Extrapolation-based Path Invariants for Abstraction Refinement of Fifo Systems

Alexander Heußner[1], Tristan Le Gall[2], and Grégoire Sutre[1]

[1] LaBRI, Université Bordeaux, CNRS {sutre,heussner}@labri.fr
[2] Université Libre de Bruxelles (ULB) tlegall@ulb.ac.be

Abstract. The technique of counterexample-guided abstraction refinement (CEGAR) has been successfully applied in the areas of software and hardware verification. Automatic abstraction refinement is also desirable for the safety verification of complex infinite-state models. This paper investigates CEGAR in the context of formal models of network protocols, in our case, the verification of fifo systems. Our main contribution is the introduction of *extrapolation-based path invariants* for abstraction refinement. We develop a range of algorithms that are based on this novel theoretical notion, and which are parametrized by different extrapolation operators. These are utilized as subroutines in the refinement step of our CEGAR semi-algorithm that is based on recognizable partition abstractions. We give sufficient conditions for the termination of CEGAR by constraining the extrapolation operator. Our empirical evaluation confirms the benefit of extrapolation-based path invariants.

1 Introduction

Distributed processes that communicate over a network of reliable and unbounded fifo channels are an important model for the automatic verification of client-server architectures and network protocols. In this paper, we focus on communicating fifo systems that consist of a set of finite automata that model the processes, and a set of reliable, unbounded fifo queues that model the communication channels. This class of infinite-state systems is unfortunately Turing-complete even in the case of one fifo queue [BZ83]. In general, two approaches for the automatic verification of Turing-complete infinite-state models have been considered in the literature: (a) exact semi-algorithms that compute forward or backward reachability sets (e.g., [BG99, BH99, FIS03] for fifo systems) but may not terminate, and (b) algorithms that always terminate but only compute an over-approximation of these reachability sets (e.g., [LGJJ06, YBCI08] for fifo systems). In the last decade, **c**ounter**e**xample-**g**uided **a**bstraction **r**efinement [CGJ+03] has emerged as a powerful technique that bridges the gap between these two approaches. CEGAR plays a prominent role in the automatic, iterative approximation and refinement of abstractions and has been applied successfully in the areas of software [BR01, HJMS02] and hardware verification [CGJ+03]. Briefly, the CEGAR approach to the verification of a safety property utilizes an abstract–check–refine loop that searches for a counterexample in a conservative

over-approximation of the original model, and, in the case of finding a false negative, refines the over-approximation to eliminate this spurious counterexample.

Our Contribution. We present a CEGAR semi-algorithm for safety verification of fifo systems based on finite partition abstractions where equivalence classes are recognizable languages of queue contents, or, equivalently, QDDs [BG99]. The crucial part in CEGAR-based verification is refinement, which must find a new partition that is both (1) precise enough to rule out the spurious counterexample and (2) computationally "simple". In most techniques, refinement is based on the generation of *path invariants*; these are invariants along the spurious counterexample that prove its unfeasibility (in our case, given by a series of recognizable languages). We follow this approach, and present several generic algorithms to obtain path invariants based on parametrized extrapolation operators for queue contents. Our path invariant generation procedures are fully generic with respect to the extrapolation. Refining the partition consists in splitting abstract states occurring on the counterexample with the generated path invariant.

We formally present the resulting CEGAR semi-algorithm and give partial termination results that, in contrast to the classical CEGAR literature, do not rely on an "a priori finiteness condition" on the set of all possible abstractions. Actually, our results depend mainly on our generic extrapolation-based path invariant generation. In particular we show that our semi-algorithm always terminates if (at least) one of these conditions is satisfied: (1) the fifo system under verification is unsafe, or (2) it has a finite reachability set and the parametrized extrapolation has a finite image for each value of the parameter.

We have implemented our approach in the tool MCSCM [McS] that performs CEGAR-based safety verification of fifo systems. Experimental results on a suite of (simple) network protocols allow for a first discussion of our approach's advantages.

Related Work. Exact semi-algorithms for reachability set computations of fifo systems usually apply *acceleration techniques* [BG99, BH99, FIS03] that, intuitively, compute the effect of iterating a given "control flow" loop. The tools LASH [Las] (for counter/fifo systems) and TREX [TRe] (for lossy fifo systems) implement these techniques. However, recognizable languages equipped with Presburger formulas (CQDDs [BH99]) are required to represent (and compute) the effect of so-called *counting* loops [BG99, FIS03]. Moreover such tools may only terminate when the fifo system can be flattened into an equivalent system without nested loops. Our experiments show that our approach can cope with both counting loops and nested loops that cannot be flattened.

The closest approach to ours is ***a**bstract **r**egular **m**odel **c**hecking* [BHV04], an extension of the generic regular model-checking framework based on the abstract–check–refine paradigm. As in classical regular model-checking, a system is modeled as follows: configurations are words over a finite alphabet and the transition relation is given by a finite-state transducer. The analysis consists in an over-approximated forward exploration (by Kleene iteration), followed, in case of a non-empty intersection with the bad states, by an exact backward computation along the reached sets. Two parametrized automata abstraction

schemes are provided in [BHV04], both based on state merging. These schemes fit in our definition of extrapolation, and therefore can also be used in our framework. Notice that in ARMC, abstraction is performed on the data structures used to represent sets of configurations, whereas in our case the system itself is abstracted. After each refinement step, ARMC restarts (from scratch) the approximated forward exploration from the refined reached set, whereas our refinement is *local* to the spurious counterexample path. Moreover, the precision of the abstraction is *global* in ARMC, and may only increase (for the whole system) at each refinement step. In contrast, our path invariant generation procedures only use the precision *required* for each spurious counterexample, and ongoing benchmarks encourage our adaptive approach as in most protocols (un-)safety depends solely on a "highly" precise abstraction of few control loops. Last, but not least, our approach is not tied to words and automata. We only focus in this work on fifo systems, but our framework is fully generic and could be applied to other infinite-state systems (e.g., hybrid systems), provided that suitable parametrized extrapolations are designed (e.g., on polyhedra).

Outline. We recapitulate fifo systems in Section 2 and define their partition abstractions in Section 3. Refinement and extrapolation-based generation of path invariants are developed in Section 4. In Section 5, we present the general CEGAR semi-algorithm, and analyse its correctness and termination. Experimental results are presented in Section 6, along with some perspectives.

2 Fifo Systems

This section presents basic definitions and notations for fifo systems that will be used throughout the paper. For any set S we write $\wp(S)$ for the set of all subsets of S, and S^n for the set of n-tuples over S (when $n \geq 1$). For any $i \in \{1, \ldots, n\}$, we denote by $\boldsymbol{s}(i)$ the i^{th} *component* of an n-tuple $\boldsymbol{s}$. Given $\boldsymbol{s} \in S^n$, $i \in \{1, \ldots, n\}$ and $u \in S$, we write $\boldsymbol{s}[i \leftarrow u]$ for the n-tuple $\boldsymbol{s'} \in S^n$ defined by $\boldsymbol{s'}(i) = u$ and $\boldsymbol{s'}(j) = \boldsymbol{s}(j)$ for all $j \in \{1, \ldots, n\}$ with $j \neq i$. Let Σ denote an *alphabet* (i.e., a finite, non empty set). We write Σ^* for the set of all *finite words* (*words* for short) over Σ. The *empty word* is written ε and we denote by Σ^+ the set $\Sigma^* \setminus \{\varepsilon\}$. For any two words $w, w' \in \Sigma^*$, we write $w \cdot w'$ for their *concatenation*. A *language* is any subset of Σ^*. For singleton languages, we will sometimes omit the curly brackets, e.g., $\{w\} \subseteq \Sigma^*$ will be written simply as word w when no confusion is possible.

Safety Verification of Labeled Transition Systems. We will use labeled transition systems to formally define the behavioral semantics of fifo systems. A *labeled transition system* is any triple $LTS = (\mathcal{C}, \Sigma, \rightarrow)$ where $\mathcal{C}$ is a set of *configurations*, Σ is a finite set of *actions* and $\rightarrow \subseteq \mathcal{C} \times \Sigma \times \mathcal{C}$ is a (labeled) *transition relation*. We often simply write $c \xrightarrow{l} c'$ when $(c, l, c') \in \rightarrow$.

A *finite path* (*path* for short) in LTS is any pair $\pi = (c, u)$ where $c \in \mathcal{C}$, and u is either the empty sequence, or a non-empty finite sequence of transitions $(c_0, l_0, c'_0), \ldots, (c_{h-1}, l_{h-1}, c'_{h-1})$ such that $c_0 = c$ and $c'_{i-1} = c_i$ for every $0 <$

$i < h$. We simply write π as $c_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} c_h$. The natural number h is called the *length* of π. We say that π is a *simple path* if $c_i \neq c_j$ for all $0 \leq i < j \leq h$. For any two sets $Init \subseteq \mathcal{C}$ and $Bad \subseteq \mathcal{C}$ of configurations, a *path from $Init$ to Bad* is any path $c_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} c_h$ such that $c_0 \in Init$ and $c_h \in Bad$. Observe that if $c \in Init \cap Bad$ then c is a path (of zero length) from $Init$ to Bad. The *reachability set* of LTS from $Init$ is the set of configurations c such that there is a path from $Init$ to $\{c\}$.

In this paper, we focus on the verification of safety properties on fifo systems. A safety property is in general specified as a set of "bad" configurations that should not be reachable from the initial configurations. Formally, a *safety condition* for a labeled transition system $LTS = (\mathcal{C}, \Sigma, \rightarrow)$ is a pair $(Init, Bad)$ of subsets of $\mathcal{C}$. We say that LTS is $(Init, Bad)$*-unsafe* if there is a path from $Init$ to Bad in LTS, which is called a *counterexample*. We say that LTS is $(Init, Bad)$*-safe* when it is not $(Init, Bad)$-unsafe.

Fifo Systems. The asynchronous communication of distributed systems is usually modeled as a set of local processes together with a network topology given by channels between processes. Each process can be modeled by a finite-state machine that sends and receives messages on the channels to which it is connected. Let us consider a classical example, which will be used in the remainder of this paper to illustrate our approach. The connection/disconnection protocol [JR86] – abbreviated c/d protocol – between two hosts is depicted in Figure 1. This model is composed of two processes, a client and a server, as well as two unidirectional channels.

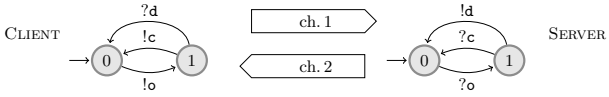

Fig. 1: The Connection/Disconnection Protocol [JR86]

To simplify the presentation, we restrict our attention to the case of one finite-state control process. The general case of multiple processes can be reduced to this simpler form by taking the asynchronous product of all processes. For the connection/disconnection protocol, the asynchronous product of the two processes is depicted in Figure 2.

Definition 2.1. *A* fifo system $\mathcal{A}$ *is a* 4*-tuple* $\langle Q, M, n, \Delta \rangle$ *where:*

- Q *is a finite set of* control states*,*
- M *is a finite alphabet of* messages*,*
- $n \geq 1$ *is the number of fifo queues,*
- $\Delta \subseteq Q \times \Sigma \times Q$ *is a set of* transition rules*,*
  where $\Sigma = \{1, \ldots, n\} \times \{!, ?\} \times M$ *is the set of* fifo actions *over* n *queues.*

Simplifying notation, fifo actions in Σ will be shortly written $i!m$ and $i?m$ instead of $(i, !, m)$ and $(i, ?, m)$. The intended meaning of fifo actions is the following: $i!m$ means "emission of message m in queue i" and $i?m$ means "reception

of message m from queue i". The operational semantics of a fifo system $\mathcal{A}$ is formally given by its associated labeled transition system $[\![\mathcal{A}]\!]$ defined as follows.

Definition 2.2. *The* operational semantics *of a fifo system $\mathcal{A} = \langle Q, M, n, \Delta\rangle$ is the labeled transition system $[\![\mathcal{A}]\!] = \langle \mathcal{C}, \Sigma, \rightarrow\rangle$ defined as follows:*

- $\mathcal{C} = Q \times (M^*)^n$ *is the set of configurations,*
- $\Sigma = \{1, \ldots, n\} \times \{!, ?\} \times M$ *is the set of actions,*
- *the transition relation $\rightarrow \subseteq \mathcal{C} \times \Sigma \times \mathcal{C}$ is the set of triples $((q, \boldsymbol{w}), l, (q', \boldsymbol{w'}))$ such that $(q, l, q') \in \Delta$ and which satisfies the two following conditions:*
  - *if $l = i!m$ then $\boldsymbol{w'}(i) = \boldsymbol{w}(i) \cdot m$ and $\boldsymbol{w'}(j) = \boldsymbol{w}(j)$ for all $j \neq i$,*
  - *if $l = i?m$ then $\boldsymbol{w}(i) = m \cdot \boldsymbol{w'}(i)$ and $\boldsymbol{w'}(j) = \boldsymbol{w}(j)$ for all $j \neq i$.*

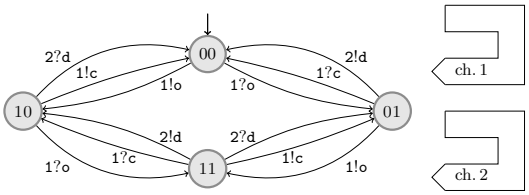

Fig. 2: Fifo System Representing the Connection/Disconnection Protocol

Example 2.3. The fifo system $\mathcal{A} = \langle\{\mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{11}\}, \{\mathtt{o}, \mathtt{c}, \mathtt{d}\}, 2, \Delta\rangle$ that corresponds to the c/d protocol is displayed in Figure 2. A set of bad configurations for this protocol is the set $Bad = \{\mathtt{00}, \mathtt{10}\} \times (\mathtt{c} \cdot M^* \times M^*)$. This set contains configurations where the server is in control state `0` but the first message in the queue is `close`. This is the classical case of an *undefined reception* which results in a (local) *deadlock* for the server. Setting the initial configuration to $c_0 = (\mathtt{00}, \varepsilon, \varepsilon)$, a counterexample to the safety condition $(\{c_0\}, Bad)$ is the path $(\mathtt{00}, \varepsilon, \varepsilon) \xrightarrow{1!\mathtt{o}} (\mathtt{10}, \mathtt{o}, \varepsilon) \xrightarrow{1?\mathtt{o}} (\mathtt{11}, \varepsilon, \varepsilon) \xrightarrow{2!\mathtt{d}} (\mathtt{10}, \varepsilon, \mathtt{d}) \xrightarrow{1!\mathtt{c}} (\mathtt{00}, \mathtt{c}, \mathtt{d})$ in $[\![\mathcal{A}]\!]$. □

3 Partition Abstraction for Fifo Systems

In the context of CEGAR-based safety verification, automatic abstraction techniques are usually based on predicates [GS97] or partitions [CGJ+03]. In this work, we develop partition-based abstraction and refinement techniques for fifo systems. A *partition* of a set S is any set P of non-empty pairwise disjoint subsets of S such that $S = \bigcup P$. Elements of a partition P are called *classes*. For any $s \in S$, we write $[\, s \,]_P$ for the class containing s.

At the labeled transition system level, partition abstraction consists of merging configurations that are equivalent with respect to a given equivalence relation, or a given partition. In practice, it is often desirable to maintain different partitions for different control states, to keep partition sizes relatively small. We follow this approach in our definition of partition abstraction for fifo systems, by associating a partition of $(M^*)^n$ with each control state. To ease notation, we write $\overline{L} = (M^*)^n \setminus L$ the *complement* of any subset L of $(M^*)^n$.

To effectively compute partition abstractions for fifo systems, we need a family of finitely representable subsets of $(M^*)^n$. A natural candidate is the class of recognizable subsets of $(M^*)^n$, or, equivalently, of QDD-definable subsets of $(M^*)^n$ [BG99], since this class is effectively closed under Boolean operations. Recall that a subset L of $(M^*)^n$ is *recognizable* if (and only if) it is a finite union of subsets of the form $L_1 \times \cdots \times L_n$ where each L_i is a regular language over M [Ber79]. We extend recognizability in the natural way to subsets of the set $\mathcal{C} = Q \times (M^*)^n$ of configurations. A subset $C \subseteq \mathcal{C}$ is *recognizable* if $\{\boldsymbol{w} \mid (q, \boldsymbol{w}) \in C\}$ is recognizable for every $q \in Q$. We denote by $\mathcal{R}ec\,((M^*)^n)$ the set of recognizable subsets of $(M^*)^n$, and write $\mathbb{P}\,((M^*)^n)$ for the set of all finite partitions of $(M^*)^n$ where classes are recognizable subsets of $(M^*)^n$.

Definition 3.1. *Consider a fifo system $\mathcal{A} = \langle Q, M, n, \Delta\rangle$ and a* partition map *$P : Q \to \mathbb{P}\,((M^*)^n)$. The* partition abstraction *of $[\![\mathcal{A}]\!]$ induced by P is the labeled transition system $[\![\mathcal{A}]\!]^\sharp_P = \langle \mathcal{C}^\sharp_P, \Sigma, \to^\sharp_P\rangle$ defined as follows:*

- *$\mathcal{C}^\sharp_P = \{(q,p) \mid q \in Q$ and $p \in P(q)\}$ is the set of abstract configurations,*
- *$\Sigma = \{1,\ldots,n\} \times \{!,?\} \times M$ is the set of actions,*
- *the abstract transition relation $\to^\sharp_P \subseteq \mathcal{C}^\sharp_P \times \Sigma \times \mathcal{C}^\sharp_P$ is the set of triples $((q,p), l, (q',p'))$ such that $(q, \boldsymbol{w}) \xrightarrow{l} (q', \boldsymbol{w'})$ for some $\boldsymbol{w} \in p$ and $\boldsymbol{w'} \in p'$.*

To relate concrete and abstract configurations, we define the *abstraction function* $\alpha_P : \mathcal{C} \to \mathcal{C}^\sharp_P$, and its extension to $\wp(\mathcal{C}) \to \wp(\mathcal{C}^\sharp_P)$, as well as the *concretization function* $\gamma_P : \mathcal{C}^\sharp_P \to \mathcal{C}$, extended to $\wp(\mathcal{C}^\sharp_P) \to \wp(\mathcal{C})$, as expected:

$$\begin{array}{rclcrcl} \alpha_P((q,\boldsymbol{w})) & = & (q, [\,\boldsymbol{w}\,]_P) & \qquad & \alpha_P(C) & = & \{\alpha(c) \mid c \in C\} \\ \gamma_P((q,p)) & = & \{q\} \times p & & \gamma_P(C^\sharp) & = & \bigcup \{\gamma(c^\sharp) \mid c^\sharp \in C^\sharp\} \end{array}$$

To simplify notations, we shall drop the P subscript when the partition map can easily be derived from the context. Intuitively, an abstract configuration (q,p) of $[\![\mathcal{A}]\!]^\sharp$ represents the set $\{q\} \times p$ of (concrete) configurations of $[\![\mathcal{A}]\!]$. The abstract transition relation $\to^\sharp$ is the existential lift of the concrete transition relation $\to$ to abstract configurations.

The following forward and backward language transformers will be used to capture the effect of fifo actions. The functions $post : \Sigma \times \wp((M^*)^n) \to \wp((M^*)^n)$ and $pre : \Sigma \times \wp((M^*)^n) \to \wp((M^*)^n)$ are defined by:

$$\begin{array}{rcl} post(i!m, L) & = & \{\boldsymbol{w}[i \leftarrow u] \mid \boldsymbol{w} \in L, u \in M^* \text{ and } \boldsymbol{w}(i) \cdot m = u\} \\ post(i?m, L) & = & \{\boldsymbol{w}[i \leftarrow u] \mid \boldsymbol{w} \in L, u \in M^* \text{ and } \boldsymbol{w}(i) = m \cdot u\} \\ pre(i!m, L) & = & \{\boldsymbol{w}[i \leftarrow u] \mid \boldsymbol{w} \in L, u \in M^* \text{ and } \boldsymbol{w}(i) = u \cdot m\} \\ pre(i?m, L) & = & \{\boldsymbol{w}[i \leftarrow u] \mid \boldsymbol{w} \in L, u \in M^* \text{ and } m \cdot \boldsymbol{w}(i) = u\} \end{array}$$

Obviously, $post(l, L)$ and $pre(l, L)$ are effectively recognizable subsets of $(M^*)^n$ for any $l \in \Sigma$ and for any recognizable subset $L \subseteq (M^*)^n$. Moreover, we may use *post* and *pre* to characterize the abstract transition relation of a partition abstraction $[\![\mathcal{A}]\!]^\sharp_P$, by: for any rule $(q, l, q') \in \Delta$ and for any pair $p \in P(q), p' \in P(q')$, we have $(q,p) \xrightarrow{l}{}^\sharp (q',p')$ iff $post(l,p) \cap p' \neq \emptyset$ iff $p \cap pre(l,p') \neq \emptyset$.

Lemma 3.2. *For any fifo system $\mathcal{A}$ and partition map $P : Q \to \mathbb{P}((M^*)^n)$, $[\![\mathcal{A}]\!]^\sharp$ is effectively computable. For any recognizable subset $C \subseteq \mathcal{C}$, $\alpha(C)$ is effectively computable.*

We extend α to paths in the obvious way: $\alpha(c_0 \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} c_h) = \alpha(c_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp \alpha(c_h)$. Observe that $\alpha(\pi)$ is an abstract path in $[\![\mathcal{A}]\!]^\sharp$ for any concrete path π in $[\![\mathcal{A}]\!]$. We therefore obtain the following safety preservation property.

Proposition 3.3. *Consider a fifo system $\mathcal{A}$ and a safety condition $(Init, Bad)$ for $[\![\mathcal{A}]\!]$. For any partition abstraction $[\![\mathcal{A}]\!]^\sharp$ of $[\![\mathcal{A}]\!]$, if $[\![\mathcal{A}]\!]^\sharp$ is $(\alpha(Init), \alpha(Bad))$-safe then $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe.*

The converse to this proposition does not hold in general. An abstract counterexample $\pi^\sharp$ is called *feasible* if there exists a concrete counterexample π such that $\pi^\sharp = \alpha(\pi)$, and $\pi^\sharp$ is called *spurious* otherwise.

Lemma 3.4. *For any fifo system $\mathcal{A}$, partition map $P : Q \to \mathbb{P}((M^*)^n)$, and any safety condition $(Init, Bad)$ for $[\![\mathcal{A}]\!]$, feasibility of abstract counterexamples is effectively decidable.*

Example 3.5. Continuing the discussion of the c/d protocol, we consider the abstraction induced by the following partition map:

$q \in Q$	00	10	01	11
$P(q)$	$(\varepsilon \times M^*), (M^+ \times M^*)$	$(\mathsf{o}^* \times M^*), (\overline{\mathsf{o}^*} \times M^*)$	$M^* \times M^*$	$M^* \times M^*$

The set of initial abstract configurations is $\alpha(Init) = \{(\mathtt{00}, \varepsilon \times M^*)\}$, and the set of bad abstract configurations is $\alpha(Bad) = \{(\mathtt{00}, M^+ \times M^*), (\mathtt{10}, \overline{\mathsf{o}^*} \times M^*)\}$. A simple graph search reveals several abstract counterexamples, for instance $(\mathtt{00}, \varepsilon \times M^*) \xrightarrow{1!\mathsf{o}}{}^\sharp (\mathtt{10}, \mathsf{o}^* \times M^*) \xrightarrow{1!\mathsf{c}}{}^\sharp (\mathtt{00}, M^+ \times M^*)$. This counterexample is spurious since the only concrete path that corresponds to this abstract counterexample is $(\mathtt{00}, \varepsilon, \varepsilon) \xrightarrow{1!\mathsf{o}} (\mathtt{10}, \mathsf{o}, \varepsilon) \xrightarrow{1!\mathsf{c}} (\mathtt{00}, \mathsf{oc}, \varepsilon) \notin Bad$. □

4 Counterexample-based Partition Refinement

The abstraction-based verification of safety properties relies on refinement techniques to gradually increase the precision of abstractions in order to rule out spurious abstract counterexamples. Refinement for partition abstractions simply consists in splitting some classes into a sub-partition.

Given two partitions P and $\widetilde{P}$ of a set S, we say that $\widetilde{P}$ *refines* P when each class $\widetilde{p} \in \widetilde{P}$ is contained in some class $p \in P$. Moreover we then write $[\widetilde{p}]_P$ for the class $p \in P$ containing $\widetilde{p}$.

Let us fix, for the remainder of this section, a fifo system $\mathcal{A} = \langle Q, M, n, \Delta \rangle$ and a safety condition $(Init, Bad)$ for $[\![\mathcal{A}]\!]$. Given two partition maps $P, \widetilde{P} : Q \to \mathbb{P}((M^*)^n)$, we say that $\widetilde{P}$ *refines* P if $\widetilde{P}(q)$ refines $P(q)$ for every control state $q \in Q$. If $\widetilde{P}$ refines P, then for any abstract path $(q_0, \widetilde{p}_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, \widetilde{p}_h)$

in $[\![\mathcal{A}]\!]^\sharp_{\widetilde{P}}$, it holds that $(q_0, [\,\widetilde{p}_0\,]_{P(q_0)}) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, [\,\widetilde{p}_h\,]_{P(q_h)})$ is an abstract path in $[\![\mathcal{A}]\!]^\sharp_P$. This fact shows that, informally, refining a partition abstraction does not introduce any new spurious counterexample.

When a spurious counterexample is found in the abstraction, the partition map must be refined so as to rule out this counterexample. We formalize this for an abstract path $\pi^\sharp_P = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$ in $[\![\mathcal{A}]\!]^\sharp_P$ from $\alpha_P(Init)$ to $\alpha_P(Bad)$ as follows: a refinement $\widetilde{P}$ of P is said to *rule out* the abstract counterexample $\pi^\sharp_P$ if there exists no path $\pi^\sharp_P = (q_0, \widetilde{p}_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, \widetilde{p}_h)$ from $\alpha_{\widetilde{P}}(Init)$ to $\alpha_{\widetilde{P}}(Bad)$ in $[\![\mathcal{A}]\!]^\sharp_{\widetilde{P}}$ satisfying $\widetilde{p}_i \subseteq p_i$ for all $0 \leq i \leq h$. Note that if $\pi^\sharp_P$ is a feasible counterexample, then no refinement of P can rule it out. Conversely, if $\widetilde{P}$ is a refinement of P that rules out $\pi^\sharp_P$ then any refinement of $\widetilde{P}$ also rules out $\pi^\sharp_P$. The main challenge in CEGAR is the discovery of suitable refinements which are computationally "simple" but "precise enough". In this work, we focus on counterexample-guided refinements based on path invariants.

Definition 4.1. *Consider a partition map P and a spurious counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$ in $[\![\mathcal{A}]\!]^\sharp_P$. A* path invariant *for $\pi^\sharp$ is any sequence $L_0, \ldots, L_h$ of recognizable subsets of $(M^*)^n$ such that:*

(i) we have $(\{q_0\} \times p_0) \cap Init \;\subseteq\; \{q_0\} \times L_0$, and
(ii) we have $post(l_i, L_i \cap p_i) \;\subseteq\; L_{i+1}$ for every $0 \leq i \leq h-1$, and
(iii) we have $(\{q_h\} \times L_h) \cap Bad \;=\; \emptyset$

Observe that condition *(ii)* is more general than $post(l_i, L_i) \subseteq L_{i+1}$ which is classically required for inductive invariants. With this relaxed condition, path invariants are tailored to the given spurious counterexample, and therefore can be simpler (e.g., be coarser or have more empty L_i).

Proposition 4.2. *Consider a partition map P and a simple spurious counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$. Given a path invariant $L_0, \ldots, L_h$ for $\pi^\sharp$, the partition map $\widetilde{P}$ defined below is a refinement of P that rules out $\pi^\sharp$:*

$$\widetilde{P}(q) \;=\; (P(q) \setminus \{p_i \mid i \in I(q)\}) \;\cup\; \textstyle\bigcup_{i \in I(q)} \{p_i \cap L_i, p_i \cap \overline{L_i}\} \setminus \{\emptyset\}$$

where $I(q) = \{i \mid 0 \leq i \leq h, q_i = q\}$ for each control state $q \in Q$.

We propose a generic approach to obtain path invariants by utilizing a parametrized approximation operator for queue contents. The parameter (the k in the definition below) is used to adjust the precision of the approximation.

Definition 4.3. *A* (parametrized) extrapolation *is any function ∇ from $\mathbb{N}$ to $\mathcal{R}ec((M^*)^n) \to \mathcal{R}ec((M^*)^n)$ that satisfies, for any $L \in \mathcal{R}ec((M^*)^n)$, the two following conditions (with $\nabla(k)$ written as ∇_k):*

(i) we have $L \subseteq \nabla_k(L)$ for every $k \in \mathbb{N}$,
(ii) there exists $k_L \in \mathbb{N}$ such that $L = \nabla_k(L)$ for every $k \geq k_L$.

Our definition of extrapolation is quite general, in particular, it does not require monotonicity in k or in L, but it is sufficient for the design of path invariant generation procedures. The most simple extrapolation is the naïve extrapolation that maps each $k \in \mathbb{N}$ to the identity on $\mathcal{R}ec((M^*)^n)$. The parametrized automata approximations of [BHV04] and [LGJJ06] also satisfy the requirements of Definition 4.3. Notice that extrapolations are closed under functional union, intersection and composition. The choice of an appropriate extrapolation with respect to the underlying domain of fifo systems is crucial for the implementation of CEGAR's refinement step and will be mentioned in Section 6.

We now present two extrapolation-based path invariant generation procedures. Recall that the parameter k of an extrapolation intuitively indicates the desired precision of the approximation. The first algorithm, UPInv, performs an approximated *post* computation along the spurious counterexample, and iteratively increases the precision k of the approximation until a path invariant is obtained. The applied precision in UPInv is uniform along the counterexample. Due to its simplicity, the termination analysis of CEGAR in the following section will refer to UPInv. The second algorithm, APInv, first performs an exact *pre* computation along the spurious counterexample to identify the "bad" coreachable subsets B_i. The path invariant is then computed with forward traversal that uses the Split subroutine to simplify each *post* image while remaining disjoint from the B_i. The precision used in Split is therefore tailored to each *post* image, which may lead to simpler path invariants. Naturally, both algorithms may be "reversed" to generate path invariants backwards.

Observe that if the extrapolation ∇ is effectively computable, then all steps in the algorithms UPInv, Split and APInv are effectively computable. We now prove correctness and termination of these algorithms. Let us fix, for the remainder of this section, an extrapolation ∇ and a partition map $P : Q \to \mathbb{P}((M^*)^n)$, and assume that $Init$ and Bad are recognizable.

Proposition 4.4. *For any spurious abstract counterexample $\pi_P^\sharp$, the execution of $\mathsf{UPInv}(\nabla, Init, Bad, \pi_P^\sharp)$ terminates and returns a path invariant for $\pi_P^\sharp$.*

Lemma 4.5. *For any two recognizable subsets L_0, L_1 of $(M^*)^n$, if $L_0 \cap L_1 = \emptyset$ then $\mathsf{Split}(\nabla, L_0, L_1)$ terminates and returns a recognizable subset L of $(M^*)^n$ that satisfies $L_0 \subseteq L \subseteq \overline{L_1}$.*

Proposition 4.6. *For any spurious abstract counterexample $\pi_P^\sharp$, the execution of $\mathsf{APInv}(\nabla, Init, Bad, \pi_P^\sharp)$ terminates and returns a path invariant for $\pi_P^\sharp$.*

Example 4.7. Assume an extrapolation ∇ satisfying $\nabla_0(\varepsilon \times M^*) = \varepsilon \times M^*$, $\nabla_0(\mathtt{o} \times M^*) = \mathtt{o} \times M^*$ and $\nabla_0(\mathtt{oc} \times M^*) = \mathtt{o}^+\mathtt{c} \times M^*$. The UPInv algorithm, applied to the spurious counterexample $(\mathtt{00}, \varepsilon \times M^*) \xrightarrow{1!\mathtt{o}}{}^\sharp (\mathtt{10}, \mathtt{o}^* \times M^*) \xrightarrow{1!\mathtt{c}}{}^\sharp (\mathtt{00}, M^+ \times M^*)$ of Example 3.5, would produce the path invariant $(\varepsilon, \mathtt{o}, \mathtt{o}^+\mathtt{c})$. According to Proposition 4.2, the partition map would be refined to :

$q \in Q$	00	10	01	11
$P(q)$	$\{\varepsilon, \mathtt{o}^+\mathtt{c}, M^+ \setminus \mathtt{o}^+\mathtt{c}\} \times M^*$	$\{\mathtt{o}, \mathtt{o}^* \setminus \mathtt{o}, \overline{\mathtt{o}^*}\} \times M^*$	$M^* \times M^*$	$M^* \times M^*$

The refined partition clearly rules out the spurious counterexample. □

$\mathsf{UPInv}\,(\nabla, Init, Bad, \pi_P^\sharp)$

Input: extrapolation ∇, recognizable subsets $Init, Bad$ of $Q\times(M^*)^n$, spurious counterexample $\pi_P^\sharp = (q_0,p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h,p_h)$

```
1   k ← 0
2   do
3      L_0 ← ∇_k (p_0 ∩ {w | (q_0, w) ∈ Init})
4      for i from 1 upto h
5         F_i ← post(l_{i-1}, L_{i-1} ∩ p_{i-1})
6         if F_i ∩ p_i = ∅
7            L_i ← ∅
8         else
9            L_i ← ∇_k(F_i)
10     k ← k + 1
11  while ({q_h} × L_h) ∩ Bad ≠ ∅
12  return (L_0, ..., L_h)
```

$\mathsf{APInv}\,(\nabla, Init, Bad, \pi_P^\sharp)$

Input: (viz. UPInv above)

```
1   B_h ← p_h ∩ Bad
2   i ← h
3   while B_i ≠ ∅ and i > 0
4      i ← i − 1
5      B_i ← p_i ∩ pre(l_{i+1}, B_{i+1})
6   if i = 0
7      I ← p_0 ∩ {w | (q_0, w) ∈ Init}
8      L_0 ← Split (∇, I, B_0)
9   else
10     (L_0, ..., L_i) ← ((M*)^n, ..., (M*)^n)
11  for j from i upto h − 1
12     L_{j+1} ← Split (∇, post(l_j, L_j ∩ p_j), B_{j+1})
13  return (L_0, ..., L_h)
```

$\mathsf{Split}\,(\nabla, L_0, L_1)$

Input: extrapolation ∇, $L_0,\ L_1 \subseteq \mathcal{R}ec\,((M^*)^n)$ disjoint

```
1   k ← 0
2   while ∇_k(L_0) ∩ L_1 ≠ ∅
3      k ← k + 1
4   return ∇_k(L_0)
```

5 Safety Cegar Semi-Algorithm for Fifo Systems

We are now equipped with the key ingredients to present our Cegar semi-algorithm for fifo systems. The semi-algorithm takes as input a fifo system $\mathcal{A}$, a recognizable safety condition $(Init, Bad)$, an initial partition map P_0, and a path invariant generation procedure PathInv. The initial partition map may be the trivial one, mapping each control state to $(M^*)^n$. We may use any path invariant generation procedure, such as the ones presented in the previous section. The semi-algorithm iteratively refines the partition abstraction until either the abstraction is precise enough to prove that $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe (line 10), or a feasible counterexample is found (line 4). If the abstract counterexample picked at line 2 is spurious, a path invariant is generated and is used to refine the partition. The new partition map obtained after the **foreach** loop (lines 8–9) is precisely the partition map $\widetilde{P}$ from Proposition 4.2, and hence it rules out this

abstract counterexample. Recall that Lemmata 3.2 and 3.4 ensure that the steps at lines 1 and 3 are effectively computable. The correctness of the CEGAR semi-

CEGAR $(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$

Input: fifo system $\mathcal{A} = \langle Q, M, n, \Delta \rangle$, recognizable subsets $Init, Bad$ of $Q \times (M^*)^n$, partition map $P_0 : Q \to \mathbb{P}((M^*)^n)$, procedure PathInv

1 **while** $[\![\mathcal{A}]\!]_P^\sharp$ is $(\alpha_P(Init), \alpha_P(Bad))$-unsafe
2 pick a simple abstract counterexample $\pi^\sharp$ in $[\![\mathcal{A}]\!]_P^\sharp$
3 **if** $\pi^\sharp$ is a feasible abstract counterexample
4 **return** $\lightning$
5 **else**
6 write $\pi^\sharp$ as the abstract path $(q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$
7 $(L_0, \ldots, L_h) \leftarrow \mathsf{PathInv}\left(Init, Bad, \pi^\sharp\right)$
8 **foreach** $i \in \{0, \ldots, h\}$
9 $P(q_i) \leftarrow (P(q_i) \setminus \{p_i\}) \cup \left(\left\{p_i \cap L_i, p_i \cap \overline{L_i}\right\} \setminus \{\emptyset\}\right)$
10 **return** ✓

algorithm is expressed by the following proposition, which directly follows from Proposition 3.3 and from the definition of feasible abstract counterexamples.

Proposition 5.1. *For any terminating execution of* CEGAR$(\mathcal{A}, Init, Bad, P_0,$ PathInv*), if the execution returns* ✓ *(resp.* $\lightning$*) then* $[\![\mathcal{A}]\!]$ *is* $(Init, Bad)$*-safe (resp.* $(Init, Bad)$*-unsafe).*

Termination of the CEGAR semi-algorithm cannot be assured as otherwise it would solve the reachability problem known to be undecidable for fifo systems [BZ83]. However, $(Init, Bad)$-unsafety is semi-decidable for fifo systems by forward or backward symbolic exploration when $Init$ and Bad are recognizable [BG99]. Moreover, this problem obviously becomes decidable for fifo systems having a finite reachability set from $Init$.

We investigate in this section the termination of the CEGAR semi-algorithm when $\mathcal{A}$ is $(Init, Bad)$-unsafe or has a finite reachability set from $Init$. On the contrary to other approaches where abstractions are refined globally (e.g., predicate abstraction [GS97]), partition abstractions [CGJ+03] are refined locally by splitting abstract configurations along the abstract counterexample (viz. lines 8 – 9 of the CEGAR semi-algorithm). The abstract transition relation only needs to be refined around the abstract configurations that have been split, and hence its refinement can be computed efficiently. However, this local nature of refinement complicates the analysis of the algorithm. We fix an extrapolation ∇ and we focus on the path invariant generation procedure UPInv presented in Section 4.

Proposition 5.2. *For any breadth-first execution of* CEGAR$(\mathcal{A}, Init, Bad, P_0,$ UPInv $(\nabla))$*, if the execution does not terminate then the sequence* $(h_\theta)_{\theta \in \mathbb{N}}$ *of lengths of counterexamples picked at line 2 is nondecreasing and diverges.*

Corollary 5.3. *If* $[\![\mathcal{A}]\!]$ *is* $(Init, Bad)$*-unsafe then any breadth-first execution of* CEGAR$(\mathcal{A}, Init, Bad, P_0,$ UPInv $(\nabla))$ *terminates.*

It would also be desirable to obtain termination of the CEGAR semi-algorithm when $\mathcal{A}$ has a finite reachability set from $Init$. However, as demonstrated by the example in Appendix D, this condition is not sufficient to guarantee that $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$ has a terminating execution. It turns out that termination can be guaranteed for fifo systems with a finite reachability set when ∇_k has a finite image for every $k \in \mathbb{N}$. This apparently strong requirement, formally specified in Definition 5.4, is satisfied by the extrapolations of [BHV04] and [LGJJ06], which are based on state equivalences up to a certain depth.

Definition 5.4. *An* extrapolation ∇ *is* restricted *if for every* $k \in \mathbb{N}$*, the set* $\{\nabla_k(L) \mid L \in \mathcal{R}ec((M^*)^n)\}$ *is finite.*

Remark that if ∇ is restricted then for any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$, the execution terminates if and only if the number of iterations of the **while**-loop of the algorithm UPInv is bounded. As shown by the following proposition, if moreover $[\![\mathcal{A}]\!]$ has a finite reachability set from $Init$ then the execution necessarily terminates.

Proposition 5.5. *Assume that* ∇ *is restricted. If* $[\![\mathcal{A}]\!]$ *has a finite reachability set from* $Init$*, then any execution of* $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$ *terminates.*

6 Experimentation and Perspectives

Our prototypical tool MCSCM that implements the previous algorithms is written in OCAML and relies on a library by Le Gall and Jeannet [Scm] for the classical regular language operations, the fifo operations, and the colored bisimulation-based extrapolation.

Colored Bisimulation-based Extrapolation. Our underlying extrapolation is the bisimulation-based construction of [LGJJ06]. In a nutshell, abstractions of finite automata are easily given by a congruence on the automata's states. The authors of [LGJJ06] present a congruence relation based on a colored bisimulation equivalence, i.e., two states are equal if they have the same color (regarding an a priori given partition of the state space) and are bisimulation equivalent, ergo, behave "equally". A discussion that favors this extrapolation for the verification of fifo systems is presented in [LGJJ06] whereas their focus remains on applying this extrapolation as widening in an abstract interpretation based approach.

Benchmarks. Our implementation includes the two path invariant generation algorithms UPInv and APInv of Section 4. We actually implemented a "single split" backward variant of APInv, reminiscent of the classical CEGAR implementation [CGJ+03] (analogous to APInv but applying the split solely to the "failure" abstract configuration). Therefore our implemented variant APInv' leads to more CEGAR loops than would be obtained with APInv, and this explains in part why UPInv globally outperforms APInv'. Several pluggable subroutines can be used to search for counterexamples (depth-first or breadth-first exploration).

We tested the prototype on a suite of protocols that includes the classical alternating bit protocol ABP [AJ96], a simplified version of TCP – also in the setting of one server with two clients that share their channels, as well as protocols for leader election due to Peterson and token passing in a ring topology. Further, we provide certain touchstones for our approach, for example, an enhancement of the c/d protocol with nested loops and a protocol with strictly non-regular configurations. A detailed presentation of the protocols is provided in Appendix A. Except for the c/d protocol, which is unsafe, all other examples are safe.

protocol	states/trans.	refmnt.	time [s]	mem [MiB]	loops	states$^\sharp$/trans$^\sharp$	expl.
ABP	16/64	APInv'	0.47	1.3	84	99/588	bfs
		UPInv	3.48	2.23	167	326/1658	dfs
c/d protocol	4/12	APInv'	0.01	0.36	6	10/39	bfs
		UPInv	0.01	0.36	5	11/34	bfs
nested c/d protocol	6/17	APInv'	0.01	0.36	8	13/25	bfs
		UPInv	0.01	0.36	14	39/93	dfs
non-regular protocol	9/18	APInv'	0.02	0.59	13	21/47	dfs
		UPInv	0.05	0.59	8	25/39	dfs
(simplified) TCP	196/588	APInv'	2.63	2.7	147	342/1230	dfs
		UPInv	1.16	1.53	120	386/1240	bfs
server with 2 clients	255/2160	APInv'	2.62	2.7	291	662/2230	bfs
		UPInv	2.61	2.7	291	662/2230	bfs
token ring	625/4500	APInv'	(>1h)	—	—	—	—
		UPInv	2.43	4.81	194	857/5807	bfs
Peterson	10648/56628	APInv'	12.34	29.66	296	10943/58634	dfs
		UPInv	2.97	28.95	53	10709/56987	dfs

The above table gives a summary of the results, obtained by MCSCM on an off-the-shelf computer (2.4Ghz Intel Core 2 Duo). The last column indicates the graph search used to find abstract counterexamples. All examples are analyzed with UPInv in a few seconds, and memory is not a limitation.

We compared MCSCM with TREX [TRe], the (publicly available) tool closest to ours regarding the verification of *unbounded* channel systems but which focuses *lossy* channel semantics. As TREX has an efficient implementation based on *simple regular expressions* (and not general QDDs as we do), it needs in most cases less than 1 second to build the reachability set. However, TREX assumes a *lossy* fifo semantics, and therefore is not able to verify all *reliable* fifo protocols correctly (e.g., when omitting the `disconnect` messages in the c/d protocol, TREX is still able to reach *Bad* due to the possible loss of messages, albeit the protocol is safe). Furthermore, TREX suffers (as would also LASH [Las]) from the main drawback of acceleration techniques, which in general cannot cope with nested loops, whereas this is no drawback for our tool (viz. nested loop protocol on which TREX did not finish after 1 hour). MCSCM can also handle a simple non-regular protocol (with a counting loop) that is beyond the QDD-based approaches [BG99], as the representation of the reachability set would require recognizable languages equipped with Presburger formulas (CQDDs [BH99]).

Conclusion and Perspectives. Our prototypical implementation confirms our expectations that the proposed CEGAR framework with extrapolation-based path

invariants is a promising alternative approach to the automatic verification of fifo systems. The framework developped in this paper is actually not specific to fifo systems, and we intend to investigate its practical relevance to other infinite-state models. Future work also includes the safety verification of more complex fifo systems that would allow the exchange of numerical data over the queues (e.g., the sliding window protocols). Several decidable classes of fifo systems have emerged in the literature (in particular lossy fifo systems) and we intend to investigate termination of our CEGAR semi-algorithm (equipped with the path invariant generation procedure $\mathsf{UPInv}\,(\nabla)$) for these classes.

References

[AJ96] P. A. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. *Information and Computation*, 127(2):91–101, 1996.

[Ber79] J. Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.

[BG99] B. Boigelot and P. Godefroid. Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs. *Formal Methods in System Design*, 14(3):237–255, 1999.

[BH99] A. Bouajjani and P. Habermehl. Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations. *Theoretical Computer Science*, 221(1-2):211–250, 1999.

[BHV04] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract Regular Model Checking. In *Proc. Computer Aided Verification 2004*, volume 3114 of *LNCS*, pages 372–386. Springer, 2004.

[BR01] T. Ball and S. K. Rajamani. Automatically Validating Temporal Safety Properties of Interfaces. In *Proc. Model Checking Software, SPIN Workshop 2001*, volume 2057 of *LNCS*, pages 103–122. Springer, 2001.

[BZ83] D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CGJ+03] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM*, 50(5):752–794, 2003.

[FIS03] A. Finkel, S. P. Iyer, and G. Sutre. Well-Abstracted Transition Systems: Application to FIFO Automata. *Information and Computation*, 181(1):1–31, 2003.

[FKK+07] P. Flocchini, E. Kranakis, D. Krizanc, F. L. Luccio, and N. Santoro. Leader Election and Sorting in Anonymous Asynchronous Rings 1, Nov. 2007. (unpublished preprint).

[GS97] S. Graf and H. Saïdi. Construction of Abstract State Graphs with PVS. In *Proc. Computer Aided Verification 1997*, volume 1245 of *LNCS*, pages 72–83, 1997.

[HJMS02] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In *Proc. Symposium on Principles of Programming Languages 2002*, pages 58–70. ACM, 2002.

[JR86] C. Jard and M. Raynal. De la nécessité de spécifier des propriétés pour la verification des algorithmes distribués. Rapports de Recherche 590, IRISA Rennes, December 1986.

[Las] Lash. Tool homepage. `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[LGJJ06] T. Le Gall, B. Jeannet, and T. Jéron. Verification of Communication Protocols using Abstract Interpretation of FIFO queues. In *Proc. Conference on Algebraic Methodology and Software Technology 2006*, volume 4019 of *LNCS*, pages 204–219. Springer, July 2006.

[McS] Tool homepage. McScM – Model Checker for Systems of Communicating Fifo Machines `http://www.labri.fr/~heussner/mcscm/`.

[Pet82] G. L. Peterson. An O(n log n) Unidirectional Algorithm for the Circular Extrema Problem. *ACM Transactions on Programming Languages and Systems*, 4(4):758–762, 1982.

[Scm] Tools and libraries for static analysis and verification. `http://gforge.inria.fr/projects/bjeannet/`.

[TRe] TReX. Tool homepage. `http://www.liafa.jussieu.fr/~sighirea/trex`.

[YBCI08] F. Yu, T. Bultan, M. Cova, and O. H. Ibarra. Symbolic String Verification: An Automata-Based Approach. In *Proc. Model Checking Software, SPIN Workshop 2008*, volume 5156 of *LNCS*, pages 306–324, 2008.

A Protocols of the Experimental Evaluation (Section 6)

We present in this section the suite of protocols (except for the c/d protocol which was already introduced in Section 2) on which we tested our prototype McScM. Each protocol is specified as a system of communicating processes. In each case, the resulting fifo system is the asynchronous product of the processes. The queues are initially empty, and each process has a single initial state that is graphically indicated by an arrow with no source state. We provide with each protocol the set of bad configurations used in our experimental evaluation.

A.1 Alternating Bit Protocol

This is the classical example protocol for automatic verification for communicating fifo systems in the formalization of Le Gall et al. [LGJJ06]. The two participating peers exchange control data over the channels 0 and 1 as well as data over channel 2.

Starting from the initial states with empty queues, we check whether a configuration is reachable which has more than one message on a queue (i.e., in which the control data is ignored when sending data). This can be encoded purely in control states due to the protocols acknowledgement mechanisms.

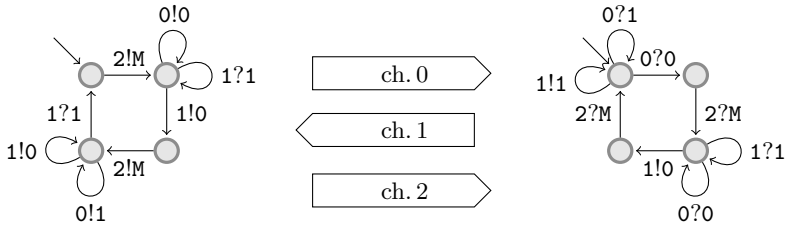

Fig. 3: Alternating Bit Protocol

Remark A.1. Regarding our prototype McScM, single-split refinement methods are not able to terminate in reasonable time compared in contrast to path invariant (UPInv) based refinement.

A.2 Nested Loop Protocol

Systems with nested loops overburden standard acceleration techniques, which rely on the analysis of simple loops and cannot accelerate nested loops. We extended the connection-disconnection protocol with a simple loop for sending a

data token m over the channel. Our CEGAR method was able to prove (at least) the following safety property: "a message m cannot be received when the session is closed on the server-side" (state 0).

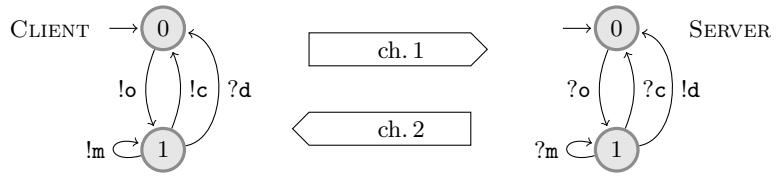

Fig. 4: Extending the Connection/Disconnection Protocol with Nested Loops

A.3 Non-Regular Protocol

This is a simple example where the reachability set is non-recognizable. Indeed, the set of reachable queue contents in control state 00 is $\{(\varepsilon, a^m, \varepsilon, b^m) \mid m \in \mathbb{N}\}$ which is not recognizable. The safety property is given explicitly by the control state 02, which should not be reachable.

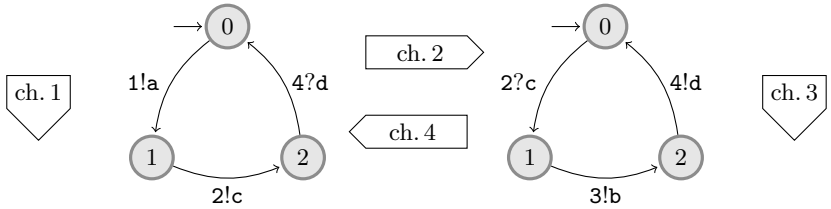

Fig. 5: Non-Regular Protocol using Channels 1,3 like Stacks

Remark A.2. Even though we only utilize recognizable subsets to compute invariants, our approach is able to verify the safety property. Other techniques that are based on recognizable subsets, but that rely on an exhaustive exploration of the state space, like the acceleration with QDDs [BG99], are not able to handle non-recognizable systems at all. One may argue that our technique is limited to safety properties that can be proved with recognizable invariants.

A.4 Simplified TCP

Based on the underlying state transition of the TCP protocol and by ignoring all the additional timing constraints as well as the sophisticated data transport

(sliding windows etc.), we modeled the three-way handshake of TCP as well as the passive/active close with respect to a simple client-server-setting with one bidirectional channel.

The diagram in Figure 6 depicts the client which can open a connection to a server that is waiting for it. We only utilize the messages `s(yn)`, `a(ck)`, `f(in)`, `d(ata)` and depict the client which opens a connection by sending `s` to the server which mirrors the behavior.

We simply verified that the connection establishment and termination work by checking whether one of the peers remains in the *closed* state whereas the other assumes the connection to be *established*.

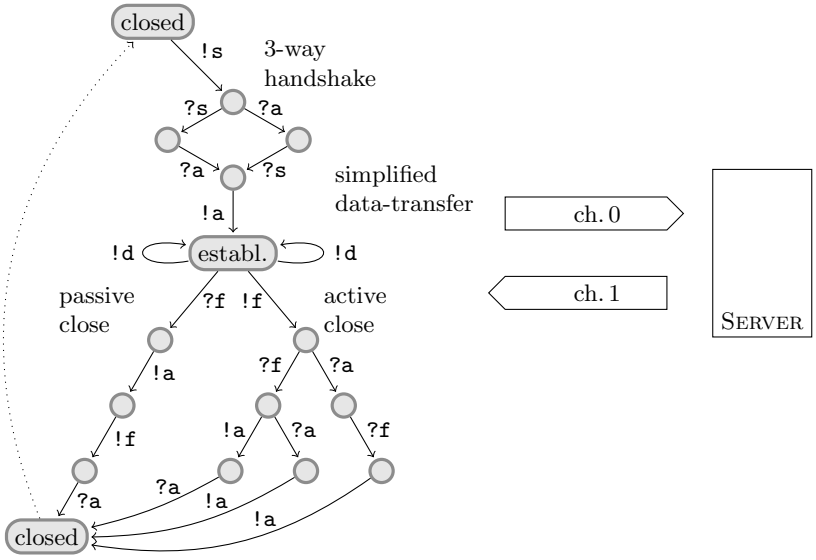

Fig. 6: Simplified Transmission Control Protocol

A.5 Server with Two Clients

This is a simple extension of the (simplified) TCP protocol, where we verify the correctnes of connection establishment and termination in the case of a second client that tries to interfere with the server on the same channels as the original client.

A.6 Ring Protocol

This is an example of a token passing protocol, where n identical processes, set in a ring architecture, can pass some tokens. At the beginning, each process has

0 or 1 token (control states `0` or `1`). A process is in a "bad" configuration `b` when it has two tokens. Therefore, it sends an alert message a before sending a token t. When a process receives an alert message, it ignores it (if it has no token) or sends immediately its token to the following process, without an alert message. This is the reason why the only outgoing transition of control state `3` is to send a token.

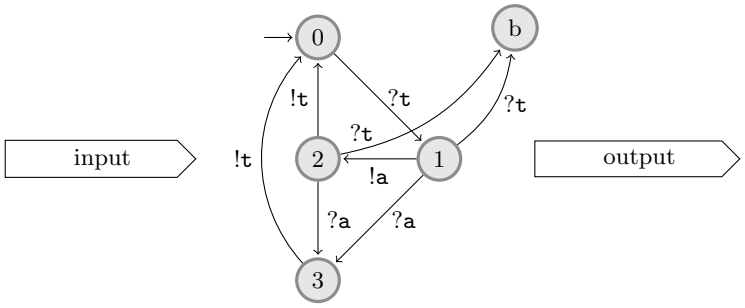

Fig. 7: Single Peer of Token Ring Protocol

A.7 Peterson's Leader Election

This is a translation of Peterson's leader election algorithm [Pet82] (viz. Figure 8 for pseudocode taken from [FKK+07]) into a fifo system. The algorithm is modeled as a set of finite state automata which are executed distributively (and asynchronously) in a ring topology. We check whether more than one process asserts that he is the leader. In our case, the number of peers is fixed to 3 (we do not perform parametrized model checking).

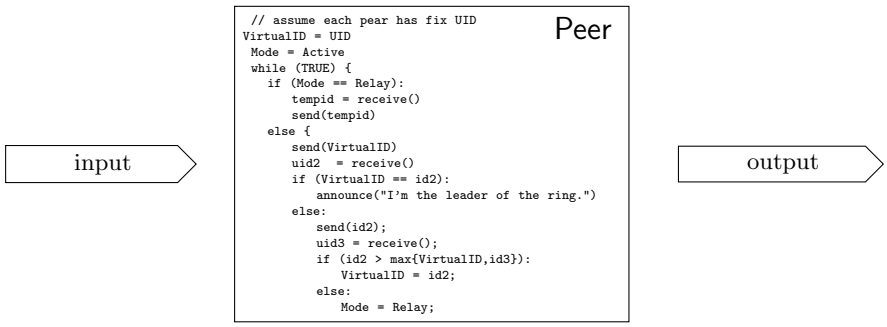

Fig. 8: Leader Election in an Asynchronous Ring following Peterson

B Proofs

C.1 Proofs of Section 3

Lemma 3.2. *For any fifo system $\mathcal{A}$ and partition map $P : Q \to \mathbb{P}((M^*)^n)$, $[\![\mathcal{A}]\!]^\sharp$ is effectively computable. For any recognizable subset $C \subseteq \mathcal{C}$, $\alpha(C)$ is effectively computable.*

Proof. The lemma follows from (1) closure under intersection, complement and *post* (or *pre*) of recognizable subsets of $(M^*)^n$, and (2) decidability of emptiness for recognizable subsets of $(M^*)^n$. □

Proposition 3.3. *Consider a fifo system $\mathcal{A}$ and a safety condition $(Init, Bad)$ for $[\![\mathcal{A}]\!]$. For any partition abstraction $[\![\mathcal{A}]\!]^\sharp$ of $[\![\mathcal{A}]\!]$, if $[\![\mathcal{A}]\!]^\sharp$ is $(\alpha(Init), \alpha(Bad))$-safe then $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe.*

Proof. If $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-unsafe then there is a path π in $[\![\mathcal{A}]\!]$ from $Init$ to Bad, and hence $\alpha(\pi)$ is an abstract path from $\alpha(Init)$ to $\alpha(Bad)$ in $[\![\mathcal{A}]\!]^\sharp$. □

Lemma 3.4. *For any fifo system $\mathcal{A}$, partition map $P : Q \to \mathbb{P}((M^*)^n)$, and any safety condition $(Init, Bad)$ for $[\![\mathcal{A}]\!]$, feasibility of abstract counterexamples is effectively decidable.*

Proof. Given an abstract counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$, we deduce from the definition of feasibility that $\pi^\sharp$ is feasible iff the subset $L \subseteq (M^*)^n$ defined below is non-empty:

$$L \;=\; p_h \cap post(l_{h-1}, (p_{h-1} \cap \cdots \cap post(l_1, p_1 \cap post(l_0, p_0 \cap Init)) \cdots)) \cap Bad$$

Since L is an effectively computable recognizable subset of $(M^*)^n$, we may effectively decide whether L is non-empty, which concludes the proof. □

C.2 Proofs of Section 4

Proposition 4.2. *Consider a partition map P and a simple spurious counterexample $\pi^\sharp = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$. Given a path invariant $L_0, \ldots, L_h$ for $\pi^\sharp$, the partition map $\widetilde{P}$ defined below is a refinement of P that rules out $\pi^\sharp$:*

$$\widetilde{P}(q) \;=\; (P(q) \setminus \{p_i \mid i \in I(q)\}) \;\cup\; \bigcup_{i \in I(q)} \{p_i \cap L_i, p_i \cap \overline{L_i}\} \setminus \{\emptyset\}$$

where $I(q) = \{i \mid 0 \leq i \leq h, q_i = q\}$ for each control state $q \in Q$.

Proof. For any control sate $q \in Q$, since $\pi^\sharp$ is simple, we have $p_i = p_j \Rightarrow i = j$ for every $i, j \in I(q)$. The function $\widetilde{P}$ defined in the proposition is therefore a partition map that refines P by definition. We need to show that $\widetilde{P}$ rules out $\pi^\sharp$.

By contradiction, assume there exists a path $\pi^\sharp_P = (q_0, \widetilde{p}_0) \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} (q_h, \widetilde{p}_h)$ from $\alpha_{\widetilde{P}}(Init)$ to $\alpha_{\widetilde{P}}(Bad)$ in $[\![\mathcal{A}]\!]^\sharp_{\widetilde{P}}$ satisfying $\widetilde{p}_i \subseteq p_i$ for all $0 \leq i \leq h$.

We first show that $\widetilde{p}_i \in \{p_i \cap L_i, p_i \cap \overline{L_i}\}$ for every $0 \leq i \leq h$. Consider any integer i with $0 \leq i \leq h$. Observe that $i \in I(q_i)$. If $\widetilde{p}_i \in P(q_i)$ then $\widetilde{p}_i = p_i$ as $\widetilde{p}_i \subseteq p_i$. Hence, $\widetilde{p}_i \notin (P(q_i) \setminus \{p_j \mid j \in I(q_i)\})$. Since $\widetilde{p}_i \in \widetilde{P}(q_i)$, we obtain that $\widetilde{p}_i \in \{p_j \cap L_j, p_j \cap \overline{L_j}\}$ for some $j \in I(q_i)$. Let us prove that $i = j$. Remark that $q_i = q_j$ as $j \in I(q_i)$. Moreover, we get $\widetilde{p}_i \subseteq p_j$, and hence $\widetilde{p}_i \subseteq p_i \cap p_j$. Therefore $p_i = p_j$ since p_i and p_j are classes of the same partition $P(q_i)$. We come to $(q_i, p_i) = (q_j, p_j)$ which implies that $i = j$ since $\pi^\sharp$ is simple. We have thus shown that $\widetilde{p}_i \in \{p_i \cap L_i, p_i \cap \overline{L_i}\}$ for every $0 \leq i \leq h$.

Recall that $L_0, \ldots, L_h$ is a path invariant for $\pi^\sharp$. We prove by induction on i that $\widetilde{p}_i = p_i \cap L_i$ for every $0 \leq i \leq h$. For the basis, we derive from item (i) of Definition 4.1 that $\{q_0\} \times (p_0 \cap \overline{L_0})$ is disjoint from $Init$. Since $(q_0, \widetilde{p}_0) \in \alpha_{\widetilde{P}}(Init)$, we get that $\{q_0\} \times \widetilde{p}_0$ intersects $Init$. Therefore $\widetilde{p}_0 \neq p_0 \cap \overline{L_0}$, and hence $\widetilde{p}_0 = p_0 \cap L_0$. For the induction step, assume that $\widetilde{p}_i = p_i \cap L_i$ for some $0 \leq i < h$. We have $post(l_i, \widetilde{p}_i) \subseteq L_{i+1}$ according to item (ii) of Definition 4.1. Therefore, we get that $p_{i+1} \cap \overline{L_{i+1}}$ is disjoint from $post(l_i, \widetilde{p}_i)$. Since $(q_i, \widetilde{p}_i) \xrightarrow{l_i}{}^\sharp (q_{i+1}, \widetilde{p}_{i+1})$ is an abstract transition in $[\![\mathcal{A}]\!]^\sharp_{\widetilde{P}}$, we get that $\widetilde{p}_{i+1}$ intersects $post(l_i, \widetilde{p}_i)$. Therefore $\widetilde{p}_{i+1} \neq p_{i+1} \cap \overline{L_{i+1}}$, and hence $\widetilde{p}_{i+1} = p_{i+1} \cap L_{i+1}$.

We thus obtain that $\widetilde{p}_h = p_h \cap L_h$, and we derive from item (iii) of Definition 4.1 that $\{q_h\} \times \widetilde{p}_h$ is disjoint from Bad, which contradicts the assumption that $(q_h, \widetilde{p}_h) \in \alpha_{\widetilde{P}}(Bad)$. □

Proposition 4.4. *For any spurious abstract counterexample $\pi^\sharp_P$, the execution of $\mathsf{UPInv}(\nabla, Init, Bad, \pi^\sharp_P)$ terminates and returns a path invariant for $\pi^\sharp_P$.*

Proof. Consider a spurious counterexample $\pi^\sharp_P = (q_0, p_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, p_h)$. Let us define the sequence $R_0, \ldots, R_h$ of subsets of $(M^*)^n$ by $R_0 = p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in Init\}$ and $R_i = post(l_{i-1}, L_{i-1} \cap p_{i-1})$ for all $1 \leq i \leq h$. Notice that $(\{q_h\} \times R_h) \cap Bad = \emptyset$ since $\pi^\sharp_P$ is spurious. According to Definition 4.3, there exists $k_R \in \mathbb{N}$ such that $\nabla_k(R_i) = R_i$ for every $0 \leq i \leq h$. Consequently, the **while**-loop of algorithm UPInv (lines 2–11) is re-iterated at most k_R times. Indeed, if $k = k_R$ at some iteration of the **while**-loop, then for this iteration we have $L_i = R_i$ for each $0 \leq i \leq h$ and therefore $(\{q_h\} \times L_h) \cap Bad = \emptyset$. We conlude that the execution of $\mathsf{UPInv}(\nabla, Init, Bad, \pi^\sharp_P)$ terminates.

Let $(L_0, \ldots, l_h)$ denote the value returned by $\mathsf{UPInv}(\nabla, Init, Bad, \pi^\sharp_P)$. It obvioulsy holds that $(\{q_h\} \times L_h) \cap Bad = \emptyset$. Recall that according to Definition 4.3, we have $L \subseteq \nabla_k(L)$ for every $L \in \mathcal{R}ec((M^*)^n)$ and $k \in \mathbb{N}$. We deduce from the definition of the **while**-loop (lines 2–11) that $L_0 \supseteq p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in Init\}$ and $L_i \supseteq post(l_{i-1}, L_{i-1} \cap p_{i-1})$ for all $1 \leq i \leq h$. We conlude that $(L_0, \ldots, l_h)$ is a path invariant. □

Lemma 4.5. *For any two recognizable subsets L_0, L_1 of $(M^*)^n$, if $L_0 \cap L_1 = \emptyset$ then $\mathsf{Split}(\nabla, L_0, L_1)$ terminates and returns a recognizable subset L of $(M^*)^n$ that satisfies $L_0 \subseteq L \subseteq \overline{L_1}$.*

Proof. Consider any two disjoint recognizable subsets L_0, L_1 of $(M^*)^n$. According to Definition 4.3, we have $L = \nabla_k(L)$ for some $k_L \in \mathbb{N}$, and therefore $\mathsf{Split}(\nabla, L_0, L_1)$ terminates. There exists $k \in \mathbb{N}$ such that the returned value L satisfies $L = \nabla_k(L_0)$ and $\nabla_k(L_0) \cap L_1 = \emptyset$. Since $L_0 \subseteq \nabla_k(L_0)$ from Definition 4.3, we obtain that $L_0 \subseteq L \subseteq \overline{L_1}$. □

Proposition 4.6. *For any spurious abstract counterexample $\pi_P^\sharp$, the execution of $\mathsf{APInv}(\nabla, Init, Bad, \pi_P^\sharp)$ terminates and returns a path invariant for $\pi_P^\sharp$.*

Proof (Sketch). The algorithm terminates as Split terminates. Further, assume the result is $(L_0, \ldots, L_h)$. We have to consider two cases: (a) that the backward reachability search of lines 1–5 reaches p_0 or (b) that it finds a p_j that is not *pre*-reachable from $p_j \cap B_j$.
Let us consider (a): lines 7 assure requirement (i); further, we deduce from Lemma 4.5 with line 12 that (iii) and (ii) hold. Now, we can show that also for case (b) the result is a path invariant: requirements (iii) and (i) are analogous to case (a), (ii) is guaranteed by $L_0, \ldots, L_i = (M^*)^n$ and for $L_{i+1}, \ldots, L_h$ analogous to case (a) above. □

C.3 Proofs of Section 5

We first introduce some new notations. For any set $\mathcal{L}$ of subsets of $(M^*)^n$, we denote by $\Psi(\mathcal{L})$ the set of equivalence classes of the equivalence relation $\sim_{\mathcal{L}}$ on $(M^*)^n$ defined by: $\boldsymbol{w} \sim_{\mathcal{L}} \boldsymbol{w'}$ if for every $L \in \mathcal{L}$, we have $\boldsymbol{w} \in L$ if and only if $\boldsymbol{w'} \in L$. Intuitively, $\Psi(\mathcal{L})$ is the partition "generated" by $\mathcal{L}$. Recall that if $\mathcal{L}$ is finite then so is $\Psi(\mathcal{L})$.

Given an execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$, and for each iteration $\theta \in \mathbb{N}$ of the **while**-loop, we take a "snaphot" between lines 7 and 8, and remember the current partition map as P_θ, the simple abstract counterexample as $\pi_\theta^\sharp$ and its length as h_θ, and the path invariant as $(L_0^\theta, \ldots, L_{h_\theta}^\theta)$. Moreover we shortly write $[\![\mathcal{A}]\!]_\theta^\sharp$, $\mathcal{C}_\theta^\sharp$ and α_θ instead of $[\![\mathcal{A}]\!]_{P_\theta}^\sharp$, $\mathcal{C}_{P_\theta}^\sharp$ and α_{P_θ}, respectively. We also define $Init_\theta^\sharp = \alpha_\theta(Init)$ and $Bad_\theta^\sharp = \alpha_\theta(Bad)$. For any bound $b \in \mathbb{N}$, we let $Reach_\theta^{\leq b}$ denote the set of abstract configurations $(q, p) \in \mathcal{C}_\theta^\sharp$ such that there exists in $[\![\mathcal{A}]\!]_\theta^\sharp$ a path of length at most b from $Init_\theta^\sharp$ to (q, p).

Lemma C.1. *Consider any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{PathInv})$. For any iteration θ and for any $(q, p) \in \mathcal{C}_\theta^\sharp$, we have $p \in \Psi(\mathcal{L}_\theta(q))$ where:*

$$\mathcal{L}_\theta(q) \;=\; P_0(q) \cup \{L_i^\eta \mid 0 \leq \eta < \theta \text{ and } 0 \leq i \leq h_\eta\}$$

Proof. We prove the lemma by induction on θ. The basis is trivial, since $p \in P_0(q)$ for every $(q, p) \in \mathcal{C}_0^\sharp$. Assume that the lemma holds for the iteration θ and let us

show that the lemma also holds for the iteration $\theta+1$. Let $(q,p) \in \mathcal{C}^\sharp_{\theta+1}$. If $(q,p) \in \mathcal{C}^\sharp_\theta$, then we get that $p \in \Psi(\mathcal{L}_\theta(q))$. Since $\mathcal{L}_\theta(q) \subseteq \mathcal{L}_{\theta+1}(q)$, we obtain that $p \in \Psi(\mathcal{L}_{\theta+1}(q))$. Assume now that $(q,p) \notin \mathcal{C}^\sharp_\theta$. Since $p \in P_{\theta+1}(q) \setminus P_\theta(q)$, we get that p was added to $P(q)$ during the iteration θ at line 9. We deduce from line 9 that $p \in \left\{p' \cap L, p' \cap \overline{L}\right\}$ for some $(q',p') \in \mathcal{C}^\sharp_\theta$ and $L \in \left\{L^\theta_i \;\middle|\; 0 \leq i \leq h_\theta\right\}$. We deduce from the induction hypothesis $p' \in \Psi(\mathcal{L}_\theta(q))$ and therefore $p \in \Psi(\mathcal{L}_{\theta+1}(q))$. □

Proposition C.2. *For any non-terminating execution of* CEGAR$(\mathcal{A}, Init, Bad, P_0,$ PathInv$)$*, the set* $\left\{L^\theta_i \;\middle|\; \theta \in \mathbb{N} \text{ and } 0 \leq i \leq h_\theta\right\}$ *is infinite.*

Proof. Consider a non-terminating execution and let us show that the set $\mathcal{L} = \left\{L^\theta_i \;\middle|\; \theta \in \mathbb{N}, 0 \leq i \leq h_\theta\right\}$ is infinite. We get from Lemma C.1 that for every $q \in Q$ and $\theta \in \mathbb{N}$, we have $P_\theta(q) \subseteq \Psi(P_0(q) \cup \mathcal{L})$. According to line 9 of the CEGAR semi-algorithm, $P_{\theta+1}$ refines P_θ for every $\theta \in \mathbb{N}$, and moreover $P_{\theta+1} \neq P_\theta$ since $P_{\theta+1}$ rules out $\pi^\sharp_\theta$. We deduce that there exists $q \in Q$ such that the nondecreasing sequence $(|P_\theta(q)|)_{\theta \in \mathbb{N}}$ diverges. Since $P_0(q)$ is finite and $P_\theta(q) \subseteq \Psi(P_0(q) \cup \mathcal{L})$, we conclude that $\mathcal{L}$ is infinite. □

Lemma C.3. *Consider any breadth-first execution of* CEGAR$(\mathcal{A}, Init, Bad, P_0,$ PathInv$)$*. For any iteration* $\theta \geq 1$ *and for any* $(q,p) \in Init^\sharp_\theta \setminus Init^\sharp_{\theta-1}$*, there exists* $p_0 \in P_{\theta-1}(q)$ *such that* $p = p_0 \cap L_0^{\theta-1}$ *and* $(\{q\} \times p_0) \cap Init = (\{q\} \times p) \cap Init$*.*

Proof. Consider an iteration $\theta+1$ (with $\theta \in \mathbb{N}$) and let (q,p) be any abstract configuration in $Init^\sharp_{\theta+1} \setminus Init^\sharp_\theta$. Observe that $(\{q\} \times p) \cap Init$ is non-empty, and therefore $(q,p) \notin \mathcal{C}^\sharp_\theta$ since otherwise we would have $(q,p) \in Init^\sharp_\theta$. Since $p \in P_{\theta+1}(q) \setminus P_\theta(q)$, we get that p was added to $P(q)$ during the iteration θ at line 9. Let us write $\pi^\sharp_\theta$ as $\pi^\sharp_\theta = (q_0,p_0) \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} (q_h,p_h)$. We deduce from line 9 that $p \in \left\{p_{i_0} \cap L^\theta_{i_0}, p_{i_0} \cap \overline{L^\theta_{i_0}}\right\}$ for some $0 \leq i_0 \leq h$ such that $q = q_{i_0}$. Observe that $(q_{i_0}, p_{i_0}) \in Init^\sharp_\theta$ since we have $p \subseteq p_{i_0}$ and $(\{q\} \times p) \cap Init \neq \emptyset$. We come to $i_0 = 0$ since the abstract counterexample $\pi^\sharp_\theta$ is among the shortest ones. Hence, we get that $q = q_0$ and $p \in \left\{p_0 \cap L^\theta_0, p_0 \cap \overline{L^\theta_0}\right\}$. Since $(L^\theta_0, \ldots, L^\theta_h)$ is a path invariant for $\pi^\sharp_\theta$, we have $(\{q_0\} \times p_0) \cap Init \subseteq \{q_0\} \times L^\theta_0$ and hence $\{q_0\} \times (p_0 \cap \overline{L^\theta_0})$ is disjoint from $Init$. We deduce that $p = p_0 \cap L^\theta_0$ and moreover we get that $(\{q\} \times p_0) \cap Init = (\{q\} \times p) \cap Init$. □

Lemma C.4. *Consider any breadth-first execution of* CEGAR$(\mathcal{A}, Init, Bad, P_0,$ PathInv$)$*. For any iteration* θ*, for any* $b \in \mathbb{N}$ *and for any* $(q,p) \in Reach^{\leq b}_\theta$*, we have* $p \in \Psi\big(\mathcal{L}^b_\theta(q)\big)$ *where:*

$$\mathcal{L}^b_\theta(q) \;=\; P_0(q) \cup \{L^\eta_i \mid 0 \leq \eta < \theta, i \leq h_\theta \text{ and } i \leq b\}$$

Proof. For any iteration θ and for any $b \in \mathbb{N}$, let use denote by (H^b_θ) the property: for any $(q,p) \in \mathcal{C}^\sharp_\theta$, if there exists in $[\![\mathcal{A}]\!]^\sharp_\theta$ a path of length at most b from $Init^\sharp_\theta$ to (q,p), then $p \in \Psi\big(\mathcal{L}^b_\theta(q)\big)$. We prove by double induction on θ and b that (H^b_θ) holds for any iteration θ and for any $b \in \mathbb{N}$.

Let us prove the basis $\forall b\,(H_0^b)$ of the induction on θ. Observe that $\mathcal{L}_0^b(q) = P_0(q)$ for every $q \in Q$. Therefore $p \in P_0(q) = \Psi(P_0(q))$ for any $(q,p) \in \mathcal{C}_0^\sharp$, and we conclude that the basis obviously holds. We now prove the induction step $\forall\theta\,\big(\forall b\,(H_\theta^b) \implies \forall b\,(H_{\theta+1}^b)\big)$ of the induction on θ. Consider an iteration $\theta+1$ (with $\theta \in \mathbb{N}$) and assume that (H_θ^b) holds for every $b \in \mathbb{N}$. We prove by induction on b that $(H_{\theta+1}^b)$ holds for any $b \in \mathbb{N}$. Observe that the basis $(H_{\theta+1}^0)$ may equivalently be rephrased as: for any $(q,p) \in \mathit{Init}_{\theta+1}^\sharp$, we have $p \in \Psi\big(\mathcal{L}_{\theta+1}^0\big)$. Let $(q,p) \in \mathit{Init}_{\theta+1}^\sharp$. If $(q,p) \in \mathit{Init}_\theta^\sharp$ then we deduce from (H_θ^0) that $p \in \Psi\big(\mathcal{L}_\theta^0\big)$. Since $\mathcal{L}_\theta^0 \subseteq \mathcal{L}_{\theta+1}^0$ we obtain that $p \in \Psi\big(\mathcal{L}_{\theta+1}^0\big)$. Otherwise, we obtain from Lemma C.3 that $p = p_0 \cap L_0^\theta$ for some $p_0 \in P_\theta(q)$. We deduce from (H_θ^0) that $p_0 \in \Psi\big(\mathcal{L}_\theta^0\big)$ and therefore $p \in \Psi\big(\mathcal{L}_{\theta+1}^0\big)$. We therefore have proved that the basis $(H_{\theta+1}^0)$ of the induction on b holds.

Let us now show the induction step $\forall b\,\big((H_{\theta+1}^b) \implies (H_{\theta+1}^{b+1})\big)$ of the induction on b. Consider any bound $b \in \mathbb{N}$ and assume that $(H_{\theta+1}^b)$ holds. Recall that (H_θ^c) holds for every $c \in \mathbb{N}$. Let (q,p) be any abstract configuration in $\mathcal{C}_{\theta+1}^\sharp$ such that there is in $[\![\mathcal{A}]\!]_{\theta+1}^\sharp$ a path $\pi^\sharp$ of length at most $b+1$ from $\mathit{Init}_{\theta+1}^\sharp$ to (q,p). We show that $p \in \Psi\big(\mathcal{L}_{\theta+1}^{b+1}(q)\big)$. Recall that $P_{\theta+1}$ refines P_θ and define $\hat{p} = [\,p\,]_{P_\theta}$, i.e. $\hat{p}$ is the class in P_θ that contains p. Observe that $(q,\hat{p})$ is an abstract configuration in $\mathcal{C}_\theta^\sharp$. The "lift" of $\pi^\sharp$ to P_θ yields a path of length at most $b+1$ in $[\![\mathcal{A}]\!]_\theta^\sharp$ from $\mathit{Init}_\theta^\sharp$ to $(q,\hat{p})$. We deduce from (H_θ^{b+1}) that $\hat{p} \in \Psi\big(\mathcal{L}_\theta^{b+1}(q)\big)$. Since $\mathcal{L}_\theta^{b+1} \subseteq \mathcal{L}_{\theta+1}^{b+1}$ we obtain that $\hat{p} \in \Psi\big(\mathcal{L}_{\theta+1}^{b+1}\big)$. If $p \in P_\theta(q)$ then $p = \hat{p} \in \Psi\big(\mathcal{L}_\theta^{b+1}(q)\big)$. Otherwise, $p \in P_{\theta+1}(q) \setminus P_\theta(q)$ and we get that p was added to $P(q)$ during the iteration θ at line 9. Let us write $\pi_\theta^\sharp$ as $\pi_\theta^\sharp = (q_0,p_0) \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} (q_h,p_h)$. We deduce from line 9 that $p \in \left\{p_{i_0} \cap L_{i_0}^\theta, p_{i_0} \cap \overline{L_{i_0}^\theta}\right\}$ for some $0 \le i_0 \le h$ such that $q = q_{i_0}$. Moreover $p_{i_0} = \hat{p}$ since p_{i_0} and $\hat{p}$ both contain p. Remark that we may replace in $\pi_\theta^\sharp$ the prefix $(q_0,p_0) \xrightarrow{l_0} \cdots \xrightarrow{l_{i_0-1}} (q_{i_0},\hat{p})$ with the "lift" of $\pi^\sharp$ to P_θ. The resulting abstract path is also an abstract counterexample in $[\![\mathcal{A}]\!]_\theta^\sharp$, and its length is $h - i_0 + (b+1)$. Since $\pi_\theta^\sharp$ is among the shortest ones, we get that $i_0 \le b+1$. As $p_{i_0} = \hat{p} \in \Psi\big(\mathcal{L}_\theta^{b+1}(q)\big)$, we conclude that $p \in \Psi\big(\mathcal{L}_\theta^{b+1}(q)\big)$. □

Lemma C.5. *Consider any breadth-first execution of* CEGAR$(\mathcal{A}, \mathit{Init}, \mathit{Bad}, P_0,$ PathInv$)$*, and define* $\mathcal{I}_\theta = \left\{p \cap \{\boldsymbol{w} \mid (q,\boldsymbol{w}) \in \mathit{Init}\} \;\middle|\; (q,p) \in \mathit{Init}_\theta^\sharp\right\}$ *for any iteration* θ*. It holds that* $\mathcal{I}_\theta \subseteq \mathcal{I}_{\theta-1}$ *for any iteration* $\theta \ge 1$*.*

Proof. Consider an iteration $\theta \ge 1$ and let $L \in \mathcal{I}_\theta$. There exists $(q,p) \in \mathit{Init}_\theta^\sharp$ such that $L = p \cap \{\boldsymbol{w} \mid (q,\boldsymbol{w}) \in \mathit{Init}\}$. Notice that $\{q\} \times L = (\{q\} \times p) \cap \mathit{Init} \neq \emptyset$. If $(q,p) \in \mathit{Init}_{\theta-1}^\sharp$ then $L \in \mathcal{I}_{\theta-1}$. Otherwise, we obtain from Lemma C.3 that $(\{q\} \times p_0) \cap \mathit{Init} = (\{q\} \times p) \cap \mathit{Init}$ for some $p_0 \in P_{\theta-1}(q)$. We thus come to $L = p_0 \cap \{\boldsymbol{w} \mid (q,\boldsymbol{w}) \in \mathit{Init}\}$. Since $L \neq \emptyset$, we get that $(q,p_0) \in \mathit{Init}_{\theta-1}^\sharp$ and we conclude that $L \in \mathcal{I}_{\theta-1}$. □

The following proposition shows that for any bound b, there is an iteration after which the abstract configurations that are reachable from $\mathit{Init}^\sharp$ by a path

of length at most b are never split, or, put differently, the "reachability set up to depth b" of the abstraction remains constant.

Proposition C.6. *For any $b \in \mathbb{N}$ and for any non-terminating breadth-first execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv})$, the two following sets are finite:*

$$\bigcup_{\theta \in \mathbb{N}} Reach_\theta^{\leq b} \qquad \text{and} \qquad \left\{ L_i^\theta \;\middle|\; \theta \in \mathbb{N}, i \leq h_\theta \text{ and } i \leq b \right\}$$

Proof. We prove the proposition by induction on b. Let us first show the basis. For any $\theta \in \mathbb{N}$, define $\mathcal{I}_\theta$ as in Lemma C.5. We infer from Lemma C.5 that $\mathcal{I}_\theta \subseteq \mathcal{I}_0$. We derive from the definition of the algorithm UPInv that for any iteration $\theta \in \mathbb{N}$, there exists $(q, p) \in Init_\theta^\sharp$ and $k \in \mathbb{N}$ such that $L_0^\theta = \nabla_k \left(p \cap \{\boldsymbol{w} \,|\, (q, \boldsymbol{w}) \in Init\}\right)$, and therefore $L_0^\theta = \nabla_k(L)$ for some $L \in \mathcal{I}_\theta$. Recall that according to Definition 4.3, the set $\{\nabla_k(L) \mid k \in \mathbb{N}\}$ is finite for any recognizable subset L of $(M^*)^n$. Since $\mathcal{I}_0$ is finite, we obtain that $\{\nabla_k(L) \mid L \in \mathcal{I}_0, k \in \mathbb{N}\}$ is finite. Consequently, the set $\left\{ L_0^\theta \;\middle|\; \theta \in \mathbb{N} \right\}$ is finite. Moreover, according to Lemma C.4, we have $p \in \Psi\big(P_0(q) \cup \left\{ L_0^\theta \;\middle|\; \theta \in \mathbb{N} \right\}\big)$ for every $(q, p) \in Reach_\theta^{\leq 0}$. We deduce that $\bigcup_{\theta \in \mathbb{N}} Reach_\theta^{\leq 0}$ is finite.

Let us now show the induction step. Assume that the proposition holds for some bound $b \in \mathbb{N}$. Let us define $\mathcal{H} = \left\{ L_b^\theta \cap p \;\middle|\; \theta \in \mathbb{N}, b \leq h_\theta, (q, p) \in Reach_\theta^{\leq b} \right\}$. The sets $\bigcup_{\theta \in \mathbb{N}} Reach_\theta^{\leq b}$ and $\left\{ L_b^\theta \;\middle|\; \theta \in \mathbb{N}, b \leq h_\theta \right\}$ are both finite according to the induction hypothesis, and therefore $\mathcal{H}$ is finite. We derive from the definition of the algorithm UPInv that for any iteration $\theta \in \mathbb{N}$ with $h_\theta \geq b + 1$, if L_{b+1}^θ is non-empty then there exists $(q, p) \in Reach_\theta^{\leq b}$, $l \in \Sigma$ and $k \in \mathbb{N}$ such that $L_{b+1}^\theta = \nabla_k \left(post(l, L_b^\theta \cap p)\right)$, and therefore $L_{b+1}^\theta = \nabla_k \left(post(l, L)\right)$ for some $L \in \mathcal{H}$. Recall that according to Definition 4.3, the set $\{\nabla_k (L) \mid k \in \mathbb{N}\}$ is finite for any subset L of $(M^*)^n$. Since $\mathcal{H}$ and Σ are both finite, we obtain that $\{\nabla_k (post(l, L)) \mid l \in \Sigma, L \in \mathcal{H}, k \in \mathbb{N}\}$ is finite. We deduce that the set $\left\{ L_{b+1}^\theta \;\middle|\; \theta \in \mathbb{N}, b + 1 \leq h_\theta \right\}$ is finite, and we get from the induction hypothesis that $\left\{ L_i^\theta \;\middle|\; \theta \in \mathbb{N}, i \leq h_\theta, i \leq b + 1 \right\}$ is also finite. Moreover, according to Lemma C.4, we have $p \in \Psi\big(P_0(q) \cup \left\{ L_i^\theta \;\middle|\; \theta \in \mathbb{N}, i \leq h_\theta, i \leq b + 1 \right\}\big)$ for every $(q, p) \in Reach_\theta^{\leq b+1}$. We deduce that $\bigcup_{\theta \in \mathbb{N}} Reach_\theta^{\leq b+1}$ is finite. □

Proposition 5.2. *For any breadth-first execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$, if the execution does not terminate then the sequence $(h_\theta)_{\theta \in \mathbb{N}}$ of lengths of counterexamples picked at line 2 is nondecreasing and diverges.*

Proof. Consider a non-terminating breadth-first execution and let us show that the sequence $(h_\theta)_{\theta \in \mathbb{N}}$ is nondecreasing and diverges. Let $\eta, \theta \in \mathbb{N}$ such that $\eta < \theta$, and observe that the partition map P_θ refines P_η. The "lift" of $\pi_\theta^\sharp$ to P_η yields a counterexample in $[\![\mathcal{A}]\!]_\eta^\sharp$. Since $\pi_\eta^\sharp$ is a counterexample in $[\![\mathcal{A}]\!]_\eta^\sharp$ among the shortest ones, we get that its length h_η statisfies $h_\eta \leq h_\theta$. This concludes the proof that $(h_\theta)_{\theta \in \mathbb{N}}$ is nondecreasing.

By contradiction, assume that there exists $b, \theta_1 \in \mathbb{N}$ such that $h_\theta = b$ for every $\theta \geq \theta_1$. We obtain from Proposition C.6 that $\bigcup_{\theta \in \mathbb{N}} Reach_\theta^{\leq b}$ is finite.

Therefore, there exists $\theta_2 \geq \theta_1$ such that $Reach^{\leq b}_{\theta_2} = Reach^{\leq b}_{\theta_2+1}$. Let us write $\pi^\sharp_{\theta_2}$ as $\pi^\sharp_{\theta_2} = (q_0, p_0) \xrightarrow{l_0} \cdots \xrightarrow{l_{b-1}} (q_b, p_b)$. Observe that $(q_i, p_i) \in Reach^{\leq b}_{\theta_2+1}$ for every $0 \leq i \leq b$. We deduce that $\pi^\sharp_{\theta_2}$ is also a counterexample in $[\![\mathcal{A}]\!]^\sharp_{\theta_2+1}$, which contradicts the fact that P_{θ_2+1} is a refinement of P_{θ_2} that rules out $\pi^\sharp_{\theta_2}$. We conclude that $(h_\theta)_{\theta\in\mathbb{N}}$ diverges. □

Corollary 5.3. *If $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-unsafe then any breadth-first execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$ terminates.*

Proof. Assume that there exists in $[\![\mathcal{A}]\!]$ a path π from $Init$ to Bad and let b denote the length of π. Consider any breadth-first execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$. Observe that for any iteration θ, $\alpha_\theta(\pi)$ is an abstract counterexample of length b in $[\![\mathcal{A}]\!]^\sharp_\theta$. Hence, we have $h_\theta \leq b$ for every iteration $\theta \in \mathbb{N}$, and we conclude with Proposition 5.2 that the execution terminates. □

Proposition 5.5. *Assume that ∇ is restricted. If $[\![\mathcal{A}]\!]$ has a finite reachability set from $Init$, then any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$ terminates.*

Proof. Assume that $[\![\mathcal{A}]\!]$ has a finite reachability set from $Init$, and consider any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$. For each $q \in Q$, let us write $RS(q)$ the finite set of $\boldsymbol{w} \in (M^*)^n$ such that there is a path in $[\![\mathcal{A}]\!]$ from $Init$ to $(q, \boldsymbol{w})$. Define $\mathcal{L} = \bigcup_{q\in Q} RS(q)$ and remark that $\mathcal{L}$ is finite. Recall that according to Definition 4.3, for any recognizable subset L of $(M^*)^n$, there exists $k_L \in \mathbb{N}$ such that $L = \nabla_k(L)$ for every $k \geq k_L$. Since $\mathcal{L}$ is finite, we infer that there exists $K \in \mathbb{N}$ such that $L = \nabla_k(L)$ for every $k \geq K$ and $L \subseteq \mathcal{L}$. Let us define $\mathcal{H} = \wp(\mathcal{L}) \cup \{\nabla_k(L) \mid k < K, L \in \mathcal{R}ec((M^*)^n)\}$. Observe that $\mathcal{H}$ is finite since ∇ is restricted.

We show that $L^\theta_i \in \mathcal{H}$ for any iteration θ and for any $0 \leq i \leq h_\theta$. Consider an iteration θ, and let us write $\pi^\sharp_\theta$ as $\pi^\sharp_\theta = (q_0, p_0) \xrightarrow{l_0} \cdots \xrightarrow{l_{h-1}} (q_h, p_h)$, with $h = h_\theta$. Notice that (q_{i-1}, l_{i-1}, q_i) is a transition rule in Δ for each $1 \leq i \leq h$. Let us define $R_0 = p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in Init\}$ and $R_i = post(l_{i-1}, L^\theta_{i-1} \cap p_{i-1})$ for every $1 \leq i \leq h$. We derive from the definition of the algorithm UPinv that there exists $k \in \mathbb{N}$ such that: $L^\theta_0 = \nabla_k(R_0)$, and $L_i = \emptyset$ or $L_i = \nabla_k(R_i)$ for every $1 \leq i \leq h$. If $k < K$ then we get that $L^\theta_i \in \mathcal{H}$ for every $0 \leq i \leq h$. Otherwise, we have $k \geq K$ and therefore $L_i = R_i$ for every $0 \leq i \leq h$. An immediate induction on i shows that $R_i \subseteq RS(q_i)$ for every $0 \leq i \leq h$. We deduce that $L^\theta_i \subseteq \mathcal{L}$ and hence $L^\theta_i \in \mathcal{H}$ for every $0 \leq i \leq h$.

We obtain that $\left\{L^\theta_i \;\middle|\; \theta \in \mathbb{N} \text{ and } 0 \leq i \leq h_\theta\right\} \subseteq \mathcal{H}$. Since $\mathcal{H}$ is finite, we conclude with Proposition C.2 that the execution terminates. □

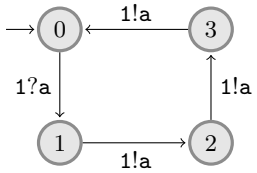

Fig. 9: Fifo system of Example D.1 showing non-termination of CEGAR.

D Non-Termination of CEGAR for Finite Fifo Systems

The following example shows that the CEGAR semi-algorithm may not terminate in general for fifo systems with a finite reachability set.

Example D.1. Consider the fifo system $\mathcal{A}$ depicted in Figure 9. This fifo system has a single message a and a single queue. The safety condition $(Init, Bad)$ is defined by the recognizable subsets $Init = \{(0, \varepsilon)\}$ and $Bad = \{0\} \times (\{a\} \cdot \{aa\}^*)$. Notice that the reachability set from $Init$ is equal to $Init$, which is finite, and hence $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe. Define the initial partition map P_0 by $P_0(q) = \{\{a\}^*\}$. We consider the extrapolation ∇ defined by $\nabla_0(\{\varepsilon\}) = \{\varepsilon, aa\}$ and $\nabla_k(L) = L$ if $k > 0$ or $L \neq \{\varepsilon\}$. Let us detail the first iterations of an execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$.

0. $\pi_0^\sharp$ is the empty path $(0, \{a\}^*)$ and the path invariant is $(\{\varepsilon, aa\})$.
1. $\pi_1^\sharp = (0, \{\varepsilon, aa\}) \xrightarrow{1?a} (1, \{a\}^*) \xrightarrow{1!a} (2, \{a\}^*) \xrightarrow{1!a} (3, \{a\}^*) \xrightarrow{1!a} (0, \overline{\{\varepsilon, aa\}})$, and the path invariant is $(\{\varepsilon, aa\}, \{a\}, \{aa\}, \{a^3\}, \{a^4\})$.
2. $\pi_2^\sharp = (0, \{\varepsilon, aa\}) \xrightarrow{1?a} (1, \{a\}) \xrightarrow{1!a} (2, \{aa\}) \xrightarrow{1!a} (3, \{a^3\}) \xrightarrow{1!a} (0, \{a^4\}) \xrightarrow{1?a} (1, \overline{\{a\}}) \xrightarrow{1!a} (2, \overline{\{aa\}}) \xrightarrow{1!a} (3, \overline{\{a^3\}}) \xrightarrow{1!a} (0, \overline{\{\varepsilon, aa, a^4\}})$, and the path invariant is $(\{\varepsilon, aa\}, \{a\}, \{a^2\}, \{a^3\}, \{a^4\}, \{a^3\}, \{a^4\}, \{a^5\}, \{a^6\})$.

These first iterations suggest that the execution may not terminate, and we can actually prove that it necessarily does not terminate. Consider any execution of $\mathsf{CEGAR}(\mathcal{A}, Init, Bad, P_0, \mathsf{UPInv}(\nabla))$. For any iteration θ, the path invariant $(L_0^\theta, \ldots, L_{h_\theta}^\theta)$ computed by $\mathsf{UPInv}(\nabla)$ satisfies $L_0^\theta = \{\varepsilon, aa\}$ and $L_{4i}^\theta = \{a^4 \cdot a^{2(i-1)}\}$ for any $1 \leq i \leq \frac{h_\theta}{4}$. We deduce that, for each iteration θ, there exists a finite subset F_θ of $\{a\}^*$ such that $\{\{\varepsilon, aa\}, \overline{F_\theta}\} \subseteq P_\theta(0)$. Observe that $(0, \{\varepsilon, aa\}) \in Init_\theta^\sharp$ and $(0, \overline{F_\theta}) \in Bad_\theta^\sharp$. Moreover, for every $i \geq 1$, there is a concrete path in $[\![\mathcal{A}]\!]$ from $(0, aa)$ to $(0, a^{2i})$. Hence, there is an abstract path in $[\![\mathcal{A}]\!]_\theta^\sharp$ from $(0, \{\varepsilon, aa\})$ to $(0, \overline{F_\theta})$. We obtain that $[\![\mathcal{A}]\!]_\theta^\sharp$ is $(Init^\sharp, Bad^\sharp)$-unsafe for every iteration θ, which, combined with Proposition 5.1, implies that the execution does not terminate since $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe. □