Parameterized Verification of Communicating Automata under Context Bounds

Benedikt Bollig, Paul Gastin, Jana Schubert

Parameterized Verification of Communicating Automata under Context Bounds

Benedikt Bollig[1], Paul Gastin[1], and Jana Schubert[2]

[1] LSV, ENS Cachan & CNRS
[2] Fakultät für Informatik, TU Dresden

Abstract. We study the verification problem for parameterized communicating automata (PCA), in which processes synchronize via message passing. A given PCA can be run on any topology of bounded degree (such as pipelines, rings, or ranked trees), and communication may take place between any two processes that are adjacent in the topology. Parameterized verification asks if there is a topology from a given topology class that allows for an accepting run of the given PCA. In general, this problem is undecidable even for synchronous communication and simple pipeline topologies. We therefore consider context-bounded verification, which restricts the behavior of each single process. For several variants of context bounds, we show that parameterized verification over pipelines, rings, and ranked trees is decidable. Our approach is automata-theoretic and uniform. We introduce a notion of graph acceptor that identifies those topologies allowing for an accepting run. Depending on the given topology class, the topology acceptor can then be restricted, or adjusted, so that the verification problem reduces to checking emptiness of finite automata or tree automata.

1 Introduction

Communicating automata (CA) are a fundamental and well studied model of parallel systems [7]. They consist of finite-state machines that exchange messages over channels determined by a fixed and known communication topology. CA are known to be Turing equivalent so that even basic problems of formal verification such as reachability are undecidable. Therefore, modifications and restrictions have been considered which bring back decidability. Reachability is decidable, for example, when the analysis is restricted to executions with a fixed maximum number of pending messages, or when channels are lossy [2].

In some contexts such as ad-hoc networks, multi-core programming, or communication-protocol verification, assuming a fixed and known communication topology is not appropriate. Lately, there has been a lot of (ongoing) research in the area of *parameterized* verification [1, 3, 8, 12], which aims to validate a given system independently of the number of processes and the communication topology. A lot of different models of such systems have been proposed (cf. [11] for a recent survey). In this paper, we investigate the reachability problem for parametrized communicating automata (PCAs). A PCA is a collection of finite

automata that can be plugged into *any* communication topology of bounded degree. PCAs have recently been introduced to initiate a logical study of parameterized systems [5]. Their verification problem has not been considered. Roughly, it can be stated as follows: Given a PCA and a regular set $\mathfrak{T}$ of pipeline, ring, or tree topologies, is there a topology $\mathcal{T} \in \mathfrak{T}$ such that $\mathcal{A}$ has an accepting run on $\mathcal{T}$? Here, "regular" means given by some finite automaton (for pipelines and rings) or tree automaton (for tree topologies), which is part of the input. Note that there is also a universal variant of that problem, and our decision procedures will take care of that case as well.

We actually consider a restriction of PCAs with rendez-vous synchronization, albeit distinguishing between send and receive events. This considerably simplifies the presentation, but the overall approach can be extended to systems with asynchronous bounded channels. Note that rendez-vous communication can also be seen as an underapproximation of the latter.

While bounding the channel capacity or imposing rendez-vous communication bring back decidability of reachability for CA with fixed communication topology, this is no longer true in the case of PCA. For various other (undecidable) models of concurrent systems, decidability is achieved by introducing a context (or "phase") bound, limiting the part of the model simulating synchronization or communication of concurrent processes [6, 13, 15, 16]. We adopt the general approach, but introduce new definitions of contexts that are suitable for our setting. An *interface-context* restricts communication of a process to one neighbor in the topology (e.g., the left neighbor in the pipeline). Another context type separates send from receive events while restricting reception to one interface. Note that, in both cases, there may still be an unbounded number of switches between two given threads.

We show that context-bounded parameterized verification is decidable: it is PSPACE-complete for pipelines and rings, and EXPTIME-complete for ranked trees. Our decidablity proof is automata-theoretic and uniform. We transform a given PCA $\mathcal{A}$, in several steps, into a graph acceptor that recognizes the set of *acyclic* topologies allowing for an accepting run of $\mathcal{A}$. This solves our problem for pipelines and trees. For rings, an additional adjustment is needed, which rules out cyclic behaviors that the graph acceptor is not able to detect on its own.

Related Work. Parameterized verification can be classified into verification of multithreaded programs running on a single core, and protocol verification. Context-bounded verification for systems consisting of an unbounded number of threads has already been considered [4,14]. In [4], a model with process creation is presented, in which a context switch is observed when an active thread is interrupted and resumed. In [14], an unbounded number of threads are scheduled in several rounds. In both cases, the context bound does not impose a bound on the number of threads. However, every thread will be resumed and become active a bounded number of times, which is not the case in our framework.

On the other hand, protocol verification is based on the concept of independent (finite-state) processes communicating over a network-like structure. Due to the absence of a global scheduler, the above context definition is not adequate

for protocol verification. A versatile framework for parameterized verification, capturing rendez-vous communication in pipelines, rings, and trees, is presented in [1]. The verification problem is phrased in terms of minimal bad configurations, which does not necessitate context bounds. Motivated by ad-hoc networks, [8] considers systems modeled by finite automata that communicate in a broadcast or unicast manner. In the case of unicast communication, the recipient is chosen nondeterministically from the set of neighbors, which is incomparable with the unicast communication of PCAs. Direction-aware token-passing systems [3,9,10] can be modeled in our framework as far as bounded-degree structures such as rings are concerned. To the best of our knowledge, neither context bounds nor the PCA model have been considered yet for protocol verification.

Outline. Section 2 recapitulates basic notions such as words and finite (tree) automata. In Section 3, we introduce topologies, PCAs, and several context-bounded verification problems. Section 4 is the heart of our automata-theoretic approach, where we first translate PCA into a sort of cellular automaton and, in a second step, into topology acceptors. These constructions will be exploited, in Section 5, to solve the parameterized verification problem for the classes of pipelines, rings, and ranked trees. Missing proofs can be found in the appendix.

2 Preliminaries

For $n \in \mathbb{N}$, we set $[n] := \{1, \dots, n\}$. Let $\mathbb{A}$ be an alphabet, i.e., a nonempty finite set. The set of finite words over $\mathbb{A}$ is denoted by $\mathbb{A}^*$, which includes the empty word ε. The concatenation of words $w_1, w_2 \in \mathbb{A}^*$ is denoted by $w_1 \cdot w_2$ or $w_1.w_2$. Given an index set I and a tuple $a = (a_i)_{i \in I} \in \mathbb{A}^I$, we write $a|_i$ to denote a_i.

A *finite automaton* over $\mathbb{A}$ is a tuple $\mathcal{B} = (S, \Longrightarrow, \iota, F)$ where S is the finite set of states, $\iota \in S$ is the initial state, $F \subseteq S$ is the set of final states, and $\Longrightarrow \subseteq S \times \mathbb{A} \times S$ is the transition relation. We write $s \overset{a}{\Longrightarrow} s'$ instead of $(s, a, s') \in \Longrightarrow$. A run of $\mathcal{B}$ on a word $w = a_1 \dots a_n \in \mathbb{A}^*$ is a sequence $s_0 s_1 \dots s_n \in S^*$ of states such that $s_0 = \iota$ and $s_{i-1} \overset{a_i}{\Longrightarrow} s_i$ for all $i \in [n]$. The run is accepting if $s_n \in F$. Finally, the language of $\mathcal{B}$ is defined as $L(\mathcal{B}) := \{w \in \mathbb{A}^* \mid \text{there is an accepting run of } \mathcal{B} \text{ on } w\}$.

For trees, we fix a (maximal) *rank* $r \in \mathbb{N}$. An *r-tree* over $\mathbb{A}$ is a pair (V, π) where V is a nonempty finite prefix-closed subset of $\{1, \dots, r\}^*$, and $\pi : V \to \mathbb{A}$ is a labeling function. The set V is the set of nodes of the tree, and ε is its root. For $u \in V$ and $l \in [r]$ with $u.l \in V$, we say that $u.l$ is the l-th child of u. An *r-tree automaton* over $\mathbb{A}$ is a tuple $\mathcal{B} = (S, \Delta, F)$ where S is the finite set of states, $F \subseteq S$ is the set of final states, and $\Delta \subseteq S \times \mathbb{A} \times (S \uplus \{\bot\})^r$ is the transition relation. A run of $\mathcal{B}$ on an r-tree (V, π) is a mapping $\rho : V \to S$ such that, for all $u \in V$, $(\rho(u), \pi(u), (s_l)_{l \in [r]}) \in \Delta$ where $s_l = \rho(u.l)$ if $u.l \in V$, and $s_l = \bot$ if $u.l \notin V$. The run is accepting if $\rho(\varepsilon) \in F$. By $L(\mathcal{B})$, we denote the set of r-trees accepted by $\mathcal{B}$.

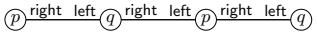

Fig. 1. Pipeline

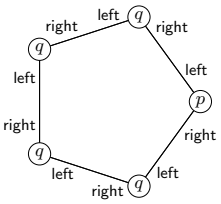

Fig. 2. Ring

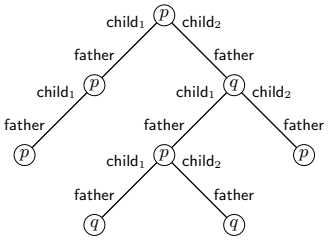

Fig. 3. Tree

3 Parameterized Communicating Automata

In this section, we introduce our model of a communicating system that can be run on arbitrary topologies of bounded degree.

Topologies. A topology is a graph, whose nodes are connected via interfaces. The idea is that each node runs a finite-state process (of type $p, q, \ldots$). In Figure 1, for example, nodes are arranged in a pipeline, which allows a process to communicate with a left and a right neighbor (if they exist). When a node u emits a message m via its interface right, then m can be received by the neighbor on the right of u, using interface left. Let $\mathcal{N} = \{a, b, c, \ldots\}$ and $\mathcal{P} = \{p, q, \ldots\}$ be finite sets of *interface names* and *process types*, respectively.

Definition 1. *A* topology *over $\mathcal{N}$ and $\mathcal{P}$ is a tuple $\mathcal{T} = (V, \nu, \pi)$ where V is the nonempty finite set of* nodes *(or* processes*), $\pi : V \to \mathcal{P}$ associates with every node a process type, and $\nu : V \times \mathcal{N} \rightharpoonup V$ is a partial mapping. Intuitively, $\nu(u, a) = v$ means that the interface a of u points to v. We suppose that, for all $u \in V$, there is at last one $a \in \mathcal{N}$ such that $\nu(u, a)$ is defined. Moreover, we require that $\nu(u, a) = v$ implies (1) $u \neq v$, (2) $\nu(v, b) = u$ for some $b \in \mathcal{N}$, and (3) $\nu(u, a') = v'$ implies $[a = a'$ iff $v = v']$, for all $a' \in \mathcal{N}$ and $v' \in P$.*

We write $u \overset{a\ b}{\longmapsto} v$ if $\nu(u, a) = v$ and $\nu(v, b) = u$, and we write $u \longmapsto v$ if $u \overset{a\ b}{\longmapsto} v$ for some $a, b \in \mathcal{N}$. We call $\mathcal{T}$ *acyclic* if the undirected graph $(V, \longmapsto)$ is acyclic. This paper will focus on three topology classes:

Pipelines. A *pipeline* over a nonempty finite set $\mathcal{P}$ of process types is a topology over $\mathcal{N} = \{\mathsf{left}, \mathsf{right}\}$ and $\mathcal{P}$. It is of the form $\mathcal{T} = (\{1, \ldots, n\}, \nu, \pi)$, with $n \geq 2$, such that $\nu(i, \mathsf{right}) = i + 1$ and $\nu(i + 1, \mathsf{left}) = i$ for all $i \in \{1, \ldots, n - 1\}$, and $\nu(1, \mathsf{left})$ and $\nu(n, \mathsf{right})$ are both undefined. A finite automaton $\mathcal{B}$ over $\mathcal{P}$ can be seen as a pipeline recognizer. Indeed, a pipeline is uniquely given by the sequence $\pi(1) \ldots \pi(n) \in \mathcal{P}^*$. So, we let $L_{pipe}(\mathcal{B})$ denote the set of pipelines $(\{1, \ldots, n\}, \nu, \pi)$ over $\mathcal{P}$ such that $\pi(1) \ldots \pi(n) \in L(\mathcal{B})$. Instead of $\mathcal{B}$, we may use a classical regular expression. An example pipeline is depicted in Figure 1. It is uniquely given by the word *pqpq*.

Rings. A *ring* over $\mathcal{P}$ is a topology over $\mathcal{N} = \{\mathsf{left}, \mathsf{right}\}$ and $\mathcal{P}$ of the form $\mathcal{T} = (\{1, \ldots, n\}, \nu, \pi)$, with $n \geq 3$, where $\nu(i, \mathsf{right}) = (i \bmod n) + 1$ and

$\nu((i \bmod n) + 1, \mathsf{left}) = i$ for all $i \in [n]$. Similarly to pipelines, a finite automaton $\mathcal{B}$ over $\mathcal{P}$ can be used as a *ring* recognizer: we let $L_{ring}(\mathcal{B})$ denote the set of rings $(\{1, \ldots, n\}, \nu, \pi)$ over $\mathcal{P}$ such that there is $i \in [n]$ satisfying $\pi(i) \ldots \pi(n)\pi(1) \ldots \pi(i-1) \in L(\mathcal{B})$. This takes into account that, a priori, rings do not have an "initial" node. Figure 2 depicts a ring with five nodes.

Trees. For $r \geq 2$, an *r-tree topology* over $\mathcal{P}$ is a topology $\mathcal{T} = (V, \nu, \pi)$ over $\{\mathsf{father}, \mathsf{child}_1, \ldots, \mathsf{child}_r\}$ and $\mathcal{P}$ such that (V, π) is an r-tree over $\mathcal{P}$, $\nu(\varepsilon, \mathsf{father})$ is undefined, and for all $u \in V$ and $l \in [r]$, we have (1) $u.l \in V$ implies $\nu(u, \mathsf{child}_l) = u.l$ and $\nu(u.l, \mathsf{father}) = u$, and (2) $u.l \notin V$ implies that $\nu(u, \mathsf{child}_l)$ is undefined. An r-tree automaton $\mathcal{B}$ over $\mathcal{P}$ can be seeen as a recognizer for tree topologies: we write $L_{tree}(\mathcal{B})$ for the set of r-tree topologies (V, ν, π) such that $(V, \pi) \in L(\mathcal{B})$. A sample 2-tree topology is depicted in Figure 3.

The Automata Model. Next, we introduce our system model. As suggested above, a parameterized communicating automaton is a collection of finite-state processes whose actions refer to an interface. Unless stated otherwise, we assume that $\mathcal{N}$ is a *fixed* nonempty finite set of interface names (or, simply, interfaces).

Definition 2. *A* parameterized communicating automaton (PCA) *over $\mathcal{N}$ is a tuple $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p \in \mathcal{P}})$ where $\mathcal{P}$ is a nonempty finite set of process types, Msg is a nonempty finite set of* messages*, and $\mathcal{A}_p$ is a finite automaton over the set of actions $\Sigma_{\mathcal{A}} := \{a!m\,,\ a?m \mid a \in \mathcal{N} \text{ and } m \in Msg\}$, for every $p \in \mathcal{P}$.*

A *pipeline PCA* or *ring PCA* is a PCA over $\{\mathsf{left}, \mathsf{right}\}$. Moreover, for $r \geq 2$, an *r-tree PCA* is a PCA over $\{\mathsf{father}, \mathsf{child}_1, \ldots, \mathsf{child}_r\}$.

The idea is the following: When $\mathcal{A}$ is run on a topology (V, ν, π) with adjacent processes $u \overset{a\ b}{\longmapsto} v$, then u runs a copy of $\mathcal{A}_{\pi(u)}$ and can emit a message m through interface a by executing $a!m$. Process v receives the message if it is ready to execute $b?m$. We assume that communication is by rendez-vous, i.e., messages are received instantaneously.

For convenience, we write Σ instead of $\Sigma_{\mathcal{A}}$. Sometimes, we will even mention Σ without any reference to $\mathcal{A}$. However, notice that the alphabet depends on a PCA (more precisely, on $\mathcal{N}$ and a set of messages). Let $\Sigma_! := \{a!m \mid a \in \mathcal{N} \text{ and } m \in Msg\}$ and let $\Sigma_?$ be defined accordingly. These sets are further refined to $\Sigma_{a!}$ and $\Sigma_{a?}$, containing only those actions that refer to interface $a \in \mathcal{N}$.

Semantics of PCAs. Let $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p \in \mathcal{P}})$ be a PCA over $\mathcal{N}$, with $\mathcal{A}_p = (S_p, \Longrightarrow_p, \iota_p, F_p)$ for all $p \in \mathcal{P}$. The PCA $\mathcal{A}$ can be run on any topology $\mathcal{T} = (V, \nu, \pi)$ over $\mathcal{N}$ and $\mathcal{P}$. The semantics of $\mathcal{A}$ wrt. $\mathcal{T}$ is a finite automaton $[\![\mathcal{A}]\!]^{\mathcal{T}} = (S, \Longrightarrow, \iota, F)$ over $\Sigma^{\mathcal{T}} \subseteq (\Sigma \cup \{\varepsilon\})^V$. The alphabet $\Sigma^{\mathcal{T}}$ contains, for all $v \overset{a\ b}{\longmapsto} v'$ and $m \in Msg$, the tuple $\langle v, m, v' \rangle := (\sigma_u)_{u \in V}$ where $\sigma_v = a!m$, $\sigma_{v'} = b?m$, and $\sigma_u = \varepsilon$ for all $u \in V \setminus \{v, v'\}$. For $W = \gamma_1 \ldots \gamma_n \in (\Sigma^{\mathcal{T}})^*$ and $u \in V$, we define the projection of W to u as $W|_u := (\gamma_1|_u) \cdot \ldots \cdot (\gamma_n|_u) \in \Sigma^*$.

For $u \in V$, we write $\mathcal{A}_u, S_u, \Longrightarrow_u, \iota_u, F_u$ for $\mathcal{A}_{\pi(u)}, S_{\pi(u)}, \Longrightarrow_{\pi(u)}, \iota_{\pi(u)}, F_{\pi(u)}$, respectively. The set of states of $[\![\mathcal{A}]\!]^{\mathcal{T}}$ is $S = \prod_{u \in V} S_u$, keeping track of the local state of every process in the topology. Accordingly, the initial state is

$\iota = (\iota_u)_{u \in V}$, and the set of final states is $F = \prod_{u \in V} F_u$. The transition relation $\Longrightarrow \subseteq S \times \Sigma^{\mathcal{T}} \times S$ is defined as follows. Let $\mathrm{s} = (s_u)_{u \in V} \in S$, $\mathrm{s}' = (s'_u)_{u \in V} \in S$, and $\sigma = (\sigma_u)_{u \in V} \in \Sigma^{\mathcal{T}}$. Then, $\mathrm{s} \overset{\sigma}{\Longrightarrow} \mathrm{s}'$ if, for all $u \in V$, we have that $\sigma_u \neq \varepsilon$ implies $s_u \overset{\sigma_u}{\Longrightarrow}_u s'_u$, and $\sigma_u = \varepsilon$ implies $s_u = s'_u$. The language of $\mathcal{A}$ wrt. $\mathcal{T}$ is defined as $L(\mathcal{A}, \mathcal{T}) := L([\![\mathcal{A}]\!]^{\mathcal{T}})$.

Example 1. Consider the PCA $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p, \mathcal{A}_{\bar{p}}, \mathcal{A}_q, \mathcal{A}_{\bar{q}}))$ over the set of interfaces $\mathcal{N} = \{\mathsf{left}, \mathsf{right}\}$, where $\mathcal{P} = \{p, \bar{p}, q, \bar{q}\}$, $Msg = \{m\}$, and the local languages are given as follows:

$$L(\mathcal{A}_p) = \{(\mathsf{right}!m)\} \qquad L(\mathcal{A}_q) = \{(\mathsf{left}?m)(\mathsf{right}!m)\}$$
$$L(\mathcal{A}_{\bar{p}}) = \{(\mathsf{left}?m)\} \qquad L(\mathcal{A}_{\bar{q}}) = \{(\mathsf{right}!m)(\mathsf{left}?m)\}$$

Let Q be the regular expression $(q + \bar{q})^*$. For all pipelines $\mathcal{T}$ over $\mathcal{P}$, we have $L(\mathcal{A}, \mathcal{T}) \neq \emptyset$ iff $\mathcal{T} \in L_{pipe}((pQ\bar{p})^*)$. In particular, for $n \geq 2$ and the pipeline $\mathcal{T}$ given by $pq^{n-2}\bar{p}$, we have $L(\mathcal{A}, \mathcal{T}) = \{\langle 1, m, 2\rangle\langle 2, m, 3\rangle \ldots \langle n-1, m, n\rangle\}$. When we consider rings instead, the reasoning is less evident. Note that $L(\mathcal{A}, \mathcal{T}) = \emptyset$ for all $\mathcal{T} \in L_{ring}(q^* + \bar{q}^*)$: though two successive processes qq or $\bar{q}\bar{q}$ match locally, meaning that an open send encounters an open receive, closing a sequence q^n or $\bar{q}^n$ towards a ring is not possible due to the causal dependencies that are created. In q^n, for example, the receive that remains open is always scheduled before the remaining open send. Thus, matching both will create a cyclic dependency and not lead to a valid run of $\mathcal{A}$. However, pathologic dependencies are "resolved" in a ring over $\{q, \bar{q}\}$ whenever it contains $q\bar{q}$ (which implies that there is also $\bar{q}q$ when the ring is seen as a cyclic word). We actually have, for all rings $\mathcal{T}$ over $\mathcal{P}$, that $L(\mathcal{A}, \mathcal{T}) \neq \emptyset$ iff $\mathcal{T} \in L_{ring}((pQ\bar{p})^* + Qq\bar{q}Q)$. Detecting cyclic dependencies will be one challenge when we tackle the verification problem for rings in Section 5.

Context-Bounded Parameterized Emptiness. Next, we define several variants of contexts, which restrict the behavior of each process of a PCA. A word $w \in \Sigma^*$ is called an

- ($\mathsf{s}\oplus\mathsf{r}$)-context if $w \in \Sigma_!^* \cup \Sigma_?^*$,
- ($\mathsf{s1}+\mathsf{r1}$)-context if $w \in (\Sigma_{a!} \cup \Sigma_{b?})^*$ for some $a, b \in \mathcal{N}$,
- ($\mathsf{s}\oplus\mathsf{r1}$)-context if $w \in \Sigma_!^* \cup \Sigma_{a?}^*$ for some $a \in \mathcal{N}$, and
- intf-context if $w \in (\Sigma_{a!} \cup \Sigma_{a?})^*$ for some $a \in \mathcal{N}$.

The case $\mathsf{s1}\oplus\mathsf{r}$ ($w \in \Sigma_{a!}^* \cup \Sigma_?^*$ for some $a \in \mathcal{N}$) is symmetric to $\mathsf{s}\oplus\mathsf{r1}$, and we only consider the latter. All results hold verbatim when we replace $\mathsf{s}\oplus\mathsf{r1}$ with $\mathsf{s1}\oplus\mathsf{r}$.

Let $k \geq 1$ and $t \in \{\mathsf{s}\oplus\mathsf{r}, \mathsf{s1}+\mathsf{r1}, \mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$ be a context type. We say that $w \in \Sigma^*$ is (k,t)-bounded if there are $w_1, \ldots, w_k \in \Sigma^*$ such that $w = w_1 \cdot \ldots \cdot w_k$ and w_i is a t-context, for all $i \in [k]$. The set of all (k,t)-bounded words (over a fixed Σ) is denoted by $\mathbb{W}_{(k,t)}$. For a PCA $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p \in \mathcal{P}})$ and a topology $\mathcal{T} = (V, \nu, \pi)$, we define $L_{(k,t)}(\mathcal{A}, \mathcal{T}) := \{W \in L(\mathcal{A}, \mathcal{T}) \mid W|_u \in \mathbb{W}_{(k,t)} \text{ for all } u \in V\}$. Note that $\mathbb{W}_{(k,t)}$ is a regular word language that is recognized by a finite automaton $\mathcal{B}_{(k,t)}$ whose number of states is linear in k and at most quadratic in $|\mathcal{N}|$ (but linear for the decidable cases of t). Let $\mathcal{A}'$ be the PCA

PIPELINE-NONEMPTINESS(t)	
I:	pipeline PCA $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p\in\mathcal{P}})$ $k \geq 1$; finite automaton $\mathcal{B}$ over $\mathcal{P}$
Q:	$L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset$ for some $\mathcal{T} \in L_{pipe}(\mathcal{B})$?

RING-NONEMPTINESS(t)	
I:	ring PCA $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p\in\mathcal{P}})$ $k \geq 1$; finite automaton $\mathcal{B}$ over $\mathcal{P}$
Q:	$L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset$ for some $\mathcal{T} \in L_{ring}(\mathcal{B})$?

TREE$_r$-NONEMPTINESS(t)	
I:	r-tree PCA $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p\in\mathcal{P}})$ $k \geq 1$; r-tree automaton $\mathcal{B}$ over $\mathcal{P}$
Q:	$L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset$ for some $\mathcal{T} \in L_{tree}(\mathcal{B})$?

	s⊕r1	intf
pipelines	PSPACE-c	PSPACE-c
rings	PSPACE-c	PSPACE-c
trees	EXPTIME-c	EXPTIME-c

Table 1. Context-bounded nonemptiness problems and summary of results

$(\mathcal{P}, Msg, (\mathcal{A}_p \times \mathcal{B}_{(k,t)})_{p\in\mathcal{P}})$ where $\mathcal{A}_p \times \mathcal{B}_{(k,t)}$ is the classical product of two finite automata. It is easy to see that $L_{(k,t)}(\mathcal{A}, \mathcal{T}) = L(\mathcal{A}', \mathcal{T})$. This means that the context-bound restriction can be built into the PCA.

Applying the definitions to the PCA $\mathcal{A}$ from Example 1, we have $L(\mathcal{A}, \mathcal{T}) = L_{(2,\mathsf{s}\oplus\mathsf{r1})}(\mathcal{A}, \mathcal{T}) = L_{(2,\mathsf{intf})}(\mathcal{A}, \mathcal{T})$ for all topologies over $\{\mathsf{left}, \mathsf{right}\}$ and $\{p, \bar{p}, q, \bar{q}\}$.

For $t \in \{\mathsf{s}\oplus\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}, \mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$, we consider the problems listed in Table 1. Note that the context bound k is always part of the input. We assume that k is encoded in unary. The table also gives a summary of the positive results of the paper. For some context types, however, all problems are undecidable.

Theorem 1. *All problems listed in Table 1 are undecidable for $t \in \{\mathsf{s}\oplus\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}\}$, even when we restrict to one context for each process.*

Proof (sketch). Figures 4 and 5 demonstrate how to generate grid-like structures of arbitrary height i and width j, using only one context on each single process. Figure 4, for example, visualizes an execution of the form

$$(\langle 1, m_{(1,1)}, 2\rangle\langle 2, m_{(1,2)}, 3\rangle \ldots \langle j, m_{(1,j)}, j+1\rangle) \ldots (\langle 1, m_{(i,1)}, 2\rangle\langle 2, m_{(i,2)}, 3\rangle \ldots \langle j, m_{(i,j)}, j+1\rangle)\,.$$

The idea is now to simulate a Turing machine, using the (unbounded) vertical dimension to encode its tape, which changes along the (unbounded) horizontal line. More precisely, the leftmost process generates a sequence of messages $(m_{(1,1)}, \ldots, m_{(i,1)})$ that corresponds to the initial configuration with arbitrarily many cells. Each further process may locally change that configuration while passing it to its right neighbor, and so on. In the case of $\mathsf{s}\oplus\mathsf{r}$, the transfer of a configuration is sometimes accomplished by a receive context. Here, reception is blocking if the messages coming from the right neighbor do not correspond to the next configuration. Obviously, the encoding also works for rings as well as for the (more general) trees. It is important to note that it does not matter if the semantics is via rendez-vous (as it is our case) or asynchronous. □

4 From PCAs to Topology Acceptors

In this section, we lay the basis for a uniform approach to parameterized verification under context bounds of type $\mathsf{s}\oplus\mathsf{r1}$ or intf. We first show that, assuming

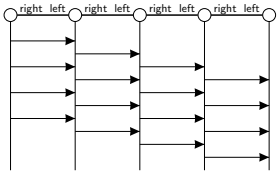

Fig. 4. Undecidability for s1+r1

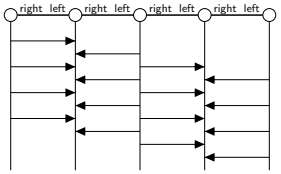

Fig. 5. Undecidability for s⊕r

a context bound of type s⊕r1 or intf, a given PCA can be simulated by a sort of *cellular automaton*, which works similarly to PCAs but performs only a bounded number of steps on each process. The idea is that single contexts are *summarized* to one step of maximally $|\mathcal{N}| + 1$ processes. This transformation works for both context types, with only minor differences.

In a further step, which is then independent of a context type, the cellular automaton is translated into a *topology acceptor*, which no longer has an operational semantics but a "static" one: a run is a tiling of a topology with transitions. When we restrict a topology acceptor to pipelines or tree topologies, we obtain a finite automaton or tree automaton, respectively. Thus, this transformation solves our verification problems for tree and pipeline topologies.

Unfortunately, the topology acceptor is only guaranteed to be correct for acyclic topologies. It may actually accept cyclic topologies that do not admit an accepting run of the given PCA. Nevertheless, for the class of rings, we can enrich a topology acceptor with additional information that helps keeping track of interdependencies between contexts of different processes (cf. Example 1). This will allow us to build a finite automaton recognizing the unfoldings of rings having an accepting PCA run (Lemma 3).

We explain the general idea by means of Figure 6. Assume context type s⊕r1, which is the more difficult case. The figure depicts an execution (in fact, a set of "order"-equivalent executions) that is $(3, \mathsf{s}\oplus\mathsf{r1})$ bounded. Processes 1, 2, and 3 use three contexts, while 4 and 5 can do with a single one. The dotted areas on a process line suggest that we actually consider an arbitrary number of actions. Our aim is to aggregate these unboundedly many actions in a bounded number of summaries Si so that a finite automaton can read the pipeline (i.e., the word *pppqq*) from left to right, while verifying that the summaries can be glued together towards an accepting run of the given PCA.

As process 4 alternates between sending to 3 and sending to 5, its summaries have to include the behavior of processes 3 and 5. A summary is then given by a *cell transition* of the form $\mathrm{s} \xrightarrow{pqq} \mathrm{s}'$. Here, *cell* refers to *pqq*, which represents an isomorphism type of a pipeline of length three. Moreover, $\mathrm{s}, \mathrm{s}' \in S_p \times S_q \times S_q$ denote how states evolve in that particular fragment within a bigger pipeline, for example when executing all actions gathered in S5. Cells have bounded size so that the set of cell transitions can be effectively computed and represented.

Now, the behavior of process 4 can only be captured when we use at least two cell transitions (for S5 and S6). The reason is that receives of process 3 from 4 are interrupted by receives from process 2. Similarly, the receive con-

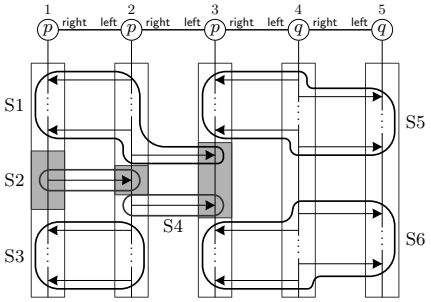

Fig. 6. Cell transitions wrt. s⊕r1

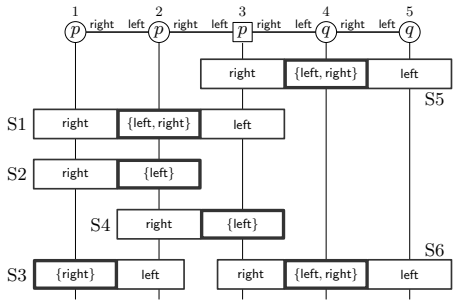

Fig. 7. Run of topology acceptor

text in the middle of process 3 will belong to two different summaries, as it is "interrupted" by a context switch on process 2. The splitting is not unique, as we could have merged S3 and S4. However, the total number of splits can be bounded: a send (receive) context is split whenever the complementary receives (sends, respectively) belong to distinct contexts. Thus, it is divided into at most $k \cdot |\mathcal{N}|$ summaries. As a result, any $(k, \mathsf{s}{\oplus}\mathsf{r1})$-bounded execution of a PCA is captured by a sequence of cell transitions such that each process is involved at most $k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$ times (cf. appendix). This gives us a bounded abstraction of a priori unbounded behaviors so that we can build a finite automaton that guesses such an abstraction and, simultaneously, checks if it corresponds to an accepting run of the PCA. A run of the finite automaton (topology acceptor in the general case) is depicted in Figure 7. We come back to it later.

Let us formalize these ideas. In the following, we do not restrict to pipelines, so $\mathcal{N}$ is an arbitrary alphabet of interface names. A topology $\mathcal{C} = (V, \nu, \pi)$ is called a *cell* if there is $\bar{u} \in V$ such that (1) for all $u \in V \setminus \{\bar{u}\}$, we have $\bar{u} \longmapsto u$, and (2) for all $u, u' \in V \setminus \{\bar{u}\}$, we do *not* have $u \longmapsto u'$. We call $\bar{u}$ a *center* of $\mathcal{C}$. Note that $\mathcal{C}$ is star-shaped, and we have $|V| \leq |\mathcal{N}| + 1$.

In the following, we define cell transitions to represent summaries. Let $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p \in \mathcal{P}})$ be a PCA over $\mathcal{N}$, with $\mathcal{A}_p = (S_p, \Longrightarrow_p, \iota_p, F_p)$. Moreover, let $\mathcal{C} = (V, \nu, \pi)$ be a cell (over $\mathcal{N}$ and $\mathcal{P}$). Suppose that $[\![\mathcal{A}]\!]^{\mathcal{C}} = (S, \Longrightarrow, \iota, F)$ and $\mathrm{s}, \mathrm{s}' \in S = \prod_{u \in V} S_u$. We let

- $\mathrm{s} \xrightarrow{\mathcal{C}}_{\mathsf{intf}} \mathrm{s}'$ if $|V| = 2$ and $L((S, \Longrightarrow, \mathrm{s}, \{\mathrm{s}'\})) \setminus \{\varepsilon\} \neq \emptyset$;
- $\mathrm{s} \xrightarrow{\mathcal{C}}_{\mathsf{s}\oplus\mathsf{r1}} \mathrm{s}'$ if there are a center $\bar{u} \in V$ of $\mathcal{C}$ and a word $W \in L((S, \Longrightarrow, \mathrm{s}, \{\mathrm{s}'\}))$ such that $W|_{\bar{u}} \in \Sigma_!^+$ and $W|_u \in \Sigma_?^+$ for all $u \in V \setminus \{\bar{u}\}$. Thus, if $\mathcal{C}$ contains at least three nodes, then there is a unique center, and this center "executes" only send actions.

Next, we apply cell transitions on arbitrary topologies $\mathcal{T} = (V, \nu, \pi)$. For $t \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$, we define a finite automaton $\langle \mathcal{A} \rangle_t^{\mathcal{T}} = (S, \Longrightarrow_t, \iota, F)$ where S, ι, and F are as in $[\![\mathcal{A}]\!]^{\mathcal{T}}$. Moreover, $\Longrightarrow_t$ is a subset of $S \times \Gamma \times S$, where Γ is the set of sub-structures of $\mathcal{T}$ that are cells. For states $\mathrm{s} = (s_u)_{u \in V}$ and $\mathrm{s}' = (s'_u)_{u \in V}$ from S and a cell $\mathcal{C} \in \Gamma$ with set of nodes $U \subseteq V$, we let $\mathrm{s} \overset{\mathcal{C}}{\Longrightarrow}_t \mathrm{s}'$ if we have both $(s_u)_{u \in U} \xrightarrow{\mathcal{C}}_t (s'_u)_{u \in U}$ and $s_u = s'_u$ for all $u \in V \setminus U$.

For $K \geq 1$, we define $L_K(\langle \mathcal{A} \rangle_t^{\mathcal{T}})$ to be the set of words $W = \mathcal{C}_1 \dots \mathcal{C}_n \in L(\langle \mathcal{A} \rangle_t^{\mathcal{T}})$ such that, for all $u \in V$, we have $|\{i \in [n] \mid u \text{ is a node of } \mathcal{C}_i\}| \leq K$.

Lemma 1. *Let $t \in \{\mathsf{s} \oplus \mathsf{r1}, \mathsf{intf}\}$, $k \geq 1$, and $K = k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$. Let $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p \in \mathcal{P}})$ be a PCA such that $L(\mathcal{A}_p) \subseteq \mathbb{W}_{(k,t)}$ for all $p \in \mathcal{P}$. Then, for all topologies $\mathcal{T}$ (over $\mathcal{N}$ and $\mathcal{P}$), we have $L_K(\langle \mathcal{A} \rangle_t^{\mathcal{T}}) \neq \emptyset$ iff $L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset$.*

Thus, the behavior of a context-bounded PCA can be faithfully represented by a "cellular" automaton that performs a bounded number of transitions on each process. The next step is to translate cellular automata into topology acceptors, which are similar to graph acceptors [18]. A run of a topology acceptor is a covering of a topology with cell-shaped tilings.

Definition 3. *A* topology acceptor (TA)*, over $\mathcal{N}$ and $\mathcal{P}$, is a pair $\mathcal{G} = (\mathcal{S}, \Delta)$ where $\mathcal{S}$ is a finite set of states and Δ is a finite set of transitions. A transition is a triple $(\mathcal{C}, \bar{u}, \rho)$ where $\mathcal{C} = (V, \nu, \pi)$ is an isomorphism type of a cell, $\bar{u} \in V$ is a center of $\mathcal{C}$, and $\rho : V \to \mathcal{S}$ is a mapping.*

Let $\mathcal{T} = (V, \nu, \pi)$ be a topology and $\gamma : V \to \mathcal{S}$ be a mapping. Given $\bar{u} \in V$, we denote by $cell(\mathcal{T}, \bar{u}, \gamma)$ the triple $(\mathcal{C}, \bar{u}, \rho)$ where $\mathcal{C}$ is the cell induced in $\mathcal{T}$ by the center $\bar{u}$ (the set of nodes of $\mathcal{C}$ being $V' = \{\bar{u}\} \cup \{u \in V \mid \bar{u} \longmapsto u\}$), and ρ is the restriction of γ to V'. We say that γ is a run of $\mathcal{G}$ on $\mathcal{T}$ if, for all $\bar{u} \in V$, we have that $cell(\mathcal{T}, \bar{u}, \gamma) \in \Delta$. By $L(\mathcal{G})$, we denote the set of topologies for which there is a run. Note that there is no particular acceptance condition in a TA.

Lemma 2. *Let $t \in \{\mathsf{s} \oplus \mathsf{r1}, \mathsf{intf}\}$ and $k \geq 1$. Let $\mathcal{A}$ be a PCA over $\mathcal{N}$ with process types $\mathcal{P}$. In exponential time, we can construct a TA $\mathcal{G}$ (of exponential size) such that, for all* acyclic *topologies $\mathcal{T}$ (over $\mathcal{N}$ and $\mathcal{P}$), we have $\mathcal{T} \in L(\mathcal{G})$ iff $L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset$.*

Proof. Let $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p \in \mathcal{P}})$ be the given PCA over $\mathcal{N}$, with $\mathcal{A}_p = (S_p, \Longrightarrow_p, \iota_p, F_p)$. We will assume that the sets S_p are pairwise disjoint. We also assume that $L(\mathcal{A}_p) \subseteq \mathbb{W}_{(k,t)}$ for all $p \in \mathcal{P}$. Recall that this can be achieved at the expense of a linear blow-up in the size of $\mathcal{A}_p$.

Note that there are $|\mathcal{P}| \cdot ((|\mathcal{N}| \cdot |\mathcal{P}|) + 1)^{|\mathcal{N}|}$ isomorphism types $\mathcal{C} = (V, \nu, \pi)$ of a cell. For each of them and all states $\mathrm{s}, \mathrm{s}' \in \prod_{u \in V} S_u$ of the finite automaton $[\![\mathcal{A}]\!]^{\mathcal{C}}$, we first determine whether $\mathrm{s} \xrightarrow{\mathcal{C}}_t \mathrm{s}'$, as defined above. Once the $\mathcal{C}, \mathrm{s}, \mathrm{s}'$ are fixed, this check reduces to a reachability analysis that can be done in polynomial time (since $|\mathcal{N}|$ is fixed).

Now, we construct the desired TA $\mathcal{G} = (\mathcal{S}, \Delta)$ as follows. Set $K = k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$. Recall that K is the bound that we imposed, in Lemma 1, on the number of local steps in the finite automaton $\langle \mathcal{A} \rangle_t^{\mathcal{T}}$. A state from $\mathcal{S}$ is an alternating sequence $\theta = (s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_K} s_K)$ where for some $p \in \mathcal{P}$ we have $s_0 = \iota_p$, $s_K \in F_p$ and all other states are also in S_p, and each τ_i is either an interface name $\tau_i \in \mathcal{N}$, or a set of interface names $\tau_i \subseteq \mathcal{N}$, or ε. The sequence θ describes a summary of the run performed by $\mathcal{A}$ on some node u of the topology (to which θ is assigned in a run of $\mathcal{G}$). We use $\tau_i \subseteq \mathcal{N}$ to describe a summary

step $s_{i-1} \xrightarrow{\tau_i} s_i$ of a cell $\mathcal{C}$ in which u is a center and communicates through the interface names in τ_i. We use $\tau_i \in \mathcal{N}$ to indicate that u is taking part in a summary step $s_{i-1} \xrightarrow{\tau_i} s_i$ of a cell $\mathcal{C}$ in which u is *not* the center and u communicates only through the interface name τ_i. Inserting ε-transitions allows us to consider sequences of equal length, which somehow simplifies the presentation. Hence, we assume that $s_{i-1} = s_i$ when $\tau_i = \varepsilon$.

Let $\theta, \theta' \in \mathcal{S}$ be sequences with labels $\tau_1, \ldots, \tau_K$ and $\tau'_1, \ldots, \tau'_K$, respectively. We let $f_\theta(a, i) = |\{\ell \leq i \mid a \in \tau_\ell \vee a = \tau_\ell\}|$ be the number of times interface $a \in \mathcal{N}$ was used up to transition i in θ. Also, we say that θ and θ' match on a link $\overset{a\ b}{\longmapsto}$ if we have (1) $f_\theta(a, K) = f_{\theta'}(b, K)$, (2) for all $i \in [K]$ with $a \in \tau_i$ there exists $j \in [K]$ with $b = \tau'_j$ and $f_\theta(a, i) = f_{\theta'}(b, j)$, and (3) the symmetric condition.

Transitions of $\mathcal{G}$ will check that the summary runs guessed locally at each node are compatible. This is illustrated in Figure 7, showing a run of $\mathcal{G}$ that corresponds to the PCA run in the adjoining figure. For simplicity, states are omitted. Let $\mathcal{C} = (V, \nu, \pi)$ be a cell, $\bar{u} \in V$ be a center of $\mathcal{C}$, and $\rho : V \to \mathcal{S}$. Then, $(\mathcal{C}, \bar{u}, \rho)$ is a transition of $\mathcal{G}$ if the following hold:

(a) For all $u \in V$, the states that occur in $\rho(u)$ are all from $S_{\pi(u)}$.

(b) For all $u \in V$ and $a, b \in \mathcal{N}$ such that $\bar{u} \overset{a\ b}{\longmapsto} u$ in $\mathcal{C}$, the sequences $\rho(\bar{u})$ and $\rho(u)$ match on $\overset{a\ b}{\longmapsto}$.

(c) For all $i \in [K]$ such that the i-th transition label of $\rho(\bar{u})$ is $\tau_i \subseteq \mathcal{N}$, we require that there is a sub-cell $\mathcal{D} = (V', \nu', \pi')$ of $\mathcal{C}$, with center $\bar{u}$ and $\mathrm{dom}(\nu'(\bar{u})) = \tau_i$, and there is a transition $\mathrm{s} \xrightarrow{\mathcal{D}}_t \mathrm{s}'$ where $\mathrm{s} = (s_u)_{u \in V'}$ and $\mathrm{s}' = (s'_u)_{u \in V'}$ are states from $\prod_{u \in V'} S_u$ such that the following hold:
   - the i-th transition of $\rho(\bar{u})$ is $s_{\bar{u}} \xrightarrow{\tau_i} s'_{\bar{u}}$, and
   - for all $a \in \tau_i$, there are $u \in V'$ and $b \in \mathcal{N}$ such that $\bar{u} \overset{a\ b}{\longmapsto} u$ in $\mathcal{D}$ and there is $j \in [K]$ such that the j-th transition of $\rho(u)$ is $s_u \xrightarrow{b} s'_u$ and $f_{\rho(\bar{u})}(a, i) = f_{\rho(u)}(b, j)$.

This concludes the construction of $\mathcal{G}$. The correctness proof, which crucially relies on Lemma 1, can be found in the appendix. □

The above construction is only correct for acyclic topologies. Consider the PCA from Example 1. Figure 8 illustrates that the corresponding TA $\mathcal{G}$ accepts the ring $\mathcal{T}$ induced by $qqqq$, though we have $L(\mathcal{A}, \mathcal{T}) = \emptyset$. The reason is that $\mathcal{G}$ checks *locally* whether a run encodes a composition of cell transitions, without retrieving cyclic dependencies that violate the run conditions of PCAs.

5 Context-Bounded Parameterized Verification

When we are only interested in pipelines, rings, or trees, a TA can be translated to a finite automaton or, respectively, tree automaton, which is then checked for emptiness. This allows us to apply the construction and ideas presented in the previous section to context-bounded verification over trees, pipelines, and rings. The case of rings, however, needs some additional subtle argument.

Theorem 2. *The problem* TREE$_r$-NONEMPTINESS(t) *is* EXPTIME*-complete, for all* $r \geq 2$ *and* $t \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$.

Proof. We first show the upper bound. By Lemma 2, we can transform an r-tree PCA with set of process types $\mathcal{P}$ into a TA $\mathcal{G}$ over $\{\mathsf{father}, \mathsf{child}_1, \ldots, \mathsf{child}_r\}$ and $\mathcal{P}$ of exponential size. It is standard to transform the TA $\mathcal{G}$ into a tree automaton $\mathcal{B}$ (with polynomial blow up) such that, for all r-tree topologies $\mathcal{T}$ over $\mathcal{P}$, we have $\mathcal{T} \in L(\mathcal{G})$ iff $\mathcal{T} \in L_{tree}(\mathcal{B})$. As emptiness of tree automata is decidable in polynomial time, this gives us an EXPTIME decision procedure.

The lower bound is by reduction from the intersection problem for binary-tree automata, which is EXPTIME-complete [17]. 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 $k \geq 1$ and binary-tree automata $\mathcal{B}_1, \ldots, \mathcal{B}_k$, we construct, in polynomial time, a PCA $\mathcal{A}$ such that, for all 2-tree topologies $\mathcal{T}$, we have $L_{(2k,\mathsf{s}\oplus\mathsf{r1})}(\mathcal{A}, \mathcal{T}) \neq \emptyset$ iff $L_{(3k,\mathsf{intf})}(\mathcal{A}, \mathcal{T}) \neq \emptyset$ iff $\mathcal{T} \in L_{tree}(\mathcal{B}_1) \cap \ldots \cap L_{tree}(\mathcal{B}_k)$. The idea is that each internal node u in the tree topology chooses transitions $\delta_1, \ldots, \delta_k$ of $\mathcal{B}_1, \ldots, \mathcal{B}_k$, respectively, that are applied at u. These transitions are sent to the children $u.1$ and $u.2$ of u. When $u.1$ (or $u.2$) receives a transition δ_i, it immediately sends a corresponding transition δ_i' to its own children. This is why the PCA works with $2k$ and $3k$ contexts. □

Theorem 3. *The problem* PIPELINE-NONEMPTINESS(t) *is* PSPACE*-complete, for all* $t \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$.

Proof. For the upper bound, we again use Lemmas 1 and 2, which allow us to transform a PCA into a TA $\mathcal{G}$. The latter, in turn, can be transformed into a finite automaton (we perform a more sophisticated construction below for rings). However, the construction has to be done "on-the-fly" so that it takes only polynomial space. The pre-computation of the cell transitions $\mathrm{s} \xrightarrow{\mathcal{C}}_t \mathrm{s}'$ can be done in polynomial time. Moreover, a single state of $\mathcal{G}$ is representable using polynomial space. Clearly, checking if a sequence of at most three consecutive states of $\mathcal{G}$ form a valid transition can be done in polynomial space. Putting all that together, we obtain a PSPACE procedure.

The lower-bound proof is an adaptation of the lower-bound proof for Theorem 2, but uses intersection of finite automata, which is PSPACE-complete. □

Theorem 4. *The problem* RING-NONEMPTINESS(t) *is* PSPACE*-complete, for all* $t \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$.

The rest of this section is devoted to the proof of Theorem 4. Recall that we cannot simply apply Lemma 2, since the constructed TA $\mathcal{G}$ is only guaranteed to be correct when running on acyclic topologies. Consider the $(2, \mathsf{s}{\oplus}\mathsf{r1})$- and $(2, \mathsf{intf})$-bounded PCA $\mathcal{A}$ from Example 1. For simplicity, we suppose $K = 2$ (cf. Lemma 1). Let $\mathcal{T}$ be the ring topology induced by $qqqq$. Though $L(\mathcal{A}, \mathcal{T}) = \emptyset$, there is a run of the associated TA $\mathcal{G}$ on $\mathcal{T}$, which is illustrated in Figure 8. Essentially, the run checks, locally, if two successive local executions fit together. There is, by now, no mechanism that keeps track of the causal dependency

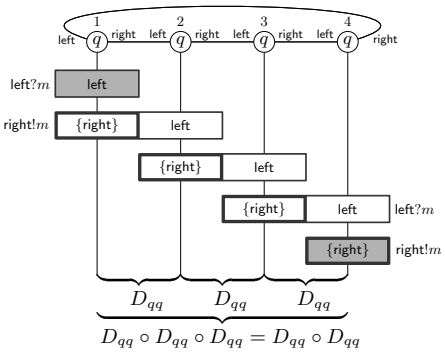

Fig. 8. Topology acceptor on a ring

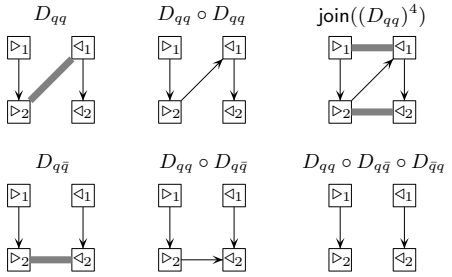

Fig. 9. Dependence graphs (reduced, without transitive closure)

between events created during a run. In particular, it does not record that the gray-shaded left-event (which arised from a receive action) is scheduled *before* the gray-shaded right-event (which arised from a send action). Thus, those events cannot be matched, i.e., the run of $\mathcal{G}$ does not reflect a run of $\mathcal{A}$. We will, therefore, enrich $\mathcal{G}$ to obtain a decision procedure for rings.

To be able to infer dependencies from cell transitions, we slightly modify the definition of $\mathrm{s} \xrightarrow{\mathcal{C}}_{\mathsf{s}\oplus\mathsf{r1}} \mathrm{s}'$: we require that there are a center $\bar{u} \in V$ of $\mathcal{C}$ and a word $W \in L((S, \Longrightarrow, \mathrm{s}, \{\mathrm{s}'\}))$ such that (1) $W|_{\bar{u}} \in \Sigma_!^+$ and $|V| = 2$, or (2) $W|_{\bar{u}} \in \Sigma_!^*(\Sigma_{\mathsf{left}!}^+\Sigma_{\mathsf{right}!}^+\Sigma_{\mathsf{left}!}^+)\Sigma_!^* \cup \Sigma_!^*(\Sigma_{\mathsf{right}!}^+\Sigma_{\mathsf{left}!}^+\Sigma_{\mathsf{right}!}^+)\Sigma_!^*$. Condition (2) guarantees that there are at least two direction switches in the sending phase of $\bar{u}$. Note that Lemma 1 is still valid at the cost of a linear blow up of K.

The idea is now to add *dependence graphs* that keep track of the causal dependencies between cell transitions. They arise naturally when we combine the behavior of two processes in terms of states of $\mathcal{G}$. For processes 1 and 2 in Figure 8, we obtain the dependence graph D_{qq} depicted in Figure 9. There are two kinds of constraints, an undirected one representing synchronizations (the thick gray lines), and a directed one for strict causality (depicted by arrows). The effect of appending a further process of type q can be computed as a composition $D_{qq} \circ D_{qq}$, which we obtain by merging the right side of the first graph with the left side of the second, taking the transitive closure of the constraints, and removing the middle nodes (in the figure, we represent the transitive closure by a minimal set of constraints). Now, "closing" the pipeline $qqqq$ towards a ring corresponds to joining the left and right hand side of $(D_{qq})^4 = (D_{qq})^2$, depicted as $\mathsf{join}((D_{qq})^4)$ in Figure 9. But this creates a cycle using at least one constraint of type $\rightarrow$, which has to be interpreted as a violation of the run condition of PCAs. However, if the third process is of type $\bar{q}$ (executing $(\mathsf{right}!m)(\mathsf{left}?m)$), D_{qq} is composed with $D_{q\bar{q}}$, which results in $D_{qq} \circ D_{q\bar{q}}$. Closing the pipeline corresponds to appending $D_{\bar{q}q}$, which resolves the "bad" dependency and results in the graph at the bottom right, whose join is harmless. Let us implement these ideas.

Lemma 3. *Suppose $t \in \{\mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$ and $k \geq 1$. Let $\mathcal{A}$ be a ring PCA with process types $\mathcal{P}$. We can construct a finite automaton $\mathcal{B}$ over $\mathcal{P}$ of exponential size such that, for all rings $\mathcal{T}$ over $\mathcal{P}$, we have $\mathcal{T} \in L_{ring}(\mathcal{B})$ iff $L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset$.*

Proof. We start from the TA $\mathcal{G} = (\mathcal{S}, \Delta)$ over $\{\mathsf{left}, \mathsf{right}\}$ and $\mathcal{P}$ that we obtain according to the proof of Lemma 2. Recall that a state of $\mathcal{G}$ is a sequence $s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_K} s_K$ where $K = 18k$ (due to the new definition of $\xrightarrow{\mathcal{C}}_{\mathsf{s}\oplus\mathsf{r1}}$).

First, we introduce dependence graphs. Let $\Omega = \{\triangleright_1, \dots, \triangleright_K\} \cup \{\triangleleft_1, \dots, \triangleleft_K\}$. A *dependence graph* is a graph $D = (\Omega, \to, \sim)$ where $\to, \sim \subseteq \Omega \times \Omega$ with $\sim$ symmetric. The *composition* $D_1 \circ D_2$ of two graphs $D_1 = (\Omega, \to_1, \sim_1)$ and $D_2 = (\Omega, \to_2, \sim_2)$ is obtained as follows. In D_1, we rename every $\triangleleft_i$ into $\flat_i$. Similarly, in D_2, rename every $\triangleright_i$ into $\flat_i$. The composition is then defined as $(\Omega, \to, \sim)$ where $\to$ is the restriction of $E^* \circ (\to_1 \cup \to_2) \circ E^*$ to Ω, where $E = \to_1 \cup \sim_1 \cup \to_2 \cup \sim_2$, and $\sim$ is the restriction of $(\sim_1 \cup \sim_2)^+$ to Ω. We also define a join operation on dependence graphs $D = (\Omega, \to, \sim)$ and let $\mathsf{join}(D)$ be the dependence graph $(\Omega, \to, \sim \cup \{(\triangleright_i, \triangleleft_i), (\triangleleft_i, \triangleright_i) \mid i \in [K]\})$ (cf. Figure 9).

With a pair $(\theta, \theta') \in \mathcal{S} \times \mathcal{S}$, we now associate a dependence graph $D_{\theta\theta'}$. Suppose the labels of θ are $\tau_1, \dots, \tau_K$ and the labels of the sequence θ' are $\tau'_1, \dots, \tau'_K$. Then, we define $D_{\theta\theta'} = (\Omega, \to, \sim)$ by $\to = \{(\triangleright_i, \triangleright_j), (\triangleleft_i, \triangleleft_j) \mid 1 \leq i < j \leq K\}$ and $\sim = \{(\triangleright_i, \triangleleft_j), (\triangleleft_j, \triangleright_i) \mid (\mathsf{right} \in \tau_i \text{ or } \mathsf{right} = \tau_i)$ and $(\mathsf{left} \in \tau'_j \text{ or } \mathsf{left} = \tau'_j)$ and $f_\theta(\mathsf{right}, i) = f_{\theta'}(\mathsf{left}, j)\}$ (cf. Figures 8 and 9).

Let us determine the finite automaton $\mathcal{B}$ over $\mathcal{P}$ such that $\mathcal{T} \in L_{ring}(\mathcal{B})$ iff $L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset$, for all rings $\mathcal{T}$ over $\mathcal{P}$. To allow for a modular construction, we will actually build a finite automaton reading words over $\mathcal{P} \times \mathcal{S}$. We define $\mathcal{B}$ as the product of two finite automata. First, $\mathcal{B}_{\text{run}}$ accepts those words $w = (p_1, \theta_1) \dots (p_n, \theta_n) \in (\mathcal{P} \times \mathcal{S})^*$ of length $n \geq 3$ such that the mapping $\{1 \mapsto \theta_1, \dots, n \mapsto \theta_n\}$ is a run of $\mathcal{G}$ on the ring with set of nodes $V = \{1, \dots, n\}$ that is induced by $p_1 \dots p_n$. As is the case for pipelines and trees, this construction is easy and omitted. The second finite automaton, $\mathcal{B}_{\text{dep}}$, checks the dependencies that we create during a run. A state of $\mathcal{B}_{\text{dep}}$ is either ι, or a triple (θ_1, θ_2, D) where $\theta_1, \theta_2 \in \mathcal{S}$ denote the very first and, respectively, the last state of $\mathcal{G}$ that were read so far (thus, θ_1 is invariant along a run), and D is a dependence graph. Let $w = (p_1, \theta_1)(p_2, \theta_2) \in (\mathcal{P} \times \mathcal{S})^2$. We have an initial transition $\iota \overset{w}{\Longrightarrow} (\theta_1, \theta_2, D_{\theta_1\theta_2})$. For simplicity, we hereby allow $\mathcal{B}_{\text{dep}}$ to read two letters at once. Now, let $\theta_1, \theta_2 \in \mathcal{S}$, $(p, \theta_3) \in \mathcal{P} \times \mathcal{S}$, and D be a dependence graph. Then, we have a transition $(\theta_1, \theta_2, D) \overset{(p,\theta_3)}{\Longrightarrow} (\theta_1, \theta_3, D \circ D_{\theta_2\theta_3})$. Thus, the "last" state θ_2 is updated to θ_3, and the dependence graph is updated to the new dependency created in terms of θ_3. Finally, (θ_1, θ_2, D) is a final state if $\mathsf{join}(D \circ D_{\theta_2\theta_1})$ does not have a cycle using edges in $\to \cup \sim$, and using $\to$ at least once. Intuitively, this means that the pipeline can be closed towards a ring.

We now obtain $\mathcal{B}$ as the projection (to $\mathcal{P}$) of the product $\mathcal{B}_{\text{run}} \times \mathcal{B}_{\text{dep}}$. One can show that the dependence graphs faithfully represent the causal dependencies created between the states of the TA assigned to the first and the last process of a pipeline. Thus, the join allows us to verify if the pipeline can be closed. □

Proof (of Theorem 4). Like for pipelines, we perform the constructions given in the proofs of Lemmas 2 and 3 on the fly. Since the graphs $D_{\theta\theta'}$, their composition, and their join can be computed in polynomial time, we still get a PSPACE procedure. For the lower bound, we again exploit PSPACE-hardness of intersection of finite automata. The adaptation to rings is straightforward. □

6 Conclusion

We showed that verification of PCAs running on pipelines, rings, and trees is decidable under certain context bounds. Our approach also yields decidability of all *universal* verification problems: Do *all* topologies accepted by a finite (tree) automaton allow for an accepting run of the given PCA? This involves complementation of finite (tree) automata. For future work, it will be worthwhile to consider model checking against temporal logics, and automata models that may run over topologies of unbounded degree such as stars and unranked trees.

References

1. P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *VMCAI'13*, volume 7737 of *LNCS*, pages 476–495. Springer, 2013.
2. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proc. of LICS'93*, pages 160–170, 1993.
3. B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281, 2014.
4. M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Log. Methods Comput. Sci.*, 7(4), 2011.
5. B. Bollig. Logic for communicating automata with parameterized topology. In *CSL-LICS'14*. ACM, 2014. to appear.
6. A. Bouajjani and M. Emmi. Bounded phase analysis of message-passing programs. In *TACAS'12*, volume 7214 of *LNCS*, pages 451–465. Springer, 2012.
7. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.
8. 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.
9. E. A. Emerson and V. Kahlon. Parameterized model checking of ring-based message passing systems. In *CSL'04*, volume 3210 of *LNCS*, pages 325–339, 2004.
10. E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.
11. J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS'14*, volume 25 of *LIPIcs*, pages 1–10, 2014.
12. J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, LNCS, pages 124–140, 2013.
13. S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS'08*, volume 4963 of *LNCS*, pages 299–314, 2008.
14. S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV'10*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.
15. P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL'11*, pages 283–294. ACM, 2011.
16. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
17. H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.
18. W. Thomas. Elements of an automata theory over partial orders. In *POMIV'96*, volume 29 of *DIMACS*. AMS, 1996.

A Proof of Lemma 1

Lemma 1. *Let $t \in \{\mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$, $k \geq 1$, and $K = k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$. Let $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p\in\mathcal{P}})$ be a PCA such that $L(\mathcal{A}_p) \subseteq \mathbb{W}_{(k,t)}$ for all $p \in \mathcal{P}$. Then, for all topologies $\mathcal{T}$ (over $\mathcal{N}$ and $\mathcal{P}$), we have $L_K(\langle\mathcal{A}\rangle_t^{\mathcal{T}}) \neq \emptyset$ iff $L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset$.*

Proof. We only consider the more difficult case of context type $\mathsf{s}\oplus\mathsf{r1}$. The case intf follows the same lines (and we get a slightly better bound $K = k+(k\cdot|\mathcal{N}|)$). So let, in the following, $t = \mathsf{s}\oplus\mathsf{r1}$.

Let $k \geq 1$ and set $K = k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$. Let $\mathcal{A} = (\mathcal{P}, Msg, (\mathcal{A}_p)_{p\in\mathcal{P}})$ be a PCA such that $L(\mathcal{A}_p) \subseteq \mathbb{W}_{(k,t)}$ for all $p \in \mathcal{P}$. Moreover, fix a topology $\mathcal{T} = (V, \nu, \pi)$ over $\mathcal{N}$ and $\mathcal{P}$. The direction from cell transitions to PCA follows easily from the definitions. We will show

$$L_{(k,t)}(\mathcal{A}, \mathcal{T}) \neq \emptyset \implies L_K(\langle\mathcal{A}\rangle_t^{\mathcal{T}}) \neq \emptyset\,.$$

The rough idea is to consider an automata model that is able to execute both, PCA transitions and cell transitions, and to transform, step by step, all PCA transitions into cell transitions. To this aim, we define another finite automaton $(\!|\mathcal{A}|\!)_t^{\mathcal{T}} := (S, \Longrightarrow \cup \Longrightarrow_t, \iota, F)$ over $\Sigma^{\mathcal{T}} \cup \Gamma$ where S, $\Longrightarrow$, ι, and F are as in $[\![\mathcal{A}]\!]^{\mathcal{T}}$ and $\Longrightarrow_t \subseteq S \times \Gamma \times S$ is as in $\langle\mathcal{A}\rangle_t^{\mathcal{T}}$, i.e., Γ is the set of substructures of $\mathcal{T}$ that are cells.

For a word $W = \gamma_1 \dots \gamma_n \in (\Sigma^{\mathcal{T}} \cup \Gamma)^*$ and a node $u \in V$, we define the projection of W to u as $(\gamma_1)|_u \cdot \ldots \cdot (\gamma_n)|_u \in (\Sigma \cup \{\top\})^*$. Here, for $\mathcal{C} \in \Gamma$, we let $\mathcal{C}|_u = \top$ if u is a node of $\mathcal{C}$. Otherwise, $\mathcal{C}|_u = \varepsilon$. The definition of contexts for words $w \in (\Sigma \cup \{\top\})^*$ is the same as for words over Σ with one difference: one ocurrence of $\top$ is counted as one context. This naturally leads to an extended definition of (ℓ, t)-boundedness for words from $(\Sigma^{\mathcal{T}} \cup \Gamma)^*$, for $\ell \geq 1$. We let $L_{(\ell,t)}((\!|\mathcal{A}|\!)_t^{\mathcal{T}})$ denote the set of (ℓ, t)-bounded words that are accepted by $(\!|\mathcal{A}|\!)_t^{\mathcal{T}}$.

An important property of $L_{(\ell,t)}(\mathcal{A}, \mathcal{T})$, $L_{(\ell,t)}(\langle\mathcal{A}\rangle_t^{\mathcal{T}})$, and $L_{(\ell,t)}((\!|\mathcal{A}|\!)_t^{\mathcal{T}})$ is that they are closed under permutation of *independent* actions, which are executed by distinct sets of processes. We say that two actions $\gamma, \gamma' \in \Sigma^{\mathcal{T}} \cup \Gamma$ are independent if $loc(\gamma) \cap loc(\gamma') = \emptyset$, where $loc(\langle u, m, v\rangle) = \{u, v\}$ and $loc(\mathcal{C})$ is the set of nodes of $\mathcal{C}$.

Now, suppose $W \in L_{(k,t)}(\mathcal{A}, \mathcal{T})$, which implies $W \in L_{(k,t)}((\!|\mathcal{A}|\!)_t^{\mathcal{T}})$. Pick some $\bar{u} \in V$ and a maximal, nonempty set I of positions in W that belong to (1) the first send context of $\bar{u}$, and (2) a nonempty uninterrupted receive context for each involved adjacent node of $\bar{u}$.

Suppose that the receiving processes are $U \subseteq V$. Moreover, let $\mathcal{C}$ be the cell induced by $U \cup \{\bar{u}\}$ in $\mathcal{T}$. By reordering of independent actions, W is equivalent to some word $W' = \gamma_1 \dots \gamma_n \in L_{(k,t)}((\!|\mathcal{A}|\!)_t^{\mathcal{T}})$ such that, for some $i \leq j$, the infix $\gamma_i \dots \gamma_j$ corresponds to the word induced by the positions I selected in W. Let $\mathrm{s}_0\mathrm{s}_1 \dots \mathrm{s}_n \in S^*$ be an accepting run of $(\!|\mathcal{A}|\!)_t^{\mathcal{T}}$ on W'. It follows from the definitions that $\mathrm{s}_{i-1} \overset{\mathcal{C}}{\Longrightarrow}_t \mathrm{s}_j$. Thus, $\mathrm{s}_0 \dots \mathrm{s}_{i-1}\mathrm{s}_j \dots \mathrm{s}_n$ is an accepting run of $(\!|\mathcal{A}|\!)_t^{\mathcal{T}}$ on $W'' = \gamma_1 \dots \gamma_{i-1}\,\mathcal{C}\,\gamma_{j+1} \dots \gamma_n$. Note that W'' is $(k+1, t)$-bounded, since we possibly split the send context and some receive contexts into two. Now, we choose the next send context on $\bar{u}$ and so on. When there are no more send

contexts on $\bar{u}$, then we found a word in $L(\langle\!|\mathcal{A}|\!\rangle_t^{\mathcal{T}})$ whose projection to $\bar{u}$ consists of at most $k + k \cdot |\mathcal{N}|$ contexts. Moreover, in the projection of each neighbor of $\bar{u}$, we created at most as many (i.e., $k + k \cdot |\mathcal{N}|$) new contexts. We now pick another node $\bar{u}$ and so on, until all actions from $\Sigma^{\mathcal{T}}$ were transformed into cell transitions and we end up in a word $W \in L(\langle\!|\mathcal{A}|\!\rangle_t^{\mathcal{T}}) \cap \Gamma^*$. Putting everything together, we obtain that W is (K, t)-bounded for

$$K = k + (k \cdot |\mathcal{N}|) + (|\mathcal{N}| \cdot (k + k \cdot |\mathcal{N}|)) = k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)\,.$$

The first summand in the term following K is for "old" contexts, the second summand is for new send contexts, and the third one for new receive contexts (this is why we have a factor $|\mathcal{N}|$). Finally, since W consists of letters from Γ only, we have $W \in L_K(\langle\mathcal{A}\rangle_t^{\mathcal{T}})$. □

B Correctness Proof for Lemma 2

Let $K = k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$. By Lemma 1, it suffices to show that, for all acyclic topologies $\mathcal{T}$ over $\mathcal{N}$ and $\mathcal{P}$, we have $\mathcal{T} \in L(\mathcal{G})$ iff $L_K(\langle\mathcal{A}\rangle_t^{\mathcal{T}}) \neq \emptyset$. Assume $\langle\mathcal{A}\rangle_t^{\mathcal{T}} = (S, \Longrightarrow_t, \iota, F)$, and let $\mathcal{T} = (V, \nu, \pi)$ be an acyclic topology.

We first show the direction "$\Longleftarrow$". Let $W = \mathcal{C}_1 \dots \mathcal{C}_n \in L_K(\langle\mathcal{A}\rangle_t^{\mathcal{T}})$ and suppose $\xi = \mathrm{s}_0\mathrm{s}_1 \dots \mathrm{s}_n$ is an accepting run of $\langle\mathcal{A}\rangle_t^{\mathcal{T}}$ on W. For every $i \in [n]$, we fix a center $\bar{u}_i$ of $\mathcal{C}_i$. Let us determine a run $\gamma : V \to \mathcal{S}$ of $\mathcal{G}$ on $\mathcal{T}$. For $u \in V$, we define $\gamma(u) = (s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_K} s_K)$ as follows. First, $s_0 = \iota_u$. Suppose that s_i is the j-th state in $\mathrm{s}_1 \dots \mathrm{s}_n$ such that u is a node of $\mathcal{C}_i = (V', \nu', \pi')$. Then, we let $s_j = \mathrm{s}_i|_u$. Moreover, $\tau_j = a$ if $\nu'(u, a) = \bar{u}_i$. Finally, $\tau_j = \mathrm{dom}(\nu'(u))$ if $u = \bar{u}_i$. The sequences are filled up to length K by ε-labels and repeating the last local state, respectively.

We show that γ is a run of $\mathcal{G}$ on $\mathcal{T}$ by verifying that, for every $u \in V$, the tuple $(\mathcal{C}, u, \rho) := \mathit{cell}(\mathcal{T}, u, \gamma)$ is a transition of $\mathcal{G}$. Note that $(\mathcal{C}, u, \rho)$ has to fulfill the properties (a)–(c) in the definition of Δ. Properties (a) and (b) follow directly from the definition of γ. For (c), the argument goes as follows: for all $i \in [K]$ such that the i-th transition label of $\rho(u)$ is $\tau_i \subseteq \mathcal{N}$, there is $j \in [n]$ such that τ_i was induced by $\mathcal{C}_j$. Thus, there is a sub-cell $\mathcal{D}$ of $\mathcal{C}_j$ as required in property (c).

Now, we show "$\Longrightarrow$". Let $\gamma : V \to \mathcal{S}$ be a run of $\mathcal{G}$ on $\mathcal{T}$. This implies that, for every $u \in V$ and every $i \in [K]$ such that the i-th transition label of $\gamma(u)$ is $\tau_{(u,i)} \subseteq \mathcal{N}$, there is a sub-cell $\mathcal{D}_{(u,i)}$ fulfilling (c) in the definition of $\mathcal{G}$. Without loss of generality, we assume that all ε-labels in $\gamma(u)$ are scheduled at the end.

Let $\mathbb{D}$ be the set of these sub-cells $\mathcal{D}_{(u,i)}$.[3] Moreover, let $g : \mathbb{D} \times V \rightharpoonup [K]$ be a partial mapping that associates with every node v of $\mathcal{D}_{(u,i)}$ the corresponding transition j in $\gamma(v)$. This mapping is well-defined, as for every $n \in [K]$, $\theta \in \mathcal{S}$, and $a \in \mathcal{N}$, there is at most one step j in θ such that $f_\theta(a, j) = n$ and $\tau_j = a$.

[3] We rather deal with a multiset. Several sub-cells may be identical, but we will still distinguish them. In other words, when we write $\mathcal{D}_{(u,i)}$, we actually mean (u, i), but we identify the pair with a cell, since it is convenient.

Let $\sqsubset \subseteq \mathbb{D} \times \mathbb{D}$ be defined as the least strict preorder satisfying $\mathcal{D}_{(u,i)} \sqsubset \mathcal{D}_{(v,j)}$ whenever $\mathcal{D}_{(u,i)}$ and $\mathcal{D}_{(v,j)}$ share a node v' such that $g(\mathcal{D}_{(u,i)}, v') < g(\mathcal{D}_{(v,j)}, v')$. The restriction to acyclic topologies guarantees that $(\mathbb{D}, \sqsubset)$ is actually a (strict) partial-order relation. Let $W = \mathcal{C}_1 \dots \mathcal{C}_n$ be a linearization of $(\mathbb{D}, \sqsubset)$. By induction, we will show that $W \in L(\langle \mathcal{A} \rangle_t^{\mathcal{T}})$, which directly implies that $W \in L_K(\langle \mathcal{A} \rangle_t^{\mathcal{T}})$. As there cannot be a $\mathcal{D} \in \mathbb{D}$ such that $\mathcal{D} \sqsubset \mathcal{C}_1$, we have $g(\mathcal{C}_1, v) = 1$ for all v in $\mathcal{C}_1$. Let $s_{(u,i)}$ denote the state in $\gamma(u)$ reached after the i-th step. Then, there is a transition $\iota \xRightarrow{\mathcal{C}_1}_t (s_u)_{u \in V}$ in $\langle \mathcal{A} \rangle_t^{\mathcal{T}}$ where

$$s_u = \begin{cases} s_{(u,1)} & \text{if } u \text{ is a node of } \mathcal{C}_1 \\ \iota_u & \text{otherwise} \end{cases}$$

Now, assume that we constructed, for a strict prefix $\mathcal{C}_1 \dots \mathcal{C}_\ell$ of W, a sequence of transitions in $\langle \mathcal{A} \rangle_t^{\mathcal{T}}$ of the form $\iota \xRightarrow{\mathcal{C}_1}_t \dots \xRightarrow{\mathcal{C}_\ell}_t (s_u)_{u \in V}$. By definition of $(\mathbb{D}, \sqsubset)$ and the assumption that W is a linearization, we have $s_u = s_{(u,j)}$ for all nodes u of $\mathcal{C}_{\ell+1}$ and $j = g(\mathcal{C}_{\ell+1}, u) - 1$. This implies that there is a transition $(s_u)_{u \in V} \xRightarrow{\mathcal{C}_{\ell+1}}_t (s'_u)_{u \in V}$ with

$$s'_u = \begin{cases} s_{(u,j')} & \text{if } u \text{ in } \mathcal{C}_{\ell+1} \text{ and } j' = g(\mathcal{C}_{\ell+1}, u) \\ s_u & \text{otherwise.} \end{cases}$$

Obviously, after treating W completely, we end up in a final state of $\langle \mathcal{A} \rangle_t^{\mathcal{T}}$, which proves $W \in L(\langle \mathcal{A} \rangle_t^{\mathcal{T}})$. We deduce $L_K(\langle \mathcal{A} \rangle_t^{\mathcal{T}}) \neq \emptyset$.