is that each process of a topology runs one and the same automaton, whose transitions are labeled with actions of the form $a!m$, which emits a message m through interface a, or $a?m$, which receives m from interface a. The acceptance condition of a parameterized communicating automaton is given as a boolean combination of conditions of the form "at least n processes end in state s", written $\langle \#(s) \geq n \rangle$.

Definition 5. *A* parameterized communicating automaton (PCA) *over $\mathcal{N}$ is a tuple $\mathcal{S} = (S, \iota, Msg, \Delta, F)$ where S is the finite set of* states*, $\iota \in S$ is the* initial state*, Msg is a nonempty finite set of* messages*, $\Delta \subseteq S \times (\Sigma \times Msg) \times S$ is the* transition relation*, and F is the* acceptance condition*, a finite boolean combination of statements of the form $\langle \#(s) \geq n \rangle$, with $s \in S$ and $n \in \mathbb{N}$.*

The *size* $|F|$ of the acceptance condition of $\mathcal{S}$ is defined as the length of its encoding, where all integer values are written in binary.

Let $M = (P, E, \rightarrow, \lhd, \pi, \lambda)$ be an MSC. A run of $\mathcal{S}$ on M will be a mapping $\rho : E \rightarrow S$ satisfying some requirements. Intuitively, $\rho(e)$ is the local state of $\pi(e)$ after executing e. To determine when ρ is a run, we define another mapping, $\rho^- : E \rightarrow S$, denoting the source state of a transition: whenever $f \rightarrow e$, we let $\rho^-(e) = \rho(f)$; moreover, if e is $\rightarrow$-minimal, we let $\rho^-(e) = \iota$. With this, we say that ρ is a run of $\mathcal{S}$ on M if, for all $(e, f) \in \lhd$, there is a message $m \in Msg$ such that $(\rho^-(e), (\lambda(e), m), \rho(e)) \in \Delta$, and $(\rho^-(f), (\lambda(f), m), \rho(f)) \in \Delta$. A run

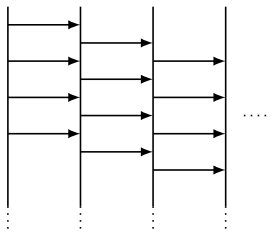


Fig. 3. Undecidability with pipelines and rendez-vous synchronization

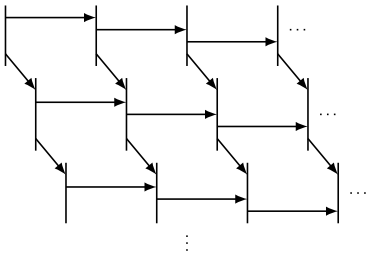


Fig. 4. Undecidability with a bounded number of actions on each process, using grid topologies

ρ is accepting if it satisfies the acceptance condition. In particular, ρ satisfies $\langle \#(s) \geq n \rangle$ when $|\{e \in E \mid e \text{ is } \rightarrow\text{-maximal and } \rho(e) = s\}| \geq n$.

The set of MSCs that allow for an accepting run is denoted by $L(\mathcal{S})$. Given a topology $\mathcal{T}$, we let $L_{\mathcal{T}}(\mathcal{S}) = L(\mathcal{S}) \cap \mathsf{MSC}_{\mathcal{T}}$.

Remark 6. We could add labels from a finite alphabet to the events of MSCs and to the transitions of PCAs. Such labels can be handled similarly to the λ-labeling, and all our results can easily be adapted to this setting. Similarly, allowing internal transitions for PCAs would not add any technical difficulties.

Verification problems. The *non-emptiness problem* asks, given a PCA $\mathcal{S}$, whether its language $L(\mathcal{S})$ is non empty; or in other words, whether $L_{\mathcal{T}}(\mathcal{S}) \neq \varnothing$ for some topology $\mathcal{T}$. The *(local) reachability problem* asks, given a PCA $\mathcal{S}$ and a state s of $\mathcal{S}$, whether there exists a run of $\mathcal{S}$ in which some process reaches the state s. It can be seen as a special instance of the non-emptiness problem, by modifying the acceptance condition of $\mathcal{S}$ to $\langle \#(s) \geq 1 \rangle$.

Notice that the non-emptiness problem $L_{\mathcal{T}}(\mathcal{S}) \neq \varnothing$ for a *fixed* topology $\mathcal{T}$ is already undecidable, since two finite automata connected by two queues can easily simulate a Turing machine. Furthermore, many decidable restrictions over fixed topologies remain undecidable in the parameterized case: for instance, bounding the number of contexts, or even of actions, performed by each process, or imposing rendez-vous synchronization (even when restricted to pipeline topologies [6]). The idea is that the unbounded number of processes can be used to construct a PCA whose behaviors are grid-like MSCs of arbitrary height and width (see Figures 3 and 4). It is then easy to encode runs of a Turing machine: the unbounded horizontal direction encodes the tape of the machine, and the vertical direction its evolution with time.

In the remaining of the paper, we will study several decidable underapproximations of the problem. For a family $(\mathcal{C}_i)_i$ of classes of MSCs (for the concrete families studied in Section 4, the index i is a tuple of integers), we define the problem $\mathcal{C}$-NonEmptiness as follows (and similarly, $\mathcal{C}$-Reachability):

Input: i in unary, a set of interfaces $\mathcal{N}$, a PCA $\mathcal{S}$ over $\mathcal{N}$.
Question: $L(\mathcal{S}) \cap \mathcal{C}_i \neq \varnothing$?

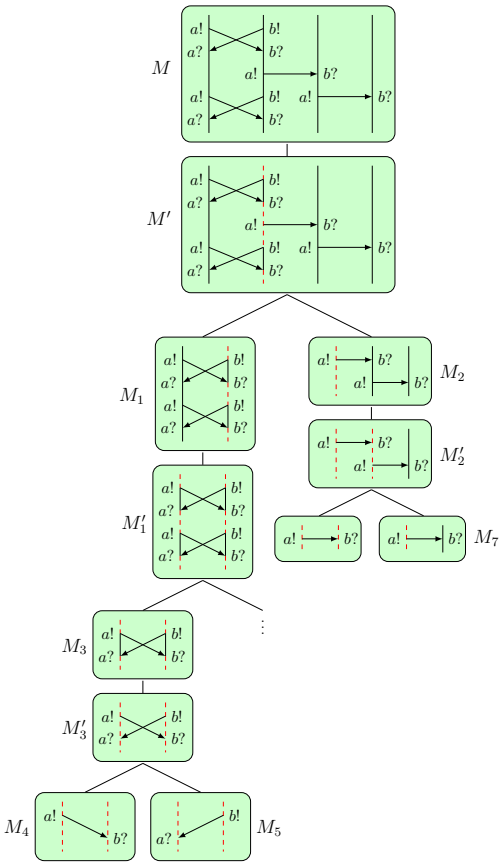


Fig. 5. A split decomposition of width 4

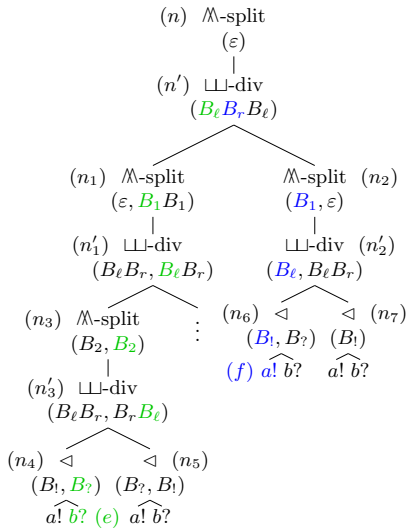


Fig. 6. 4-DST associated with the decomposition of Figure 5

3 Split-width

In this section, we introduce the notion of split-width, and state our decidability result for MSCs of bounded split-width. The main motivation behind split-width is that it allows to design generic decision procedures that apply to many under-approximation classes, instead of having to develop for each class a specific decision procedure with its complexity. Several examples of classes of MSCs that are captured with split-width will be given in Section 4.

The idea of split-width is to decompose an MSC into *atomic* pieces consisting of a pair of matching send and receive events, using two operations: *split* (removing some process edges of the MSC), and *divide* (separating the resulting graph into two independent parts). This is described below as a two-player game. First, we introduce the notion of split MSC (an MSC missing some of its process edges).

Definition 7. *A* split (pre-)MSC *is a tuple* $M = (P, O, E, \rightarrow, \dashrightarrow, \lhd, \pi, \lambda)$, *where* $(P, E, \rightarrow \cup \dashrightarrow, \lhd, \pi, \lambda)$ *is a (pre-)MSC,* $\dashrightarrow \cap \rightarrow = \varnothing$, $O \subseteq P$ *is the set of* open processes *of* M, *and every split process is open, i.e.,* $\{p \in P \mid (E_p)^2 \cap \dashrightarrow \neq \varnothing\} \subseteq O$.

The $\dashrightarrow$ edges are called *elastic*, and the $\rightarrow$ edges *rigid*. Processes of $P \setminus O$ are called *closed*. The intuition is that an open process of M is a process that may be missing some of its events or process edges, and an elastic edge represents a missing part of a process between two events. Any MSC can be seen as a split MSC, by taking O and $\dashrightarrow$ empty.

A *block* of a split (pre-)MSC M is a maximal connected component of $(E, \rightarrow)$ on some *open* process. In particular, M has exactly $|O| + |\dashrightarrow|$ blocks.

Several split MSCs are depicted in Figure 5. Elastic edges are represented by red dotted lines. Open processes are indicated by dotted lines at their extremities. For instance, M' has one open process with 3 blocks, and M'_2 has 2 open processes, with resp. 1 and 2 blocks.

We call *splitting* an edge of M the action of making elastic some rigid edge $e \rightarrow f$ of M. The resulting split MSC is $M' = (P, O \cup \{\pi(e)\}, E, \rightarrow \setminus \{(e, f)\}, \dashrightarrow \cup \{(e, f)\}, \lhd, \pi, \lambda)$. For instance in Figure 5, M' is obtained by splitting two process edges of M.

We say that M can be *divided* into $M_1 = (P_1, O_1, E_1, \rightarrow_1, \dashrightarrow_1, \pi_1, \lambda_1)$ and $M_2 = (P_2, O_2, E_2, \rightarrow_2, \dashrightarrow_2, \pi_2, \lambda_2)$ when M_1 and M_2 are split (pre)-MSCs, and $E = E_1 \uplus E_2$, $\rightarrow = \rightarrow_1 \uplus \rightarrow_2$, $\lhd = \lhd_1 \uplus \lhd_2$, $\pi = \pi_1 \uplus \pi_2$, $\lambda = \lambda_1 \uplus \lambda_2$, and for $i \in \{1, 2\}$, $O_i = O \cap P_i$ and $\dashrightarrow_i \subseteq (\rightarrow \cup \dashrightarrow)^+$. For instance, in Figure 5, M' can be divided into M_1 and M_2. A *connected component* of M is a split MSC $M_1 = (P_1, O_1, E_1, \rightarrow_1, \dashrightarrow_1, \pi_1, \lambda_1)$ such that E_1 is a connected component of $(E, \rightarrow \cup \lhd)$. Then, either $M = M_1$ or M can be divided into M_1 and some M_2.

Split-game. Let M be a split MSC with at most k blocks. A *split-game with budget* k on M is a two player game in which the existential player (Eve) tries to prove that M has split-width at most k, while the universal player (Adam) tries to disprove it. Eve begins by trying to disconnect M by splitting some of its process edges, with the condition that the resulting split MSC M' has at most k blocks. Adam then chooses a connected component M'' of M', and the game resumes on M''. Eve wins a play if it ends in an *atomic* MSC, i.e. a pair of matching send and receive events. She loses if she cannot disconnect a non-atomic MSC without introducing more than k blocks.

The *split-width* of an MSC M is the minimal k such that Eve wins the split-game with budget k on M. It is defined identically for pre-MSCs. We denote by SW_k the set of MSCs of split-width at most k.

Example 8. Eve wins the split-game with budget 4 on M (see Figure 5). She starts by splitting two process edges of M, which results in the split MSC M' with three blocks. M' has two connected components, M_1 and M_2, providing two choices for Adam. If he chooses M_2, Eve wins by cutting the only remaining process edge: both connected components of the resulting M'_2 are atomic. If he chooses M_1, Eve split the middle process edge on the first process, which creates

two more blocks, and results in a total of four blocks. M_1' has two isomorphic connected components, so Adam's choices are all equivalent. Eve can then cut the two remaining process edges while still respecting her budget of four blocks.

Shuffle and merge. One can also give a bottom-up description of split-width. The duals of the split and divide operations are called respectively *merge* ($\divideontimes$) and *shuffle* ($\sqcup\!\sqcup$). $\divideontimes(M)$ is the set of split MSCs that can be obtained by making some elastic edges of M rigid, and/or closing some of its open processes. $M_1 \sqcup\!\sqcup M_2$ is the set of split MSCs M such that M can be divided into M_1 and M_2.

An MSC has split-width at most k when it can be obtain by combining atomic MSCs with shuffle and merge operations, while keeping the number of blocks at most k at each step.

Remark 9. The notion of open and closed processes is new. Our bound on the number of blocks (i.e. the number of open processes plus the number of elastic edges) replaces the bound on the number of elastic edges only that was used in [1,2] for the split-width of MSCs over fixed architectures. When the topology $\mathcal{T}$ is fixed, the two definitions are equivalent since the number of open processes is already bounded by the number of processes in $\mathcal{T}$. This is no longer true in the parameterized case. For instance, the families of MSCs defined in Figures 3 and 4 can be decomposed into atomic pieces while using only two elastic edges, but this introduces an unbounded number of open processes. In fact, they embed a grid, hence they should have unbounded width.

Remark 10. For MSCs, split-width is equivalent to tree-width and clique-width: there are linear bounds between the three measures (the proof is an easy adaptation from the non-parameterized case [10]). The motivation for introducing split-width rather than using existing measures on graphs is that it allows to take into account the specificities of MSCs, and is thus both simpler to understand and to use. In particular, using tree-width or clique-width would result in more involved automata constructions in Section 5.

Notice also that a bound on the split-width of an MSC M induces a bound on the tree-width of the observable topology $\mathcal{T}_M$ (See Theorem 20).

Decidability. The non-emptiness problem becomes decidable when restricted to MSCs of bounded split-width. Roughly, the proof goes as follows. First, we show that trees representing Eve's winning strategies can be abstracted by trees over a *finite* alphabet (that depends only on the bound k on split-width). Then, we reduce the verification problems for PCAs to emptiness problems on tree automata. The details for these constructions will be given in Section 5, and the complexity lower bounds are proven in Appendix G. The proof is inspired from the non-parameterized case [2], and we show that the complexity remains the same in our setting.

Theorem 11. SW-NonEmptiness *is* Exptime*-complete, and only polynomial in the number of states and transitions of the input PCA.*

4 Classes of bounded split-width

The decision procedures based on split-width are generic and apply to various classes of MSCs (the main condition being that MSCs in the class have bounded split-width). When the topology is fixed, this covers many well-studied restrictions [10]. In this section, we give two examples of such classes that can be generalized to the parameterized setting: existentially bounded MSCs, and context-bounded MSCs. We also define a further extension of context-bounded MSCs, called *tile-bounded* MSCs, and show that it is equivalent to bounded split-width.

Existentially bounded MSCs. M is called *existentially k-bounded* when there exists a linearization $\leq_{\text{lin}}$ of its events (i.e. a total order extending $\leq$) such that there are at most k *process or message edges* going out of any prefix of the linearization: for all $g \in E$,

$$\left|\left\{(e,f) \in E^2 \mid (e \lhd f \vee e \to f) \wedge e \leq_{\text{lin}} g <_{\text{lin}} f\right\}\right| \leq k\,.$$

In the case of MSCs over a fixed topology, this is equivalent to bounding the number of pending messages at each prefix of the linearization, which is the usual definition of existentially bounded. This is no longer the case when considering topologies with an unbounded number of processes. For instance, the MSC of Figure 3 is not existentially k-bounded. It is possible to find a linearization for which every prefix has at most one pending message, but it is not possible to simultaneously bound the number of non-terminated processes.

We denote by EB_k the set of all existentially k-bounded MSCs over $\mathcal{N}$.

Lemma 12. *An existentially k-bounded MSC has split-width at most $k+2$.*

Proof. Eve's strategy is as follows. She successively isolates the first events of the linearization by splitting the process edges originating from them, until a pair of matching send/receive events is disconnected. Adam chooses the remaining component, and Eve continues as before. In the split MSC obtained by isolating the first events $e_1 <_{\text{lin}} \ldots <_{\text{lin}} e_i$ of the linearization and removing the disconnected messages, each block either consists of a single e_j $(1 \leq j \leq i)$, or only contains events that occur after e_i in the linearization, and is the last block on some open process. Blocks of the first kind are necessarily send events whose matching receive event occurs after e_i, hence they correspond to pending messages at e_i. Now consider a block of the second kind, and let f be its first event. Then f must occur after e_i in the linearization (otherwise, its block would be of the first kind). Moreover, since its process is open, there must be some $e \in \{e_1, \ldots, e_i\}$ such that $e \to f$ in the initial MSC. Hence, each block of the second kind correspond to a pending process edge at e_i (the edge $e \to f$). Thus, there are in total at most k blocks. Eve introduces at most two extra blocks when splitting a process edge. Hence she wins with budget $k+2$. □

Eve's winning strategy results in a tree that is word-like [2, 10], i.e., at every binary node, one of the subtree is small (bounded size). Hence, we can use word automata instead of tree automata, resulting in a better complexity for the verification problems.

Theorem 13. EB-NonEmptiness *is* Pspace*-complete.*

Context-bounded MSCs. A context is an interval of events on a process, in which only one interface is accessed, and in a single direction (send or receive). More formally, let $M = (P, E, \rightarrow, \lhd, \pi, \lambda)$ be an MSC. A *context* of M is a subset $c = \{e_1, \ldots, e_n\}$ of E such that $e_1 \rightarrow \cdots \rightarrow e_n$ and $\lambda(e_i) = \lambda(e_j)$ for all i, j.

An MSC M is *k-context bounded* when for all $p \in P$, there are contexts $c_1, \ldots, c_i$ with $i \leq k$ such that $E_p = c_1 \uplus \ldots \uplus c_i$. The class of k-context bounded MSCs has unbounded split-width. Actually, this is even the case for MSCs having a bounded number of events on every process (see Figure 4). However, we obtain a bound on split-width when we additionally require that the topology has bounded tree-width (cf. Appendix B). The proof is in Appendix D.

Lemma 14. *If M is a k-context-bounded MSC and $\mathcal{T}_M$ has tree-width at most h, then M has split-width at most $k(h+1)+2$.*

We denote by $\mathsf{CB}_{k,h}$ the set of all k-context bounded MSCs over topologies of tree-width at most h.

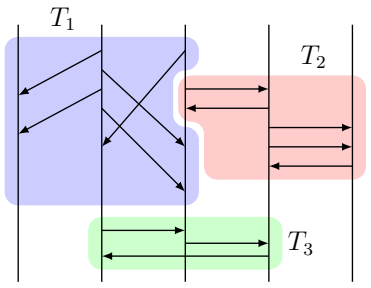
Theorem 15. CB-NonEmptiness *is* Exptime*-complete, and polynomial in the number of states and transitions of the input PCA.*

Tile-bounded MSCs. We can generalize the notion of contexts introduced above to *tiles*, which are independant parts of an MSC involving a *bounded* number of processes. In some sense, this section gives the link between fixed and parameterized topologies, when it comes to conditions ensuring decidability. Intuitively, an MSC with split-width at most k over an arbitrary topology (involving arbitrarily many processes) can be decomposed in tiles involving at most some fixed number (k) of processes. Moreover, each tile has bounded split-width and each process intersects at most some fixed number of tiles.

A *k-tile* is a split MSC T of split-width at most k and having only open processes. In particular, T has at most k blocks, hence at most k processes.

Let $M = (P, E, \rightarrow, \lhd, \pi, \lambda)$ be an MSC. A *(k,ℓ)-tile-decomposition* of M is a sequence $T_1, \ldots, T_n$ of k-tiles such that $M \in \mathbb{M}(T_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup T_n)$ and every process $p \in P$ is part of at most ℓ tiles. An MSC is called *(k,ℓ)-tile-bounded* when it admits some (k,ℓ)-tile-decomposition.

Example 16. A $(5,3)$-decomposition with three tiles is depicted on the right. The first and the last process intersect with one tile, the middle one with 3 tiles and the other two processes intersect with two tiles. Tiles T_1 and T_3 have split-width exactly 5 (note that the first process is counted as open in T_1). Tile T_2 tile has split-width 4.



Example 17. A k-context bounded MSC M (see page 11) admits a $(2k+2, 2|\mathcal{N}|)$-tile-decomposition. The tile decomposition is given by defining, for each pair of processes (p, q) connected in $\mathcal{T}_M$, a tile $T_{p,q}$ induced by the contexts of p in which it sends to q, and the contexts of q in which it receives from p. Notice that a tile needs not be a connected graph. In particular, each tile has at most $2k$ blocks, and can be decomposed by disconnecting one by one its messages, which introduces at most 2 extra blocks. Each process of M takes part in at most $2|\mathcal{N}|$ tiles, one for each type $(a!, a?, \ldots)$ of contexts it has.

We obtain the same results as for context-bounded MSCs (cf. Appendix E.1). We denote by $\mathsf{TB}_{k,\ell,h}$ the set of (k, ℓ)-tile-bounded MSCs M such that $\mathcal{T}_M$ has tree-width at most h.

Lemma 18. *Let $M \in \mathsf{TB}_{k,\ell,h}$. Then M has split-width at most $2k^2\ell^2(h+1)$.*

Theorem 19. TB-NonEmptiness *is* Exptime*-complete, and polynomial in the number of states and transitions of the input PCA.*

In fact, such bounds are equivalent to bounding split-width, as shown by the theorem below (proof is in Appendix E.2).

Theorem 20. *Let $M \in \mathsf{SW}_k$. Then $\mathcal{T}_M$ has tree-width at most $k-1$, and M is $(k^2+2k, 3|\mathcal{N}|^k)$-tile-bounded.*

5 Tree interpretation

We present the decision procedures leading to our complexity results. The general idea is to encode MSCs of bounded split-width into binary trees over a finite alphabet, and reduce our verification problems to problems on tree automata.

Split-terms. The encoding of MSCs of bounded split-width into trees is based on the bottom-up description of split-width. Recall that an MSC has split-width at most k if it can be constructed by combining through shuffles and merges split MSCs with at most k blocks, the starting points being atomic MSCs. This construction can be described by a *split-term*, that is, a term over the following grammar: $s ::= a! \lhd b? \mid \mathbb{M}(s) \mid s \sqcup\!\sqcup s$ (with $a, b \in \mathcal{N}$).

However, since the merge and shuffle operations are ambiguous, a split-term may correspond to several MSCs. The next step is to disambiguate these operations by adding labels to the nodes of split-terms, describing respectively how the blocks of the children are shuffled, or which blocks are merged and which processes are closed.

Compared to the non-parameterized case [1,2], the difficulty is that the number of processes may grow arbitrarily along the DST, instead of being fixed from the beginning – and we still need to use labels from a bounded domain. The solution comes from the distinction between open and closed processes, and the fact that the number of *open* processes stays bounded. Merge and shuffle operations only act on open processes: a merge makes some elastic edges rigid (which are

all located on open processes, by definition), and/or closes some open processes. Similarly, a shuffle of two split MSCs M_1 and M_2 may only combine some pairs of *open* processes of M_1 and M_2 by shuffling their blocks and adding elastic edges between them. It simply takes the disjoint union of closed processes.

Thus, the disambiguated labels will focus on open processes. The idea is to describe how many blocks each process has after the operation, and the origin of each block.

A *k-disambiguated split-term (k-DST)* is a split-term in which each internal node is labeled by a tuple of words $(w_p)_{1\leq p\leq m}$ such that $\sum_{p=1}^{m} |w_p| \leq k$, and

- For a $\sqcup\!\sqcup$-node, the word $w_p \in \{B_\ell, B_r\}^+$ describes the composition of some open process, where B_ℓ stands for a block coming from the left child, and B_r stands for a block coming from the right child (see for instance the label of n' in Figure 6 which describes the origin of the 3 blocks of the open process of M' in Figure 5).
- For a $\mathbb{M}$-node, the word $w_p \in \{B_i \mid 1 \leq i \leq k\}^*$ describes how the blocks of the p-th open process of its child are merged: B_i stands for a block resulting from the merge of i consecutive blocks of the child. We use $w_p = \varepsilon$ to indicate that process p is closed, merging all its blocks if any. For instance, in Figures 5 and 6 the label (B_2, B_2) of node n_3 indicates that on both process of M'_3, the two blocks are merged in M_3.
- For a $\lhd$-node, $m \leq 2$ and the word $w_p \in \{B_!, B_?\}$ indicates that the p-th open process consists of the send (resp. receive) event. If $m = 2$ then $w_1 \neq w_2$. For instance, n_7 is labeled $(B_!)$, which means that in M_7 the process of the send event is open whereas the process of the receive event is closed.

Further examples and explanations are given in Appendix A. We denote by DST^k the set of all k-DSTs. A k-DST is called *valid* when the label of each node is coherent with the number of processes and blocks appearing in the label of its child/children (see Appendix A for a formal definition). For instance, we cannot have $w_p = B_2B_1$ at a $\mathbb{M}$-node if its child does not have 3 blocks on process p.

A valid k-DST t can be associated with a unique split pre-MSC (which is not necessarily a split MSC) $M_t = (P, O, E, \rightarrow, \dashrightarrow, \lhd, \pi, \lambda)$, defined as follows. E is the set of leaves of t, and λ associates with a leaf e its label. We let $e \lhd f$ whenever e and f are respectively the left and right children of a same $\lhd$-node.

To determine whether two leaves e and f are connected by a $\rightarrow$-edge, we proceed as follows. We track the block associated with leaf e, until reaching a $\mathbb{M}$-node n in which it is merged with the block on its right (see example in green in Figure 6). Similarly, we track the block associated with leaf f, until reaching a $\mathbb{M}$-node n' in which it is merged with the block on its left (in blue in Figure 6). We set $e \rightarrow f$ if $n = n'$ and the blocks coincide. Similarly, we let $e \dashrightarrow f$ when no merge ever occurs on the right of the block of e or on the left of the block of f, and at the root, the block of f is located just after the block of e.

We identify processes with connected components of $(E, \rightarrow \cup \dashrightarrow)$. To determine whether the process of an event e is open or closed, we walk up the tree remembering the process of e, until reaching a $\mathbb{M}$-node in which it is closed, or the root (in which case it is open).

For example, in Figures 5 and 6 the split pre-MSC associated with the k-DST starting in node n_i (resp. n'_i) is M_i (resp. M'_i).

The next lemma states that the conditions for M_t to be an MSC can be checked by a tree automaton (proof given in Appendix A). We denote by $\mathsf{DST}^k_{\mathsf{msc}}$ the set of all valid k-DSTs t such that M_t is an MSC.

Lemma 21. *One can construct in space polynomial in k and $|\mathcal{N}|$ a deterministic bottom-up tree automaton $\mathcal{A}^k_{\mathsf{msc}}$ with $2^{\mathcal{O}(|\mathcal{N}|k^4)}$ states such that $L(\mathcal{A}^k_{\mathsf{msc}}) = \mathsf{DST}^k_{\mathsf{msc}}$.* ∎

From PCAs to tree automata. Given a PCA, we can construct a tree automaton that accepts a tree $t \in \mathsf{DST}^k_{\mathsf{msc}}$ iff M_t is accepted by the PCA.

Lemma 22. *Let $\mathcal{S} = (S, \iota, Msg, \Delta, F)$ be a PCA, and $k \in \mathbb{N}$. There is a bottom-up tree automaton $\mathcal{A}^k_{\mathcal{S}}$ of size $|S|^{\mathcal{O}(k|F|^2)}$ such that $L(\mathcal{A}^k_{\mathcal{S}}) \cap \mathsf{DST}^k_{\mathsf{msc}} = \{t \in \mathsf{DST}^k_{\mathsf{msc}} \mid M_t \in L(\mathcal{S})\}$. It can be constructed in space polynomial in k and $|F|$, and logarithmic in $|S|$ and $|\Delta|$.*

Proof. $\mathcal{A}^k_{\mathcal{S}}$ guesses a run of $\mathcal{S}$ on M_t, and inductively checks that it is a valid accepting run. To do so, it remembers the states of $\mathcal{S}$ before and after each block in the split MSC associated with the current subtree, that is, a pair (ρ^-, ρ^+) of partial functions from $[k]$ to S. (The blocks are numbered according to their position in the concatenation $w_1 w_2 \dots w_m$ of the words in the label of the current node.) In addition, for each $s \in S$ appearing in F, $\mathcal{A}^k_{\mathcal{S}}$ remembers the number n_s of closed processes that ends in state s, up to the maximal n such that $\langle \#(s) \geq n \rangle$ appears in F. A state is accepting if it satisfies F.

At leaves, $\mathcal{A}^k_{\mathcal{S}}$ remembers the type of action executed (in Σ), and at a $\lhd$-node of the form $a! \lhd b?$, it guesses a message m and transitions $p \xrightarrow{a!m} p'$ and $q \xrightarrow{b?m} q'$ of $\mathcal{S}$. The functions ρ^- and ρ^+ of the $\lhd$-node are initialized accordingly. For instance, after reading $a! \lhd_{(B_!, B_?)} b?$ and guessing the transitions, $\mathcal{A}^k_{\mathcal{S}}$ goes to the state where $\rho^-(1) = p, \rho^+(1) = p', \rho^-(2) = q, \rho^+(2) = q'$ and ρ^-, ρ^+ are undefined elsewhere. After reading $a! \lhd_{(B_?)} b?$, $\mathcal{A}^k_{\mathcal{S}}$ checks that $p = \iota$ and increments $n_{p'}$, and moves to state $\rho^-(1) = q, \rho^+(1) = q'$.

The functions ρ^- and ρ^+ are updated at each $\sqcup\!\sqcup$- and $\between$-node according to the renaming of the blocks. At a $\between$-node, $\mathcal{A}^k_{\mathcal{S}}$ checks than whenever two blocks b and $b+1$ are merged, $\rho^+(b) = \rho^-(b+1)$. It also checks that each process being closed starts in ι, and increments the counter n_s of its end state s (unless it has already reached its maximal value). ∎

We then have $L(\mathcal{S}) \cap \mathsf{SW}_k \neq \varnothing$ iff $L(\mathcal{A}^k_{\mathcal{S}} \cap \mathcal{A}^k_{\mathsf{msc}}) \neq \varnothing$, which leads to an algorithm in time polynomial in $|S|$ and exponential in k, $|\mathcal{N}|$ and $|F|$ to decide the non-emptiness of $L(\mathcal{S}) \cap \mathsf{SW}_k$. This proves the upperbound of Theorem 11.

Classes of bounded split-with. The above decision procedure can be adapted for any class $\mathcal{C}$ of MSCs of split-width at most k, provided we can construct an automaton $\mathcal{A}_{\mathcal{C}}$ that accepts only encodings of MSCs in $\mathcal{C}$, and at least one encoding for each $M \in \mathcal{C}$. Under those assumptions, and given a PCA $\mathcal{S}$, deciding whether $L(\mathcal{S}) \cap \mathcal{C} = \varnothing$ e.g. reduces to deciding whether $L(\mathcal{A}^k_{\mathcal{S}} \cap \mathcal{A}^k_{\mathsf{msc}} \cap \mathcal{A}_{\mathcal{C}}) = \varnothing$.

	$\mathcal{C}_k = \mathsf{EB}_k$	$\mathcal{C}_{(k,h)} = \mathsf{CB}_{k,h}$	$\mathcal{C}_k = \mathsf{SW}_k$	$\mathcal{C}_{(k,\ell,h)} = \mathsf{TB}_{k,\ell,h}$
$\mathcal{C}$-Reachability	Pspace-c	Exptime-c		
$\mathcal{C}$-NonEmptiness				
$\mathcal{C}$-CPDL-Sat/MC				
$\mathcal{C}$-ICPDL-Sat/MC	Expspace-c	2-Exptime-c		
$\mathcal{C}$-MSO-Sat/MC	Non-elementary			

Fig. 7. Complexity results. All problems are only polynomial in the number of states and transitions of the input PCA.

This applies in particular to existentially bounded MSCs, and context- or tile-bounded MSCs over topologies of bounded tree-width. The construction of $\mathcal{A}_\mathcal{C}$ is in all three cases based on the proof that $\mathcal{C}$ has split-width at most k (cf. Appendix C, D and E, resp.).

Note that this would also apply for instance to any class of bounded split-width that is recognized by a PCA.

6 Further results

Model-checking. The results presented in Sections 3 and 5 can be generalized to model-checking problems (the details are in Appendix F). Our most general decidability result states that the model-checking of PCAs against Monadic Second-Order (MSO) formulas is decidable. The idea is to construct, for a given specification φ, a tree automaton $\mathcal{A}^k_\varphi$ that accepts a valid k-DST t iff M_t satisfies φ. The bounded split-width model-checking problem then reduces to testing whether $L(\mathcal{A}^k_S \cap \mathcal{A}^k_{\neg\varphi} \cap \mathcal{A}^k_{\mathsf{msc}}) = \varnothing$, and similarly for the other classes.

However, when the specification φ is given by an MSO formula, the construction of $\mathcal{A}^k_\varphi$ is non-elementary. Towards a better complexity, we study model checking against PDL specifications. PDL is both expressive (it subsumes most if not all temporal logics), easy to use and understand, and enjoys a very good complexity. We show that model-checking against PDL formulas is Exptime-complete, i.e., not harder than non-emptiness and reachability. It remains in Exptime when we extend PDL with converse (CPDL), and is 2-Exptime-complete for ICPDL (PDL with converse and intersection). A summary of our results is given in Figure 7.

Multi-pushdown processes. Our model could be extended by adding one or several stacks to processes, similarly to what is done in the case of fixed architectures [2]. We could also allow several FIFO channels between any pair of processes. This means relaxing the definition of topologies to allow loops or multiple edges, and similarly adapt the definition of MSCs. The definition of split-width and k-DSTs is the same, except that at $\lhd$-nodes, the send and receive events may be placed on the same process. Our decision procedures remain correct, with an additional check by $\mathcal{A}^k_{\mathsf{msc}}$ of the LIFO conditions on the stacks. The results on existentially-bounded MSCs, context-bounded MSCs, or tile decompositions are also still valid.

References

1. C. Aiswarya and P. Gastin. Reasoning about distributed systems: WYSIWYG (invited talk). In *FSTTCS'14*, volume 29 of *LIPIcs*, pages 11–30. Leibniz-Zentrum für Informatik, 2014.
2. C. Aiswarya, P. Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *ATVA'14*, volume 8837 of *LNCS*, pages 1–17. Springer, 2014.
3. B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281. Springer, 2014.
4. B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR 2014*, volume 8704 of *LNCS*, pages 109–124, 2014.
5. B. Bollig. Logic for communicating automata with parameterized topology. In *CSL-LICS'14*. ACM, 2014.
6. B. Bollig, P. Gastin, and A. Kumar. Parameterized communicating automata: Complementation and model checking. In *FSTTCS'14*, volume 29 of *LIPIcs*, pages 625–637, 2014.
7. B. Bollig, P. Gastin, and J. Schubert. Parameterized Verification of Communicating Automata under Context Bounds. In *RP'14*, volume 8762 of *LNCS*, pages 45–57. Springer, 2014.
8. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.
9. B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.
10. A. Cyriac. *Verification of Communicating Recursive Programs via Split-width*. PhD thesis, ENS Cachan, 2014. `http://www.lsv.ens-cachan.fr/~cyriac/download/Thesis_Aiswarya_Cyriac.pdf`.
11. A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR'12*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.
12. G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR'10*, volume 6269 of *LNCS*, pages 313–327. Springer, 2010.
13. G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FoSSaCS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.
14. A. Durand-Gasselin, J. Esparza, P. Ganty, and R. Majumdar. Model checking parameterized asynchronous shared-memory systems. In *CAV 2015*, volume 9206 of *LNCS*, pages 67–84. Springer, 2015.
15. E.A. Emerson and K.S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.
16. B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.
17. ITU-TS Recommendation Z.120anb: Formal Semantics of Message Sequence Charts, 1998.
18. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

A Details on k-DSTs

Disambiguated split-terms. We give more detailed explanations on the labels of DSTs, that should be helpful in understanding the examples and the formalization of our constructions.

As explained in the main body of the paper, the idea is to describe how many blocks each process has after the operation, and the origin of each block. In this way, a $\sqcup\!\sqcup$-node is labeled by a tuple $(w_p)_{1\leq p\leq m}$, where m is the number of open processes after the shuffle, and each w_p describes the composition of some open process. More precisely, w_p is a word over $\{B_\ell, B_r\}^*$, where B_ℓ stands for a block coming from the left child, and B_r stands for a block coming from the right child. For instance, M' in Figure 5 has only one open process, with 3 blocks, so the label of the associated node n' in Figure 6 consists of a single word w of length 3. The first and last blocks come from M_1 (the left child), and the second block comes from M_2 (the right child), so $w = B_\ell B_r B_\ell$. There is no ambiguity regarding which blocks of M_1 are respectively represented by the two occurrences of B_ℓ, since the shuffle preserves by definition the order of the blocks on each process. To handle the case of nodes with several open processes, we add the convention that the order on the open processes (implied by the labels of the DST) is preserved by the shuffle. For instance, the label of n'_2 is $(B_\ell, B_\ell B_r)$, which implies that the first open process is the first open process of the left child, and the second open process comes from the shuffle of the second open process of the left child, and of the first open process of the right child.

A $\between$-node is labeled by a tuple $(w_p)_{1\leq p\leq m}$, where m is the number of open processes *before* the merge. When an open process p is closed (and all its elastic edges made rigid if it still had some), we set $w_p = \varepsilon$. For instance, the label of n indicates that all the elastic edges of M' are made rigid in M, and that the corresponding open process of M' is closed in M. Else, w_p is a word over $\{B_1, \ldots, B_k\}^*$, where B_i stands for a block resulting from the merge of i successive blocks (meaning that the $i-1$ elastic edges separating these blocks have been made rigid). For instance, the label (B_2, B_2) of node n_3 indicates that on both processes of M'_3, the two blocks are merged in M_3.

Finally, a $\lhd$-node is labeled by an element of $\{(), (B_!), (B_?), (B_!, B_?), (B_?, B_!)\}$, where $B_!$ (resp. $B_?$) stands for a block consisting of the send (resp. receive) event (meaning in particular that its process is open). The label of the node indicates which processes of the associated split MSC are open, and how they are ordered. For instance, n_7 is labeled $(B_!)$, which means that in M_7 the process of the send event is open whereas the process of the receive event is closed. Both n_4 and n_5 are labeled by a pair of words, so their associated split MSC M_4 and M_5 are isomorphic. The only difference is that in the rest of t, the sending process of M_4 will always come before its receiving process, while it is the opposite for M_5.

Valid k-DST. The *type* of a node n in a k-DST t consists of its number of open processes, together with the number of blocks on each open process. More precisely, if n has label $(w_p)_{1\leq p\leq m}$, $\mathsf{type}(n)$ is the list obtained by removing all occurrences of 0 in $(|w_p|)_{1\leq p\leq m}$. For instance, node n_1 in Figure 5 has type (2),

and node n'_1 has type $(2,2)$. A k-DST is called *valid* when for all nodes n with label $(w_p)_{1\leq p\leq m}$, the following conditions hold:

- Assume n is a $\mathbb{M}$-node with child n'. Then $\mathsf{type}(n')$ is of the form $(x_p)_{1\leq p\leq m}$, and for all p such that $w_p \neq \varepsilon$, writing $w_p = B_{i_1} \dots B_{i_s}$, we have $x_p = \sum_{j=1}^{s} i_j$.
- Assume n is a $\sqcup\!\sqcup$-node with left and right children n_ℓ and n_r. Let $p_1 < \ldots < p_s$ be the indices p such that $|w_p|_{B_\ell} \geq 1$. Then $\mathsf{type}(n_\ell) = (|w_{p_i}|_{B_\ell})_{1\leq i\leq s}$. Similarly, for the right child.

The set of all valid k-DSTs is denoted by $\mathsf{DST}^k_{\text{valid}}$. It can be recognized by a deterministic bottom-up tree automaton $\mathcal{A}^k_{\text{valid}}$ with $k^{\mathcal{O}(k)}$ states: $\mathcal{A}^k_{\text{valid}}$ remembers the type of the last seen node, and checks the validity conditions and membership in DST^k, i.e., that each node has at most k blocks.

The conditions for M_t to be an MSC (instead of any pre-MSC) can also be checked by a tree automaton.

Lemma 21. *One can construct in space polynomial in k and $|\mathcal{N}|$ a deterministic bottom-up tree automaton $\mathcal{A}^k_{\mathsf{msc}}$ with $2^{\mathcal{O}(|\mathcal{N}|k^4)}$ states such that $L(\mathcal{A}^k_{\mathsf{msc}}) = \mathsf{DST}^k_{\mathsf{msc}}$.*

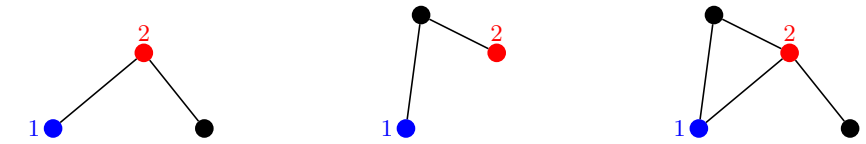
Proof. $\mathcal{A}^k_{\mathsf{msc}}$ is the intersection of $\mathcal{A}^k_{\text{valid}}$ with a tree automaton $\mathcal{A}_{\mathsf{msc}}$ that keeps in its state the known neighbors of each open process, that is, partial functions $\mathsf{neighbor}_a : [k] \to [k] \cup \{*\}$. The meaning of $\mathsf{neighbor}_a(p) = q$ is that the a-interface of the p-th open process points to the q-th open process (the order on open processes refers to the label $(w_p)_{1\leq p\leq m}$). We let $\mathsf{neighbor}_a(p) = *$ when the a-interface of the p-th open process points to some closed process. $\mathsf{neighbor}_a$ is undefined for all open processes p on which there is no event of type $a!$ or $a?$ yet. Also, $\mathcal{A}_{\mathsf{msc}}$ remembers the set of pairs of blocks that are connected by $\lhd$-edges, that is, a set $\lhd \subseteq [k]^4$ (the i-th block on the p-th open process being represented by the pair (p,i)). Then, $\mathcal{A}_{\mathsf{msc}}$ checks at each $\sqcup\!\sqcup$-node that:

- If two open processes p_ℓ and p_r of the left and right children are shuffled into some process p and both had a-neighbors, then their a-neighbors are open processes q_ℓ and q_r which are also shuffled in a same process q.
- For all $a \neq b$, if a process p_ℓ had an open a-neighbor q_ℓ, and is shuffled with a process p_r that had an open b-neighbor q_r, then q_ℓ and q_r are mapped to different processes by the shuffle.

Moreover, states that violate the FIFO policy, that is, such that $\lhd$ contains two pairs $((p,i),(q,j))$ and $((p,i'),(q,j'))$ such that $i < i'$ and $j' < j$, are disabled. Similarly, in order to check that in the resulting graph $\leq = (\lhd \cup \rightarrow)^*$ is a partial order, states such that the graph $([k]^2, \lhd \cup \{(p,i),(p,i+1)\})$ contains a cycle are disabled. □

Remark 23. We could also want to reason about a specific class $\mathfrak{T}$ of topologies among the classes of pipelines, rings, or trees, as in [6,7]. This can be done with slight modifications of $\mathcal{A}^k_{\mathsf{msc}}$, resulting in an automaton that recognizes exactly the set of valid k-DSTs t such that $\mathcal{T}_{M_t} \in \mathfrak{T}$.

B Tree-width



$\tau_1 = \mathsf{forget}_3(\boldsymbol{12} /\!\!/ \boldsymbol{23})$ $\tau_2 = \mathsf{forget}_3(\boldsymbol{13} /\!\!/ \boldsymbol{23})$ $\tau_3 = \tau_1 /\!\!/ \tau_2$

Fig. 8. Graphs denoted by τ_1, τ_2, and τ_3, respectively

There are many equivalent definitions of tree-width. We use a definition in terms of a graph algebra [9], which is closer to our definition of split-width.

We consider undirected simple graphs $G = (V, E)$ equipped with a *partial injective* coloring function $c : V \mapsto C$ of their vertices. We identify isomorphic graphs. Given a finite set of colors C and $x, y \in C$, we denote by

- $\boldsymbol{x}$ the graph consisting of a single vertex colored x,
- $\boldsymbol{xy}$ the graph with two vertices labeled x and y and one edge between them,
- forget_x the operation of uncoloring a vertex,
- $\mathsf{rename}_{x,y}(G, c)$ the graph (G, c') obtained by permuting colors x and y in G: $c'(v) = y$ if $c(v) = x$, $c'(v) = x$ if $c(v) = y$, and $c'(v) = c(v)$ otherwise,
- $(G, c) /\!\!/ (G', c')$ the union of G and G' where the vertices with same color are fused.

A *k-term* is a term over the algebra

$$\tau ::= \varnothing \mid \boldsymbol{x} \mid \boldsymbol{xy} \mid \tau /\!\!/ \tau \mid \mathsf{forget}_x(\tau) \mid \mathsf{rename}_{x,y}(\tau)\,, \quad \text{with } x, y \in [k]\,.$$

The semantics (G_τ, c_τ) of a k-term is as expected. The *tree-width* of a graph G is the least k such that $G = G_\tau$ for some $(k+1)$-term τ.

Theorem 24. *Let M be an MSC. If M has split-width at most k, then $\mathcal{T}_M$ has tree-width at most $k-1$.*

Proof. Let t be a k-DST for M. We construct by induction on t a k-term $\tau(t)$ for $\mathcal{T}_M$, such that the open processes of t are the colored vertices of $G_{\tau(t)}$. More precisely, the translation depends on the labels of the k-DST and is shown below:

- $\tau(a! \lhd_{(w_1,w_2)} b?) = \boldsymbol{xy}$, $\tau(a! \lhd_{(w_1)} b?) = \mathsf{forget}_x(\boldsymbol{xy})$, and $\tau(a! \lhd_{()} b?) = \mathsf{forget}_y(\mathsf{forget}_x(\boldsymbol{xy}))$.
- $\tau(t_1 \sqcup\!\sqcup_{(w_p)} t_2) = \tau(t_1) /\!\!/ \tau_2'$ where $\tau_2' = \mathsf{rename}_{x_1,y_1}(\cdots \mathsf{rename}_{x_i,y_i}(\tau(t_2))\cdots)$ is obtained from $\tau(t_2)$ by renaming colors so that the pairs of processes from $G_{\tau(t_1)}$ and $G_{\tau_2'}$ having the same color are exactly the pairs of processes being fused in the shuffle.
- $\tau(\curlywedge_{(w_p)}(t_1)) = \mathsf{forget}_{x_1}(\ldots(\mathsf{forget}_{x_i}(\tau(t_1)))\ldots)$, where $x_1, \ldots, x_i$ are the colors in $G_{\tau(t_1)}$ of the processes that are closed in the merge. □

C Details for existentially-bounded MSCs

Lemma 25. *One can construct in space polynomial in k a deterministic automaton $\mathcal{A}_{\mathsf{EB}_k}$ with $2^{\mathcal{O}(k)}$ states over $(k+2)$-DSTs such that $\mathsf{EB}_k = \{M_t \mid t \in L(\mathcal{A}_{\mathsf{EB}_k} \cap \mathcal{A}_{\mathsf{msc}}^{k+2})\}$.*

Proof. The construction of $\mathcal{A}_{\mathsf{EB}_k}$ is based on the proof of Lemma 12. We describe $\mathcal{A}_{\mathsf{EB}_k}$ as a top-down automaton. Thus, we interpret $\between$-node as splitting edges rather than merging blocks (or opening rather that closing processes), and $\sqcup\!\sqcup$-nodes as the removal of a single message edge (which is the case for Eve's strategy as described in the proof of Lemma 12).

Recall that in the proof of Lemma 12, events are detached one by one, according to the order given by the linearization. More precisely, we associate with any k-bounded linearization $e_1 <_{\text{lin}} \cdots <_{\text{lin}} e_m$ of a split MSC M a $(k+2)$-DST t such that $M = M_t$, with m $\between$-nodes denoted $x_1, \ldots, x_m$ (x_1 being the root, and x_m the $\between$-node of greatest depth). It is defined uniquely by the following conditions:

- If e_i is the only event on its process, then x_i opens its process.
- If e_i is not $\rightarrow$-maximal, then x_i splits its outgoing process edge.
- If e_i is $\rightarrow$-maximal and is not the only event on its process, then x_i does not do anything (i.e. $w_p \in (B_1)^*$ for any word of its label).
- If e_i is a receive event, then x_i is immediately followed by a $\sqcup\!\sqcup$-node removing its associated message pair.

The idea is for $\mathcal{A}_{\mathsf{EB}_k}$ to accepts all, and only, such DSTs. We associate with each node x_i the prefix $\{e_1, \ldots, e_i\}$ of the linearization (and with any $\sqcup\!\sqcup$-node, the prefix of its parent $\between$-node).

In the top-down view, $\mathcal{A}_{\mathsf{EB}_k}$ has a priori no information on the composition of the different blocks at each node. To handle this, it guesses at each $\between$-node whether the two blocks resulting from the split (or the block resulting from opening a process) consist of a single send event, a single receive event, or several events. Later, it checks that its guesses are correct: it verifies that a block guessed as a singleton is never split at a $\between$-node, and that the labels of $\lhd$-nodes are coherent with the guessed type (send or receive) of the singleton blocks.

In addition, $\mathcal{A}_{\mathsf{EB}_k}$ also guesses a linearization as follows. At each $\between$-node, it checks that at most one block is split and at most one process is open. If some block is split, the left part is guessed to be a singleton and its event is added to the linearization. This singleton block is marked. If no edge is split but a process is opened, $\mathcal{A}_{\mathsf{EB}_k}$ guesses that its block is a singleton, adds its event to the linearization and marks it. If no edge is split and no process is opened, $\mathcal{A}_{\mathsf{EB}_k}$ chooses non-deterministically a singleton block which is not yet marked and which is the first block on its process. As before, it marks this block and adds its event to the linearization.

A run of $\mathcal{A}_{\mathsf{EB}_k}$ on a DST t thus defines an enumeration of the events of M_t. To ensure that it is a linearization, $\mathcal{A}_{\mathsf{EB}_k}$ checks that at each $\sqcup\!\sqcup$-node, both events being removed are marked, and that at each $\between$-node, if the event added

to the linearization is a receive, then the next node is a $\sqcup\!\sqcup$-node removing its associated message pair.

To ensure that the linearization $e_1 <_{\text{lin}} \cdots <_{\text{lin}} e_m$ associated with a run of $\mathcal{A}_{\mathsf{EB}_k}$ is k-bounded, it is sufficient to prove that for all *send* events e_i we have $|\{(j, j') \mid (e_j \lhd e_{j'} \vee e_j \to e_{j'}) \wedge j \leq i < j'\}| \leq k$. Indeed, when e_i is a receive event, adding e_i to the prefix adds at most one pending process edge, and removes one pending message. Hence the number of pending edges never increases at receive events.

Let $\{e_1, \ldots, e_i\}$ be some prefix of the linearization, such that e_i is a send event. Let x_i be the $\between$-node at which e_i is added to the linearization, and M_i its associated MSC. For all $1 \leq j \leq i$, event e_j is either a singleton block of M_i, or is part of a message pair which has been already removed. Moreover, all events occurring after e_i are events of M_i (and thus so are their matching events). So any edge $e_j \lhd e_{j'}$ with $j \leq i < j'$ can be associated with the block formed by e_j in M_i. Similarly, any edge $e_j \to e_{j'}$ with $j \leq i < j'$ can be associated with the block of $e_{j'}$ in M_i (note that $e_{j'}$ is necessarily on an open process, since its incoming process edge is missing). In that case $e_{j'}$ is necessarily the first event of its block, so this mapping is injective.

Thus, to ensures that the linearization is k-bounded, $\mathcal{A}_{\mathsf{EB}_k}$ can simply check that at each $\between$-node, if the event being added to the linearization is a send, then there are at most k blocks.

Conversely, it is clear that all $(k + 2)$-DSTs obtained from the proof of Lemma 12 and described above are accepted by $\mathcal{A}_{\mathsf{EB}_k}$. □

D Details for context-bounded MSCs

Lemma 14. *If M is a k-context-bounded MSC and $\mathcal{T}_M$ has tree-width at most h, then M has split-width at most $k(h+1)+2$.*

Proof. We first describe Eve's strategy in the case of tree topologies, which have tree-width 1. Eve starts by splitting all the edges separating the contexts of the root p_0 of $\mathcal{T}_M$. This creates at most k blocks, and divides M into one component per child of p_0. Adam chooses one of these components, associated with some child p_1 of p_0. Eve splits all the edges separating the contexts of p_1. The resulting split MSC can be divided in two parts, M_1 and M_2, where M_1 consists of all the contexts in which p_0 and p_1 communicate with one another. In M_1, each connected component has the form described in Figure 9: it is composed of some contexts in which p_0 sends to p_1, and matching receive contexts of p_1, or the opposite. Since the messages are received in the same order as they are sent,

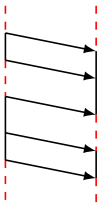


Fig. 9. A connected component of M_1

they can be detached one by one with two splits, which creates two extra blocks. So Eve wins on this part with budget $2k+2$. Therefore, Adam must choose a component of the remaining part M_2. But M_2 is again a k-context-bounded split MSC over a tree topology (with root p_1), whose blocks are the contexts of the root p_1. So whatever component of M_2 Adam chooses, Eve can apply the same strategy as for the previous move.

In the general case where $\mathcal{T}_M$ has tree-width bounded by h, the strategy of Eve is similar: she separates at each step all the contexts of some processes. Her strategy for choosing these processes is determined by some $(h+1)$-term τ for $\mathcal{T}_M$.

Alternatively, we can construct in a bottom-up fashion a DST $\overline{\tau}$ for M by induction on τ. The colored vertices of τ correspond to the open processes of $\overline{\tau}$ and the blocks of each open process correspond to some of its contexts. We can assume that τ contains no occurrence of $\varnothing$, $\boldsymbol{x}$, and $\mathsf{rename}_{x,y}$, and that for each subterm $\tau_1 /\!\!/ \tau_2$ and pairs of colors (x,y) occurring both in τ_1 and τ_2, if there is an edge in G_{τ_1} (resp. G_{τ_2}) between the vertices colored x and y, then there is none in G_{τ_2} (resp. G_{τ_1}).

– $\overline{\boldsymbol{xy}}$ is some $(2k+2)$-DST $t_{p,q}$ denoting the split MSC induced by all the contexts in which the processes p and q colored x and y in $G_{\boldsymbol{xy}}$ communicate

with one another. We have explained above for tree topologies how to obtain such a $(2k+2)$-DST.
- $\overline{\tau_1 \mathbin{/\!\!/} \tau_1}$ is the shuffle of $\overline{\tau_1}$ and $\overline{\tau_2}$, according to the partial order of M.
- $\overline{\mathsf{forget}_x(\tau_1)}$ is the merge of $\overline{\tau_1}$ closing the process colored x in G_{τ_1}, and keeping unchanged all the other processes. Notice that the blocks/contexts of a process are merged when it is closed.

At any node of $\overline{\tau}$ (except subterms of $t_{p,q}$), there are at most $(h+1)$ open processes and at most k blocks on each open process. Hence $\overline{\tau}$ is a $(k(h+1)+2)$-DST. □

Lemma 26. *One can construct in space polynomial in k and h a tree-automaton $\mathcal{A}_{\mathsf{CB}_{k,h}}$ with $|\mathcal{N}|^{\mathcal{O}(kh)}$ states such that $\mathsf{CB}_{k,h} = \{M_t \mid t \in L(\mathcal{A}_{\mathsf{CB}_{k,h}} \cap \mathcal{A}^{k'}_{\mathsf{msc}})\}$, where $k' = k(h+1)+2$.*

Proof. $\mathcal{A}_{\mathsf{CB}_{k,h}}$ recognizes DSTs such that: (1) at any moment, there are at most $h+1$ open processes (this is given by the node labels); (2) until a process is closed, each of its blocks are contexts ($\mathcal{A}_{\mathsf{CB}_{k,h}}$ remembers the type of each block/context in its state, and checks that there is no merge between two blocks of different types); (3) when a process is closed, it has at most k blocks, hence at most k contexts; (4) there are always at most $k+1$ blocks on each process, and at most k blocks on a process which communicates with at least two different processes.

For any k-context bounded MSC M such that $\mathcal{T}_M$ has tree-width at most h, we can construct (based on the proof of Lemma 14) a $(k(h+1)+2)$-DST accepted by $\mathcal{A}_{\mathsf{CB}_{k,h}}$. Conversely, it is clear that any MSC accepted by $\mathcal{A}_{\mathsf{CB}_{k,h}}$ is k-context-bounded. The fact that its topology has tree-width at most h is a consequence from the fact that it uses at most $h+1$ open processes (see proof of Theorem 24). □

E Details for tile-bounded MSCs

E.1 Decidability

Lemma 18. *Let $M \in \mathsf{TB}_{k,\ell,h}$. Then M has split-width at most $2k^2\ell^2(h+1)$.*

Proof. Let $\{T_1, \ldots, T_n\}$ be a (k,ℓ)-tile-decomposition of M. Note that, if the observable topology $\mathcal{T}_{T_i}$ of a k-tile T_i is not connected, we can replace T_i by several k-tiles having connected observable topologies. This does not change the number of tiles intersecting each process. Hence, w.l.o.g., we can assume that the observable topology $\mathcal{T}_{T_i}$ is connected for all i. We denote by $P_i \subseteq P$ the set of processes of T_i, and by t_i some k-DST denoting T_i.

We are going to construct a DST for M from successive shuffles of the t_i, in an order defined according to some $(h+1)$-term τ for $\mathcal{T}_M$. The idea is to follow the order in which the processes are added to the topology in the tree-term, and to add a tile T_i as soon as all its processes have been added to the topology and *uncolored*. The elastic edges separating the different tiles are made rigid as soon as possible, and open processes are closed as soon as all their tiles have been added to the split MSC.

For $P' \subseteq P$, we let $M_{P'}$ be the sub-MSC of M resulting from the shuffle of all the T_i's such that $P_i \subseteq P'$, followed by a merge (merging all blocks that are consecutive in M, and closing all processes that already have all their events). In particular, $M = M_P$.

For each subterm τ' of τ, we define a DST $\overline{\tau'}$ such that, with $P_{\tau'}$ the set of all *uncolored* processes of $G_{\tau'}$, we have $M_{\overline{\tau'}} = M_{P_{\tau'}}$. To keep the definition simple, we do not provide the disambiguating labels. At shuffle nodes, we follow on each open process the ordering of the blocks defined by M. At merge nodes, we merge consecutive blocks and we close processes as soon as possible, as explained above.

- If $\tau' = \boldsymbol{x}$ or $\tau' = \boldsymbol{xy}$, then $\overline{\tau'}$ is empty since $P_{\tau'} = \emptyset$.
- if $\tau' = \tau_1 /\!\!/ \tau_2$ then $\overline{\tau'} = \mathbb{M}(\overline{\tau_1} \sqcup\!\sqcup \overline{\tau_2} \sqcup\!\sqcup t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j})$ where $i_1, \ldots, i_j$ are the indices i such that $P_i \subseteq (P_{\tau_1} \cup P_{\tau_2})$ but $P_i \nsubseteq P_{\tau_1}$ and $P_i \nsubseteq P_{\tau_2}$. Since $P_{\tau'} = P_{\tau_1} \uplus P_{\tau_2}$, using the induction hypothesis, we can check that $M_{\overline{\tau'}} = M_{P_{\tau'}}$.
- If $\tau' = \mathsf{forget}_x(\tau'')$: let p be the process being uncolored. We let $\overline{\tau'} = \mathbb{M}(\overline{\tau''} \sqcup\!\sqcup t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j})$, where $i_1, \ldots, i_j$ are the indices i such that $P_i \subseteq P_{\tau'} = P_{\tau''} \uplus \{p\}$, but $P_i \nsubseteq P_{\tau''}$. Since $P_{\tau'} = P_{\tau''} \uplus \{p\}$, using the induction hypothesis, we can check that $M_{\overline{\tau'}} = M_{P_{\tau'}}$.

To prove that τ is a $2k^2\ell^2(h+1)$-DST, we show that before each $\mathbb{M}$-node, there are at most $2k\ell(h+1)$ open processes. Since there are at most ℓ tiles on each process and k blocks on each tile, there is always at most $k\ell$ blocks on each process.

First, consider some open process q of $M_{\overline{\tau'}}$, for a subterm τ' of τ. Since processes are closed as soon as possible, some tile T_i of q must still be missing in $M_{\overline{\tau'}}$ (cf. Figure 10). In our construction, a tile is added as soon as all its processes have been added and uncolored in the tree-term, so this means that

some process of T_i must be missing or colored in $G_{\tau'}$. Suppose for instance that some processes of T_i are missing (represented by dotted lines in Figure 10). Using the fact that $\mathcal{T}_{T_i}$ is connected, there must be processes u and v in T_i such that u is in $G_{\tau'}$, but v is not. Then u must be a colored process of $G_{\tau'}$.

So in all cases, an open process q of $M_{\overline{\tau'}}$ must share a tile with some colored process of $G_{\tau'}$. Now, each colored process u is part of at most ℓ tiles, and each tile has at most k processes (see Figure 11). So there are at most $k\ell$ process sharing a tile with u. Moreover, there are at most $(h+1)$ colored process. So, at most $k\ell(h+1)$ processes of $M_{\overline{\tau'}}$ share a tile with (any) colored process of $G_{\tau'}$.

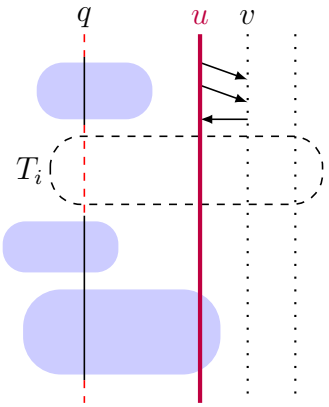


Fig. 10. Tiles of an open process q

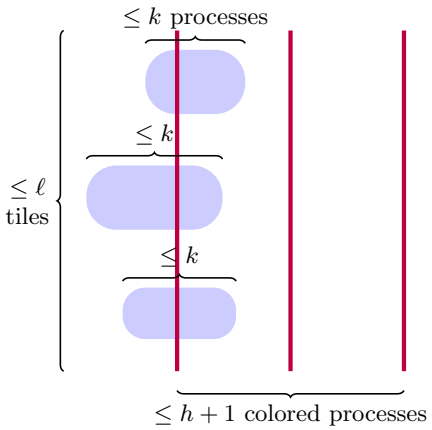


Fig. 11. Number of open processes

Finally, observe that our reasoning about open processes of $M_{\overline{\tau'}}$ also holds for processes q of $G_{\tau'}$ that are not in $M_{\overline{\tau'}}$. (All we need is that some tile of q is missing.) Thus, there are at most $k\ell(h+1)$ processes of $G_{\tau'}$ that are either open processes of $M_{\overline{\tau'}}$ or not in $M_{\overline{\tau'}}$.

We are now ready to prove that $\overline{\tau}$ is a $2k^2\ell^2(h+1)$-DST:

- All nodes of the t_i's have at most k blocks, since the t_i's are k-DSTs.
- Consider a subterm $\overline{\tau'} = \mathbb{M}(\overline{\tau_1} \sqcup\!\sqcup \overline{\tau_2} \sqcup\!\sqcup t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j})$ of $\overline{\tau}$, as defined above. We show that $t' = \overline{\tau_1} \sqcup\!\sqcup \overline{\tau_2} \sqcup\!\sqcup t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j}$ has at most $2k\ell(h+1)$ open processes. Notice that each process of t', be it open or closed, is in $P_{\tau'} = P_{\tau_1} \uplus P_{\tau_2}$. Hence, we show that t' has at most $k\ell(h+1)$ open processes in P_{τ_1}, and at most $k\ell(h+1)$ open processes in P_{τ_2}.
  If an open process $p \in P_{\tau_1}$ of t' is also a process of $\overline{\tau_1}$, then it must be open as well in $\overline{\tau_1}$. Moreover, at most $k\ell(h+1)$ processes of G_{τ_1} are either not processes of $\overline{\tau_1}$, or open processes of $\overline{\tau_1}$. So t' has at most $k\ell(h+1)$ open processes in P_{τ_1}.
- Consider a subterm $\overline{\tau'} = \mathbb{M}(\overline{\tau''} \sqcup\!\sqcup t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j})$ of $\overline{\tau}$, as defined above. We show that $t' = \overline{\tau''} \sqcup\!\sqcup t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j}$ has at most $k\ell(h+1)+1$ open processes.

As in the previous case, t' has at most $k\ell(h+1)$ open processes in $P_{\tau''}$. Since all processes of t' are in $P_{\tau'} = P_{\tau''} \uplus \{p\}$, t' has in total at most $k\ell(h+1)+1$ open processes. □

Lemma 27. *One can construct in space polynomial in k, ℓ, h a tree-automaton $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ with $2^{O(k^2\ell h^2)}$ states such that $\mathsf{TB}_{k,\ell,h} = \{M_t \mid t \in L(\mathcal{A}_{\mathsf{TB}_{k,\ell,h}} \cap \mathcal{A}^{k'}_{\mathsf{msc}})\}$, where $k' = 2k^2\ell^2(h+1)$.*

Proof. The idea is that $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ recognizes DSTs of the form described in the proof of Lemma 18. In particular, using the same notations as in that proof, they are all of the form:

$$\overline{\tau} ::= \text{M}(t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j}) \mid \text{M}(\overline{\tau} \sqcup\!\sqcup (t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j})) \mid \text{M}((\overline{\tau} \sqcup\!\sqcup \overline{\tau}) \sqcup\!\sqcup (t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j}))\,,$$

where $t_{i_1}, \ldots, t_{i_j}$ denote k-tiles. Notice that in the proof of Lemma 18, some of the $\overline{\tau_1}, \overline{\tau_2}$ may be empty and thus are not truly part of the complete DST (and then, neither is their father M-node). This explain why we can have DST of the form $\text{M}(t_{i_1} \sqcup\!\sqcup \ldots \sqcup\!\sqcup t_{i_j})$. However, we assume that all M-nodes described in the proof (except those such that there child node is empty) are actual nodes of the DST, though they may perform no action (i.e. be such that $w_p \in (B_1)^*$ for any word of the label).

To check if a DST is of the above form, the bottom-up automaton $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ remembers for each node a value $\mathsf{stage} \in \{0,1,2\}$, with $\mathsf{stage} = 0$ for the internal nodes of the t_i's, $\mathsf{stage} = 1$ for the root of a t_i or a shuffle of several t_i's, and $\mathsf{stage} = 2$ for all other nodes. So initially, $\mathsf{stage} = 0$, and at some point, $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ guesses that a tile is complete and goes to $\mathsf{stage} = 1$. Then only certain shuffles and merges are allowed, so that $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ recognizes DSTs of the form described above (it uses some additional information to distinguish, among nodes such that $\mathsf{stage} = 2$, those of the form $\overline{\tau}$, $\overline{\tau} \sqcup\!\sqcup \overline{\tau}$, $\overline{\tau} \sqcup\!\sqcup t$, or $(\overline{\tau} \sqcup\!\sqcup \overline{\tau}) \sqcup\!\sqcup t$, where the root of t is in state $\mathsf{stage} = 1$).

As long as $\mathsf{stage} = 0$, $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ checks that it is indeed reading some k-DST and not just any k'-DST, and that no process is ever closed. This can be achieved with a slight modification of the automaton $\mathcal{A}^k_{\text{valid}}$. Then, when $\mathsf{stage} \in \{1,2\}$, $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ remembers the number of tiles currently intersecting with each open process, and checks that it never gets greater than ℓ. This ensures that all DSTs accepted by $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ are (k,ℓ)-tile-bounded.

In addition, $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ guesses an $(h+1)$-term τ for the observable topology $\mathcal{T}$ of the current split MSC. Following the proof of Lemma 18, it guesses τ in such a way that all processes of $\mathcal{T}$ (open or closed) are uncolored processes of G_τ. It forgets about the $(h+1)$-term itself, as well as what happens on closed processes of $\mathcal{T}$ (all of them are uncolored in τ, and thus do not intervene in the remaining of the tree-term, nor naturally in the remaining of the DST). However, it keeps in its state the subgraph H_τ of G_τ induced by the other processes, i.e. colored vertices of G_τ, uncolored vertices of G_τ that are not part of $\mathcal{T}$ yet, and open processes of $\mathcal{T}$. In the proof of Lemma 18, there are always at most $m = 2k\ell(h+1)$ such processes.

First, for all nodes such that $\mathsf{stage} \in \{0,1\}$ (i.e. when the current split MSC is a tile in construction, or a disjoint shuffle of tiles), $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ remembers the observable topology $\mathcal{T}$ of the current split MSC. Recall that all processes of the split MSCs associated with such nodes are open, and thus can be identified with elements of $[k']$. We ignore the interface names, and represent $\mathcal{T}$ by a subset of $[k']^2$.

Then, for each node such that $\mathsf{stage} = 2$, $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ keeps in its state a graph H_τ with set of vertices included in $[m]$, together with a partial coloring function $c : [m] \to [h+1]$, and a partial injective function $\mathsf{num} : [m] \to [k]$, such that c and num have disjoint domains. The intuition is that $\mathsf{num}(v) = p$ when the vertex v of H_τ is the p-th open process of $\mathcal{T}$. When $c(v)$ is defined, it gives the color of the vertex of H_τ associated with v. We explain simultaneously the construction of H_τ performed by $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$, and how it can be associated inductively with an $(h+1)$-term τ, such that $\mathcal{T}$ is a subgraph of G_τ that uses only uncolored vertices, and H_τ is the subgraph of G_τ induced by all the vertices that are not closed processes of $\mathcal{T}$.

- Consider a $\mathbb{M}$-node with child n' where the value for stage goes from 1 to 2. This corresponds to a set of tiles being merged. (In the proof of Lemma 18, this happens when enough processes have finally been uncolored in the tree-term to cover all processes of some tiles.)
  $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ must initialize H_τ. To do so, it guesses a graph $H_\tau = G_\tau$ of tree-width at most h and functions c and num such that the topology $\mathcal{T}'$ stored at node n' is a subgraph of H_τ (the correspondence between the vertices being given by num).
- Consider a shuffle between a node n_1 such that $\mathsf{stage} = 2$ and a node n_2 such that $\mathsf{stage} = 1$. This means that some tiles (denoted by n_2) are added to the current MSC (denoted by n_1). This can only happens when all the processes of n_2 are already uncolored in the tree-term.
  $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ takes for H_τ the graph stored for n_1, and checks that the topology $\mathcal{T}_2$ stored for n_2 is a subgraph of H_τ. If a process of n_2 is shuffled with a process p_1 of n_1, then it must correspond to the process $\mathsf{num}_1(p_1)$. Other processes of n_2 (all of which are open) can be associated with any process of H_τ for which num_1 is undefined. The function num_1 is then updated and extended into num to follow the possible additions of open processes from n_2, and renumbering of the open processes of n_1.
  $\mathcal{T}_2$ is thus a subgraph of G_τ, and by induction hypothesis, so is the topology $\mathcal{T}_1$ of the split MSC associated with n_1 (both respecting num). Thus, the topology $\mathcal{T}$ after the shuffle is still a subgraph of G_τ.
- Consider a shuffle between two nodes n_1 and n_2 for which $\mathsf{stage} = 2$. Let H_{τ_1} and H_{τ_2} be the graphs stored respectively for the right and left child.
  In the proof of Lemma 18, a node of this form correspond to a node $\tau_1 \mathbin{/\!\!/} \tau_2$ in the tree-term. To construct the DST $\overline{\tau_1 \mathbin{/\!\!/} \tau_2}$, the first step is to take the shuffle of $\overline{\tau_1}$ and $\overline{\tau_2}$ (afterwards, a second shuffle corresponding to the previous case is performed to add missing tiles). Recalled that the processes of $\overline{\tau_1}$ are uncolored processes of τ_1, and similarly for τ_2. Thus, they are not

fused in $\tau_1 /\!\!/ \tau_2$, and the shuffle $\overline{\tau_1} \sqcup\!\sqcup \overline{\tau_2}$ is disjoint (that is, $w_p \in B_\ell^* \cup B_r^*$ for each word of the label).
At such a node, $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ thus start by checking that the shuffle is disjoint. It also checks that $H_{\tau_1} /\!\!/ H_{\tau_2}$ contains at most m vertices, where H_{τ_1} and H_{τ_2} are the graphs stored respectively for the left and right children. It then takes $H_\tau = H_{\tau_1} /\!\!/ H_{\tau_2}$.
This correspond to taking $\tau = \tau_1 /\!\!/ \tau_2$. By induction hypothesis, the topologies $\mathcal{T}_1$ and $\mathcal{T}_2$ for the left and right child are subgraphs of resp. G_{τ_1} and G_{τ_2} that only use uncolored vertices, hence are disjoint subgraphs of G_τ. Moreover, the topology $\mathcal{T}$ is the disjoint union of $\mathcal{T}_1$ and $\mathcal{T}_2$. Thus it is also a subgraph of G_τ.
- Consider a $\mathbb{M}$-node such that $\mathsf{stage} = 2$ for the child node n'. Then $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ may guess an uncoloring of a vertex of the graph $H_{\tau'}$ stored for n', or a parallel union with a graph H of tree-width at most h and such that $H_{\tau'} /\!\!/ H$ has at most m vertices. $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ then removes from the resulting graph all the processes that are closed during the merge, and takes it as the new value for H_τ.
This corresponds to taking for $\tau = \tau'$, $\tau = \mathsf{forget}_x(\tau')$, or $\tau = \tau' /\!\!/ \tau''$, where τ'' is an $(h+1)$-term for H. Since $\mathcal{T}$ does not change, it is still a subgraph of G_τ.

Any valid k'-DST accepted by $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$ then denotes a (k,ℓ)-bounded MSC over a topology of tree-width at most h. Conversely, all k'-DSTs described in the proof of Lemma 18 are accepted by $\mathcal{A}_{\mathsf{TB}_{k,\ell,h}}$. □

E.2 Equivalence with bounded split-width

Lemma 28. *Let t be a valid k-DST, and $M = M_t$. Let M' be some sub-MSC of M (i.e., there exists N' such that $M \in \mathbb{M}(N \sqcup\!\sqcup N')$, with b blocks. Then N has split-width at most $k + b$.*

Proof. We write $M = (P, O, E, \to, \dashrightarrow, \lhd, \pi, \lambda)$, $M' = (P', O', E', \to', \dashrightarrow', \lhd', \pi', \lambda')$, and for a k-DST r, $M_r = (P_r, O_r, E_r, \to_r, \dashrightarrow_r, \lhd_r, \pi_r, \lambda_r)$.

We define a $(k+b)$-DST t' for M', by restricting t to the events that are in N. More precisely, for each subterm s of t, we define a $(k+b)$-DST s' such that $M_{s'}$ is a sub-MSC of M_s, $E_{s'} = E_s \cap E'$, $O_{s'} = P_{s'} \cap (O_s \cup O')$, and $\dashrightarrow_{s'} = E_{s'}^2 \cap (\dashrightarrow_s \cup \dashrightarrow')$.

- If s is a leaf of t, we let $s' = s$ if s denotes an event of N, and $s' = \varnothing$ otherwise.
- If $s = \mathbb{M}_{(w_p)}(s_1)$, we let $s' = \mathbb{M}_{(w'_p)}(s'_1)$, where (w'_p) is defined so that for all elastic edge $e \dashrightarrow f$ of M_{s_1} that are also in $M_{s'_1}$, if $e \dashrightarrow f$ is made rigid in M_s, and is rigid in N, then it is also made rigid in $M_{s'}$.
- If $s = s_1 \sqcup\!\sqcup_{(w_p)} s_2$, we let $s' = s'_1 \sqcup\!\sqcup_{(w'_p)} s'_2$, where (w'_p) is defined so that the order in which the blocks of s'_1 and s'_2 are shuffled respect the order defined by s.

To prove that t' is a $(k+b)$-DST, we need to show that for all subterm s of t, s' has at most k blocks, i.e. $|O_{s'}| + |{\dashrightarrow_{s'}}| \leq k + b$.

By definition, we have $|O_{s'}| \leq |O_s| + |O'|$, and $|{\dashrightarrow_{s'}}| \leq |{\dashrightarrow_s}| + |{\dashrightarrow'}|$. Then, since t is a k-DST, $|O_s| + |{\dashrightarrow_s}| \leq k$. And M has b blocks, i.e. $|O'| + |{\dashrightarrow'}| \leq b$. Hence $|O_{s'}| + |{\dashrightarrow_{s'}}| \leq k + b$. □

Theorem 20 is obtained as the combination of Lemma 24 and 29.

Lemma 29. *Let M be an MSC with split-width at most k. Define*

$$\ell = \begin{cases} 1 + 2k & \text{if } |\mathcal{N}| = 2 \\ 1 + |\mathcal{N}| \dfrac{(|\mathcal{N}| - 1)^k - 1}{|\mathcal{N}| - 2} & \text{otherwise.} \end{cases}$$

Then M admits a (k^2+2k, ℓ)-tile decomposition where each tile contains at most k processes.

Proof. Let t be a valid k-DST such that $M = M_t = (P, E, \rightarrow, \lhd, \pi, \lambda)$. We first introduce some notations.

For each subterm s of t we consider the associated split MSC M_s. We denote by O_s (C_s resp.) the set of open (closed resp.) processes of M_s. In particular, $O_t = \varnothing$ and $C_t = P$. We also denote by E_s the set of events of M_s.

A subset $F \subseteq E$ is $\lhd$-closed if for all $e \lhd f$, either both events are in F or none of them are in F. For a $\lhd$-closed subset $F \subseteq E$, we denote by M_F the *split MSC induced by* F, i.e., the split MSC whose set of events is F, and such that two events of F are connected by a $\rightarrow$- or $\lhd$-edge in M_F iff they are in M. The open processes of M_F are those on which some events of M are missing. The elastic edges are such that the order of the events in M_F is compatible with M. Notice that $M \in \mathbb{M}(M_F \sqcup\!\sqcup M_{E \setminus F})$.

We will define a tuple of tiles $(T_p)_{p \in P}$ which forms a (k^2+2k, ℓ) decomposition of M. These tiles will be defined below by induction on t. Intuitively, whenever a process p is closed at some subterm s of t, we define the tile T_p as the split-MSC induced by the set of events that are not yet covered by some tile and that are connected to p via open processes.

For each $p \in P$, let $X_p \subseteq E$ be the subset of events of the tile T_p. We will ensure the following, where s is an arbitrary subterm of t and $F_s = \bigcup_{p \in C_s} X_p$:

1. For all $p \in P$, T_p is a $(k^2 + 2k)$-tile which contains at most k processes, each of which being at distance at most k from p in the topology $\mathcal{T}_M$. The tiles are pairwise disjoint: $X_p \cap X_q = \varnothing$ for all $p \neq q$.
2. The tiles $(T_p)_{p \in C_s}$ are contained in M_s and cover all events of the closed processes of M_s: $\{e \in E \mid \pi(e) \in C_s\} \subseteq F_s \subseteq E_s$. On the other hand, some events of M_s located on open processes may not be in F_s.
3. The split MSC M_{F_s} induced by F_s has at most k blocks on each process.

Before defining the tile-decomposition $(T_p)_{p \in P}$, we first show that the above conditions allow to derive the theorem. First, by 1, each T_p is a $(k^2 + 2k)$-tile,

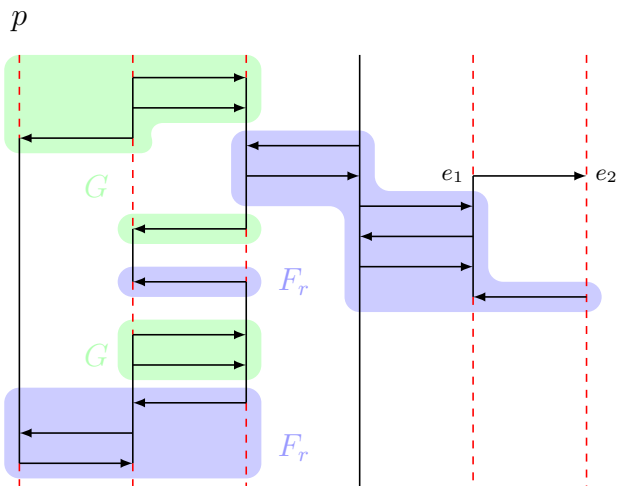


Fig. 12. Definition of T_p. The set of events covered by the previous tiles, F_r, is represented in blue. It consists of all events of the closed processes (here, the fourth process only), and some of the events of the open processes. All events that are not in F_r are added to the new tile T_p (in green), except e_1 and e_2, since the path from their processes to p (leftmost process) goes through a closed process.

the tiles are pairwise disjoint and since $C_t = P$, by 2, they cover M. Hence, $M \in \mathbb{M}(\bigsqcup_{p\in P} T_p)$. It remains to show that each process $q \in P$ intersects at most ℓ tiles (where ℓ is defined in the statement of the theorem). By 1, if a tile T_p intersects process q then p is at distance at most k from q in $\mathcal{T}_M$. It is easy to check that the number of processes at distance at most k from q is at most ℓ.

We turn now to the definition of the tiles $(T_p)_{p\in P}$. Assume that process p is closed at subterm $s = \mathbb{M}(r)$ of t. We may assume that no blocks are merged on other processes and no other processes are closed. Hence, the set of closed processes is $C_s = C_r \uplus \{p\}$. By induction, T_q is already defined for all $q \in C_r$.

Let G be the set of events in $E_r \setminus F_r$ which are located on processes connected to p in $\mathcal{T}_{M_r}$ *via open processes of* M_r *only*: an event $e \in E_r \setminus F_r$ is in G if $\pi(e) = p$ or $\pi(e) = p_1 \longmapsto \cdots \longmapsto p_i = p$ in $\mathcal{T}_{M_r}$ for some $p_1, \ldots, p_i \in O_r$.

Let us show that G is $\lhd$-closed. So consider an edge $e \lhd f$ with one end in G, say $e \in G$. Assume $f \in F_r$. Since each tile is $\lhd$-closed, so is F_r and we deduce $e \in F_r$, a contradiction. Hence, $f \in E_r \setminus F_r$ and by 2 we have $\pi(f) \in O_r$. Moreover, $\pi(e)$ is connected to p via processes in O_r only, and thus so is $\pi(f)$. Therefore, $f \in G$.

We define T_p as the tile of M induced by G (see Fig. 12). We show that conditions 1, 2, and 3 are satisfied at s.

Proof of condition 1. By definition, T_p is disjoint from the existing tiles $(T_q)_{q\in C_r}$.

Moreover, since t is a k-DST, there are at most k open processes in any subterm, in particular $|O_r| \leq k$. We deduce that T_p contains at most k processes, each of which being at distance at most k from p in the topology $\mathcal{T}_M$.

It remains to show that T_p has split-width at most $k^2 + 2k$. By Lemma 28, it is enough to show that T_p has at most $k^2 + k$ blocks.

A block of T_p is an interval on some block of M_r, delimited on each side either by (a) an end of the said block, or (b) by a block of the split MSC M_{F_r} induced by F_r (see Fig. 13). This is the case since for every process $q \in O_r$, the set $\{e \in E_r \setminus F_r \mid \pi(e) = q\}$ is either entirely contained in G or disjoint from G.

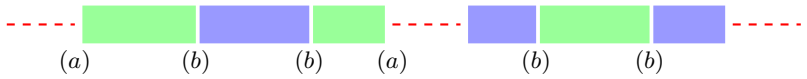


Fig. 13. Blocks of T_p (in green) and of M_{F_r} (in blue) on the third process of Figure 12.

Since M_r has at most k blocks, there are at most $2k$ blocks of T_p that are delimited on at least one side by the end of a block of M_r (like the first two ones in Figure 13). Moreover, by induction hypothesis 3 at r, M_{F_r} has at most k blocks on each process. Thus, on each process, there are at most $(k-1)$ blocks of T_p that are delimited on both sides by blocks of M_{F_r}. Counting all processes, T_p has at most $k(k-1)$ blocks of this kind. Hence, T_p has in total at most $2k + k(k-1) = k^2 + k$ blocks.

Proof of condition 2. Recall that $C_s = C_r \uplus \{p\}$. We have $F_s = F_r \uplus G$. By induction hypothesis, we know that $\{e \in E \mid \pi(e) \in C_r\} \subseteq F_r \subseteq E_r = E_s$. So we only need to show that $\{e \in E \mid \pi(e) = p\} \subseteq F_s = F_r \uplus G$. This is immediate, since by definition, G includes all events $e \in E_r \setminus F_r$ which are on process p. This means all events in $E \setminus F_r$ which are on process p, since p is closed at s.

Proof of condition 3. Observe from the definition of G that on any process q of T_p, the blocks of M_{F_s} are the blocks of M_r, and that on any other open process q, the blocks of M_F are the blocks of M_{F_r}. □

F Satisfiability and Model-Checking

Monadic second-order logic. The set of monadic second-order formulas (over $\mathcal{N}$) is denoted by $\mathrm{MSO}_{\mathcal{N}}$, and is given by the following syntax:

$$\begin{aligned}\varphi \ ::= \ & x = y \mid x \lhd y \mid x \to y \mid a!(x) \mid a?(x) \\ & \mid x \in X \mid \exists x.\varphi \mid \exists X.\varphi \mid \neg\varphi \mid \varphi \vee \varphi \\ & \mid x@u \mid u \overset{a\ b}{\longmapsto} v \mid u = v \mid u \in U \mid \exists u.\varphi \mid \exists U.\varphi\end{aligned} \tag{1}$$

where $a \in \mathcal{N}$, $x, y, \ldots$ and $u, v, \ldots$ denote first-order variables ranging respectively over events and processes, and $X, Y, \ldots$ and $U, V, \ldots$ denote second-order variables ranging respectively over sets of events and sets of processes.

$\mathrm{MSO}_{\mathcal{N}}$ formulas are interpreted over MSCs $M = (P, E, \to, \lhd, \pi, \lambda)$. Free variables are interpreted by a partial function $\mathcal{I}$ that maps a variable x to an event $\mathcal{I}(x) \in E$, and X to a set $\mathcal{I}(X) \subseteq E$. A variable u is mapped to a process $\mathcal{I}(u) \in P$, and U to a set $\mathcal{I}(U) \subseteq P$. We write $M, \mathcal{I} \models x@u$ when $\pi(\mathcal{I}(x)) = \mathcal{I}(u)$, and $M, \mathcal{I} \models u \overset{a\ b}{\longmapsto} v$ when $\mathcal{I}(u) \overset{a\ b}{\longmapsto} \mathcal{I}(v)$ in the observable topology $\mathcal{T}_M$ of M. The other cases are as expected. When φ is a closed formula, we simply write $M \models \varphi$ instead of $M, \mathcal{I} \models \varphi$.

Since $\mathcal{T}_M$ is entirely determined by the events of M, the part of the logic that involves processes is redundant. Any formula $\varphi \in \mathrm{MSO}_{\mathcal{N}}$ can be translated into an equivalent formula over the syntax defined by the first two lines of (1).

Lemma 30. *For any closed formula $\varphi \in \mathrm{MSO}_{\mathcal{N}}$, one can construct a tree automaton $\mathcal{A}^k_\varphi$ such that $L(\mathcal{A}^k_\varphi) \cap \mathsf{DST}^k_{\mathsf{msc}} = \{t \in \mathsf{DST}^k_{\mathsf{msc}} \mid M_t \models \varphi\}$.*

Proof. The construction is by induction on φ. The only non-trivial case for atomic formula is the $\to$ relation. It can be retrieved by a deterministic bottom-up tree automaton with $O(k^4)$ states, which accepts a valid k-DST t with two marked leaves e and f iff $e \to f$ in M_t. It remembers the blocks B and B' of the two marked leaves (that is, pairs in $[k]^2$, where (p, i) represents the i-th block on the p-th open process), and checks that there is no merge on the right of B or the left of B' until reaching a common ancestor where the two blocks are merged.

The inductive cases are treated with standard constructions: union for disjunction, complementation for negation, and projection for existential quantifications. □

Theorem 31. *For $\mathcal{C} \in \{\mathsf{SW}, \mathsf{EB}, \mathsf{CB}, \mathsf{TB}\}$, the satisfiability and model-checking problems are decidable:*

$\mathcal{C}$-MSO-Sat:	*$\mathcal{C}$-MSO-MC:*
Input: $\mathcal{N}$, $\varphi \in \mathrm{MSO}_{\mathcal{N}}$, i.	*Input: $\mathcal{N}$, $\varphi \in \mathrm{MSO}_{\mathcal{N}}$, $\mathcal{S}$, i.*
Question: $\exists M \in \mathcal{C}_i, M \models \varphi$?	*Question: $\forall M \in L(\mathcal{S}) \cap \mathcal{C}_i, M \models \varphi$?*

Proof. The bounded split-width satisfiability problem reduces to testing whether $L(\mathcal{A}^k_\varphi \cap \mathcal{A}^k_{\mathsf{msc}}) \neq \varnothing$, and the bounded split-width model-checking problem to testing whether $L(\mathcal{A}^k_{\mathcal{S}} \cap \mathcal{A}^k_{\neg\varphi} \cap \mathcal{A}^k_{\mathsf{msc}}) = \varnothing$. For the other classes, we add the corresponding automaton to the intersection, as for the non-emptiness problem. □

Propositional Dynamic Logic. The syntax of $\mathrm{ICPDL}_{\mathcal{N}}$ is given by

$$\Phi ::= \mathsf{E}\sigma \mid \Phi \vee \Phi \mid \neg\Phi \qquad \text{(global formula)}$$
$$\sigma ::= a! \mid a? \mid \sigma \vee \sigma \mid \neg\sigma \mid \langle\pi\rangle\sigma \qquad \text{(state formula)}$$
$$\pi ::= \lhd \mid \rightarrow \mid \mathsf{test}(\sigma) \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^* \mid \pi \cap \pi \mid \pi^{-1} \qquad \text{(path formula)}$$

where $a \in \mathcal{N}$. If intersection $\pi \cap \pi$ is not allowed, the fragment is PDL with converse (CPDL). If backward paths π^{-1} are not allowed the fragment is called PDL with intersection (IPDL). In simple PDL neither backward paths nor intersection are allowed.

State formula and path formula are respectively evaluated over events and pairs of events of an MSC. We only give a few cases:

- $M, e \models a!$ if $\lambda(e) = a!$
- $M, e \models \langle\pi\rangle\sigma$ if $M, (e, f) \models \pi$ and $M, f \models \sigma$ for some $f \in E$
- $M, (e, f) \models \lhd$ if $(e, f) \in \lhd$
- $M, (e, f) \models \mathsf{test}(\sigma)$ if $e = f$ and $M, e \models \sigma$.

The operations $+$, $\cdot$, *, $\cap$ and $^{-1}$ correspond respectively to union, composition, reflexive and transitive closure, intersection, and converse of the relations defined by path formulas.

MSC properties are specified using global formulas. For the base case, $M \models \mathsf{E}\sigma$ if there exists $e \in E$ such that $M, e \models \sigma$.

Lemma 32. *For any $\Phi \in \mathrm{CPDL}_{\mathcal{N}}$ (resp. $\Phi \in \mathrm{ICPDL}_{\mathcal{N}}$), one can construct in* PSpace *(resp.* ExpSpace*) a tree automaton A^k_Φ such that $L(\mathcal{A}^k_\Phi) \cap \mathsf{DST}^k_{\mathsf{msc}} = \{t \in \mathsf{DST}^k_{\mathsf{msc}} \mid M_t \models \Phi\}$.*

Proof. The construction is again by induction on the formula, but uses alternating two-way tree automata (A2A) rather than bottom-up automata. For any $\mathrm{CPDL}_{\mathcal{N}}$ (resp. $\mathrm{ICPDL}_{\mathcal{N}}$) formula Φ, we can construct an A2A of polynomial (resp. exponential) size that accepts a valid k-DST t iff $M_t \models \Phi$ (see [2] for a more detailed and very similar proof). It can then be translated into an exponentially larger (ordinary) tree automaton. □

Theorem 33. *For bounded split-width, context-bounded or tile-bounded MSCs (resp. existentially-bounded MSCs), the* CPDL *satisfiability and model-checking problems are* Exptime*-complete (resp.* Pspace*-complete), and the* ICPDL *satisfiability and model-checking problems are 2-*Exptime*-complete (resp.* Expspace*-complete). The model-checking problem is (in all cases) only polynomial in the number of states and transitions of the PCA.*

G Complexity lower bounds

Notice that reachability reduces to non-emptiness, and satisfiability to model-checking.

Lemma 34. *When the set of interfaces $\mathcal{N}$ and the bound k on split-width are fixed rather than part of the input, and such that $|\mathcal{N}| \geq 2$ and $k \geq 3$,* SW-Reachability *is* Ptime*-hard.*

Similarly, when $|\mathcal{N}| \geq 3$, a bound $k \geq 3$ on the number of contexts, and a bound $h \geq 1$ on the tree-width of the topologies are fixed, CB-Reachability *is* Ptime*-hard.*

Proof. The proof is by reduction from the non-emptiness problem for binary tree automata. Without loss of generality, we assume here that tree automata accept only trees where the root and every internal node have exactly two children. Given a tree automaton $\mathcal{A}$, we can construct in logarithmic space a PCA $\mathcal{S}$ over $\mathcal{N} = \{\mathsf{father}, \mathsf{child}_1, \mathsf{child}_2\}$ with a distinguished state accept, such that there exists a MSC $M \in \mathsf{SW}_3$ (resp. $M \in \mathsf{CB}_2$) and a run ρ of $\mathcal{S}$ on M with $\rho(e) = \mathsf{accept}$ for some event of M iff $L(\mathcal{A}) \neq \varnothing$. More precisely, we define $\mathcal{S}$ in such a way that for any connected MSC M, if there exist a run of $\mathcal{S}$ on M in which $\mathcal{S}$ reaches accept, then M has split-width at most 3 and is existentially 3-bounded, $\mathcal{T}_M$ is a tree, and there exists a labeling of $\mathcal{T}_M$ such that the labeled tree is accepted by $\mathcal{A}$; and such that conversely, for any $t \in L(\mathcal{A})$, there exists a MSC M with observable topology t, and such that there is a run ρ of $\mathcal{S}$ on M such that $\rho(e) = \mathsf{accept}$ for some event e of M.

The idea is that each process guesses (a) whether it is the a leaf, internal, or the root; and (b) a transition of $\mathcal{A}$. More precisely, if a process guesses that it is a leaf, then it chooses a transition of $\mathcal{A}$ of the form $\xrightarrow{a} q$, and sends q to its father. If a process guesses that it is internal, it must start by receiving some message q_1 from its left child, then some q_2 from its right child. It then guesses a transition $q_1, q_2 \xrightarrow{a} q$ of $\mathcal{A}$ and sends q to its father. If a process guesses that it is a root, then it must receive some q_1 from its left child, then some q_2 from its right child, such that there exists a transition of $\mathcal{A}$ of the form $q_1, q_2 \xrightarrow{a} q$ with q accepting. In that case, it goes to state accept.

Let M be a connected MSC which allows for a run ρ of $\mathcal{S}$ such that $\rho(e) = \mathsf{accept}$ for some e. By definition of $\mathcal{S}$, $\mathcal{T}_M$ only contains edges of the form $p \xmapsto{\mathsf{father\ \ child}_i} q$. Moreover, there is no edge $\pi(e) \xmapsto{\mathsf{father\ \ child}_i} q$. Thus, $\mathcal{T}_M$ is indeed a tree. Clearly, M is 3-context-bounded. It also has split-width at most 3: we can disconnect M by splitting the edge of its root, which create two blocks. In both of the resulting components, only one block (one event) remains on the root. We can disconnect the corresponding message by splitting the first process edge of the child node, leading to a total of 3 blocks. We are then back at the initial situation, except that the root of the subtree is already open. □

Lemma 35. *When $|\mathcal{N}| \geq 2$ and $k \geq 3$ are fixed, the problem* EB-Reachability *is* Nlogspace*-hard.*

Proof. The proof is the same as that of Lemma 34, but using word automata instead of tree automata, and pipelines instead of trees for topologies. □

Lemma 36. SW-Reachability *and* CB-Reachability *are* Exptime-*hard.*

Proof. The proof is similar to that of Lemma 34, but we reduce the intersection problem for k tree automata. Each process simply repeats k times the behaviour described above. (More details can be found in the proof of Lemma 37, which uses the same kind of reduction but for word automata instead of tree automata.) Each process has at most $3k$ events, hence the MSC is $3k$-context bounded, hence has split-width at most $6k + 2$. □

Lemma 37. EB-Reachability *is* Pspace-*hard.*

Proof. The proof is by reduction from the intersection problem for finite automata. Given finite automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$, we can construct in polynomial time a PCA $\mathcal{S}$ over $\mathcal{N} = \{\mathsf{left}, \mathsf{right}\}$, with a distinguished state accept, such that $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k) = \varnothing$ iff there exist $M \in \mathsf{EB}_{(k+2)}$ and a run ρ of $\mathcal{S}$ on M with $\rho(e) = \mathsf{accept}$ for some event of M. The idea is that if $\mathcal{S}$ accept some connected MSC M, then there exists a word $w \in L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k)$ such that $\mathcal{T}_M$ is a pipeline with $|w|$ processes, and conversely.

Each process guesses a transition for each $\mathcal{A}_i$ using the same (guessed) letter, and sends it to its right neighbor. The (guessed) leftmost process must choose transitions starting in initial states, and the (guessed) rightmost process checks for all $\mathcal{A}_i$ that it can choose a transition ending in an accepting state. It then goes to state accept. More precisely, $\mathcal{S}$ only accepts MSCs of the form described in Figure 3 (or disjoint unions of such MSCs), with an arbitrary number of processes, but k events on the leftmost process. The fact that the topology does not contain any cycle is ensured by the fact that at least one process has no right neighbor (the process reaching accept).

Any MSC accepted by $\mathcal{S}$ is existentially $(k + 2)$-bounded: the linearization consists in taking first the events of the leftmost process (in order), then the events of its right neighbor, and so on. □

Lemma 38. SW-CPDL-Sat *and* CB-CPDL-Sat *are* Exptime-*hard, and* SW-ICPDL-Sat *and* CB-ICPDL-Sat *are 2-*Exptime-*hard.*

Proof. The proofs are by reduction from the satisfiability problems for (I)CPDL over binary trees with labels in some finite alphabet A.

First, there is a PDL formula Φ_0 over $\mathcal{N} = A \times \{\mathsf{father}, \mathsf{child}_1, \mathsf{child}_2\}$ that recognizes the set of MSCs M such that $\mathcal{T}_M$ is a tree (and each process has at most 3 interfaces, of the form $(x, \mathsf{father}), (x, \mathsf{child}_1), (x, \mathsf{child}_2)$ for some $x \in A$), and each process is composed of first a receive from its father (if any), then a send to its left child (if any), then a send to its right child (if any). We let t_M be the tree whose nodes are those of $\mathcal{T}_M$, and such that the label x of each node is determined by its set of interfaces in $\mathcal{T}_M$. This defines a bijection between the set of labeled binary trees and $L(\Phi_0)$. Moreover, for any $k \geq 3$, $h \geq 1$, $L(\Phi_0) \subseteq \mathsf{SW}_k$ and $L(\Phi_0) \subseteq \mathsf{CB}_{k,h}$.

Next, any (I)CPDL formula Ψ over trees can be translated in polynomial time into an (I)CPDL formula Φ over MSCs, such that for any $M \in L(\Phi_0)$, $M \models \Phi \iff \mathcal{T}_M \models \Psi$. The formula Ψ is constructed inductively, by interpreting a node in t_M by the first event of the corresponding process in M. For instance,

- a state formula $\sigma \equiv x$ (for some $x \in A$) over trees is translated into the state formula $((x, \mathsf{father})? \vee (x, \mathsf{child}_1)! \vee (x, \mathsf{child}_2)!) \wedge \neg\langle \rightarrow^{-1} \rangle \top$ over MSCs.
- a path formula $\pi = \downarrow_1$ over trees, denoting a path between a node and its left child, is translated into the path formula $\rightarrow^* \cdot \mathsf{test}(\bigvee_{x \in A}(x, \mathsf{child}_1)!) \cdot \lhd$ over MSCs.

We then have $L(\Phi_0 \wedge \Phi) \neq \varnothing \iff L(\Psi) \neq \varnothing$. □

Lemma 39. EB-CPDL-SAT *is* PSPACE*-hard.* EB-ICPDL-SAT *is* EXPSPACE*-hard.*

Proof. The proof is an in Lemma 38, but using words instead of trees. □