REACHABILITY ANALYSIS OF COMMUNICATING PUSHDOWN SYSTEMS

Alexander Heussner, Jérôme Leroux, Anca Muscholl, Grégoire Sutre

Logical Methods in Computer Science
Vol. 8(3:23)2012, pp. 1–20
www.lmcs-online.org

Submitted Dec. 15, 2010
Published Sep. 26, 2012

REACHABILITY ANALYSIS OF COMMUNICATING PUSHDOWN SYSTEMS *

ALEXANDER HEUSSNER, JÉRÔME LEROUX, ANCA MUSCHOLL, AND GRÉGOIRE SUTRE

LaBRI, Université de Bordeaux, CNRS – France
e-mail address: {alexander.heussner, jerome.leroux, anca, gregoire.sutre}@labri.fr

Abstract. The reachability analysis of recursive programs that communicate asynchronously over reliable Fifo channels calls for restrictions to ensure decidability. Our first result characterizes communication topologies with a decidable reachability problem restricted to eager runs (i.e., runs where messages are either received immediately after being sent, or never received). The problem is ExpTime-complete in the decidable case. The second result is a doubly exponential time algorithm for bounded context analysis in this setting, together with a matching lower bound. Both results extend and improve previous work from [21].

Introduction

Checking safety properties for distributed programs like client/server environments, peer-to-peer applications, or asynchronous programs on multi-core processors is a standard task in verification. However, it is well established that the automatic analysis of distributed programs is a quite challenging objective.

A basic feature of the programs used in the applications mentioned above is that they need to exchange information asynchronously, over point-to-point channels that are unbounded and reliable. Such information is used for instance to perform function calls on remote processes. This amounts to considering a model that combines recursion with asynchronous communication. Such a combined model is similar in spirit to, e.g., process rewrite systems [25], that mix recursion and Petri nets. We denote the combination of recursion and asynchronous communication as *Recursive Communicating ProcesseS* (RCPS for short) here. The model has been recently studied by La Torre, Madhusudan, and Parlato [21], who were mainly interested in applying bounded context analysis to this setting.

Since RCPS subsume the well-studied class of communicating finite-state machines [8], reachability is already undecidable without recursion. Moreover, it is well-known that reachability for pushdown systems that synchronize by rendezvous is undecidable as well [28]. Therefore, our main motivation was to separate these two sources of undecidability. We

1998 ACM Subject Classification: D.2.4, F.2.
Key words and phrases: Reachability analysis, communicating processes, pushdown systems.
* An extended abstract of this paper appeared in FoSSaCS'10.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-8(3:23)2012

consider here behavioral restrictions for which reachability for communicating *finite-state* machines is decidable, and then look under which conditions recursion can be added to the model.

The reachability question for communicating finite-state machines can be tackled in three different ways, either by restricting the communication topology, or by assuming that channels are lossy, or by considering only executions on channels of fixed size. In general, the last two approaches provide approximated solutions to the reachability problem. On the positive side, the last idea yields exact solutions in some special cases, either for certain restricted topologies (e.g., acyclic ones) or under certain behavioral restrictions on the communication (e.g., mutex communication, see below).

As already mentioned, our starting point is the work of La Torre et al. [21]. They introduced a syntactic restriction on the combined use of channels and pushdowns, that prevents the synchronization of pushdowns leading to an undecidable reachability question. An RCPS is called *well-queueing* in [21] if pushdown processes can only read messages when their stack is empty (they can send messages without any restriction). Well-queueing expresses an event-based programming paradigm: tasks are executed by threads without interrupt, i.e., a thread accepts the next task only after it finished the current one. One of the results of [21] is that well-queueing RCPS have a decidable reachability problem if and only if the topology is a directed forest; in the decidable case, they provide a doubly exponential algorithm by a reduction to bounded-phase multi-stack pushdown systems [20].

We extend the results of [21] in several directions. First, we add a dual notion to well-queueing: a pushdown process can send messages only with empty stack (but can read messages without restriction). This dual notion arises naturally if one wants to model interrupts: a server might need to accept tasks from high priority clients independently of the status of the running task. We use these two restrictions by fixing the type of each communication channel, to be either well-queueing or the dual notion. A communication topology, together with channel types, is called a *typed topology*.

We give in Section 2 a precise characterization of those typed topologies for which the RCPS model has a decidable reachability problem over so-called *eager* runs. A run is eager if the sending of a message is immediately followed by its reception (if any). This notion is closely related to bounded communication [23]. Communicating finite-state machines with existential channel bounds, i.e., where each run *can* be reordered into a run over bounded channels, are a well-studied model enjoying good expressiveness and decidability properties [15][1]. Here, we simply use eager runs in order to rule out undecidability due to unbounded channels, since reachability for finite-state communicating machines over eager runs is decidable. We show that reachability of RCPS over eager runs is ExpTime-complete in the decidable case. Our result generalizes and improves the doubly exponential time decision procedure of [21], which holds for topologies without undirected cycles (called *polyforests*).

The restriction to eager runs appears to be strong at a first glance. However, we show in Section 3 that it arises rather naturally, by imposing a behavioral restriction on the communication: the *mutex* restriction requires that in every reachable configuration there is no more than one non-empty channel per cycle of the network. In particular, RCPS over polyforest architectures are mutex. Mutex can also be seen as a generalization of the half-duplex restriction studied in [9].

[1]Machines with the property that each run can be reordered into an eager one, are a special instance of existentially 1-bounded machines. Eagerness is related to a *global* channel bound [23].

La Torre et al. propose in [21] a second approach to solve the reachability problem for RCPS, inspired by successful work on reachability with bounded contexts in the verification of concurrent Boolean programs [27]. They show that bounded-context reachability for well-queueing RCPS is decidable in time doubly exponential in the number of contexts. Again, this result is obtained by a reduction to bounded-phase multi-stack pushdown systems [20]. Our result in Section 4 extends the bounded-context result of [20] to RCPS that allow for the two dual notions of well-queueing. Moreover, our algorithm is direct and simpler than the one involving bounded-phase multi-stack pushdown systems. We also provide a matching lower bound for the complexity.

Related work. In the context of multi-thread programming, other notions of synchronization between pushdowns arise naturally. Earlier publications considered synchronization via shared memory, such as local/global memory in [6, 7] or bags in [29, 17]. The paper [6] showed that bounded-context reachability can be solved in exponential time, whereas [29] provided an exponential space lower bound for reachability with atomic methods (without context bounds). Also, synchronization in the form of state observation was considered in [4]. The latter model was shown to be decidable only for acyclic architectures, and is strongly related to lossy systems [1, 14]. For the shared memory model, [18] shows how to reduce concurrent pushdowns to a single pushdown, assuming a priority preemptive scheduling policy. Lately, [30, 2] proposed a general strategy to reduce bounded-phase reachability questions on different multi-stack pushdown automata models to a single stack. This is close in spirit to our proof technique in Section 2, although we do not rely on a phase-bounded model for our first result.

1. Recursive Communicating Processes

Given a set P and a P-indexed family of sets $(S^p)_{p\in P}$, we write elements of the Cartesian product $\prod_{p\in P} S^p$ in bold face. For any $\mathbf{s}$ in $\prod_{p\in P} S^p$ and any $p \in P$, we let $s^p \in S^p$ denote the p-component of $\mathbf{s}$. Moreover, we identify $\mathbf{s}$ with the indexed family of elements $(s^p)_{p\in P}$.

An *alphabet* is any finite set of *letters*. Given an alphabet Σ, we write Σ^* for the set of all *finite words* (*words* for short) over Σ, and we let ε denote the *empty word*.

A *labeled transition system* (LTS for short) $\mathcal{A} = \langle S, s_{\mathcal{I}}, A, \to\rangle$ is given by a set of *states* S, an initial state $s_{\mathcal{I}}$, an *action* alphabet A, and a (labeled) *transition relation* $\to$, which is a subset of $S \times A \times S$. For simplicity, we usually write $s \xrightarrow{a} s'$ in place of $(s, a, s') \in \to$. The *size* of $\mathcal{A}$ is defined by $|\mathcal{A}| = |S|^2 \cdot |A|$ when S is finite.

Throughout the paper we use standard complexity classes such as polynomial space (PSpace), deterministic exponential time (ExpTime), and deterministic doubly-exponential time (2-ExpTime). For detailed definitions the reader is referred to, e.g., [26].

1.1. Communication Topologies.

In this paper, we consider processes from a finite set P, that communicate over point-to-point, error-free Fifo channels from a set C. They exchange messages over a given topology, which is simply a directed graph whose vertices are processes and whose edges represent channels:

Definition 1.1. A *topology* $\mathcal{T}$ is a tuple $\langle P, C, src, dst\rangle$ where P is a finite set of *processes*, and C is a finite set of point-to-point *channels* equipped with two functions $src, dst : C \to P$ that map every channel $c \in C$ to a *source* $src(c) \in P$ and a *destination* $dst(c) \in P$, such that $src(c) \neq dst(c)$.

The *size* of $\mathcal{T}$ is defined by $|\mathcal{T}| = |P| + |C|$. For each channel $c \in C$, we write $\xrightarrow{c}$ for the binary relation on the set of processes P defined by $p \xrightarrow{c} q$ if $p = src(c)$ and $q = dst(c)$. We also use the undirected binary relation $\xleftrightarrow{c}$, defined by $p \xleftrightarrow{c} q$ if $p \xrightarrow{c} q$ or $q \xrightarrow{c} p$.

An *undirected path* in $\mathcal{T}$ is an alternating sequence $(p_0, c_1, p_1, \ldots, c_n, p_n)$, of processes $p_i \in P$ and channels $c_i \in C$, such that $p_{i-1} \xleftrightarrow{c_i} p_i$ for all i. Moreover, the undirected path is called *simple* if the processes $p_0, \ldots, p_n$ are distinct. A *simple undirected cycle* in $\mathcal{T}$ is an undirected path $(p_0, c_1, p_1, \ldots, c_n, p_n)$ with $p_0 = p_n$ such that $p_1, \ldots, p_n$ are distinct, and $c_1, \ldots, c_n$ are distinct. The topology $\mathcal{T}$ is called *polyforest* if it contains no simple undirected cycle.

1.2. **Communicating Processes.** Consider a topology $\mathcal{T} = \langle P, C, src, dst\rangle$. Given a message alphabet M, we denote by $Com^p(\mathcal{T}, M)$ the set of *possible communication actions* of a process $p \in P$, defined by $Com^p(\mathcal{T}, M) = \{c!m \mid c \in C, src(c) = p, m \in M\} \cup \{c?m \mid c \in C, dst(c) = p, m \in M\}$. As usual, $c!m$ denotes sending message m into channel c, whereas $c?m$ denotes receiving message m from channel c. Note that $Com^p(\mathcal{T}, M)$ and $Com^q(\mathcal{T}, M)$ are disjoint when p and q are distinct processes.

Definition 1.2. A *system of communicating processes (*CPS *for short)* $\mathcal{Q} = \langle \mathcal{T}, M, (\mathcal{A}^p)_{p\in P}\rangle$ is given by a topology $\mathcal{T}$, a *message* alphabet M, and, for each process $p \in P$, an LTS $\mathcal{A}^p = \langle S^p, s^p_{\mathcal{I}}, A^p, \to_p\rangle$ such that:

- the action alphabets A^p, $p \in P$, are pairwise disjoint, and
- $A^p_{com} = A^p \cap (C \times \{!, ?\} \times M)$ is contained in $Com^p(\mathcal{T}, M)$ for each $p \in P$.

Actions in A^p_{com} are called *communication actions* of p, whereas $A^p_{loc} = A^p \backslash A^p_{com}$ is the set of *local actions*. States $s^p \in S^p$ are called *local states* of p. We write $\mathbf{S} = \prod_{p\in P} S^p$ for the set of *global states*. Note that the sets S^p, and hence $\mathbf{S}$, may be infinite. Indeed, the local transition systems $\mathcal{A}^p$ could be, for example, counter or pushdown systems. When $\mathbf{S}$ is finite, $\mathcal{Q}$ is called a *finite* CPS, and its *size* is defined by $|\mathcal{Q}| = |\mathcal{T}| + |M| + \sum_{p\in P} |\mathcal{A}^p|$.

As usual, the semantics of CPS is defined in terms of a global LTS $\langle X, x_{\mathcal{I}}, A, \to\rangle$, where $X = \mathbf{S} \times (M^*)^C$ is the set of *configurations*, $x_{\mathcal{I}} = (\mathbf{s}_{\mathcal{I}}, (\varepsilon)_{c\in C})$ is the initial configuration, $A = \bigcup_{p\in P} A^p$ is the set of actions, and $\to \subseteq X \times A \times X$ is the transition relation with $(\mathbf{s_1}, \mathbf{w_1}) \xrightarrow{a} (\mathbf{s_2}, \mathbf{w_2})$, where $a \in A^p$, if the following conditions are satisfied:

(i) $s^p_1 \xrightarrow{a}_p s^p_2$ and $s^q_1 = s^q_2$ for all $q \in P$ with $q \neq p$,
(ii) if $a \in A^p_{loc}$ then $\mathbf{w_1} = \mathbf{w_2}$,
(iii) if $a = c!m$ then $w^c_2 = w^c_1 \cdot m$ and $w^d_2 = w^d_1$ for all $d \in C$ with $d \neq c$,
(iv) if $a = c?m$ then $m \cdot w^c_2 = w^c_1$ and $w^d_2 = w^d_1$ for all $d \in C$ with $d \neq c$.

Given a process $p \in P$, we call *move* of p any transition $x_1 \xrightarrow{a} x_2$ with $a \in A^p$. A move is local if $a \in A^p_{loc}$.

A *run* in the LTS $\mathcal{Q}$ is a finite, alternating sequence $\rho = (x_0, a_1, x_1, \ldots, a_n, x_n)$ of configurations $x_i \in X$ and actions $a_i \in A$ satisfying $x_{i-1} \xrightarrow{a_i} x_i$ for all i. We say that ρ is a run from x_0 to x_n. The *length* of ρ is n, and is denoted by $|\rho|$. A run of length zero consists of a single configuration. The *trace* of a run $\rho = (x_0, a_1, x_1, \ldots, a_n, x_n)$ is the sequence of actions $trace(\rho) = a_1 \cdots a_n$. A pair of send/receive actions $a_i = c!m, a_j = c?m$ is called *matching* in ρ if $i < j$ and the number of receives on c within $a_i \cdots a_j$ equals the length of c in x_i. If ρ, ρ' are two runs such that the last configuration of ρ is equal to the first configuration of ρ', then we write $\rho \cdot \rho'$ for their concatenation.

We define the *order-equivalence* relation $\sim$ over runs as the finest congruence such that $(x_0, a, x_1, b, x_2) \sim (x_0, b, x_1', a, x_2)$ whenever a, b are actions on different processes. Informally, $\rho \sim \rho'$ if they can be transformed one into the other by iteratively commuting adjacent transitions that (i) are *not* located on the same process, and (ii) do *not* form a matching send/receive pair. The following is easy to check:

Fact 1.3. If ρ, ρ' are order-equivalent runs of a CPS, then they start in the same configuration and end in the same configuration.

A configuration $x \in X$ is *reachable* in a CPS $\mathcal{Q}$ if there exists a run of $\mathcal{Q}$ from the initial configuration $x_{\mathcal{I}}$ to x. We define the *reachability set* of $\mathcal{Q}$ as $\mathit{Reach}(\mathcal{Q}) = \{x \in X \mid x \text{ is reachable in } \mathcal{Q}\}$.

The *state reachability problem* for CPS asks, for a given CPS $\mathcal{Q}$ and a global state $\mathbf{s} \in \mathbf{S}$, whether $\mathit{Reach}(\mathcal{Q})$ intersects $\{\mathbf{s}\} \times (M^*)^C$. It is well-known that this problem is undecidable for finite CPS, even if we restrict the topology to two processes connected by two channels [8].

The undecidability of the state reachability problem for CPS is based on the fact that one cannot control how "fast" messages are received. A simple idea that rules out such behaviors is to consider only runs where the reception is immediate (if it exists):

Definition 1.4. A run $\rho = (x_0, a_1, x_1, \ldots, a_n, x_n)$ is *eager* if for all $1 \leq i \leq n$, if a_i is a receive action then $i > 1$ and a_{i-1} is its matching send action.

Thus, each send action along an eager run is either immediately followed by its matching receive, or it is never matched. In the latter case, all later sends into the channel are never received, and we say that the channel is in its "growing phase". In the former case, the adjacent matched send/receive actions act like a rendezvous synchronization between the two processes. Formally, given a channel $c \in C$, we call *rendezvous on* c any run (of length 2) $\rho = (x, c!m, x', c?m, x'')$ such that $x = (\mathbf{s}, \mathbf{w})$ with $w^c = \varepsilon$. The rendezvous *involves process* p if $p \in \{\mathit{src}(c), \mathit{dst}(c)\}$.

We introduce now the "eager" variants of the reachability notions presented previously. A configuration $x \in X$ is *eager-reachable* in a CPS $\mathcal{Q}$ if there exists an eager run from the initial configuration $x_{\mathcal{I}}$ to x. The *eager-reachability set* of $\mathcal{Q}$ is the set $\mathit{Reach}_{eag}(\mathcal{Q})$ of eager-reachable configurations. We say that a CPS $\mathcal{Q}$ is *eager* when $\mathit{Reach}_{eag}(\mathcal{Q}) = \mathit{Reach}(\mathcal{Q})$. In the next section, we show how eager CPS occur under some natural (and decidable) restrictions on cyclic communication. The simplest example arises over polyforest topologies.

The *state eager-reachability problem* for CPS asks, for a CPS $\mathcal{Q}$ and a global state $\mathbf{s} \in \mathbf{S}$, whether $\mathit{Reach}_{eag}(\mathcal{Q})$ intersects $\{\mathbf{s}\} \times (M^*)^C$. It is readily seen that this problem is decidable for *finite* CPS in PSpace.

Eager runs, modulo the fact that Definition 1.4 allows for runs which end in a sequence of (unmatched) send actions, are closely related to the notion of globally 1-bounded runs. Eager CPS subsume existentially globally 1-bounded communicating machines [23, 16]. However, as we will see in Section 3, it is undecidable whether a finite CPS is eager (in contrast, one can decide whether a finite, deadlock-free communicating machine is existentially globally 1-bounded [16]). On the positive side, Section 3 shows a decidable subclass of finite, eager CPS.

1.3. **Recursive Communicating Processes.** In the following we introduce RCPS together with a symmetric version of the "well-queueing" restriction used in [21]. Informally, RCPS (recursive CPS) are CPS where each local transition system is a pushdown system.

A well-queueing RCPS in [21] is one where a process can only receive when its stack is empty. Here, we dualize this concept by also allowing channels where the sender (but not the receiver) must have an empty stack. Well-queueing was motivated in [21] by the case where recursive processes need to finish their tasks before accepting new ones. Adding the dual notion of well-queueing is interesting when modeling interrupts: a recursive process may have to interrupt its current task to treat one with a higher priority, hence, it has to preserve its current state on the stack to return later.

Definition 1.5. A *typed* topology $\langle \mathcal{T}, \tau \rangle$ consists of a topology $\mathcal{T}$, together with a *type* $\tau \subseteq P \times C$, such that $(p, c) \in \tau$ implies $p \in \{src(c), dst(c)\}$.

Given a process $p \in P$ and a channel $c \in C$, we call p *restricted* on c if $(p, c) \in \tau$ (and *unrestricted* otherwise). Informally, a communicating pushdown process p as defined below will be restricted on c if p's stack must be empty when communicating over channel c.

Definition 1.6. A *pushdown system* $\mathcal{D} = \langle Z, z_{\mathcal{I}}, A, A_\varepsilon, \Gamma, \Delta \rangle$ is given by a finite set Z of *control states*, an *initial control state* $z_{\mathcal{I}} \in Z$, an alphabet A of *actions*, a subset $A_\varepsilon \subseteq A$, a stack alphabet Γ, and a transition relation $\Delta \subseteq Z \times A \times Z$, such that A contains the set $A_{stack} = \{push(\gamma), pop(\gamma) \mid \gamma \in \Gamma\}$ of *stack actions*.

We define the *size* of $\mathcal{D}$ by $|\mathcal{D}| = |Z|^2 \cdot |A|$. Actions in $A_\varepsilon \subseteq A \setminus A_{stack}$ are tests for empty stack. Naturally, for a pushdown system embedded in a CPS, the set of actions $A \setminus A_{stack}$ may contain communication (and local) actions. Depending on the typed topology, some communication actions may require an empty stack. This will be enforced by putting these communication actions in the set A_ε.

According to the informal description given above, we define now the semantics of pushdown processes. The semantics of $\mathcal{D} = \langle Z, z_{\mathcal{I}}, A, A_\varepsilon, \Gamma, \Delta \rangle$ is the LTS $\langle S, s_{\mathcal{I}}, A, \to \rangle$ with set of states $S = Z \times \Gamma^*$, initial state $s_{\mathcal{I}} = (z_{\mathcal{I}}, \varepsilon)$, and (labeled) transition relation $\to$ defined as expected: stack actions $push(\gamma)$ and $pop(\gamma)$ behave as usual ($pop(\gamma)$ blocks if the top of the stack is not γ), actions from $A \setminus A_{stack}$ do not change the stack, and actions in A_ε are possible only if the stack is empty.

Definition 1.7. A *recursive* CPS (RCPS for short) $\mathcal{R} = \langle \mathcal{T}, \tau, M, (\mathcal{D}^p)_{p \in P} \rangle$ is given by a typed topology $\langle \mathcal{T}, \tau \rangle$, a *message* alphabet M, and, for each process $p \in P$, a pushdown system $\mathcal{D}^p = (Z^p, z^p_{\mathcal{I}}, A^p, A^p_\varepsilon, \Gamma^p, \Delta^p)$ such that:

- the action alphabets A^p, for $p \in P$, are pairwise disjoint,
- $A^p_{com} = A^p \cap (C \times \{!, ?\} \times M)$ is contained in $Com^p(\mathcal{T}, M)$ for each $p \in P$, and
- $A^p_\varepsilon \supseteq \{c!m \in A^p_{com} \mid (p, c) \in \tau\} \cup \{c?m \in A^p_{com} \mid (p, c) \in \tau\}$ for each $p \in P$.

We associate with $\mathcal{R}$ the CPS $\langle \mathcal{T}, M, (\mathcal{A}^p)_{p \in P} \rangle$ where, for each $p \in P$, the LTS $\mathcal{A}^p$ is the semantics of the pushdown system $\mathcal{D}^p$. The *size* of $\mathcal{R}$ is defined by $|\mathcal{R}| = |\mathcal{T}| + |M| + \sum_{p \in P} |\mathcal{D}^p|$.

We write $\mathbf{Z} = \prod_{p \in P} Z^p$ for the set *global control states*. Abusing notation, a global state $\mathbf{s}$ of $\mathcal{R}$ will also be written $\mathbf{s} = (\mathbf{z}, \mathbf{u})$ where $s^p = (z^p, u^p)$ for each $p \in P$. The *state reachability problem* for RCPS asks, for a given RCPS $\mathcal{R}$ and a global control state $\mathbf{z} \in \mathbf{Z}$, whether $Reach(\mathcal{R})$ intersects $\{\mathbf{z}\} \times (\prod_{p \in P} (\Gamma^p)^*) \times (M^*)^C$. The *state eager-reachability problem* for RCPS is defined similarly, using $Reach_{eag}(\mathcal{R})$ instead of $Reach(\mathcal{R})$.

2. Topologies with Decidable State Reachability

Several factors lead to the undecidability of the state reachability problem for RCPS. In particular, the model is already undecidable without any pushdown. Our goal in this section is a decidability condition that concerns the interplay between pushdowns and communication, assuming that the communication is *not* the reason for undecidability. For this reason, we consider a restricted version of the state reachability problem, namely the one on eager runs.

Definition 2.1. A typed topology $\langle \mathcal{T}, \tau \rangle$ is called *confluent* if it contains a simple undirected path $(p_0, c_1, p_1, \ldots, c_n, p_n)$, with $n \geq 1$, such that p_0 is unrestricted on c_1 and p_n is unrestricted on c_n.

Notice that non-confluence implies that every channel is either restricted at the source, or at the destination, or at both ends (see Figure 1).

We say that a typed topology $\langle \mathcal{T}, \tau \rangle$ has a decidable RCPS state eager-reachability problem if the latter question is decidable for the class of RCPS with typed topology $\langle \mathcal{T}, \tau \rangle$. We show in this section that the notion of confluence gives a complete characterization of typed topologies with respect to the decidability of the above problem.

Theorem 2.2. *A typed topology has a decidable* RCPS *state eager-reachability problem if and only if it is non-confluent. Moreover, the problem is* ExpTime*-complete in the latter case.*

The rest of the section is devoted to the proof of this theorem. We first show the undecidability result in the confluent case.

Proposition 2.3. *Every confluent typed topology has an undecidable* RCPS *state (eager-) reachability problem.*

Proof. Consider a typed topology $\langle \mathcal{T}, \tau \rangle$ that is confluent. There is a simple undirected path $p_0 \overset{c_1}{\longleftrightarrow} p_1 \cdots p_{n-1} \overset{c_n}{\longleftrightarrow} p_n$ satisfying the conditions of Definition 2.1. Since p_0 is unrestricted on c_1 and p_n is unrestricted on c_n, both may use their stack while communicating over the channels c_1 and c_n, respectively. Recall that checking non-emptiness of the intersection of two context-free languages is undecidable. To prove the lemma, we reduce this problem to the state eager-reachability problem for RCPS with typed topology $\langle \mathcal{T}, \tau \rangle$.

Given two context-free languages K and L over the alphabet $\{\mathtt{0}, \mathtt{1}\}$, the process p_0 guesses a word in K while p_n guesses a word in L, and both processes check that they guessed the same word via synchronizations along the undirected path $p_0 \overset{c_1}{\longleftrightarrow} p_1 \cdots p_{n-1} \overset{c_n}{\longleftrightarrow} p_n$. Intermediate processes $p_1, \ldots, p_{n-1}$ do not use their stack, they simply convey the

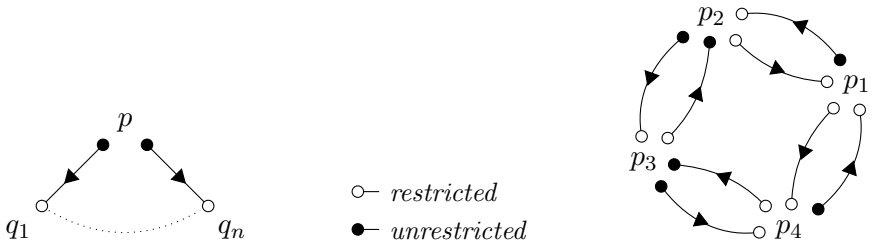
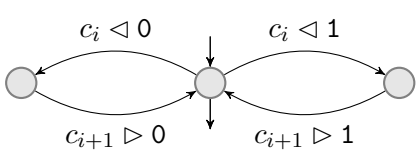


Figure 1: Examples of non-confluent typed topologies

information about the common input guessed by p_0 and p_n. The labeled transition system $\mathcal{A}^{p_i}$, $1 \leq i < n$, is depicted below.



$$\lhd \;=\; \begin{cases} ! & \text{if } p_i = src(c_i) \\ ? & \text{if } p_i = dst(c_i) \end{cases} \qquad \rhd \;=\; \begin{cases} ! & \text{if } p_i = src(c_{i+1}) \\ ? & \text{if } p_i = dst(c_{i+1}) \end{cases}$$

Similarly, the pushdown systems $\mathcal{D}^{p_0}$ and $\mathcal{D}^{p_n}$ are obtained from pushdown automata accepting K and L, respectively, by replacing tape-reading actions with communications ($c_1 \rhd 0/1$ for p_0 and $c_n \lhd 0/1$ for p_n).

Finally, we only need to make sure that the channels are empty at the end. As usual, this can be enforced by augmenting M with a new symbol \$, and by sending and receiving \$ on each channel c at the end of the simulation.

The construction guarantees that the intersection $K \cap L$ is non-empty if and only if there is an (eager) run in the RCPS from the initial configuration to a global control state where each process is accepting. □

We now focus on non-confluent typed topologies. Let us first prove the ExpTime lower bound of Theorem 2.2.

Proposition 2.4. *The state eager-reachability problem for* RCPS *with non-confluent typed topology is* ExpTime*-hard.*

Proof. It is well-known (and probably folklore) that the following problem is ExpTime-complete: given a context-free language K and n regular languages L_i, check the non-emptiness of $K \cap \bigcap_i L_i$. The hardness follows easily by a reduction from linearly bounded alternating Turing machines. Actually, a closely related problem is shown to be ExpTime-hard in [12], namely the reachability problem for pushdown systems with checkpoints.

Notice that the intersection $K \cap \bigcap_i L_i$ can be simulated on the non-confluent, typed topology $\langle \mathcal{T}, \tau \rangle$ where $P = \{p, q_1, \ldots, q_n\}$, $C = \{c_1, \ldots, c_n\}$, and, for each $1 \leq i \leq n$, $p \xrightarrow{c_i} q_i$ with p unrestricted on c_i and q_i restricted on c_i (see left part of Figure 1). That is, process p simulates a pushdown automaton accepting the context-free language K, whereas process q_i simulates a finite-state automaton accepting L_i. Communication guarantees that the simulations use the same input word. As in the previous proposition, one needs to enforce the emptiness of the channels by using an extra symbol. □

Before considering the upper bound we need to introduce some vocabulary. Consider a run $\rho = (x_0, a_1, x_1, \ldots, a_n, x_n)$ of an RCPS $\mathcal{R}$. Given a process $p \in P$, we say that ρ is *well-formed for* p if the projection of $a_1 \cdots a_n$ on A^p_{stack} is a Dyck word. This well-formedness condition merely stipulates that each push action of p in ρ is matched by a pop action, and vice versa. We call ρ *well-formed* if ρ is well-formed for each process $p \in P$. For instance, every run that starts and ends with empty stacks is well-formed. A stronger condition is that of well-bracketing, which requires that push and pop actions for distinct processes must be nested recursively. Formally, we say that $\rho = (x_0, a_1, x_1, \ldots, a_n, x_n)$ is *well-bracketed* if the following two conditions are satisfied:

(1) the projection of $a_1 \cdots a_n$ on the disjoint union $\bigcup_{p \in P} A^p_{stack}$ is a Dyck word, and

(2) for every process p and every $h < i < j < k$, if the pairs (a_h, a_k) and (a_i, a_j) are matching push/pop actions of p, then the sub-runs $(x_{h-1}, a_h, x_h, \ldots, a_i, x_i)$ and $(x_{j-1}, a_j, x_j, \ldots, a_k, x_k)$ are well-formed for all $q \neq p$.

Observe that if $\rho \cdot \rho'$ is defined, then $\rho \cdot \rho'$ is well-formed (resp. well-bracketed) if ρ and ρ' are both well-formed (resp. well-bracketed). Note also that well-formedness is preserved under order-equivalence: if ρ is well-formed and $\rho \sim \rho'$ then ρ' is also well-formed. However, well-bracketing is not preserved under order-equivalence.

The following proposition provides the main ingredient to show the ExpTime upper bound of Theorem 2.2.

Proposition 2.5. *Given an* RCPS $\mathcal{R}$ *with non-confluent typed topology, every eager, well-formed run in* $\mathcal{R}$ *is order-equivalent to an eager, well-bracketed run.*

Proof. By induction on the length of runs. The basis is trivial. Consider a run ρ, of non-zero length, that is both eager and well-formed. We assume that ρ starts with a push action (otherwise, the existence of an order-equivalent run that is both eager and well-bracketed immediately follows by induction). Let $a = push(\gamma)$ denote the first action of ρ, and let p denote the process with $a \in A^p$. Let ρ' denote an order-equivalent eager run obtained from ρ by scheduling the actions of p as early as possible, while maintaining adjacent send/receive pairs. It is readily seen that ρ' may be written as:

$$\rho' \;=\; x \xrightarrow{push(\gamma)} x' \cdot \pi_0 \cdot \chi_1 \cdot \sigma_1 \cdot \pi_1 \cdots \chi_n \cdot \sigma_n \cdot \pi_n \cdot y \xrightarrow{pop(\gamma)} y' \cdot \mu$$

where the runs π_i, χ_i and σ_i satisfy the following conditions:

(a) π_i consists of moves of process p which are either local actions or sends that are unmatched in ρ,
(b) χ_i contains no move of process p,
(c) σ_i is a rendezvous involving p,
(d) the transitions $x \xrightarrow{push(\gamma)} x'$ and $y \xrightarrow{pop(\gamma)} y'$ are matching stack actions (of process p),
(e) for each $1 \leq i \leq n$, the run $\chi_i \cdot \sigma_i$ is not order-equivalent to a run of the form $\chi_i' \cdot \sigma_i' \cdot \chi_i''$ where $|\chi_i'| < |\chi_i|$ and σ_i' is a rendezvous involving p.

The scheduling of p's actions as early as possible is expressed by condition (e) (notice that σ_i and σ_i' correspond to the same send/receive pair).
We first show the following claim.

Claim. For each $1 \leq i \leq n$, all processes that move in χ_i have an empty stack at the start and end of χ_i.

To prove the claim, let us denote by $P_i = \{q_1, \ldots, q_k\}$ the set of processes that move in χ_i, ordered by their last occurrence in χ_i. Since the last action in χ_i is performed by q_k, we derive from (e) that the rendezvous σ_i is on a channel between p and q_k. Now let $1 \leq h < k$. It follows from (e) that the last action of q_h in χ_i is a communication action b_h. We have two cases to consider:

- b_h is a send action: If there was no matching receive in ρ', then this send action could be scheduled after σ_i, contradicting (e). Hence, ρ' contains a matching receive, which, by eagerness, is the next action in ρ'. This matching receive is performed by a process q_g with $h < g$.

- b_h is a receive action: Since ρ' is eager, the matching send is the previous action in ρ'. This matching send is performed by a process q_g. Moreover, we must have $h < g$ since, otherwise, this matched send/receive pair could be scheduled after σ_i, contradicting (e).

We obtain that, for every $1 \leq h < k$, the last action of q_h in χ_i is a communication action over a channel c_h satisfying $q_h \overset{c_h}{\longleftrightarrow} q_g$ for some $h < g \leq k$. Let c_k denote the channel of the rendezvous σ_i, and recall that $q_k \overset{c_k}{\longleftrightarrow} p$. Observe that p is unrestricted on c_k since, according to (d), the stack of p is non-empty in σ_i. As the typed topology of $\mathcal{R}$ is non-confluent, we derive that q_h is restricted on c_h for each $1 \leq h \leq k$, since there is a simple undirected path $q_h \overset{c_h}{\longleftrightarrow} \cdots \longleftrightarrow q_k \overset{c_k}{\longleftrightarrow} p$ for each h. It follows that q_h has an empty stack at the end of χ_i.

We have thus shown that, for each $1 \leq i \leq n$, all processes that move in χ_i have an empty stack at the end of χ_i. Now, recall that ρ' is well-formed since it is order-equivalent to ρ. Therefore, all processes that move in χ_i also have an empty stack at the start of χ_i, which concludes the proof of the claim.

It follows from the claim that each run χ_i is well-formed, so μ is also well-formed. Since the runs χ_i and μ are eager, we derive from the induction hypothesis that each χ_i is order-equivalent to a run χ'_i that is both eager and well-bracketed, and, similarly, μ is order-equivalent to a run μ' that is both eager and well-bracketed. Replacing in ρ' each χ_i by χ'_i and μ by μ', yields a run $\rho'' \sim \rho$ that is both eager and well-bracketed (the second condition for well-bracketed runs is satisfied since the runs χ_i contain no move of p). This concludes the proof of the proposition. □

Well-bracketed runs in an (arbitrary) RCPS cannot exploit the full power of the multiple stacks. Indeed, the well-bracketing property ensures that the individual process stacks do not "interact" with each other: a single, global stack is sufficient to simulate the run. More precisely, given an RCPS $\mathcal{R} = \langle \mathcal{T}, \tau, M, (\mathcal{D}^p)_{p \in P} \rangle$, with $\mathcal{D}^p = (Z^p, z^p_{\mathcal{I}}, A^p, \Gamma^p, \Delta^p)$ for each $p \in P$, we construct a product pushdown system $\mathcal{D}^{\otimes}$ that simulates the well-bracketed eager runs of $\mathcal{R}$. Its set of control states is $Z^{\otimes} = P \times (\prod_{p \in P} Z^p) \times 2^P \times 2^C$. A control state $(p, \mathbf{z}, E, G) \in Z^{\otimes}$ means that p is the active process, $\mathbf{z}$ is the current global control state, E is the set of processes that have an empty stack, and G is the set of channels that are "growing", i.e., for which no receive action is possible anymore. The stack alphabet of $\mathcal{D}^{\otimes}$ is the disjoint union $\Gamma^{\otimes} = \bigcup_{p \in P} \Gamma^p$. The stack of $\mathcal{D}^{\otimes}$ will be the concatenation of $|P|$ words $u^p \in (\Gamma^p)^*$, one for each process p, where u^p is empty if and only if $p \in E$.

Let us explain how the simulation of eager, well-bracketed runs works. First, an active process r is non-deterministically chosen, leading to the control state $(r, (z^p_{\mathcal{I}})_{p \in P}, P, \emptyset)$. Then, $\mathcal{D}^{\otimes}$ simulates the behavior of r as expected, using its stack as r would do, but also updates the set E accordingly. To simulate send actions $c!m$, $\mathcal{D}^{\otimes}$ non-deterministically decides whether $c!m$ is actually part of a rendezvous on c (provided that $c \notin G$), or will never be matched. In the former case, $\mathcal{D}^{\otimes}$ simulates (in a single step) the rendezvous $c!m \cdot c?m$. In the latter case, the channel c is added to the set G of "growing" channels. Moreover, in both cases, the communication is performed only if the typed topology allows it, which can be checked using the set E.

The pushdown system $\mathcal{D}^{\otimes}$ may choose non-deterministically, at any time, to switch the active process to some process q. Since the run simulated by $\mathcal{D}^{\otimes}$ is well-bracketed, either q's stack is empty ($q \in E$) or the top stack symbol must belong to Γ^q. Thus, $\mathcal{D}^{\otimes}$ performs this check and then sets the active process to q.

By construction, the pushdown system $\mathcal{D}^{\otimes}$ simulates all runs of $\mathcal{R}$ that are both eager and well-bracketed, and only those runs. Moreover, the size of $\mathcal{D}^{\otimes}$ is bounded by

$|\mathcal{R}|^{\mathcal{O}(|P|\cdot|C|)}$. Since every RCPS can be easily modified in order to reach a given state with all stacks empty we obtain:

Proposition 2.6. *State eager-reachability of an* RCPS *of size n with non-confluent typed topology $\langle \mathcal{T} = (P, C), \tau \rangle$ reduces in* ExpTime *to state reachability for a pushdown system of size $n^{\mathcal{O}(|P|\cdot|C|)}$.*

Since the state reachability problem for pushdown systems is decidable in deterministic polynomial time, we obtain the upper bound:

Proposition 2.7. *The state eager-reachability problem for* RCPS *over a non-confluent typed topology is in* ExpTime.

3. Eager CPS and the Mutex Restriction

The previous section showed how to decide the state eager-reachability problem provided that the topology behaves well w.r.t. pushdowns and communication. A first natural question is whether one can decide if eager runs suffice for solving the reachability problem. A second legitimate question is whether the restriction to eager runs is realistic. We answer to the first question negatively. However, on the positive side we show a restricted class of CPS where eager runs suffice: CPS over cyclic topologies with the mutex restriction. We focus in this section on CPS since the eager condition talks about communication only.

Definition 3.1. A configuration x of a CPS $\mathcal{Q}$ is *mutex* if for every simple undirected cycle $(p_0, c_1, p_1, \ldots, c_n, p_n = p_0)$ in the topology of $\mathcal{Q}$, at most one of the channels c_i is non-empty in x. A run ρ in $\mathcal{Q}$ is *mutex* if each configuration in ρ is mutex.

A CPS $\mathcal{Q}$ is called *mutex* if every configuration reachable in $\mathcal{Q}$ is mutex. We show later in this section that the mutex property is decidable for finite CPS. Notice also that every CPS with polyforest topology is mutex.

Before discussing mutex we first comment on the results of [21] and explain their relation with Theorem 2.2 and Corollary 3.4 below. The latter paper shows that state reachability is decidable for finite CPS over polyforest topologies, and for well-queueing RCPS over directed forests. The proof of the result for RCPS relies on the idea that, on tree topologies, one can reorder runs such that the resulting run has a bounded number of contexts, where in each context only one process executes all its actions by reading on one unique incoming channel from its tree parent (and—in the case of RCPS—solely when its local stack is empty). Hence, the problem reduces to the control-state reachability for a bounded-phase multi-stack pushdown system, a question which was proven to be decidable in doubly exponential time [20]. A simple channel reversal argument allows us to reduce the question for finite CPS over polyforest topologies to directed forests.

We show in the following that mutex CPS are eager. This allows us to apply the results of the previous section and to obtain the decidability of state reachability (for both finite CPS over polyforest topologies and well-queueing RCPS over directed forests) via a direct proof. Moreover, recall that the complexity of the algorithm of the previous section is ExpTime, so one exponential less than the results obtained in [20] for polyforest architectures.

Remark 3.2. Over a topology of two finite processes connected by two channels, mutex runs are referred to as "half-duplex communication". For these, it is known how to decide the reachability problem through an effective construction of the recognizable reachability set [10]. Quasi-stable systems are a semantic ad-hoc extension of this idea to finite CPS with larger, cyclic topologies [9].

Proposition 3.3. *Given a* CPS $\mathcal{Q}$*, every mutex run starting with empty channels admits an order-equivalent eager run.*

Proof. By induction on the length of runs. The basis is trivial. Consider a mutex run ρ of non-zero length, that starts with empty channels. In particular, each receive action in ρ has a matching send in ρ. We write $P_\rho \subseteq P$ for the (non-empty) set of all processes p that move in ρ. For each $p \in P_\rho$, let e_p denote the last action of p in ρ. If some e_p is a local action, or a send action that is not matched in ρ, we may schedule it last, which preserves the run's mutex property, and derive the existence of an eager run $\rho' \sim \rho$ by induction. Otherwise, for each $p \in P_\rho$, the action e_p is a communication action that is matched in ρ, and we let c_p denote the channel of e_p. Note that each c_p, for $p \in P_\rho$, is a channel between p and another process in P_ρ, which we call its last peer. We may build an infinite sequence of processes in P_ρ by picking an arbitrary process in P_ρ and iteratively moving to its last peer. By the pigeonhole principle, there exist $p_0, \ldots, p_n$ in P_ρ, with $n > 0$, such that $(p_0, c_{p_0}, \ldots, p_n, c_{p_n}, p_0)$ is an undirected path in $\mathcal{T}$ and $p_0, \ldots, p_n$ are distinct. Moreover, we may assume w.l.o.g. that p_0 is the process that moves last in ρ among $\{p_0, \ldots, p_n\}$. To simplify notation, let us simply write e_i in place of e_{p_i}, and c_i in place of c_{p_i}. Remark that the undirected path $(p_0, c_0, \ldots, p_n, c_n, p_0)$ must be a simple undirected cycle if $c_0 \neq c_1$.

Let us show that e_1, e_0 is a pair of matching send/receive actions. Since $p_0 \overset{c_0}{\longleftrightarrow} p_1$ and p_1 stops moving before p_0 in ρ, the communication action e_0, which is matched in ρ, must be a receive action $e_0 = c_0?m_0$. We obtain that ρ is of the form:

$$\rho \;=\; \chi \cdot x' \xrightarrow{e_1} y' \cdot \chi' \cdot x'' \xrightarrow{c_0?m_0} y'' \cdot \chi''$$

with no move of p_1 in χ', and no move of p_0, p_1 in χ''. It follows that c_0 is non-empty in y'. Since ρ is a mutex run, x' and y' are mutex configurations. If $c_0 \neq c_1$, then c_0 is also non-empty in x', hence c_1 must be empty in both x' and y', which is impossible since e_1 is communication action on c_1. Therefore, we get that $c_0 = c_1$, and, hence, e_1 is the last send action on c_0 in ρ. Since e_1 is matched in ρ, it follows that e_1 is the matching send of e_0, which implies that $e_1 = c_0!m_0$.

We may now conclude the proof of the proposition. Recall that e_1, e_0 are the last actions of p_1 and p_0 in ρ, respectively. Since $e_1 = c_0!m_0$ and $e_0 = c_0?m_0$ are matched, we may schedule e_1, e_0 last. This leads to a run ρ' that is order-equivalent to ρ, and of the form:

$$\rho' \;=\; \chi \cdot \mu \cdot x_0 \xrightarrow{c_0!m_0} x_1 \cdot \xrightarrow{c_0?m_0} \cdot x_2$$

where the trace of μ satisfies $\mathit{trace}(\mu) = \mathit{trace}(\chi') \cdot \mathit{trace}(\chi'')$. It follows from the previous trace equality that, for each configuration $(\mathbf{s}, \mathbf{w})$ occurring in μ, there exists

- either a configuration $(\mathbf{s}', \mathbf{w}')$ in χ'' with $\mathbf{w} = \mathbf{w}'$,
- or a configuration $(\mathbf{s}', \mathbf{w}')$ in χ' such that $w'^{c_0} = w^{c_0} \cdot m_0$ and $w'^{c} = w^{c}$ for all $c \neq c_0$.

In both cases, we derive that $(\mathbf{s}, \mathbf{w})$ is mutex since $(\mathbf{s}', \mathbf{w}')$ is mutex. Therefore, the run μ is mutex. Moreover, the run χ is also mutex since it is a prefix of the mutex run ρ. We derive from the induction hypothesis that $\chi \cdot \mu$ is order-equivalent to an eager run μ'.

Replacing $\chi \cdot \mu$ by μ' in ρ' yields a run $\rho'' \sim \rho$ that is eager. This concludes the proof of the proposition. □

Corollary 3.4. *Every mutex* CPS *is eager.*

Remark 3.5. A closer look at the proof of Proposition 3.3 shows that the result still holds for the following weaker variant of the mutex property: a configuration x of a CPS $\mathcal{Q}$ is *weakly mutex* if for every simple undirected cycle $(p_0, c_1, p_1, \ldots, c_n, p_n)$ in the topology of $\mathcal{Q}$, at most one of the channels c_1, c_2 is non-empty in x.

We derive the following result as an immediate consequence of Corollary 3.4. The upper bound is obtained as an on-the-fly simulation: since we simulate eager runs we do not have to store any message, but keep track of growing channels. The lower bound follows from the non-emptiness test of the intersection of several regular languages.

Proposition 3.6. *The state reachability problem for finite, mutex* CPS *is* PSPACE*-complete.*

Remark 3.7. State reachability remains decidable for particular *infinite-state* mutex CPS. For example, if each local LTS is a Petri net (i.e., the CPS in question is a FIFO net [13]), then the state reachability problem reduces to the Petri net reachability problem, which is known to be decidable [24, 19].

We end this section by showing that, for finite CPS, the mutex property is decidable (unlike the eager one).

Proposition 3.8. *The question whether a finite* CPS *is mutex, is* PSPACE*-complete.*

Proof. Assume that $\mathcal{Q}$ is not mutex and consider a run ρ of minimal length from $x_{\mathcal{I}}$ to a configuration x that is not mutex. By minimality, all configurations in ρ up to x are mutex. Let x' be the predecessor of x in ρ.

By Proposition 3.3 we can reach x' by an eager run ρ' (which is generated on-the-fly in PSPACE) and test whether there exists in $\mathcal{Q}$ a transition $x' \xrightarrow{c!m} x$ that violates the mutex condition for x. We guess ρ' in PSPACE (see remark above) and check whether there exists a simple undirected cycle $(p_0, c_1, p_1, \ldots, c_n, p_n)$ in the topology of $\mathcal{Q}$ such that one channel c_i is non-empty in x' and the action $c!m$ would write on another channel of this cycle (i.e., $c = c_j$ for some $j \neq i$).

PSPACE-hardness follows, again, by reducing from the non-emptiness test of the intersection of several regular languages. □

Proposition 3.9. *The question whether a finite* CPS *is eager, is undecidable.*

Proof. We show a reduction from the universality problem for rational relations [5]. Given such a relation $K \subseteq A^* \times B^*$, we ask whether $K = A^* \times B^*$. Here, K is described by a finite automaton $\mathcal{A}_K$ over the alphabet $A \cup B$.

We describe a finite CPS over four processes, called $p_0, \ldots, p_3$, and four channels $c_{01}, c_{10}, c_{12}, c_{13}$ satisfying $p_0 \xrightarrow{c_{01}} p_1$, $p_1 \xrightarrow{c_{10}} p_0$, $p_0 \xrightarrow{c_{02}} p_2$, $p_0 \xrightarrow{c_{03}} p_3$. Process p_0 is described in Fig. 2. The ingoing (outgoing, resp.) edges of $\mathcal{A}_K$ lead to the initial state (from the final states, resp.). Transition labels $a \in A$ in $\mathcal{A}_K$ are replaced by $c_{02}!a$, and labels $b \in B$ are replaced by $c_{03}!b$.

Process p_1 is described in Fig. 3. The LTS $\mathcal{A}^{p_2} = \mathcal{A}^{p_3}$ of processes p_2, p_3 consist of a single (initial) state without any transition. Therefore, when talking about "state components" below we only mention processes p_0, p_1.

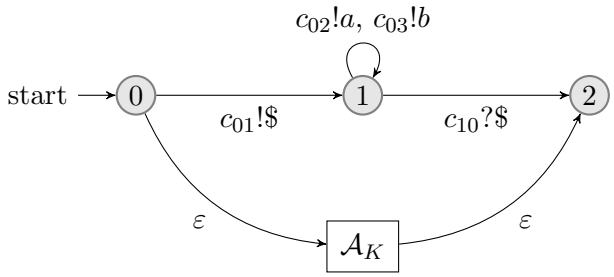


Figure 2: Process p_0 ($a \in A, b \in B$)

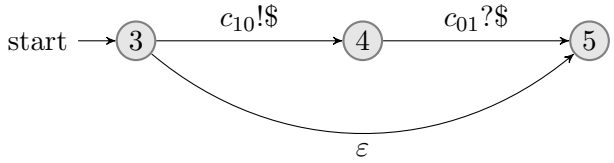


Figure 3: Process p_1

The only runs of the above CPS that cannot be reordered into an eager run are produced by p_0 and p_1 using all four \$-transitions. The state component of these configurations is $(2, 5)$. The channel contents are ε for c_{01} and c_{10}, A^* for c_{02} and B^* for c_{03}. Each of these configurations can be also reached by an eager run if and only if $K = A^* \times B^*$. □

4. Bounded Phase Reachability

Bounded-context reachability has shown to be a successful under-approximation method for the analysis of concurrent Boolean programs [27]. For RCPS, bounded-context reachability allows us to attack the reachability problem from a different angle than in Section 2. In this section, we neither restrict the typed topology, nor constrain the runs to be eager (or mutex). The price to pay is a (strong) restriction on the form of the possible runs, namely a bounded number of switches between processes (i.e., phases). Our construction subsumes the 2-ExpTime algorithm for bounded-context reachability of well-queueing recursive communicating processes, as described in [21]. Recall that the latter algorithm is based on a reduction to bounded-phase reachability for multi-stack systems. In contrast, our construction below is direct and simpler.

A *phase* of an RCPS is a run consisting of moves of a unique process, called the *phase process*. In order to get decidability results one needs to introduce further restrictions over the communications performed during a phase. The first, obvious, restriction is on the typed topology $\langle \mathcal{T}, \tau \rangle$: for every channel c, either the source or the destination process is restricted on c. Moreover, we assume for simplicity that for each channel c, one of the two processes is unrestricted on c. The second type of restriction concerns the kind of communication a process is allowed to perform during a phase, and is defined by two (dual) types of phases, called *mux-phases* and *demux-phases*, respectively.

Let c be a channel with *source* process p that is restricted on c. A phase of process p is a *mux-phase* (with channel c) if the allowed communication for p is either sending into c, or receiving on channels d such that the source process is restricted on d, see also Figure 4.

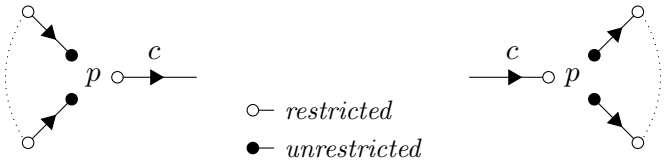


Figure 4: Phases of an RCPS: mux (on the left) and demux (on the right)

Dually, let c be a channel with *destination* p that is restricted on c. A phase of process p is a *demux-phase* (with channel c) if the allowed communication for p is either receiving on c, or sending on channels d such that the destination process is restricted on d. Demux-phases are precisely the phases/contexts used by [21].

A run ρ of an RCPS is said to be *k-bounded*, if it can be decomposed as $\rho = \rho_1 \cdots \rho_k$ where each ρ_j is a mux- or demux-phase. A configuration $x \in X$ is *k-bounded-reachable* in a RCPS $\mathcal{R}$ if there exists a k-bounded run of $\mathcal{R}$ from the initial configuration $x_{\mathcal{I}}$ to x. We define the *k-bounded-reachability set* of $\mathcal{R}$ as $Reach_k(\mathcal{R})$, the set of $x \in X$ that are k-bounded-reachable in $\mathcal{R}$. The *state bounded-reachability problem* for RCPS asks for a given RCPS $\mathcal{R}$, a global control state $\mathbf{z} \in \mathbf{Z}$ and an integer k (in unary encoding), whether $Reach_k(\mathcal{R})$ intersects $\{\mathbf{z}\} \times (\prod_{p \in P} (\Gamma^p)^*) \times (M^*)^C$.

In the remainder of this section we will use an extended version of phases, still denoted as phase for convenience. A *phase* $\phi = (p, \mathcal{D}, z_F)$ will consist, as previously, of a *phase process* $p \in P$ and a pushdown system $\mathcal{D} = (Z, z_I, A, A_\epsilon, \Gamma^p, \Delta)$ as in Section 1.3 (which may be, e.g., the pushdown system of process p in the RCPS, up to changing the initial state). In addition we specify a (control) state $z_F \in Z$, which will be the target state of the phase. A phase is said to be *local* if A_{com} is empty. The *size* $|\phi|$ of a phase ϕ is the number of control states of $\mathcal{D}$. We associate with a phase ϕ the binary relation $\xrightarrow{\phi}$ over $(\prod_{p \in P} (\Gamma^p)^*) \times (M^*)^C$, defined by $(\mathbf{u}_I, \mathbf{v}_I) \xrightarrow{\phi} (\mathbf{u}_F, \mathbf{v}_F)$ if there exists a run from the configuration $(\mathbf{z}_I, \mathbf{u}_I, \mathbf{v}_I)$ to the configuration $(\mathbf{z}_F, \mathbf{u}_F, \mathbf{v}_F)$ in the RCPS obtained by fixing the processes $q \neq p$ to the trivial pushdown system with one state and no transition and the process p to the pushdown system $\mathcal{D}$. A sequence $\Phi = (\phi_1, \ldots, \phi_k)$ of mux- or demux-phases is called an *md-sequence*. Such a sequence is said to be *satisfiable* if the following relation holds:

$$((\varepsilon)_{p \in P}, (\varepsilon)_{c \in C}) \xrightarrow{\phi_1} \cdots \xrightarrow{\phi_k} ((\varepsilon)_{p \in P}, (\varepsilon)_{c \in C})$$

The *size* of an md-sequence $\Phi = (\phi_1, \ldots, \phi_k)$ is $|\Phi| = |\phi_1| + \cdots + |\phi_k|$.

We will decide the satisfiability of md-sequences by reducing the problem to sequences of local phases. The reduction is performed by replacing one by one (de)mux-phases by local phases. We introduce a preorder over md-sequences, that will decrease during the reduction. Let us first define the preorder $\sqsubseteq$ over phases by letting $\phi \sqsubseteq \psi$ if phases ϕ and ψ have the same phase process and the communication actions of ϕ are included in the communication actions of ψ. This preorder is extended component-wise over md-sequences by letting $(\phi_1, \ldots, \phi_k) \sqsubseteq (\psi_1, \ldots, \psi_k)$ if $\phi_j \sqsubseteq \psi_j$ for every j.

Proposition 4.1. *Let $\Phi = (\phi_1, \ldots, \phi_k)$ be an md-sequence with at least one non-local phase. We can compute a finite set F of md-sequences with $|F| \leq |\Phi|^k$ in time $\mathcal{O}(|F|)$ such that Φ is satisfiable if and only if F contains a satisfiable md-sequence, and such that for every $\Psi = (\psi_1, \ldots, \psi_k) \in F$:*

- $|\Psi| \leq 2|\Phi|^2$
- $\Psi \sqsubseteq \Phi$ *and there exists* j *such that* ψ_j *is local whereas* ϕ_j *is not local.*

Proof. Since Φ contains at least one non-local phase, there exists a maximal index j such that ϕ_j is demux non-local, or there exists a minimal index j such that ϕ_j is mux non-local. We first explain why these two cases are symmetric. Given a phase $\phi = (p, \mathcal{D}, z_F)$ where $\mathcal{D} = (Z, z_I, A, A_\epsilon, \Gamma, \Delta)$, let $\bar{\phi} = (p, \bar{\mathcal{D}}, z_I)$ be the phase with $\bar{\mathcal{D}} = (A, z_F, A, A_\epsilon, \bar{\Delta})$ the pushdown system obtained from $\mathcal{D}$ by reversing the channels, exchanging push/pop actions and send/receive actions, and reversing the transition relation. We observe that $(\mathbf{u}, \mathbf{v}) \xrightarrow{\phi} (\mathbf{u}', \mathbf{v}')$ if and only if $(\mathbf{u}', \mathbf{v}') \xrightarrow{\bar{\phi}} (\mathbf{u}, \mathbf{v})$. In particular $(\phi_1, \ldots, \phi_k)$ is satisfiable if and only if $(\bar{\phi}_k, \ldots, \bar{\phi}_1)$ is satisfiable. Since ϕ is a mux (resp. demux) phase if and only if $\bar{\phi}$ is a demux (resp. mux) phase, we obtain that the two cases above are symmetric. Thus, in the remainder of this proof we assume that there exists a maximal index j such that ϕ_j is a non-local demux-phase.

Let $\phi_j = (p, \mathcal{D}, z_F)$ and $\mathcal{D} = (Z, z_I, A, A_\epsilon, \Gamma, \Delta)$ be the pushdown system of ϕ_j. Since ϕ_j is a demux-phase, messages are received from a unique channel, say c. Moreover, process p is restricted on this channel. Let us define the md-sequence Φ^ε from Φ by removing communication actions in the j-th phase.

In the sequel, we show how to build md-sequences $\Phi^\pi = (\phi_1^\pi, \ldots, \phi_k^\pi)$, where Φ^π is parametrized by a sequence $\pi = (z_r)_{s \leq r \leq j}$ of control states $z_r \in Z$ with $s < j$. Each sequence Φ^π is such that $\Phi^\pi \sqsubseteq \Phi$ with ϕ_j^π a local phase. In order to obtain a local phase ϕ_j^π, i.e., a phase without any communication action, all communications with the pushdown system $\mathcal{D}$ are simulated in the phases $\phi_s, \ldots, \phi_j$. Here, the integer s is the index of the first phase that sends messages into channel c, that are received in the j-th phase. We show below that Φ is satisfiable if and only if Φ^ε or Φ^π is satisfiable for some sequence π.

The state sequence $\pi = (z_r)_{s \leq r \leq j}$ provides checkpoints of the simulation of $\mathcal{D}$ during the phases $\phi_s, \ldots, \phi_j$. In particular, states $z_r \in Z$ in π will be assumed by process p with empty stack, and the communication on channel c during phase r takes place between state z_r and state z_{r+1}.

Since p is restricted on channel c, it receives messages from c in the j-th phase with empty stack. Moreover, by the choice of j and the fact that a satisfiable md-sequence must end with empty channels, process p sends no message during phase j (otherwise, there would exist some demux, non-local phase after j, namely one where such messages would be received). By a well-known *saturation algorithm* we can compute in polynomial time (see for example [11]) from $\mathcal{D}$ the set R of pairs of control states $(z, z') \in Z \times Z$ such that there exists an execution of $\mathcal{D}$, consisting of stack actions and local actions only, from (z, ϵ) to (z', ϵ), (i.e., from empty stack to empty stack). Let $\phi_r = (q_r, \mathcal{D}_r, t_{F,r})$ where $\mathcal{D}_r = (T_r, t_{I,r}, A, \Gamma, \Delta_r)$ with $s \leq r \leq j$.

We first provide the definition of ϕ_r^π with $s < r < j$. Recall that $\pi = (z_r)_{s \leq r \leq j}$. The pushdown system $\mathcal{D}_r^\pi$ is obtained by considering $|Z|$ many copies of $\mathcal{D}_r$. Control states of these copies are identified by pairs $(t, z) \in T_r \times Z$. In these copies, actions that send messages to the channel c are directly matched with actions that receive messages in $\mathcal{D}$. More formally for every $(t, c!m, t') \in \Delta_r$ and $(z, c?m, z') \in \Delta$ we add a local action from (t, z) to (t', z'). We also add transitions that simulate the effect of the stack of $\mathcal{D}$. More precisely we add a local action from (t, z) to (t, z') for every $t \in T_r$ and for every $(z, z') \in R$. The initial state $t_{I,r}$ and the final state $t_{F,r}$ are replaced by $(t_{I,r}, z_r)$ and $(t_{F,r}, z_{r+1})$, resp.

The definition of ϕ_s^π follows almost the same construction except that we should take into account the fact that in this phase we first perform moves that potentially send messages in c and then non-deterministically we start to simulate the pushdown system $\mathcal{D}$. The difference is due to the fact that some messages into channel c can be received during some phase before the j-th one. The simulation is performed with the construction presented in the previous paragraph. However we keep in $\mathcal{D}_s^\pi$ the original pushdown system $\mathcal{D}_s$ and we add a local action from t to (t, z_s) for every $t \in T_r$. The initial state $t_{I,s}$ is left unchanged and the final state $t_{s,F}$ is replaced by $(t_{s,F}, z_{s+1})$.

The definition of ϕ_j^π is obtained by a simpler construction. Since messages received from c are simulated in the previous phases, we can remove the communication actions of $\mathcal{D}$. Since the j-th phase may start or end with non-empty stack, we need in addition an extra copy of $\mathcal{D}$ (also without communication actions). The copy of a control state z is denoted by $\tilde{z}$. We then add a local action from z_s to $\tilde{z}_j$ with the empty stack guard, i.e., this local action belongs to A_ϵ. This action accounts for the simulation of $\mathcal{D}$ between state z_s and state z_j. Moreover, the initial control state z_I is left unchanged and the final state is replaced by $\tilde{z}_F$.

Finally, the phases ϕ_r^π with $r < s$ or $r > j$ are equal to ϕ_r. We observe that Φ is satisfiable if and only if Φ^ε is satisfiable or there exists a sequence π such that Φ^π is satisfiable. Defining F as the set of md-sequences Φ^π and the additional md-sequence Φ^ε concludes the proof. □

Corollary 4.2. *The satisfiability of an md-sequence Φ of length k can be checked in time doubly exponential in k (but polynomial in the size of Φ).*

Proof. Since the reduction introduced by applying Proposition 4.1 transforms at least one non local phase into a local one, after at most k steps we obtain a finite set F of local phases. Moreover an immediate induction based on Proposition 4.1 also shows that every $\Psi \in F$ has size $|\Psi| \leq 2^k|\Phi|^{2^k}$. The size of F can be bounded by the number of leaves of a tree of height k with rank bounded by $(2^k|\Phi|^{2^k})^k$. Thus $|F| \leq ((2^k|\Phi|^{2^k})^k)^k$. The satisfiability of a sequence $\Psi \in F$ can be performed in time $\mathcal{O}(|\Psi|^2)$, since the empty stack control state reachability problem for pushdown systems is decidable in polynomial time. We conclude that the satisfiability of an md-sequence can be checked in 2-ExpTime, but polynomially in $|\Phi|$ when k is fixed. □

Theorem 4.3. *The state bounded-reachability problem for* RCPS *with typed topology such that each channel is restricted at least at one extremity, is* 2-ExpTime*-complete. If the number of phases and the typed topology are not part of the input, the problem can be solved in polynomial time.*

Proof. For the upper bound we can assume w.l.o.g. that we reach the target control state with all stacks and channels empty. For this, we can choose non-deterministically the push actions that will not be matched and, for each channel, the first message that will be no longer received. The bound follows then from Corollary 4.2.

For the lower bound we can adapt proof ideas from [3, 22], by showing how to simulate alternating Turing machines M of exponential space by RCPS with typed topology as in the statement of the theorem. If the space bound of M is 2^k we use $\mathcal{O}(k)$ processes, called p_0 and p_i, q_i^o, q_i^e, $1 \leq i \leq k$. Process p_0 is the only one using a stack, storing an accepting computation tree of M. We will not go into the details how to encode the tree (it is the usual depth-first traversal of the tree, plus appropriate encoding of transitions), see e.g. [3]

for details. Instead we explain now how to check that the contents of the stack of p_0 is a word of the form $(w\#)^m$ for some $w \in \{0,1\}^{2^k}$ and $m > 0$.

In the first phase, process p_0 empties its stack and while doing this, sends the following to q_1^o, q_1^e:

- q_1^o: every symbol of w at an odd position,
- q_1^e: every symbol of w at an even position.

Assuming that the stack content of p_0 is $w_1\# \cdots w_m\#$, the outgoing channels of p_0 will contain after this first stage, the following words (u^o and u^e denotes the subword of u at odd and even positions, respectively):

- $w_1^o\# \cdots \#w_m^o\#$ for (p_0, q_1^o),
- $w_1^e\# \cdots \#w_m^e\#$ for (p_0, q_1^e).

In the second and third phase, process q_1^o, and then q_1^e, receives from p_0 and resends each message to p_1. In phases 4 and 5, process p_1 receives $w_1^o\# \cdots \#w_m^o\#$ from q_1^o, and then $w_1^e\# \cdots \#w_m^e\#$ from q_1^e. In each of these phases p_1 resends to q_2^o and q_2^e its odd/even subwords as p_0 above, adding a separator \$ between the two halves. So process p_1 acts basically like p_0, but on "input" of the form $w_1^o\# \cdots \#w_m^o\#\$w_1^e\# \cdots \#w_m^e\#$, where one has to check equality for words of length 2^{k-1}: $w_1^o = \cdots = w_m^o$ and $w_1^e = \cdots = w_m^e$, respectively. This procedure is iterated up to process p_k, that simply checks that it receives two words from q_k^o, q_k^e of the form $((0\#0\# + 1\#1\#)\$^+)^*$.

The above proof for stack contents of the form $w\#w\# \cdots w\#$ for some $w \in \{0,1\}^{2^k}$, is of course a special case of the Turing machine simulation, however it captures the main idea. For the Turing machine it is readily seen how to extend the proof to a sequence of configurations $w_1\#w_2\# \cdots w_k\#$, where w_{i+1} is the successor configuration of w_i. Here, it helps to see each w_i as a sequence of 3 tape symbols, i.e., each position stores the current symbol, plus its neighbors. In addition, one encodes the transitions leading from w_i to w_{i+1}, say after each #. For the final check, process p_k will check that the first triple is consistent with the middle symbol of the second triple. □

5. Conclusion

Applications. CPS combine an automata-based local process model with point-to-point communication, which results in an intuitive and simple framework.

Since we subsume well-queueing RCPS, we also inherit their application domains, e.g., event-based programs. The dual restriction to well-queueing (i.e., that sending on a channel is only possible if the stack is empty) covers, e.g., "interrupt based" programming models, i.e., threads that can receive messages *while* still in recursion, as well as extended sensor networks where peers can collect and send data *while* using their pushdown for computations.

Figure 5 shows an example for non-confluent typed topologies that are on the rise with the current focus on distributed computing. The topology corresponds to a hierarchical overlay network as implemented, for example, in master-worker protocols. Intuitively, each master distributes tasks to its workers and uses their results during its own computation. When the latter is finished, i.e., when its stack is empty, the master sends a result to its own master. Therefore, channel restrictions respect the hierarchy: channels between a master and a worker must be restricted on the worker's side. In fact, our generic non-confluence

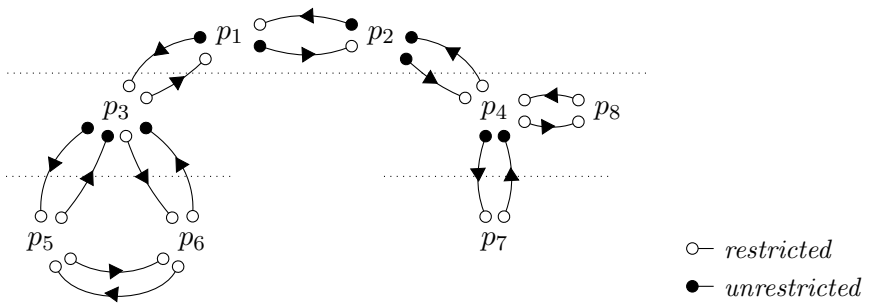


Figure 5: Non-confluent typed topology in a hierarchical master-worker setting

criterion permits additional communications: workers of the same master may communicate with each other via channels on which they are restricted (e.g., p_5 and p_6), and we may have a communication cycle between top-level masters (e.g., p_1 and p_2). Notice also the use of the dual notion to well-queueing, when sending information from lower to higher levels.

Proposition 3.3 allows for further applications, since it does not assume that the CPS is finite: we can combine locally decidable models for multi-threaded programs (with or without local data), as well as local event-based programs together with eager (or mutex) communication architectures; natural candidates for local models would be Petri Nets, well-structured transition systems [14], or multi-set pushdown systems [29].

Summary. We discussed in detail the class of eager RCPS (as well as mutex CPS) which both generalize the current lineup of decidable models for asynchronously communicating pushdown systems. Further, we presented an optimal decision procedure for eager RCPS over non-confluent architectures in ExpTime, as well as a direct and simpler construction for bounded phase reachability for RCPS.

Outlook. This paper dealt with the most basic form of verification, namely control-state reachability. More general reachability questions (w.r.t. configurations) may be interesting to consider. Further decision problems for CPS, like boundedness or liveness, will be investigated in future work.

References

[1] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996.

[2] M. F. Atig. From multi to single stack automata. In *Proc. of CONCUR 2010*, volume 6269 of *LNCS*, pages 117–131. Springer, 2010.

[3] M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-complete. In *Proc. of DLT 2008*, volume 5257 of *LNCS*, pages 121–133. Springer, 2008.

[4] M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *Proc. of CONCUR 2008*, volume 5201 of *LNCS*, pages 356–371. Springer, 2008.

[5] J. Berstel. *Transductions and context-free languages.* Teubner Studienbücher, Stuttgart, 1979.

[6] A. Bouajjani, J. Esparza, S. Schwoon, and J. Strejcek. Reachability analysis of multithreaded software with asynchronous communication. In *Proc. of FSTTCS 2005*, volume 3821 of *LNCS*, pages 348–359. Springer, 2005.

[7] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *Proc. of CONCUR 2005*, volume 3653 of *LNCS*, pages 473–487. Springer, 2005.

[8] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. of ACM*, 30(2):323–342, 1983.

[9] G. Cécé and A. Finkel. Programs with quasi-stable channels are effectively recognizable. In *Proc. of CAV 1997*, volume 1254 of *LNCS*, pages 304–315, 1997.
[10] G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
[11] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. of CAV 2000*, volume 1855 of *LNCS*, pages 232–247. Springer, 2000.
[12] J. Esparza, A. Kucera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355–376, 2003.
[13] A. Finkel and L. E. Rosier. A survey on the decidability questions for classes of FIFO nets. In *European Workshop on Applications and Theory of Petri Nets*, volume 340 of *LNCS*, pages 106–132. Springer, 1987.
[14] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
[15] B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.
[16] B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80:147–167, 2007.
[17] R. Jhala and R. Majumdar. Interprocedural analysis of asynchronous programs. In *Proc. of POPL 2007*, pages 339–350. ACM, 2007.
[18] N. Kidd, S. Jagannathan, and J. Vitek. One stack to run them all. In *Proc. of SPIN 2010*, volume 6349 of *LNCS*, pages 245–261. Springer, 2010.
[19] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. of STOC 1982*, pages 267–281. ACM, 1982.
[20] S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *Proc. of LICS 2007*, pages 161–170. IEEE Computer Society, 2007.
[21] S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *Proc. of TACAS 2008*, volume 4963 of *LNCS*, pages 299–314. Springer, 2008.
[22] S. La Torre, P. Madhusudan, and G. Parlato. An infinite automaton characterization of double exponential time. In *Proc. of CSL 2008*, volume 5213 of *LNCS*, pages 33–48. Springer, 2008.
[23] M. Lohrey and A. Muscholl. Bounded MSC communication. *Inf. Comput.*, 189(2):160–181, 2004.
[24] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.
[25] R. Mayr. Process rewrite systems. *Inf. Comput.*, 156(1-2):264–286, 2000.
[26] C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
[27] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *Proc. of TACAS 2005*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
[28] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.
[29] K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *Proc. of CAV 2006*, volume 4414 of *LNCS*, pages 300–314. Springer, 2006.
[30] A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In *Proc. of CAV 2010*, volume 6174 of *LNCS*, pages 615–628. Springer, 2010.