Reachability Analysis of Communicating Pushdown Systems

Alexander Heußner*, Jérôme Leroux, Anca Muscholl, and Grégoire Sutre

LaBRI, Université Bordeaux, CNRS – France

Abstract. The reachability analysis of recursive programs that communicate asynchronously over reliable FIFO channels calls for restrictions to ensure decidability. We extend here a model proposed by La Torre, Madhusudan and Parlato [16], based on communicating pushdown systems that can dequeue with empty stack only. Our extension adds the dual modality, which allows to dequeue with non-empty stack, and thus models interrupts for working threads. We study (possibly cyclic) network architectures under a semantic assumption on communication that ensures the decidability of reachability for finite state systems. Subsequently, we determine precisely how pushdowns can be added to this setting while preserving the decidability; in the positive case we obtain exponential time as the exact complexity bound of reachability. A second result is a generalization of the doubly exponential time algorithm of [16] for bounded context analysis to our symmetric queueing policy. We provide here a direct and simpler algorithm.

Introduction

The verification of safety properties for distributed programs, e.g., client/server environments, peer-to-peer networks, or Grid applications, relies on the decidability of the reachability problem. In this paper we reconsider *recursive queueing concurrent processes* (RQCP), one possible model for such systems which was studied recently by La Torre, Madhusudan, and Parlato [16]. It is a natural idea to combine peer-to-peer asynchronous communication (via point-to-point, unbounded, reliable FIFO channels) with some automaton-based model for individual peers (e.g., pushdown automata or Petri nets). We call such combined models *queueing concurrent processes* (QCP). Since communicating finite-state automata are the most elementary instantiation of QCP, reachability is in general undecidable [6]. Furthermore, adding recursion (i.e., replacing finite-state by pushdown automata) yields an additional source of undecidability. One of the main motivations in this paper is to separate these two sources of undecidability: we consider behavioral restrictions for which reachability for communicating finite-state machines is decidable, and then look under which conditions we can add recursion to the model. The challenging task is to derive conditions that conserve the simplicity and expressiveness of the model.

* Work supported by the ANR project AVERISS.

L. Ong (Ed.): FOSSACS 2010, LNCS 6014, pp. 267–281, 2010.

In general, there are three main directions to cope with the undecidability of communicating finite-state machines: restricting the communication architecture, assuming that channels are lossy, or adding semantic restrictions, e.g., that sending/receiving of messages can be scheduled in such a way that runs can be executed with channels of bounded size. In this paper we stick to the latter approach, as described in more detail below. Our point of departure is the work of La Torre et al. [16], which introduced RQCP together with a behavioral restriction on the combined use of channels and pushdowns. Informally, a RQCP is *well-queueing* if pushdown processes can only dequeue (read) messages when the stack is empty (they can enqueue messages without 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 [16] is that RQCP have a decidable reachability problem if and only if their communication architecture is a directed forest; in the decidable case, the latter paper provides a doubly exponential upper bound by a reduction to bounded-phase multi-stack pushdown systems [15].

Our contribution. We extend the work of La Torre et al. [16] in several directions. First, we add a dual notion to well-queueing: a pushdown process can enqueue (send) messages only with empty stack (but can dequeue messages without restriction). *Oriented* communication architectures, as presented here, combine these two notions, by fixing the behaviour of the two endpoints of each channel. This dual notion to well-queueing 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.

Second, we exhibit a precise characterization of those oriented architectures for which the RQCP model has a decidable reachability problem over so-called *eager* runs. Informally, a run is eager if the sending of a message is immediately followed by its reception, a notion closely related to existentially 1-bounded communication [14]. Eagerly communicating finite-state machines are a well-studied model, enjoying good expressiveness and decidability properties [10]. Here, we use eager runs in order to rule out undecidability stemming from unbounded channels. We show that reachability of RQCP over eager runs is EXPTIME-complete in the decidable case. This result generalizes and improves the doubly exponential time decision procedure of [16], which holds for architectures without undirected cycles (*polyforest* architectures).

Eagerness is a relatively strong requirement, hence, we show how it arises rather naturally, by imposing a semantic restriction on the communication flow: the *mutex* restriction demands that in every reachable configuration there is no more than one non-empty channel per cycle. In particular, QCP over polyforest architectures are mutex. Actually mutex can be seen as a generalization of the half-duplex restriction studied in [7].

La Torre et al. [16] propose a second approach to solve the reachability problem for RQCP, inspired by recent work on reachability with bounded contexts in the verification of concurrent programs [19]. They show that bounded-context reachability for well-queueing RQCP 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 [15]. Our second main contribution is to extend the bounded-context result to RQCP 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.

A long version of this paper that includes all proofs omitted due to space limitations can be found at `http://hal.archives-ouvertes.fr/hal-00443529/`.

Related work. In the context of thread programming, other notions of synchronization between pushdowns arise naturally. Earlier publications considered synchronization via shared memory, such as local/global memory in [4,5] or bags in [20,12]. The paper [4] showed that bounded-context reachability can be solved in exponential time, whereas [20] provided an exponential space lower bound for reachability (without context bounds). More recently, synchronization in the form of state observation was considered in [2]. The latter model was shown to be decidable only for acyclic architectures, and is strongly related to lossy systems [3,9].

1 Queueing Concurrent Processes

We write $X \cup X'$ for the *disjoint* union of X and X'. Let Σ denote an *alphabet* (i.e., a finite set of *letters*). We write Σ^* for the set of all *finite words* (*words* for short) over Σ, and we let ε denote the *empty word*. Moreover, we use standard complexity classes such as polynomial space (PSPACE), deterministic exponential time (EXPTIME), and doubly exponential time (2EXPTIME). For more detailed definitions the reader is referred to textbooks like [18].

A *communication architecture* $\mathcal{T}$ (or architecture for short) is a pair $\langle \mathcal{P}, \mathrm{Ch} \rangle$ with a finite non-empty set $\mathcal{P}$ of processes and a finite set of point-to-point channels $\mathrm{Ch} \subseteq (\mathcal{P} \times \mathcal{P}) \setminus id_{\mathcal{P}}$.

Remark 1. Our definition of architecture forbids self-loops, as well as two distinct channels in the same direction between a pair of processes (without further restriction, these settings are immediately Turing equivalent).

Definition 1. *A system of* queueing concurrent processes (QCP) *over a given architecture $\mathcal{T} = \langle \mathcal{P}, Ch \rangle$ is a tuple $\mathcal{A} = \langle (S_p)_{p\in\mathcal{P}}, (\Sigma_p)_{p\in\mathcal{P}}, (\Delta_p)_{p\in\mathcal{P}}, (s^0_p)_{p\in\mathcal{P}}, M \rangle$ with M a finite message alphabet. For each process $p \in \mathcal{P}$, the tuple $\langle S_p, \Sigma_p, \Delta_p, s^0_p \rangle$ describes a (local) transition system on the state set S_p with actions from $\Sigma_p = \Sigma^{loc}_p \uplus \Sigma^{com}_p$, which are either local (i.e., in Σ^{loc}_p) or communication actions in $\Sigma^{com}_p = \{p!q(m) \mid (p,q) \in Ch \text{ and } m \in M\} \cup \{p?q(m) \mid (q,p) \in Ch \text{ and } m \in M\}$. Local transitions are given by the rules in $\Delta_p \subseteq S_p \times \Sigma_p \times S_p$, and the initial state of process p is s^0_p.*

The global state space is $S = \prod_{p\in\mathcal{P}} S_p$ and the global initial state is $s^0 = (s^0_p)_{p\in\mathcal{P}} \in S$. By $\Sigma = \bigcup_{p\in\mathcal{P}} \Sigma_p$ we denote the set of all possible actions in $\mathcal{A}$. The size *of $\mathcal{A}$ is $\sum_{p\in\mathcal{P}} |\Delta_p|$.*

As usual, $p!q(m)$ denotes the send of message m from process p to process q, whereas $q?p(m)$ denotes the matching receive on process q.

Note also that the S_p and hence S need not necessarily be finite. If S is finite, we will call $\mathcal{A}$ a *finite* QCP (or communicating finite-state machine, CFM). The local transition systems given by $\langle S_p, \Sigma_p, \Delta_p, s_p^0 \rangle$ could be, for example, finite automata, counter automata (including Petri nets), pushdown automata. As usual, we define the semantics of $\mathcal{A}$ as labeled infinite-state transition system:

Definition 2. *A* QCP $\mathcal{A}$ *represents an* LTS $[\![\mathcal{A}]\!] = \langle C, \Sigma, \rightarrow, c^0 \rangle$ *with configurations* $C = S \times (M^*)^{Ch}$ *and the initial configuration* $c^0 = \big(s^0, (\varepsilon, \ldots, \varepsilon)\big)$*, i.e., all channels are initially empty. We write a configuration as* $c = \langle s, w \rangle$ *where* $s = (s_p)_{p \in \mathcal{P}}$ *is the global state and* $w = (w_{p,q})_{(p,q) \in Ch}$ *are the channel contents. Further, for any* $p \in \mathcal{P}$ *and* $a \in \Sigma_p$, $\langle s, w \rangle \xrightarrow{a} \langle s', w' \rangle$ *is a transition in* $C \times \Sigma \times C$ *with* $s' = (s'_p)_{p \in \mathcal{P}}$, $w' = (w'_{p,q})_{(p,q) \in Ch}$, *if* $(s_p, a, s'_p) \in \Delta_p$ *and the following holds:*

(i) $s_q = s'_q$ *for all* $q \neq p$,
(ii) if $a \in \Sigma_p^{loc}$ *then* $w = w'$,
(iii) if $a \in \Sigma_p^{com}$ *with* $a = p!q(m)$ *then* $w'_{p,q} = w_{p,q} m$ *and* $w'_{s,t} = w_{s,t}$ *for* $(s,t) \in Ch \backslash \{(p,q)\}$,
(iv) if $a \in \Sigma_p^{com}$ *with* $a = p?q(m)$ *then* $w_{q,p} = m w'_{q,p}$ *and* $w'_{s,t} = w_{s,t}$ *for* $(s,t) \in Ch \backslash \{(q,p)\}$.

A *finite run* ρ in the labeled transition system $[\![\mathcal{A}]\!]$ from a configuration c_0 to $c_n \in C$ is a sequence $\langle c_0, a_1, c_1, a_2, c_2, \cdots, a_n, c_n \rangle$ where $c_{i-1} \xrightarrow{a_i} c_i$ for $1 \leq i \leq n$. The *length* of ρ is n, and a run of length 0 is defined as $\rho = \langle c_0 \rangle$.

A configuration $c \in \mathcal{C}$ is *reachable* in the QCP $\mathcal{A}$ if there exists a finite run $\rho = \langle c_0, a_1, c_1, \cdots, c_n \rangle$ starting in the initial configuration $c_0 = c^0$ and ending in $c_n = c$. We define the *reachability set* as $Reach_{\mathcal{A}} = \{c \in \mathcal{C} \mid c \text{ is reachable in } \mathcal{A}\}$. The *reachability question* asks for a given $\mathcal{A}$ and a configuration $c \in \mathcal{C}$ whether $c \in Reach_{\mathcal{A}}$. Given a state $s \in S$, the *control state reachability question* asks whether one can reach a configuration with (control) state component s regardless of the channel content, i.e., whether $\{s\} \times (M^*)^{\mathrm{Ch}} \cap Reach_{\mathcal{A}} \neq \emptyset$. Both questions are undecidable for finite QCP with at least two processes that are connected by two channels [6].

The *trace* of a run $\rho = \langle c_0, a_1, c_1, a_2, c_2, \cdots, a_n, c_n \rangle$ is the sequence of actions $tr(\rho) = a_1 \cdots a_n \in \Sigma^*$. Since channels are FIFO, we can speak about *matching* send/receive pairs: a_i, a_j form such a pair if (1) $a_i = p!q(m)$, $a_j = q?p(m)$, and (2) $|\{\ell \mid \ell \leq i, a_\ell = p!q(n), n \in M\}| = |\{\ell \mid \ell \leq j, a_\ell = q?p(n), n \in M\}|$. We call two runs ρ, ρ' *order-equivalent* if they can be transformed one into the other by iteratively commuting adjacent transitions labeled by a and b, resp., such that (i) a, b do *not* belong to the same process, and (ii) a, b are *not* a matching send/receive pair.

Lemma 1. *If* ρ, ρ' *are order-equivalent runs of* $\mathcal{A}$ *starting in the same configuration, then* ρ, ρ' *end in configurations with the same control state.*

Definition 3. *A run* ρ *with trace* $tr(\rho) = a_1 \cdots a_n$ *is* eager *if the following holds: if* $a_i = p!q(m)$ *for some* $0 \leq i < n$ *then either* $a_{i+1} = q?p(m)$ *or no action* a_j *with* $j > i$ *is a receive on* (p, q).

A channel that does not permit further receives is in its "growing phase".

A QCP $\mathcal{A}$ is *eager* if each $c \in \mathit{Reach}_{\mathcal{A}}$ is reachable by some eager run. Eager runs, modulo the fact that Definition 3 allows for runs which end in a sequence of (unmatched) send actions, are closely related to globally 1-bounded runs, whereas eager QCP are close to existentially globally 1-bounded CFM [14,11]. The *(control-state) reachability question on eager runs* asks whether one can reach a given (control state) configuration by some eager run.

Recursive QCPs and oriented communication architectures. In the following we introduce RQCP together with a symmetric version of the "well-queueing" property from [16]. Informally speaking, RQCP are QCP where all basic processes are pushdown automata.

A *recursive* QCP (RQCP) is a QCP given by $\langle (S_p)_{p\in\mathcal{P}}, \Sigma, (\Delta_p)_{p\in\mathcal{P}}, s^0, M, \Gamma\rangle$ where each process p is a pushdown process over the local states $S_p \subseteq Z_p \times \Gamma^*$ with a finite set Z_p of control states and the content of the pushdown stack represented by a word over the stack alphabet Γ; further, the local actions Σ_p^{loc} contain $\mathit{push}(\gamma)$ and $\mathit{pop}(\gamma)$ for each $\gamma \in \Gamma$, and we assume that in the initial state all stacks are empty, i.e., $s^0 \in \prod_{p\in\mathcal{P}}(Z_p \times \{\varepsilon\})$.

A well-queueing RQCP in [16] 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.

Definition 4. *An architecture $\mathcal{T} = \langle \mathcal{P}, \mathit{Ch}\rangle$ together with a labeling of the channels $\mathit{Ch} = \mathit{Ch}_{\mathtt{s}} \cup \mathit{Ch}_{\mathtt{r}}$ as "send restricted" ($\mathit{Ch}_{\mathtt{s}}$) and/or "receive restricted" ($\mathit{Ch}_{\mathtt{r}}$), is called* oriented.

For pushdown networks the previous definition translates, informally speaking, as follows: a process p can send on a channel $(p, q) \in \mathrm{Ch}_s$ only with empty stack. Symmetrically, p can receive on a channel $(q, p) \in \mathrm{Ch}_r$ only with empty stack. By definition, channels are restricted at least at one end.

The semantics of an RQCP $\mathcal{R}$ is given by a labeled transition system $[\![\mathcal{A}]\!] = \langle C, \Sigma, \rightarrow, c^0\rangle$ analogously to Definition 2 except for transitions $(s, w) \xrightarrow{a} (s', w')$ that correspond to a (local) pushdown transition $(s_p, a, s'_p) \in \Delta_p$:

(i)' if $a \in \Sigma_p^{loc}$,
then $w = w'$ and further *push* and *pop* behave as local pushdown actions;

(ii)' for $a = p!q(m) \in \Sigma_p^{com}$,
we demand additionally to (ii) that if $(p, q) \in \mathrm{Ch}_{\mathtt{s}}$ then $s_p \in Z_p \times \{\varepsilon\}$;

(iii)' for $a = p?q(m) \in \Sigma_p^{com}$,
we demand additionally to (iii) that if $(q, p) \in \mathrm{Ch}_{\mathtt{r}}$ then $s_p \in Z_p \times \{\varepsilon\}$.

Given an oriented architecture $\mathcal{T} = \langle \mathcal{P}, \mathrm{Ch}\rangle$ we will use the following notation, that forgets about the direction of the channels and focuses on the (un)limited use of pushdowns: for two processes $p, q \in \mathcal{P}$ we write $p \bullet\!\!-\!\!\circ\, q$ if $(p, q) \in \mathrm{Ch} \setminus \mathrm{Ch}_{\mathtt{s}}$ or $(q, p) \in \mathrm{Ch} \setminus \mathrm{Ch}_{\mathtt{r}}$. Moreover, we write $p \circ\!\!-\!\!\circ\, q$ if $(p, q) \in \mathrm{Ch}_{\mathtt{s}} \cap \mathrm{Ch}_{\mathtt{r}}$ or $(q, p) \in \mathrm{Ch}_{\mathtt{s}} \cap \mathrm{Ch}_{\mathtt{r}}$.

Informally, $p \bullet\!\!-\!\!\circ\, q$ means that for at least one channel between p and q, process p can use its stack without restriction. Similarly,

$p \circ\!\!-\!\!\circ q$ means that neither p nor q can use their stacks when communicating. Finally, $p \circ\!\!-\!\!\bullet q$ is equivalent to $q \bullet\!\!-\!\!\circ p$.

As this notation refers implicitly to a given channel between p and q, we might have both $p \bullet\!\!-\!\!\circ q$ as well as $p \circ\!\!-\!\!\bullet q$ (or $p \circ\!\!-\!\!\circ q$) — since both channels (p, q) and (q, p) may exist.

Remark 2. A channel can be both send and receive restricted, but we exclude — per definition — channels that are unrestricted at both ends, as this leads immediately to undecidability: one can reduce right away the intersection of two context-free languages to the reachability question on a topology with a single channel that is unrestricted at both ends (and eager runs suffice).

2 Decidable Oriented Architectures

Several factors lead to the undecidability of the (control-state) reachability question for RQCP. Particularly, the model is already undecidable even without pushdowns. Our actual motivation in this section is therefore to separate the undecidability which stems from unbounded FIFO queues from the undecidability originating from an unrestricted usage of pushdowns. Hence, we consider a restricted version of the control state reachability question, namely the one on eager runs. In the next section, we show how eager QCP naturally arise from some natural (and decidable) restrictions on cyclic communication. The most simple example of an eager QCP is an RQCP on a polyforest architecture.

Definition 5. *An oriented architecture $\mathcal{T} = \langle \mathcal{P}, Ch \rangle$ is called* confluent *if there exist distinct processes $p = r_0, r_1, \ldots, r_k, r_{k+1} = q$ ($k \geq 1$) in $\mathcal{T}$, satisfying the following conditions: (1) $(r_i, r_{i+1}) \in Ch \cup Ch^{-1}$ for all $0 \leq i \leq k$, and (2) $p \bullet\!\!-\!\!\circ r_1$ and $r_k \circ\!\!-\!\!\bullet q$.*

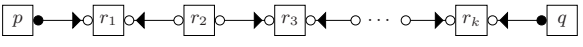

Fig. 1. Example of a confluent architecture (mixing $\bullet\!\!-\!\!\circ$ and $\rightarrow = $ Ch notation)

Theorem 1. *An oriented architecture $\mathcal{T} = \langle \mathcal{P}, Ch \rangle$ admits a decidable* RQCP *control-state reachability problem on eager runs if and only if it is non confluent. Moreover, the problem is* EXPTIME*-complete in the latter case.*

The only-if direction of the theorem above is not difficult to show. The if-direction is based on two main ingredients: first, we show how to reorder runs such that we can identify subruns on subarchitectures that start and end with empty stacks; second, we use induction on subarchitectures.

The subsequent lemma is the core of the remaining proof of Theorem 1.

Lemma 2. *Let $\mathcal{R}$ be an* RQCP *over a non-confluent architecture $\mathcal{T}$, and consider an eager run ρ of $\mathcal{R}$ starting with all stacks empty. Further assume that $\rho = \rho_1 a \, \rho_2 b \, \rho_3$ with $a, b \in \Sigma$ such that for some process $p \in \mathcal{P}$:*

(i) the stack of p is continuously empty during both subruns ρ_1 and ρ_3, and continuously non-empty during ρ_2, respectively;
(ii) there is no *eager run ρ' that is order-equivalent to ρ and has the form $\rho' = \rho_1 c\, \rho_2' \, d\, \rho_3$ where $c, d \in \Sigma$ with $c \neq a$ or $d \neq b$.*

Then, the stacks of all processes occurring in ρ_2 are empty after both ρ_1 and ρ_2.

Proof. By (ii), each process q occurring in ρ_2 has a "proof" for its presence in ρ_2. This proof consists of a simple, unoriented path in $\mathcal{T}$ between p and q.

We explain this more formally: notice first that a is necessarily a push action and b its matching pop action on p. Consider now the first action of some q occurring in ρ_2: this is a communication either with p, or with some different process r. Thus, we can assume inductively that there is a simple unoriented path between p and r, which can be extended to a path between p and q.

Since p's stack is continuously nonempty during ρ_2, we know that the communication between p and the first node p' on the path above is on a channel of type $p \bullet\!\!-\!\!\circ p'$. Due to the non-confluent property, we cannot have $r \circ\!\!-\!\!\bullet q$; hence, q's stack must be empty at the beginning of ρ_2. A symmetric argument applies at the end of ρ_2. □

Proposition 1. *The control-state reachability problem for* RQCP *on eager runs over non-confluent architectures is* EXPTIME *complete.*

Proof. For the upper bound we show how to compute inductively in EXPTIME all pairs of global states $(s, s') \in S^2$ such that there is an eager run over $\mathcal{T} = \langle \mathcal{P}, \mathrm{Ch} \rangle$, starting in s with stacks and queues empty, as well as ending in s' with possibly empty stacks and queues. Actually, we need to compute more, namely for which sets $P \subseteq \mathcal{P}$ of connected processes we can reach s' from s (with empty stacks and queues). W.l.o.g. we assume that $\mathcal{T}$ is connected.

We (arbitrarily) order the processes in $\mathcal{P}$ and choose the first process $p \in \mathcal{P}$ that has at least one edge (channel) of type $p \bullet\!\!-\!\!\circ *$ in $\mathcal{T}$ *and* at least one of type either $p \circ\!\!-\!\!\bullet *$ or $p \circ\!\!-\!\!\circ *$ (with $*$ denoting an arbitrary process in $\mathcal{P} \setminus \{p\}$). Since the architecture is non confluent, only two cases can occur if such a process does *not* exist: either (i) $\mathcal{T}$ contains no $\bullet\!\!-\!\!\circ$ edge at all, or (ii) $\mathcal{T} = \langle \mathcal{P} = \langle r, r_1, \ldots, r_k \rangle, \mathrm{Ch} \rangle$, where:

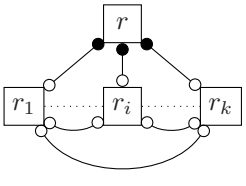

Fig. 2. Case (ii)

- a channel of type $\bullet\!\!-\!\!\circ$ is one of (r, r_i), for some i,
- a channel of type $\circ\!\!-\!\!\circ$ is one of (r_i, r_j), for some i, j.

Let us first consider case (i) where $\mathcal{T}$ has no channel of type $\bullet\!\!-\!\!\circ$, i.e., all channels in $\mathcal{T}$ are of type $\circ\!\!-\!\!\circ$. In this case we can reorder any eager run into an order-equivalent eager run where messages alternate with local pushdown runs, each of them starting and ending with empty stack. This amounts to solving the control-state reachability problem for *one* pushdown automaton of size exponential in $|\mathcal{P}|$, which is possible in EXPTIME.

We consider now case (ii). Here, we may assume w.l.o.g. that $r \bullet\!\!-\!\!\circ r_i$ for every $1 \leq i \leq k$ (just add channels of type $r \bullet\!\!-\!\!\circ r_i$ if there is no channel between r and r_i). Assume that there exists an eager run from s to s', starting with all stacks empty. For simplicity we also assume that the run ends with all channels empty (the special case of unmatched sends can be handled similarly). This run can be reordered into an order-equivalent eager run ρ of the following form:

$$\rho = \sigma_0\alpha_0\, m_1\sigma_1\alpha_1\, m_2\sigma_2\alpha_2 \cdots m_n\sigma_n\alpha_n \tag{1}$$

where

- process r does not occur in $\sigma_0 \cdots \sigma_n$,
- each m_i is a message (i.e., send/receive pair) between r and some r_j,
- each α_i is a sequence of local (pushdown) actions of r,
- each σ_i $(i < n)$ starts and ends with all stacks of processes r_j empty.

We obtain the previous reordering by scheduling the messages between r and the r_j as late as possible. That is, the run starts first with actions not involving r (subrun σ_0), plus some pushdown actions of r (subrun α_0). Then the first message m_1 between r and some r_j follows. All remaining actions are in the future of this message, and σ_1 is a run over the r_j (which may synchronize among themselves by communication), etc.

We check that the stacks of the r_j are always empty at the beginning/end of each σ_i by considering the subruns $\sigma_0\alpha_0 m_1\sigma_1$, $\sigma_0\alpha_0 m_1\sigma_1 m_2\alpha_2\sigma_2$, etc.: consider the first occurrence of some process q occurring e.g. in σ_1. Either this occurrence is the message m_1 between q and r (so $r \bullet\!\!-\!\!\circ q$), or it is a synchronization with some r_i, so $r_i \circ\!\!-\!\!\circ q$. In particular, q's stack must be empty at the end of σ_0.

The existence of an eager run as above can be checked by first pre-computing the control-state reachability for $\mathcal{T} \setminus r$, which corresponds to the first case previously considered in this proof (recall that we compute summaries, i.e., all pairs of global states that can be reached starting/ending with empty stacks and queues). Then the question for $\mathcal{T}$ reduces to the control-state reachability of a pushdown automaton (that of process p) of size exponential in $|\mathcal{P}|$, thus showing the claim.

We now get back to the situation where $\mathcal{T}$ contains some process $p \in \mathcal{P}$ with at least one channel of type $p \bullet\!\!-\!\!\circ *$ in $\mathcal{T}$, and at least one channel of type either $p \circ\!\!-\!\!\bullet *$, or $p \circ\!\!-\!\!\circ *$ in $\mathcal{T}$.

Consider an (eager) run over $\mathcal{T}$, starting (and possibly ending) with all stacks empty. Again we assume, for convenience, that all channels are empty at the end. The run can be reordered such that we obtain an order-equivalent run of the form $\rho_0\sigma_0\rho_1 \cdots \sigma_{n-1}\rho_n$, where:

- every subrun σ_i $(i < n)$ starts and ends with empty stacks for all processes $q \neq p$ occurring in σ_i,
- p's stack is continuously empty during ρ_i, and continuously non-empty during σ_i, for each i.

We need to explain why we may assume that the subruns σ_i start / end with empty stacks for each $q \neq p$. The reason is that we schedule the internal

push / pop actions of p that start / end a phase with non-empty p-stack such that push actions have lowest priority, and pop ones have the highest one. Such push / pop pairs on p delimit the subruns σ_i and Lemma 2 can be applied to $\rho_0\sigma_0\rho_1$, $(\rho_0\sigma_0\rho_1)\sigma_1\rho_2$, etc.

The existence of suitable subruns σ_i can be checked inductively: notice that channels of type $p \circ\!\!-\!\!\bullet *$ and $p \circ\!\!-\!\!\circ *$ can be removed from $\mathcal{T}$, since p's pushdown is non-empty during every σ_i, hence such channels are not used. More formally, we check for each (connected) subset $P \subseteq \mathcal{P}$ with $p \in P$, and each pair of starting/ending states $s = (s_q)_{q\in P}$, $s' = (s'_q)_{q\in P}$ whether there is a run of processes from P from s to s' in the modified architecture, starting/ending with empty stacks and queues.

Consider now a set P corresponding to processes occurring in some run σ_i. Notice first that every process in P can be reached from p via messages involving only P; symmetrically, every process in P can reach p via messages involving only P. For each such set P we can introduce new synchronization messages between processes in P such that we replace σ_i by a sequence S_i of new messages with the following properties: the first message in S_i is sent by p, the last message is received by p, and every process in P occurs among the receivers in S_i. Such a sequence S_i can be used to enforce that processes in P_i go (in a sort of meta-transition) from state $s = (s_q)_{q\in P}$ to state $s' = (s'_q)_{q\in P}$, thus replacing σ_i. We can enforce that the new messages occur only in form of sequences S_i, by encoding S_i with message contents. In order to avoid an exponential blow-up in the size of the RQCP we record the possible sequences S_i separately. Notice that using these sequences in the base step of the induction does not affect the EXPTIME upper bound.

We can now apply induction on the modified RQCP in order to check whether there is some run of the form $\rho' = \rho_0 S_0 \rho_1 S_1 \dots S_{n-1}\rho_n$. The induction is possible since we can transform the channels of type $p \bullet\!\!-\!\!\circ *$ into type $\circ\!\!-\!\!\circ$ (since p does not use its pushdown in ρ').

To summarize, the induction is done on two parameters: either we decrease the overall number of channels, or we change at least one channel of type $\bullet\!\!-\!\!\circ$ into type $\circ\!\!-\!\!\circ$. We first check the existence of runs σ_i inductively in exponential time, by computing reachability for every pair of global states and subset of processes (of which there are exponentially many). Then we modify the RQCP according to the previous calls and check inductively the existence of a *single* run ρ' (again, this is done for each pair of global states and set of processes).

Finally, let us comment on the lower bound: It is known (and probably folklore) that the following problem is EXPTIME-complete: checking the emptiness of the intersection of a pushdown with n finite automata. 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 [8], namely the reachability problem for pushdowns with checkpoints. Clearly, the intersection between a pushdown and n finite automata can be simulated on a topology $\mathcal{T} = \langle \mathcal{P}, \mathrm{Ch}\rangle$ with $\mathcal{P} = \{r, r_1, \dots, r_n\}$ and $r \bullet\!\!-\!\!\circ r_i$ for each i. □

3 From Mutex QCP to Eager QCP

The previous section showed how to decide the control state reachability for RQCP (and therewith finite QCP) on eager runs. Nevertheless, restricting the communication to eager runs seems rather strong at first glance. In the following, we will show how eagerness arises naturally on two practically relevant communication architectures: polytrees and cyclic architectures with mutex restriction. Further, we discuss the reduction of the control state reachability problem for (possibly infinite) mutex QCP to their underlying local transition systems, like e.g., Petri nets.

Any communication architecture $\mathcal{T}$ can be regarded as directed graph $\langle \mathcal{P}, \mathrm{Ch} \rangle$; let $\mathit{UCycle}(\mathcal{T})$ be the set of its undirected simple cycles. A cycle is *undirected* if we ignore the direction of the channels, and *simple* if it has no subcycle of smaller length.

Definition 6. *A configuration c of a* QCP $\mathcal{A}$ *is* mutex *with respect to a given architecture $\mathcal{T}$ if for every cycle α of $\mathit{UCycle}(\mathcal{T})$ at most one of the channels occurring in α is non-empty in c. A* QCP $\mathcal{A}$ *is called* mutex *with respect to a given architecture $\mathcal{T}$ if every $c \in \mathit{Reach}_{\mathcal{A}}$ is mutex.*

Before discussing mutex QCP in detail, we first recall two known results that are subsumed by our definition of mutex:

Remark 3. A special case of mutex QCP was considered in [16]: polyforest architectures over finite QCP, as well as (well-queueing) root-to-leaf directed forests for RQCP. Their decidability proof relied on the idea that, on any tree architecture, we can reorder runs such that first all actions of the root process are scheduled, and then, in breadth-first order, the actions of all others. Consequently, each run could be partitioned into a bounded number of contexts (bounded by $|\mathcal{P}|$) 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 RQCP— solely when its local stack is empty). Hence, the decidability problem reduced to the control state reachability for a bounded-phase multi-stack pushdown system, which is known to be decidable in doubly exponential time [15].

We will show in the following that mutex QCP are eager, and, consequently, apply the results of the previous section to obtain the decidability of control-state reachability via a direct proof. Moreover, recall that the complexity of the algorithm of the previous section is EXPTIME, so one exponential less than the positive results of [16].

Remark 4. Runs over an architecture of two finite processes connected by two channels where each reachable configuration is mutex are known as "half-duplex communication". For these, it is known how to decide the (general) reachability question by computing a recognizable description of the channel contents [7]. Quasi-stable systems are a semantic ad-hoc extension of this idea to larger, cyclic architectures of finite QCP [7], which is subsumed by our mutex condition.

Proposition 2. *Given a* QCP $\mathcal{A}$ *that is mutex with respect to a given architecture* $\mathcal{T}$*, each of its runs has an order-equivalent eager run.*

Proof. In the following, we will differentiate the occurrences of one action by referring to them as events. For each process $p \in \mathcal{P}$ occurring in ρ there is a first, initial event w.r.t. ρ which will be abbreviated f_p; further, each receive event has a preceding matching send in ρ. All events that belong to the same process are totally ordered. Consequently, we define the partial order "before" (denoted by $<$) between events as the transitive closure of the previous two cases. In the following we will focus only on matched communication events, by considering send actions on channels that already entered their growing phase as internal actions.

We will inductively define a reordering for a run ρ whose first configuration $c_0 = \langle s, w\rangle$ fulfills $w_{p,q} = \varepsilon$ if there exists a receive event $q?p$ in ρ.

Assume we have a run ρ from c to c' and c fulfills the previous property. First, we pick an initial send event f_p on a channel from process p to q which either (i) has no matching receive in ρ (i.e., it is the first send in a growing phase), or (ii) its matching receive on process q is also initial, hence, equal to f_q. In case (i) we schedule f_p first and reorder inductively the remaining run starting from c'' with $c \xrightarrow{f_p} c''$ towards c'. For case (ii) we first schedule f_p and then f_q before we inductively reorder the remaining run starting from c'' with $c \xrightarrow{f_p} c''' \xrightarrow{f_q} c''$. Note than in both cases c'' satisfies our requirement.

Next we have to show that it is always possible to apply cases (i) or (ii) above, to any run ρ of our mutex QCP. The general idea is as follows. Suppose that we pick an initial send event f_{p_0} on process p_0 that has a matching receive e_{p_0} on process p_1, but e_{p_0} is not initial. Then we can restart our search for an initial event from p_1 on. If f_{p_1} is a send, then we proceed as for p_0; else, if f_{p_1} is a receive, we continue with its matching send on process p_2. As we only have finitely many processes, an unsuccessful, repeated search leads to a cycle in $UCycle(\mathcal{T})$: $\langle p_0, p_1, p_2, \ldots, p_i, p_{i+1}, \ldots, p_k, p_{k+1}\rangle$ with $p_{k+1} = p_i$ and all p_i $(i \leq k)$ pairwise different. Moreover, we show the existence of at least two non-empty channels on this cycle.

In the following, we slightly abuse notation by writing f_i and instead of f_{p_i} for the initial event of process p_i. We focus on the initial events $f_i, \ldots, f_k$ and their matching events $e_i, \ldots, e_k$ on processes $p_{i+1}, \ldots, p_k, p_{k+1} = p_i$. Obviously, $f_{j+1} < e_j$ for all $i \leq j \leq k$, since both f_{j+1}, e_j occur on process p_{j+1} and f_{j+1} is initial. We distinguish the following cases:

(a) all initial events $f_i, \ldots, f_k$ are receives (cf. Fig. 3(a)), then $e_{j+1} < f_{j+1} < e_j$ for all $i \leq j \leq k$; hence, we arrive at the contradiction $e_i < e_i$;
(b) there are at least two sends among the initial actions, for example f_j and f_l with $i \leq j < l \leq k$; consequently, $c \xrightarrow{f_j} c'' \xrightarrow{f_l} c'''$ leads to a configuration which is not mutex (cf. Fig. 3(b)) and, hence another contradiction;
(c) there is only one send event among the initial events of the cycle, say f_i. Then, $f_i = f_{k+1}$ is before e_k, and e_k is a send event, too (since all f_j with $i < j \leq k$ are receives). It is easy to see that $e_k < f_k < e_{k-1} < \cdots < f_{i+1} < e_i$. In particular, all events on each of $p_{i+1}, \ldots, p_k$ are after e_k.

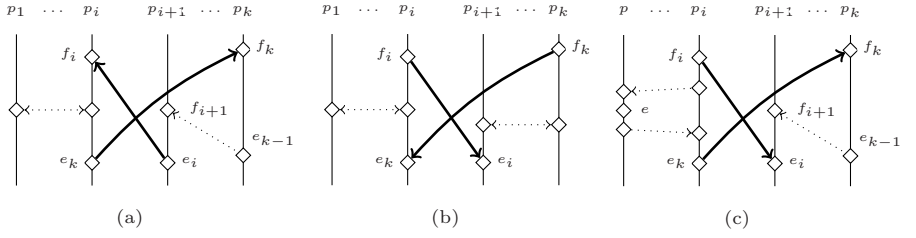

Fig. 3. Cycles in the proof of Lemma 2 ($\diamond$: events, arrows: messages)

Consider now an event e with $f_i < e < e_k$. Notice that e cannot belong to any of $p_{i+1}, \ldots, p_k$, as all events on these processes must take place after e_k (cf. Fig. 3(c) for e on a process p that does not participate in the cycle); consequently, the configuration obtained after executing all events before e_k is not mutex, as the channels (p_i, p_{i+1}) and (p_i, p_k) are both non-empty. □

Corollary 1. *If a* QCP $\mathcal{A}$ *is mutex with respect to a given architecture* $\mathcal{T}$ *then* $\mathcal{A}$ *is eager.*

Proposition 3. *The control state reachability for finite* QCP *that are mutex with respect to the given architecture is* PSPACE*-complete.*

Remark 5. Control-state reachability is decidable for particular *infinite-state* mutex QCP. For example, if all local transition systems are Petri nets, then the control state reachability question reduces to a Petri net reachability question which is known to be decidable [17,13].

The mutex property can be checked effectively for QCP.

Proposition 4. *It is* PSPACE*-complete to check whether a finite* QCP *is mutex with respect to a given architecture.*

4 Bounded Phase Reachability

Besides their proven practical relevance in the verification of concurrent programs [19], bounded-context reachability allows to attack the (control-state) reachability problem on QCP from a different angle. In this section, we neither restrict the communication architecture, nor constrain the runs to be eager (or mutex). The price we pay is a (strong) restriction on the form of the possible runs, by fixing the number of contexts. We present in this section a construction that subsumes the 2EXPTIME algorithm for bounded-context reachability for well-queueing RQCP described in [16]. 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.

We define a *phase* in an RQCP run ρ over an oriented architecture as a (contiguous) subrun of ρ consisting of actions on a unique process, say p, that are subject to one of the two restrictions below (defining M-phases and N-phases, resp.):

(M) Receives are from a unique process, say q, with $(q,p) \in \mathrm{Ch}$ of type $p \circ\!\!-\!\!\bullet\, q$ or $p \circ\!\!-\!\!\circ\, q$. Sends go to (arbitrarily many) processes r with $(p,r) \in \mathrm{Ch}$ of type $p \bullet\!\!-\!\!\circ\, r$.

(N) Sends go to a unique process, say q, with $(p,q) \in \mathrm{Ch}$ of type $p \circ\!\!-\!\!\bullet\, q$ or $p \circ\!\!-\!\!\circ\, q$. Receives come from (arbitrarily many) processes r with $(r,p) \in \mathrm{Ch}$ of type $p \bullet\!\!-\!\!\circ\, r$.

Notice that M-phases are precisely the phases (contexts) used in [16], whereas N-phases represent the dual notion. We will refer below to the channel (q,p) (M-phase) resp. (p,q) (N-phase) as the *special* channel of the phase. A run ρ of an RQCP is *K-bounded*, if we can write $\rho = \rho_1 \cdots \rho_K$, with each ρ_i a phase as above.

Theorem 2. *Given an* RQCP $\mathcal{A}$ *and an integer K, the K-bounded control state reachability problem for $\mathcal{A}$ can be solved in time doubly exponential in the number K of phases (but polynomial in the size of $\mathcal{A}$).*

Sketch of proof. The basic idea is to decrease the number of phases in a particular order: M-phases are deleted for right to left (a sort of pre-computation), whereas N-phases are deleted from left to right (post-computation). Deleting a phase i belonging to some process p amounts to synchronizing a finite automaton obtained from $\mathcal{A}_p$ and phase i with the current automaton $\mathcal{A}_q$ of the process communicating with p on the special channel of phase i. We obtain this finite automaton by exploiting the fact that p's stack is empty while communicating in phase i on the special channel. In addition we must ensure that the phase i that we delete starts and ends with empty p-stack. Finally, for a single phase we need to solve a reachability problem for a single pushdown with doubly exponentially many states. The details can be found in the long version of this paper.

Remark 6. Adapting proof ideas from [15,1], we can show that the complexity bound in Theorem 2 is tight.

5 Conclusion

Applications. QCP 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 RQCP, 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.

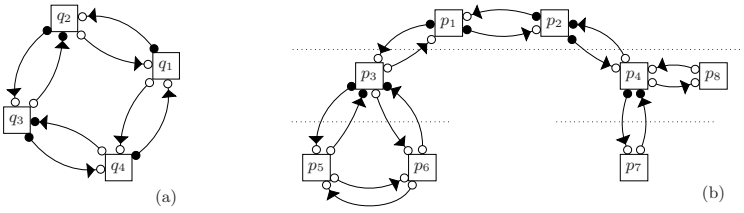

Fig. 4. Non-confluent architectures: (a) ring, and (b) hierarchical master-worker setting — tree-like architecture with •—○-channels between master and workers (distribute tasks and collect results while in computation, send result to own master when computation finished, i.e., stack empty) as well as ○—○-channels between workers of the same master; note the •—○-cycle on the top level

Fig. 4 (b) shows an example for non-confluent architectures that are on the rise with the current focus on Grid computing. The topology depicts a hierarchical overlay network as implemented, for example, in a master-worker protocols. Here, mutual communication is restricted with respect to the hierarchy (in general: •—○ top-down and ○—○ between siblings). Notice also the use of the dual notion to well-queueing, when sending information from lower to higher levels.

Proposition 2 allows for further applications, since it does not assume that the QCP 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, WSTS, or multi-set pushdown systems [20].

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

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 QCP, like boundedness or liveness, will be investigated in future work.

References

1. Atig, M.F., Bouajjani, A., Habermehl, P.: Emptiness of multi-pushdown automata is 2ETIME-complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
2. Atig, M.F., Bouajjani, A., Touili, T.: On the reachability analysis of acyclic networks of pushdown systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 356–371. Springer, Heidelberg (2008)
3. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. 127(2), 91–101 (1996)

4. Bouajjani, A., Esparza, J., Schwoon, S., Strejcek, J.: Reachability analysis of multithreaded software with asynchronous communication. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 348–359. Springer, Heidelberg (2005)
5. Bouajjani, A., Müller-Olm, M., Touili, T.: Regular symbolic analysis of dynamic networks of pushdown systems. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 473–487. Springer, Heidelberg (2005)
6. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. of the ACM 30(2), 323–342 (1983)
7. Cécé, G., Finkel, A.: Verification of programs with half-duplex communication. Inf. Comput. 202(2), 166–190 (2005)
8. Esparza, J., Kucera, A., Schwoon, S.: Model checking LTL with regular valuations for pushdown systems. Inf. Comput. 186(2), 355–376 (2003)
9. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1-2), 63–92 (2001)
10. Genest, B., Kuske, D., Muscholl, A.: A Kleene theorem and model checking algorithms for existentially bounded communicating automata. Inf. Comput. 204(6), 920–956 (2006)
11. Genest, B., Kuske, D., Muscholl, A.: On communicating automata with bounded channels. Fundamenta Informaticae 80, 147–167 (2007)
12. Jhala, R., Majumdar, R.: Interprocedural analysis of asynchronous programs. In: POPL 2007, pp. 339–350. ACM, New York (2007)
13. Rao Kosaraju, S.: Decidability of reachability in vector addition systems. In: STOC 1982, pp. 267–281. ACM, New York (1982)
14. Lohrey, M., Muscholl, A.: Bounded MSC communication. Inf. Comput. 189(2), 160–181 (2004)
15. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS 2007, pp. 161–170. IEEE, Los Alamitos (2007)
16. La Torre, S., Madhusudan, P., Parlato, G.: Context-bounded analysis of concurrent queue systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
17. Mayr, E.W.: An algorithm for the general Petri net reachability problem. SIAM J. Comput. 13(3), 441–460 (1984)
18. Papadimitriou, C.: Computational Complexity. Addison Wesley, Reading (1994)
19. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
20. Sen, K., Viswanathan, M.: Model checking multithreaded programs with synchronous atomic methods. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 300–314. Springer, Heidelberg (2006)