On Parameterized Verification over Tree Topologies

Romain Delpy
LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Anca Muscholl
LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Grégoire Sutre
LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Abstract

Parameterized verification of finite-state processes with rendez-vous synchronization is notoriously undecidable when processes are linearly ordered. In this paper we study two kinds of bounds under which we determine the complexity of safety checking over tree topologies. When bounding the depth we obtain that the complexity is related to the fast growing hierarchy. Our second bound limits the alternations between upwards and downwards synchronizations in the tree (phases), and occurs naturally in many concrete settings. If we fix the number of phases then the complexity of safety checking is EXPSPACE complete, and if the number of phases is part of the input it is 2EXPSPACE complete (both for arbitrary depth).

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Concurrent programming, Parameterized verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2026.31

Related Version *Full Version*: `https://arxiv.org/abs/2606.27172` [17]

Funding This work was (partially) supported by the grant ANR-23-CE48-0005 of the French National Research Agency ANR (project PaVeDyS).

1 Introduction

Analyzing the correctness of concurrent or distributed systems is an inherently challenging task, due to inter-process interactions that can result in very large state spaces. Beyond this state explosion problem, a different challenge arises in settings where one is interested in proving correctness independently of the system size. This is the realm of parameterized verification, that has gained intensive momentum over the past decade (cf. surveys [9, 19]).

In this paper we investigate the complexity of safety checking of parameterized systems over tree topologies. Our focus on trees is motivated by two key ideas. First, trees naturally model modern execution environments: in recursive process creation, parent and child processes synchronize through shared communication channels. This structure also arises in cloud infrastructures (master-worker hierarchies), file systems, and hierarchical cache coherence protocols. Second, while parameterized communication over general graphs is notoriously difficult to define – often requiring complex machinery like MSO and bounded tree-width [3], or specialized logics [11, 12] – tree topologies offer a balance between expressive power and algorithmic tractability.

Consider a concurrent web scraper, modeled according to a recursive `Rust` implementation (`github.com/dawksh/mscraper/`) which scans for URLs embedded in web pages. In this system, when a thread discovers a new URL, it spawns a child thread to scan the corresponding page; this process continues recursively until a predefined depth is reached. Communication follows the tree structure: each process aggregates URL lists from its children and propagates

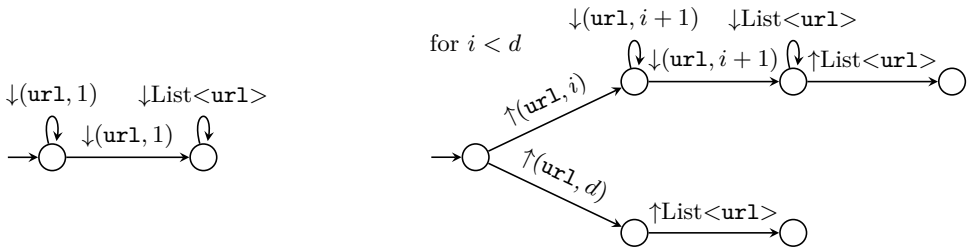

Figure 1 Automata representing the processes of the web scrapper, where d is the maximal depth. The left one represents the main process, and the right one represents the spawned threads.

the results back to its parent. Formally, we model each process as a finite-state automaton where transitions represent either synchronizations with a parent (written as ↑), or with a child (written as ↓, see Figure 1). Data exchange involves passing URLs coupled with depth counters; once a process receives a depth value equal to the maximum threshold, it terminates its recursive spawning and stops synchronization with further child processes.

In models with peer-to-peer communication, the undecidability of safety checking is immediate under asynchronous communication [13], regardless of parameterization. We therefore focus on rendez-vous synchronization. Even in this synchronous setting, however, safety checking on trees, or even arrays, remains undecidable [4, 20]. This motivates the search for restrictions under which safety checking is decidable.

We investigate the complexity of two under-approximations of systems with rendez-vous synchronization over trees. The first one consists in bounding the tree depth. This is arguably the most natural restriction that we can impose on our systems. However, we show that in this case the complexity of safety checking is prohibitively high, characterized by the fast growing hierarchy. Specifically, for a fixed depth d, complexity lies between the levels $\mathbf{F}_{\Omega_{d-1}}$ and $\mathbf{F}_{\Omega_d}$ of the hierarchy. This result motivates our second restriction, called *phase bound*. Within a phase a process may synchronize either exclusively with its parent or exclusively with its children. Our web scraper example above has phase bound 3. It is also interesting to note that many of the standard examples of parametrized protocols over trees are phase bounded, e.g. the Leader election protocol [1] (electing a leader among leaves) has also phase bound 3. We show that by fixing the number of phases, safety checking becomes Expspace-complete, and if the number of phases is part of the input then it becomes 2Expspace-complete. Somewhat surprisingly, these complexity results hold independently of the tree depth.

Related work. The verification of parameterized tree-structured systems has been extensively studied. Tree Regular Model Checking [2, 22] provides a symbolic framework relying on tree transducers. It often results in semi-algorithmic procedures and is typically restricted to ranked architectures. To cope with non-termination and implementation complexity, alternative symbolic approaches have been proposed, such as backward reachability [1] and abstraction-refinement techniques [10].

A parallel line of research focuses on cutoff-based verification, which reduces the parameterized problem to a finite check on a representative system size. While cutoffs have been established for various logics, including indexed LTL and CTL [3], and their existence proved to be polynomially decidable for certain rendez-vous protocols [6], this approach remains sensitive to the state-space explosion problem.

Our work also relates to recent efforts to identify decidable fragments through structural and behavioral constraints. In the context of pi-calculus, bounding the tree depth of process creation leads to a complexity of $\mathbf{F}_{\varepsilon_0}$ [5]. A different phase-based abstraction has been

explored for broadcast communication [21]. In that setting, a phase is defined by the direction of communication (exclusively broadcasting or receiving); however, safety remains undecidable even for a small number of phases.

For convenience, technical terms and notations in the electronic version of this manuscript are hyper-linked to their definitions (cf. `https://ctan.org/pkg/knowledge`). Proofs that are missing in the main text can be found in the full version [17].

2 Definitions

In this paper, we consider communicating processes over tree topologies. A rooted tree $\mathcal{T} = (P, Ch)$ is a directed graph without self-loops, where P is a set of processes and $Ch \subseteq P \times P$ is a set of channels, with one distinct node (the root) such that every node has a unique path from the root. We will denote the root of tree $\mathcal{T}$ by $\mathsf{root}_{\mathcal{T}}$ (or just root if $\mathcal{T}$ is clear from the context). The orientation of the edges specifies how processes interact in the tree: $(p, q) \in Ch$ means process p is the parent of process q.

We let M denote a finite set of message contents, and we write:

- $\mathtt{Dw} = \{\downarrow m \mid m \in M\}$ for the set of synchronization actions with some child process (downwards synchronizations).
- $\mathtt{Up} = \{\uparrow m \mid m \in M\}$ for the set of synchronization actions with the parent process (upwards synchronizations).

By $Act = \mathtt{Dw} \cup \mathtt{Up} \cup \{\tau\}$ we denote the set of all actions (with τ an internal action).

A communicating automaton is a finite set of processes that exchange messages, each process being given as a finite LTS. In the parametric setting, the number of processes is not known beforehand. All processes have the same state space, as in rendez-vous protocols [20].

▶ **Definition 2.1.** *A* Parametrized Communicating Automaton *(*PCA *for short) is a tuple* (S, Δ, s^{init}). *Here* S *is the finite set of* states, $\Delta \subseteq S \times Act \times S$ *the (finite) set of* transitions, *and* $s^{init} \in S$ *the* initial *state.*

We will use *synchronous communication* as synchronization mechanism. Given a PCA $\mathcal{A} = (S, \Delta, s^{init})$ and a (tree) topology $\mathcal{T} = (P, Ch)$, we define a *configuration* as a total function $\mathbf{c} : P \to S$ mapping each process to its current state. We denote by $\mathcal{C}_{\mathcal{A},\mathcal{T}}$ the set of all configurations of $\mathcal{A}$ over $\mathcal{T}$. We call $\mathbf{c}^{init}_{\mathcal{A},\mathcal{T}}$ the initial configuration of $\mathcal{A}$ over $\mathcal{T}$, with $\mathbf{c}^{init}_{\mathcal{A},\mathcal{T}}(p) = s^{init}$ for all $p \in P$ (we write $\mathbf{c}^{init}$ when $\mathcal{A}$ and $\mathcal{T}$ are clear from the context).

To define transitions we need to enrich actions by process names. We denote by $Act_{\mathcal{T}} = \{p \bowtie q(m) \mid (p, q) \in Ch, m \in M\} \cup \{p : \tau \mid p \in P\}$ the set of synchronous communication actions. As expected, $p \bowtie q(m)$ is a synchronization between p and q with content m.

The transition relation $\to_{\mathcal{A},\mathcal{T}} \subseteq \mathcal{C}_{\mathcal{A},\mathcal{T}} \times Act_{\mathcal{T}} \times \mathcal{C}_{\mathcal{A},\mathcal{T}}$ over the set $\mathcal{C}_{\mathcal{A},\mathcal{T}}$ of configurations of $\mathcal{A}$ over $\mathcal{T}$ is defined as expected. Given two configurations $\mathbf{c}, \mathbf{c}' \in \mathcal{C}_{\mathcal{A},\mathcal{T}}$, we let $\mathbf{c} \xrightarrow{a}_{\mathcal{A},\mathcal{T}} \mathbf{c}'$ if either is true:

- $a = p \bowtie q(m)$ for some $(p, q) \in Ch$ and $m \in M$, $\mathbf{c}(p) \xrightarrow{\downarrow m} \mathbf{c}'(p) \in \Delta$, $\mathbf{c}(q) \xrightarrow{\uparrow m} \mathbf{c}'(q) \in \Delta$, and $\mathbf{c}(t) = \mathbf{c}'(t)$ for all $t \in P \backslash \{p, q\}$.
- $a = p : \tau$ for some $p \in P$, $\mathbf{c}(p) \xrightarrow{\tau} \mathbf{c}'(p) \in \Delta$ and $\mathbf{c}(q) = \mathbf{c}'(q)$ for all $q \neq p$.

A run of a PCA $\mathcal{A}$ over a topology $\mathcal{T}$ is a sequence $\rho = \mathbf{c}_0, a_1, \mathbf{c}_1, \ldots, a_n, \mathbf{c}_n$ such that $\mathbf{c}_0 = \mathbf{c}^{init}_{\mathcal{A},\mathcal{T}}$, and $\mathbf{c}_i \xrightarrow{a_{i+1}}_{\mathcal{A},\mathcal{T}} \mathbf{c}_{i+1}$ for all $0 \leq i < n$. We call $a_1 \cdots a_n$ the *trace* of the run ρ, and denote it by $\mathsf{trace}(\rho)$. A configuration $\mathbf{c}$ is *reachable* if there is a run ending in $\mathbf{c}$.

The projection of a trace $u \in Act_{\mathcal{T}}{}^*$ over a process p, written as $u|_p \in Act^*$, is the subsequence of actions in which p participates: $p \bowtie q(m)$ projects to $\downarrow m$, $q \bowtie p(m)$ projects to $\uparrow m$, and $p : \tau$ projects to τ.

The size of a PCA $\mathcal{A} = (S, Act, \Delta, s^{init})$ is defined as $|S| + |\Delta|$.

A common task in parametrized verification is to check safety, i.e., that no error state is reachable, no matter how many processes participate in a run. Formally, the Safety problem asks, given a PCA $\mathcal{A}$ and a state $s \in S$, whether for all rooted trees $\mathcal{T}$, there is no run ending in some configuration $\mathbf{c}$ such that $\mathbf{c}(\mathsf{root}_{\mathcal{T}}) = s$. For ease of reading, we will reason on the dual, Coverability problem:

Coverability	
Input:	A PCA $\mathcal{A} = (S, \Delta, s^{init})$ and a state $s \in S$.
Output:	Yes if there exists some rooted tree $\mathcal{T}$ and a run of $\mathcal{A}$ over $\mathcal{T}$ ending in some configuration $\mathbf{c}$ such that $\mathbf{c}(\mathsf{root}_{\mathcal{T}}) = s$.

▶ Remark 2.2. The reader may have noticed that all processes have the same underlying LTS, unlike our example in Figure 1. However, we can simulate systems where the root has a different LTS, say LTS_{root}, by letting a process choose at the beginning of the run between LTS_{root} and the LTS of the non-root processes. For this we create a new LTS as the disjoint union of $\mathrm{LTS}_{\mathsf{root}}$ (with initial state s^{init}_{root}) and the LTS of the non-root processes (with initial state s_0), adding a new initial state s^{init}. State s^{init} has only two τ-transitions, leading to s^{init}_{root} and to s_0. We also remove all upward actions from $\mathcal{S}_{\mathsf{root}}$, and set the final state we want to reach s_f in $\mathcal{S}_{\mathsf{root}}$. This way we can ensure that the root has chosen $\mathcal{S}_{\mathsf{root}}$ if s_f is reached; if a non-root process chooses to go to $\mathcal{S}_{\mathsf{root}}$ then it cannot receive messages from its parent, so its subtree is disconnected. ⌟

The Coverability problem defined above is undecidable, as to be expected because processes on a branch of the tree are linearly ordered [4]. A simple argument for undecidability in our setting is to reduce the halting problem for Minsky machines with two counters, using a branch to encode each counter in unary, together with a special symbol at the end of the branch to do zero tests (see also [18] for a similar construction).

In order to regain decidability we will study the behavior of PCA under two restrictions. The first one, arguably the most natural, is to bound the depth of trees. The second one is to bound the number of alternations each process can do between synchronizing downwards and upwards in the tree.

A run ρ is said to be *k-phase-bounded* if for every process $p \in P$, $\mathsf{trace}(\rho)|_p \in ((\mathtt{Dw} + \tau)^* + (\mathtt{Up} + \tau)^*)^{\leq k}$. In the example given in Figure 1, the tree depth is bounded by d, and the phase number for each process is bounded by 3: first receive an URL from the parent, then send URLs and receive URL lists from the children, and finally send an URL list back to the parent. A state $s \in S$ is *(d, k)-coverable* in $\mathcal{A}$ if there exists a tree $\mathcal{T}$ of depth at most d and a k-phase-bounded run ending in some configuration $\mathbf{c}$ such that $\mathbf{c}(\mathsf{root}_{\mathcal{T}}) = s$. By writing $d = \infty$ we mean that there is no bound on the depth of the topology. Likewise, for $k = \infty$ we mean that there is no bound on the number of phases.

We will consider four variants of the problem, depending on whether d and k are fixed or are part of the input. If d and/or k are inputs of the problem, they are in $\mathbb{N}$ and given in unary, otherwise they can be in $\mathbb{N} \cup \{\infty\}$, with ∞ meaning unbounded.

(d,k)-Coverability	
Input:	A PCA $\mathcal{A}$ and a state $s \in S$.
Output:	Yes if s is (d,k)-coverable in $\mathcal{A}$.

Coverability(d,k)	
Input:	A PCA $\mathcal{A}$, a state $s \in S$ and two integers $d, k \in \mathbb{N}$.
Output:	Yes if s is (d,k)-coverable in $\mathcal{A}$.

(d)-Coverability(k)	
Input:	A PCA $\mathcal{A}$, a state $s \in S$ and an integer $k \in \mathbb{N}$.
Output:	Yes if s is (d,k)-coverable in $\mathcal{A}$.

(k)-Coverability(d)	
Input:	A PCA $\mathcal{A}$, a state $s \in S$ and an integer $d \in \mathbb{N}$.
Output:	Yes if s is (d,k)-coverable in $\mathcal{A}$.

The next two tables summarize our results:

<table>
<tr><th>d \ k</th><th>1</th><th>≥ 2</th><th>∞</th></tr>
<tr><td>1</td><td colspan="3">Expspace-c</td></tr>
<tr><td>≥ 2</td><td rowspan="2">(Lem 3.1)</td><td rowspan="2">Expspace-c
(Lem 3.1, Thm 4.7)</td><td>between $\mathbf{F}_{\Omega_{d-1}}$
and $\mathbf{F}_{\Omega_d}$(Lem 3.4, Lem 3.5)</td></tr>
<tr><td>∞</td><td>Undec</td></tr>
</table>

Figure 2 Complexity of (d,k)-Coverability for PCA.

d	(d)-Coverability(k)
1	Expspace-c (Lem 3.1)
≥ 2	2Expspace-c
∞	(Thm 4.4, Thm 4.7)

k	(k)-Coverability(d)
≥ 1	Expspace-c (Lem 3.1, Thm 4.7)
∞	$\mathbf{F}_{\varepsilon_0}$-c (Lem 3.4, Lem 3.5)

Coverability(d,k)
2Expspace-c (Thm 4.4, Thm 4.7)

Figure 3 Complexity of Coverability for PCA when d and/or k are part of the input.

Let us comment on some boundary cases for our parameters. If we restrict to 1-phase-bounded runs, the root process can talk to its children; but those processes will not be able anymore to communicate with their children. Thus a state is $(\infty,1)$-coverable iff it is $(1,1)$-coverable. Similarly, if the depth is one then every run is 1-phase-bounded, so a state is $(1,\infty)$-coverable iff it is $(1,1)$-coverable.

3 Depth-bounded PCA

In this section, we will show that coverability of PCA on bounded-depth trees is closely related to coverability in Nested Counter Systems [16, 5]. We start by showing that $(1,k=\infty)$-coverability is equivalent to VASS coverability.

3.1 Depth one PCA

Let m be a positive integer. An *m-VASS* is a tuple $\mathcal{V} = (Q, q^{init}, \Delta)$ where Q is a finite set of states, q^{init} is the initial state, and $\Delta \subseteq Q \times \mathbb{Z}^m \times Q$ a transition relation.

A *configuration* of $\mathcal{V}$ is a tuple $(q, \mathbf{x})$ where $q \in Q$ is a state and $\mathbf{x} \in \mathbb{N}^m$ is a vector of non-negative integers. The initial configuration is $(q^{init}, \mathbf{0})$. The step relation between two configurations is defined by $(q, \mathbf{x}) \xrightarrow{\mathbf{v}} (q', \mathbf{x}')$ if $(q, \mathbf{v}, q') \in \Delta$ and $\mathbf{x}' = \mathbf{x} + \mathbf{v} \geq \mathbf{0}$. It is understood that $\leq$ denotes the component-wise extension of the usual order $\leq$ on $\mathbb{N}$. A run of $\mathcal{V}$ is a sequence $\sigma = (q_0, \mathbf{x}_0) \xrightarrow{\mathbf{v}_1} (q_1, \mathbf{x}_1) \xrightarrow{\mathbf{v}_2} \dots \xrightarrow{\mathbf{v}_n} (q_n, \mathbf{x}_n)$, where $(q_0, \mathbf{x}_0)$ is initial.

Let $(q, \mathbf{x})$ and $(q', \mathbf{x}')$ be two configurations. We say that $(q', \mathbf{x}')$ covers $(q, \mathbf{x})$ if $q = q'$ and $\mathbf{x} \leq \mathbf{x}'$, and we denote this as $(q, \mathbf{x}) \preceq (q', \mathbf{x}')$.

The size of an m-VASS $\mathcal{V}$ is $|\mathcal{V}| = |Q| + \sum_{(q,\mathbf{v},q') \in \Delta} \|\mathbf{v}\|_1$ where $\|\mathbf{v}\|_1 = \sum_{i=1}^{m} |\mathbf{v}[i]|$.

We recall the coverability problem for VASS:

VASS-Coverability	
Input:	An m-VASS $\mathcal{V}$ and a state $q \in Q$.
Output:	Yes if $(q, \vec{0})$ is coverable in $\mathcal{V}$.

This problem is known to be Expspace-complete [26, 28].

To show that $(1, k = \infty)$-Coverability for PCA is in Expspace, we can use counter abstraction [20]. We simulate a PCA $\mathcal{A} = (S, Act, \Delta, s^{init})$ on trees of depth 1 by counting how many children of the root are in a given state. Thus we construct in polynomial time an m-VASS $\mathcal{V}$ with $m = |S|$, in which coverability for a state $s \in S$ can be checked in exponential space [28].

For the lower bound, we reduce VASS-Coverability to $(1, k = \infty)$-Coverability. Let $\mathcal{V}$ be an m-VASS. We construct a PCA $\mathcal{A}$ where process root keeps track of the state and simulates the transitions of $\mathcal{V}$, while the children will store the counter values. Any child process will be either in state s^{init} or in s_i $(1 \leq i \leq m)$, and the number of children in state s_i will represent the value of the i-th counter.

To do so, we replace each transition $q \xrightarrow{\mathbf{v}} q'$ of $\mathcal{V}$ by a series of transitions that will move $\mathbf{v}[i]$ children from state s^{init} to state s_i if $\mathbf{v}[i] \geq 0$, and will move $-\mathbf{v}[i]$ children from state s_i to state s^{init} if $\mathbf{v}[i] < 0$, for each $1 \leq i \leq m$.

From this we get that configuration $(q, \vec{0})$ is coverable in $\mathcal{V}$ iff there exists a tree $\mathcal{T}$ of depth 1 and some configuration $\mathbf{c} \in \mathcal{C}_{\mathcal{A},\mathcal{T}}$ reachable in $\mathcal{A}$ over $\mathcal{T}$ where $\mathbf{c}(\mathsf{root}) = q$.

▶ **Lemma 3.1.** *$(1, k = \infty)$-Coverability for PCA is* Expspace*-complete.*

3.2 Coverability for Nested Counter Systems

We have seen that for depth one, PCA are equivalent to VASS. It seems natural to look at nested counters for PCA over trees of fixed depth. We will show in the next subsection a tight correspondence between PCA over trees of depth at most d and d-Nested Counter Systems [16, 5]. The latter manipulate higher-order counters, in a similar manner to Nested Petri Nets [27]. We first recall the definition of Nested Counter Systems.

Let $d \in \mathbb{N}$ be an integer. A *d-Nested Counter System* (d-NCS) is a tuple $\mathcal{N} = (Q, \Delta)$ where Q is a finite set of *states* and $\Delta \subseteq \bigcup_{1 \leq i,j \leq d+1} (Q^i \times Q^j)$ is a finite set of *rules*. The set of *configurations* $\mathcal{C}_\mathcal{N}$ of $\mathcal{N}$ is is defined as the set of all labelled rooted trees of depth at most d, with labels from the set Q. Formally, a configuration $C \in \mathcal{C}_\mathcal{N}$ is a tuple $C = (V, E, \mathsf{root}, h)$ where (V, E, root) is a rooted tree of depth at most d, and $h : V \to Q$ is a labeling function.

The transition relation $\to \subseteq \mathcal{C}_\mathcal{N} \times \mathcal{C}_\mathcal{N}$ on configurations is defined as follows. Let $r = ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \Delta$ be a rule, with $0 \leq i \leq j \leq d$. We say that a configuration C moves to some configuration C' using rule r (written $C \xrightarrow{r} C'$) if there is a path $v_0, \dots, v_i$ in C starting at the root labeled by $q_0, \dots, q_i$, and C' is obtained from C by changing the

label of each v_k to q'_k for $0 \leq k \leq i$, and for $i < k \leq j$ creating a new vertex v_k as the child of v_{k-1}, labeled by q'_k. Similarly, suppose $r = ((q_0, \ldots, q_i), (q'_0, \ldots, q'_j)) \in \Delta$ is a rule, with $0 \leq j < i \leq d$. Then $C \xrightarrow{r} C'$ if there is a path $v_0, \ldots, v_i$ in C starting at the root and labeled by $q_0, \ldots, q_i$, and C' is obtained from C by changing the label of each v_k to q'_k for $0 \leq k \leq j$ and removing the subtree rooted at v_{j+1}.

We write $C \to C'$ if there exists a rule $r \in \Delta$ such that $C \xrightarrow{r} C'$. Moreover, we say that C' is *reachable* from C (written $C \xrightarrow{*} C'$) if there exist $C_0, \ldots, C_n$, $n \geq 0$, such that $C = C_0 \to \cdots \to C_n = C'$. We say that C' *covers* C (written $C \preceq C'$) if C is an induced subtree of C' with the same root. The coverability problem for NCS is:

d-NCS Coverability	
Input:	A d-NCS $\mathcal{N}$ and two configurations $C_i, C_f \in \mathcal{C}_\mathcal{N}$.
Output:	Yes if there exists C such that $C_i \xrightarrow{*} C$ and $C_f \preceq C$.

It is readily seen that 1-NCS are equivalent to VASS, so 1-NCS Coverability is Expspace-complete. To state the complexity of d-NCS Coverability with $d \geq 2$, we need to introduce the *fast-growing hierarchy* of complexity classes [29].

We represent ordinal numbers using the *Cantor Normal Form* (CNF) as $\alpha = \omega^{\alpha_1} + \cdots + \omega^{\alpha_n}$, where $\alpha_1 \geq \cdots \geq \alpha_n$ are ordinals also written in CNF. We denote *limit ordinals* by λ. These are ordinals such that $\alpha + 1 < \lambda$ for every $\alpha < \lambda$.

A *fundamental sequence* for a limit ordinal λ is a sequence of ordinals $(\lambda_n)_{n\in\mathbb{N}}$ with supremum λ. We consider the same fundamental sequence as in [16, 7], which is defined by

$$(\alpha + \omega^{\beta+1})_n = \alpha + \underbrace{\omega^\beta + \cdots + \omega^\beta}_{n} \qquad \text{and} \qquad (\alpha + \omega^{\lambda'})_n = \alpha + \omega^{\lambda'_n}$$

for ordinals β and limit ordinals λ'. For example, a fundamental sequence for ω^2 is given by $(\omega \cdot n)_{n\in\mathbb{N}}$, a fundamental sequence for ω^ω by $(\omega^n)_{n\in\mathbb{N}}$, etc.

The *fast-growing hierarchy* is the ordinal-indexed family of functions $F_\alpha : \mathbb{N} \to \mathbb{N}$ for $\alpha < \varepsilon_0$ inductively defined by

$$F_0(n) = n + 1 \qquad F_{\alpha+1}(n) = \underbrace{F_\alpha(\cdots(F_\alpha}_{n \text{ times}}(n))\cdots) \quad \text{and} \quad F_\lambda(n) = F_{\lambda_n}(n)\ .$$

The corresponding hierarchy $(\mathbf{F}_\alpha)_{\alpha<\varepsilon_0}$ of *fast-growing complexity classes* is defined in terms of the fast-growing functions F_α, see [29] for details. We recall that $\mathbf{F}_\omega$ corresponds to the class of problems solvable in Ackermannian time, and $\mathbf{F}_{\omega^\omega}$ to the class of problems solvable in hyper-Ackermannian time.

To state what is known about the complexity of coverability for NCS, we let Ω_n denote the tower of ω's of height $n-1$, i.e., $\Omega_1 = \omega$ and $\Omega_n = \omega^{\Omega_{n-1}}$. The ordinal ε_0 is the limit ordinal with fundamental sequence $(\Omega_n)_{n\in\mathbb{N}}$.

▶ **Theorem 3.2** ([16, 7]). *The complexity of d-NCS Coverability is between $\mathbf{F}_{\Omega_{d-1}}$ and $\mathbf{F}_{\Omega_d}$ when $d \geq 2$ is fixed, and it is $\mathbf{F}_{\varepsilon_0}$-complete when d is part of the input.*

The complexity of d-NCS Coverability for a fixed $d \geq 2$ was shown in [16] to be between $\mathbf{F}_{\Omega_{d-1}}$ and $\mathbf{F}_{\Omega_{2d}}$. However, as pointed out in [7], the proof of the lower bound is flawed. The flawed construction of [16] is fixed in [7], albeit for the extended model of NRCS that adds reset operations to NCS. More precisely, it is shown in [7] that d-NRCS Coverability is $\mathbf{F}_{\Omega_d}$-complete (so the upper bound is reduced to $\mathbf{F}_{\Omega_d}$, despite the additional reset operations).

The lower bound of [7] does not (at least directly) apply to d-NCS Coverability. We show in the next subsection that d-NRCS Coverability can be reduced to $(d+1)$-NCS Coverability. This provides a proof of the claimed lower bound of [16].

3.3 From Nested Reset Counter Systems to Nested Counter Systems

This subsection presents a simulation of Nested Reset Counter Systems by plain Nested Counter Systems. We first recall the definition of Nested Reset Counter Systems from [7].

A *d-Nested Reset Counter System* (d-NRCS) is a tuple $\mathcal{N} = (Q, \Delta_u, \Delta_r)$ where Q is a finite set of *states*, $\Delta_u \subseteq \bigcup_{1 \leq i,j \leq d+1}(Q^i \times Q^j)$ is a finite set of *update rules*, and $\Delta_r \subseteq \bigcup_{1 \leq i \leq d}(Q^i \times Q \times Q^i)$ is a finite set of *reset rules*. For clarity, an update rule $((q_0, \ldots, q_i), (q'_0, \ldots, q'_j)) \in \Delta_u$ is written $(q_0, \ldots, q_i) \xrightarrow{\text{upd}} (q'_0, \ldots, q'_j)$ and a reset rule $((q_0, \ldots, q_i), p, (q'_0, \ldots, q'_i)) \in \Delta_r$ is written $(q_0, \ldots, q_i) \xrightarrow{\text{rst}(p)} (q'_0, \ldots, q'_i)$. Notice that a d-NCS (as defined in the previous subsection) is a tuple (Q, Δ) such that $(Q, \Delta, \emptyset)$ is a d-NRCS.

The set of *configurations* $\mathcal{C}_\mathcal{N}$ of a d-NRCS $\mathcal{N} = (Q, \Delta_u, \Delta_r)$ is identical to the set of configurations of the d-NCS (Q, Δ_u). Similarly, the transition relation $\xrightarrow{r}$ of an update rule $r \in \Delta_u$ is identical to the transition relation $\xrightarrow{r}$ of the d-NCS (Q, Δ_u). Given a reset rule $r = (q_0, \ldots, q_i) \xrightarrow{\text{rst}(p)} (q'_0, \ldots, q'_i)$ with $0 \leq i < d$, we say that a configuration C moves to some configuration C' using rule r (written $C \xrightarrow{r} C'$) if there is a path $v_0, \ldots, v_i$ in C starting at the root labeled by $q_0, \ldots, q_i$, and C' is obtained from C by changing the label of each v_k to q'_k for $0 \leq k \leq i$, and removing every subtree rooted at a child v of v_i such that v is labeled by p. The coverability problem for NRCS is defined in the same way as for NCS.

▶ **Lemma 3.3.** *For every fixed $d \geq 1$, the d-NRCS Coverability problem is reducible in polynomial time to the $(d+1)$-NCS Coverability problem.*

The rest of this subsection presents our reduction. We focus on reset rules $(q_0, \ldots, q_i) \xrightarrow{\text{rst}(p)} (q'_0, \ldots, q'_i)$ that are applied to nodes of depth at least one (the depth of the root is zero), i.e., such that $i \geq 1$. Our recipe to simulate such a reset rule on a configuration C may be summarized as follows:

1. pick a path $v_0, \ldots, v_i$ in C starting at the root labeled by $q_0, \ldots, q_i$,
2. change the label of each v_k to q'_k for $0 \leq k \leq i$,
3. copy the subtree rooted at the node v_i except for the children v of v_i that are labeled with p, and
4. delete the subtree rooted at v_i.

It is understood that the root v'_i of the copy performed in step (3) is a child of v_{i-1} (i.e., v'_i is a sibling of v_i). As we cannot create a sibling of the root, this approach only works for $i \geq 1$.

The full version [15] of the paper [16] shows, albeit succinctly, how to copy a subtree in lossy manner. The main idea is to copy the source subtree twice, in DFS fashion, using a label `i` to mark the source node and two labels $\mathtt{o}_1$ and $\mathtt{o}_2$ to mark the copied nodes. The source subtree is deleted along the way (when the DFS moves up).

Our simulation of reset rules (formally presented below) is inspired from the copy gadget of [15]. This simulation only uses update rules and is *lossy* in the following sense:

- any configuration obtained after the gadget is $\preceq$-smaller than the configuration obtained after the reset rule, and
- there is a run of the gadget that is not lossy, i.e., that provides the configuration obtained after the reset rule.

We want to stress that this simulation only works for reset rules $(q_0, \dots, q_i) \xrightarrow{\mathrm{rst}(p)} (q'_0, \dots, q'_i)$ with $i \geq 1$. To simulate an arbitrary d-NRCS $\mathcal{N} = (Q, \Delta_u, \Delta_r)$ containing possibly reset rules $(q_0) \xrightarrow{\mathrm{rst}(p)} (q'_0)$ that apply to the root, one may simply introduce an extra level, i.e., consider the $(d+1)$-NRCS $\mathcal{N}' = (Q \cup \{x\}, \Delta'_u, \Delta'_r)$ where each update rule $(q_0, \dots, q_i) \xrightarrow{\mathrm{upd}} (q'_0, \dots, q'_j)$ is replaced by $(x, q_0, \dots, q_i) \xrightarrow{\mathrm{upd}} (x, q'_0, \dots, q'_j)$ and each reset rule $(q_0, \dots, q_i) \xrightarrow{\mathrm{rst}(p)} (q'_0, \dots, q'_i)$ is replaced by $(x, q_0, \dots, q_i) \xrightarrow{\mathrm{rst}(p)} (x, q'_0, \dots, q'_i)$. The $(d+1)$-NRCS $\mathcal{N}'$ contains no reset rule on the root, so it can be transformed into an equivalent (with respect to coverability) $(d+1)$-NCS.

Let us now formally present our simulation of reset rules with update rules. Consider a d-NRCS $\mathcal{N} = (Q, \Delta_u, \Delta_r)$ and a reset rule $r = (q_0, \dots, q_i) \xrightarrow{\mathrm{rst}(p)} (q'_0, \dots, q'_i)$ with $d > i \geq 1$. We introduce new states $\mathtt{init}(r)$, $\mathtt{par}(r)$, $\mathtt{cp}(r)$, $\mathtt{cpd}(r, q)$, $\mathtt{cpu}(r)$, $\mathtt{src}(q)$ and $\mathtt{dst}(q)$, where q ranges over Q. The rules of the gadget simulating r are given below. In these rules,

- σ stands for the sequence $\sigma = q'_1, \dots, q'_{i-1}$,
- m stands for an arbitrary integer in $\{0, \dots, d-i-1\}$, and
- $x_1, \dots, x_m, y, z$ stand for arbitrary states in Q (note that the sequence $x_1, \dots, x_m$ is possibly empty).

As d is fixed, the gadget simulating r contains polynomially many rules.

Initialization: case $i = 1$.

$$\begin{aligned} (q_0, q_1) &\xrightarrow{\mathrm{upd}} (\mathtt{init}(r), \mathtt{src}(q'_1)) \\ (\mathtt{init}(r)) &\xrightarrow{\mathrm{upd}} (\mathtt{cp}(r), \mathtt{dst}(q'_1)) \end{aligned}$$

Initialization: case $i \geq 2$.

$$\begin{aligned} (q_0, \dots, q_i) &\xrightarrow{\mathrm{upd}} (\mathtt{init}(r), q'_1, \dots, q'_{i-2}, \mathtt{par}(r), \mathtt{src}(q'_i)) \\ (\mathtt{init}(r), q'_1, \dots, q'_{i-2}, \mathtt{par}(r)) &\xrightarrow{\mathrm{upd}} (\mathtt{cp}(r), q'_1, \dots, q'_{i-2}, q'_{i-1}, \mathtt{dst}(q'_i)) \end{aligned}$$

Move down and copy.

$$\begin{aligned} (\mathtt{cp}(r), \sigma, x_1, \dots, x_m, \mathtt{src}(y), z) &\xrightarrow{\mathrm{upd}} (\mathtt{cpd}(r, z), \sigma, x_1, \dots, x_m, y, \mathtt{src}(z)) \\ (\mathtt{cpd}(r, z), \sigma, x_1, \dots, x_m, \mathtt{dst}(y)) &\xrightarrow{\mathrm{upd}} (\mathtt{cp}(r), \sigma, x_1, \dots, x_m, y, \mathtt{dst}(z)) \end{aligned}$$

Recall that we want to copy the subtree rooted at the node v_i except for the children v of v_i that are labeled with p. To do so, **the above two rules are *not* included when $m = 0$ and $z = p$.**

Move up and delete source.

$$\begin{aligned} (\mathtt{cp}(r), \sigma, x_1, \dots, x_m, y, \mathtt{src}(z)) &\xrightarrow{\mathrm{upd}} (\mathtt{cpu}(r), \sigma, x_1, \dots, x_m, \mathtt{src}(y)) \\ (\mathtt{cpu}(r), \sigma, x_1, \dots, x_m, y, \mathtt{dst}(z)) &\xrightarrow{\mathrm{upd}} (\mathtt{cp}(r), \sigma, x_1, \dots, x_m, \mathtt{dst}(y), z) \end{aligned}$$

End.

$$\begin{aligned} (\mathtt{cp}(r), \sigma, \mathtt{src}(y)) &\xrightarrow{\mathrm{upd}} (\mathtt{end}(r), \sigma) \\ (\mathtt{end}(r), \sigma, \mathtt{dst}(y)) &\xrightarrow{\mathrm{upd}} (q'_0, \sigma, y) \end{aligned}$$

Note that in the two rules above we could equivalently replace y by q'_i.

3.4 From PCA to Nested Counter Systems and back

We show in the remainder of this section that $(d, k = \infty)$-Coverability for PCA is (polynomially-) equivalent to d-NCS Coverability. We start with the upper bound, by providing a reduction from coverability for PCA over trees of depth at most d to coverability for NCS of nesting depth at most d.

Let $\mathcal{A} = (S_\mathcal{A}, Act, \Delta_\mathcal{A}, s^{init})$ be a PCA and $d \in \mathbb{N}$ a positive integer. We construct a d-NCS $\mathcal{N} = (Q_\mathcal{N}, \Delta_\mathcal{N})$, with $Q_\mathcal{N} = S_\mathcal{A} \cup \{\mathsf{start}\}$ and $\Delta_\mathcal{N}$ defined as follows. For the initialization, the root is in state start and it generates the tree with a set of rules spawning new levels in state s^{init}. The initialization is finished with the root going from start to s^{init}: For all $0 \leq i < d$, $((\mathsf{start}, \underbrace{s^{init}, \ldots, s^{init}}_{i \text{ times}}), (\mathsf{start}, \underbrace{s^{init}, \ldots, s^{init}}_{i \text{ times}}, s^{init})) \in \Delta_\mathcal{N}$, and $((\mathsf{start}), (s^{init})) \in \Delta_\mathcal{N}$.

For each transition, we have one rule per tree level: For all $q \xrightarrow{\downarrow m} q' \in \Delta_\mathcal{A}$, $s \xrightarrow{\uparrow m} s' \in \Delta_\mathcal{A}$, $0 \leq i < d$, and $q_0, \ldots, q_{i-1} \in S_\mathcal{A}$, we let $((q_0, \ldots, q_{i-1}, q, s), (q_0, \ldots, q_{i-1}, q', s')) \in \Delta_\mathcal{N}$; for all $q \xrightarrow{\tau} q' \in \Delta_\mathcal{A}$, $0 \leq i \leq d$, and $q_0, \ldots, q_{i-1} \in S_\mathcal{A}$, we have $((q_0, \ldots, q_{i-1}, q), (q_0, \ldots, q_{i-1}, q')) \in \Delta_\mathcal{N}$.

This NCS simulates $\mathcal{A}$ for every tree of depth at most d. Let $\mathcal{T}$ be a tree of depth at most d. For every configuration $\mathbf{c}$ of $\mathcal{A}$ over $\mathcal{T}$ we can construct a configuration C of $\mathcal{N}$ over $\mathcal{T}$ such that $h(p) = \mathbf{c}(p)$. We have that a configuration $\mathbf{c}$ is reachable in $\mathcal{A}$ on trees of depth at most d iff there exists a configuration C of $\mathcal{N}$ simulating $\mathbf{c}$ that is reachable from the configuration start. So a state s is $(d, k = \infty)$-coverable in $\mathcal{A}$ iff the configuration with the root in state s is coverable from configuration start in $\mathcal{N}$.

▶ **Lemma 3.4.** *The complexity of $(d, k = \infty)$-Coverability for PCA (with $d \in \mathbb{N}$) is in $\mathbf{F}_{\Omega_d}$, and the complexity of $(k = \infty)$-Coverability(d) is in $\mathbf{F}_{\varepsilon_0}$.*

For the lower bound we show how to simulate a d-NCS with a PCA on trees of depth at most d. Informally, we use each process to store one state of $\mathcal{N}$, with its children representing its subconfiguration. When doing a transition, we nondeterministically synchronize a branch of the tree, making sure every node of the branch has the correct state, and then change each state along the branch.

Let $d \in \mathbb{N}$ be a positive integer, $\mathcal{N} = (Q_\mathcal{N}, \Delta_\mathcal{N})$ a d-NCS, and C^{init} and C^{final} two configurations of $\mathcal{N}$. For the reduction it is convenient to construct a PCA $\mathcal{A}$ with a distinct LTS for the root (see Remark 2.2). The set of states of $\mathcal{B}$ is $Q_\mathcal{N}$, plus sets of intermediary states used in initialization, application of rules and termination. The transitions are obtained as follows. For every rule, we initialize the transition and choose a branch by synchronizing processes from the root, checking at the same time the states on the path. Then we resolve the transition from the process at the deepest level to the root. If the rule removes part of the configuration, we put the corresponding processes in a state s^{dead}, in which they will not be able to synchronize anymore, cutting the subconfiguration from the rest of the tree.

For the initialization, we construct the desired configuration of $\mathcal{A}$ from the leaves to the root. Each leaf sends the vertex it represents to its parent, and the latter waits to receive every vertex of its children before sending itself to its parent, and we repeat this up to root_C. To check if the final configuration is covered, we use a similar construction.

A formal definition of the transitions used in this reduction can be seen in the full version [17].

The relationship between $\mathcal{N}$ and $\mathcal{A}$ can be formalised by the following notion of simulation. Let $C = (V, E, \mathsf{root}_C, h)$ be a configuration of $\mathcal{N}$, and $\mathbf{c}$ be a configuration of $\mathcal{A}$ over some tree $\mathcal{T} = (P, Ch)$. We say that $\mathbf{c}$ simulates C if all conditions below hold:

- C is a subtree of $\mathcal{T}$ with the same root.
- For every $v \in V$, $\mathbf{c}(v) = h(v)$.
- For every $v \in P \backslash V$, $\mathbf{c}(v) = s^{init}$, $\mathbf{c}(v) = s^{dead}$ or there exists a ancestor v' of v in $P \backslash V$ such that $\mathbf{c}(v') = s^{dead}$.

For $C^{init} = (V, E, \mathsf{root}_C, h)$, and for every tree $\mathcal{T}$ that has (V, E) as subtree, the first configuration $\mathbf{c}$ reachable by $\mathcal{A}$ over $\mathcal{T}$ such that $\mathbf{c}(\mathsf{root}) \in Q_{\mathcal{N}}$ simulates C^{init}. If $\mathcal{T}$ is not big enough, then root will never reach a state in $Q_{\mathcal{N}}$. We also have that for every configuration C of $\mathcal{N}$ reachable from C^{init}, there exists a tree $\mathcal{T}$ and a reachable configuration $\mathbf{c}$ of $\mathcal{A}$ over $\mathcal{T}$ that simulates C. Conversely, for any tree $\mathcal{T}$, if a reachable configuration $\mathbf{c}$ of $\mathcal{A}$ over $\mathcal{T}$ simulates some configuration C of $\mathcal{N}$, then this configuration is reachable from C^{init}. From this we get that C^{final} is coverable from C^{init} in $\mathcal{N}$ iff s^{final} is $(d, k = \infty)$-coverable in $\mathcal{A}$.

▶ **Lemma 3.5.** *The problem $(d, k = \infty)$-Coverability for PCA (with $d \in \mathbb{N}$) is $\mathbf{F}_{\Omega_{d-1}}$-hard, and $(k = \infty)$-Coverability(d) is $\mathbf{F}_{\varepsilon_0}$-hard.*

4 Phase-bounded PCA

In this section we present our results on the complexity of the coverability problem for phase-bounded PCA.

4.1 Lower bound for phase-bounded PCA

We start with the lower bound for (∞, k)-Coverability, so for k phases and no restriction on the depth. We show a reduction from the coverability problem for a succinct model of Petri nets, called Transducer Defined Petri Nets [8]. We recall that a Petri Net is given as a tuple $N = (P, T, F, p_0)$ with P a finite set of places, T a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ the flow relation, and $p_0 \in P$ the initial place. A marking of N is a function $\mathbf{m} \in P \to \mathbb{N}$, and we say that there are $\mathbf{m}(p)$ tokens on place p. A transition $t \in T$ is firable in a marking $\mathbf{m}$ if $m(p) > 0$ for all p such that $(p, t) \in F$. It can then be fired, leading to the marking $\mathbf{m}'$ with $\mathbf{m}'(p) = \mathbf{m}(p) + F(t, p) - F(p, t)$ for all $p \in P$. We also write $\mathbf{m} \xrightarrow{t} \mathbf{m}'$. We say that a marking $\mathbf{m}$ is reachable in N if there exists a sequence $\mathbf{m}_0 \xrightarrow{t_1} \mathbf{m}_1 \dots \mathbf{m}_{n-1} \xrightarrow{t_n} \mathbf{m}$ where $\mathbf{m}_0(p_0) = 1$ and $\mathbf{m}_0(p) = 0$ for all $p \neq p_0$. A marking $\mathbf{m}$ is coverable in N if there exists a marking $\mathbf{m}'$ such that $\mathbf{m}(p) \leq \mathbf{m}'(p)$ for all $p \in P$. The question whether a marking is coverable is Expspace-complete [26, 28], whereas the reachability of a marking is Ackermann-complete [25, 24, 14].

We fix an alphabet Σ. An *n-ary transducer* $\mathcal{M} = (Q, q_0, Q_f, \Delta)$ over alphabet Σ consists of a set of states Q, an initial state $q_0 \in Q$, a set of final states $Q_f \subseteq Q$ and a transition relation $\Delta \subseteq Q \times \Sigma^n \times Q$. We write $q \xrightarrow{a_1, \dots, a_n} q'$ for a transition between q and q'. The size of $\mathcal{M}$ is $|\mathcal{M}| = |\Delta|$. An n-ary transducer $\mathcal{M}$ defines the relation $\mathcal{L}(\mathcal{M})$ containing the n-tuples $(w_1, \dots, w_n)$ of words over Σ for which there exists a sequence of transitions $q_0 \xrightarrow{a_{1,1}, \dots, a_{n,1}} q_1 \xrightarrow{a_{1,2}, \dots, a_{n,2}} \dots \xrightarrow{a_{1,m}, \dots, a_{n,m}} q_m$ of $\mathcal{M}$ with $q_m \in Q_f$ and $a_{i,1} \dots a_{i,m} = w_i$ for all $i \in [1 \dots n]$.

▶ **Definition 4.1.** *A* transducer-defined Petri Net *(*TdPN*) $\mathcal{N} = (\ell, \mathcal{M}_{move}, \mathcal{M}_{join}, \mathcal{M}_{fork}, w_{init})$ consists of an integer $\ell \in \mathbb{N}$, a binary transducer $\mathcal{M}_{move}$ and two ternary transducers $\mathcal{M}_{join}, \mathcal{M}_{fork}$, all over input/output alphabet Σ, and an initial word $w_{init} \in \Sigma^\ell$. The associated Petri Net $PN(\mathcal{N}) = (P, T, F, p_0)$ has $P = \Sigma^\ell$ as set of places and set T of transitions defined as disjoint union of:*

- $T_m = \{(w, w') \in \Sigma^\ell \times \Sigma^\ell \mid (w, w') \in \mathcal{L}(\mathcal{M}_{move})\}$

- $T_j = \{(w, w', w'') \in \Sigma^\ell \times \Sigma^\ell \times \Sigma^\ell \mid (w, w', w'') \in \mathcal{L}(\mathcal{M}_{join})\}$ *and*
- $T_f = \{(w, w', w'') \in \Sigma^\ell \times \Sigma^\ell \times \Sigma^\ell \mid (w, w', w'') \in \mathcal{L}(\mathcal{M}_{fork})\}$

The initial place of PN($\mathcal{N}$) is $p_0 = w_{init}$. The semantics of transitions in terms of flow relation is:

- *if* $t = (p, p') \in T_m$ *then* $(p, t), (t, p') \in F$
- *if* $t = (p, p', p'') \in T_j$ *then* $(p, t), (p', t), (t, p'') \in F$
- *if* $t = (p, p', p'') \in T_f$ *then* $(p, t), (t, p'), (t, p'') \in F$

An accepting run of any of the transducers corresponds to a single transition of PN($\mathcal{N}$), and is called a transducer move. The size of $\mathcal{N}$ is defined as $|\mathcal{N}| = \ell + |\mathcal{M}_{move}| + |\mathcal{M}_{fork}| + |\mathcal{M}_{join}|$, with ℓ in unary encoding.

A marking $\mathbf{m}$ of a TdPN $\mathcal{N}$ is a total function $\mathbf{m} \in \Sigma^\ell \to \mathbb{N}$. Coverability of markings (or places) is defined as usual.

TdPN-Coverability	
Input:	A TdPN $\mathcal{N}$ and a word $w_{final} \in \Sigma^\ell$.
Output:	Yes if the place w_{final} is coverable in PN($\mathcal{N}$).

We will use the following result:

▶ **Theorem 4.2** ([8]). *TdPN-Coverability is 2Expspace-complete.*

By abuse of language, we call a total function $\mathbf{m} : \Sigma^l \to \mathbb{N}$ a marking of $\mathcal{N}$, and we say that there are $\mathbf{m}(w)$ tokens on place $w \in \Sigma^l$.

To reduce TdPN-Coverability to $(d = \infty)$-Coverability(k), we construct a PCA on a rooted tree of depth 2. Informally, the root simulates the transducers and each child of the root stores a word, representing the encoding of a place of the TdPN. To do so, a node at depth 1 stores in each of its children a letter of the word, together with its position from $[1 \dots \ell]$.

Each transition of the TdPN is initiated by the root by first synchronizing with one, two or three children (depending on the step), to define their role in the transition. Then root will communicate each letter of the words needed for that transition to the participating children, checking at the same time the validity of the transition of the chosen transducer. At the end of the transition, children that have participated as inputs (outputs, resp.) go to state dead (full, resp.). The latter state means that such nodes are ready to be used as inputs later. At a given point, the number of children in state full with children corresponding to some word $w \in \Sigma^\ell$ represents the number of tokens in the TdPN place named w.

In the reduction we use a different LTS for root. This is not a restriction, see Remark 2.2.

Let $\mathcal{N} = (\ell, \mathcal{M}_{move}, \mathcal{M}_{join}, \mathcal{M}_{fork}, w_{init})$ be the TdPN and $w_{final} \in \Sigma^\ell$ the final word. We construct a PCA $\mathcal{A}(\mathcal{N}, w_{final})$ with two distinct LTS, $\mathcal{A}_{\mathsf{root}}$ for the root and $\mathcal{A}$ for the other processes, over the same set of actions Act. The state space of $\mathcal{A}_{\mathsf{root}}$ will be the disjoint union of $\{init_r, final_r, idle\}$, a set of states for each of the two words w_{init} and w_{final}, and one set of states per transducer. The general structure of this LTS is shown in Figure 4.

The LTS $\mathcal{A}$ will have two tasks. The first one is for processes at level 1, and consists to store a word when asked by the root, and then send it back later. Such a process p is awakened when it synchronizes with root with the message $\mathsf{start}^{\mathrm{arg}}$ (for some $1 \leq \mathrm{arg} \leq 3$). Process p guesses each letter of a word of length ℓ, communicates it to one of its dormant children together with its position, and waits for the root to acknowledge the letter. After the whole word has been stored, p is in state *full*.

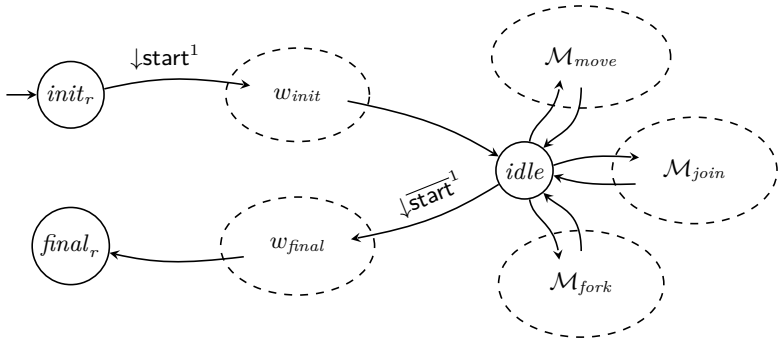

Figure 4 Automaton for root. $\overline{\mathsf{start}}^{\mathrm{arg}}$ means consuming a word, whereas $\mathsf{start}^{\mathrm{arg}}$ means creating a word.

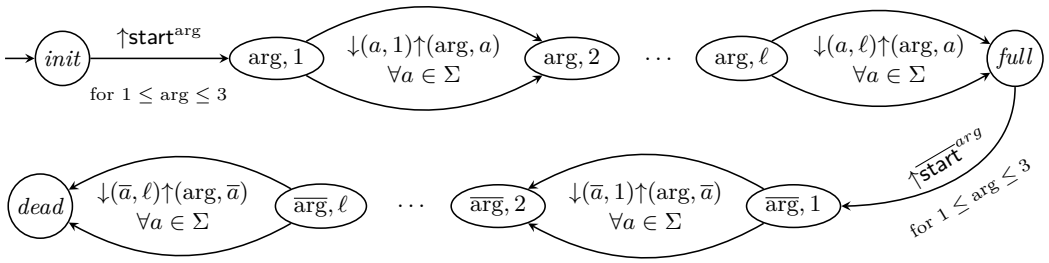

Figure 5 Automaton for processes at level 1.

Later, a level 1 process can be called by root to give its word back, when root synchronizes with it with some message $\overline{\mathsf{start}}^{\mathrm{arg}}$. It will then recover one by one the letters of the word from its children and relay the information to the root. After the word is sent, p is in state *dead*. The corresponding LTS is depicted in Figure 5. The second part of $\mathcal{A}$ will be for processes at level 2. Such processes will only synchronize to store a letter and a position, and wait to communicate back the corresponding letter and position. This LTS is depicted in Figure 6. Note that the states *init* and *dead* are common to both parts of this LTS.

We say that a configuration is *stable* if the root is in state *idle* and all processes of depth 1 have their state in $\{init, full, dead\}$. These configurations represent markings of $\mathcal{N}$. We say that the stable configuration $\mathbf{c}$ simulates marking $\mathbf{m}$, noted $\mathbf{c} \sim \mathbf{m}$, if for every word $w \in \Sigma^\ell$, there exist exactly $\mathbf{m}(w)$ processes at level 1 which have exactly one child in state $(w[i], i)$ for each $0 < i \leq \ell$, and all their other children are in state *init*. An example of a stable configuration simulating $\mathbf{m} = [\![ab, ab, ba]\!]$ can be seen in Figure 7. We give a small example of the execution of a move of a token from ab to ba in our system in Figure 8.

We can show:

▶ **Lemma 4.3.** *A marking* $\mathbf{m}$ *is reachable in* $\mathcal{N}$ *iff there exists a stable configuration* $\mathbf{c}$ *reachable in* $\mathcal{A}(\mathcal{N}, w_{final})$ *such that* $\mathbf{c} \sim \mathbf{m}$.

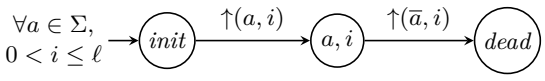

Figure 6 Automaton for processes at level 2.

From Lemma 4.3 we know that $\mathcal{A}(\mathcal{N}, w_{final})$ can reach a stable configuration with some process at level 1 storing w_{final} iff there exists a reachable marking with a token in w_{final} in $\mathcal{N}$. As root needs to consume w_{final} as its final operation, we know that it can reach $final_r$ iff w_{final} is coverable in $\mathcal{N}$.

Moreover, the system $\mathcal{A}(\mathcal{N}, w_{final})$ is phase-bounded. Indeed, root always does only one phase and so do processes at level 2. By construction, processes at level 1 need $1 + 2 \cdot \ell$ phases before reaching *full*, and then $1 + 2 \cdot \ell$ other phases to go to state *dead*. So they need at most $2 + 4 \cdot \ell$ phases.

We can construct $\mathcal{A}(\mathcal{N}, w_{final})$ in polynomial time, as for $\mathcal{A}_{\mathsf{root}}$ we only need to copy and unwrap the transitions of each transducers into letter transitions, and then add the production of w_{init} and consumption of w_{final}. For $\mathcal{A}$ the number of states and transitions is polynomial in the size of Σ and ℓ. We can conclude from [8]:

▶ **Theorem 4.4.** 2-*Coverability*(k) *for PCA is* 2Expspace*-hard.*

4.2 Upper bound for phase-bounded PCA

In this section we show that (∞, k)-Coverability is in Expspace and that $(d = \infty)$-Coverability(k) is in 2Expspace. To do so, we will prove that on phase-bounded runs, the language of receives of each process is regular. We will show how to compute automata for these languages on subtrees of depth one first, and then apply the technique recursively.

We define an *Interface PCA* (*IPCA* for short) as a pair $\mathcal{A} = (\mathcal{A}_{\mathsf{root}}, \mathcal{A}_\ell)$, with the LTS $\mathcal{A}_{\mathsf{root}}$ for root, and $\mathcal{A}_\ell$ for non-root processes. The only difference between IPCA and PCA is that the root process can do actions in Up without a parent to synchronize with. So a run ρ of a IPCA $\mathcal{A}$ is labelled by $\mathsf{trace}(\rho) \in (\mathtt{Up} \cup Act_{\mathcal{T}})^*$ where $\mathsf{trace}(\rho)_{|p} \in (Act_{\mathcal{T}})^*$ for every $p \neq \mathsf{root}$. Recall that $Act_{\mathcal{T}} = \{p \bowtie q(m) \mid (p,q) \in C, m \in M\} \cup \{p : \tau \mid p \in P\}$ is the set of synchronous communication actions.

The language of an IPCA $\mathcal{A}$ is the set of words $w \in \mathtt{Up}^*$ such that there exists some tree $\mathcal{T}$ and a run ρ of $\mathcal{A}$ over $\mathcal{T}$ with $w = \mathsf{trace}(\rho)_{|\mathtt{Up}}$. We denote by $L_k^{d \leq 1}(\mathcal{A})$ the language of IPCA $\mathcal{A}$ over trees of depth 1 and k-phase bounded runs.

We already know that $(1, \infty)$-Coverability is equivalent to VASS coverability. We will now extend this observation by showing that languages of IPCA are also VASS languages. To do so, we will enrich our VASS with action transitions.

Let $m > 0$ be a strictly positive integer and A be a finite alphabet. An *m-VASS with actions* from A is a tuple $\mathcal{V} = (Q, q^{init}, \Delta_V, A_\tau, \Delta_A)$ where Q is a finite set of states, q^{init} is the initial state, $\Delta_V \subseteq Q \times \mathbb{Z}^m \times Q$ is a counter transition relation, $A_\tau = A \cup \{\tau\}$ is the alphabet and $\Delta_A \subseteq Q \times A_\tau \times Q$ is an action transition relation.

The size of an m-VASS with actions $\mathcal{V}$ is $|\mathcal{V}| = |Q| + \sum_{(q,\mathbf{v},q') \in \Delta_V} \|\mathbf{v}\|_1 + |\Delta_A|$. Configurations are defined as for VASS. The step relation of a m-VASS with actions is $(q, \mathbf{x}) \xrightarrow{a} (q', \mathbf{x}')$ if either $(q, a, q') \in \Delta_V$, $a = \mathbf{v} \in \mathbb{Z}^m$, $\mathbf{x}' = \mathbf{x} + \mathbf{v} \geq \mathbf{0}$, or $(q, a, q') \in \Delta_A$, $a \in A_\tau$, $\mathbf{x} = \mathbf{x}'$. In other words, actions (i.e., transitions from Δ_A) do not affect the counters.

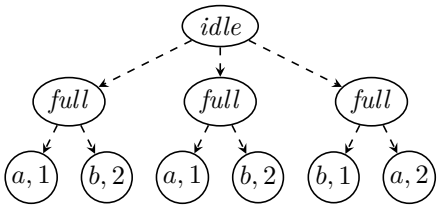

Figure 7 Configuration simulating $\mathbf{m} = [\![ab, ab, ba]\!]$. All unspecified processes have their state in $\{init, dead\}$.

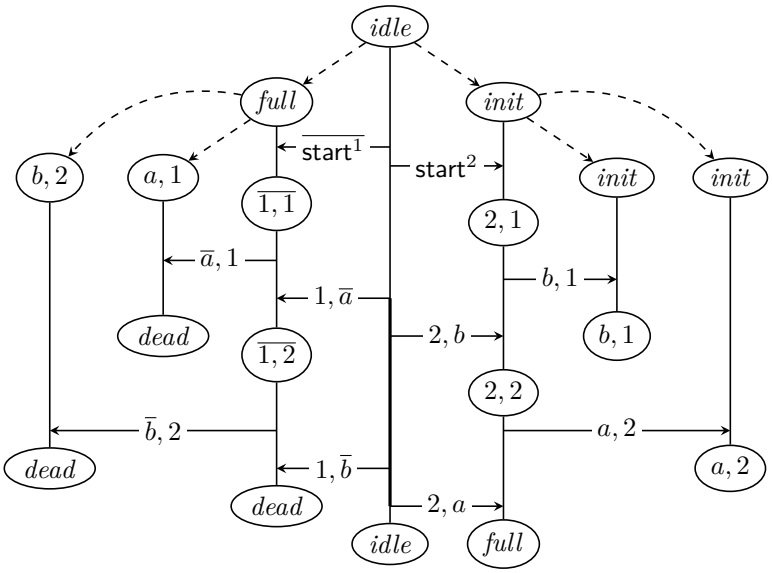

Figure 8 Execution of a move transition from place ab to place ba. The simulation of $\mathcal{M}_{move}$ by root is depicted in bold. Dotted edges are tree edges, full edges represent transitions.

A run of $\mathcal{V}$ is a sequence $\sigma = (q_0, \mathbf{x}_0) \xrightarrow{a_1} (q_1, \mathbf{x}_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (q_n, \mathbf{x}_n)$ where $(q_0, \mathbf{x}_0)$ is initial. The trace of a run σ is the sequence of actions labelling it, and is denoted as $\mathsf{trace}(\sigma) \in {A_\tau}^*$. The language of a VASS with actions $\mathcal{V}$ is defined as $L(\mathcal{V}) = \{w \in A^* \mid w = \mathsf{trace}(\sigma)_{|A} \text{ for some run } \sigma\}$.

We say that a run σ of $\mathcal{V}$ is k-phase bounded if there exist $\sigma_1, \dots, \sigma_k$ such that $\sigma = \sigma_1 \cdot \sigma_2 \dots \sigma_k$, and every σ_i has all its transitions either in Δ_V or in Δ_A. We define $L_k(\mathcal{V})$ as the language of $\mathcal{V}$ restricted to k-phase bounded runs.

We will now show how to construct a VASS with actions $\mathcal{V}$ from an IPCA $\mathcal{A}$ over trees of depth 1, which will simulate the interface language of $\mathcal{A}$.

▶ **Lemma 4.5.** *Let $\mathcal{A} = (\mathcal{A}_{\mathsf{root}}, \mathcal{A}_\ell)$ be an IPCA over trees of depth one. We can construct an m-VASS with actions $\mathcal{V} = (Q, q^{init}, \Delta_V, A_\tau, \Delta_A)$, with $Q = S_{\mathsf{root}}$, $m = |S_\ell|$, and $A_\tau = \mathtt{Up} \cup \{\tau\}$ such that, for any $k \in \mathbb{N}_\infty$, $L_k(\mathcal{V}) = L_k^{d \leq 1}(\mathcal{A})$.*

Next we show how to construct a finite automaton for the language of phase-bounded runs of a VASS with actions. Let m and k be two integers. Let $\mathcal{V}$ be an m-VASS with actions, and σ be a k-phase bounded run of $\mathcal{V}$. There exist $(q_1, \mathbf{x}_1), \dots, (q_k, \mathbf{x}_k)$ such that $\sigma = (q_0, \mathbf{x}_0), \sigma_1, (q_1, \mathbf{x}_1), \sigma_2, (q_2, \mathbf{x}_2) \dots (q_k, \mathbf{x}_k)$, where every σ_i is either only on Δ_V, or only on Δ_A. The important observation is that every subrun σ_i over Δ_A can be replaced by any other subrun σ_i' over Δ_A starting and ending in the same states as σ_i, as these phases have no effect on the counters.

▶ **Lemma 4.6.** *Let m and k be two integers. Let $\mathcal{V} = (Q, q^{init}, \Delta_V, A, \Delta_A)$ be an m-VASS with actions. There exists an automaton $\mathcal{B}$ with $O(k \times (2 \times |Q|)^{k+1})$ states such that $L(\mathcal{B}) = L_k(\mathcal{V})$, that can be constructed in $2^{O(m)} \cdot \log(k \times |\mathcal{V}|)$-space.*

Proof. Let $t = (q_0, d_1, q_1, d_2 \dots, d_k, q_k)$ be a sequence of states and transition types, with $q_i \in Q$ and $d_i \in \{V, A\}$ for each $0 < i \leq k$. We say that t is *valid* if there exists a run $\sigma = (q_0, \mathbf{x}_0), \sigma_1, (q_1, \mathbf{x}_1), \sigma_2, (q_2, \mathbf{x}_2), \dots, \sigma_k, (q_k, \mathbf{x}_k)$ where for each $0 < i \leq k$, subrun σ_i is only over Δ_{d_i}. In that case we say that σ *complies* with t.

For every valid sequence t, we will construct a finite automaton $\mathcal{B}_t$ which language is the set of words $w \in A^*$ such that there exists a run $\sigma = (q_0, \mathbf{x}_0), \sigma_1, (q_1, \mathbf{x}_1), \sigma_2, (q_2, \mathbf{x}_2) \dots (q_k, \mathbf{x}_k)$ complying with t, and with $w = \mathsf{trace}(\sigma)_{|A}$.

Let $t = (q_0, d_1, q_1, d_2 \dots, d_k, q_k)$ be a valid sequence. We construct $\mathcal{B}_t = (Q_{\mathcal{B}_t}, q^{init}_{\mathcal{B}_t}, \Delta_{\mathcal{B}_t}, q^{final}_{\mathcal{B}_t})$, with $Q_{\mathcal{B}_t}$ the set of states, $q^{init}_{\mathcal{B}_t} \in Q_{\mathcal{B}_t}$ the initial state, $\Delta_{\mathcal{B}_t} \subseteq Q_{\mathcal{B}_t} \times A_\tau \times Q_{\mathcal{B}_t}$ the transition relation, and $q^{final}_{\mathcal{B}_t} \in Q_{\mathcal{B}_t}$ the final state, as follows:

- $Q_{\mathcal{B}_t} = Q \times k$, with $(q, i) \in Q_{\mathcal{B}_t}$ meaning that the automaton is in state q and is simulating the i-th phase of a run.
- $q^{init}_{\mathcal{B}_t} = (q_0, 1)$.
- For each $0 < i \leq k$, if $d_i = V$, then $(q_{i-1}, i) \xrightarrow{\tau} (q_i, i) \in \Delta_{\mathcal{B}_t}$, otherwise if $d_i = A$, then for all $q \xrightarrow{a} q' \in \Delta_A$, we have $(q, i) \xrightarrow{a} (q', i) \in \Delta_{\mathcal{B}_t}$. Moreover, we have $(q_{i-1}, i-1) \xrightarrow{\tau} (q_{i-1}, i) \in \Delta_{\mathcal{B}_t}$.
- $q^{final}_{\mathcal{B}_t} = (q_k, k)$.

A run of $\mathcal{B}_t$ is a sequence $\rho = q_0 \xrightarrow{a_1} q_1 \dots q_{n-1} \xrightarrow{a_n} q_n$ where $q_1 = q^{init}_{\mathcal{B}_t}$, $q_n = q^{final}_{\mathcal{B}_t}$, and for each $0 < i \leq n$, $q_{i-1} \xrightarrow{a_i} q_i \in \Delta_{\mathcal{B}_t}$. We denote by $\mathsf{trace}(\rho) \in A_\tau$ the trace of a run ρ. The language of $\mathcal{B}_t$ is the set $L(\mathcal{B}_t) = \{w \mid w = \mathsf{trace}(\rho)_{|A} \text{ for some run } \rho \text{ of } \mathcal{B}_t\}$. Now we show that $L(\mathcal{B}_t)$ is exactly the set of all projections on A of runs of $\mathcal{V}$ complying with t. Let $\sigma = (q_0, \mathbf{x}_0), \sigma_1, (q_1, \mathbf{x}_1), \sigma_2, (q_2, \mathbf{x}_2) \dots (q_k, \mathbf{x}_k)$ be a run complying with t, and let $w = \mathsf{trace}(\sigma)_{|A}$. There exist $w_1, \dots, w_k$ such that $w_i = \mathsf{trace}(\sigma_i)_{|A}$. We have that if $d_i = V$ then $w_i = \varepsilon$. Then when $d_i = A$, one can read w_i between (q_{i-1}, i) and (q_i, i) in $\mathcal{B}_t$ by construction, and when $d_i = V$, one can go from state (q_{i-1}, i) to state (q_i, i) in $\mathcal{B}_t$ reading τ (as when $d_i = V$ then $w_i = \varepsilon$). Moreover, one can change phases through transitions $(q_{i-1}, i-1) \xrightarrow{\tau} (q_{i-1}, i)$. So w is accepted by $\mathcal{B}_t$.

Now let $w \in L(\mathcal{B}_t)$. There exists some run ρ of $\mathcal{B}_t$ such that $\mathsf{trace}(\rho)_{|A} = w$. By construction, there exist $\rho_1, \dots, \rho_k$ such that $\rho = \rho_1 \cdot \rho_2 \dots \rho_k$, where for each $0 < i \leq k$ if $d_i = V$, then $\rho_i = (q_{i-1}, i-1) \xrightarrow{\tau} (q_{i-1}, i) \xrightarrow{\tau} (q_i, i)$, and if $d_i = A$ then $\rho_i = (q_{i-1}, i-1) \xrightarrow{\tau} (q_{i-1}, i) \xrightarrow{u_i} (q_i, i)$ for some sequence of actions $u_i \in {A_\tau}^*$. By construction, u_i can be read in $\mathcal{V}$ between any two configurations $(q_{i-1}, \mathbf{x})$ and $(q_i, \mathbf{x})$. And as t is valid, one can construct a k-bounded run $\sigma = (q_0, \mathbf{x}_0), \sigma_1, (q_1, \mathbf{x}_1), \sigma_2, (q_2, \mathbf{x}_2) \dots (q_k, \mathbf{x}_k)$ where for each $0 < i \leq k$, if $d_i = V$, then σ_i makes only counter updates, and if $d_i = A$ then $\mathsf{trace}(\sigma_i) = u_i$. So σ is complying with t and $\mathsf{trace}(\sigma)_{|A} = w$.

Finally we construct $\mathcal{B}$ by taking the disjoint union of every $\mathcal{B}_t$, over all valid sequences t. This automaton has at most $(k \times |Q|) \times (2 \times |Q|)^k$ states.

Finally we can check if a sequence $t = (q_1, d_1, q_2, \dots, d_{k-2}, q_k)$ is valid through a reduction to a coverability question on an m-VASS with $|Q| \times k$ states. The details are in the full version [17].

As coverability in an n-VASS $\mathcal{U}$ can be checked in $2^{O(n)} \times \log(|\mathcal{U}|)$-space (see [23]), we can check if $t = (q_1, d_1, q_2, \dots, q_k)$ is valid in $2^{O(m)} \times \log(k \times |\mathcal{V}|)$-space. ◀

Using the above reasoning we can check if a state of a PCA is $(d = \infty, k)$-coverable, so for arbitrary depth. First we demonstrate how to use the technique for depth 2. Let $k > 0$ be a phase bound, $\mathcal{A} = (S, Act, \Delta, s^{init})$ a PCA, and $s \in S$ a state. By Lemma 4.5 we can simulate the language of any process at depth 1 on k-phase-bounded runs using an m-VASS with actions $\mathcal{V} = (S, s^{init}, \Delta_V, A_\tau, \Delta_A)$, where $m = |S|$, $A_\tau = \mathtt{Up} \cup \{\tau\}$, that can be constructed in polynomial time. Moreover, using Lemma 4.6, we can replace $\mathcal{V}$ by a finite automaton $\mathcal{B}$ of size $O(k \times (2|S|)^{k+1})$ that can be built in $2^{O(|S|)} \times \log(O(k \times |S| \times |\Delta|))$-space. So we can reduce $(2, k)$-coverability to $(1, k)$-coverability. Using the same technique as for the upper bound of Lemma 3.1, we can check if s is coverable in $2^{O(k \times (2|S|)^{k+1})} \times \log(O(|S|))$-space.

If we want to apply the same technique for depth 3, we first replace all processes of depth 2 by automata of size $O(k \times (2|S|)^{k+1})$, and then we transform all processes of depth 1 into m-VASS with actions where $m = O(k \times (2|S|)^{k+1})$, and which statespace is still S. Using Lemma 4.6 again we can construct a finite automaton of size $O(k \times (2|S|)^{k+1})$ which can be built in $2^{O(k \times (2|S|)^{k+1})} \times \log(O(k \times |S| \times |\Delta|))$-space. Then one can check coverability of s in $2^{O(k \times (2|S|)^{k+1})} \times \log(O(|S|))$-space again.

One can see that repeating the process of compression does not require more space after depth 3, as the automaton produced will always be of size $O(k \times (2|S|)^{k+1})$. Moreover, as the size of $\mathcal{B}$ is bounded, we know that if s is coverable with k-bounded runs, it is coverable with k-bounded runs over a tree of depth at most $O(k \times (2|S|)^{k+1})$.

We can conclude:

▶ **Theorem 4.7.** *The problem $(d = \infty, k)$-Coverability for PCA (with $d \in \mathbb{N}$) is* Expspace*-complete. If k is encoded in unary, then the problem $(d = \infty)$-Coverability(k) is* 2Expspace*-complete.*

5 Conclusion

We investigated two kinds of bounds under which we determine the complexity of safety checking of systems with rendez-vous synchronization over tree topologies. When bounding the depth we obtained that the complexity is related to the fast growing hierarchy. The second bound on the phases led to complexity Expspace-complete if the number of phases is fixed, resp. 2Expspace-complete if the number of phases is part of the input.

An interesting question left open is whether given d, k and a PCA $\mathcal{A}$, if all executions of $\mathcal{A}$ over d-bounded trees are k phase-bounded. A further natural question is to close the complexity gap when the depth is fixed.

References

1 Parosh Aziz Abdulla, Noomene Ben Henda, Giorgio Delzanno, Frédéric Haziza, and Ahmed Rezine. Parameterized tree systems. In Kenji Suzuki, Teruo Higashino, Keiichi Yasumoto, and Khaled El-Fakih, editors, *Formal Techniques for Networked and Distributed Systems - FORTE 2008, 28th IFIP WG 6.1 International Conference, Tokyo, Japan, June 10-13, 2008, Proceedings*, Lecture Notes in Computer Science, pages 69–83. Springer, 2008. doi:10.1007/978-3-540-68855-6_5.

2 Parosh Aziz Abdulla, Bengt Jonsson, Pritha Mahata, and Julien d'Orso. Regular tree model checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, Lecture Notes in Computer Science, pages 555–568. Springer, 2002. doi:10.1007/3-540-45657-0_47.

3 Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. *Distributed Comput.*, 31(3):187–222, 2018. doi:10.1007/S00446-017-0302-6.

4 Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986. doi:10.1016/0020-0190(86)90071-2.

5 A. R. Balasubramanian. Complexity of coverability in depth-bounded processes. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, Warsaw, Poland, September 12-16, 2022*, LIPIcs, pages 17:1–17:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CONCUR.2022.17.

6 A. R. Balasubramanian, Javier Esparza, and Mikhail A. Raskin. Finding cut-offs in leaderless rendez-vous protocols is easy. *Log. Methods Comput. Sci.*, 19(4), 2023. doi:10.46298/LMCS-19(4:2)2023.

7 A. R. Balasubramanian and Franzisco Schmidt. The complexity of nested reset counter systems. In Claudia Faggian and Joost-Pieter Katoen, editors, *41st Annual Symposium on Logic in Computer Science, LICS 2026, Lisbon, Portugal, July 20-23, 2026*, volume 380 of *LIPIcs*, pages 11:1–11:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPICS.LICS.2026.11.

8 Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. The complexity of bounded context switching with dynamic thread creation. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, Saarbrücken, Germany (Virtual Conference), July 8-11, 2020*, volume 168 of *LIPIcs*, pages 111:1–111:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.111.

9 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. doi:10.2200/S00658ED1V01Y201508DCT013.

10 Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomás Vojnar. Abstract regular (tree) model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):167–191, 2012. doi:10.1007/S10009-011-0205-Y.

11 Marius Bozga, Javier Esparza, Radu Iosif, Joseph Sifakis, and Christoph Welzel. Structural invariants for the verification of systems with parameterized architectures. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, Lecture Notes in Computer Science, pages 228–246. Springer, 2020. doi:10.1007/978-3-030-45190-5_13.

12 Marius Bozga and Radu Iosif. Specification and safety verification of parametric hierarchical distributed systems. In Gwen Salaün and Anton Wijs, editors, *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*, Lecture Notes in Computer Science, pages 95–114. Springer, 2021. doi:10.1007/978-3-030-90636-8_6.

13 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

14 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1229–1240. IEEE, 2021. doi:10.1109/FOCS52979.2021.00120.

15 Normann Decker and Daniel Thoma. On freeze LTL with ordered attributes, 2016. arXiv:1504.06355.

16 Normann Decker and Daniel Thoma. On freeze LTL with ordered attributes. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 269–284. Springer, 2016. doi:10.1007/978-3-662-49630-5_16.

17 Romain Delpy, Anca Muscholl, and Grégoire Sutre. On parameterized verification over tree topologies, 2026. arXiv:2606.27172.

18 Allen E. Emerson and Kedar S. Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(04):527–549, 2003. doi:10.1142/S0129054103001881.

19 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science, STACS 2014, Lyon, France, March 5-8, 2014*, LIPIcs, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.1.

20 Steven M. German and Prasad A. Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.

21 Lucie Guillou, Arnaud Sangnier, and Nathalie Sznajder. Phase-bounded broadcast networks over topologies of communication. In Rupak Majumdar and Alexandra Silva, editors, *35th International Conference on Concurrency Theory, CONCUR 2024, Calgary, Canada, September 9-13, 2024*, LIPIcs, pages 26:1–26:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CONCUR.2024.26.

22 Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. *Theor. Comput. Sci.*, 256(1-2):93–112, 2001. doi:10.1016/S0304-3975(00)00103-1.

23 Marvin Künnemann, Filip Mazowiecki, Lia Schütze, Henry Sinclair-Banks, and Karol Wegrzycki. Coverability in VASS revisited: Improving Rackoff's bounds to obtain conditional optimality. *J. ACM*, 72(5):33:1–33:27, 2025. doi:10.1145/3762178.

24 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1241–1252. IEEE, 2021. doi:10.1109/FOCS52979.2021.00121.

25 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785796.

26 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976. URL: http://cpsc.yale.edu/sites/default/files/files/tr63.pdf.

27 Irina A. Lomazova and Philippe Schnoebelen. Some decidability results for nested petri nets. In Dines Bjørner, Manfred Broy, and Alexandre V. Zamulin, editors, *Perspectives of System Informatics, Third International Andrei Ershov Memorial Conference, PSI'99, Akademgorodok, Novosibirsk, Russia, July 6-9, 1999, Proceedings*, volume 1755 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 1999. doi:10.1007/3-540-46562-6_18.

28 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

29 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Transactions on Computation Theory (TOCT)*, 8(1):1–36, 2016. doi:10.1145/2858784.