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 `{heussner, sutre}@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. 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).

CEGAR. 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 that occur 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 two 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 (small to medium size) 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 *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 that are 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 entire system) at each refinement step. In contrast, our path invariant generation procedures only use the precision *required* for each spurious counterexample. Preliminary benchmarks demonstrate the benefit of our local and adaptive approach for the larger examples, where a "highly" precise abstraction is required only for a few control loops. Last, our approach is not tied to words and automata. In this work we only focus 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 analyze its correctness and termination. Section 6 provides an overview of the extrapolation used in our implementation. Experimental results are presented in Section 7, along with some perspectives.

Due to space limitations, all proofs were omitted in this paper. A long version with detailed proofs and additional material can be obtained from the authors.

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 non-empty set of *letters*). We write Σ^* for the set of all *finite words* (*words* for short) over Σ, and we let ε denote the *empty word.* For any two words $w, w' \in \Sigma^*$, we write $w \cdot w'$ for their *concatenation.* A *language* is any subset of Σ^*. For any language L, we denote by L^* its *Kleene closure* and we write $L^+ = L \cdot L^*$. The *alphabet of* L, written $alph(L)$, is the least subset A of Σ such that $L \subseteq A^*$. For any word $w \in \Sigma^*$, the singleton language $\{w\}$ 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 $LTS = \langle \mathcal{C}, \Sigma, \rightarrow \rangle$ where $\mathcal{C}$ is a set of *configurations*,

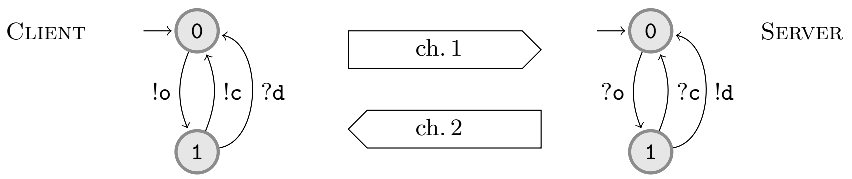

Fig. 1. The Connection/Disconnection Protocol [JR86]

Σ is a finite set of *actions* and $\rightarrow \subseteq \mathcal{C} \times \Sigma \times \mathcal{C}$ is a (labeled) *transition relation.* We say that *LTS* is *finite* when $\mathcal{C}$ is finite. For simplicity, we will often write $c \xrightarrow{l} c'$ in place of $(c, l, c') \in \rightarrow$.

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 $\mathit{Init} \subseteq \mathcal{C}$ and $\mathit{Bad} \subseteq \mathcal{C}$ 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 $c_0 \in \mathit{Init}$ and $c_h \in \mathit{Bad}$. Observe that if $c \in \mathit{Init} \cap \mathit{Bad}$ 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 $\mathit{LTS} = \langle \mathcal{C}, \Sigma, \rightarrow \rangle$ is a pair $(\mathit{Init}, \mathit{Bad})$ of subsets of $\mathcal{C}$. We say that *LTS* is $(\mathit{Init}, \mathit{Bad})$*-unsafe* if there is a path from *Init* to *Bad* in *LTS*, which is called a *counterexample.* We say that *LTS* is $(\mathit{Init}, \mathit{Bad})$*-safe* when it is not $(\mathit{Init}, \mathit{Bad})$-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.

Example 2.1. The connection/disconnection protocol [JR86] – abbreviated as 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.

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

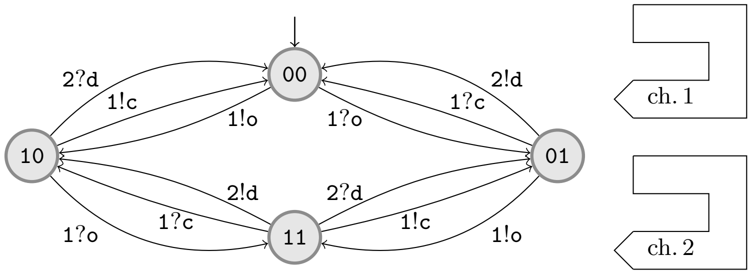

Fig. 2. Fifo System Representing the Connection/Disconnection Protocol

– 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 on 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 below.

Definition 2.3. *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 that satisfy 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* $j \neq i$*.*

Example 2.4. 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. The set of initial configurations is $Init = \{(\mathtt{00}, \varepsilon, \varepsilon)\}$. A set of bad configurations for this protocol is $Bad = \{\mathtt{00}, \mathtt{10}\} \times (\mathtt{c} \cdot M^* \times M^*)$. This set contains configurations where the server is in control state `0` but the first message in the first 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 focus on 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 $S = \bigcup_{p \in P} p$. Elements p of a partition P are called *classes*. For any element s in S, we denote by $[\, s \,]_P$ the class in P 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 $\overline{L} = (M^*)^n \setminus L$ for 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 $(M^*)^n$ [BGWW97], 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 finite 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:

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

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

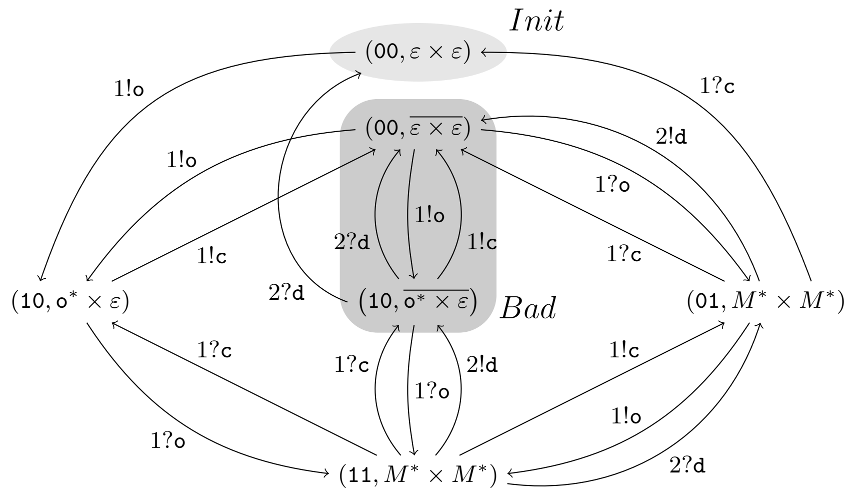

Fig. 3. Example Partition Abstraction of the C/D Protocol (Example 3.5)

abstract transition relation $\rightarrow^\sharp$ is the existential lift of the concrete transition relation $\rightarrow$ to abstract configurations.

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

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

Obviously, $post(l, L)$ and $pre(l, L)$ are effectively recognizable subsets of $(M^*)^n$ for any $l \in \Sigma$ and any recognizable subset $L \subseteq (M^*)^n$. Moreover, we may use *post* and *pre* to characterize the abstract transition relation of a partition abstraction $[\![\mathcal{A}]\!]^\sharp_P$, as follows: for any rule $(q, l, q') \in \Delta$ and for any pair $(p, p') \in P(q) \times P(q')$, 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 $P : Q \rightarrow \mathbb{P}((M^*)^n)$, $[\![\mathcal{A}]\!]^\sharp$ 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 generally. 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 $\mathcal{A}$, any 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 partition abstraction induced by the following partition map:

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

The set of initial abstract configurations is $\alpha(Init) = \{(00, \varepsilon \times \varepsilon)\}$, and the set of bad abstract configurations is $\alpha(Bad) = \{(00, \overline{\varepsilon \times \varepsilon}), (10, \overline{\mathtt{o}^* \times \varepsilon})\}$. The resulting partition abstraction is the finite labeled transition system depicted in Figure 3. A simple graph search reveals several abstract counterexamples, for instance $\pi^\sharp = (00, \varepsilon \times \varepsilon) \xrightarrow{1!\mathtt{o}}{}^\sharp (10, \mathtt{o}^* \times \varepsilon) \xrightarrow{1!\mathtt{c}}{}^\sharp (00, \overline{\varepsilon \times \varepsilon})$. This counterexample is spurious since the only concrete path that corresponds to $\pi^\sharp$ (i.e., whose image under α is $\pi^\sharp$) is $\pi = (00, \varepsilon, \varepsilon) \xrightarrow{1!\mathtt{o}} (10, \mathtt{o}, \varepsilon) \xrightarrow{1!\mathtt{c}} (00, \mathtt{oc}, \varepsilon) \notin Bad$.

4 Counterexample-based Partition Refinement

The abstraction-based verification of safety properties relies on refinement techniques that 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 concept 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 $\pi^\sharp_{\widetilde{P}} = (q_0, \widetilde{p}_0) \xrightarrow{l_0}{}^\sharp \cdots \xrightarrow{l_{h-1}}{}^\sharp (q_h, \widetilde{p}_h)$ 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, that are computationally "simple" but "precise enough" to rule out spurious counterexamples. 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 $L_0, \dots, L_h$ of recognizable subsets of $(M^*)^n$ such that:*

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

Observe that condition (ii) is more general than $\mathit{post}(l_i, L_i) \subseteq L_{i+1}$ 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 $L_0, \dots, L_h$ 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 $q \in Q$.

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 adequate for the design of path invariant generation procedures. The most simple extrapolation is the *identity 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. 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 discussed in Section 6.

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

$k \leftarrow 0$
do
    $L_0 \leftarrow \nabla_k\,(p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in \mathit{Init}\})$
    for $i$ from 1 upto $h$
        $F_i \leftarrow \mathit{post}(l_{i-1}, p_{i-1} \cap L_{i-1})$
        if $p_i \cap F_i = \emptyset$
            $L_i \leftarrow \emptyset$
        else
            $L_i \leftarrow \nabla_k(F_i)$
    $k \leftarrow k + 1$
while $(\{q_h\} \times L_h) \cap \mathit{Bad} \neq \emptyset$
return $(L_0, \ldots, L_h)$
```

```
Split $(\nabla, L_0, L_1)$
Input: extrapolation $\nabla$, disjoint recognizable subsets $L_0$, $L_1$ of $(M^*)^n$

$k \leftarrow 0$
while $\nabla_k(L_0) \cap L_1 \neq \emptyset$
    $k \leftarrow k + 1$
return $\nabla_k(L_0)$
```

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

$B_h \leftarrow p_h \cap \{\boldsymbol{w} \mid (q_h, \boldsymbol{w}) \in \mathit{Bad}\}$
$i \leftarrow h$
while $B_i \neq \emptyset$ and $i > 0$
    $i \leftarrow i - 1$
    $B_i \leftarrow p_i \cap \mathit{pre}(l_i, B_{i+1})$
if $i = 0$
    $I \leftarrow p_0 \cap \{\boldsymbol{w} \mid (q_0, \boldsymbol{w}) \in \mathit{Init}\}$
    $L_0 \leftarrow$ Split $(\nabla, I, B_0)$
else
    $(L_0, \ldots, L_i) \leftarrow ((M^*)^n, \ldots, (M^*)^n)$
for $j$ from $i$ upto $h - 1$
    $L_{j+1} \leftarrow$ Split $(\nabla, \mathit{post}(l_j, p_j \cap L_j), B_{j+1})$
return $(L_0, \ldots, L_h)$
```

Fig. 4. Extrapolation-based Path Invariant Generation Algorithms

Remark 4.4. Extrapolations are closed under various operations, such as functional composition, union and intersection, as well as round-robin combination.

We now present two extrapolation-based path invariant generation procedures (Figure 4). 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 a 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 4.5. *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 $\pi^\sharp_P$.*

Lemma 4.6. *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 4.7. *For any spurious abstract counterexample $\pi^\sharp_P$, the execution of $\mathsf{APInv}(\nabla, Init, Bad, \pi^\sharp_P)$ terminates and returns a path invariant for $\pi^\sharp_P$.*

Example 4.8. Consider again the c/d protocol, and assume an extrapolation ∇ satisfying $\nabla_0(L \times \varepsilon) = (alph(L))^* \times \varepsilon$ for all $L \subseteq M^*$, and $\nabla_1(u \times \varepsilon) = u \times \varepsilon$ for each $u \in \{\varepsilon, \mathsf{o}, \mathsf{oc}\}$, e.g., the extrapolation ρ'' presented in Remark 6.1. The UPInv algorithm, applied to the spurious counterexample $(\mathsf{00}, \varepsilon \times \varepsilon) \xrightarrow{1!\mathsf{o}}{}^\sharp$ $(\mathsf{10}, \mathsf{o}^* \times \varepsilon) \xrightarrow{1!\mathsf{c}}{}^\sharp$ $(\mathsf{00}, \overline{\varepsilon \times \varepsilon})$ of Example 3.5, would perform two iterations of the **while**-loop and produce the path invariant $(\varepsilon \times \varepsilon,\ \mathsf{o} \times \varepsilon,\ \mathsf{oc} \times \varepsilon)$. These iterations are detailed in the table below. The mark $\lightning$ or ✓ indicates whether the condition at line 11 is satisfied.

	L_0	L_1	L_2	Line 11
$k = 0$	$\varepsilon \times \varepsilon$	$\mathsf{o}^* \times \varepsilon$	$\{\mathsf{o}, \mathsf{c}\}^* \times \varepsilon$	$\lightning$
$k = 1$	$\varepsilon \times \varepsilon$	$\mathsf{o} \times \varepsilon$	$\mathsf{oc} \times \varepsilon$	✓

Following Proposition 4.2, the partition map would be refined to:

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

This refined partition map clearly rules out the spurious counterexample.

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-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. ↯) then $[\![\mathcal{A}]\!]$ is $(Init, Bad)$-safe (resp. $(Init, Bad)$-unsafe).*

A detailed example execution of CEGAR on the c/d protocol is provided in the long version. Termination of the CEGAR semi-algorithm cannot be assured as, otherwise, it would solve the general reachability problem, which is 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 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 $Init$. In contrast 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 locally around the abstract configurations which 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

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

∇ 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 in the long version, this condition is not sufficient to guarantee that CEGAR $(\mathcal{A}, Init, Bad, P_0,$ 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 presented in [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 CEGAR $(\mathcal{A}, Init, Bad,$ $P_0,$ UPInv $(\nabla))$, the execution terminates if and only if the number of iterations of the **while**-loop of the algorithm UPInv is bounded[3]. 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 CEGAR$(\mathcal{A}, Init, Bad, P_0,$ UPInv $(\nabla))$ terminates.*

[3] Remark that this bound is not a bound on the length of abstract counterexamples.

6 Overview of the (Colored) Bisimulation Extrapolation

This section briefly introduces the bisimulation-based extrapolation underlying the widening operator introduced in [LGJJ06]. This extrapolation assumes an automata representation of recognizable subsets of $(M^*)^n$, and relies on bounded-depth bisimulation over the states of the automata. For simplicity, we focus in this section on fifo systems with a single queue, i.e., $n = 1$. In this simpler case, recognizable subsets of $(M^*)^n$ are regular languages contained in M^*, which can directly be represented by finite automata over M. The general case of $n \geq 2$, which is discussed in detail in the long version, requires the use of QDDs, that are finite automata accepting recognizable subsets of $(M^*)^n$ via an encoding of n-tuples in $(M^*)^n$ by words over an extended alphabet. Still, the main ingredients rest the same.

In the remainder of this section, we lift our discussion from regular languages in M^* to finite automata over M. Consider a finite automaton over M with a set Q of states. As in abstract regular model checking [BHV04], we use quotienting under equivalence relations on Q to obtain over-approximations of the automaton. However, we follow the approach of [LGJJ06], and focus on bounded-depth bisimulation equivalence (other equivalence relations were used in [BHV04]).

Given a priori an equivalence relation *col* on Q, also called "coloring", and a bound $k \in \mathbb{N}$, the (colored) *bisimulation equivalence of depth k* is the equivalence relation $\sim_k^{col}$ on Q defined as expected: $\sim_0^{col} = col$ and two states are equivalent for $\sim_{k+1}^{col}$ if (1) they are $\sim_k^{col}$-equivalent and (2) they have $\sim_k^{col}$-equivalent m-successors for each letter $m \in M$. The ultimately stationary sequence $\sim_0^{col} \supseteq \sim_1^{col} \supseteq \cdots \supseteq \sim_k^{col} \supseteq \sim_{k+1}^{col} \supseteq \cdots$ of equivalence relations on Q leads to the colored bisimulation-based extrapolation.

We define a coloring *std*, called *standard coloring*, by $(q_1, q_2) \in std$ if either q_1 and q_2 are both final states or q_1 and q_2 are both non-final states. The *bisimulation extrapolation* is the function ρ from $\mathbb{N}$ to $\mathcal{R}ec\,(M^*) \to \mathcal{R}ec\,(M^*)$ defined by $\rho_k(L) = L/{\sim_k^{std}}$, where L is identified to a finite automaton accepting it. Notice that ρ is a restricted extrapolation.

Remark 6.1. We could also choose other colorings or define the sequence of equivalences in a different way. For instance, better results are sometimes obtained in practice with the extrapolation ρ' that first (for $k = 0$) applies a quotienting with respect to the equivalence relation $Q \times Q$ (i.e., all states are merged), and then behaves as ρ_{k-1} (for $k \geq 1$). Analogously, the extrapolation ρ'' defined by $\rho''_0 = \rho'_0$ and $\rho''_k = \rho_k$ for $k \geq 1$ was used in Example 4.8.

Example 6.2. Consider the regular language $L = \{aac, baaa\}$ over the alphabet $M = \{a, b, c, d, e\}$, represented by the automaton FA_L of Figure 5a. The previously defined extrapolation ρ applies to L as follows: ρ_0 splits the states of FA_L according to *std*, hence, $\rho_0(L) = \{a, b\}^* \cdot \{a, c\}$ (viz. Figure 5c). Then ρ_1 merges the states that are bisimulation equivalent up to depth 1, i.e., the states ○ of FA_L (Figure 5d). As all states of FA_L are non equivalent for $\sim_k^{std}$ with $k \geq 2$, we

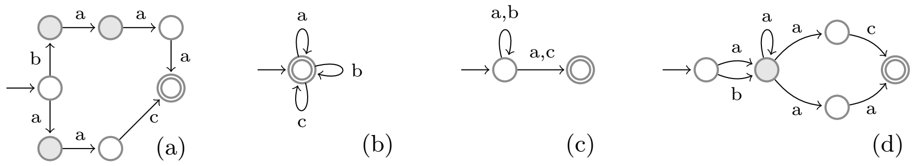

Fig. 5. Finite Automata Representations for Extrapolating L (Example 6.2)

have $\rho_k(L) = L$ (again Figure 5a). The variants ρ' and ρ'' mentioned previously would lead to $\rho'_0(L) = \rho''_0(L) = (alph(L))^* = \{a, b, c\}^*$ (viz. Figure 5b).

The benefits of the bisimulation extrapolation for the abstraction of fifo systems were already discussed in [LGJJ06]. The following example shows that this extrapolation can, in some common cases, discover *exact* repetitions of message sequences in queues, without the need for acceleration techniques.

Example 6.3. Let us continue the running example of the c/d protocol, considered here as having a single-queue by restricting it to operations on the first queue. The application of acceleration techniques on the path $(\mathsf{00}, \varepsilon) \xrightarrow{1!\mathsf{o}}\xrightarrow{1!\mathsf{c}} (\mathsf{00}, \mathsf{oc}) \xrightarrow{1!\mathsf{o}}\xrightarrow{1!\mathsf{c}} (\mathsf{00}, \mathsf{ococ}) \cdots$ would produce the set of queue contents $(\mathsf{oc})^+$. The bisimulation extrapolation ρ applied to the singleton language $\{\mathsf{ococ}\}$, represented by the obvious automaton, produces the following results for the first two parameters: $\rho_0(\{\mathsf{ococ}\}) = \{\mathsf{o}, \mathsf{c}\}^* \cdot \mathsf{c}$ and $\rho_1(\{\mathsf{ococ}\}) = (\mathsf{oc})^+$.

7 Experimental Evaluation

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 finite automata and Qdd operations, the fifo *post*/*pre* symbolic computations, as well as the colored bisimulation-based extrapolation. The standard coloring with final and non-final states is used by default in our tool, but several other variants are also available.

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' for larger examples. 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 interfere on their shared channels, a sliding window protocol, as well as protocols for leader election due to Peterson

protocol	states/trans.	refmnt.	time [s]	mem [MiB]	loops	$\text{states}^\sharp/\text{trans}^\sharp$
ABP	16/64	APInv'	0.30	1.09	72	87/505
		UPInv	2.13	1.58	208	274/1443
c/d protocol	5/17	APInv'	0.02	0.61	8	12/51
		UPInv	0.01	0.61	6	11/32
nested c/d protocol	6/17	APInv'	0.68	1.09	80	85/314
		UPInv	1.15	1.09	93	100/339
non-regular protocol	9/18	APInv'	0.02	0.61	13	21/47
		UPInv	0.06	0.61	14	25/39
Peterson	10648/56628	APInv'	7.05	32.58	299	10946/58536
		UPInv	2.14	32.09	51	10709/56939
(simplified) TCP	196/588	APInv'	2.19	3.03	526	721/3013
		UPInv	1.38	2.06	183	431/1439
server with 2 clients	255/2160	APInv'	$(> 1h)$	—	—	—
		UPInv	9.61	4.97	442	731/7383
token ring	625/4500	APInv'	85.15	19.50	1720	2344/19596
		UPInv	4.57	6.42	319	1004/6956
sliding window	225/2010	APInv'	16.43	9.54	1577	1801/15274
		UPInv	0.93	2.55	148	388/2367

Table 1. Benchmark results of McScM on a suite of protocols.

and token passing in a ring topology. Further, we provide certain touchstones for our approach: an enhancement of the c/d protocol with nested loops for the exchange of data, and a protocol with a non-recognizable reachability set. A detailed presentation of the protocols is provided in the long version. Except for the c/d protocol, which is unsafe, all other examples are safe.

Table 1 gives a summary of the results obtained by McScM on an off-the-shelf computer (2.4 GHz Intel Core 2 Duo). Breadth-first exploration was applied in all examples to search for abstract counterexamples. The bisimulation extrapolation ρ presented in Section 6 was used except for the server with 2 clients, where we applied the variant ρ' of ρ presented in Remark 6.1, as the analysis did not terminate after one hour with ρ. All examples are analyzed with UPInv in a few seconds, and memory is not a limitation.

We compared McScM with TReX [Tre], which is, to the best of our knowledge, the sole publicly available and directly usable model-checker for the verification of *unbounded* fifo systems. Note, however, that the comparison is biased as TReX focuses on *lossy* channels. We applied TReX to the first six examples of Table 1. TReX has an efficient implementation based on *simple regular expressions* (and not general QDDs as we do), and needs in most cases less than 1 second to build the reachability set (the latter allows to decide the reachability of bad configurations by a simple additional look-up). Further, TReX implements communicating timed and counter automata that are – at this stage – beyond the focus of our tool. Nonetheless, 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). Moreover, TReX suffers (as would also a symbolic model-checker based on the LASH library [Las]) from the main drawback of acceleration techniques, which in general cannot cope with nested loops, whereas they seem to have no adverse effect on our tool (viz. nested c/d protocol on which TReX did not finish after one 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]).

To obtain a finer evaluation of our approach, we prototypically implemented the global abstraction refinement scheme of [BHV04] in our tool. While this ARMC implementation seems to be advantageous for some small protocols, larger examples confirm that the local and adaptive approach refinement approach developed in this paper outperforms a global refinement one in protocols that demand a "highly" precise abstraction only for a few control loops (e.g., Peterson's leader election and token ring). Further, our ARMC implementation was not able to handle the non-regular protocol nor the server with 2 clients.

8 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.

Our approach relies on partition abstractions where equivalence classes are recognizable languages of queue contents. Our main contribution is the design of generic path invariant generation algorithms based on parametrized extrapolation operators for queue contents. Because of the latter, our CEGAR semi-algorithm enjoys additional partial termination properties.

The framework developed in this paper is not specific to fifo systems, and we intend to investigate its practical relevance to other families of infinite-state models. Future work also includes the safety verification of more complex fifo systems that would allow the exchange of unbounded numerical data over the queues, or include parametrization (e.g., over the number of clients). 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 (when equipped with the path invariant generation algorithms developed in this paper) for these classes.

Acknowledegments. We thank the anonymous reviewers for supporting and guiding the genesis of this publication, and we are especially grateful for the fruitful and challenging discussions with Jérôme Leroux and Anca Muscholl.

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.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The Power of QDDs. In *Proc. Static Analysis Symposium 1997*, volume 1302 of *LNCS*, pages 172–186. Springer, 1997.

[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.

[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. Springer, 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 Vérification des Algorithmes Distribués. Rapports de Recherche 590, IRISA Rennes, December 1986.

[Las] Liège Automata-based Symbolic Handler (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. Algebraic Methodology and Software Technology 2006*, volume 4019 of *LNCS*, pages 204–219. Springer, 2006.

[Mcs] Model Checker for Systems of Communicating Fifo Machines (McScM). Tool Homepage. `http://www.labri.fr/~heussner/mcscm/`.

[Scm] Tools and Libraries for Static Analysis and Verification. Tool Homepage. `http://gforge.inria.fr/projects/bjeannet/`.

[Tre] Tool for Reachability Analysis of CompleX Systems (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. Springer, 2008.