Phases-loop acceleration for count-and-queue systems*

Benedikt Bollig[1] Étienne Lozes[1,2] Jules Villard[1]

[1] LSV, ENS Cachan, CNRS UMR 8643, FR
[2] MOVES, RWTH Aachen University, DE

Abstract. The set of reachable configurations of a system from a set of initial configurations Γ can be represented as the fixpoint $\mathsf{Post}^*(\Gamma)$ of the Post operator that computes the set of configurations reached after at most one system's transition. To ensure the convergence of fixpoint computations on some classes of systems, one may compute the fixpoint of an acceleration operator Acc instead of Post, such that $\mathsf{Post}(\Gamma) \subseteq \mathsf{Acc}(\Gamma) \subseteq \mathsf{Post}^*(\Gamma)$. We define such an acceleration for systems using queues and counters, and we prove that it terminates on a large class of systems including flat and half-duplex ones. We introduce a symbolic representation for infinite sets of configurations based on Parikh finite automata, and illustrate it on a load-balancing algorithm for which no symbolic representation was previously known.

Introduction

Queue systems are machines communicating through unbounded queues. This model of computation is a very convenient abstraction for a mass of verification problems. To quote some recent examples, Atig & al. [2] observed that programs run over the Relaxed Memory Ordering (RMO), a quite popular relaxed memory model, are naturally modeled by queue systems, and Villard & al. [10] and Fähndrich & al. [5] advocated that part of the analysis of copyless message-passing programs can be reduced to the verification of their contracts, which are another instance of queue systems. However, most of the results on decision problems for queue systems are undecidability ones. Queue systems are thus often assumed to use bounded slacks, or to rely on half-duplex communications, which makes many decision problems decidable. Out of that, to the best of our knowledge, the only class of queue systems whose exact set of reachable configurations can be computed are the flat ones, using loop acceleration [3] techniques.

The set of reachable configurations of a system can be represented as the least fixpoint $\mathsf{Post}^*(\Gamma)$ of the Post operator, which maps a set of configurations Γ to the set of configurations $\mathsf{Post}(\Gamma)$ reached after at most one transition from one configuration in Γ. Acceleration consists in computing Post^* as the fixpoint of an operator Acc, where Acc is such that $\mathsf{Post}(\Gamma) \subseteq \mathsf{Acc}(\Gamma) \subseteq \mathsf{Post}^*(\Gamma)$. Intuitively, Acc will possibly compute configurations reached after sequences of transitions of a certain shape. Unlike widening, the purpose of acceleration is not to ensure the termination of the fixpoint computation on any system by means of approximations, but to guarantee it on some relevant classes of systems while computing the *exact* fixpoint. Since acceleration may

* Work partially supported by PANDA (ANR-09-BLAN-0169) and VERIDYC (ANR-09-SEGI-016), French ANR projects.

manipulate infinite sets of configurations, a symbolic representation (a.k.a. abstract domain) should be designed to make Acc an effective operator.

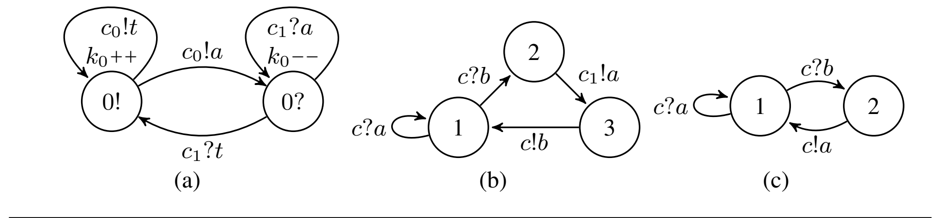
Loop acceleration consists in computing in one step the set of reachable configurations after a sequence of transitions of the form $(\tau_1 \dots \tau_n)^*$. For queue systems, Bouajjani & Habermehl proposed a symbolic representation called Constrained Queue Decision Diagrams (CQDD). This representation goes beyond regular model checking [6], hence induces some complications, in particular for set complement computation [3]. On the other hand, CQDD suffer from several restrictions, and cannot represent all regular queue languages.

In this paper, we go beyond the work of Bouajjani & Habermehl in several directions. First, we consider an extension of queue systems with counters, which we call count & queue systems (CQS). Counters are indeed naturally supported by our approach, and turn out to be very useful for practical case studies. Atomic instructions are hence either the queuing of message a over queue c ($c!a$), or the dequeuing of a message a over c ($c?a$), or the increment/decrement of the counter k by n ($k := k + n$). Second, we design a new acceleration, called phases-loop acceleration, which terminates for flat, slack bounded, and half-duplex CQS, and, more generally, for a class of machines that we call phases-flat. We partition the set of queue instructions in respectively internal and phase-switching ones. A $c\dagger$-phase is then an (unbounded) sequence of either counters increment or internal instructions of the form $c\dagger a$, ended with a phase-switching instruction $c\dagger a'$. Phases-loop acceleration then computes in one step the set of reachable configurations after a sequence of transitions of the form $(c_1\dagger_1^* a_1 \dots c_n\dagger_n^* a_n)^*$, where each $c_i\dagger_i^* a_i$ represents a $c_i\dagger_i$-phase ended by $c_i\dagger_i a_i$. Third, we introduce a symbolic representation we call Symbolic Counting Queues (SCQ) that supports this acceleration. SCQs are strictly more expressive than both CQDD and regular sets of configurations, and are essentially based on Parikh Finite Automata (PFA), introduced by Klaedke & Ruess for deciding the monadic second order logic with cardinalities [8]. Fourth, as inclusion testing is not decidable for SCQ, we propose an approach to partially decide it and hence decide when the fixpoint iteration should be stopped. We illustrate our technique on a small but challenging case study coming from Heaphop [11], that features a load-balancing algorithm. As its reachability set is not CQDD representable, even abstracting counters, we believe that this example cannot be handled by existing techniques, which was our initial motivation.

Outline In Section 1, we introduce our case study, and formalize the definitions of count & queue systems and phases-loop. In Section 2, we recall the principles of acceleration techniques, and show some results concerning phases-loop acceleration. In Section 3, we introduce symbolic counting queues, our symbolic representation, and establish, in Section 4, that it supports phases-loop acceleration.

Related works Many approaches have been proposed to model-check queue systems; we list here those that we initially expected to work for our case study. Cécé & Finkel introduced an acceleration technique for half-duplex queue systems [4] that keeps symbolic configurations regular, but the load-balancing example is not half-duplex, and half-duplex acceleration actually diverges on it. The set of reachable configurations of our example can be covered by allowing only three context-switches, but there is no

Figure 1 (a) Abstraction of one of the two threads used in the load-balancing algorithm. (b) Another phases-flat CQS. (c) A CQS that is not phases-flat.



known mean of computing the set of reachable configurations after one context-switch: we considered modeling counters as stacks to reuse existing techniques for bounded context-switch analysis of recursive queue systems [9,1], but our example does not enjoy the well-dequeuing assumptions used in these works. Moreover, these bounded context-switch analyses consider the set of reachable control states, whereas we are concerned with the more general problem of computing the set of reachable configurations.

1 Phases-Loops of Count & Queue Systems

Consider a program that has to entirely scan a binary tree using two threads; each node of the tree is a subtask t, thus any subtask is either "simple" (a leaf of the tree), or "complex" (an intermediate node), and generates two new subtasks. The shape of the task tree is not known in advance, so one cannot statically assign the subtasks to each thread in a balanced way. The algorithm roughly consists in letting thread 0 treat all the left children of any node of the tree (including the left child of the right child of the root, for instance), and similarly letting thread 1 treat all the right children. The mode of operation of thread 0 (resp. 1) is thus either to process a task if it is simple, or, when it is complex, to send the right (resp. left) subtree to the other thread, repeating the same process with the left (resp. right) subtree until a simple task is reached. Thus, threads communicate through two channels c_0, c_1 (see Fig 1a) over which they send and receive tasks (message t). In order to know when they should stop, they must be able to determine whether their incoming queue is empty and will remain empty whatever the other thread will do. To do so, each thread $i \in \{0, 1\}$ maintains a local counter k_i to store the number of tasks that it has sent and are still pending, and stops as soon as k_i becomes null. In order to keep this counter up to date, each thread sends an acknowledgment message a for each simple task it has completely treated. The counter k_i is then incremented when a new task is sent over c_i, and decremented when an acknowledgment is received over c_{1-i}. The automaton depicted in Fig. 1a models one of these threads, the other one being identical up to swapping indexes 0 and 1.

The whole program can thus be abstracted as a count & queue system (see Def. 1 below) with sixteen[1] control states and two counters. Its initial configuration is the

[1] Note that one thread has four control states in our CQS model. For conciseness, we often skip intermediate control states and use multi-instruction transitions in our figures.

control state $(0!, 1!)$, with both counters equal to one, and both queues empty. The final configurations are in control state $(0?, 1?)$, with both counters equal to zero. Interesting questions about this program are whether it terminates (provided the whole task tree is finite), whether it deadlocks, and whether or not it ends with pending tasks still in the queues.

Let us now introduce the model we use to represent this problem. Essentially, a count & queue system is a finite-state machine that can send and receive messages over a finite set of unbounded FIFO queues (or channels), and increment/decrement counters without testing them:

Definition 1 (CQS) *A* count & queue system (CQS) *is a tuple*

$$\mathcal{M} = (S, K, C, \Sigma, T)$$

where S is a finite set of control states, K is a finite set of counters, C is a finite set of channels, Σ is a finite set of messages, and T is a finite set of transitions. Each transition is a triple (s_1, α, s_2), sometimes written $s_1 \xrightarrow{\alpha} s_2$, where $s_1, s_2 \in S$ and α is an instruction of the following form, where $c \in C$, $k \in K$, $n \in \mathbb{Z}$, and $a \in \Sigma$:

$$\alpha ::= c!a \mid c?a \mid k := k + n$$

Let $P = (\mathsf{Int}, \mathsf{Switch})$ be a 2-partition of $C \times \Sigma$. An instruction α is called *internal* for P if it is either $k := k + n$ or $c\dagger a$ with $\dagger \in \{!, ?\}$ and $(c, a) \in \mathsf{Int}$; otherwise, α is called *phase-switching*. Let us fix some $c \in C$ and $\dagger \in \{!, ?\}$. A sequence of instructions $\alpha_1 \dots \alpha_n$ with $n \geq 1$ is a $c\dagger$*-phase* for P if, for all $i \in \{1, \dots, n\}$, (1) α_i is of the form either $k := k + n$ or $c\dagger a$ for some a, and (2) α_i is phase-switching if and only if $i = n$. A *phases-loop language* for P is a regular language of the form $(c_1\dagger_1^* a_1 \dots c_n\dagger_n^* a_n)^*$, where $c_i\dagger_i^* a_i$ denotes the regular language of $c_i\dagger_i$-phases whose final message is a_i. We say that a sequence of instructions is potentially triggered by a CQS if it labels consecutive transitions. Finally, for a language L, we write $\downarrow L$ to denote the downward closure of L for infix order, *i.e.* the set of words w for which there are w_1, w_2 such that $w_1 w w_2 \in L$.

Definition 2 (Phases-loop) *A CQS is a* phases-loop *if there is a 2-partition $P = (\mathsf{Int}, \mathsf{Switch})$ and a phases-loop language L for P such that all potentially triggered sequences of instructions are in $\downarrow L$. A CQS $\mathcal{M}$ is* phases-flat *if, for all sets $S' \subseteq S$ of strongly connected control states, the restriction of $\mathcal{M}$ to S' is a phases-loop.*

Note that all CQS that are flat (*i.e.* all strongly connected components are elementary cycles) are also phases-flat, and that phases-flatness is decidable.

Example 1.1. The following CQS are phases-flat:

- the thread of the load-balancing algorithm depicted in Fig. 1a: the strongly connected set of states $\{0!, 0?\}$ is a phases-loop taking (c_0, t) and (c_1, a) as internal, and $(c_1, t), (c_0, a)$ as phase-switching.
- the CQS obtained by scheduling first thread 0, then thread 1 of the load-balancing algorithm; note that the 2-partition must be taken different for each of the two maximal sets of strongly connected control states.

– the tread of Fig 1b: (c, a) is internal, and (c, b) and (c_1, a) are phase-switching.

The following CQS are not phases-flat:

– the CQS depicted on Fig 1c: there is no partition P that allows one to define a correct phases-loop language on the maximal set of strongly connected states.
– the CQS obtained by switching from one thread to another after each transition in the load-balancing algorithm, and as a consequence the CQS that models all possible interleaving of the two threads are not phases-flat.

2 Phases-Loop Acceleration

We associate a semantics to every CQS in a standard way, by means of a transition system. Let $\mathcal{M} = (S, K, C, \Sigma, T)$ be a CQS. The set $\mathsf{Conf}(\mathcal{M})$ of *configurations* of $\mathcal{M}$ is $S \times (\Sigma^*)^C \times \mathbb{Z}^K$. We simply write Conf if $\mathcal{M}$ is understood.

Definition 3 (Transition System) *The relation $\gamma \xrightarrow{\tau} \gamma'$, where $\tau = (s, \alpha, s') \in T$ and $\gamma = (s, \mathbf{w}, \mathbf{n})$, $\gamma' = (s', \mathbf{w}', \mathbf{n}')$ are configurations of $\mathcal{M}$, is defined as follows:*

– *if $\alpha = c!a$ or $\alpha = c?a$, then (1) $w'_c = w_c a$ if $\alpha = c!a$ and $w_c = a w'_c$ if $\alpha = c?a$, (2) $w'_{c'} = w_{c'}$ for all $c' \neq c$, and (3) $\mathbf{n}' = \mathbf{n}$;*
– *if $\alpha = k := k + n$, then (1) $n'_k = n_k + n$, (2) $n'_{k'} = n_{k'}$ for all $k' \neq k$, and (3) $\mathbf{w}' = \mathbf{w}$.*

The transition relation $\rightarrow_M \subseteq \mathsf{Conf}(\mathcal{M}) \times \mathsf{Conf}(\mathcal{M})$ relates a configuration γ to a configuration γ': $\gamma \rightarrow_{\mathcal{M}} \gamma'$ if there exists $\tau \in T$ such that $\gamma \xrightarrow{\tau} \gamma'$.

We write Γ to denote a subset of $\mathsf{Conf}(\mathcal{M})$, and $\Gamma[s]$ for the subset of configurations of Γ in control state s. We write $\mathsf{Post}_{\mathcal{M}}$ to denote the function $2^{\mathsf{Conf}} \rightarrow 2^{\mathsf{Conf}}, \Gamma \mapsto \Gamma \cup \{\gamma' \in \mathsf{Conf} \mid \exists \gamma \in \Gamma. \gamma \rightarrow_{\mathcal{M}} \gamma'\}$. For $F : 2^{\mathsf{Conf}} \rightarrow 2^{\mathsf{Conf}}$, we let F^* denote the function $2^{\mathsf{Conf}} \rightarrow 2^{\mathsf{Conf}}$, $\Gamma \mapsto \bigcup_{i \geq 0} F^i(\Gamma)$.

Definition 4 (Safety problem) *The safety problem[2] consists in deciding, given a CQS $\mathcal{M}$, a set Γ_0 of initial configurations and a set Γ_{bad} of "bad" configurations, whether $\mathsf{Post}^*_{\mathcal{M}}(\Gamma_0) \cap \Gamma_{bad} = \emptyset$.*

Most of the properties one would like to check over CQS can be reduced to the safety problem (for instance, all properties listed before for the load-balancing algorithm, except termination, can be formulated in terms of safety properties). It is well known that the safety problem is undecidable for arbitrary CQS, and decidable on some classes of queue systems using strong restrictions. One way to tackle the safety problem in full generality is to use acceleration techniques. An acceleration operator is a function $\mathsf{Acc}_{\mathcal{M}} : 2^{\mathsf{Conf}} \rightarrow 2^{\mathsf{Conf}}$ such that $\mathsf{Acc}^*_{\mathcal{M}} = \mathsf{Post}^*_{\mathcal{M}}$. The semi-algorithm that uses this operator to solve the safety problem is described in Fig 2. We say that the acceleration operator $\mathsf{Acc}_{\mathcal{M}}$ terminates for a machine $\mathcal{M}$ and a set Γ_0 of initial configurations if there is some i such that $\mathsf{Acc}^i_{\mathcal{M}}(\Gamma_0) = \mathsf{Post}^*_{\mathcal{M}}(\Gamma_0)$.

[2] We focus on forward analysis, but all our work could be presented as a backward analysis as well, thanks to the intrinsic reversibility of CQS.

Figure 2 Acceleration algorithm

Input: $\mathcal{M}$, Γ_0, Γ_{bad}.
Repeat $(\Gamma_0, \Gamma) \leftarrow (\mathsf{Acc}_{\mathcal{M}}(\Gamma_0), \Gamma_0)$; If $\Gamma_0 \cap \Gamma_{bad} \neq \emptyset$ Then Return *unsafe* Until $\Gamma_0 \subseteq \Gamma$;
Return *safe*.

Definition 5 (Phases-loop acceleration) *Let $\mathcal{M}$ be a CQS and $\mathcal{M}'$ be a CQS obtained from $\mathcal{M}$ by considering a subset of its transitions such that $\mathcal{M}'$ is a phases-loop. Let $Ph(\mathcal{M})$ be the set of all such $\mathcal{M}'$. The phases-loop acceleration of $\Gamma \subseteq \mathsf{Conf}(\mathcal{M})$ with respect to $\mathcal{M}'$, written $\mathsf{Acc}_{\mathcal{M},\mathcal{M}'}(\Gamma)$, is $\mathsf{Post}^*_{\mathcal{M}'}(\Gamma)$. The phases-loop acceleration of $\mathcal{M}$ over Γ is the set*

$$\mathsf{Acc}_{\mathcal{M}}(\Gamma) \triangleq \bigcup_{\mathcal{M}' \in Ph(\mathcal{M})} \mathsf{Acc}_{\mathcal{M},\mathcal{M}'}(\Gamma) \cup \mathsf{Post}_{\mathcal{M}}(\Gamma) \ .$$

Let us recall that a configuration γ is B-slack-bounded for a certain $B \in \mathbb{N}$ if all queues contain at most B messages, and a CQS is slack-bounded if there exists $B \in \mathbb{N}$ such that for all set Γ of B-slack-bounded configurations, $\mathsf{Post}^*(\Gamma)$ is a set of B-slack-bounded configurations. Let us moreover recall that a configuration is half-duplex if at most one queue is non empty, and a CQS is half-duplex if all set Γ of half-duplex configurations, $\mathsf{Post}^*(\Gamma)$ is a set of half-duplex configurations. The reachability set $\mathsf{Post}^*(\Gamma)$ of a half-duplex queue system is regular and can be obtained by first considering the set Γ_1 of all configurations reachable with a slack of size 1, and then all configurations reachable from Γ_1 through queuing transitions only [4]. As a consequence:

Proposition 6 *Phases-loop acceleration terminates over the following classes of CQS: phases-loop flat, flat, half-duplex, and slack-bounded ones.*

Example 2.1. Let $|w|_l$ denote the number of occurrences of the letter l in the word w. Let $\mathcal{M}_i$ be the thread i of the load-balancing algorithm of Fig 1a.

- Let Γ be the phases-loop acceleration of $\mathcal{M}_0$ over the singleton set Γ_0 of the initial configuration (control state $(0!, 1!)$, empty queues c_0, c_1, and $k_0 = k_1 = 1$). Then, $\Gamma[0?, 1!]$ is the set of configurations where c_1 is empty, $k_1 = 1$, and c_0 contains the word $w_0 = t^{k_0 - 1}a$.
- Let Γ' be the phases-loop acceleration of $\mathcal{M}_1$ over $\Gamma[0?, 1!]$. Then, $\Gamma'[0?, 1?]$ is the set of configurations such that c_0 contains $w_0 \in \varepsilon + t^*a$, c_1 contains a word $w_1 \in (t^*a)^*$, and $k_i = |w_i|_t + |w_{1-i}|_a$ for $i \in \{0, 1\}$.
- Let Γ'' be the phases-loop acceleration of $\mathcal{M}_0$ over $\Gamma'[0?, 1?]$. Then, the configurations in $\Gamma''[0?, 1?]$ are the ones for which c_i contains $w_i \in (t^*a)^*$ and $k_i = |w_i|_t + |w_{1-i}|_a$, which is the fixpoint. Hence phases-loop acceleration terminates over the load-balancing algorithm, although it is not phases-flat.

In order to effectively compute an acceleration, one needs to provide a symbolic representation for infinite sets of configurations. Minimum requirements for symbolic configurations could be the following:

Definition 7 (Symbolic representation) *A symbolic representation for* $\mathcal{M}$ *consists of a set of effective objects* $\mathcal{O}$ *and a denotation* $[\![.]\!] : \mathcal{O} \to 2^{\mathsf{Conf}(\mathcal{M})}$ *such that the following operations are effective:*

- *all finite sets of configurations can be coded into objects;*
- *there are effective operations on objects denoting union and intersection;*
- *emptiness testing is decidable.*

A symbolic representation is perfect *if, moreover, inclusion testing is decidable. An acceleration* F *is* supported *by a symbolic representation* $(\mathcal{O}, [\![.]\!])$ *if, for every* $O \in \mathcal{O}$*, there is an effective* $O' \in \mathcal{O}$ *such that* $F([\![O]\!]) = [\![O']\!]$*.*

Perfection is important if we aim at implementing the termination test of the algorithm of Fig 2. As we will see in the next section, our symbolic representation is not perfect and, even worse, it is tight in the sense that any other symbolic representation supporting phases-loop should be at least as expressive. As a consequence:

Proposition 8 *There is no perfect symbolic representation that supports phases-loop acceleration.*

Compared to SCQ, CQDD have the good property to be a perfect symbolic representaion, although they are not closed under complement. In the next section, we will show how to overcome this situation. On the other hand, having a symbolic representation, even imperfect, that supports phases-loop acceleration already implies the following result:

Theorem 9 *The safety problem is decidable for phases-flat CQS.*

3 Symbolic Counting Queues

We now define a symbolic representation that supports phases-loop acceleration. Note that CQDD [3] do not represent all regular sets of configurations, which are all achieved by half-duplex machines. As a consequence, it can already be observed that CQDD do not support phases-loop acceleration (it can also be observed that the reachability set of the load-balancing algorithm is not CQDD definable).

Presburger arithmetic is the multiplier-free fragment of first-order arithmetic over integers. Let us recall the definition of Parikh Finite Automata [8] (PFA, for short), slightly adapted to our setting:

Definition 10 (PFA) *A Parikh Finite Automaton is a tuple*

$$\mathcal{P} = (S, \Sigma, s_0, S_f, K, \Phi, T)$$

where S *is a finite set of states,* Σ *is a finite alphabet,* $s_0 \in S$ *is the initial state,* $S_f \subseteq S$ *is the set of final states,* K *is a finite set of counters,* Φ *is a Presburger formula over* K*, and* T *is a set of transitions* (s_1, ℓ, s_2)*, where* $s_1, s_2 \in S$ *and* ℓ *is either a read label* $\mathsf{read}(a)$ *with* $a \in \Sigma$*, or a counter label* $k := k + n$ *with* $k \in K$ *and* $n \in \mathbb{Z}$*.*

A K-*indexed language* (over Σ) is a set $L \subseteq \Sigma^* \times \mathbb{Z}^K$. For a K-indexed language L and $\mathbf{n} \in \mathbb{Z}^K$, let $L_\mathbf{n}$ denote the set of words $w \in \Sigma^*$ such that $(w, \mathbf{n}) \in L$. Union, intersection, concatenation, and left/right quotient are defined for K-indexed languages index-wise; for instance, the concatenation $L.L'$ is defined by $(L.L')_\mathbf{n} \triangleq L_\mathbf{n}.L'_\mathbf{n}$. The *emptiness index set* of L is the set of indexes $\mathbf{n}$ such that $L_\mathbf{n} = \emptyset$.

The K-indexed language $L(\mathcal{P})$ recognized by the PFA $\mathcal{P}$ is the set of pairs $(w, \mathbf{n})$ such that there is a final state $s \in S_f$ and a sequence of transitions $\theta \in T^*$ such that $s_0 \xrightarrow{\theta} s$, θ reads $w \in \Sigma^*$, the sum of all k increments along θ is n_k, and $\mathbf{n} \models \Phi$. A K-indexed language L is called *PFA recognizable* if there is a PFA $\mathcal{P}$ that recognizes it. For instance, the indexed language $\{(a^n b^n c^n, n) : n \in \mathbb{N}\}$ is PFA recognizable[3]. The expressive power of PFA has been established [8] to be the same as that of Ibarra's [7] reversal-bounded counter machines[4]. The price to pay for considering such an expressive class of languages is that the universality and inclusion problems are undecidable, and the set complement of a K-indexed PFA recognizable language is not PFA recognizable in general.

A homomorphism for indexed languages is a pair $\phi = (\sigma, f)$ where $\sigma : \Sigma_1 \to \Sigma_2^*$ is a substitution and $f : \mathbb{Z}^{K_1} \times \mathbb{Z}^{\Sigma_1} \to \mathbb{Z}^{K_2}$ is a Presburger definable function. For $w \in \Sigma_1^*$ and $\mathbf{n} \in \mathbb{Z}^{K_1}$, we let $\phi(w, \mathbf{n}) \triangleq \big(w\sigma, f(\mathbf{n}, (|w|_a)_{a \in \Sigma_1})\big) \in \Sigma_2^* \times \mathbb{Z}^{K_2}$.

Lemma 11 *The set of PFA recognizable K-indexed languages is effectively closed under union, intersection, concatenation, left/right quotients and direct/inverse homomorphisms. Moreover, the emptiness index set can be described by a Presburger formula.*

All proofs are straightforward adaptions of the proofs of Klaedtke and Ruess [8]. PFAs can be used to represent the set of possible contents of one queue. Our symbolic representation builds on this idea; intuitively, a set of configurations is represented by a finite set of tuples of PFAs working on some auxiliary counters, and an additional Presburger constraints relating all auxiliary counters to the counters manipulated by the considered CQS.

Definition 12 (ASCQ, SCQ) *An* atomic symbolic counting queue *(ASCQ) for a CQS $\mathcal{M} = (S, K, C, \Sigma, T)$ is a tuple*

$$\mathcal{A} = \big(s, (K_c)_{c \in C}, \Phi, (\mathcal{P}_c)_{c \in C}\big)$$

such that

- *$s \in S$ is a control state of $\mathcal{M}$,*
- *for all $c \in C$, K_c is a set of counters,*
- *Φ is a Presburger formula over $K \cup \bigcup_{c \in C} K_c$, and*
- *for all $c \in C$, $\mathcal{P}_c$ is a PFA with alphabet Σ and counters K_c.*

[3] The Dyck language, however, is *not* PFA recognizable [8].

[4] Note that, unlike for reversal-bounded counters machines, counters in the PFA model are never tested during computation. This is the reason why our count & queue systems do not authorize to test counters either. However, tests on reversal-bounded counters could probably be handled, but would yield a more complex presentation of our results.

A Symbolic Counting Queue *(SCQ) for $\mathcal{M}$ is a finite set of ASCQs for $\mathcal{M}$.*

Let $\mathcal{S}$ be a SCQ for $\mathcal{M}$. The set $[\![\mathcal{S}]\!]$ of *configurations* of $\mathcal{S}$ is the union of those of its ASCQs, and the set of configurations of an ASCQ $(s, (K_c)_{c\in C}, \Phi, (\mathcal{P}_c)_{c\in C})$ is the set of configurations $(s, \mathbf{w}, \mathbf{n})$ of $\mathcal{M}$, where $(\mathbf{w}, \mathbf{n}) \in (\Sigma^*)^C \times \mathbb{Z}^K$ are such that (1) for each $c \in C$ there is some $\mathbf{m}_c \in \mathbb{Z}^{K_c}$ satisfying $(w_c, \mathbf{m}_c) \in L(\mathcal{P}_c)$, and (2) $\mathbf{n}, \mathbf{m}_{c_1}, \ldots, \mathbf{m}_{c_n} \vDash \Phi$. As an example, the set of configurations with only a_1 messages in c_1 and a_2 messages in c_2 such that there are as many messages in each queue is definable by an ASCQ. The reason for introducing both SCQs and ASCQs is that ASCQs are not closed under finite union: for a CQS manipulating two queues, the set of configurations such that each queue contains a unique message, but such that both queues do not contain the same message, is not definable by an ASCQ.

Proposition 13 *Let $\mathcal{M}$ be a CQS. Then, SCQs for $\mathcal{M}$ form a symbolic representation.*

As mentioned before, SCQs do not form a perfect symbolic representation since inclusion testing is not decidable for PFAs. A possible way to establish inclusion between two PFAs (and hence between two SCQs) is to associate to a PFA $(S, \Sigma, s_0, S_f, K, \Phi, T)$ the labeled transition system such that for all $s, s' \in S$, $a \in \Sigma$, and $\mathbf{m} \in \mathbb{Z}^K$, $s \xrightarrow{a,\mathbf{m}} s'$ if there is a sequence of transitions $\theta \in T^*$ from s to s' such that exactly one read label $\mathsf{read}(a)$ appears in θ, and the sum of counter increments is $\mathbf{m}$. We define $s \xrightarrow{\varepsilon,\mathbf{m}} s'$ similarly. A relation $R \subseteq S \times S$ is a weak simulation if (1) whenever $s_1 R s_2$ and $s_1 \xrightarrow{a,\mathbf{m}} s_1'$ for some $s_1, s_2, s_1' \in S$, some $\mathbf{m} \in \mathbb{Z}^K$, and some $a \in \Sigma \cup \{\varepsilon\}$, there is $s_2' \in S$ such that $s_2 \xrightarrow{a,\mathbf{m}} s_2'$ and $s_1' R s_2'$, and (2) whenever s_1 is a final state, s_2 is a final state. Then weak simulation clearly induces language inclusion. Moreover, although the transition system is infinite, weak simulation is decidable, as it can be represented in Presburger arithmetic using Parikh's theorem. We can hence implement our acceleration algorithm using simulation checking instead of language inclusion.

4 How to Compute the Acceleration

In this section, we establish that SCQ supports phases-loop acceleration. We first introduce the notion of bi-runs, which will be used to accelerate a CQS manipulating only one channel and no counters, then explain how to recognize bi-runs using PFA, and then conclude the proof.

4.1 Bi-Runs

Let $\mathcal{L}$ be a phases-loop of a CQS using only one queue c and no counter. Let $\mathsf{Post}_{\mathcal{L}} : 2^{\Sigma^*} \to 2^{\Sigma^*}$ be the function that associates to a set of ($\emptyset$-indexed) words the set of words obtained after one phases-loop iteration.

The parametrized transitive closure $\mathsf{PPost}_{\mathcal{L}}$ of $\mathsf{Post}_{\mathcal{L}}$ is the function $2^{\Sigma^*} \to 2^{\Sigma^* \times \mathbb{N}}$ that associates to a ($\emptyset$-indexed) language L_0 the $\{i_c\}$-indexed language of indexed words (w', n) such that $w' \in \mathsf{Post}^n_{\mathcal{L}}(L_0)$.

In this section and the next one, we will explain how to represent $\mathsf{PPost}_{\mathcal{L}}(L_0)$ using PFA. But first, let us give some intuition about why we should care about it for computing the acceleration, which will be presented more formally in Sec. 4.3:

Example 4.1. Consider again the CQS of Fig 1b. Assume we want to accelerate it taking as initial set of configurations those in control state 1 with a word in $(ab)^*$ in queue c and queue c_1 empty. Then we need to compute the parametrized transitive closures of the phases-loop $\mathcal{L}_c$ over c (resp. $\mathcal{L}_{c_1}$ over c_1) obtained by replacing c_1 (resp. c) transitions by ε transitions in the phases-loop: $\mathsf{PPost}_{\mathcal{L}_{c_1}}(\varepsilon) = \{(a^{i_{c_1}}, i_{c_1})\}_{i_{c_1} \geq 0}$ is the acceleration over c_1, and $\mathsf{PPost}_{\mathcal{L}_c}((ab)^*) = \{\big((ab)^* b^{i_c}, i_c\big)\}_{i_c \geq 0} \cup \{(b^k, i_c)\}_{i_c \geq k > 0}$ is the acceleration over c. Then the acceleration of the CQS is the ASCQ composed of these two queue languages, with the additional constraint that $i_{c_1} = i_c$.

The most complex part of the construction of $\mathsf{PPost}_{\mathcal{L}}(L_0)$ can already be witnessed on this example. Consider the result of $\mathsf{PPost}_{\mathcal{L}_c}((ab)^*)$ a bit more carefully: $\{\big((ab)^* b^{i_c}, i_c\big)\}_{i_c \geq 0}$ is obtained through an execution where all dequeued messages were there initially, whereas $\{(b^k, i_c)\}_{i_c \geq k > 0}$ is obtained through what we will call "hungry snake[5] executions", where some b's that are queued during acceleration are later dequeued also during acceleration. Hungry snakes are not always possible (for instance, for c_1 in previous example), and in this case, the expression of the resulting queue content is $(w_?^{-1}.w_0).w_!$, where $w_?$ is the word dequeued, w_0 is the initial queue content, and $w_!$ the word queued. It is then rather straightforward to check that $\mathsf{PPost}_{\mathcal{L}}(L_0)$ is PFA definable as soon as L_0 is PFA definable, since PFA languages are stable by left quotient and concatenation. $\mathsf{PPost}_{\mathcal{L}}(L_0)$ turns out to be also PFA definable if we consider hungry snake executions, but the proof is more involved.

Let us now present some intuitions about how we will compute $\mathsf{PPost}_{\mathcal{L}}(L_0)$. Let us for instance consider the pair $(bb, 4) \in \mathsf{PPost}_{\mathcal{L}_c}((ab)^*)$ obtained starting from initial queue $abab$ through a hungry snake execution $?a?b!b?a?b!b?b!b?b!b$. Let us tag every instruction with the phase number at which this instruction occurred:

$$\begin{array}{cccccccccc} ?a & ?b & !b & ?a & ?b & !b & ?b & !b & ?b & !b \\ 1 & 1 & 2 & 3 & 3 & 4 & 5 & 6 & 7 & 8 \end{array}$$

If we group every $?l$ instruction with its matching l letter of $abab$ or with its matching $!l$ instruction, and write $\square$ under unmatched send instructions, we obtain the following picture (the framed part corresponds to the hungry snake mode):

$$\begin{array}{cccc|cc|cc} \cline{5-6} &&&& 2 & 4 & 6 & 8 \\ a & b & a & b & !b & !b & !b & !b \\ ?a & ?b & ?a & ?b & ?b & ?b & \square & \square \\ 1 & 1 & 3 & 3 & 5 & 7 & & \\ \cline{5-6} \end{array}$$

This representation is convenient, because it is easy to compute the resulting word bb by an homomorphism. Moreover, if the correct phase numbers are given, it is possible to reconstruct the actual run in a unique way. However, not all choices for phase numbers are correct, and we will have to clarify what are the admissible choices. There are in fact three conditions: the first condition considers the send (resp. receive) instructions only, and ensures that the phase number associated to an instruction is a correct choice with respect to previous send (resp. receive) instructions. For instance, the choice below is not a correct choice, because the phase number of the third send instruction is too large:

[5] A hungry snake eats its own tail!

				2	4	8	10
a	b	a	b	$!b$	$!b$	$!b$	$!b$

The second condition considers the first and last lines, and ensures that whenever a send instruction is matched with a receive instruction, the send instruction happened before the receive instruction in the original run; this is expressed by ensuring that on each row of the hungry snake mode, the number on the first line is smaller than the one on the last line. The third condition is that the last phase numbers correspond to the last phase of a loop, and represent the same number of loop iterations for send and receives.

With these three conditions, it is then clear how to reconstruct a valid run using the counter information. However, we will use slightly more complex counters in our definition of *bi-runs*, as we need to compute the number of loop iterations we use while accelerating. Instead, we decompose the phase number in two numbers, called quotient and remainder: the quotient is the number of loop iterations already completed, and the remainder is the phase number with respect to the last, uncompleted loop iteration. For instance, the previous run will be represented by the following bi-run:

remainder					0	0	0	0
quotient					1	2	3	4
	a	b	a	b	$!b$	$!b$	$!b$	$!b$
	$?a$	$?b$	$?a$	$?b$	$?b$	$?b$	$\Box$	$\Box$
quotient	0	0	1	1	2	3	4	4
remainder	1	1	1	1	1	1	0	0

We now introduce the notion of bi-runs more formally. Let us fix the channel c of $\mathcal{L}$ and the alphabet Σ of L_0. We consider the alphabets $\Sigma_\dagger = \{\dagger\} \times \Sigma$, $\Sigma_{?\Box} = \Sigma_? \uplus \{\Box\}$, and $\Sigma_{!?} \triangleq (\Sigma \cup \Sigma_!) \times \Sigma_{?\Box}$. We consider the sub-alphabet $\Sigma_{synch} \subseteq \Sigma_{!?}$ such that what is received matches what is sent: $\Sigma_{synch} \triangleq \bigcup_{a\in\Sigma}\{a, !a\} \times \{?a, \Box\}$.

We consider $\mathcal{L}_!$ (resp. $\mathcal{L}_?$) the language over $\Sigma_!$ (resp. $\Sigma_?$) of one iteration of the phases-loop obtained by replacing all dequeuing transitions (resp. all queuing transitions) by ε transitions. For instance, $\mathcal{L}_?$ and $\mathcal{L}_!$ are respectively $(?a)^*?b$ and $!b$ for the computation of the parametric transitive closure over c of the CQS of Fig 1b.

Let $a_1 \dots a_n$ be the sequence of phase-switching messages such that the accelerated phases-loop $\mathcal{L}$ is $c\dagger_1^* a_1 \dots c\dagger_n^* a_n$. For all $j \in \{1, \dots, n-1\}$, let $\mathcal{L}[j] \triangleq c\dagger_1^* a_1 \dots c\dagger_j^* a_j$, and $\mathcal{L}_\dagger[j]$ obtained from $\mathcal{L}[j]$ by replacing $\dagger a$ instructions by ε. A *quotient* and a *remainder* of a prefix w of $\mathcal{L}_\dagger^*$ is a pair $(i, j) \in \mathbb{N} \times \{0, 1, \dots, n-1\}$ such that w is a prefix of $\mathcal{L}_\dagger^i.\mathcal{L}_\dagger[j]$ and one of its prefixes is in $\mathcal{L}_\dagger^{i'}.\mathcal{L}_\dagger[j']$, where (i', j') is the predecessor of (i, j) in lexicographic order. Due to this definition, there are sometimes several possible pairs quotient-remainder with regard to the same word w. The *principal* quotient and remainder of a prefix w of $\mathcal{L}_\dagger^*$ is the smallest pair (i, j) for the lexicographic order. For instance, for $\mathcal{L} = (?a)^*?b!b$ as in the previous example, the unique quotient and remainder of $?a?a \in \mathcal{L}_?^*$ is $(0, 1)$; but for $?a?a?b \in \mathcal{L}_?^*$, one can chose between $(0, 1)$, which is the principal quotient and remainder, and $(1, 0)$; for $!b!b \in \mathcal{L}_!^*$, there are $(2, 0)$, and the principal $(1, 1)$.

We call *bi-run* a finite sequence ρ of PFA labels over $\Sigma_{!?}$, *i.e.* either read labels over alphabet $\Sigma_{!?}$, or counter labels. We write $w(\rho)$ to denote the $\Sigma_{!?}$-word read by ρ. If w

is a word and p is a position in w, then $w^{\leq p}$ denotes the prefix of w consisting of the first p letters. Finally, the value of a counter k at the position p of a bi-run ρ is the sum of all increments of k on labels occurring before p.

Definition 14 *We say that a bi-run ρ is* well-synchronized *w.r.t. $\mathcal{L}$ and L_0 if the following conditions are met:*

1. *the first projection of $w(\rho)$ is $w_{11}w_{12}$ for some $w_{11} \in L_0$ and $w_{12} \in \mathcal{L}_!^*$;*
2. *the second projection of $w(\rho)$ is $w_{21}w_{22}$ for some $w_{21} \in \mathcal{L}_?^*$ and $w_{22} \in \Box^*$;*
3. *$w(\rho)$ belongs to Σ_{synch}^*;*
4. *there are counters $k_!$, $k'_!$, $k_?$, $k'_?$ whose values at position p of a read label in ρ are respectively the principal quotient and remainder of $w_{12}^{\leq p-|w_{11}|}$ and a quotient and a remainder of $w_{21}^{\leq p}$;*
5. *the final values of $k_!$ and $k_?$ are equal, the final values of $k'_!$ and $k'_?$ are both 0, $(k_!, k'_!)$ are a quotient and a remainder of w_{12}, and $(k_?, k'_?)$ are a quotient and a remainder of w_{21}.*
6. *at every position, $(k_!, k'_!) \leq (k_?, k'_?)$ for the lexicographic order.*

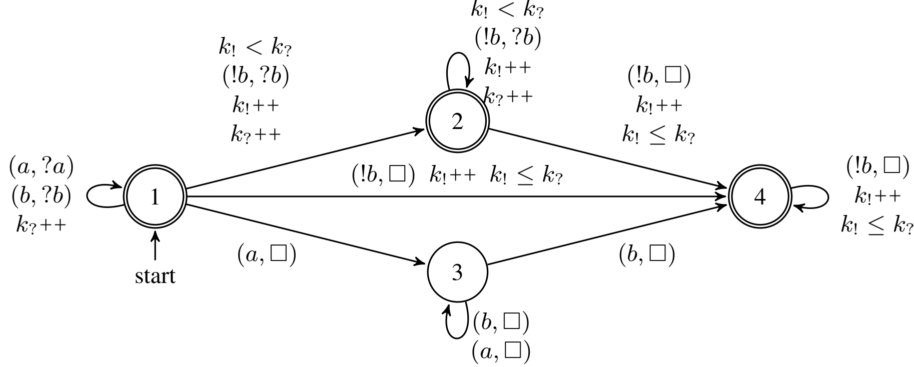
Let us give some further intuitions about these conditions: a bi-run will be mapped to an indexed word by erasing the prefix in $(\Sigma \cup \Sigma_!) \times \Sigma_?$ and mapping the suffix in $\Sigma \cup \Sigma_! \times \{\Box\}$ to Σ, which will be the content of the queue at the end of acceleration. Then the conditions in the definition of the bi-run can be understood as follows:

1. the resulting queue content is a suffix of the concatenation $w_{11}w_{12}$ of an original queue content and a word queued by a sequence of ! instructions potentially triggered with respect to the phases-loop;
2. the prefix that should be erased in $w_{11}w_{12}$ is a word w_{21} dequeued by a sequence of ? instructions potentially trigerred with respect to the phases-loop;
3. w_{21} is indeed a prefix of $w_{11}w_{12}$;
4. this condition is the definition of counters $k_\dagger$;
5. w_{12} and w_{21} correspond to the same number of loop iterations;
6. each receive matches a send that happened earlier in the original run of the CQS (or a piece of the initial contents of the queue); this last condition is essential to check that an "hungry snake execution" is effectively possible.

Example 4.2. Consider again the CQS of Fig 1b. Let $\mathcal{L}_c$ be the phases-loop as defined before, and let $L_0 = (ab)^*$ be the language of the initial content of c. Let moreover R be the set of well-synchronized bi-runs with respect to $\mathcal{L}_c$ and L_0. Then R is recognized by the automaton depicted in Fig 3 with accepting condition $k_! = k_?$ (we omit counters $k'_!$ and $k'_?$). For the purpose of the example, we allow transitions of this automaton to perform several operations at once, and to test the values of the counters. Sec. 4.2 is devoted to proving that well-synchronized bi-runs are in fact PFA recognizable.

Consider the homomorphism ϕ that associates to a bi-run ρ the indexed word $\phi(\rho) = (w', i_c)$ such that $i_c = k_! = k_?$ and w' is obtained by replacing in $w(\rho)$ the letters of the form $(a, \Box)$ or $(!a, \Box)$ by a and all the other ones by ε. Then:

Figure 3 An automaton recognizing well-synchronized bi-runs.



Lemma 15 *Let (w, i_c) be a $\{i_c\}$-indexed word. Then the following are equivalent:*

1. *there exists a well-synchronized bi-run ρ w.r.t. $\mathcal{L}$ and L_0 such that $\phi(\rho) = (w, i_c)$;*
2. $(w, i_c) \in \mathsf{PPost}_{\mathcal{L}}(L_0)$.

The proof of this result relies on the following auxiliary lemmata:

Lemma 16 *Let Σ_0, Σ_1 be two alphabets with $\{0, 1\} \notin \Sigma_0 \cup \Sigma_1$ and $\Sigma_0 \cap \Sigma_1 = \emptyset$. For $i = 0, 1$, let $\pi_i : (\Sigma_0 \cup \Sigma_1) \to \Sigma_i \cup \{1 - i\}$ be the morphism that sends $a \in \Sigma_{1-i}$ on letter $1 - i$, and is the identity over Σ_i. Let $s, w_{\mathcal{L}} \in (\Sigma_0 \cup \Sigma_1)^*$. Then the following statements are equivalent:*

1. $s \in w_{\mathcal{L}}^*$
2. $\pi_0(s) \in \pi_0(w_{\mathcal{L}}^*)$ *and* $\pi_1(s) \in \pi_1(w_{\mathcal{L}}^*)$.

Proof Straightforward. $\square$

Lemma 17 *Let $\to \subseteq \left((\{!, ?\} \times \Sigma)^*\right)^2$ be the rewriting system over sequences of instructions with rewrite rules $!a?b \to ?b!a$ for all $a, b \in \Sigma$. Let $\theta, \theta' \in (\{!, ?\} \times \Sigma)^*$ be such that $\theta \to \theta'$. Then for all $w \in \Sigma^*$, there is $w' \in \Sigma^*$ such that $\mathsf{Post}_{\theta'}(\{w\}) \subseteq \mathsf{Post}_{\theta}(\{w\}) \subseteq \{w'\}$.*

Proof Straightforward. $\square$

Proof *(of Lemma 15)*

Implication (1)⇒(2). Let ρ be a well-synchronized bi-run. Let p be a position of a read label in ρ where a send instruction $!a$ appears on the first component of the read letter. Let $k_{p,!a} = |\mathcal{L}|.k_! + k'_!$ be the phase number obtained from the quotient and the remainder at this position p, where $|\mathcal{L}|$ is the number of phases in $\mathcal{L}$. We also define the phase number $k_{p,?a}$ of the receive instruction appearing on position p of ρ. Let $\prec$ be the order on send and receive instructions appearing in ρ such that $(p, \dagger a) \prec (p', \dagger' a')$

iff either $k_{p,\dagger a} < k_{p',\dagger' a'}$, or $k_{p,\dagger a} = k_{p',\dagger' a'}$ and $p < p'$. Finally, let $\theta \in (\Sigma \times \{!, ?\})^*$ be the sequence of instructions obtained by ordering all instructions $\dagger a$ appearing in ρ according to $\prec$.

- Fact 1: θ is a sequence of instructions in $\mathcal{L}^*$.
  θ is $\theta_1 \dots \theta_m$, where θ_i is the sequence of instructions with phase number n_i, and $n_1, \dots, n_m$ is an increasing sequence. Let r_i be the remainder of i modulo $|\mathcal{L}|$. For each θ_i, the first or second projection of $w(\rho)$ contains θ_i as a sub-word delimited by phase-switching instructions. So, by conditions 1 and 2, each θ_i is a sequence of instructions potentially triggered in the r_ith phase of the loop. As a consequence, θ is in $\mathcal{L}^*$ if and only if the sequence $s \in \{0, \dots, |\mathcal{L}| - 1\}^*$ of remainders $r_1 \dots r_m$ is in $w_{\mathcal{L}}^*$, where $w_{\mathcal{L}} = 0.1 \dots (|\mathcal{L}| - 1)$. Let Σ_0 be the set of r_i corresponding to send phases, and Σ_1 the set of r_i corresponding to receive phases. By condition 1, the subsequence of remainders corresponding to ! phases in s is a sequence of remainders of $\mathcal{L}_!^*$, and by condition 2, the subsequence of ? phases in s is a sequence of remainders of $\mathcal{L}_?^*$. Moreover, by condition 4 and 5, between two consecutive ! phases, there are exactly as many ? phases as constrained by $\mathcal{L}$, and vice versa between two ? phases. So we may apply Lemma 16, and as a consequence, $s \in w_{\mathcal{L}}^*$, hence $\theta \in \mathcal{L}^*$.
- Fact 2 : when θ is applied to w_{11}, it results in the word $\phi(\rho)$.
  Let θ' be the sequence of instructions obtained from ρ by the morphism sending $(!a, ?a)$ to $!a.?a$, $(a, ?a)$ to $?a$, and $(!a, \square)$ to $!a$. Then by condition 6, $\theta \to^* \theta'$ for $\to$ being the rewriting system of Lemma 17. Moreover, $\phi(\rho)$ is the result of applying θ' on w_{11}, hence the claim.

As a consequence of these two facts, and since $w_{11} \in L_0$ by condition 1, $\phi(\rho) \in \mathsf{PPost}_{\mathcal{L}}(L_0)$.

Implication (2)$\Rightarrow$(1). Conversely, let $w_{11} \in L_0$ and $w \in \mathsf{PPost}_{\mathcal{L}}(L_0)$ be obtained from $\theta \in \mathcal{L}^*$ once executed over initial queue w_{11}. Let ρ be the bi-run obtained by grouping receive instructions with their matching initial letter or send instruction, and counters $k_!, k_?, k'_!, k'_?$ correctly updated before starting any new phase. Then ρ is well-synchronized. Let θ' be the sequence of instructions defined from ρ as before. Then $\phi(\rho)$ is the result of applying θ', and since $\theta \to^* \theta'$, the result of applying θ as well, hence $\phi(\rho) = w$. $\square$

4.2 Recognizing Well-Synchronized Bi-Runs

Since PFA recognizable languages are stable by homomorphism, it is sufficient to prove that well-synchronized bi-runs are PFA recognizable for establishing that $\mathsf{PPost}_{\mathcal{L}}(L_0)$ is PFA recognizable. It is easy to check that all conditions in Def. 14 are PFA definable, except for condition 6. Indeed, the naive encoding for the latter using tests on counters during the recognition process is not supported by PFA (see previous example): condition 6 expresses a constraint on all positions of the run, although PFA can only express constraints on the final values of the counters at the end of the run. We hence introduce extra counters, and express condition 6 by other conditions on the final values of these extra counters.

Definition 18 *A bi-run ρ is said to be a* candidate *w.r.t. $\mathcal{L}$ and L_0 iff the first 5 conditions of Def 14 hold, and there are additional counters $k_{1?}$, $k_{2?}$, $k_{1!}$, k_Δ and $\overline{k_\Delta}$ such that*

1. *$k_{1?} = k_?$ at every position of a letter in $\Sigma \times \Sigma_?$, and $k_{1?}$ is constant afterwards;*
2. *$k_{2?}$ such that $k_{2?} = 0$ at every position of a letter in $\Sigma \times \Sigma_?$, and $k_{2?} = k_? - k_{1?}$ afterwards;*
3. *$k_{1!} = k_!$ at every position of a letter in $(\Sigma \cup \Sigma_!) \times \Sigma_?$ and is constant afterwards;*
4. *$k_\Delta = k_{1!} - k_{2?}$ at every position;*
5. *for all position p, the value of $\overline{k_\Delta}$ at p is the max of the values of k_Δ at positions $p' \leq p$;*

Example 4.3. The figure below represents the extra counters of the candidate bi-run obtained from the bi-run of the run $?a?b!b?a?b!b?b!b?b!b$ detailed at the beginning of Sec. 4.1:

$k_!$	0	0	0	0	1	2	3	4
$k_{1!}$	0	0	0	0	1	2	2	2
	a	b	a	b	$!b$	$!b$	$!b$	$!b$
	$?a$	$?b$	$?a$	$?b$	$?b$	$?b$	$\square$	$\square$
$k_{2?}$	0	0	0	0	1	2	3	3
$k_{1?}$	0	0	1	1	1	1	1	1
$k_?$	0	0	1	1	2	3	4	4
k_Δ	0	0	0	0	0	0	-1	-1
$\overline{k_\Delta}$	0	0	0	0	0	0	0	0

It can be shortly observed that all these extra counters, but $\overline{k_\Delta}$, are easily definable in the PFA model. Before we explain how to encode this counter, let us first precise how we can identify well-synchronized bi-runs among candidate bi-runs using these extra counters:

Lemma 19 *A candidate bi-run ρ is well synchronized iff the following conditions hold:*

1. *the final values of $k_{1?}$ and $\overline{k_\Delta}$ are such that $\overline{k_\Delta} \leq k_{1?}$;*
2. *for all positions where $k_\Delta = k_{1?}$, it holds that $k'_! \leq k'_?$.*

Proof Let us first prove that condition 14.6 of Def. 14 implies 19.1 and 19.2.

1. from the definitions, $k_?$ and $k_!$ are non-decreasing, $k_! \geq k_{1!}$ and from 18.1-2 and 14.2, $k_? = k_{1?} + k_{2?}$. Condition 14 thus becomes $k_\Delta \leq k_{1?}$ at every position. The fact that $k_{1?}$ is non-decreasing finally gives us that, at the end of the run, $\overline{k_\Delta} \leq k_{1?}$.
2. direct consequence of 14.6 and the definitions of the auxiliary counters.

Let us now prove that 19.1 and 19.2 implies 14.6 Again, 19.1 asserts that $k_\Delta \leq k_{1?}$ at any position, that is $k_! \leq k_?$. So, 14.6 is equivalent to 19.2. $\square$

As mentioned before, $\overline{k_\triangle}$ does not seem PFA definable a priori. However, its behavior is very constrained. Let us use the hungry snake image one more time: $k_\triangle$ represents the length of the tail of the snake up to some constant. During the hungry snake mode, either the snake grows quicker as it eats, and $k_\triangle$ is increasing, or it eats as much as it grows, and $k_\triangle$ is constant (as in the previous example, where one b is queued and one b is dequeued at each loop iteration), or it eats more than it grows, and $k_\triangle$ is decreasing. The important observation is intuitively that, due to the regularity of the sequence of phase-switching messages, the snake has only one of these three behaviors during the whole run, and this is fixed by the phases-loop considered. Let us put it more formally:

Lemma 20 *If ρ is a candidate bi-run that is also well-synchronized, then there is a position p such that $k_\triangle \in \{0, -1\}$ for all position before p, and either one of the conditions below is satisfied:*

(inc) for all position after p, it holds that $k_\triangle \geq \overline{k_\triangle} - 1$
(dec) for all position after p, it holds that $k_\triangle \leq -1$

Proof Let us consider the position p_0 in ρ of the first letter in $\Sigma_! \times \Sigma_?$. Let $w_{!?}$ be the word of phase-switching messages that are queued and latter dequeued, that is the word of phase-switching messages that is read from position p_0 to the last letter in $\Sigma_! \times \Sigma_?$. Let $a_1, \ldots, a_u$ and $a'_1, \ldots a'_v$ be the phase-switching messages such that $\mathcal{L}_! = c!^* a_1 \ldots c!^* a_u$ and $\mathcal{L}_? = c?^* a'_1 \ldots c?^* a'_v$. Let $w_! = a_1 \ldots a_u$ be the word of phase-switching messages that are queued during one iteration of the loop starting at the beginning of the loop. Let $j \in \{1, \ldots, |\mathcal{L}_?|\}$ be the remainder in $\mathcal{L}_?$ obtained from the value of $k'_?$ at position p, and $w_? = a'_j a'_{j+1} \ldots a'_v a'_1 \ldots a'_{j-1}$ the word of phase-switching messages that are dequeued during one iteration of the loop from phase j to phase $j-1$ of $\mathcal{L}_?$. Then $w_{!?}$ is both $w_?^n$ for some n and a prefix of $w_!^m$ for some m. Let us reason with such a minimal m. Then while queuing messages that are dequeued, m or $m-1$ increments of $k_\triangle$ happen periodically, and while dequeuing messages that were queued, n decrements of $k_\triangle$ happen periodically. There are then two possibilities:

- $n \leq m$: then between two consecutive decrements of $k_\triangle$, there is at least $\lfloor \frac{m}{n} \rfloor \geq 1$ increment, so (inc) holds for the position $p = p_0$.
- $n \geq m$: symmetrically, between two consecutive increments there is at least one decrement. (dec) holds for the position p just before the first time $k_\triangle = -2$, or any position if $k_\triangle$ does not reach -2. □

Lemma 21 *The language of well-synchronized bi-runs is PFA definable.*

Proof By Lemma 19, we only need to prove that the language of candidate bi-runs and the conditions of Lem. 19 are PFA definable. It is clear that conditions 14.1-5 and 18.1-4 are PFA definable using phase-switching words to identify when counters $k_\dagger, k'_\dagger$ should be updated. Provided $\overline{k_\triangle}$ is PFA definable, condition 18.6 is also clearly PFA definable. We now detail the two remaining conditions

- condition 18.5: the automaton guesses the behavior of $k_\triangle$ and the position p as in Lemma 20. Before the position p, the value of $\overline{k_\triangle}$ is 0 and does not require any care. Afterwards, either the (dec) behavior was guessed and no special care is needed, or the (inc) behavior was guessed, and then $\overline{k_\triangle}$ is incremented synchronously with

k_Δ whenever the previous update of k_Δ was an increment, and this only requires finite memory. It can be also detected that k_Δ violates Lemma 20, in which case the word is rejected.

– condition 19.2: the tricky part is to determine, for a position p, whether the test $k'_! \leq k'_?$ is needed or not, but the test itself is straightforward since $k'_!$ and $k'_?$ are finite counters. Due to Lemma 20 (see the proof) and condition 6f, k_Δ goes at most once below $k_{1?}$ after the first time it has reached it. It is then sufficient to guess the first position p_1 where $k_\Delta = k_{1?}$, the position $p_2 \geq p_1$ where $k_\Delta = k_{1?} - 1$ (if guessed so), and the position $p_3 \geq p_2$ where $k_\Delta = k_{1?}$ again. Saving the value of the counter at these positions, it can be checked at the end of the run that the guess was correct. □

4.3 Main theorem

Let us summarize what we have proved so far:

Lemma 22 *If $\mathcal{L}$ is a phases-loop of a CQS using only one queue with messages in Σ and no counter and L_0 is a PFA recognizable language over Σ, then $\mathsf{PPost}_{\mathcal{L}}(L_0)$ is effectively PFA recognizable.*

Proof Straightforward consequence of Lemma 15, Lemma 21, and of the stability of PFA recognizable languages by homomorphisms. □

We can add counter instructions in $\mathcal{L}$ and keep the same result. Let $\mathcal{L}$ be a phases-loop of a CQS using only one queue, and manipulating a set of counters K. Let $\mathsf{Post}_{\mathcal{L}}$: $2^{\Sigma^* \times \mathbb{Z}^K} \to 2^{\Sigma^* \times \mathbb{Z}^K}$ be the function that associates to a set of K-indexed words the set of K-indexed words obtained after one phases-loop iteration.

The parametrized transitive closure $\mathsf{PPost}_{\mathcal{L}}$ of $\mathsf{Post}_{\mathcal{L}}$ is the function $2^{\Sigma^* \times \mathbb{Z}^K} \to 2^{\Sigma^* \times \mathbb{Z}^{K \uplus \{i_c\}}}$ that associates to a K-indexed language L_0 the $K \uplus \{i_c\}$-indexed language of indexed words $(w, \mathbf{n}, n)$ such that $(w, \mathbf{n}) \in \mathsf{Post}^n_{\mathcal{L}}(L_0)$. Then:

Lemma 23 *Let L_0 be a PFA recognizable language. Then $\mathsf{PPost}_{\mathcal{L}}(L_0)$ is effectively PFA recognizable.*

Proof Let $T_\dagger$ be the set of transitions with counter update occurring in a † phase. Let $\Sigma' = \Sigma \cup T_! \cup T_?$. Let $K' = K \cup K_T$, where K_T contains a counter k_t for each transition in $T_! \cup T_?$. and π the morphism that erases letters in $T_! \cup T_?$, and stores in k_t the number of occurrences of t. Finally, let $\mathcal{L}'$ be the phases-loop without counter updates obtained by replacing, for each transition t in $T_\dagger$, the counter update by $\dagger t$ (hence $\mathcal{L}'$ is a phases-loop). Then $\mathsf{PPost}_{\mathcal{L}}(L_0) = f(\pi(\mathsf{PPost}_{\mathcal{L}'}(\pi^{-1}(L_0))))$, where f is the homomorphism that computes the final value of k in K by incrementing its initial value with the weighted sum of the number of times each transition incrementing k has been triggered. □

Note that this result could have been derived directly, considering counter instructions everywhere in previous sections, but, for the sake clarity, we avoided to confuse "machine" counters with the internal counters used by the recognition of bi-runs.

We now have all the ingredients for establishing our main result:

Theorem 24 *SCQ supports phases-loop acceleration: for any CQS $\mathcal{M}$, if $\mathcal{S}$ is a SCQ, then $\mathsf{Acc}_{\mathcal{M}}([\![\mathcal{S}]\!])$ can be represented by an effective SCQ.*

Proof Let $\mathcal{L} \subseteq \mathcal{M}$ be a phases-loop, and $(\mathbf{A}, \Phi)$ an ASCQ. For all channel c, let $\mathcal{L}_c$ be the phases-loop obtained from $\mathcal{L}$ by replacing all the actions that are not in $c!$ or $c?$ phases by ε transitions. Let (A'_c, Φ_c) be the PFA that defines $\mathsf{PPost}_{\mathcal{L}_c}(L(A_c))$ obtained by Lemma 23, for some fresh counter i_c counting the number of loop iterations with regard to channel c. Then $\mathsf{Acc}_{\mathcal{M},\mathcal{L}}(\mathbf{A}, \Phi)$ is represented by the ASCQ $(\mathbf{A}', \Phi')$, where Φ' expresses that the initial values of the counters in K satisfy Φ, that their final values are the initial values incremented by the sum, over all channels c, of increments obtained while accelerating c, and that all i_c are equal. $\square$

Conclusion

We introduced a new acceleration technique for queue systems with counters, which terminates for a large class of systems, among which the classes of flat, half-duplex, slack-bounded, and phases-flat machines. We illustrated our approach on a case study that, to our knowledge, could not be proved by any previous automatic methods. We are currently implementing our method and we will use it to ensure deadlock and leak freedom in Heaphop [11]. First experiments let us expect that, even with some naive implementations of automata, examples like the ones we described will be handled in a reasonable time. We actually hope to keep all automata constructions in polynomial time, since we never need to determinize nor complement. However, language inclusion is still an open issue for which we only sketched here a possible approach, and experimental evaluation of our technique is still matter for future work.

References

1. A. Muscholl A. Heussner, J. Leroux and G. Sutre. Reachability analysis of communicating pushdown systems. To Appear in FOSSACS'10.
2. M. Faouzi Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *Proceedings of POPL 2010*, pages 7–18. ACM.
3. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo channel systems with nonregular sets of configurations (extended abstract). In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, pages 560–570. Springer.
4. G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
5. M. Fähndrich, M. Aiken, C. Hawblitzel, O. Hodson, G. C. Hunt, J. R. Larus, and S. Levi. Language Support for Fast and Reliable Message-Based Communication in Singularity OS. In *EuroSys*, 2006.
6. D. Fisman and A. Pnueli. Beyond regular model checking. In *Proceedings of FSTTCS 2001*, volume 2245 of *LNCS*, pages 156–170. Springer, 2001.
7. O. H. Ibarra. Reversal-bounded multi-counter machines and their decision problems. *JACM*, 25:116–133, 1978.
8. F. Klaedtke and H. Rueß. Monadic second-order logics with cardinalities. In *Proceedings of ICALP 2003*, volume 2719 of *LNCS*, pages 681–696. Springer, 2003.

9. S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *Proceedings of TACAS 2008*, volume 4963 of *LNCS*, pages 299–314. Springer.
10. J. Villard, É. Lozes, and C. Calcagno. Proving copyless message passing. In *Proceedings of APLAS 2009*, volume 5904 of *LNCS*, pages 194–209. Springer.
11. J. Villard, É. Lozes, and C. Calcagno. Tracking heaps that hop with Heap-Hop. To appear in TACAS' 2010.