Reachability Problems in Piecewise FIFO Systems

NAGHMEH GHAFARI
Critical Systems Labs
and
ARIE GURFINKEL
Software Engineering Institute, Carnegie Mellon University
and
NILS KLARLUND
Google Inc.
and
RICHARD TREFLER
David R. Cheriton School of Computer Science, University of Waterloo

Systems consisting of several finite 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.

In this article, we study the problem of computing the set of reachable states of a FIFO system composed of *piecewise* components. This problem is closely related to calculating the set of all possible channel contents, i.e. the *limit language*, for each control location. We present an algorithm for calculating the limit language of a system with a single communication channel. For multi-channel systems, we show that the limit language is piecewise if the initial language is piecewise. Our construction is not effective in general; however, we provide algorithms for calculating the limit language of a restricted class of multi-channel systems 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.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Model checking

General Terms: Verification
Additional Key Words and Phrases: FIFO systems, reachability analysis, verification, infinite-state systems

1. 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., [Brand and Zafiropulo 1983; Boigelot et al. 1997; Abdulla et al. 1999; Pachl 1987; Cece et al. 1996; Bond et al. 2001; Wodey et al. 2003]). Even though all physically constructible systems have finite size channels, their size is often an implementation parameter that is typically left unspecified. Modeling such systems with unbounded channels often makes reasoning about them simpler. The abstraction may of course fail to reveal certain deadlock situations that occur if the chan-

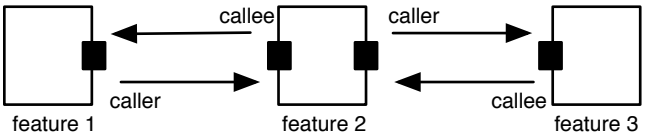

Fig. 1. BoxOS call structure.

nels fill up, but the abstract system behaves otherwise essentially as the system with finite channels.

Unboundedness of communication channels provides a useful modeling abstraction, but it does in a theoretical sense complicate analysis if compared to a system of a given fixed size, say with queues of length 1024. In fact, Brand and Zafiropulo [Brand and Zafiropulo 1983] 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., [Abdulla et al. 1999; Abdulla and Jonsson 1993; Boigelot 1998; Boigelot and Godefroid 1999; Boigelot et al. 1997; Bouajjani et al. 2000; Bouajjani et al. 2001; Cece et al. 1996; Pachl 1987]).

In this article, 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. 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 automaton 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 $M_0^* a_0 M_1^* \cdots a_{n-1} M_n^*$; here, each M_i is a subset of the alphabet Σ and each a_i is an element of Σ.

1.1 Motivating Example

Although piecewise languages may look restrictive, they can be used to express descriptions of IP-telephony features [Ghafari and Trefler 2006] and seem amenable to describing composite web services specified in Business Process Execution Language (BPEL) [IBM 2007]. For example, [Ghafari and Trefler 2006] studied the behavior of the telephony features in BoxOS which is a generation of telecommunication service over IP developed at AT&T Research [Bond et al. 2001; Jackson and Zave 1998]. As shown in Fig. 1, an active call is represented by a graph of telephony features (referred to as *boxes*) while communication between neighboring boxes is handled via unbounded perfect FIFO channels. Boxes at the end points represent telephones, intermediate boxes represent call features, for example call-forwarding-on-busy. At a sufficient level of abstraction, boxes may all be viewed as finite state transducers. Communication in these protocols begins with an initiator trying to reach a given destination. A call is built recursively. The current endpoint begins the call initiation protocol with a chosen neighbor, the *callee*. If this initiation results in a stable connection, the *callee* becomes the new endpoint and the call construction continues. Call termination is required to proceed in a reverse order and in general required to begin at a call endpoint.

In order to manage inter-feature communication, it is desired that communication between features have certain pattern [Bond et al. 2001]. Thus, all of the feature boxes

implement a communication template that consists of three phases (cf. [Bond et al. 2001]): *setup* phase, *transparent* phase, and *teardown* phase. Fig. 2 describes a *transparent box* that represents such a communication template. The transparent box communicates with two neighbors across four separate channels. Messages to/from the upstream (initiating), *caller*, are sent/received via *ro/ri* channels. Messages to/from the downstream (receiving), *callee*, are sent/received via *eo/ei* channels. A message is received with the '?' symbol and sent with the '!' symbol. For example *ri?setup* indicates a *call setup message* received from the *ri* channel. Interestingly, this communication template can be expressed by piecewise languages. To achieve piecewiseness, we have abstracted the transparent box by replacing the original LINKED state and its left and right neighbors, shown in shaded rectangle on the top right corner of Fig. 2, by the LINKED state, shown in the shaded rectangle in the middle of the figure. Both of these states have the same functionality. The difference is the addition of *conditional actions* of the form $ri?status \rightarrow eo!status$, where the *status* message is sent to the callee only if the *status* message has been received from the caller first.

It is crucial to be able to reason about safety and deadlock properties of BoxOS implementations with multiple features, somethings that the techniques in [Bond et al. 2001] fell short to address.

1.2 Our Contributions

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 [Cece et al. 1996], and even when the limit language is known to be regular, determining it may still be undecidable [Cece et al. 1996].

In this article, we show that piecewise languages play an important role in the analysis of FIFO channel systems. In particular, we focus on computing the limit languages in piecewise FIFO systems. Our main contributions are summarized as below:

—For single-channel piecewise FIFO systems, we show in Sec. 4 that the limit language is regular (piecewise) if the initial channel language is regular (piecewise). We provide an algorithm to compute the limit language and discuss its complexity.

—For multi-channel piecewise FIFO systems, we show in Sec. 5 that the limit language is regular, in fact piecewise, if the initial channel language is piecewise. However, the construction of the limit language may not always be effective. In Sec. 6, we show for systems with *acyclic communication graph* the limit language is piecewise if the initial channel language is piecewise. We present an algorithm to calculate the limit language and discuss its complexity.

The rest of the article is organized as follows. An overview of piecewise languages and their properties is given in Sec. 2, and is followed by a description of the system model in Sec. 3. Our main contributions are presented in Sec. 4, Sec. 5, and Sec. 6. We review related work in Sec. 7, and conclude in Sec. 8.

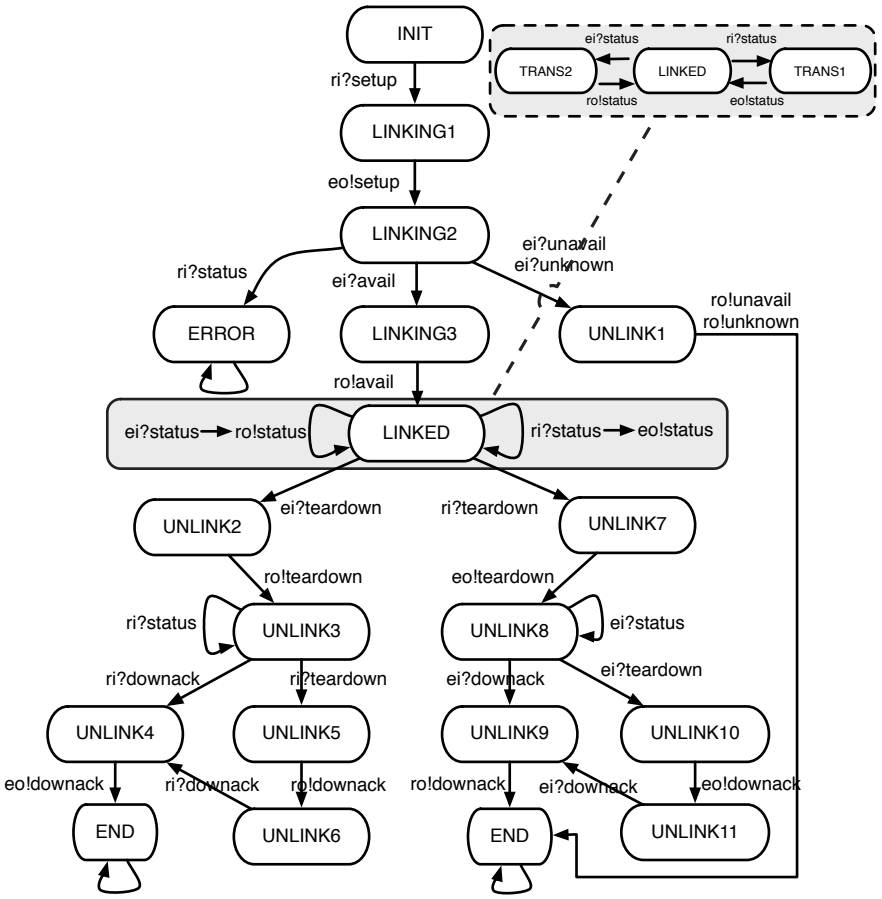

Fig. 2. Transparent feature box.

2. PRELIMINARIES AND NOTATIONS

In this section, we introduce some preliminary notation and give an overview of piecewise languages and their properties.

Let Σ be a finite alphabet and ϵ the empty string. Let w_1 and w_2 be two strings in Σ^*. In the sequel, $w_1 + w_2$ denotes the non-deterministic choice between w_1 and w_2 and $w_1 \cdot w_2$ denotes concatenation of the elements of w_1 and w_2. We sometimes omit '$\cdot$', i.e. we may write $w_1 w_2$ instead of $w_1 \cdot w_2$. A regular expression (RE) over Σ is defined by the following grammar $R ::= a \in \Sigma \mid R \cdot R \mid R + R \mid R^* \mid \mathbf{0} \mid \mathbf{1}$. The symbol $\mathbf{0}$ denotes the empty language, and $\mathbf{1}$ denotes the language $\{\epsilon\}$; in particular, we have $\mathbf{1} = \mathbf{0}^*$. We sometimes write ϵ instead of $\mathbf{1}$.

The language $\mathcal{L}(R)$ of a RE R is defined in the usual way. We sometimes write R to mean $\mathcal{L}(R)$. In a further abuse of notation, we often regard a set $M \subseteq \Sigma \cup \{\epsilon\}$ as an RE, namely the sum of elements in M. For a language $L \subseteq \Sigma^*$, we use $\complement L$ to denote the

complement of L: $\Sigma^* \backslash L$. The expression $test(R)$ is $\mathbf{1}$ if $\mathcal{L}(R) \neq \emptyset$ and $\mathbf{0}$ if $\mathcal{L}(R) = \emptyset$. We now introduce a new fragment of regular languages called *piecewise languages*.

Definition* 2.1 (*Piecewise Languages) A language is *simply piecewise* if it can be expressed by an RE of the form $M_0^* a_0 M_1^* \cdots a_{n-1} M_n^*$, where each $M_i \subseteq \Sigma$ and $a_i \in \Sigma \cup \{\epsilon\}$. A *piecewise* language is a finite (possibly empty) union of simply piecewise languages. A language is *simply repetition piecewise* if it can be expressed by an RE of the form $M_0^* a_0 M_1^* \cdots a_{n-1} M_n^*$, where for all i, a_i is ϵ. A *repetition piecewise* language is a finite (possibly empty) union of simply repetition piecewise languages.

For example, $(a+b)^* c$ is simply piecewise, where $M_0 = \{a, b\}$ and $a_0 = c$, but $(ab)^*$ is not piecewise according to a simple application of the pumping lemma. For completion, we give the definition of a finite state automaton.

Definition* 2.2 (*FSA) A *finite state automaton (FSA)* A is a tuple $(\Sigma, Q, q^0, \delta, F)$, where Σ is a finite alphabet; Q is a finite set of states; $q^0 \in Q$ is the initial state; $\delta : Q \times \Sigma \to \mathbf{2}^Q$ is the transition relation; and $F \subseteq Q$ is a set of accepting (or final) states. When F is omitted, it is assumed that $F = Q$.

For $a \in \Sigma$ we write $\delta(q, a, q')$ or $q \xrightarrow{a} q'$ to mean that $q' \in \delta(q, a)$. We write $q \to q'$ when we do not distinguish the specific symbol on the transition of q to q'. Given $q \in Q$, and $w \in \Sigma^*$, $\delta(q, w)$ is defined as usual: $\delta(q, \epsilon) \triangleq \{q\}$, and $\delta(q, wa) \triangleq \{p \mid \exists r \in \delta(q, w),\ p \in \delta(r, a)\}$. We say that a word w is accepted by A if and only if $(\delta(q^0, w) \cap F) \neq \emptyset$. The language of A is defined as $\mathcal{L}(A) \triangleq \{w \in \Sigma^* \mid \delta(q^0, w) \cap F \neq \emptyset\}$. A *run* in A is a finite or infinite sequence of states denoted $P = q_0 \to q_1 \to \ldots$, where q_0 is the initial state and for all i, $q_i \to q_{i+1} \in \delta$. We define the *size* of an FSA A as: $|A| \triangleq |Q| + |\delta|$.

We often use RE notation with automata. For example, $A_1 \cdot A_2$ stands for concatenation of two automata, $A_1 + A_2$ for an automaton with language $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$.

Definition* 2.3 (*PO-FSA) A *partially ordered automaton (PO-FSA)* is a tuple $(A, \preceq)$, where $A = (\Sigma, Q, q^0, \delta, F)$ is an automaton, and $\preceq \subseteq Q \times Q$ is a partial order on states such that $\forall a \in \Sigma$, $q' \in \delta(q, a)$ implies that $q \preceq q'$.

Proposition 2.4 *A language is piecewise if and only if it is recognized by a PO-FSA.*

PROOF. ($\Leftarrow$) Consider the PO-FSA $A = ((\Sigma, Q, \delta, q^0, F), \preceq)$. Consider all acyclic runs $P = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{k-2}} q_{k-1} \xrightarrow{a_{k-1}} q_k$, where any $q_i \in Q$ for $i \in [0..k]$ occurs at most once and $q_0 = q^0$ is initial and $q_k \in F$ is an accepting state. The number of such runs is finite. For each q_i we can associate M_i, the set of a's such that $\delta(q_i, a) = q_i$. Let $L_P = M_0^* a_0 M_1^* ... M_k^*$. Then, $L_P \subseteq \mathcal{L}(A)$. Let $L' = \bigcup_P L_P$, where the union is over all appropriate runs P as just considered. We clearly have that $L' \subseteq \mathcal{L}(A)$. To see that $\mathcal{L}(A) \subseteq L'$, we use that automaton A is partially ordered. Consider $w \in \mathcal{L}(A)$. Thus, w defines a run P from which an acyclic run P' for a word w' can be constructed by deleting letters a_i in w for which $\delta(q_i, a_i) = q_i$. Then, w' is a scattered subword of $w : w = u_0 b_0 \ldots b_n u_{n+1}$, where $w' = b_0 \ldots b_n$ and u_i, for $0 \leq i \leq n+1$, is in M_i^*. Hence, $w \in L_{P'}$. As a result, $\mathcal{L}(A) \subseteq L'$.

($\Rightarrow$) A piecewise language is a finite union of simply piecewise languages. Each simply piecewise language is recognized by a totally ordered automaton. Consider the simply piecewise language $M_0^* a_0 \cdots M_{k-1}^* a_{k-1} M_k^*$. This language is recognized by an automaton $A = (\Sigma, Q, \delta, q^0, F)$, where $Q = \{q_0, q_1, \ldots, q_k\}$, $q^0 = q_0$, and $F = \{q_k\}$. The

transition relation δ is defined as follows:

$$(q, a, q') \in \delta \Leftrightarrow (\text{for } i \in [1..k], q = q_{i-1} \wedge q' = q_i \wedge a = a_{i-1}) \vee \\ (\text{for } i \in [0..k], q = q' = q_i \wedge a \in M_i)$$

By construction, for $i, j \in [0..k]$, the relation $q_i \preceq q_j \Leftrightarrow i \leq j$ is a total order satisfying the constraint of Definition 2.3. Let $L = L_1 + L_2$ be a piecewise language, where L_1 and L_2 are simply piecewise languages recognized by PO-FSAs $A_1 = ((\Sigma, Q_1, \delta_1, q_1^0, F_1), \preceq_1)$ and $A_2 = ((\Sigma, Q_2, \delta_2, q_2^0, F_2), \preceq_2)$, respectively, with Q_1 and Q_2 disjoint. Then, L is recognized by a PO-FSA $A = ((\Sigma, Q, \delta, q^0, F), \preceq)$, where $Q = Q_1 \cup Q_2 \cup q^0$, and q^0 is a new state not appearing in $Q_1 \cup Q_2$, $F = F_1 \cup F_2$, and δ is defined as follows:

$$(q, a, q') \in \delta \Leftrightarrow (q, q' \in Q_1 \wedge (q, a, q') \in \delta_1) \vee \\ (q, q' \in Q_2 \wedge (q, a, q') \in \delta_2) \vee \\ (q = q^0 \wedge ((q_1^0, a, q') \in \delta_1 \vee (q_2^0, a, q') \in \delta_2))$$

It is easy to see that A is a PO-FSA with the partial order $\preceq$ defined as follows:

$$q \preceq q' \Leftrightarrow \begin{cases} \text{true} & \text{if } q = q^0 \\ q \preceq_1 q' & \text{if } q, q' \in Q_1 \\ q \preceq_2 q' & \text{if } q, q' \in Q_2 \end{cases}$$

□

The following proposition summarizes the properties of the piecewise languages.

Proposition 2.5 *Piecewise languages are closed under finite unions* $(+)$*, finite intersections* $(\cap)$*, concatenation* $(\cdot)$*, shuffle* $(||)$[1] *, letter-to-letter mappings, and inverse homomorphisms, but not under complementation and homomorphisms.*

PROOF. **Finite unions, intersections, and concatenation.** Closure under finite unions and concatenation follows immediately from Definition 2.1. Closure under finite intersections is shown in [Bouajjani et al. 2001], Proposition 1.

Shuffle. To show that piecewise languages are closed under shuffle, we show that PO-FSAs are closed under shuffle. Let L_1 and L_2 be two piecewise languages recognized by PO-FSAs, $A_1 = ((\Sigma, Q_1, \delta_1, q_1^0, F_1), \preceq_1)$ and $A_2 = ((\Sigma, Q_2, \delta_2, q_2^0, F_2), \preceq_2)$, respectively. Let $L = L_1||L_2$. Then, L is recognized by a PO-FSA, $A = ((\Sigma, Q, \delta, q^0, F), \preceq)$, where $Q = Q_1 \times Q_2$, $q^0 = (q_1^0, q_2^0)$, and $F = F_1 \times F_2$. The transition relation δ is defined as follows:

$$((q_1, q_2), a, (q_1', q_2')) \in \delta \Leftrightarrow ((q_1, a, q_1') \in \delta_1 \wedge q_2 = q_2') \vee ((q_2, a, q_2') \in \delta_2 \wedge q_1 = q_1')$$

It is easy to see that A is a PO-FSA, with the partial order $\preceq$ defined as follows:

$$(q_1, q_2) \preceq (q_1', q_2') \Leftrightarrow \begin{cases} q_1 \preceq_1 q_1' & \text{if } q_2 = q_2' \\ q_2 \preceq_2 q_2' & \text{if } q_1 = q_1' \end{cases}$$

Therefore, by Proposition 2.4, the language of the shuffled automaton A is piecewise.

Letter-to-letter mappings. Let $T : \Sigma \to \Sigma$ be a letter-to-letter mapping over a finite alphabet Σ. Consider simply piecewise language $M_0^* a_0 M_1^* \dots M_k^*$, where $M_0 =$

[1]The shuffle of two words w and w', $w||w'$, is the set of words that are obtained by interleaving w and w'; for example $ab||cd = \{abcd, acbd, acdb, cabd, cdab, cadb\}$ $(||)$. In the sequel, $\mathcal{L}||\mathcal{L}' = \bigcup_{\forall w \in \mathcal{L}, w' \in \mathcal{L}'} w||w'$.

$\{b_0, \ldots, b_i\}, M_1 = \{c_0, \ldots, c_j\}$, and so on. Applying T on this language results in the simply piecewise language $M_0'^* a_0' M_1'^* \ldots M_k'^*$ where $M_0' = \{T(b_0), \ldots, T(b_i)\}$, $a_0' = T(a_0), M_1' = \{T(c_0), \ldots, T(c_j)\}$ and so on. Clearly, this is a simply piecewise language. Since T distributes over union, the result follows for arbitrary piecewise languages as well.

Inverse homomorphisms. Let $A = ((\Sigma, Q, \delta, q^0, F), \preceq)$ be a partially ordered automaton accepting piecewise language L. Let Δ be an alphabet, and h a homomorphism from Δ to Σ^*. We construct automaton A' over Δ that accepts $h^{-1}(L)$. Intuitively A' works by reading a symbol a in Δ and simulating PO-FSA A on $h(a)$. Formally, let $A' = ((\Delta, Q, \delta', q^0, F), \preceq)$, and define $\delta'(q, a)$, for $q \in Q$ and $a \in \Delta$ to be $\delta(q, h(a))$. Since $h(a)$ may be a long string or ϵ, δ is defined on all strings by extension. It is easy to show by induction on $|x|$ that $\delta'(q^0, x) = \delta(q^0, h(x))$. Therefore, A' accepts x if and only if A accepts $h(x)$. That is , $\mathcal{L}(A') = h^{-1}(\mathcal{L}(A))$. The transition relation of A', δ', simulates the transition relation of A on $h(x)$ for any symbol $x \in \Delta$, thus it respects the partial order relation on states of A. Hence, $\mathcal{L}(A')$ is also piecewise.

Homomorphism. Piecewise languages are not closed under homomorphisms. For example, the piecewise language a^* under the homomorphisms $[a \mapsto (ab)]$ is $(ab)^*$, which is not piecewise.

Complementation. Piecewise languages are not closed under complementation. For example, consider a piecewise language $L = \Sigma^* aa\Sigma^* + \Sigma^* bb\Sigma^*$, with $\Sigma = \{a, b\}$. The complement of L is the set of sequences where a's and b's alternate — which is not piecewise. □

Proposition 2.6 *A language is repetition piecewise if and only if it is recognized by a PO-FSA $A = ((\Sigma, Q, \delta, q^0, F), \preceq)$, where $F = Q$ and δ satisfies the following two conditions. Let $q_i, q_j, q_l \in Q$ and $a, b \in \Sigma$. Then,*

(I) $(q_i, a, q_j) \in \delta \implies (q_j, a, q_j) \in \delta$, *and*
(II) $(q_i, a, q_j) \in \delta \wedge (q_j, b, q_l) \in \delta \implies (q_i, b, q_l) \in \delta$.

PROOF. ($\Rightarrow$) Let L be a simply repetition piecewise language and $L = M_0^* M_1^* \ldots M_k^*$. Let $A = ((\Sigma, Q, \delta, q^0, F), \preceq)$ be a PO-FSA with $k+1$ states where $Q = \{q_0, \ldots, q_k\}$, $q^0 = q_0$, and $F = Q$. For $i, j \in [0..k]$, δ is defined as follows:

$$(q_i, a, q_j) \in \delta \Leftrightarrow i \leq j \wedge a \in M_j \,.$$

The transition relation δ satisfies the conditions (I) and (II). The partial ordering is defined as follows: $q_i \preceq q_j \Leftrightarrow i \leq j$. We show that A recognizes L, i.e., $\mathcal{L}(A) = L$.

Let w be a word in L. Then, $w = P_0 \cdot P_1 \cdots P_k$, where $P_i \in M_i^*$. We use this partitioning to define an accepting run $\rho = \rho(0) \to \rho(1) \to \ldots \to \rho(n)$ of A on w as follows:

$$\rho(i) = q_j \Leftrightarrow \Sigma_{t=0}^{j-1} |P_t| \leq i < \Sigma_{t=0}^{j} |P_t|$$

Intuitively, the automaton goes to state q_i when reading a letter from partition P_i. It is easy to see that the run is well defined. It is accepting since every state of A is accepting. Thus, $L \subseteq \mathcal{L}(A)$.

To show $\mathcal{L}(A) \subseteq L$, assume $\rho = q_0 \to \ldots \to q_n$ is an accepting run of A on a word w, where $q_0 = q^0$. Then, ρ induces a partitioning $P_0, \ldots, P_k$ on w, such that $P_i \in M_i^*$. Hence, $w \in L$. Thus, $\mathcal{L}(A) \subseteq L$.

A repetition piecewise language is a finite union of simply repetition piecewise languages. Consider a repetition piecewise language $L = L_1 + L_2$, where L_1 and L_2 are

two simply repetition piecewise languages that are recognized by PO-FSAs A_1 and A_2, respectively, satisfying conditions (I) and (II). Similarly to the proof of Proposition 2.4, we construct PO-FSA A that recognizes L. It is easy to show that the construction satisfies conditions (I) and (II).

($\Leftarrow$) Let A be a PO-FSA satisfying conditions (I) and (II). For each state $q_i \in Q$, let $M_{q_i} = \{a \mid (q_i, a, q_i) \in \delta\}$. Let $\rho = q_0 \to \ldots \to q_n$ be an acyclic run of A, where every $q_i \in Q$ for $i \in [0..k]$ occurs at most once, and $q_0 = q^0$. The number of such runs is finite. Let the language L_ρ be defined as $M^*_{\rho(0)} \cdots M^*_{\rho(n)}$. It is easy to see that $L_\rho \in \mathcal{L}(A)$. Similarly, let $L' = \bigcup L_\rho$ over all such acyclic runs. Then, $L' \subseteq \mathcal{L}(A)$. Since L' is repetition piecewise, we only need to show that $\mathcal{L}(A) \subseteq L'$. Let w be a word in $\mathcal{L}(A)$, and ρ an accepting run of A on w. Let ρ' be a maximal subsequence of ρ in which every state in Q appears at most once. For example, if ρ is $q_0 \to q_0 \to q_1 \to q_1$, then ρ' is $q_0 \to q_1$. Then, ρ' is acyclic, and $w \in L_{\rho'}$. Hence, $\mathcal{L}(A) \subseteq L'$. □

The following proposition summarizes the properties of the repetition piecewise languages.

Proposition 2.7 *Repetition piecewise languages are closed under finite unions and intersections, concatenation, shuffle, and letter-to-letter mappings, but not under homomorphisms and inverse homomorphisms.*

PROOF. **Finite unions, intersections, concatenation, shuffle.** Closure under finite unions and concatenation follows immediately from Definition 2.1. To show closure under finite intersections, let L_1 and L_2 be two repetition piecewise languages. By Proposition 2.6, they are recognized by PO-FSAs A_1 and A_2, respectively, such that both A_1 and A_2 satisfy conditions (I) and (II) of the proposition. It is easy to check that conditions (I) and (II) are preserved by intersection and shuffle. Thus, the automata $A_1 \cap A_2$ and $A_1||A_2$ are PO-FSAs satisfying conditions (I) and (II). Hence, by Proposition 2.6 their languages are repetition piecewise.

Letter-to-letter mapping. The proof is similar to that of piecewise languages (Proposition 2.5).

Homomorphisms. Repetition piecewise languages are not closed under homomorphisms. For example, repetition piecewise language a^* under the homomorphisms $[a \mapsto (ab)]$ is $(ab)^*$ which is not piecewise.

Inverse homomorphisms. Repetition piecewise languages are not closed under inverse homomorphisms. For example, let $\Sigma = \{0\}$, and $\Sigma' = \{a, b\}$, and h be a homomorphism from Σ to Σ'^* such that $h(0) = ab$. Then, the repetition piecewise language $L = a^*b^*$ under the inverse homomorphism is $h^{-1}(L) = \{\epsilon, 0\}$ which is not repetition piecewise. □

For $a \in \Sigma$ and regular expressions R, S, the *left residual operation* (or derivative [Brzozowski and Simon 1973]) is defined as:

$$\begin{aligned}
a^{-1}\mathbf{0} &\triangleq \mathbf{0} \\
a^{-1}\mathbf{1} &\triangleq \mathbf{0} \\
a^{-1}b &\triangleq test(a \cap b) \\
a^{-1}(R \cdot S) &\triangleq (a^{-1}R \cdot S) + (test(R \cap \mathbf{1}) \cdot (a^{-1}S)) \\
a^{-1}(R + S) &\triangleq (a^{-1}R) + (a^{-1}S) \\
a^{-1}(R^*) &\triangleq (a^{-1}R) \cdot R^*
\end{aligned}$$

It is easy to see that $\mathcal{L}(a^{-1}R) = \{v \mid a \cdot v \in \mathcal{L}(R)\}$. Similarly, we may define a residual operation for M^*, where $M \subseteq \Sigma$:

$$\begin{aligned}
(M^*)^{-1}\mathbf{0} &\triangleq \mathbf{0} \\
(M^*)^{-1}\mathbf{1} &\triangleq \mathbf{1} \\
(M^*)^{-1}a &\triangleq a + test(a \cap M) \\
(M^*)^{-1}(R \cdot S) &\triangleq (((M^*)^{-1}R) \cdot S) + (test(R \cap M^*) \cdot ((M^*)^{-1}S)) \\
(M^*)^{-1}(R + S) &\triangleq ((M^*)^{-1}R) + ((M^*)^{-1}S) \\
(M^*)^{-1}(R^*) &\triangleq (((M^*)^{-1}R) \cdot R^*) + \mathbf{1}
\end{aligned}$$

Then, it can be verified that

$$\mathcal{L}((M^*)^{-1}R) \;=\; \{v \mid \exists u \in \mathcal{L}(M^*),\; u \cdot v \in \mathcal{L}(R)\}\,.$$

We conclude this section with a review of recognizable (or regular) relations.

Definition* 2.8 (*Recognizable Relation) [Yu 1997] A relation $\rho \subseteq (\Sigma^*)^K$ is *recognizable* (or regular) if and only if

$$\rho \;=\; \bigcup_{0 \le i < I} \mathcal{L}(R_0^i) \times \cdots \times \mathcal{L}(R_{K-1}^i)$$

for some natural number I and regular expressions R_j^i over Σ.

Similarly, we say that a relation is *piecewise* if and only if the expressions R_j^i above are piecewise, and say that a relation is *repetition piecewise* if and only if expressions R_j^i above are repetition piecewise.

Proposition 2.9 *[Yu 1997] Let ρ be a K-ary relation over Σ^*. Define $\mathcal{L}^{\#}(\rho) \triangleq \{w_0 \cdot \# \cdots \# \cdot w_{K-1} \mid (w_0, \ldots, w_{K-1}) \in \rho\}$. Then $\mathcal{L}^{\#}(\rho)$ is a regular language over $\Sigma \cup \{\#\}$ if and only if ρ is recognizable. Moreover, $\mathcal{L}^{\#}(\rho)$ is piecewise if and only if ρ is a piecewise relation.*

It is easy to see that regular and piecewise relations are closed under finite unions and intersections.

3. FIFO SYSTEMS AND THE REACHABILITY PROBLEM

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

A *channel* over an alphabet Σ is a FIFO queue whose contents is given by a word $w \in \Sigma^*$. 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, $?a : abb = bb$ and $!a : bb = bba$.

Let $\Sigma_{rw} \triangleq \{?, !\} \times \Sigma$ denote the read/write(rw)-alphabet over Σ. For a set of channels $C = \{c_1, \ldots, c_k\}$ this alphabet is extended as follows: $\Sigma_{rw}(C) \triangleq [1..k] \times \Sigma_{rw}$. Thus, an action $4?a$ corresponds to reading a from channel c_4, and $6!b$ corresponds to writing b to channel c_6. In the sequel, we drop C from the notation when it is clear from the context. We call Σ_{rw} an *action alphabet*, and any subset of Σ_{rw}^* an *action language*.

A *channel configuration* for a system with k channels is a k-tuple $\mathbf{w} \in (\Sigma^*)^k$. We use $\langle w_1, \ldots, w_k \rangle$ to denote a tuple, where w_i is the content of channel i. In single-channel

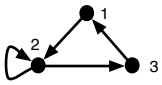

Fig. 3. An example of a communication graph for a set of actions $Act = \{1?a \rightarrow 2!a, 2?b \rightarrow 3!b, 3?b \rightarrow 1!a, 2?b \rightarrow 2!b\}$.

systems, a configuration is just the content of the single channel. We use bold fonts to differentiate between channel configurations in multi-channel and single-channel systems. Let $\mathbf{w}[i]$ denote the content of channel i in $\mathbf{w}$ and $\mathbf{w}[i \mapsto y]$ denote a channel configuration obtained from $\mathbf{w}$ by replacing the content of channel i with y.

In the single-channel case, for $X \subseteq \Sigma^*_{rw}$ and $W \subseteq \Sigma^*$, 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* 3.1 (*Action Language Semantics*)** Let $W \subseteq \Sigma^*$ be a set of words over Σ, and X an action language, then $X : W$ is defined as follows:

$$?a : W \triangleq (a^{-1})W \qquad !a : W \triangleq W \cdot a$$

$$\{x \cdot y\} : W \triangleq y : (x : W) \qquad X : W \triangleq \bigcup_{x \in X} (x : W)$$

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

Definition 3.1 is extended to a k-channel system as follows. Given $\mathbf{w} \in (\Sigma^*)^k$ and an action language X, then $X : \mathbf{w}$ for a single action is defined as shown below:

$$i?a : \mathbf{w} \triangleq \mathbf{w}[i \mapsto (?a : \mathbf{w}[i])] \qquad i!a : \mathbf{w} \triangleq \mathbf{w}[i \mapsto (!a : \mathbf{w}[i])]$$

and is extended to words identically to Definition 3.1. For example, given a 2-channel system, $(\{1?a\ 2!b, 1?a\ 2!c\} : \langle ab, b\rangle) = \{\langle b, bb\rangle, \langle b, bc\rangle\}$.

We write $?a \rightarrow !b$ for a *conditional action* that means "b is written only if a is first read." In other words, $?a \rightarrow !b$ is an abbreviation for a sequence of simple actions: $?a!b$. Given an action alphabet $\Sigma_{rw}(C)$ over a set of channels C, we define a conditional action alphabet $\Sigma_{rwc}(C)$ that treats conditional actions as letters:

$$\Sigma_{rwc}(C) \triangleq \Sigma_{rw}(C) \cup ((C \times \{?\} \times \Sigma) \cdot (C \times \{!\} \times \Sigma))\,.$$

For example, given $\Sigma = \{a\}$ and $C = \{1\}$, then $\Sigma_{rwc}(C) = \{1?a, 1!a, 1?a \rightarrow 1!a\}$.

For a set of actions $Act \subseteq \Sigma_{rwc}(C)$, a *communication graph* of Act, $CG(Act)$, is a digraph (C, E), with an edge $(i, j) \in E$ if and only if there are a and b in Σ such that $i?a \rightarrow j!b$ is in Act. For example, given $Act = \{1?a \rightarrow 2!a, 2?b \rightarrow 3!b, 3?b \rightarrow 1!a, 2?b \rightarrow 2!b\}$, $CG(Act)$ is a digraph with 3 nodes and 4 edges one for each conditional action in Act (see Fig. 3).

***Definition* 3.2 (*FIFO System*)** A FIFO system is a tuple $\mathcal{S} = (\Sigma, C, Q, q^0, \delta)$, where Σ 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 $\delta \subseteq Q \times \Sigma_{rwc} \times Q$ is a set of transition rules.

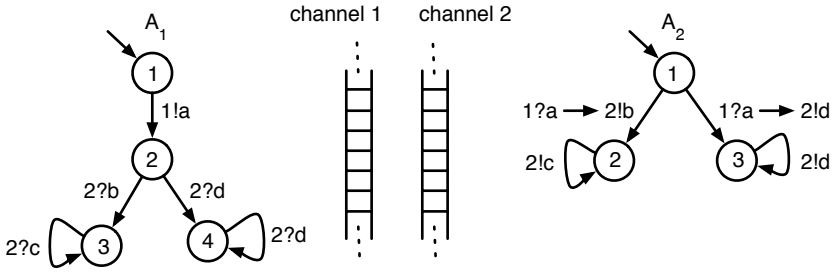

Fig. 4. An example of a FIFO system consisting of two processes and two channels.

Note that in Definition 3.2, a FIFO system is defined with respect to a conditional action alphabet Σ_{rwc}. A *global state* of $\mathcal{S}$ is a pair $(q, \mathbf{w})$ where q is a state in Q and $\mathbf{w}$ is a channel configuration. The transition relation of $\mathcal{S}$, Δ, is a set of triples of the form $((q, \mathbf{w}), op, (q', \mathbf{w}'))$, where $op \in \Sigma_{rwc}$, $(q, op, q') \in \delta$, and $\mathbf{w}' \in (op : \mathbf{w})$.

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

Most often a FIFO system is represented as a set $\{A_i\}_i^n$ of n processes communicating through a set of channels, C. Each process is a finite state automaton, $A_i = (\Sigma, Q_i, \delta_i, q_i^0)$. The corresponding FIFO system, $\mathcal{S} = (\Sigma, C, Q, q^0, \delta)$, is constructed by computing the cross product of these automata. Thus, $Q \triangleq \Pi_{i=1}^n Q_i$ and the transition relation Δ of $\mathcal{S}$ is built up from the transition relations of the A_i's such that every transition in Δ corresponds to exactly one transition in some δ_i. Formally,

$$(((q_1, \ldots, q_n), \mathbf{w}), op, ((q_1', \ldots, q_n'), \mathbf{w}')) \in \Delta \text{ if and only if}$$
$$\exists i, \quad \forall j \neq i,\ q_j = q_j' \ \wedge \ q_i' \in \delta_i(q_i, op) \ \wedge \ \mathbf{w}' \in (op : \mathbf{w}) .$$

Fig. 4 shows an example of a piecewise FIFO system consisting of two processes and two channels. Initially, we assume that both of the processes are in their initial states and both of the channels are empty, thus, the initial global state of the system is $((1,1), \langle \epsilon, \epsilon \rangle)$. Then, process A_1 writes a on channel 1 and moves from state 1 to 2. The new global state of the system is $((2,1), \langle a, \epsilon \rangle)$. Therefore, $(((1,1), \langle \epsilon, \epsilon \rangle), 1!a, ((2,1), \langle a, \epsilon \rangle)) \in \Delta$.

In this article, we are interested in the reachability problem:

FIFO Systems Reachability Problem. Given a FIFO system $\mathcal{S}$ and a set of configurations **I** (called initial), find the set of all global states reachable from **I**.

The set of all reachable global states of a FIFO system can be partitioned based on the control locations. Each partition represents the set of all reachable channel configurations at a particular control location. Thus, in order to calculate the set of all reachable global states, we need to calculate the set of all reachable channel configurations at each control location. This problem can be reduced to computing the semantics (Definition 3.1) of a regular action language.

Proposition 3.3 *Let $\mathcal{S} = (\Sigma, C, Q, q^0, \delta)$ be a FIFO system, $q \in Q$ some control location, and* **I** *a set of configurations. Then, the set of all reachable configurations of $\mathcal{S}$ at control*

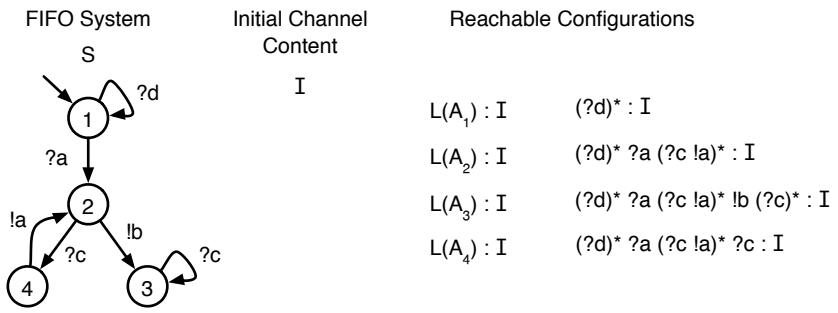

Fig. 5. An example illustrating the calculation of all reachable global states by computing the semantics of a regular action language.

location q is $(\mathcal{L}(A_q) : \mathbf{I})$, where $A_q = (\Sigma_{rwc}, Q, q^0, \delta, \{q\})$ is a finite automaton with accepting state q.

Fig. 5 is an example illustrating this reduction. On the left, we show an example of a FIFO system and on the right the set of all reachable configurations at control locations 1, 2, 3, and 4.

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 $\mathbf{W}$, compute the language of $(L_a^* : \mathbf{W})$. In the particular case of piecewise FIFO systems, L_a is further restricted to subsets of Σ_{rwc}. This is the problem we study in the rest of the article.

Proposition 3.4 *For regular (piecewise) L, it holds that $(?a : L)$, $(!a : L)$, and $(?a \rightarrow !b : L)$ are regular (piecewise).*

PROOF. For a single write action, $(!a : L) = L \cdot a$. For a read action, we have $(?a : L) = a^{-1}L$ following from the definition of derivative. For the conditional action, we have $(?a \rightarrow !b : L) = (a^{-1}L) \cdot b$. □

4. 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 language, show its correctness, and discuss its worst case complexity.

Fig. 6 shows the algorithm SINGLELIMIT for calculating the limit language. The inputs to the algorithm are an automaton A_I representing a set of single-channel configurations $I \subseteq \Sigma^*$, and a set $Act \subseteq \Sigma_{rwc}$ of actions; the output is an automaton that accepts the limit language $(Act^* : I)$. For notational convenience, in the examples we use regular expressions instead of automata to represent channel configurations.

The algorithm has two phases. In the first phase, called PHASE1 (lines 3 – 6 of the SINGLELIMIT), 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. Each iteration of PHASE1 is done using the function APPLY. Let $Act \subseteq \Sigma_{rwc}$ be partitioned into unconditional write actions $Act_w = \{!a \mid !a \in Act\}$, and the rest $Act_r = Act \setminus Act_w$. In each iteration, if V is the set of currently reachable

1: **function AUT** SINGLELIMIT(**Aut** $\mathtt{A_I}$, **Set** Act)
2: $\mathtt{R} := \epsilon,\ \mathtt{F} := \mathtt{A_I}$
3: **while** $\mathcal{L}(\mathtt{F}) \not\subseteq \mathcal{L}(\mathtt{R})$ **do**
4: $\mathtt{R} := \mathtt{R} + \mathtt{F}$
5: $\mathtt{F} := \text{APPLY}(\mathtt{F}, \mathtt{Act})$
6: **return** PHASE2($\mathtt{R}, \mathtt{Act}$)

Fig. 6. The SINGLELIMIT algorithm.

configurations, APPLY computes V' such that

$$V' \triangleq \{v \mid \exists u \in V,\ v \in (Act_r^{|u|} : u)\} \mid\mid (Act_w^* : \epsilon)\ .$$

Note that APPLY misses some reachable configurations. For example, let $Act = \{?a \rightarrow !c, ?b \rightarrow !d, !e\}$ and $I = ab$. Then, APPLY results in $\mathcal{L}(e^*ce^*de^*)$ and misses reachable configurations in $\mathcal{L}(be^*ce^*)$. This is fixed in the second phase, called PHASE2. Let W be a set of reachable configurations, the result of PHASE2 is a set W' such that

$$W' \triangleq \{w \mid \exists u, v, z,\ (v \cdot u \in W) \wedge (u \cdot z = w) \wedge (z \in \text{APPLY}(\{v\}, Act))\}\ .$$

These two phases are implemented using automata as described below.

PHASE1. As inputs PHASE1 takes an automaton $A = (\Sigma, Q, \delta, q^0, F)$, and a set of actions Act. Then, it iteratively computes a set of reachable configurations using function APPLY. Given automaton A and a set of actions Act, APPLY constructs an automaton $A' = (\Sigma, Q, \delta', q^0, F)$, where δ' consists of tuples of the form:

—(q, ϵ, q') if for some a it holds that $\delta(q, a, q')$ and $?a \in Act$, or
—(q, b, q') if for some a it holds that $\delta(q, a, q')$ and $?a \rightarrow !b \in Act$, or
—(q, c, q) if $!c \in Act$.

Intuitively, the first rule of δ' corresponds to unconditional reads, the second – to renaming the labels of the transitions according to the conditional actions, and the third – to unconditional writes.

For example, let $Act = \{?a \rightarrow !b, ?b \rightarrow !a, ?c, !a\}$ and $I = (ac)^*aba^*$. Fig. 7(a) shows automaton A recognizing $\mathcal{L}(I)$. To construct $A' = \text{APPLY}(A, Act)$, the transitions labeled by a are relabeled to b, transitions labeled by b are relabeled to a, and transitions labeled by c are replaced by ϵ-transitions. In addition, self-loop transitions labeled by a are added to every state. Fig. 7(b) shows automaton A'. Similarly, we can construct automaton $A'' = \text{APPLY}(A', Act)$ and $A''' = \text{APPLY}(A'', Act)$ which are shown in Fig. 7(c) and (d), respectively. As can be seen, applying APPLY once more results in automaton A'', thus, we have reached a fixpoint.

PHASE2. Let $A = (\Sigma, Q, q^0, \delta, F)$ be an automaton and s be a state in Q. We construct two automata: $A_1 = (\Sigma, Q, q^0, \delta, \{s\})$ and $A_2 = (\Sigma, Q, \{s\}, \delta, F)$. Let A_1' be the automaton constructed by applying APPLY to A_1, i.e., $A_1' = \text{APPLY}(A_1, Act)$. Then, the language of $A_2 \cdot A_1'$ contains a word $u \cdot z$ if and only if (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) $z \in \text{APPLY}(\{v\}, Act)$. We call this operation PREFIX(A, s, Act). It is easy to see that:

$$\text{PHASE2}(A, Act) = \bigcup_{s \in Q} \text{PREFIX}(A, s, Act)\ .$$

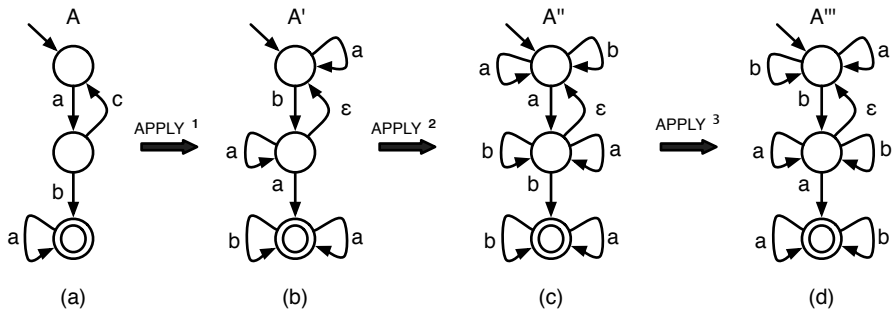

Fig. 7. An example illustrating PHASE1 with automaton A and $Act = \{?a \rightarrow !b, ?b \rightarrow !a, ?c, !a\}$ as inputs.

For our running example, Fig. 8 shows how $\textsc{Prefix}(A, s, Act)$ is implemented using automata. The leftmost automaton in Fig. 8 (automaton A) recognizes the language $I = (ac)^*aba^*$. To compute $\textsc{Prefix}(A, s, Act)$, we break A on state s (see Fig. 8), which results in two automata A_1 and A_2. We compute A_1' by applying APPLY to A_1. Then, we concatenate A_2 and A_1'. The resulting automaton represents $\textsc{Prefix}(A, s, Act)$ and is shown on the rightmost of the Fig. 8.

The algorithm in Fig. 6 always terminates. Given an automaton A, APPLY produces an automaton with the same number of states as A. Thus, the set $\{\textsc{Apply}^i(A, Act)\}_i$ is finite, and the algorithm always terminates.

Theorem 4.1 *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* $\textsc{SingleLimit}(A_I, Act)$. *Then,* $\mathcal{L}(A_L) = (Act^* : \mathcal{L}(A_I))$.

PROOF. According to the SINGLELIMIT algorithm shown in Fig. 6,

$$\mathcal{L}(A_L) = \textsc{Phase2}\left(\bigcup_{i\in\mathbb{N}} \textsc{Apply}^i(A_I, Act), Act\right).$$

Note that since in each iteration APPLY produces an automaton with the same number of states as A_I, $\bigcup_i \textsc{Apply}^i(A_I, Act)$ is a finite union.

Let $w \in (Act^* : \mathcal{L}(A_I))$ be a reachable channel content. Then, w is reached by reading the current channel content completely (and writing the results of conditional and other write actions) zero or more times, and then reading the resulting content partially. Let $\#$ – a fresh letter not in Σ, be a marker at the end of the initial channel content. The maker $\#$ is used only for establishing the proof and is eliminated later using $\textsc{Erase}_{\#}$. Then,

$$w \in (Act^* : \mathcal{L}(A_I)) \Leftrightarrow \exists u, v,\ (u \cdot v) = w \wedge \exists p, q,\ (u\#v) \in (((Act^*(?\#)(!\#))^p (Act)^q) : (\mathcal{L}(A_I) \cdot \#)).$$

At the end of each iteration of APPLY, $\#$ is read and then written again on the channel to mark the beginning of the new iteration.

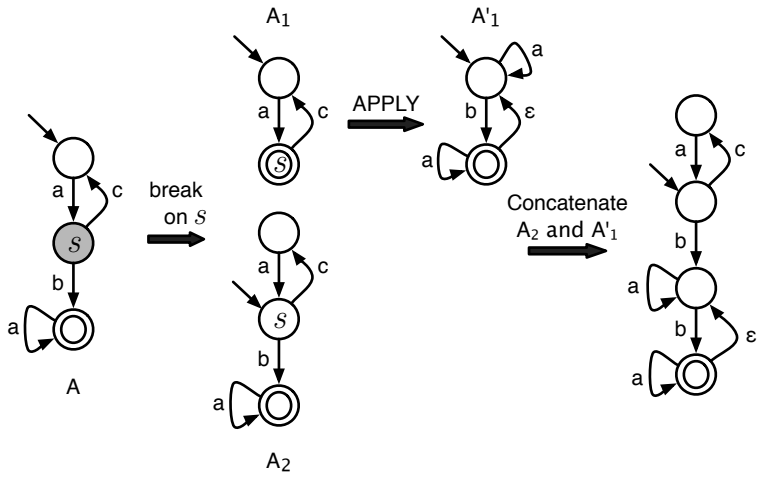

Fig. 8. An example illustrating PREFIX operation with automaton A, state s, and $Act = \{?a \rightarrow !b, ?b \rightarrow !a, ?c, !a\}$ as inputs.

The theorem follows from the following two facts:

$$(\textsc{Apply}(\mathcal{L}(A), Act) \cdot \#) = (Act^*(?\#)(!\#)) : (\mathcal{L}(A) \cdot \#)$$

and

$$\textsc{Phase2}(\mathcal{L}(A), Act) = \textsc{Erase}_{\#}(Act^* : (\mathcal{L}(A) \cdot \#)) .$$

where $\textsc{Erase}_{\#}$ projects out the letter $\#$. □

Complexity Analysis. Let $h = |A_I|$ denote the size of A_I – the automaton representing the set of initial configurations. As discussed above, $\textsc{Apply}(A_I, Act)$ produces an automaton with the same number of states as A_I by relabeling the transitions of A_I. In the worst case, each transition can be updated at most $|\Sigma|$ times. Thus, the worst case complexity of the SINGLELIMIT algorithm is $|\Sigma|^h$.

Theorem 4.2 *Let A_I be an automaton over a finite alphabet Σ representing a set of single-channel configurations, and $h = |A_I|$. Then, in the worst case, the running time of the* SINGLELIMIT *algorithm is $O(|\Sigma|^h)$.*

5. DECIDABILITY RESULTS ON MULTI-CHANNEL PIECEWISE SYSTEMS

In this section, we focus on the limit language problem for a set of actions, Act, on a k-channel system, and a set of channel configurations $\mathbf{L}$. A configuration $\langle w_1, \ldots, w_k \rangle$ of a k-channel system is represented by a word of the form $w_1 \cdot \# \cdots \# \cdot w_k$, where $\#$ is a fresh letter not in Σ. Thus, a channel configuration can be seen as an element of a relation. In the sequel, a set of channel configurations correspond to a relation over Σ^*. A *regular configuration* is a set of channel configurations that correspond to a regular relation and a

piecewise configuration is a set of channel configurations that correspond to a piecewise relation (refer to Definition 2.8).

We show that limit languages of multi-channel systems are not regular in general. However, regularity can be achieved by either restricting Act to exclude conditional actions, or restricting $\mathbf{L}$ to piecewise configurations.

Proposition 5.1 *There exists a set of actions $Act \subseteq \Sigma_{rwc}$ and a regular configuration $\mathbf{L}$, such that the limit language $(Act^* : \mathbf{L})$ is not regular.*

PROOF. Consider a 2-channel action set $Act = \{0?a \to 1!a, 0?b \to 0!b', 0?b' \to 1!b\}$ and a regular configuration $\mathbf{L} = \langle (ab)^*, \epsilon \rangle$. The idea is that first all the a's are transferred to channel 1, then all the b's (after each b has temporarily been renamed to b'). In configuration $\mathbf{L}$, channel 0 contains exactly the same number of a's and b's. After applying Act^*, there should be equal number of a and b in channel 1, and all b's should follow a's. Thus, we have

$$(Act^* : \mathbf{L}) \cap \langle \epsilon, \Sigma^* \rangle = \{ \langle \epsilon, a^n b^n \rangle \mid n \geq 0 \} .$$

Hence, $(Act^* : \mathbf{L})$ is non-regular. □

In the rest of this section, we show how to achieve regularity in the limit language. First, we show that restricting the action set to unconditional actions is sufficient to make the limit language regular. However, excluding the conditional actions significantly restricts the expressiveness of the system model. We then show that to achieve regularity in the presence of conditional actions, we need to restrict the initial channel configuration to only piecewise configurations.

5.1 Limit Languages by Restricting the Action Set

In this section, we present results on recognizability of limit languages in piecewise multi-channel systems where conditional actions are excluded from the action set.

The following proposition shows that the limit language of a set of read and write actions is regular (piecewise) if the initial channel language is regular (piecewise).

Proposition 5.2 *Let $Act \subseteq \Sigma_{rw}$ be a set of unconditional actions in a multi-channel system and $\mathbf{L}$ a regular (piecewise) configuration. Then,*

(a) $(Act^* : \mathbf{L})$ is regular (piecewise).

(b) For a piecewise expression T over Act, $(T : \mathbf{L})$ is regular (piecewise).

PROOF. Without loss of generality, we consider a 2-channel system. Generalization to systems with more channels is trivial.

(a) Since $\mathbf{L}$ is regular (piecewise), it is equivalent to $\sum_{0 \leq i < I} R_0^i \# R_1^i$ for some I, where each R_j^i is regular (piecewise), and $\#$ is a fresh letter not present in Σ (Proposition. 2.9). Let $M_k = \{a \mid k?a \in Act\}$ and $N_k = \{a \mid k!a \in Act\}$ for $k = 0$ and $k = 1$ represent the set of all letters that are read and written, respectively. Then,

$$(Act^* : \mathbf{L}) = \sum_{0 \leq i < I} ((M_0^*)^{-1} R_0^i) \cdot N_0^* \# ((M_1^*)^{-1} R_1^i) \cdot N_1^* .$$

Hence, $(Act^* : \mathbf{L})$ is regular (piecewise) if $\mathbf{L}$ is regular (piecewise).

(b) By definition, T is a sum of simply piecewise expressions of the form

$$Act_0^* \; act_0 \cdots act_\ell \; Act_{\ell+1}^*$$

on which (a) can be applied inductively. □

5.2 Limit Languages by Restricting the Initial Channel Configuration

In this section, we present results on recognizability of limit languages in multi-channel systems with initial piecewise channel configurations. First, we review some necessary definitions such as well-quasi-ordering and upward and downward closed sets. Then, we establish our main result by showing that an *arbitrary* union of repetition piecewise relations is piecewise.

5.2.1 *Preliminaries.* In the rest of this section, we assume a finite alphabet Σ and use x and y to denote elements of Σ^*.

***Definition* 5.3 (*WQO*)** [Kruskal 1972] A binary relation $\preceq$ on a set X is a *well-quasi-ordering (wqo)* on X if $\preceq$ is reflexive, transitive, and any infinite sequence of elements $x_0x_1x_2\cdots$ from X contains an increasing pair $x_i \preceq x_j$ with $i < j$. The set X is said to be *well-quasi-ordered* by $\preceq$.

Note that if X is well-quasi-ordered then it does not contain an infinite descending chain, nor an infinite set of pairwise incomparable elements.

***Definition* 5.4 (*Subword Ordering*)** The subword relation $\leq \subseteq \Sigma^* \times \Sigma^*$ is defined such that $x \leq y$ if and only if x can be obtained by deleting some letters of y.

For example, $ac \leq abc$, but $abc \not\leq abd$. The relation $\leq$ is reflexive and transitive. Furthermore, it follows from Higman's Lemma [Higman 1952] that the subword ordering relation is a wqo. Intuitively, this means that any *infinite* subset of Σ^* contains at least two words x and y such that x is a subword of y.

***Definition* 5.5 (*Upward/Downward Closure*)** For a set of words A, the *upward closure of* A, denoted $A^{\leq}$, is the set of all words y such that $x \leq y$ for some $x \in A$. Dually, the *downward closure* of A, denoted $A^{\geq}$, is the set of all words y such that $y \leq x$ for some $x \in A$.

For example, an upward closure of the singleton set $\{abc\}$ is:

$$\begin{aligned}\{abc\}^{\leq} &= \{y \mid abc \text{ is a subword of } y\} \\ &= \Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*\end{aligned}$$

The downward closure of the same set is:

$$\begin{aligned}\{abc\}^{\geq} &= \{y \mid y \text{ is a subword of } abc\} \\ &= \{abc, ab, ac, bc, a, b, c, \epsilon\}\end{aligned}$$

We say that a set A is upward closed or an upset if $A^{\leq} = A$, and that it is downward closed or a downset if $A^{\geq} = A$. For example, $\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*$ is an upset, $a^*b^*c^*$ is a downset, and $\{abc\}$ is neither an upset nor a downset.

An upward closed set is uniquely determined by its minimal elements. For an upward closed set A, let $\text{MIN}(A) \triangleq \{x \in A \mid \nexists y \in A,\ y \leq x\}$, then $A = (\text{MIN}(A))^{\leq}$. Note

that all elements of MIN(A) are pairwise incomparable and since $\leq$ is a wqo, MIN(A) is finite.

The subword ordering is extended pointwise to tuples of words. Let $\mathbf{w}, \mathbf{u} \in (\Sigma^*)^K$ be such that $\mathbf{w} = (w_1, \ldots, w_K)$ and $\mathbf{u} = (u_1, \ldots, u_K)$. Then, $\mathbf{w} \leq \mathbf{u}$ if and only if $\forall i,\ w_i \leq u_i$. This ordering is still a wqo, since it is a cross product of well-quasi-orderings.

The notions of upward and downward closures extend to sets of tuples in a natural way. In particular, an upward closure of a set containing a single tuple is:

$$\{(w_1, \ldots, w_n)\}^{\leq} = \{w_1\}^{\leq} \times \{w_2\}^{\leq} \times \cdots \times \{w_n\}^{\leq}$$

5.2.2 *Repetition Piecewise Relations.* In this section, we show that an *arbitrary* union of repetition piecewise relations is piecewise.

Repetition piecewise languages are downward closed with respect to the subword ordering. For example, let $\Sigma = \{a, b, c\}$, and $L = a^*b^*$. The downward closure of L is the set of all words that have an arbitrary number of a's followed by an arbitrary number of b's, which is L itself.

For any downset $L \subseteq \Sigma^*$, its complement, $\complement L \subseteq \Sigma^*$, is upward closed. Since subword ordering $\leq$ is a wqo, there exists a finite set $A =$ MIN$(\complement L)$ such that $\complement L = A^{\leq}$. For example, for the language L above, $\complement L = \{c, ba\}^{\leq}$.

The same is true of repetition piecewise relations: they are downward closed, and allow for a finite representation of their complements.

Lemma 5.6 *Let $R \subseteq (\Sigma^*)^n$ be a relation that is upward closed with respect to the subword ordering. Then, there exists a finite set $\{r_1, \ldots, r_k\} \subseteq (\Sigma^*)^n$ such that $R = \{r_1, \ldots, r_k\}^{\leq}$.*

PROOF. Since R is upward closed, $R =$ MIN$(R)^{\leq}$, and all elements of MIN(R) are pairwise incomparable. Since $\leq$ is a wqo, MIN(R) is finite. □

The following lemma shows that the language of the complement of the upward closure of a tuple of strings is piecewise.

Lemma 5.7 *Let $\mathbf{w} \in (\Sigma^*)^n$ be a tuple of strings, and $\{\mathbf{w}\}^{\leq}$ be its upward closure. Then, $\complement\{\mathbf{w}\}^{\leq}$ is piecewise.*

PROOF. Assume that $\mathbf{w}$ is of the form $\mathbf{w} = (w_1, \ldots, w_n)$. Then $\{\mathbf{w}\}^{\leq} = \{w_1\}^{\leq} \times \cdots \times \{w_n\}^{\leq}$. The complement of this set can be expressed as:

$$\begin{aligned} \complement\{\mathbf{w}\}^{\leq} \;=\; & (\complement\{w_1\}^{\leq} \times \Sigma^* \times \cdots \times \Sigma^*) \quad \cup \\ & (\Sigma^* \times \cdots \times \complement\{w_i\}^{\leq} \times \cdots \times \Sigma^*) \quad \cup \\ & \cdots \quad \cup \\ & (\Sigma^* \times \cdots \times \Sigma^* \times \complement\{w_n\}^{\leq})\,. \end{aligned}$$

Σ^* is trivially piecewise. Thus, it is sufficient to show that $\complement\{w_i\}^{\leq}$ is piecewise for any word w_i.

Let $w_i = w_i(1)w_i(2)\cdots w_i(k)$, where $w_i(j)$ is the j-th letter in w_i. The upward closure of w_i is the set of all words y that contain w_i as a subword. That is, $\{w_i\}^{\leq} = \mathcal{L}(\Sigma^* w_i(1)\Sigma^* w_i(2)\cdots\Sigma^* w_i(k)\Sigma^*)$. Thus, $\complement\{w_i\}^{\leq}$ is the union of a set of languages as

follows:

$$\begin{array}{c} (\Sigma - \{w_i(1)\})^* \quad \cup \\ (\Sigma - \{w_i(1)\})^* w_i(1)(\Sigma - \{w_i(2)\})^* \quad \cup \\ \cdots \quad \cup \\ (\Sigma - \{w_i(1)\})^* w_i(1)(\Sigma - \{w_i(2)\})^* \cdots w_i(k-1)(\Sigma - \{w_i(k)\})^* . \end{array}$$

$\complement\{w_i\}^{\leq}$ is a finite union of a set of simply piecewise expressions, and therefore it is piecewise. □

For example, consider a (trivial) tuple $w = ab$ and $\Sigma = \{a, b, c\}$. The upward closure of w is $\{ab\}^{\leq} = \Sigma^* a \Sigma^* b \Sigma^*$. The complement $\complement\{ab\}^{\leq}$ is represented by a piecewise expression $(b + c)^* + (b + c)^* a (a + c)^*$.

The following Lemma extends Lemma 5.6 to all downward closed relations.

Lemma 5.8 *A relation that is downward closed with respect to the subword ordering is piecewise.*

PROOF. Let R be a downward closed relation. Its complement $\complement R$ is upward closed. By Lemma 5.6, $\complement R = \{r_1, \ldots, r_k\}^{\leq} = \bigcup_{i=1}^{k} \{r_i\}^{\leq}$, and $R = \bigcap_{i=1}^{k} \complement\{r_i\}^{\leq}$. By Lemma 5.7, R is a finite intersection of piecewise relations and is therefore a piecewise relation. □

For example, (a^*b^*, a^*c^*) is downward closed and piecewise. The following lemma establishes the piecewiseness of an *arbitrary* union of repetition piecewise relations.

Proposition 5.9 *An arbitrary union of a family of repetition piecewise relations is a piecewise relation.*

PROOF. Repetition piecewise relations are downward closed and an arbitrary union of downward closed sets is downward closed. By Lemma 5.8, this union is a piecewise relation. □

5.2.3 *Main Results.* Our main result follows directly from Proposition 5.9. However, first we need to introduce the notion of an *anchor sequence*.

***Definition* 5.10** The *anchor sequences* of a piecewise expression

$$R = \textstyle\sum_{0 \leq i < I} M_1^{i*} a_1^i \cdots {M_{k(i)}^i}^* a_{k(i)}^i M_{k(i)+1}^{i^*}$$

is the set $\{a_1^i \cdots a_{k(i)}^i | 0 \leq i < I\}$. The *anchor length* of R is $\max_{0 \leq i < I} k(i)$ and the anchor length of piecewise L is the minimum anchor length of an R such that $\mathcal{L}(R) = L$.

For example, given piecewise $R = (a + b)^* cd^* ba^* + d^* ba^*$, the anchor sequences of R is $\{cb, b\}$, and the anchor length of R is 2.

The following proposition shows that an arbitrary union of piecewise languages with bounded anchor length is piecewise.

Proposition 5.11 *The union of any (possibly infinite) family of piecewise languages of bounded anchor length is piecewise.*

PROOF. Note that for any bound, there are finitely many different anchor sequences. By a rearrangement of the simply piecewise expressions of the languages of the family, it suffices to consider a finite union of languages L_ℓ, where L_ℓ is a union of simply piecewise languages all having the same anchor sequence. Since piecewise languages are closed

Initial Configuration	Channel Configuration	
	multiple w/o conditionals	multiple w/ conditionals
regular	effectively regular	non-regular
piecewise	effectively piecewise	**piecewise**

Table I. A summary of the decidability results for limit languages in piecewise FIFO systems.

under finite union, it is sufficient to show that L_ℓ is piecewise. Consider $L_\ell = \bigcup_{i\geq 0} \mathcal{L}(R^i)$, where R^i for example has the following form:

$$R^i = M_1^{i*} a_1^i \cdots a_{k(\ell)-1}^{i} {}^* M_{k(\ell)}^{i} {}^* a_{k(\ell)}^i M_{k(\ell)+1}^{i*} .$$

By renaming a_j^i's to $\#$, L_ℓ can be viewed as a union of repetition piecewise relations. According to Proposition 5.9, L_ℓ is a piecewise relation. By restoring $\#$ with a_j^i's, we get that the language L_ℓ is piecewise. □

Recall that a piecewise configuration $\mathbf{L}$ can be seen as a piecewise language; hence, it has an anchor sequence. The following proposition shows that the repeated application of a single read, write, or a conditional action to a piecewise configuration does not increase its anchor length.

Proposition 5.12 *For an action $act \in \Sigma_{rwc}$ and a piecewise channel configuration $\mathbf{L}$ with anchor length l, the anchor length of $(act^* : \mathbf{L})$ is less or equal to l.*

PROOF. Without loss of generality, we only consider a 2-channel system and conditional actions. Generalization to systems with more channels and unconditional reads and writes is trivial. The proof proceeds by induction on the anchor length of $\mathbf{L}$. The base case is trivial (and is omitted), the inductive cases are shown below. We assume $\mathbf{L} = \langle U, V \rangle$, where U and V are simply piecewise expressions, and act is a conditional action of the form $1?a \rightarrow 2!b$.

—**Case 1:** $U = (a + a_1 + \cdots + a_n) \cdot W$, then

$$((1?a \rightarrow 2!b)^* : \mathbf{L}) = \langle U, V \rangle \ \vee ((1?a \rightarrow 2!b)^* : \langle W, V \cdot b \rangle)$$

—**Case 2:** $U = (a + a_1 + \cdots + a_n)^* \cdot W$, then

$$((1?a \rightarrow 2!b)^* : \mathbf{L}) = \langle U, V \cdot b^* \rangle \ \vee ((1?a \rightarrow 2!b)^* : \langle W, V \cdot b^* \rangle)$$

—**Case 3:** $U = c \cdot W$ or $U = \epsilon$, then

$$((1?a \rightarrow 2!b)^* : \mathbf{L}) = \epsilon$$

It is easy to see that in all of the cases above the anchor length has not increased. □

The following theorem establishes our main result that the limit language is piecewise if the initial channel language is piecewise.

Theorem 5.13 *Let Act be a set of actions in a multi-channel system and $\mathbf{L}$ a set of channel configurations. Then, $(Act^* : \mathbf{L})$ is piecewise if $\mathbf{L}$ is piecewise.*

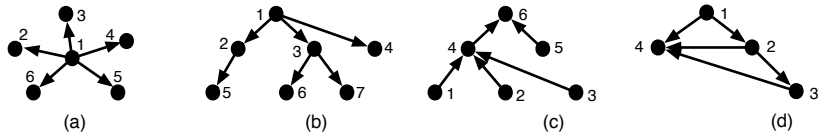

Fig. 9. Communication topologies: (a) star, (b) tree, (c) inverted tree, (d) DAG.

PROOF. Let $Act = \{act_1, \ldots, act_n\}$. Then,

$$\begin{aligned} Act^* : \mathbf{L} &= (act_1 + \cdots + act_n)^* : \mathbf{L} \\ &= ((act_1^* \cdot act_2^* \cdot \cdots \cdot act_n^*)^*) : \mathbf{L} \\ &= \textstyle\bigcup_{w \in Act^*} (\text{STARALL}(w)) : \mathbf{L} \end{aligned}$$

where for $w = w(1)w(2)\cdots$, $\text{STARALL}(w) = w(1)^* w(2)^* \cdots$.

In a k-channel system, we may assume that $\mathbf{L} = \sum_{0 \le i < I} L(i)$ for some I, where $L(i) = R_0^i \# R_1^i \# \cdots \# R_{k-1}^i$ and R_j^i for $0 \le j < k$ is simply piecewise. Then,

$$(Act^* : \mathbf{L}) \;=\; \textstyle\sum_{0 \le i < I} (\bigcup_{w \in Act^*} (\text{STARALL}(w)) : L(i)) \,.$$

Let l_i denote the anchor length of $L(i)$. By Proposition 5.12 $(\text{STARALL}(w) : L(i))$ is a union of piecewise expressions with anchor length bounded by l_i.

Let $L_w(i)$ denote $(\text{STARALL}(w) : L(i))$. Then,

$$(Act^* : \mathbf{L}) \;=\; \textstyle\sum_{0 \le i < I} (\bigcup_{w \in Act^*} L_w(i)) \,.$$

Therefore, $(Act^* : \mathbf{L})$ is a (possibly infinite) union of piecewise expressions with anchor length bounded by $\max\{l_i \mid 0 \le i < I\}$. Thus, by Proposition 5.11 it is piecewise. □

The results of this section are summarized in Table I, where the result of Theorem 5.13 is highlighted in bold. From the table, it is clear that in order for the limit language to be regular, we either need to exclude conditional actions or consider only piecewise initial channel configurations. The proof of Theorem 5.13 is non-effective: it does not provide an algorithm for computing the limit language. In the next section, we partially remedy this problem by providing algorithms for computing the limit language for restricted classes of piecewise multi-channel systems with piecewise initial channel configurations. However, answering this question in the general case remains an open problem.

6. ALGORITHMIC ANALYSIS OF MULTI-CHANNEL PIECEWISE SYSTEMS

In this section, we focus on the limit language problem for a set of actions with an acyclic communication graph. For ease of presentation, we develop the algorithm for acyclic communication graphs incrementally by restricting the topology of the graph to *star*, *tree*, *inverted tree*, and eventually *DAG*. We show correctness of each algorithm and discuss its complexity.

Throughout, we assume that all actions are conditionals. This is not a significant limitation since: (i) unconditional reads can be modeled by conditionals that write to dummy channels, and (ii) unconditional writes can be handled easily, but are omitted for presentational convenience. The algorithms are based on automata, and operate on piece-

wise configurations. A piecewise (k-channel) configuration $\mathbf{u}$ is represented by a tuple $\langle A_1, \ldots, A_k \rangle$, where each A_i is a PO-FSA over Σ. The *size* of $\mathbf{u}$ is the sum of the sizes of all of the automata in it. $\mathcal{L}$ is extended to finite sets of piecewise configurations in the usual way: $\mathcal{L}(\mathbf{U}) = \bigcup_{\mathbf{u} \in \mathbf{U}} \mathcal{L}(\mathbf{u})$. Note that $\mathcal{L}(\mathbf{U})$ can be seen as a piecewise recognizable relation.

For notational convenience, in the examples we use tuples of regular expressions instead of PO-FSA to represent piecewise configurations. For example, $\mathbf{u} = \langle a^*b, (c+d)^*e \rangle$ represents a piecewise configuration where $\mathbf{u}[1]$ is an automaton representing a^*b, and $\mathbf{u}[2]$ is an automaton representing $(c+d)^*e$. In pseudo-code, we use **Conf** for the type of piecewise configurations, and the notation "X **with** $[i] = y$" to mean $X[i \mapsto y]$.

6.1 Star Topology

A set of actions Act has a *star* topology if and only if there exists a unique channel o, the *origin*, such that for every action $i?a \rightarrow j!b$ in Act, $i = o$ and $j \neq o$, i.e., $CG(Act)$ is a star (see Fig. 9(a)). In the sequel, we assume that channel 1 is the origin channel.

Let $\mathbf{u}$ be a piecewise channel configuration. The algorithm DoRead, shown in Fig. 10, computes the limit $(Act^* : \mathcal{L}(\mathbf{u}))$. The `writeWL` and `readWL` are two global work lists used by the algorithm. DoRead is driven by the automaton $\mathbf{u}[1]$ representing the content of channel 1. For example, if $\mathbf{u}[1] = M_1^* a_1 M_2^* a_2$ then the algorithm first computes all reachable configurations $\mathbf{w}$ whose channel 1 content, $\mathbf{w}[1]$, is in $\mathcal{L}(M_1^* a_1 M_2^* a_2)$, then all configurations with $\mathbf{w}[1]$ in $\mathcal{L}(M_2^* a_2)$, then all configurations with $\mathbf{w}[1]$ being ϵ. Each iteration of the algorithm is done using functions Saturate and Step. For our running example, in the first iteration, Saturate computes all reachable configurations with $\mathbf{w}[1]$ in $\mathcal{L}(M_1^* a_1 M_2^* a_2)$ and Step computes all configurations with $\mathbf{w}[1]$ in $\mathcal{L}(M_2^* a_2)$, etc.

Saturate. Let $\mathbf{u}$ be a piecewise configuration, where $\mathbf{u}[1] = M^* \cdot Z$ for some Z, i.e., $\mathbf{u}[1]$ is a PO-FSA with a single initial state q^0 and some self-loops on q^0. Note that, $\mathbf{u}$ represents a set of configurations with an arbitrary number of letters from M at the head of channel 1. The Saturate phase computes a set of configurations that are reachable by reading an arbitrary number of these letters. Formally, $\textsc{Saturate}(\mathbf{u}, 1)$ defines a piecewise configuration $\mathbf{u}'$ such that $\mathcal{L}(\mathbf{u}')$ is $\{\mathbf{w} \mid \mathbf{w} \in (\text{Act}^* : \mathcal{L}(\mathbf{u})) \wedge (\mathbf{w}[1] \in \mathcal{L}(\mathbf{u}[1]))\}$. It corresponds to a transformation $\textsc{Saturate}(\mathbf{u}, 1) = \mathbf{u}'$ such that

$$\mathbf{u}'[i] \triangleq \begin{cases} \mathbf{u}[1] & \text{if } i = 1 \\ \mathbf{u}[i] \cdot (Act : M)^* & \text{otherwise.} \end{cases}$$

For example, given $Act = \{1?a \rightarrow 2!a, 1?a \rightarrow 3!a\}$ and $\mathbf{u} = \langle a^*(b+c), \epsilon, \epsilon \rangle$, $\textsc{Saturate}(\mathbf{u}, 1)$ computes a piecewise configuration $\mathbf{u}' = \langle a^*(b+c), a^*, a^* \rangle$.

Step. Let $\mathbf{u}$ be a piecewise channel configuration, where $\mathbf{u}[1] = (a_0 + \cdots + a_n) \cdot Z$ for some Z, i.e, $\mathbf{u}[1]$ is a PO-FSA with a single initial state with no self-loops. Here, $\mathbf{u}$ represents a set of configurations whose channel 1 content starts with a letter in $\{a_0, \ldots, a_n\}$. The Step phase computes all configurations that are reachable by reading exactly one letter from channel 1. Formally, $\textsc{Step}(\mathbf{u}, 1)$ defines a set $\mathbf{U}'$ of piecewise configurations such that $\mathcal{L}(\mathbf{U}') \triangleq \{\mathbf{w} \mid \mathbf{w} \in (Act^* : \mathcal{L}(\mathbf{u})) \wedge (\mathbf{w}[1] \in Z)\}$. It corresponds to a transformation

$$\textsc{Step}(\mathbf{u}, 1) \triangleq \bigcup_{\{1?a \rightarrow i!b \in Act \mid a \in \{a_0, \ldots, a_n\}\}} (1?a \rightarrow i!b) : \mathcal{L}(\mathbf{u}) \,.$$

```
1: global List readWL
2: global List writeWL
3: function List DoRead(Conf u, Channel ch)
4:     doReadRec(u, ch, true)
5:     return writeWL

6: procedure doReadRec(Conf u, Channel ch, bool na)  ▷ na indicates whether the
   current configuration is added to the writeWL
7:     if u[ch] = (U_1 + ... + U_n) · W  for some W then
8:         for i ∈ [1..n] do doReadRec(u with [ch] := U_i · W, ch, na)
9:     else if u[ch] = M* · W  for some W then
10:        u := Saturate(u, ch)
11:        writeWL := writeWL ∪ {u}
12:        u := (u with [ch] := W)
13:        doReadRec(u, ch, false)
14:    else if na ∧ (u[ch] = ε) then
15:        writeWL := writeWL ∪ {u}
16:    else if u[ch] = a · W  for some W then
17:        if na then writeWL := writeWL ∪ {u}
18:        U := Step(u, ch)
19:        for u' ∈ U do doReadRec(u', ch, true)
```

Fig. 10. DoRead algorithm for star topology and its supporting routines.

For example, given $Act = \{1?b \rightarrow 2!b, 1?c \rightarrow 3!c\}$ and $\mathbf{u} = \langle (b+c), \epsilon, \epsilon \rangle$, $\text{Step}(\mathbf{u}, 1)$ computes two piecewise configurations $\mathbf{u}'_1 = \langle \epsilon, b, \epsilon \rangle$ and $\mathbf{u}'_2 = \langle \epsilon, \epsilon, c \rangle$.

Detailed implementations of Saturate and Step for a set of actions Act on k channels are shown in Fig. 11.

Theorem 6.1 *Let* $\mathbf{u}$ *be a piecewise channel configuration,* Act *an action set with star topology and origin* o*, and* $\mathbf{U}'$ *the set returned by* $\text{DoRead}(\mathbf{u}, o)$*. Then,* $\mathcal{L}(\mathbf{U}') = (Act^* : \mathcal{L}(\mathbf{u}))$.

Proof. The proof is straightforward by the induction on the structure of $\mathbf{u}[1]$. □

Complexity Analysis. For a piecewise configuration $\mathbf{u}$, the depth of the recursion of DoRead is bounded by $h = |\mathbf{u}[o]|$ for the origin o. Inside each call, Saturate takes constant time and returns a single configuration; however, Step may return a set of configurations. In a k-channel system, the size of this set is bounded by $k-1$. Thus, the complexity of the DoRead algorithm is bounded by the number of internal nodes of a $(k-1)$-ary tree of height h. There are h such nodes for $k = 2$, and $((k-1)^{(h+1)} - 1)/(k-2)$ for $k > 2$.

Theorem 6.2 *Let* $\mathbf{u}$ *be a piecewise channel configuration,* Act *a set of actions with star topology on* k *channels with origin* o*, and* $h = |\mathbf{u}[o]|$*. Then, in the worst case, the running time of* $\text{DoRead}(\mathbf{u}, o)$ *is* $O(max(k^h, h))$.

6.2 Tree Topology

A set of actions Act has a *tree* (or, more generally, a *forest*) topology if and only if for all actions $i?a \rightarrow j!b$ and $i'?a' \rightarrow j'!b'$ in Act, $j = j' \Rightarrow i = i'$. That is, $CG(Act)$ is a finite union of trees (e.g. Fig. 9(b)). In this section, for presentation convenience, we consider $CG(Act)$ to be a tree and not a forest.

```
1: function Conf SATURATE(Conf u, Channel ch)
2:     let (Q, q^0, δ, F) = u[ch], M = {a | (q^0, a, q^0) ∈ δ}
3:     for all i ∈ ([1..k] \ {ch}) do                    ▷ k is the number of channels
4:         let M' = {b | (ch?a → i!b) ∈ Act ∧ a ∈ M}
5:         u'[i] := (u[i] · (M')*)
6:     return u'

7: function Set STEP(Conf u, Channel ch)
8:     U' := ∅
9:     let (Q, q^0, δ, F) = u[ch], M = {a ∈ Σ | ∃q', (q^0, a, q') ∈ δ ∧ q' ≠ q^0}
10:    for all a ∈ M ∧ i ∈ {j | ∃b, (ch?a → j!b) ∈ Act} do
11:        M' := {b | (ch?a → i!b) ∈ Act}
12:        u'[ch] := (Q, δ(q^0, a) \ {q^0}, δ, F)
13:        u'[i] := u[i] · M'
14:        U' := U' ∪ {u'}
15:    return U'
```

Fig. 11. STEP and SATURATE algorithms.

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, $\{Act_i\}$ partitions Act and each Act_i has a star topology with origin i. For example, consider the communication graph in Fig. 9(b): here, $Act_1 = \{1? \rightarrow 2!, 1? \rightarrow 3!, 1? \rightarrow 4!\}$, $Act_2 = \{2? \rightarrow 5!\}$, and $Act_3 = \{3? \rightarrow 6!, 3? \rightarrow 7!\}$.

This way, DOREAD can be used to compute $Act_i^* : \mathcal{L}(\mathbf{u})$ for any channel i and a piece-wise configuration $\mathbf{u}$. Furthermore, it can be applied iteratively to compute sequential composition of the partitions of Act. For example, computation of $(Act_1^* \cdot Act_2^*) : \mathcal{L}(\mathbf{u})$ is done by using DOREAD to first compute $\mathbf{U}'$ such that $\mathcal{L}(\mathbf{U}') = (Act_1^* : \mathcal{L}(\mathbf{u}))$, and using it again to compute $(Act_2^* : \mathcal{L}(\mathbf{U}'))$. In the following, we show how to extend this to the computation of the full limit language.

The graph $CG(Act)$ is acyclic and, therefore, induces a partial order $\preceq$ on channels (vertices of the graph). For channels i and j, $i \preceq j$ if and only if there exists a path from i to j in $CG(Act)$. Intuitively, channel i is less than channel j if the final content of j depends on the initial content of i. We say that channel j *depends* on channel i if $i \preceq j$, and that i and j are *interdependent* if either $i \preceq j$ or $j \preceq i$. Without loss of generality, we assume that the partial order $\preceq$ is extended to a total order and that the channels are numbered such that $i \leq j$ if $i \preceq j$. For example, channel 3 depends on channels 1 and 2, and 2 depends only on 1. The ordering and renaming of the channels can be done in time linear in the size of the CG.

If Act has a tree topology, every channel in $CG(Act)$ has at most one immediate predecessor. Thus, for every sequence $x \in Act^*$, there exists a sequence y such that: (i) y has the same actions as x, (ii) all reads of y are ordered, i.e., $y \in Act_1^* \cdot Act_2^* \cdot \cdots$, and (iii) if $(x : \mathbf{w}) \neq \emptyset$ for some $\mathbf{w}$, then $(y : \mathbf{w}) = (x : \mathbf{w})$. For example, for Act in Fig. 9(b), and $x = 1? \rightarrow 2!\ 1? \rightarrow 3!\ 2? \rightarrow 5!\ 1? \rightarrow 4!\ 3? \rightarrow 6!$, an equivalent sequence y is:

$$y = \underbrace{1? \rightarrow 2!\ 1? \rightarrow 3!\ 1? \rightarrow 4!}_{\in Act_1^*}\ \underbrace{2? \rightarrow 5!}_{\in Act_2^*}\ \underbrace{3? \rightarrow 6!}_{\in Act_3^*}.$$

Theorem 6.3 *Let Act be an action set on k channels such that $CG(Act)$ is a tree, and* $\mathbf{w}$

a channel configuration. Then,

$$((Act^* : \mathbf{w}) = ((Act_1^* \cdot \cdots \cdot Act_k^*) : \mathbf{w}))\,.$$

PROOF. We say that two action sequences x and y are enabled equivalent, written $x \equiv_e y$, if x and y behave identically on the input sequences that enable all actions of x. Formally,

$$x \equiv_e y \triangleq (\forall \mathbf{w},\ (x : \mathbf{w} \neq \emptyset) \Rightarrow (x : \mathbf{w} = y : \mathbf{w}))\,.$$

In the proof, we use the following equivalences (rules):

$$\begin{array}{lllll}
(1) & i?a\ j?b & \equiv_e & j?b\ i?a & (i \neq j) \\
(2) & i!a\ j!b & \equiv_e & j!b\ i!a & (i \neq j) \\
(3) & i!a\ j?b & \equiv_e & j?b\ i!a & \\
(4) & i?a\ j!b & \equiv_e & j!b\ i?a &
\end{array}$$

□

Let $p?c \rightarrow q!d$ and $i?a \rightarrow j!b$ be two conditional actions such that $i \preceq p$. Note that tree topology implies that $q \neq j$. We show that these two conditional actions can be commuted in any sequence of actions. The theorem follows by recursive application of this rule.

$$\begin{array}{ll}
 & p?c \rightarrow q!d\ i?a \rightarrow j!b \qquad \text{notation} \\
= & p?c\ q!d\ i?a\ j!b \qquad \text{using rule 3} \\
= & p?c\ i?a\ q!d\ j!b \qquad \text{using rule 2} \\
= & p?c\ i?a\ j!b\ q!d \qquad \text{using rule 1} \\
= & i?a\ p?c\ j!b\ q!d \qquad \text{using rule 4} \\
= & i?a\ j!b\ p?c\ q!d \qquad \text{notation} \\
= & i?a \rightarrow j!b\ p?c \rightarrow q!d
\end{array}$$

Thus, given a sequence of actions $x \in Act^*$, there exists an enabled equivalent sequence of actions $y \in Act_1^* \cdot \cdots \cdot Act_k^*$ where the conditional actions of x are ordered based on their read part. □

Theorem 6.3 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_i. We call this algorithm TREELIMIT (see Fig. 12). Since TREELIMIT proceeds through a finite total order of channels, it always terminates.

Theorem 6.4 *Let* $\mathbf{u}$ *be a piecewise configuration,* Act *an action set with tree topology, and* $\mathbf{U}'$ *the set of configurations returned by* TREELIMIT$(\mathbf{u})$*. Then,* $\mathcal{L}(\mathbf{U}') = (Act^* : \mathcal{L}(\mathbf{u}))$.

Complexity Analysis. Without loss of generality, we assume that $CG(Act)$ is an N-ary tree with M internal nodes and that the initial content of all the channels except the root is empty. Let $\mathbf{u}$ be a piecewise configuration, and $h = |\mathbf{u}|$. By Theorem 6.2, computation of $Act_i^* : \mathcal{L}(\mathbf{u})$ produces at most $\max(N^h, h)$ piecewise configurations, each of size at most h. TREELIMIT applies computation $Act_i^* : \mathcal{L}(\mathbf{u})$, M times, which produces at most $\max(N^{h \times M}, h^M)$ configurations.

Theorem 6.5 *Let* $\mathbf{u}$ *be a piecewise configuration,* Act *a set of actions with a tree topology of degree* N *and* M *internal nodes, and* $h = |\mathbf{u}[1]|$*. In the worst case, the running time of* TREELIMIT$(\mathbf{u})$ *is* $O(\max(N^{h \times M}, h^M))$.

```
1: function List TreeLimit(Conf u)
2:     readWL := u
3:     for ch = 1 to k do                          ▷ k is the number of channels
4:         writeWL := ∅
5:         for all u ∈ readWL do doRead(u, ch)
6:         readWL := readWL ∪ writeWL
7:     return readWL
```

Fig. 12. The TREELIMIT algorithm.

6.3 Inverted Tree Topology

A set of actions Act has an *inverted tree* topology if and only if for all conditional actions $i?a \to j!b$ and $i'?a' \to j'!b'$ in Act, $i = i' \Rightarrow j = j'$. That is, $CG(Act)$ is an inverted tree (e.g., see Fig. 9(c)).

In the inverted tree topology, a channel may depend on several pairwise independent channels. Therefore, Theorem 6.3 is no longer applicable. For example, let $Act = \{1?a \to 3!a, 2?b \to 3!b\}$, and $\mathbf{w} = \langle aa, bb, \epsilon\rangle$ be a configuration. The partial order on the channels induced by $CG(Act)$ is $\{1 \preceq 3, 2 \preceq 3\}$, with two obvious linearizations. A configuration $\langle \epsilon, \epsilon, abab\rangle$ is reachable from $\mathbf{w}$, but does not belong to either $((1?a \to 3!a)^*(2?b \to 3!b)^*) : \mathbf{w}$, or $((2?b \to 3!b)^*(1?a \to 3!a)^*) : \mathbf{w}$, 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 $CG(Act)$ 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 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 $\hat{l}_i$. Formally,

$$\text{ADDS}(i?a \to j!b, l) \triangleq \begin{cases} i?a \to \hat{l}_i!b & \text{if } j = l \\ i?a \to j!b & \text{otherwise.} \end{cases}$$

ADDS breaks dependencies between channels. Let $\widehat{Act} = \text{ADDS}(Act, l)$. If $CG(Act)$ is an inverted tree, then $CG(\widehat{Act})$ is a tree. For our running example, we introduce two shadow channels for channel 3; therefore, $\widehat{Act} = \{1?a \to \hat{3}_1!a, 2?b \to \hat{3}_2!b\}$. We use $\mathsf{S}(l)$ to denote the shadows of l.

Let $\mathbf{w}$ be a configuration, and $\hat{\mathbf{w}}$ be its extension to shadow channels. That is, $\hat{\mathbf{w}}[i] = \mathbf{w}[i]$ if $i \notin \mathsf{S}(l)$, and $\hat{\mathbf{w}}[i] = \epsilon$ otherwise. For example, if $\mathbf{w} = \langle aa, bb, b\rangle$, then by introducing two shadow channels for channel 3, $\hat{\mathbf{w}} = \langle aa, bb, b, \epsilon, \epsilon\rangle$.

The sets $(Act^* : \mathbf{w})$ and $((\widehat{Act})^* : \hat{\mathbf{w}})$ are closely related. Let $\mathbf{t} \in (x : \mathbf{w})$ be a configuration reachable from $\mathbf{w}$ by a sequence $x \in Act^*$, and $\hat{\mathbf{t}} \in (\text{ADDS}(x, l) : \hat{\mathbf{w}})$ be a configuration reachable from $\hat{\mathbf{w}}$, where ADDS is extended to sequences in an obvious way. ADDS only augments actions that write to l. Thus, $\mathbf{t}[i] = \hat{\mathbf{t}}[i]$ for any i that is different from l or its shadow channels $\mathsf{S}(l)$. By adding shadow channels for l, all the writes on l are redirected to its shadows and $\hat{\mathbf{t}}[l]$ is the initial content of l, hence, it is a prefix of $\mathbf{t}[l]$. Each shadow channel $\hat{l}_i$ keeps track of what was read from channel i and written to l, hence, $\hat{\mathbf{t}}[\hat{l}_i]$ is a subsequence of $\mathbf{t}[l]$.

```
1: function Conf MERGES(Conf u, Channel ch);
2: function CONF doWrite(Conf conf, Channel ch)
3:     return MERGES(u, ch)

4: function List MULTILIMIT(Conf u)
5:     readWL := u
6:     for ch = 1 to k do                          ▷ for every non-shadow channel ch
7:         writeWL := ∅
8:         for all u ∈ readWL do doRead(u, ch)
9:         if ch + 1 ≤ k then                      ▷ if ch is not the last channel
10:            for all u ∈ writeWL do
11:                readWL := readWL ∪ {doWrite(u, ch + 1)}
12:    return readWL
```

Fig. 13. MULTILIMIT algorithm and its supporting routines.

In our running example, $Act = \{1?a \to 3!a, 2?b \to 3!b\}$, $\widehat{Act} = \{1?a \to \hat{3}_1!a, 2?b \to \hat{3}_2!b\}$, $\mathbf{w} = \langle aa, bb, b\rangle$, and $\hat{\mathbf{w}} = \langle aa, bb, b, \epsilon, \epsilon\rangle$. Let x be a sequence in Act^*:

$$x \;=\; 1?a \to 3!a \;\; 2?b \to 3!b \;\; 1?a \to 3!a \;\; 2?b \to 3!b\,.$$

Then,

$$\text{ADDS}(x, 3) \;=\; 1?a \to \hat{3}_1!a \;\; 2?b \to \hat{3}_2!b \;\; 1?a \to \hat{3}_1!a \;\; 2?b \to \hat{3}_2!b\,.$$

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

$$\mathbf{t} \in \text{MERGES}(\hat{\mathbf{t}}, l) \Leftrightarrow (\forall i \neq l \wedge i \notin \mathsf{S}(l),\; \mathbf{t}[i] = \hat{\mathbf{t}}[i]) \wedge (\mathbf{t}[l] \in \mathcal{L}(\hat{\mathbf{t}}[l] \cdot ||_{j \in \mathsf{S}(l)} \{\hat{\mathbf{t}}[j]\})\,.$$

In the above example, let $\mathbf{t} = (x : \mathbf{w}) = \langle \epsilon, \epsilon, babab\rangle$ and $\hat{\mathbf{t}} = (\text{ADDS}(x, 3) : \hat{\mathbf{w}}) = \langle \epsilon, \epsilon, b, aa, bb\rangle$. Then, $\text{MERGES}(\hat{\mathbf{t}}, l) = \{\langle \epsilon, \epsilon, b \cdot (aa \,||\, bb)\rangle\}$ that is equal to

$$\{\langle \epsilon, \epsilon, baabb\rangle, \langle \epsilon, \epsilon, bbbaa\rangle, \langle \epsilon, \epsilon, babab\rangle, \langle \epsilon, \epsilon, bbaba\rangle, \langle \epsilon, \epsilon, babba\rangle, \langle \epsilon, \epsilon, bbaab\rangle\}\,.$$

As can be seen, $\mathbf{t} \in \text{MERGES}(\hat{\mathbf{t}}, l)$.

Theorem 6.6 *Let Act, $\widehat{Act}$, $\mathbf{w}$, $\hat{\mathbf{w}}$, and l be as above. Then,*

$$\mathbf{t} \in (Act^* : \mathbf{w}) \textit{ if and only if } \exists \hat{\mathbf{t}} \in ((\widehat{Act})^* : \hat{\mathbf{w}}),\; \mathbf{t} \in \text{MERGES}(\hat{\mathbf{t}}, l)\,.$$

PROOF. The proof follows directly from the definition of $\widehat{Act}$, $\hat{\mathbf{w}}$, and MERGES. By augmenting Act with shadow channels and extending $\mathbf{w}$ to $\hat{\mathbf{w}}$, all the writes to the channel with in-degree greater than or equal to 2 are forwarded to the corresponding shadow channels. Then, MERGES computes all possible configurations reachable by different interleavings of actions by shuffling the contents of the shadow channels. □

Both Theorem 6.6 and MERGES are easily lifted to piecewise configurations such that if $\mathbf{u}$ is a piecewise configuration, then $\text{MERGES}(\mathbf{u}, l)$ defines a piecewise configuration as well. This follows from the fact that piecewise languages are closed under concatenation and shuffle (see Proposition 2.5).

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

Fig. 14. (a) DAG communication topology, (b) Converting DAG to inverted tree using shadow channels.

shown in Fig. 13. 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 6.7 *Let* $\mathbf{u}$ *be a piecewise configuration,* Act *an action set with inverted tree topology, and* $\mathbf{U}$ *the set of configurations returned by* MULTILIMIT$(\mathbf{u})$. *Then,* $\mathcal{L}(\mathbf{U}) = (Act^* : \mathcal{L}(\mathbf{u}))$.

6.4 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. 9(d)). 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 $Act = \{1?a \rightarrow 3!a, 1?b \rightarrow 2!b, 2?b \rightarrow 3!b\}$ whose CG is shown in Fig. 14(a). 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 $\widehat{Act} = \{1?a \rightarrow \hat{3}_1!a, 1?b \rightarrow 2!b, 2?b \rightarrow \hat{3}_2!b\}$ (see Fig. 14(b)). By applying TREELIMIT to the resulting tree topology, we obtain two piecewise configurations $\{\langle a^*b^*, \epsilon, \epsilon, a^*, \epsilon\rangle, \langle b^*, b^*, \epsilon, a^*, b^*\rangle\}$. If we then proceed by merging the contents of the shadows of channel 3, as in the inverted tree topology, we obtain $\{\langle a^*b^*, \epsilon, a^*\rangle, \langle b^*, b^*, (a+b)^*\rangle\}$. 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 MULTILIMIT algorithm by modifying the shuffle used by MERGES (see Sec. 6.3) 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. The new algorithm, called MULTILIMITDAG, is shown in Figure 15.

For a system with k channels, each letter is associated with a k-tuple of indices from IDX, where IDX is $[-1..\infty)$. Intuitively, the jth 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. We write $\text{idx}(i, a)$ for the ith index of a. For example, $\text{idx}(2, a) = 4$ means that a was at some point at position 4 in channel 2, and $\text{idx}(3, a) = -1$ means that a was never in channel 3. We use $\text{ch}(a)$ to denote the latest channel that a was in. Formally, $\text{ch}(a) \triangleq \max\{i \mid \text{idx}(i, a) \neq -1\}$.

To keep track of the indices, several parts of the MULTILIMIT are modified as shown in Fig. 15. The `doReadRecDAG` procedure extends `doReadRec` (in Fig. 10) by accepting 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 13 and 19). SATURATEDAG and STEPDAG extend the corresponding algorithms in Fig. 11 by propagating and assigning indices (lines 23 and 30).

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., $i = ch(a) < ch(b)$. Furthermore, assume that a preceded b in channel i, i.e., $\text{idx}(i, a) < \text{idx}(i, b)$. 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

$$\forall p, q,\ (a = w(p) \wedge b = w(q) \wedge i = \text{ch}(a) \wedge \text{ch}(a) < \text{ch}(b) \wedge \text{idx}(i, a) < \text{idx}(i, b)) \Rightarrow p < q \tag{1}$$

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 ($ch(a) = 1$) which precedes 2 – the last channel of b ($ch(b) = 2$), thus, $ch(a) < ch(b)$. In the sequel, a preceded b in channel 1, i.e. $\text{idx}(1, a) < \text{idx}(1, b)$. Therefore, a must also precede b in channel 3.

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

Theorem 6.8 *The language WO, as defined in* (1)*, is piecewise.*

PROOF. To prove the theorem, we construct an automaton A_{WO} that recognizes WO. Then, we show that this automaton is a PO-FSA.

Let k be the number of channels. The state space of A_{WO} is $\text{IDX}[1..k][1..k]$. An interpretation of a state is

$$q[i][j] = p$$

if the automaton has seen a letter that its latest channel was i, and it was at some point at position p in channel j. The initial state q^0 is such that $\forall i, j,\ q^0[i][j] = -1$, and every state is accepting. The transition relation of A_{WO}, δ, is deterministic, and is defined as follows:

$$\begin{aligned}\delta(q, a, q') \Leftrightarrow\ & (\forall \text{ch}(a) < i \leq k,\ q[i][\text{ch}(a)] \leq \text{idx}(\text{ch}(a), a)) \wedge \\ & (\forall i, j,\ (i \neq \text{ch}(a) \Rightarrow q'[i][j] = q[i][j]) \wedge \\ & (i = \text{ch}(a) \Rightarrow q'[i][j] = \max(q[i][j], \text{idx}(j, a)))) .\end{aligned}$$

The first conjunct of δ ensures that the WO condition is satisfied, and the second updates the state. Let I denote the latest channel letter a was in. Intuitively, the automaton accepts letter a if and only if it has not seen a letter from any channel greater than I which was behind of letter a in I. The state of the automaton is updated if letter a has a greater position in any channel than any other letter that the automaton has seen in that channel.

The automaton A_{WO} is a PO-FSA, where the partial order $\preceq$ on states is:

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

□

In order to restrict MERGES to only include words that satisfy WO, we replace it with a function MERGEDAGS defined as follows. Let $\hat{\mathbf{t}}$ be a configuration reachable from an initial configuration extended with shadow channels, and l a non-shadow channel. Then, $\text{MERGEDAGS}(\hat{\mathbf{t}}, l) \triangleq \text{MERGES}(\hat{\mathbf{t}}, l) \cap \text{WO}$. Since WO is piecewise (by Theorem 6.8) and piecewise languages are closed under intersection (by Proposition 2.5), MERGEDAGS defines a piecewise configuration.

With this change, MULTILIMITDAG algorithm (see Fig. 15) computes the exact set of reachable configurations.

Theorem 6.9 *Let* $\mathbf{u}$ *be a piecewise configuration,* Act *a set of actions with DAG topology and* $\mathbf{U}'$ *a set of configurations returned by* $\text{MULTILIMIT}(\mathbf{u})$ *algorithm, where* MERGES *is replaced by* MERGEDAGS. *Then* $\mathcal{L}(\mathbf{U}') = (Act^* : \mathcal{L}(\mathbf{u}))$.

In this section, we presented 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 star and tree topologies we showed that the complexity of our algorithm is exponential in the size of the automaton representing the initial channel configuration. In the case of the inverted tree and DAG topologies, the complexity of the algorithms remains an open problem.

7. 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 [Brand and Zafiropulo 1983]. 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 article, is to identify practically useful subclasses of FIFO systems with decidable properties (e.g., [Pachl 1987; Finkel and Rosier 1988; Sistla and Zuck 1991; Klarlund and Trefler 2005]). The second approach is to look for efficient semi-algorithms that scale to realistic examples, but do not guarantee to always terminate (e.g., [Boigelot and Godefroid 1999; Boigelot et al. 1997; Gall et al. 2006]). 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 [Abdulla and Jonsson 1993; Cece et al. 1996; Abdulla et al. 1999]; however, calculating the set of all reachable states is impossible. The systems considered in this article are not lossy; all channels are perfect, i.e., they do not lose any message.

Pachl [Pachl 1987] proves that if the set of reachable channel configurations (the limit language) is recognizable then it is decidable to check for reachability of any given state. It was later shown in [Cece et al. 1996] that even though the reachability set might be recognizable, determining it may still be undecidable.

An appealing general model to distributed systems with channels is that of *FIFO nets*, which are formulated as Petri nets except that places are replaced by FIFO channels. The survey [Finkel and Rosier 1988] contains several decidability results, but they depend on the channel languages being *bounded*, i.e. a subset of some language $w_0^* \dots w_{n-1}^*$, where the w_i's are words.

Sistla and Zuck [Sistla and Zuck 1987; 1991; 1993] study the logic *restricted LTL* – Linear Temporal Logic restricted to only the *eventually* operator – and the corresponding problem of verification of FIFO systems with respect to this logic. They show that the class of languages expressed by restricted LTL is exactly characterized by a finite union of *restricted regular* sub-class of ω-regular languages. A language is restricted regular if it is of the form:

$$a_0 M_1^* a_1 M_2^* a_2 \cdots M_{m-1}^* a_{m-1} M_m^\omega$$

for some $M_1, \dots, M_m \subseteq \Sigma$ such that $a_0 \in M_1$ and for every i, $1 \leq i \leq m, a_i \in M_i \setminus M_{i+1}$. Furthermore, they show that for FIFO systems whose behavior can be described by a finite union of restricted regular languages, the verification problem with respect to restricted LTL is decidable and is in co-NP. At a first glance, *restricted regular languages* appear similar to *piecewise languages* used in this paper. Formally, they are different, for example, restricted regular languages are closed under complementation, but piecewise languages are not. However, the key distinction is in the alphabet considered when modeling FIFO systems with those languages. In the work of Sistla and Zuck, the alphabet consists of single read and write (or send and receive) actions (we call this Σ_{rw} in Section 3). In our work, the alphabet is further extended to include conditional actions, i.e., a read followed by a write (we call this Σ_{rwc} in Section 3). On one hand, conditional actions are crucial for modeling realistic communication protocols as we illustrated throughout the paper. On the other hand, FIFO systems with conditional actions can not be modeled with restricted regular languages. Whether the theory of Sistla and Zuck can be extended to handle conditional actions is an open problem.

Boigelot *et al.* [Boigelot and Godefroid 1999; Boigelot et al. 1997; Boigelot 1998] 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 [Boigelot and Godefroid 1999], automata-theoretic algorithms are given to calculate $f : L$ and $f^* : L$ for a *single* read, write, or conditional action f. Boigelot's Ph.D. thesis [Boigelot 1998] and [Boigelot et al. 1997] extend that to action sequences that preserve recognizability of channel contents. Our results are quite different and do not follow from the work on QDDs. The key difference between [Boigelot and Godefroid 1999; Boigelot et al. 1997] and our work is our focus on conditional actions that are often occur in practice in models of communication protocols. In particular, we focus on computing the limit language of the iteration of multiple conditional actions, i.e., Kleene closure of a sum or non-deterministic choice between conditional actions. This is a much harder problem since, in general, the limit language is not recognizable (i.e., regular) and the algorithms in [Boigelot 1998; Boigelot et al. 1997] do not apply. Furthermore,

we focus on computing the effects of application of a piecewise action language, which are not expressible in the QDD formalism. Therefore, none of the results in [Boigelot 1998; Boigelot et al. 1997] could be re-used and our proofs are constructed from the basic principles.

A special kind of regular expression, called *semilinear*, was introduced in [Finkel et al. 2003] as a symbolic presentation of regular, bounded languages describing channel contents. Unfortunately, a bounded language L has polynomial density: there are at most $P(n)$ words of size n for some polynomial P. This is a severe restriction. For example, it precludes sending a's and b's that are arbitrarily interspersed.

An approach for model-checking piecewise FIFO systems was studied in [Ghafari and Trefler 2006]. 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 [Ghafari and Trefler 2006] 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.

8. 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. We show that this problem is reducible to computing the limit language of a regular language of actions.

We consider single-channel and multi-channel FIFO systems separately. For the single-channel case, we show that the limit language is regular (piecewise) if the initial language is regular (piecewise). We present an 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 [Møller 2007].

For multi-channel systems, the limit language is not regular in generel. However, we show for it to be regular either we have to exclude conditional actions or consider only piecewise initial channel configurations. 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.

ACKNOWLEDGMENTS

Preliminary version of this work has appeared in [Klarlund and Trefler 2005; Ghafari et al. 2007].

REFERENCES

ABDULLA, P. A., ANNICHINI, A., AND BOUAJJANI, A. 1999. "Symbolic Verification of Lossy Channel Systems: Application to the Bounded Retransmission Protocol". In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS '99)*. LNCS, vol. 1579. 208–222.

ABDULLA, P. A. AND JONSSON, B. 1993. "Verifying Programs with Unreliable Channels". In *Proceedings of the 8th IEEE Symposium on Logic in Computer Science (LICS'93)*. 160–170.

BOIGELOT, B. 1998. Symbolic Method for Exploring Infinite States Spaces. Ph.D. thesis, Université de Liège.

BOIGELOT, B. AND GODEFROID, P. 1999. "Symbolic Verification of Communication Protocols with Infinite State Spaces Using QDDs". *Formal Methods in System Design 14*, 3, 237–255.

BOIGELOT, B., GODEFROID, P., WILLEMS, B., AND WOLPER, P. 1997. "The Power of QDDs". In *Proceedings of the 4th International Symposium on Static Analysis (SAS'97)*. LNCS, vol. 1302. 172 – 186.

BOND, G. W., IVANČIĆ, F., KLARLUND, N., AND TREFLER, R. 2001. "ECLIPSE Feature Logic Analysis". In *Proceedings of Second IP Telephony Workshop*.

BOUAJJANI, A., JONSSON, B., NILSSON, M., AND TOUILI, T. 2000. "Regular Model Checking". In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*. LNCS, vol. 1855. 403–418.

BOUAJJANI, A., MUSCHOLL, A., AND TOUILI, T. 2001. "Permutation Rewriting and Algorithmic Verification". In *Proceedings of the 16th IEEE Symposium on Logic in Computer Science (LICS'01)*. 399 – 408.

BRAND, D. AND ZAFIROPULO, P. 1983. "On Communicating Finite-State Machines". *Journal of the ACM 30*, 2, 323–342.

BRZOZOWSKI, A. AND SIMON, I. 1973. Characterization of locally testable events. *Discrete Mathematics 4*, 243–271.

CECE, G., FINKEL, A., AND IYER, S. P. 1996. "Unreliable Channels are Easier to Verify than Perfect Channels". *Information and Computation 124*, 1, 20–31.

FINKEL, A., IYER, S. P., AND SUTRE, G. 2003. "Well-abstracted transition systems: application to FIFO automata". *Information and Computation 181*, 1, 1–31.

FINKEL, A. AND ROSIER, L. 1988. "A Survey on the Decidability Questions for Classes of FIFO Nets". In *Advances in Petri Nets 1988*. LNCS, vol. 340. Springer, 106–132.

GALL, T. L., JEANNET, B., AND JÉRON, T. 2006. "Verification of Communication Protocols Using Abstract Interpretation of FIFO Queues". In *Proceedings of the 11th International Conference on Algebraic Methodology and Software Technology (AMAST'06)*. LNCS, vol. 4019. 204–219.

GHAFARI, N., GURFINKEL, A., KLARLUND, N., AND TREFLER, R. 2007. "Algorithmic Analysis of Piecewise FIFO Systems". In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design (FMCAD'07)*. IEEE Computer Society, 45–52.

GHAFARI, N. AND TREFLER, R. J. 2006. "Piecewise FIFO Channels Are Analyzable". In *Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*. LNCS, vol. 3855. 252–266.

HIGMAN, G. 1952. "Ordering by divisibility in abstract algebras". *Proceedings of the London Mathematical Society 2*, 7, 326–336.

IBM 2007. *Business Process Execution Language for Web Services (BPEL) Version 1.1*. IBM. Available from `http://www-128.ibm.com/developerworks/library/specification/ws-bpel`.

JACKSON, M. AND ZAVE, P. 1998. "Distributed Feature Composition: A Virtual Architecture for Telecommunications Services". *IEEE Transactions on Software Engineering 24*, 10, 831–847.

KLARLUND, N. AND TREFLER, R. 2005. "Regularity Results for FIFO Channels". *Electronic Notes in Theoretical Computer Science 128*, 6, 21–36.

KRUSKAL, J. B. 1972. "The Theory of Well-Quasi-Ordering: A Frequently Discovered Concept". *Journal of Combinatorial Theory, Series A 13*, 3, 297–305.

MØLLER, A. 2007. "An Automaton/Regular Expression Library for Java". Available at `http://www.brics.dk/automaton`.

PACHL, J. K. 1987. "Protocol Description and Analysis Based on a State Transition Model with Channel Expressions". In *Proceedings of the 7th International Conference on Protocol Specification, Testing and Verification*. 207–219.

SISTLA, A. P. AND ZUCK, L. D. 1991. "Automatic Temporal Verification of Buffer Systems". In *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91)*. 59–69.

SISTLA, P. AND ZUCK, L. 1987. "On the Eventuality Operation in Temporal Logic". In *Proceedings of the second symposium on Logics in Computer Science*. Boston, USA.

SISTLA, P. AND ZUCK, L. 1993. "Reasoning in a Restricted Temporal Logic". *Information and Computation 102*, 167–195.

WODEY, P., CAMARROQUE, G., BARAY, F., HERSEMEULE, R., AND COUSIN, J.-P. 2003. "LOTOS Code Generation for Model Checking of STBus Based SoC: the STBus interconnect". In *Proceedings of 1st ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2003)*.

YU, S. 1997. Regular languages. In *Handbook of language theory, Vol. I*, Rozenberg and Salomaa, Eds. Springer Verlag.

Received August 2010; accepted January 2011

```
1: global List readWL
2: global List writeWL
3: function List DoReadDAG(Conf u, Channel ch)
4:     doReadRecDAG(u, ch, 0, true)
5:     return writeWL

6: procedure doReadRecDAG(Conf u, Channel ch, int idx, bool na)  ▷ na indicates whether
   the current configuration is added to the writeWL
7:     if u[ch] = (U_1 + ... + U_n) · W  for some W then
8:         for i ∈ [1..n] do doReadRecDAG(u with [ch] := U_i · W, ch, idx, na)
9:     else if u[ch] = M* · W  for some W then
10:        u := SaturateDAG(u, ch, idx)
11:        writeWL = writeWL ∪ {u}
12:        u := (u with [ch] := W)
13:        doReadRecDAG(u, ch, idx + 1, false)
14:    else if na ∧ (u[ch] = ε) then
15:        writeWL := writeWL ∪ {u}
16:    else if u[ch] = a · W  for some W then
17:        if na then writeWL := writeWL ∪ {u}
18:        U := StepDAG(u, ch, idx)
19:        for u' ∈ U do doReadRecDAG(u', ch, idx + 1, true)

20: function Conf SaturateDAG(Conf u, Channel ch, int idx)
21:    let (Q, q^0, δ, F) = u[ch], M = {a | (q^0, a, q^0) ∈ δ}
22:    for all i ∈ ([1..k] \ {ch}) do                    ▷ k is the number of channels
23:        let M' = {b | (ch?a → i!b) ∈ Act ∧ a ∈ M ∧
               (∀j < ch, idx(j, b) = idx(j, a)) ∧ idx(ch, b) = idx}   ▷ Copy indices of a into b
24:        u'[i] := (u[i] · (M')*)
25:    return u'

26: function Set StepDAG(Conf u, Channel ch, int idx)
27:    U' := ∅
28:    let (Q, q^0, δ, F) = u[ch], M = {a ∈ Σ | ∃q', (q^0, a, q') ∈ δ ∧ q' ≠ q^0}
29:    for all a ∈ M ∧ i ∈ {j | ∃b, (ch?a → j!b) ∈ Act} do
30:        M' = {b | (ch?a → i!b) ∈ Act ∧
               (∀j < ch, idx(j, b) = idx(j, a)) ∧ idx(ch, b) = idx}   ▷ Copy indices of a into b
31:        u'[ch] := (Q, δ(q^0, a) \ {q^0}, δ, F)
32:        u'[i] := u[i] · M'
33:        U' := U' ∪ {u'}
34:    return U'

35: function Conf MergeDAGS(Conf u, Channel ch)
36: function Conf doWriteDAG(Conf conf, Channel ch)
37:    return MergeDAGS(u, ch)

38: function List MultiLimitDAG(Conf u)
39:    readWL := u
40:    for ch = 1 to k do                                ▷ for every non-shadow channel ch
41:        writeWL := ∅
42:        for all u ∈ readWL do doReadDAG(u, ch)
43:        if ch + 1 ≤ k then                            ▷ if ch is not the last channel
44:            for all u ∈ writeWL do
45:                readWL := readWL ∪ {doWriteDAG(u, ch + 1)}
46:    return readWL
```

Fig. 15. MultiLimitDAG algorithm for DAG topology and its supporting routines.