Algorithmic Analysis of Piecewise FIFO Systems

Naghmeh Ghafari⁎, Arie Gurfinkel†, Nils Klarlund‡, and Richard Trefler⁎

⁎David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
Email: ngghafari@swen.uwaterloo.ca, trefler@cs.uwaterloo.ca
†Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA
Email: arie@sei.cmu.edu
‡Google Inc.
Email: klarlund@google.com

Abstract—Systems consisting of several components that communicate via unbounded perfect FIFO channels (i.e. FIFO systems) arise naturally in modeling distributed systems. Despite well-known difficulties in analyzing such systems, they are of significant interest as they can describe a wide range of communication protocols. Previous work has shown that piecewise languages play an important role in the study of FIFO systems [16], [13].

In this paper, we present two algorithms for computing the set of reachable states of a FIFO system composed of piecewise components. The problem of computing the set of reachable states of such a system is closely related to calculating the set of all possible channel contents, i.e. the limit language. We present new algorithms for calculating the limit language of a system with a single communication channel and a class of multi-channel system in which messages are not passed around in cycles through different channels. We show that the worst case complexity of our algorithms for single-channel and important subclasses of multi-channel systems is exponential in the size of the initial content of the channels.

I. INTRODUCTION

Concurrent systems consisting of a set of finite state machines that communicate via unbounded First-In First-Out (FIFO) channels are a common model of computation for describing distributed protocols such as IP-telecommunication protocols, interacting web services, and System on Chip (SoC) architectures (e.g., [9], [5], [1], [18], [10], [6], [20]). Even though all physically constructible systems have finite size channels, their size is often an implementation parameter that is typically left unspecified. Thus, modeling with unbounded channels is more realistic than artificially bounding their size.

While unboundedness of communication channels provides a useful modeling abstraction, it complicates the analysis. Brand and Zafiropulo [9] showed that a single unbounded channel is already sufficient to simulate the tape of a Turing machine. Hence, verification of any non-trivial property, such as reachability, is undecidable. Despite these results, a substantial effort has gone into identifying subclasses of FIFO systems for which the verification problem is decidable (e.g., [1], [2], [3], [4], [5], [7], [8], [10], [16], [18]).

In this paper, we study the class of piecewise FIFO systems. These systems can be used for modeling distributed protocols such as IP-telecommunication protocols and interacting web services [16], [13]. A piecewise FIFO system is composed of components whose behaviors can be expressed by piecewise languages. Intuitively, a language is piecewise if it is accepted by a non-deterministic finite state automata whose only non-trivial strongly connected components are states with self-loops. Formally, a piecewise language is a union of sets of strings, where each set is given by a regular expression of the form \( M_0^*a_0M_1^* \cdots a_{n-1}M_n^* \), where each \( M_i \) is a subset of the alphabet \( \Sigma \) and each \( a_i \) is an element of \( \Sigma \). In [16], [13], verification problems for piecewise FIFO systems were described and shown to be decidable.

Although piecewise languages may look restrictive, they can be used to express descriptions of IP-telephony features [16], [13] and seem amenable to describing composite web services specified in Business Process Execution Language (BPEL) [14]. For example, [13] studied the behavior of the telephony features in BoxOS which is the next generation telecommunication service over IP developed in AT&T labs [6], [15], [21]. Essentially an active call is represented by a graph of telephony features (referred to as boxes) while communication between neighboring boxes are handled via unbounded perfect FIFO channels. At a sufficient level of abstraction, boxes may all be viewed as finite state transducers. It is required that the communication between different boxes follow a certain pattern. Thus, all of the feature boxes implement a communication template that consists of three phases (cf. [6]): setup phase, transparent phase, and teardown phase. Interestingly, as shown in [16], [13], this communication template can be expressed by piecewise languages.

The ability to calculate all possible channel contents that may arise from an initial state, i.e. the limit language, plays a central role for automated verification of non-trivial properties of FIFO systems. This problem is undecidable in general. Moreover, the limit language is not necessarily regular; even if the initial language is [10], and even when the limit language is known to be regular, determining it may still be undecidable [10]. Recently, in [16] it has been shown that for the piecewise FIFO systems, the limit language is regular even when conditional actions are considered. However, the construction of the limit language may not always be effective.

In this paper, we present an algorithmic analysis of piecewise FIFO systems along with complexity results. We present two new algorithms for computing the limit language: one for a single-channel system, and another for a multi-channel system with acyclic communication graph. In both algorithms,
we use automata to represent and manipulate the set of possible channel configurations.

The algorithm for single-channel systems requires that components be piecewise, and applies to any regular initial channel content. We show that the worst case complexity of the algorithm is at most exponential in the size of the automaton that represents the language of the initial channel content.

The algorithm for the multi-channel systems requires that both the components and the initial contents of the channels be piecewise, and that the communication graph be acyclic. A communication graph is a graph with channels as vertices and conditional actions as edges indicating which channels are connected by these actions. For ease of presentation, we develop the algorithm incrementally by restricting the topology of the communication graphs to star, tree, inverted tree, and directed acyclic graph (DAG) topologies. We study the worst case complexity of the algorithm for each topology. We show that for the star and tree topologies the worst case complexity of the algorithm is exponential in the size of the automaton that represents the language of the initial content of the channel in the origin of the star and the root of the tree, respectively.

The rest of the paper is organized as follows. An overview of piecewise languages and their properties is given in Sec. II, and is followed by a description of the system model in Sec. III. The algorithm for single-channel systems is presented in Sec. IV, and the one for multi-channel systems in Sec. V. We review related work in Sec. VI, and conclude in Sec. VII.

II. REGULAR AND PIECEWISE LANGUAGES

Let Σ be a finite alphabet and ε the empty string. A regular expression (RE) over Σ is defined by the following grammar R ::= ε | a | r · r | r + r | r∗. The language L(R) denoted by an RE is defined in the usual way. We sometimes just write r to mean L(r). In a further abuse of notation, we often regard a set M ⊆ Σ∪{ε} as an RE, namely the sum of elements in M.

Definition 1 (Piecewise Languages) [16], [8] A language is simply piecewise if it can be expressed by an RE of the form M0a0 · · · an−1 Mna, where each Mi ⊆ Σ and ai ∈ Σ ∪ {ε}. A piecewise language is a finite (possibly empty) union of simply piecewise languages.

For example, (a + b)c is simply piecewise, where M0 = {a, b} and α0 = c, but (ab)c is not.

Definition 2 (FSA) A finite state automaton (FSA) A is a tuple (Σ, Q, q0, δ, F), where Σ is a finite alphabet; Q is a finite set of states; q0 ∈ Q is the initial state; δ : Q × Σ → 2Q is the transition relation; and F ⊆ Q is a set of accepting (or final) states. When F is omitted, it is assumed that F = Q.

For a ∈ Σ we write δ(q, a, q′) to mean that q′ ∈ δ(q, a).

Given q ∈ Q, and w ∈ Σ∗, δ(q, w) is defined as usual: δ(q, ε) = {q}, and δ(q, wa) = {p | ∃r ∈ δ(q, w), p ∈ δ(r, a)}. We say that a word w is accepted by A iff δ(q0, w) ∩ F = ∅. The language of A is defined as L(A) = {w ∈ Σ∗ | δ(q0, w) ∩ F = ∅}. We define the size of an FSA A as usual: |A| = |Q| + |δ|.

We often use RE notation with automata. For example, A1 · A2 stands for concatenation of two automata, A1 + A2 for an automaton with language L(A1) ∪ L(A2), and (A1 matches a · W) is true iff L(A1) = L(a · W).

Definition 3 (PO-FSA) A partially ordered automaton (PO-FSA) is a tuple (A, ≤), where A = (Σ, Q, q0, δ, F) is an automaton, and is a partial order on states such that q′ ∈ δ(q, a) implies that q ≤ q′.

Proposition 1 [16] A language is piecewise iff it is recognized by a PO-FSA.

Proposition 2 (cf. [16], [8]) Piecewise languages are closed under finite unions (+), finite intersections (′), concatenation (·), shuffle (||), projections (denoted by letter-to-letter mappings), and inverse homomorphisms, but not under complementation and substitutions.

III. SYSTEM MODEL

In this section, we review the definition of FIFO systems and the reachability problem for them.

Action Languages and Semantics. A channel over an alphabet Σ is a FIFO queue whose contents is given by a word w ∈ Σ∗. We define two types of channel actions: read a, denoted by ?a, and write a, denoted by !a, that stand for reading and writing a letter a from/to a channel, respectively.

We use f : w to denote the application of an action f to a word w. For example, ?a : bbb and !a : bbb.

Proposition 3 [16] If w = w1w2 and u = u1u2, then (w1w2)!(u1u2) = w1!(w2u1)u2 = w1!(w2u1)u2.

Let Σr = {?, !} × Σ denote read/write-rw-alphabet over Σ. For a set of channels C = {c1, . . . , ck} this alphabet is extended as follows: Σr(C) = 1×k × Σr. Thus, an action 4?acoresponds to reading a from channel c4, and 6bb corresponds to writing b to channel c6. In the sequel, we drop C from the notation when it is irrelevant or clear from the context. We call Σr an action alphabet, and any subset of Σr a action language.

A channel configuration for a system with k channels is a k-tuple w ∈ (Σ∗)k. We use (w1, . . . , wk) to denote a tuple, where wi is the content of channel i. In single-channel systems, a configuration is just a content of the single channel. We use bold fonts to differentiate between channel configurations in multi-channel and single-channel systems. Let w[i] denote the content of channel i in w and w[i → y] denote a channel configuration obtained from w by replacing the content of channel i with y.

In the single-channel case, for X ⊆ Σr and W ⊆ Σ∗, we use X : W to denote the result of applying a sequence of actions from X to a word in W. This is called the concrete semantics of actions and is defined as follows:

Definition 4 (Concrete Semantics) Let W ⊆ Σ∗ be a set of strings over Σ, and X an action language, then X : W is defined as follows:

?a : W ⊆ {u | a · u ∈ W} !a : W ⊆ {w · a | w ∈ W}

{x : y} : W ⊆ {x : W} X : W = \bigcup_{x ∈ X}(x : W)

For example, (?a!b , ?a!c) : a = {b , c}.

Def. 4 is extended to a k-channel system as follows. Given w ∈ (Σ∗)k and an action language X, then X : w for a single
action is defined as shown below,

\[ i \land a : w \triangleq w_i \rightarrow (i \land a : w_i) \]

\[ i \lor a : w \triangleq w_i \rightarrow (i \lor a : w_i) \]

and is extended to words identically to Def. 4.

We write \( ?a \rightarrow !b \) for a regular action that means “\( b \) is written only if \( a \) is read first”. In other words, \( ?a \rightarrow !b \) is an abbreviation for a sequence of simple actions \( ?a!b \).

Given an action alphabet \( \Sigma_r(C) \) over a set of channels \( C \), we define a conditional action alphabet \( \Sigma_{rwuc}(C) \) that treats conditional actions as letters:

\[ \Sigma_{rwuc}(C) \triangleq \Sigma_{rw}(C) \cup ((C \times \{?\} \times \Sigma) \cup (C \times \{!\} \times \Sigma)) \]

For a set of actions \( \text{Act} \subseteq \Sigma_{rwuc}(C) \), a communication graph of \( \text{Act} \), \( CG(\text{Act}) \), is a digraph \((C,E)\), with an edge \((i,j) \in E\) if there are \( a \) and \( b \) in \( \Sigma \) such that \( ?a \rightarrow j!b \) is in \( \text{Act} \).

**Definition 5 (FIFO System)** A FIFO system is a tuple \( S = (\Sigma, C, Q, q_0, \delta) \), where \( \Sigma \) is a finite alphabet; \( C = \{c_1, \ldots, c_k\} \) is a finite set of channels; \( Q \) is a finite set of control locations; \( q_0 \in Q \) is the initial control location; and \( \delta \subseteq Q \times \Sigma_{rwuc} \times Q \) is a finite set of transition rules.

Note that in Def. 5, a FIFO system is defined with respect to a conditional action alphabet \( \Sigma_{rwuc} \). A global state of \( S \) is a pair \((q,w)\) where \( q \) is a state in \( Q \) and \( w \) is a channel configuration.

The transition relation \( \delta \) of \( S \) is a set of triples of the form \((q,w),op,(q',w')\), where \( op \in \Sigma_{rwuc} \), \( q,op,q' \in \delta \), and \( w' \in (op:w) \).

A FIFO system \( S \) is piecewise if there exists a partial order \( \preceq \) on \( Q \) such that \( q' \in \delta(q,op) \) implies that \( q \preceq q' \).

**FIFO Reachability Problem.** We are interested in the reachability problem: given a FIFO system \( S \) and a set of initial configurations \( I \), what is the set of all reachable global states?

This problem can be reduced to computing the semantics (Def. 4) of a regular action language. That is, let \( S = (\Sigma, C, Q, q_0, \delta) \), \( q \in Q \) some control location, and \( I \) a set of initial configurations. Define a finite automaton \( A_q = (\Sigma_{rwuc}, Q, q_0, \delta, \{\{q\}\}) \) where \( q \) is the only accepting state. Then, the set of all reachable configurations of \( S \) at control location \( q \) is \((L(A_q) : I)\).

Finally, computing the semantics of a regular action language is itself reducible to the limit language problem: given a regular language of actions \( L_a \) and a regular language of channel content \( L_c \), compute the language of \((L_a^* : L_c)\).

In the particular case of piecewise FIFO systems, \( L_a \) is further restricted to subsets of \( \Sigma_{rwuc} \). This is the problem we study in the rest of the paper.

**IV. ANALYSIS OF SINGLE-CHANNEL PIECEWISE SYSTEMS**

In this section, we focus on the analysis of a single-channel piecewise FIFO system. We present an algorithm for calculating the limit languages, show its correctness, and discuss its worst case complexity.

Fig. 1 shows the algorithm \textsc{SingleLimit} for calculating the limit language of \( \text{Act} \subseteq \Sigma_{rwuc} \) with respect to an initial single-channel configuration \( I \subseteq \Sigma^* \), i.e., \((\text{Act}^* : I)\).

The inputs to the algorithm are an automaton \( I \) representing initial configurations, and a set of actions \( \text{Act} \); the output is an automaton representing \((\text{Act}^* : I)\).

The algorithm has two phases. In the first phase, called \textsc{Full}, the algorithm iteratively computes all configurations reachable by (i) reading the current channel content completely, and (ii) writing the result of conditional and other write actions. Let \( \text{Act} \subseteq \Sigma_{rwuc} \) be partitioned into unconditional write actions \( \text{Act}_w = \{a | a \in \text{Act} \} \), and the rest \( \text{Act}_r = \text{Act} \setminus \text{Act}_w \). In each iteration, if \( \text{W} \) is the set of configurations reachable at current, the algorithm computes \( \text{W}' = \text{Full}(\text{W}, \text{Act}) \) such that

\[ \text{W}' = \{w | \exists u \in W, w \in ((\text{Act}_w \cup \{u\}) \cdot \delta) \} \]

Note that \textsc{Full} misses some reachable configurations. For example, let \( \text{Act} \triangleq \{?a \rightarrow \text{c}, !b \rightarrow \text{d}, le \} \) and \( I \triangleq ab \). Then, \textsc{Full} results in \( \text{L}(be^*ce^*) \) and misses reachable configurations in \( \text{L}(be^*ce^*) \). This is fixed in the second phase, called \textsc{Partial}. Let \( \text{W} \) be a set of reachable configurations, the result of \textsc{Partial} is a set \( \text{W}' \) such that

\[ \text{W}' = \{w | \exists u, v, z, (v \cdot u \in W) \land (u \cdot z = w) \land (z \in \text{Full}(\{v\}, \text{Act})) \} \]

These two phases are implemented using automata as described below.

**FULL Phase.** As inputs, \textsc{Full}(\text{A}, \text{Act}) takes an automaton \( A = (\Sigma, Q, \delta, q_0^1, F) \), and a set of actions \( \text{Act} \). As output, it constructs an automaton \( A' = (\Sigma, Q, \delta', q_0^1, F) \), where \( \delta' \) is defined as follows:

\[ \delta'(q, i, q') \iff \begin{cases} (i = \epsilon \land \delta(q, a, q') \land \exists(?a) \in \text{Act}) \lor (i = b \land \delta(q, a, q') \land \exists(?a \rightarrow !b) \in \text{Act}) \lor (i = c \land q = q' \land \exists(c) \in \text{Act}) \end{cases} \]

Intuitively, the first rule of \( \delta' \) corresponds to unconditional reads (replacing by \( \epsilon \) transitions), the second – to renaming the labels of the transitions according to the conditional actions, and the third – to unconditional writes.

**PARTIAL Phase.** Let \( A = (\Sigma, Q, q_0^1, \delta, F) \) be an automaton on a state in \( Q \). We construct two automata: \( A_1 = (\Sigma, Q, q_0^1, \delta, \{s\}) \) and \( A_2 = (\Sigma, Q, \{s\}, \delta, F) \). Let \( A'_1 \) be the automaton constructed by applying \textsc{Full} to \( A_1 \), i.e., \( A'_1 = \text{Full}(A, \text{Act}) \). Then, the language of \( A_2 \), \( A'_1 \) contains a word \( u \cdot z \) iff \( i \) there exists a word \( v \) such that \( v \cdot u \) is accepted by \( A \) via a run passing through the state \( s \), and \( ii \) \( z \in \text{Full}(\{v\}, \text{Act}) \). We call this operation \textsc{Prefix}(\text{A}, s, \text{Act}). It is easy to see that

\[ \text{Prefix}(A, s, \text{Act}) = \bigcup_{s \in Q} \text{Prefix}(A, s, \text{Act}) \]

The algorithm in Fig. 1 always terminates. Given an automaton \( A \), \textsc{Full} produces an automaton with the same number of states as \( A \). Thus, the set \( \{\text{Full}^i(A, \text{Act})\}_i \) is finite, and the algorithm always reaches a fixpoint.

**Theorem 1** Let \( A_1 \) be an automaton representing a set of configurations, \( \text{Act} \) be a set of actions, and \( A_L \) be the automaton
Let \( u = \{ a \ast b, (c + d) \ast e \} \) represent a piecewise configuration where \( u[1] \) is an automaton representing \( a \ast b \), and \( u[2] \) is an automaton representing \( (c + d) \ast e \). In pseudo-code, we use \text{Conf} \) for the type of piecewise configurations, and notation \( X \text{ with } [i] = y \) to mean \( X[i \mapsto y] \).

A. Star Topology

A set of actions \( \text{Act} \) has a star topology iff there exists a unique channel \( o \), the origin, s.t. for every action \( ?a \rightarrow j!b \) in \( \text{Act}, i = o \) and \( j \neq o \), i.e., \( CG(\text{Act}) \) is a star (see Fig. 2(a)). In the sequel, we assume that channel 1 is the origin channel.

Let \( u \) be a piecewise channel configuration. The algorithm \( \text{DOREAD} \), shown in Fig. 4, computes the limit \( (\text{Act}^* : \mathcal{L}(u)) \).

Complexity Analysis. For a piecewise configuration \( u \), the depth of the recursion of \( \text{DOREAD} \) is bounded by \( h = |u[o]| \) for the origin \( o \). Inside each call, \( \text{SATURATE} \) takes constant time and returns a single configuration; however, \( \text{STEP} \) may
1: Conf SATURATE (Conf u, Channel ch, int idx)  
2: let (Q, δ, F) = u[ch], M = {a | (qδ, a, qδ′) ∈ δ}  
3: forall i ∈ [(1...k) \ {ch}] do  
4: let M'i = {b | (ch′a → jlb) ∈ Act ∧ a ∈ M ∧  
5:   (∀j < ch, idx(j, b) = idx(j, o)) ∧ idx(ch, b) = idx}  
6: return u'  
7: Set (Conf) STEP (Conf u, Channel ch, int idx)  
8: u' := ∅  
9: let (Q, δ, F) = u[ch], M = {a ∈ Σ | ∃q′, (qδ, a, qδ′) ∈ δ ∧ q′ ≠ q′}  
10: forall a ∈ M ∧ i ∈ (j | 3b, (ch′a → jlb) ∈ Act ) do  
11:   ∀(∀j < ch, idx(j, b) = idx(j, o)) ∧ idx(ch, b) = idx}  
12: u'[ch] := u[ch] ∪ {M'i}  
13: u'[ch] := u'[ch] ∪ {M'i}  
14: ∀u' := ∀u' ∪ {u'}  
15: return u'  

Fig. 3. STEP and SATURATE algorithms.

1: List TREELIMIT ()  
2: for ch = 0 to k do  
3: writeWL := ∅  
4: forall u ∈ readWL do doRead (u, ch)  
5: readWL := writeWL  
6: return readWL

Fig. 5. The TREELIMIT algorithm.

1: Conf MERGES (Conf u, Channel ch)  
2: Conf doWrite (Conf conf, Channel ch)  
3: return MERGES (u, ch)

4: List MULTI LIMIT ()  
5: for ch = 0 to k do  
6: writeWL := ∅  
7: forall u ∈ readWL do doRead (u, ch)  
8: if ch + 1 < k then  
9:   forall u ∈ writeWL do  
10: readWL := readWL ∪ {doWrite (u, ch + 1)}  
11: return readWL

Fig. 6. MULTI LIMIT algorithm and its supporting routines.

The graph CG (Act) is acyclic and, therefore, induces a partial order ≤ on channels (vertices of the graph). For channels i and j, i ≤ j if there exists a path from i to j in CG (Act).

Theorem 4 Let u be a piecewise channel configuration, Act a set of actions with star topology on k channels with origin o, and h = [u][o]. Then, in the worst case, the running time of DoREAD (u, o) is O (max (k^3, h)).

B. Tree Topology

A set of actions Act has a tree (or, more generally, a forest) topology iff for all actions i?a → ?j b and i′?a′ → ?j′ b′ in Act,  
5. j = j' ⇒ i = i'. That is, CG (Act) is a tree (e.g., Fig. 2(b)).

The DoREAD algorithm for the star topology is not applicable to the tree topology since it assumes that all reads come from a single channel. However, an action set with the tree topology can be partitioned such that each partition has a star topology. Formally, for a set of actions Act, let Act_i denote all the actions that read from channel i. Then,  
5. {Act_i} partitions Act and each Act_i has a star topology with

Theorem 5 Let Act be an action set on k channels s.t. CG(Act) is a tree, and w a channel configuration. Then,  
5. (Act^* : w) ≠ ∅ ⇒ ((Act^* : w) = (Act^*1 : ... : Act^*k) : w))

Theorem 5 leads to an obvious algorithm for computing the limit language in the tree topology: (i) establish a total order on
channels based on the CG, and (ii) use this order to iteratively apply DOREAD to each partition Act. We call this algorithm TREELIMIT (see Fig. 5). Since TREELIMIT proceeds through a finite total order of channels it always terminates.

**Theorem 6** Let \( u \) be a piecewise configuration, Act an action set with tree topology, and \( U \) the set of configurations returned by TREELIMIT. Then, \( \mathcal{L}(U) = (\text{Act}^+ : \mathcal{L}(u)) \).

**Complexity Analysis.** W.l.o.g., we assume that \( CG(\text{Act}) \) is an \( N \)-ary tree with \( M \) internal nodes and that the initial content of all the channels except the root is empty. Let \( u \) be a piecewise configuration, and \( h = |u| \). By Theorem 4, computation of \( \text{Act}^+ : \mathcal{L}(u) \) produces at most \( \max(N^h,h) \) piecewise configurations, each of size at most \( h \). TREELIMIT applies computation \( \text{Act}^+ : \mathcal{L}(u) \), \( M \) times, which produces at most \( \max(N^{h \times M},h^M) \) configurations.

**Theorem 7** Let \( u \) be a piecewise configuration, Act a set of actions with a tree topology of degree \( N \) and \( M \) internal nodes, and \( h = |u| \). In the worst case, the running time of TREELIMIT is \( O(\max(N^{h \times M},h^M)) \).

### C. Inverted Tree Topology

A set of actions Act has an inverted tree topology iff for all conditional actions \( i?a \rightarrow j!b \) and \( i'?a' \rightarrow j'!b' \) in Act, \( i = i' \Rightarrow j = j' \). That is, \( CG(\text{Act}) \) is an inverted tree (e.g., see Fig. 2(c)).

In the inverted tree topology, a channel may depend on several pairwise independent channels. Therefore, Theorem 5 is no longer applicable. For example, let \( \text{Act} \equiv \{1?a \rightarrow 3!a, 2?b \rightarrow 3!b\} \), and \( \text{w} \equiv \langle aa, bb, c \rangle \) be a configuration. The partial order on the channels induced by \( CG(\text{Act}) \) is \( \{1 \preceq 3, 2 \preceq 3\} \), with two obvious linearizations. A configuration \( \langle e, e, abab \rangle \) is reachable from \( \text{w} \), but does not belong to neither \( \{1?a \rightarrow 3!a\} \) nor \( \{2?b \rightarrow 3!b\} \) : \( \text{w} \), nor \( \{1?a \rightarrow 3!a\} \) : \( \text{w} \), which contradicts the theorem.

For simplicity of presentation, we assume that there is a unique channel, referred to as \( l \), that has multiple dependencies, like channel 3 in the above example. That means \( l \) is the only channel whose node in \( CG(\text{Act}) \) has an in-degree greater than or equal to 2. In this case, it is possible to (i) replace channel \( l \) with new channels, called shadows of \( l \), and turn \( \text{Act} \) into a tree topology, (ii) solve the new limit problem using TREELIMIT, and (iii) combine the contents of shadow channels together. This is further explained below.

We define a function ADDS that introduces shadow channels for \( l \) by redirecting each conditional that reads from \( i \) and writes to \( l \) to write to a newly created shadow channel \( l_i \). Formally,

\[
\text{ADDS}(i?a \rightarrow j!b,l) \equiv \begin{cases} 
\hat{i}?a \rightarrow \hat{l}_i!b & \text{if } j = l \\
i?a \rightarrow j!b & \text{otherwise}
\end{cases}
\]

ADDS breaks dependencies between channels. Let \( \hat{\text{Act}} = \text{ADDS}(\text{Act},l) \). If \( CG(\text{Act}) \) is an inverted tree, then \( CG(\hat{\text{Act}}) \) is a tree. We use \( \mathcal{S}(l) \) to denote the shadows of \( l \).

Let \( \text{w} \) be a configuration, and \( \text{w} \) be its extension to shadow channels. That is, \( \text{w}[i] = \text{w}[i] \) if \( i \not\in \mathcal{S}(l) \), and \( \text{w}[i] = \epsilon \) otherwise. The sets \( \{\text{Act}^* : \text{w}\} \) and \( \{\hat{\text{Act}}^* : \text{w}\} \) are closely related. Let \( t \in \{x : \text{w}\} \) be a configuration reachable from \( \text{w} \) by a sequence \( x \in \text{Act}^* \), and \( \hat{t} \in \{\text{ADDS}(x,l) : \text{w}\} \) be a configuration reachable from \( \text{w} \), where ADDS is extended to sequences in an obvious way. ADDS only augments actions that write to \( l \). Thus, \( t[i] = \hat{t}[i] \) for any \( i \) that is different from \( l \) or its shadow channels \( \mathcal{S}(l) \). By adding shadow channels for \( l \), all the writes on \( l \) are redirected to its shadows and \( \hat{t}[l] \) is the initial content of \( l \), hence, it is a prefix of \( t[l] \). Each shadow channel \( \hat{l}_i \) keeps track of what was read from channel \( i \) and written to \( l \), hence, \( \hat{t}[\hat{l}_i] \) is a subsequence of \( t[l] \).

In order to formalize the relation between \( \{\text{Act}^* : \text{w}\} \) and \( \{\hat{\text{Act}}^* : \hat{\text{w}}\} \), we define a function MERGES. Given a configuration over shadow channels, MERGES produces all corresponding configurations without shadows. Formally,

\[
t \in \text{MERGES}(\hat{t},l) \iff (\forall i \neq l \land i \not\in \mathcal{S}(l), \ t[i] = \hat{t}[i] \land (t[l] \in \mathcal{L}(\hat{t}[l]) \land \|t[l]\| \leq \|\hat{t}[l]\|))
\]

**Theorem 8** Let \( \text{Act}, \hat{\text{Act}}, \text{w}, \hat{\text{w}}, \) and \( l \) be as above. Then,

\[
t \in (\text{Act}^* : \text{w}) \iff \exists \hat{t} \in (\hat{\text{Act}}^* : \hat{\text{w}}), \ t \in \text{MERGES}(\hat{t},l)
\]

Both Theorem 8 and MERGES are easily lifted to piecewise configurations s.t. if \( u \) is a piecewise configuration, then \( \text{MERGES}(u,l) \) defines a piecewise configuration as well. This follows from the fact that piecewise languages are closed under concatenation and shuffle (see Prop. 2).

The explained procedure can be extended to an arbitrary inverted tree. The correctness follows by induction on the number of channels. The final algorithm MULTILIMIT is shown in Fig. 6. The algorithm assumes that shadow channels are introduced where they are needed. It traverses the channels according to the partial order induced by the CG, applying read and write phases. The read phase is the same as in the star and tree topologies (done by DOREAD). The write phase uses MERGES to merge the content of all the shadows of a channel before applying a read phase to it.

**Theorem 9** Let \( u \) be a piecewise configuration, Act an action set with inverted tree topology, and \( U \) the set of configurations returned by MULTILIMIT. Then, \( \mathcal{L}(U) = (\text{Act}^+ : \mathcal{L}(u)) \).

### D. DAG Topology

In this section, we present an algorithm for computing the set of reachable configurations for a set of actions whose CG is an arbitrary directed acyclic graph (DAG) (e.g. see Fig. 2(d-e)). This subsumes the algorithms from the previous sections for star, tree, and inverted tree topologies.

What makes the DAG topology different from the inverted tree is that immediate predecessors (in the \( \preceq \) partial order on the CG) of a channel may be interdependent. For example, consider \( \text{Act} \equiv \{1?a \rightarrow 3!a, 1?b \rightarrow 2!b, 2?b \rightarrow 3!b\} \) whose CG is shown in Fig. 2(d). Channel 3 has channels 1 and 2 as its immediate predecessors, and channel 2 depends on channel 1.
This extra layer of dependence precludes the possibility of breaking the topology by simply introducing shadow channels.

For our running example, consider the computation of reachable configurations starting from \( \langle a^*b^*, \epsilon, \epsilon \rangle \). We can replace channel 3 with two shadow channels to obtain \( Act = \{1? \rightarrow 3_1a, 1? \rightarrow 2b, 2b \rightarrow 3_2b \} \). By applying TREELIMIT to the resulting tree topology, we obtain two piecewise configurations \( \{ (a^*b^*, \epsilon, a^*), (b^*, b^*, a^*, b^*) \} \). If we then proceed by merging the contents of the shadows of channel 3, as in the inverted tree topology, we obtain \( \{ (a^*b^*, \epsilon, a^*), (b^*, b^*, (a+b)^*) \} \). The second piecewise configuration includes configurations in which the content of channel 3 is in \( b^+a^+ \). These configurations are infeasible since \( a \) came before \( b \) in channel 1 in any initial configuration and this order must be preserved when the content is copied to channel 3.

To solve this problem, we extend MULTI LIMIT algorithm by modifying the shuffle used by MERGES (see Sec. V-C) to respect the dependencies between the predecessors of the channel whose shadows are merged. This requires (i) keeping track of the relative positions of each letter in a channel as it is copied between channels, and (ii) restricting the shuffle based on the history of positions of each letter.

For a system with \( k \) channels, each letter is associated with a \( k \)-tuple of indices from IDX\(^k\), where IDX \( \triangleq [-1, \infty) \). Intuitively, \( j \)th index of a letter \( a \) indicates the relative position of \( a \) when it was in channel \( j \), with \( -1 \) meaning that \( a \) was never in that channel. Let idx\((i, a)\) be a function that extracts the \( i \)th index of \( a \). For example, idx\((2, a)\) = 4 means that \( a \) was at some point at position 4 in channel 2, and idx\((3, a)\) = -1 means that \( a \) was never in channel 3. We use ch\((a)\) to denote the latest channel that \( a \) was in. Formally, ch\((a) \triangleq \max \{ i \mid \text{idx}(i, a) \neq -1 \} \).

To keep track of the indices, several parts of the MULTI LIMIT are modified. DoREAD (see Fig. 4) is extended to accept as an argument the ch-index of a letter at the head of the current channel ch, and increment it at each recursive call (lines 15 and 24). SATURATE and STEP (see Fig. 3) are extended to propagate and assign indices as well (lines 4 and 11).

The interdependence of the channels implies the following constraint on the content of every channel in every reachable configuration. Let \( w \) be a word describing a content of channel \( l \). Let \( a \) and \( b \) be letters at positions \( p \) and \( q \) in \( w \), respectively. Assume that \( i \) is the last channel \( a \) was in, and that \( i \) precedes the last channel that \( b \) was in, i.e., \( i = \text{ch}(a) < \text{ch}(b) \). Furthermore, assume that \( a \) preceded \( b \) in channel \( i \), i.e., idx\((i, a)\) < idx\((i, b)\). Then \( a \) has to precede \( b \) in \( w \), i.e., \( p < q \), since \( a \) had to be read from channel \( i \) (and placed in \( w \)) before \( b \) could be read.

We denote the set of all words that satisfy the above condition by WO. Formally, it is the set of all words \( w \) in \( (\Sigma \times \text{IDX}\^k)^* \) that satisfy

\[
\forall p, q, \ (a = w(p) \land b = w(q) \land i = \text{ch}(a) \land \text{ch}(a) < \text{ch}(b) \land \text{idx}(i, a) < \text{idx}(i, b)) \Rightarrow p < q
\]

where \( w(p) \) denotes the letter at position \( p \) of \( w \). For our running example, the word \( ba \) in channel 3 does not belong to WO: the last channel of \( a \) is 1 which precedes 2 – the last channel of \( b \), and \( a \) preceded \( b \) in channel 1, thus, it must also precede \( b \) in channel 3.

The set WO defines a piecewise language, and is recognizable by a PO-FSA.

**Theorem 10** The language WO is piecewise.

In order to restrict MERGES to only include words that satisfy WO, we replace it with a function MERGEDAGS defined as follows. Let \( l \) be a configuration reachable from an initial configuration extended with shadow channels, and \( l \) a non-shadow channel. Then, \( \text{MERGEDAGS}(t, l) \triangleq \text{MERGES}(t, l) \cap \text{WO} \). Since WO is piecewise (by Theorem 10) and piecewise languages are closed under intersection (by Prop. 2), MERGEDAGS defines a piecewise configuration.

With this change, MULTI LIMIT algorithm computes the exact set of reachable configurations.

**Theorem 11** Let \( u \) be a piecewise configuration, \( Act \) a set of actions with DAG topology and \( U' \) a set of configurations returned by MULTI LIMIT algorithm, where MERGES is replaced by MERGEDAGS. Then \( \mathcal{L}(U') = (\mathcal{L}^* : \mathcal{L}(u)) \).

**VI. RELATED WORK**

FIFO systems play key roles in description and analysis of distributed systems. It is well-known that most non-trivial verification problems for FIFO systems are undecidable [9]. However, a substantial effort has gone into analysis of these systems. In general, two main approaches have been followed for the analysis of FIFO systems. The first approach, and the one taken in this paper, is to identify practically useful subclasses of FIFO systems with decidable properties (e.g., [18], [11], [19], [16]). The second approach is to look for efficient semi-algorithms that scale to realistic examples, but do not guarantee to always terminate (e.g., [4], [5], [12]). Although this approach may look promising, in many cases finding a good bound between scalability and termination is very challenging.

The two approaches may be combined, as illustrated in the analysis of lossy channel systems in which channels may lose messages. In these systems, the problem of reachability of a given state is decidable [2], [10], [1]: however, calculating the set of all reachable states is impossible. The systems considered is this paper are not lossy; all channels are perfect and do not lose any message.

Pachl [18] proves that if the set of reachable channel configurations (limit language) is recognizable then it is decidable to check for reachability of any given state. It was later shown in [10] that even though the reachability set might be recognizable, determining it may still be undecidable.

Boigelot et al. [4], [5], [3] describe a data structure, QDD, for representing sets of queue contents, and a QDD-based semi-algorithm to compute a set of reachable states. The termination of this algorithm depends on handling iterations of arbitrary sequences of actions. This is equivalent to limit languages in our terminology. In [4], automata-theoretic algorithms are
given to calculate \( f : L \) and \( f^* : L \) for a single read, write, or conditional action \( f \). Boigelot Ph.D. Thesis [3] and [5] extend that to action sequences that preserve recognizability of channel contents. The key difference between \([4],[5]\) and our work is that we concentrate on iteration of multiple conditional actions together. Such sequences are much harder to handle since in general they do not preserve recognizability of channel contents \([16]\).

The problems of recognizability of limit languages and decidability of their computation for piecewise FIFO systems where first studied in \([16]\). Although the paper presents decidability results, it only sketches the algorithms and does not analyze their complexity. For the single-channel case, our new algorithm is simpler. It avoids the use of recurrence arguments on set-theoretic operators on alphabets. Rather, we bound the iterative construction through a direct use of the union operation on simpler versions of the original automata. For the multi-channel case, our new algorithm mends the missing arguments of \([16]\) that deal with the merging of input on dependent channels. Our concept of indexing allows such dependencies to be added and tracked in the transducer-like treatment of the DAG topology.

An approach for model-checking piecewise FIFO systems was studied in \([13]\). That work presents a procedure for calculating an abridged model of a FIFO system, which when successful, constructs such a model by computing an abstraction of the reachable channel contents. It is shown in \([13]\) that abridged models preserve path properties expressed by a restricted class of Büchi automata. In contrast, the work presented in this paper focuses on calculating the exact limit languages and applies to reachability/safety properties only.

VII. CONCLUSION

FIFO systems are a common model of computation for distributed protocols. We have studied the reachability problem for a class of FIFO systems composed of piecewise components. Since, it is well-known that this problem is reducible to computing the limit language of a regular language of actions, we concentrate exclusively on the limit language problem.

We consider single-channel and multi-channel FIFO systems separately. For the single-channel case, we present a new automata-theoretic algorithm for calculating the limit language starting with an arbitrary regular initial content. We show that the worst case complexity of our algorithm is exponential in the size of the automaton representing the initial channel content. A prototype of the algorithm was implemented using the Automaton package \([17]\).

For multi-channel systems, we present an automata-theoretic algorithm for computing the limit language subject to the following conditions: (i) the initial language is piecewise, and (ii) the communication graph of actions is acyclic. For the star and the tree topology, we show that the complexity of our algorithm is exponential in the size of the automaton representing the initial channel configuration. In the cases of inverted tree and DAG topologies the complexity of the algorithms remains an open problem.

REFERENCES


Theorem 1 Let $A_I$ be an automaton representing a set of configurations, $Act$ be a set of actions, and $A_L$ be the automaton returned by $\text{SINGLELIMIT}(A_I, Act)$. Then, $\mathcal{L}(A_L) = (Act^* : \mathcal{L}(A_I))$.

Proof: The theorem follows as a consequence of Theorem 12 shown below.

Theorem 12 Let $Act \subseteq \Sigma_{rwc}$ be a set of actions, and $\mathcal{L}_I \subseteq \Sigma^*$ a language over $\Sigma$. Then,

$$\text{PARTIAL} \left( \bigcup_{i \in \mathbb{N}} \text{FULL}^i(\mathcal{L}_I, Act) \right) = ||Act^*||(\mathcal{L}_I)$$

Proof:

Let $w \in ||Act^*||(\mathcal{L}_I)$ be a reachable channel content, then $w$ is reached by reading some initial content fully 0 or more times, and reading the resulting content partially. Formally, let $\# - a$ fresh letter not already occurring in $\Sigma$, be a marker of the end of channel’s content. Then,

$$w \in ||Act^*||(\mathcal{L}_I) \iff \exists u, v, u \cdot v = w \land \exists p, q, u\#v \in ||((Act)^(!\#)(\#))^p(Act)^q||(\mathcal{L}_I \cdot \#)$$

Theorem 10 The language $WO$ is piecewise.

Proof: To prove the theorem, we show a PO-FSA $A_{WO}$ that recognizes $WO$. Let $N$ be the number of channels. The state space of $A_{WO}$ is $\text{IDX}[1..N][1..N]$. An interpretation of a state is

$q[i][j] = p$

if the automaton saw a letter whose last channel was $i$, and that was at some point at position $p$ in channel $j$. The transition relation $\delta$ is deterministic, and is defined as follows:

$$\delta(q, a, q') \iff \langle \forall i \leq N, \ q[i][ch(a)] \leq \text{idx}(ch(a), a) \rangle \land \langle \forall i, j, i \neq ch(a) \Rightarrow q'[i][j] = q[i][j] \rangle \land \langle i = ch(a) \Rightarrow q'[i][j] = \max(q[i][j], \text{idx}(j, a)) \rangle$$

The first conjunct of $\delta$ ensures that the $WO$ condition is satisfied, and the second updates the state. The initial state $q^0$ is such that $\forall i, j, q^0[i][j] = -1$, and every state is accepting. The automaton $A_{WO}$ is a PO-FSA, where the partial order $\preceq$ on states is:

$q \preceq q' \iff \forall i, j, q[i][j] \leq q'[i][j]$