

Reachability Analysis of Communicating Pushdown Systems ^{*}

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

LaBRI, Université de 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 [LTMP08], 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 [LTMP08] 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 [LTMP08]. 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 [BZ83]. 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.

^{*} This work was partly supported by ANR project AVERISS (ANR-06-SETIN-001).

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. [LTMP08], 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 [LTMP08] 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 [LTMP07].

Our contribution. We extend the work of La Torre et al. [LTMP08] 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 [LM04]. Eagerly communicating finite-state machines are a well-studied model, enjoying good expressiveness and decidability properties [GKM06]. 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 [LTMP08], 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 [CF05].

La Torre et al. [LTMP08] 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 [QR05]. 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 [LTMP07]. 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 [BESS05, BMOT05] or bags in [SV06, JM07]. The paper [BESS05] showed that bounded-context reachability can be solved in exponential time, whereas [SV06] provided an exponential space lower bound for reachability (without context bounds). More recently, synchronization in the form of state observation was considered in [ABT08]. The latter model was shown to be decidable only for acyclic architectures, and is strongly related to lossy systems [AJ96, FS01].

1 Recursive Queueing Concurrent Processes

Throughout the paper, we shall 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 [?] or textbooks like [Pap94].

Given a set P and an P -indexed family of sets $(S^p)_{p \in P}$, we shall 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 value of \mathbf{s} at index p . 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*. For any two words $w, w' \in \Sigma^*$, we write $w \cdot w'$ for their *concatenation*.

A *labeled transition system* is a quadruple $\mathfrak{S} = \langle S, s_\bullet, A, \rightarrow \rangle$ where S is a set of *states*, $s_\bullet \in S$ is the *initial state*, A is an *action* alphabet, and \rightarrow is a subset of $S \times A \times S$ called the *labeled transition relation*. For simplicity, we shall usually write $s \xrightarrow{a} s'$ in place of $(s, a, s') \in \rightarrow$.

1.1 Queueing Concurrent Processes

In this paper, we consider classes of QCP that are defined by the specification of the authorized interactions between processes and channels. To this end, we introduce the notion of topology, which is simply a directed graph whose vertices are processes and whose edges represent channels.

Definition 1.1. A network topology \mathcal{T} (or topology for short) is a tuple $\langle P, C, \text{src}, \text{tgt} \rangle$ where P is a finite set of processes, C is a finite set of point-to-point channels equipped with two functions $\text{src}, \text{tgt} : C \rightarrow P$ that map every channel $c \in C$ to a source process $\text{src}(c)$ and a target process $\text{tgt}(c)$, respectively.

For each channel $c \in C$, we write \xrightarrow{c} the binary relation on the set of processes P defined by $p \xrightarrow{c} q$ if $p = \text{src}(c)$ and $q = \text{tgt}(c)$. We also introduce the undirected binary relation \xleftrightarrow{c} defined by $p \xleftrightarrow{c} q$ if $p \xrightarrow{c} q$ or $q \xrightarrow{c} p$.

Given a message alphabet M , we denote by $\text{Com}^p(\mathcal{T}, M)$ the set of *possible communication actions* of a process $p \in P$, defined by $\text{Com}^p(\mathcal{T}, M) = \{c!m \mid c \in C, \text{src}(c) = p, m \in M\} \cup \{c?m \mid c \in C, \text{tgt}(c) = p, m \in M\}$. As usual, $c!m$ denotes the send of message m into channel c , whereas $c?m$ denotes a receive of message m from channel c .

Definition 1.2. A system of queueing concurrent processes (QCP) \mathcal{Q} is a triple $\langle \mathcal{T}, M, (\mathbb{S}^p)_{p \in P} \rangle$ where \mathcal{T} is a topology, M is a finite message alphabet, and, for each process $p \in P$, $\mathbb{S}^p = \langle S^p, s_\bullet^p, A^p, \rightarrow_p \rangle$ is a labeled transition system such that the intersection $A_{\text{com}}^p = A^p \cap (C \times \{!, ?\} \times M)$ is a subset of $\text{Com}^p(\mathcal{T}, M)$.

Actions in A_{com}^p are called *communication actions* of p . We also introduce the set $A_{\text{loc}}^p = A^p \setminus A_{\text{com}}^p$ of *local actions* of p . States $s^p \in S^p$ are called *local states* of p . We write $\mathbf{S} = \prod_{p \in P} S^p$ the set of *global states*, and the *initial global state* is $\mathbf{s}_\bullet = (s_\bullet^p)_{p \in P}$. Note that the sets S^p , and hence \mathbf{S} , may be infinite. Indeed, the local transition systems \mathbb{S}^p could be, for instance, finite automata, counter automata (including Petri nets), or pushdown automata. When \mathbf{S} is finite, \mathcal{Q} is called a *finite* QCP. As usual, we give the (operational) semantics of QCP in terms of labeled infinite-state transition systems.

Definition 1.3. The semantics of a QCP $\mathcal{Q} = \langle \mathcal{T}, M, (\mathbb{S}^p)_{p \in P} \rangle$ is the labeled transition system $\llbracket \mathcal{Q} \rrbracket = \langle X, x_\bullet, \Sigma, \rightarrow \rangle$ where $X = \mathbf{S} \times (M^*)^C$, $x_\bullet = (\mathbf{s}_\bullet, (\varepsilon)_{c \in C})$, $\Sigma = \{(p, a) \mid p \in P, a \in A^p\}$, and \rightarrow is the set of triples $((\mathbf{s}_1, \mathbf{w}_1), (p, a), (\mathbf{s}_2, \mathbf{w}_2))$ in $X \times \Sigma \times X$ satisfying, for all $c \in C$ and $m \in M$, the four following conditions:

- (i) $s_1^p \xrightarrow{a}_p s_2^p$ and $s_1^q = s_2^q$ for all $q \in P$ with $q \neq p$,
- (ii) if $a \in A_{\text{loc}}^p$ then $\mathbf{w}_1 = \mathbf{w}_2$,
- (iii) if $a = c!m$ then $w_2^c = w_1^c \cdot m$ and $w_2^d = w_1^d$ for all $d \in C$ with $d \neq c$,
- (iv) if $a = c?m$ then $m \cdot w_2^c = w_1^c$ and $w_2^d = w_1^d$ for all $d \in C$ with $d \neq c$.

To prevent confusion, elements of X will be called *configurations*. Note that our definition imposes that all channels are initially empty.

A *run* ρ in the labeled transition system $\llbracket \mathcal{Q} \rrbracket$ is a finite alternating sequence $(x_0, \sigma_1, x_1, \dots, \sigma_n, x_n)$ of configurations $x_i \in X$ and actions $\sigma_i \in \Sigma$ satisfying $x_{i-1} \xrightarrow{\sigma_i} x_i$ for all i with $1 \leq i \leq n$. We say that ρ is a run *from* x_0 *to* x_n . The *length* of ρ is n . Note that a run of zero length consists in a single configuration. 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 $\rho = (x_0, \sigma_1, x_1, \sigma_2, x_2)$ and $\rho' = (x'_0, \sigma'_1, x'_1, \sigma'_2, x'_2)$ satisfy $\rho \sim \rho'$ if $x_0 = x'_0$, $x_2 = x'_2$, $(\sigma_1, \sigma_2) = (\sigma'_2, \sigma'_1)$ and σ_1, σ_2 are actions on two distinct processes. Informally, ρ, ρ' are order-equivalent if they can be transformed one into the other by iteratively commuting adjacent actions σ and τ that (i) do *not* belong to the same process, and (ii) do *not* form a “matching” send/receive pair.

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

The *reachability question* asks, for a given QCP \mathcal{Q} and a given configuration $x \in X$, whether $Reach(\mathcal{Q})$ contains x . The *control-state reachability question* asks, for a given QCP \mathcal{Q} and a given global state $\mathbf{s} \in \mathbf{S}$, whether $Reach(\mathcal{Q})$ intersects $\{\mathbf{s}\} \times (M^*)^C$. Both questions are undecidable for finite QCP, even if we restrict the topology to a single process with one (cyclic) channel, or to two processes connected by two channels [BZ83].

Definition 1.4. *A run $\rho = (x_0, \sigma_1, x_1, \dots, \sigma_n, x_n)$ starting from an empty-channel configuration $x_0 = (\mathbf{s}_0, (\varepsilon)_{c \in C})$ is eager if the following holds: for all i with $1 < i \leq n$, if σ_i is a receive on a channel $c \in C$ then σ_{i-1} is a send on c .*

Along an eager run, each send action is either immediately followed by a “matching” receive action, or is never received. In the latter case, all further send actions in the channel are not received, and we say that the channel is in its “growing phase”.

We introduce eager variants of the reachability notions presented previously. A configuration $x \in X$ is *eager-reachable* in \mathcal{Q} if there exists an eager run from the initial configuration x_\bullet to the configuration x . The *eager-reachability set* of \mathcal{Q} is the set $Reach_{eag}(\mathcal{Q})$ of eager-reachable configurations in \mathcal{Q} . Observe that $Reach_{eag}(\mathcal{Q}) \subseteq Reach(\mathcal{Q})$. We say that a QCP \mathcal{Q} is *eager* when $Reach_{eag}(\mathcal{Q}) = Reach(\mathcal{Q})$. 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 globally 1-bounded runs, and similarly eager QCP are close to existentially globally 1-bounded communicating finite-state machines [LM04, GKM07].

The *eager-reachability question* asks, for a given QCP \mathcal{Q} and a given configuration $x \in X$, whether $Reach_{eag}(\mathcal{Q})$ contains x . The *control-state eager-reachability question* asks, for a given QCP \mathcal{Q} and a given global state $\mathbf{s} \in \mathbf{S}$, whether $Reach_{eag}(\mathcal{Q})$ intersects $\{\mathbf{s}\} \times (M^*)^C$. It is readily seen that both questions are decidable for finite QCP.

Complexity of eager-reachability for finite QCP?

1.2 Recursive Queueing Concurrent Processes

In the following we introduce RQCP together with a symmetric version of the “well-queueing” property from [LTMP08].

Informally, recursive QCP are ordinary QCP where each local transition system is given by a pushdown process.

A well-queueing RQCP in [LTMP08] 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 1.5. A recursive topology is a triple $\langle \mathcal{T}, \circ, \bullet \rangle$ where \mathcal{T} is a topology and (\circ, \bullet) is a partition of $\{(p, c) \in P \times C \mid p = \text{src}(c) \vee p = \text{tgt}(c)\}$.

Given a process $p \in P$ and a channel $c \in C$ with $p = \text{src}(c)$ or $p = \text{tgt}(c)$, p is called *restricted* on c if $(p, c) \in \circ$, and p is called *unrestricted* otherwise. For each $*, \star \in \{\circ, \bullet\}$ we write by $* \xrightarrow{c} \star$ the binary relation on the set of processes P defined by $p \xrightarrow{c} \star q$ if $p = \text{src}(c)$, $q = \text{tgt}(c)$, $(p, c) \in *$ and $(q, c) \in \star$. We also introduce the undirected binary relation $* \overset{c}{\leftrightarrow} \star$ defined by $p \overset{c}{\leftrightarrow} \star q$ if $p \xrightarrow{c} \star q$ or $q \xrightarrow{c} \star p$.

Definition 1.6. A pushdown process is a quintuple $\mathcal{D} = \langle Z, z_\bullet, A, \Gamma, \Delta \rangle$ where Z is a finite set of control states, $z_\bullet \in Z$ is the initial control state, A is an action alphabet, Γ is a stack alphabet, and $\Delta \subseteq Z \times A \times Z$ is a set of transition rules.

We let $A_{\text{stack}} = A \cap (\{\text{push}, \text{pop}\} \times \Gamma)$ denote the set of *stack actions* of \mathcal{D} . Naturally, A may also contain communication actions. The intention behind recursive topologies is to specify conditions on the stack for communication actions to be feasible: if a pushdown process p is restricted on a channel c , then communication actions of p on c are feasible only if p has an empty stack. To this end, we define the operational semantics of pushdown processes with respect to a set of actions that require an empty stack.

Definition 1.7. The semantics of a pushdown process $\mathcal{D} = \langle Z, z_\bullet, A, \Gamma, \Delta \rangle$ w.r.t. to a subset $A_\varepsilon \subseteq (A \setminus A_{\text{stack}})$ is the labeled transition system $\llbracket \mathcal{D} \mid A_\varepsilon \rrbracket = \langle S, s_\bullet, A, \rightarrow \rangle$ where $S = Z \times \Gamma^*$, $s_\bullet = (z_\bullet, \varepsilon)$, and \rightarrow is the set of triples $((z_1, u_1), a, (z_2, u_2))$ in $S \times A \times S$ satisfying, for all $\gamma \in \Gamma$, the five following conditions:

- (i) $(z_1, a, z_2) \in \Delta$,
- (ii) if $a \in A_\varepsilon$ then $u_1 = u_2 = \varepsilon$,
- (iii) if $a \notin A_{\text{stack}}$ then $u_1 = u_2$,
- (iv) if $a = \text{push } \gamma$ then $u_2 = u_1 \cdot \gamma$,
- (v) if $a = \text{pop } \gamma$ then $u_1 = u_2 \cdot \gamma$.

Definition 1.8. A recursive QCP (RQCP) \mathcal{R} is a tuple $\langle \mathcal{T}, \circ, \bullet, M, \Gamma, (\mathcal{D}^p)_{p \in P} \rangle$ where $\langle \mathcal{T}, \circ, \bullet \rangle$ is a recursive topology, M is a message alphabet, Γ is a stack alphabet, and, for each process $p \in P$, $\mathcal{D}^p = \langle Z^p, z_\bullet^p, A^p, \Gamma, \Delta^p \rangle$ is a pushdown process such that the intersection $A_{\text{com}}^p = A^p \cap (C \times \{!, ?\} \times M)$ is a subset of $\text{Com}^p(\mathcal{T}, M)$.

We associate to \mathcal{R} the QCP $\mathcal{Q}(\mathcal{R})$ defined by $\mathcal{Q}(\mathcal{R}) = \langle \mathcal{T}, M, (\llbracket \mathcal{D}^p \mid A_\varepsilon^p \rrbracket)_{p \in P} \rangle$ where, for each $p \in P$, the set A_ε^p of actions requiring an empty stack is defined by $A_\varepsilon^p = \{c!m \in A_{\text{com}}^p \mid (p, c) \in \circ\} \cup \{c?m \in A_{\text{com}}^p \mid (p, c) \in \circ\}$. The *semantics* of \mathcal{R} , written $\llbracket \mathcal{R} \rrbracket$, is the labeled transition system $\llbracket \mathcal{Q}(\mathcal{R}) \rrbracket$.

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 2.1. A recursive topology $\langle \mathcal{T}, \circ, \bullet \rangle$ is called confluent if there exists an alternating sequence $(r_0, c_1, r_1, \dots, c_n, r_n)$, of processes in P and channels in C , with $n \geq 1$, satisfying the following conditions:

1. the processes r_0, \dots, r_n are distinct,
2. it holds that $r_{i-1} \xleftrightarrow{c_i} r_i$ for all i with $1 \leq i \leq n$,
3. r_0 is unrestricted on c_1 and r_n is unrestricted on c_n .

Definition 2.2. A recursive architecture $\langle \mathcal{T}, C_s, C_r \rangle$ is called confluent if there exist distinct processes $p = r_0, r_1, \dots, r_k, r_{k+1} = q$ ($k \geq 1$) in P , satisfying $p \bullet \circ r_1 \cdots \circ r_k \circ \bullet q$, i.e. $p \bullet \circ r_1, r_i \circ r_{i+1}$ for every $i \in \{1, \dots, k-1\}$, and $r_k \circ \bullet q$.

Theorem 2.3. A recursive architecture 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 2.3.

Lemma 2.4. 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 . \square

Proposition 2.5. *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 $\mathcal{T} = \langle \mathcal{P}, \text{Ch} \rangle$, 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) $\mathcal{T} = \langle \mathcal{P} = \langle r, r_1, \dots, r_k \rangle, \text{Ch} \rangle$, where:

- 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:

$$\rho = \sigma_0 \alpha_0 m_1 \sigma_1 \alpha_1 m_2 \sigma_2 \alpha_2 \cdots m_n \sigma_n \alpha_n \quad (1)$$

where

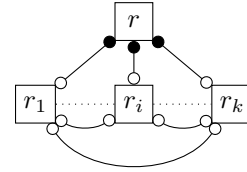


Fig. 1. Case (ii).

- 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 $\sigma_0\alpha_0m_1\sigma_1$, $\sigma_0\alpha_0m_1\sigma_1m_2\alpha_2\sigma_2$, 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 2.4 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 $p \bullet \dashv \circ *$ into type $\circ \dashv \circ$ (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 $\bullet \dashv \circ$ into type $\circ \dashv \circ$. 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 [EKS03], namely the reachability problem for pushdowns with checkpoints. Clearly, the intersection between a pushdown and n finite automata can be simulated on a topology $\mathcal{T} = \langle \mathcal{P}, \text{Ch} \rangle$ with $\mathcal{P} = \{r, r_1, \dots, r_n\}$ and $r \bullet \dashv \circ r_i$ for each i . \square

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 $\langle \mathcal{P}, \text{Ch} \rangle$; let $UCycle(\mathcal{T})$ 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 3.1. *A configuration c of a QCP \mathcal{A} is mutex with respect to a given architecture \mathcal{T} if for every cycle α of $UCycle(\mathcal{T})$ 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 $c \in Reach_{\mathcal{A}}$ is mutex.*

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

Remark 3.2. A special case of mutex QCP was considered in [LTMP08]: 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 [LTMP07].

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 [LTMP08].

Remark 3.3. 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 [CF05]. Quasi-stable systems are a semantic ad-hoc extension of this idea to larger, cyclic architectures of finite QCP [CF05], which is subsumed by our mutex condition.

Proposition 3.4. *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 that 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 $UCycle(\mathcal{T})$: $\langle p_0, p_1, p_2, \dots, p_i, p_{i+1}, \dots, p_k, p_{k+1} \rangle$ 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, \dots, f_k and their matching events e_i, \dots, e_k on processes $p_{i+1}, \dots, p_k, p_{k+1} = p_i$. 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, \dots, f_k are receives (cf. Fig. 2(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. 2(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 $e_k < f_k < e_{k-1} < \dots < f_{i+1} < e_i$. In particular, all events on each of p_{i+1}, \dots, p_k are after e_k .

Consider now an event e with $f_i < e < e_k$. Notice that e cannot belong to any of p_{i+1}, \dots, p_k , as all events on these processes must take place after e_k (cf. Fig. 2(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. \square

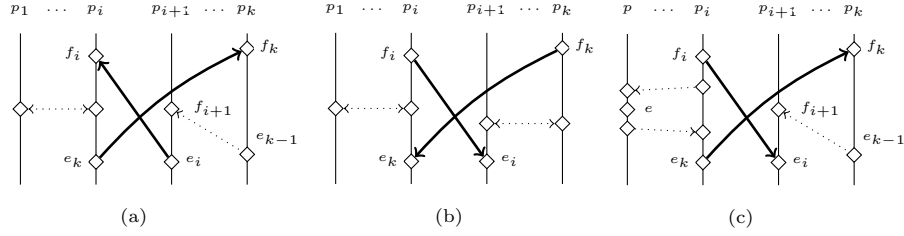


Fig. 2. Cycles in the proof of Lemma 3.4 (\diamond : events, arrows: messages).

Corollary 3.5. *If a QCP \mathcal{A} is mutex with respect to a given architecture \mathcal{T} then \mathcal{A} is eager.*

Proposition 3.6. *The control state reachability for finite QCP that are mutex with respect to the given architecture is PSPACE-complete.*

Remark 3.7. 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 [May84, Kos82].

The mutex property can be checked effectively for QCP.

Proposition 3.8. *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 [QR05], bounded-context reachability allows to attack the (control-state) reachability problem on QCP from a different angle. In this section, we neither restrict the recursive topology, 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 2-EXPTIME algorithm for bounded-context reachability for well-queueing RQCP described in [LTMP08]. 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* ϕ in an RQCP is a triple $\langle p, C_?, C_! \rangle$ where $p \in P$ is a process, $C_?, C_!$ are disjoint subsets of C such that $C_? \subseteq \{c \in C \mid tgt(c) = p\}$ and $C_! \subseteq \{c \in C \mid src(c) = p\}$. A ϕ -*restricted* run is a run ρ over actions exclusively on the process p such that communication actions are in the set $\{c!m \mid c \in C_!, m \in M\} \cup \{c?m \mid c \in C_?, m \in M\}$. In the sequel phases are subject to one of the two restrictions below (defining M -phases and N -phases, resp.):

- (M) The set of receiving channels $C_?$ contains at most one channel c and p is restricted on c . Sending channels $d \in C_!$ satisfies the restriction $\bullet \xrightarrow{d} \circ$.

- (N) The set of sending channels $C_!$ contains at most one channel c and p is restricted on c . Receiving channels $d \in C_?$ satisfies the restriction $\circ \xrightarrow{d} \bullet$.

Notice that M -phases are precisely the phases (contexts) used in [LTMP08], whereas N -phases represent the dual notion. A run ρ of an RQCP is k -bounded, if we can write $\rho = \rho_1 \cdots \rho_k$, where each ρ_j is a ϕ_j -restricted run for a $\{M, N\}$ -phase ϕ_j . This problem is solved by introducing the reachability problem for *phased RQCP*.

Let us consider a RQCP $\langle \mathcal{T}, \circ, \bullet, M, \Gamma, (\mathcal{D}^p)_{p \in P} \rangle$. We say that a finite sequence $(z_0^{p_j}, z_f^{p_j})_{1 \leq j \leq k}$ of *initial-phase* control-states $z_0^{p_j} \in Z^{p_j}$ and *final-phase* control-states $z_f^{p_j} \in Z^{p_j}$ is *reachable* from a sequence $(\phi_j)_{1 \leq j \leq k}$ of $\{M, N\}$ -phases with $\phi_j = (p_j, C_?^j, C_!^j)$ if there exists an initial run ρ in the RQCP RQCP that reaches a configuration with empty-stack and empty-queue and such that ρ can be decomposed into $\rho = \rho_1 \cdots \rho_j$ such that ρ_j is a ϕ_j -restricted run from the control state $z_0^{p_j}$ to the control-state $z_f^{p_j}$. This problem is called the *control-state phase-restricted reachability problem*. We solved this problem by induction over the integer $\sum_{j=1}^k |C_?^j| + |C_!^j|$ called the number of *channels* of $(\phi_j)_{1 \leq j \leq k}$.

Proposition 4.1. *Induction*

Proof. Let us denote by Z^j the set of control states in Z^{p_j} reachable from $z_0^{p_j}$ and coreachable from $z_f^{p_j}$. W.l.o.g by considering copies of the pushdown processes we can assume that $z_0^{p_j}$ is not reachable from a control-state in $Z^j \setminus \{z_0^{p_j}\}$ and $z_f^{p_j}$ is not coreachable from a control-state in $Z^j \setminus \{z_f^{p_j}\}$. Let us first assume that there exists a last M -phase ϕ_j such that $C_?^j \cup C_!^j$ is non empty. We denote by p the process working at the j th phase, i.e. $p = p_j$.

Let us assume that $C_!^j$ is non-empty. Since messages sent into channels in $C_!^j$ will never be received (a N -phase cannot receive a message from such a channel) and we try to find out a run such that the channels are empty in last configuration, we deduce that no message are sent in the j th phase. In particular we can replace ϕ_j by $(p_j, C_?^j, \emptyset)$ and the induction is proved.

So, we can assume that $C_!^j$ is empty. As $C_?^j \cup C_!^j$ is non empty and ϕ_j is a M -phase we deduce that $C_?^j$ contains exactly one channel, say c . We introduce the process $q = \text{src}(c)$. Observe that if there does not exist a phase ϕ_i with $1 \leq i < j$ such that $p_i = q$ then no message can be received from c since the channel is empty. In this case we can replace $C_?^j$ by the emptyset and the induction step is proved.

So we can assume that there exists a greatest i with $1 \leq i < j$ such that $p_i = q$ and such that $c \in C_?^i$. Since the j th phase is the last phase that receive messages from c we deduce that every message sent during the i th phase are received by the j th phase. We are going to simulate the j -th phase in the i th phase such that it will become possible to remove the channel c from the set $C_?^i$. Let us fix two control-states $z, z' \in Z^j$. Intuitively (z, ϵ) is the configuration just before the first action that read a message from c during the j th phase that have

been produced by the i th phase, and (z', ϵ) is the configuration just after the last action that read a message from c during the j th phase. We observe that when the stack of the process p is non-empty during the j th phase, messages cannot be received from c . Thus the language of words that can be received in the j th phase from the state (z, ϵ) to the state (z', ϵ) is a regular language. In fact it can even be recognized by a non-deterministic automaton where Z^j is the set of states that is computable in quadratic time. By “synchronizing” the i th part of \mathcal{D}^q with an automaton recognizing this regular language, we deduce a pushdown process $\mathcal{D}_{z, z'}^q$ equals to \mathcal{D}^q except for the part corresponding to Z^i such that communication actions $c!m$ have been replaced by local actions, and such that there exists in $\mathcal{D}_{z, z'}^q$ a path from a state (z_0^i, u_0^i) to a state (z_f^i, u_f^i) if and only if there exists in \mathcal{D}^q a path from a state (z_0^i, u_0^i) to a state (z_f^i, u_f^i) that sends a word of messages that corresponds to the word of messages that can be received in \mathcal{D}^p by a path from the state (z, ϵ) to (z', ϵ) . Let us consider the pushdown process $\mathcal{D}_{z, z'}^p$ obtained from \mathcal{D}^p by adding an extra copy such that every receive actions $c!m$ for some $m \in M$ have been removed in the j th part, and such that we have a transition from z to a copy of z' that can be executed if the stack of p is empty. Note that our model does not allow this kind of actions. However, by adding an extra fresh symbol $\#$ on the stack, we can simulate such a transition with the concatenation $pop \# \cdot push \#$. [TODO]

□

Given a sequence $(\phi_j)_{1 \leq j \leq k}$ of $\{M, N\}$ -phases $\phi_j = (C_2^j, p_j, C_1^j)$ and two control states \mathbf{z}_\bullet and $\mathbf{z}_!$, decide if there exists a run ρ from the empty-stack empty-channel initial configuration $x_\bullet = ((\mathbf{z}_\bullet, (\epsilon)_{p \in P}), (\epsilon)_{c \in C})$ to the empty-stack empty-channel final configuration $x_! = ((\mathbf{z}_!, (\epsilon)_{p \in P}), (\epsilon)_{c \in C})$ such that $\rho = \rho_1 \cdots \rho_k$, where each ρ_j is a ϕ_j -restricted run.

During a M -phase when the process p is receiving a message, its stack must be empty. That means a M -phase that receives at least one message can be decomposed into three sub-runs such that the first and the last part do not receive any message and such that the middle part starts and finishes with an empty-stack for p . We say that a M -phase is *just-sending* if it does not contain receiving action. We also say that a M -phase is *empty-stack* if it starts and finishes with an empty stack for p . A just-sending M -phase will be denoted by a tuple $\langle p, D \rangle_M$ and an empty-stack M -phase by $\langle c, p, D \rangle_M$. Symmetrically we define the notion of *just-receiving* N -phases characterized by $\langle p, D \rangle_N$ and the *empty-stack* N -phases characterized by $\langle c, p, D \rangle_N$. We refer by ϕ such a tuple.

By multiplying by at most 3 some phases the K -bounded control-state reachability problem reduces to the control-state ...

Theorem 4.2. *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 4.3. Adapting proof ideas from [LTMP07, ABH08], we can show that the complexity bound in Theorem 4.2 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. 3 (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: $\bullet \rightarrow \circ$ top-down and $\circ \rightarrow \circ$ between siblings). Notice also the use of the dual notion to well-queueing, when sending information from lower to higher levels.

Proposition 3.4 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 [SV06].

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.

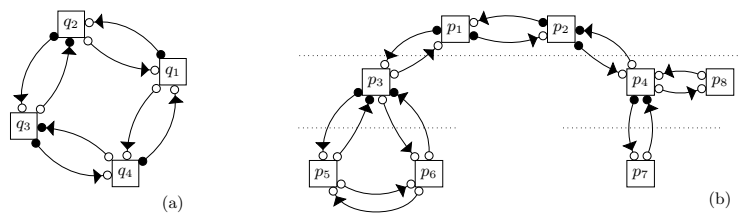


Fig. 3. Non-confluent architectures: (a) ring, and (b) hierarchical master-worker setting — tree-like architecture with \bullet - \circ -channels between master and workers (distribute tasks and collect results while in computation, send result to own master when computation finished, i.e., stack empty) as well as \circ - \circ -channels between workers of the same master; note the \bullet - \circ -cycle on the top level.

References

- [ABH08] Mohamed Faouzi Atig, Benedikt Bollig, and Peter Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *Proc. of DLT 2008*, volume 5257 of *LNCS*, pages 121–133, 2008.
- [ABT08] 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.
- [AJ96] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996.
- [BESS05] 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.
- [BMOT05] 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.
- [BZ83] D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983.
- [CF05] G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
- [EKS03] J. Esparza, A. Kucera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355–376, 2003.
- [FS01] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
- [GKM06] 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.
- [GKM07] B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80:147–167, 2007.
- [JM07] R. Jhala and R. Majumdar. Interprocedural analysis of asynchronous programs. In *Proc. of POPL 2007*, pages 339–350. ACM, 2007.
- [Kos82] S. Rao Kosaraju. Decidability of Reachability in Vector Addition Systems. In *Proceedings of STOC 1982*, pages 267–281. ACM, 1982.
- [LM04] M. Lohrey and A. Muscholl. Bounded MSC communication. *Inf. Comput.*, 189(2):160–181, 2004.
- [LTMP07] 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.
- [LTMP08] 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.
- [May84] E. W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.*, 13(3):441–460, 1984.
- [Pap94] C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
- [QR05] 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.
- [SV06] 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.

Appendix

Proofs for Section: Decidable Oriented Architectures

Proposition . *The RQCP control-state reachability problem is undecidable if the topology is confluent. This holds even if we only consider reachability on eager runs.*

Proof. Assume that we have a simple and unoriented path p, r_1, \dots, r_k, q between $p \neq q$, with $p \bullet \circ r_1$ and $r_k \circ \bullet q$. Then we simulate non-deterministically the intersection of two context-free languages on p, q , by conveying the information about the common input guessed by p, q on the path r_1, \dots, r_k . Since the path between p, q is unoriented, the intermediary processes r_i (for $1 \leq i \leq k$) may have to guess the input, and deadlock if the guess was wrong. \square

Proofs for Section: From Mutex QCP to Eager QCP

Proposition 3.6. *The control state reachability for finite QCP that are mutex with respect to the given architecture is PSPACE-complete.*

Proof. We want to know for a given control state $\mathbf{s} = (s_p)_{p \in \mathcal{P}}$ whether a configuration $c = (\mathbf{s}, \mathbf{w})$ can be reached, for some \mathbf{w} , from the initial configuration c_0 . By Lemma 3.4 we know that c can be reached by a eager run ρ . Consequently, we can guess the run ρ storing the current global state, but without recording more than one message, which gives the (usual) polynomial space bound for products of automata.

Note that we need to take extra care of a channel's growing phase by non-deterministically guessing, at each step of the previous computation, whether a channel will enter its growing phase. This additional information can be stored in polynomial space.

PSPACE-hardness follows by reducing from the non-emptiness test of the intersection of arbitrarily many finite automata. \square

Proposition 3.8. *It is PSPACE-complete to check whether a finite QCP is mutex (eager, resp.) with respect to a given architecture.*

Proof. Assume that \mathcal{A} over an architecture \mathcal{T} is not mutex and consider a run ρ of minimal length from c_0 to a configuration $c = (\mathbf{s}, \mathbf{w})$ that is not mutex. Thus, by minimality all configurations in ρ up to c are mutex.

So by an argument analogous to the construction in the proof of Lemma 3.4, we can reach c' — the predecessor of c — by a eager run ρ' , and test whether there exists a transition $c' \xrightarrow{a} c$ with a send action a that violates the mutex condition for c . Hence, we will guess ρ' without storing more than one message at each step, leading to a configuration c' in which one channel of a cycle in $UCycle(\mathcal{T})$ is non-empty and the action a would write on another channel in this cycle.

As in the previous proof for Proposition 3.6, we have to take care of growing channels by a flag construction.

PSPACE-hardness is not hard to obtain by reducing from the non-emptiness test of the intersection of arbitrarily many finite automata. We can construct the finite QCP in such a way that a “bad” cycle is reached precisely from the target (final) states of the product of the automata. \square

Proofs for Section: Bounded Phase Reachability

Theorem 4.2. *Given a QCP \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}).*

Proof. Let $\rho = \rho_1 \cdots \rho_k$ be a K -bounded run of $\mathcal{A} = \langle (S_p)_{p \in \mathcal{P}}, M, \Sigma, (\Delta_p)_{p \in \mathcal{P}}, s^0 \rangle$, with each ρ_i an M - or an N -phase. W.l.o.g. we suppose that all stacks are empty at the end of ρ . Let $(t_p)_{p \in \mathcal{P}} \in \prod_{p \in \mathcal{P}} S_p$ be the target control state. We first modify the RQCP \mathcal{A} in such a way that we explicitly refer to phases in action names (actually we only consider a subset of \mathcal{A} 's behaviors, namely those corresponding to K -bounded runs): we add to each send $p!q(m)$ the number i of the phase in which the send occurs, as well as the number j of the phase in which the matching receive occurs; in the special case $j = K + 1$, the matching receive does not occur within the first K phases. So $p!q(m)$ is replaced (non-deterministically) by $p!q(m \mid i, j)$, with $i < j \leq K + 1$. We proceed similarly for receives, this time just replacing $p?q(m)$ by $p?q(m \mid i, j)$ with $i < j \leq K$. In addition, we label also the local actions by their phase number. The transition relation Δ_p of each automaton $\mathcal{A}_p = \langle S_p, M, \Sigma_p, \Delta_p, s_p^0 \rangle$ can check that the phase labeling for p 's actions is consistent. The set S_p of local states increases polynomially in K .

At the beginning of the construction we fix for each phase $1 \leq j \leq K$: (1) its type, (2) the process to which it belongs, and (3) its special channel. Moreover, we fix the “reading graph” *Read*, which is a DAG with nodes (p, j) , where p is the process of phase j . If j is an M -phase then there is an edge to (p, j) from each (q, i) such that (a) (q, p) is the special channel of phase j and (b) q in phase i sends to p in phase j . Dually, if j is an N -phase then there is an edge from (p, j) to each (q, k) such that (a) (p, q) is the special channel of phase j and (b) p in phase j sends to q in phase k . Clearly there are some natural consistency requirements on *Read*. Note also that there are only exponentially many graphs *Read*, so that we can check them one by one.

In addition, we will fix at the beginning some state information for 4 distinguished points in each phase i (belonging to some p): the starting/ending point of the phase, together with the first and last point where p 's stack is empty (if any). Of course, some of these points might coincide. For each of the 4 points we fix p 's state at that point. Again, there are exponentially many cases to check one by one.

We now show how to reduce the K -bounded control state reachability question for \mathcal{A} to a $(K - 1)$ -bounded control state reachability question for a modified

RQCP \mathcal{A}' . Assume first that there is at least one phase of type M , and let $j \leq K$ be the index of the last M -phase in ρ . Suppose moreover that phase j belongs to process p , and that the special channel of this phase is $(q, p) \in \text{Ch}$. We will delete phase j , and adapt the two local automata $\mathcal{A}_p, \mathcal{A}_q$ accordingly.

First, we show that we may assume w.l.o.g. that ρ_j starts and ends with empty p -stack. So let $\rho_k = \alpha\beta\gamma$, where β starts and ends with empty p -stack and such that the stack of p is never empty during α , nor during γ (α, β, γ might be empty each). By the definition of M -phases, the only possible actions during α and γ are sends on channels of type $p \bullet \dashv \circ r$. Since all phases $j < \ell \leq K$ are N -phases, these sends are unmatched within ρ , hence they must be labeled as $p!r(c \mid j, K+1)$. We now declare all sends $p!r(m \mid j, K+1)$ as local p -actions, and move α to the end of the last phase $i < j$ that belongs to p . The subrun γ can be either moved to the beginning of the next phase belonging to p , or deleted if there is no such phase. Recall that we know p 's state after α and before γ , respectively. So we can adapt the information about the 4 distinguished points in the phases belonging to p that are modified.

So assume that ρ_j starts in p -state s_p and ends in s'_p , both with empty p -stack. From \mathcal{A}_p we can define a *finite automaton* \mathcal{B}_p that accepts precisely those sequences $(m_1, i_1) \cdots (m_n, i_n) \in (M \times \{1, \dots, j-1\})^*$ such that \mathcal{A}_p can go from s_p to s'_p on a sequence w of actions of phase j only, (1) starting and ending with empty stack, (2) performing $p?q$ actions only when the stack is empty, and such that (3) the projection of w on $p?q$ actions equals $p?q(m_1 \mid i_1, j) \cdots p?q(m_n \mid i_n, j)$. Note that \mathcal{B}_p can be constructed on the same state space as \mathcal{A}_p . We finally modify \mathcal{A}_p by disallowing actions from phase j .

Finally we change the automaton \mathcal{A}_q of process q into \mathcal{A}'_q in order to ensure consistency with ρ_j . The new automaton \mathcal{A}'_q starts simulating \mathcal{B}_p in parallel from state s_p on the subsequence consisting precisely of symbols $q!p(m \mid i, j)$ (for all $m \in M$ and $i < j$). Moreover, the simulation of \mathcal{B}_p must end with state s'_p .

We need now to update the state information for the 4 distinguished points in the phases that are synchronized by \mathcal{B}_p above. Assume that the incoming edges of (p, j) are $(q, i_1), \dots, (q, i_k)$ with $i_1 < \dots < i_k$. So a run of \mathcal{B}_p "splits" over (a suffix of) phase i_1 , followed by phases i_2, \dots, i_k . So for each of the phases i_1, \dots, i_k and each of the 4 distinguished points (except maybe for some of the points of i_1) we need to fix a state of \mathcal{B}_p . We then run the recursive algorithm for each choice of these states. We will see below what this means for the running time.

Now it remains to deal with the case where there is no M -phase in ρ . Here, it is quite easy to check that we may assume from the beginning that each phase starts and ends with empty stack (of all processes). So here we do not need the extra information about the 4 distinguished points for each phase.

Since there are only N -phases we work from left to right and we eliminate the first phase of the run. Suppose that the first phase belongs to process p . Since it is an N -phase, the only possible actions are sends to a unique process, say q , such that $(p, q) \in \text{Ch}$ is of type $p \circ \bullet q$ or $p \circ \dashv q$. In particular, p 's stack is empty during this first phase. So from \mathcal{A}_p we obtain directly a *finite automaton* \mathcal{B}_p that

accepts precisely those sequences $(m_1, j_1) \cdots (m_n, j_n) \in (M \times \{2, \dots, k\})^*$ such that \mathcal{A}_p can go from s_p^0 to some s_p on a sequence w of actions of phase 1 only, (1) working without stack, (2) performing only $p!q$ actions (or local actions), and such that (3) the projection of w on $p!q$ actions equals $p!q(m_1 \mid 1, j_1) \cdots p!q(m_n \mid 1, j_n)$. We also set the initial local state of \mathcal{A}_p to s_p and disable actions from phase 1. We change the automaton \mathcal{A}_q by synchronizing with \mathcal{B}_p on $q?p(m \mid 1, j)$ actions (for all $m \in M$ and $1 < j$). Moreover the synchronization starts with \mathcal{B}_p in state s_p^0 .

Let us now estimate the running time of the construction. For simplicity we focus on the elimination of M -phases. During the phase elimination, automata \mathcal{A}_q get synchronized with automata \mathcal{B}_p (with the same state space as \mathcal{A}_p at the current stage). So local states become *tuples* of local states of the original automata \mathcal{A}_r . If the length of the tuples gets at most $F(j)$ after eliminating $j < K$ phases, then we can show that the overall construction is of complexity $n^{O(K^2 \cdot F(K))}$: denote by $t(i)$ the time needed for i phases. Then $t(1) \leq n^{O(F(K))}$ and $t(i) \leq (n^{F(K-i)})^{4K} \cdot t(i-1)$ (the first factor stands for the states of the 4 distinguished points needed in the recursive step). So we have $t(K) \leq n^{4K \cdot (F(1) + \dots + F(k-1))} t(1) \leq n^{O(K^2 \cdot F(K))}$ as claimed.

Finally, it is not hard to see that $F(j)$ is exponential in j , which gives the doubly exponential upper bound (in K).

□