Decidable Topologies for Communicating Automata with FIFO and Bag Channels*

Lorenzo Clemente[1], Frédéric Herbreteau[2], and Grégoire Sutre[2]

[1] Université Libre de Bruxelles, Brussels, Belgium
[2] Univ. Bordeaux and CNRS, LaBRI, UMR 5800, Talence, France

Abstract. We study the reachability problem for networks of finite-state automata communicating over unbounded perfect channels. We consider communication topologies comprising both ordinary FIFO channels and *bag channels*, i.e., channels where messages can be freely reordered. It is well-known that when only FIFO channels are considered, the reachability problem is decidable if, and only if, there is no undirected cycle in the topology. On the other side, when only bag channels are allowed, the reachability problem is decidable for any topology by a simple reduction to Petri nets. In this paper, we study the more complex case where the topology contains *both* FIFO and bag channels, and we provide a complete characterisation of the decidable topologies in this generalised setting.

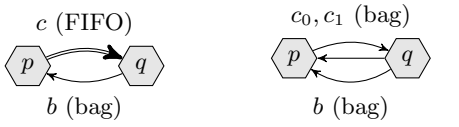
1 Introduction

Communicating finite-state automata (CFSA) are a fundamental model of computation where concurrent processes exchange messages over unbounded, reliable channels. Depending on the context, messages are delivered in the order they were sent (FIFO channel), or in any order (bag channel). On the one hand, FIFO channels can be used, e.g., to model communications through TCP sockets, as TCP preserves the order of messages. It is well-known that the reachability problem for CFSA with only FIFO channels is undecidable [5,18]. This problem becomes decidable when the communication topology is required to be acyclic [18,14]. Many other decidable subclasses and under/over-approximation techniques have been considered in the literature [3,2,7,6,14,8,4,11,10,1]. On the other hand, bag channels can be used, e.g., to model asynchronous procedure calls [19,12,9]. Indeed, libraries supporting asynchronous programming do not guarantee, in general, that procedures are executed in the order they were asynchronously called. The reachability problem for CFSA with only bag channels is decidable (without any further restriction), by an immediate reduction to reachability in Petri nets, which is known to be decidable [16,13,15].

Contributions. While the reachability problem is well-understood for communication topologies of just FIFO or bag channels, we go one step forward and we

* This work was partially supported by the ANR project Vacsim (ANR-11-INSE-004).

P. Baldan and D. Gorla (Eds.): CONCUR 2014, LNCS 8704, pp. 281–296, 2014.



(a) Opposite channels **(b)** Synchronising c

Fig. 1. Opposite channels

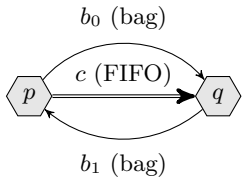


Fig. 2. Undecidable topology

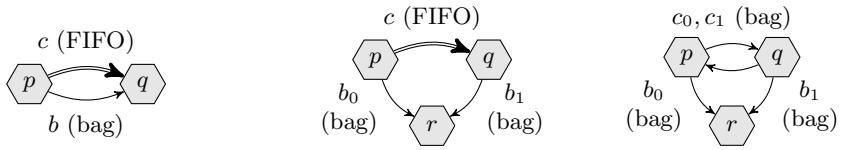
study topologies comprising *both* FIFO *and* bag channels. Our main result is a complete characterisation of decidable topologies of FIFO and bag channels, and a detailed complexity analysis in the decidable case. As a consequence of our results, we show that certain non-trivial cycles comprising FIFO and bag channels can be allowed while preserving decidability.

In addition to being the right model in some contexts, bag channels also provide a non-trivial over-approximation of FIFO channels. Indeed, it is always possible to over-approximate the reachability set by turning *all* channels into bag channels. Thanks to our characterisation, a much finer analysis may be obtained, in practice, by selectively over-approximating only *some* of the FIFO channels.

Preview. Let us illustrate our main techniques with some example. While the topology in Fig. 1a is undecidable when all channels are FIFO, it becomes decidable when b is bag. Indeed, the FIFO channel c can be "made synchronous" by forcing receptions to occur right after transmissions. This, in turn, can be implemented by replacing c with two opposite bag channels c_0, c_1 implementing a simple rendezvous protocol. We thus obtain the topology in Fig. 1b, which is decidable since it contains only bag channels.

A more difficult case is the one in Fig. 3a. As above, reachability is undecidable if all channels are FIFO, but it becomes decidable when one channel, say b, is bag. However, the correctness argument is more involved here, since, unlike in the previous example, channel c cannot be made synchronous. The problem is that making c synchronous requires rescheduling the actions of the receiver q to occur earlier. But this is not possible since q might try to read on the other channel b, which could be empty. The crucial observation is that we can always schedule all actions of p to occur before all actions of q. Therefore, in this topology, the order between transmissions and receptions can be relaxed. The only property that matters is that the string of messages which is received is the same as the one which is sent. We can thus *split* the bag channel b into two bag channels b_0 and b_1 (see Fig. 3b), where q's potentially blocking receptions on b are replaced with non-blocking transmissions on b_1. The new process r just matches incoming messages on b_0 and b_1. In the new topology, c can be made synchronous, and we proceed as above to obtain the decidable topology in Fig. 3c.

Finally, we also have undecidable topologies which mix in a non-trivial way FIFO and bag channels. For example, consider the one in Fig. 2, where c is FIFO and b_0 and b_1 are bags. This topology is undecidable, even when b_0 and b_1 are *unary* bag channels (i.e., the message alphabet is a singleton). The idea is to use the two bag channels b_0 and b_1 to implement a synchronisation protocol



(a) Parallel channels b and c **(b)** Splitting b **(c)** Synchronising c

Fig. 3. Two parallel channels

between processes p and q. This protocol is then used by p to decide which message is to be received by q from c, thus simulating a channel machine (which has undecidable reachability).

Outline. The rest of the paper is organised as follows. We start with preliminaries in Sec. 2. In Sec. 3 we show techniques for synchronising and splitting channels (decidability). In Sec. 4 we explain how unary bag channels can be used to simulate rendezvous synchronisation (undecidability). In Sec. 5 we present our characterisation of decidable topologies and study the complexity of the decidable instances. Finally, in Sec. 6 we compare our techniques with the work of Chambart and Schnoebelen [8], and in Sec. 7 we end with directions for future work.

2 Preliminaries

A *labelled transition system* (LTS for short) is a tuple $\mathcal{A} = \langle S, S_I, S_F, A, \to \rangle$ where S is a set of *states* with *initial states* $S_I \subseteq S$ and *final states* $S_F \subseteq S$, A is a finite set of *actions*, and $\to \subseteq S \times A \times S$ is a *labelled transition relation*. For simplicity, we write $s \xrightarrow{a} s'$ in place of $(s, a, s') \in \to$. An LTS is called *finite* when its set of states is finite. A *run* in $\mathcal{A}$ is an alternating sequence $(s_0, a_1, s_1, \ldots, a_n, s_n)$ of states $s_i \in S$ and actions $a_i \in A$, with $n \geq 0$, such that $s_{i-1} \xrightarrow{a_i} s_i$ for all $1 \leq i \leq n$. The natural number n, which may be zero, is called the *length* of the run. An *accepting run* is a run starting in an initial state (i.e., with $s_0 \in S_I$) and ending in a final state (i.e., with $s_n \in S_F$). Given two runs $\sigma = (s_0, a_1, s_1, \ldots, a_m, s_m)$ and $\tau = (t_0, b_1, t_1, \ldots, b_n, t_n)$ such that $s_m = t_0$, their *join* is the run $\sigma \cdot \tau = (s_0, a_1, s_1, \ldots, a_m, s_m, b_1, t_1, \ldots, b_n, t_n)$.

Topologies. We consider systems that are composed of several processes communicating through the asynchronous exchange of messages. Communications rely on point-to-point FIFO channels between processes. In our setting, each channel is equipped with a message alphabet that specifies the set of messages that can be conveyed over the channel. To simplify the presentation, we assume a special message, written $\mathtt{1}$, that is always in the message alphabet. Formally, a communication *topology* is a tuple $\mathcal{T} = \langle P, C, M, \mathsf{src}, \mathsf{dst}, \mathsf{msg} \rangle$, where P is a finite set of *processes*, C is a finite set of *channels*, M is a finite set of *messages* containing the special message $\mathtt{1}$, $\mathsf{src} : C \to P$ and $\mathsf{dst} : C \to P$ are mappings assigning to each channel a *source* and a *destination* process, and $\mathsf{msg} : C \to \{N \subseteq M \mid \mathtt{1} \in N\}$

is a mapping assigning to each channel its *message* alphabet. For convenience, we assume that the sets P, C and M are pairwise disjoint. A channel $c \in C$ is called *unary* when $\mathsf{msg}(c)$ is a singleton, i.e., when $\mathsf{msg}(c) = \{1\}$.

The following graph-theoretic concepts and notations[1] will be used throughout the paper. Consider a topology $\mathcal{T} = \langle P, C, M, \mathsf{src}, \mathsf{dst}, \mathsf{msg}\rangle$. For each channel $c \in C$, we let $\overset{c}{\Longrightarrow}$ denote the binary relation on P defined by $p \overset{c}{\Longrightarrow} q$ if $p = \mathsf{src}(c)$ and $q = \mathsf{dst}(c)$. The inverse of $\overset{c}{\Longrightarrow}$ is written $\overset{c}{\Longleftarrow}$. A *directed walk* in $\mathcal{T}$ is an alternating sequence $(p_0, c_1, p_1, \ldots, c_n, p_n)$ of processes $p_i \in P$ and channels $c_i \in C$, with $n \geq 0$, such that $p_{i-1} \overset{c_i}{\Longrightarrow} p_i$ for all $1 \leq i \leq n$. The natural number n, which may be zero, is called the *length* of the directed walk. To improve readability, directed walks will usually be written $p_0 \overset{c_1}{\Longrightarrow} p_1 \cdots \overset{c_n}{\Longrightarrow} p_n$. A directed walk is said to be *closed* when it starts and ends in the same process (i.e., when $p_0 = p_n$). A *directed path* is a directed walk in which all channels are pairwise distinct, and all processes—except, possibly, the first and last ones—are pairwise distinct. The notation $p \overset{*}{\Longrightarrow} q$ means that there is a directed walk—or, equivalently, there is a directed path—from p to q (i.e., with $p_0 = p$ and $p_n = q$). A *directed cycle* is a closed directed path of non-zero length.

We also need undirected variants of the above notions. For each channel $c \in C$, we let $\overset{c}{=\!=}$ denote the binary relation on P defined by $p \overset{c}{=\!=} q$ if $\{p, q\} = \{\mathsf{src}(c), \mathsf{dst}(c)\}$. Observe that $p \overset{c}{=\!=} q$ if, and only if, $p \overset{c}{\Longrightarrow} q$ or $p \overset{c}{\Longleftarrow} q$. The notions of *undirected walk*, *undirected path* and *undirected cycle* are defined as the directed ones, except that $\overset{c}{\Longrightarrow}$ is replaced by $\overset{c}{=\!=}$.

Communicating Processes. Given a topology $\mathcal{T} = \langle P, C, M, \mathsf{src}, \mathsf{dst}, \mathsf{msg}\rangle$, the set of possible *communication actions* for a process $p \in P$, written A^p_{com}, is the union of the set $\{c!m \mid c \in C \wedge \mathsf{src}(c) = p \wedge m \in \mathsf{msg}(c)\}$ of its *transmission actions* and of the set $\{c?m \mid c \in C \wedge \mathsf{dst}(c) = p \wedge m \in \mathsf{msg}(c)\}$ of its *reception actions*. The set of all communication actions is $A_{\mathrm{com}} = \bigcup_{p \in P} A^p_{\mathrm{com}}$. Actions not in A_{com} are called *internal actions*.

Definition 2.1. *A* system of communicating processes *is a pair* $\mathcal{S} = \langle \mathcal{T}, \{\mathcal{A}^p\}_{p \in P}\rangle$ *where* $\mathcal{T} = \langle P, \ldots\rangle$ *is a topology, and, for each* $p \in P$, $\mathcal{A}^p = \langle S^p, S^p_I, S^p_F, A^p, \rightarrow^p\rangle$ *is a finite labelled transition system such that* $A^p \cap A_{\mathrm{com}} = A^p_{\mathrm{com}}$. *For convenience, we assume*[2] *that the sets of actions* A^p, *with* $p \in P$, *are pairwise disjoint.*

We give the operational *semantics* of a system of communicating processes $\mathcal{S}$ as a global labelled transition system $[\![\mathcal{S}]\!] = \langle X, X_I, X_F, A, \rightarrow\rangle$. States of $\mathcal{S}$ are called *configurations* to prevent confusion with those of $\mathcal{A}^p$. A configuration of $[\![\mathcal{S}]\!]$ is pair $x = (\boldsymbol{s}, \boldsymbol{w})$ where $\boldsymbol{s}$ maps each process p to a state in S^p, and $\boldsymbol{w}$ maps each channel c to a word over its message alphabet $\mathsf{msg}(c)$. Formally, $X = (\prod_{p \in P} S^p) \times (\prod_{c \in C} \mathsf{msg}(c)^*)$. A configuration is initial (resp. final) when

[1] In this paper, we use contemporary graph terminology (see, e.g., [20]). For instance, the term *walk* is used for "paths" that may repeat channels and/or processes.

[2] This assumption is not restrictive as it only concerns internal actions. Indeed, the sets A^p_{com}, with $p \in P$, are already pairwise disjoint by definition.

each process is in its initial state (resp. final state) and all channels are empty. Formally, $X_I = (\prod_{p\in P} S_I^p) \times \{\boldsymbol{\varepsilon}\}$ and $X_F = (\prod_{p\in P} S_F^p) \times \{\boldsymbol{\varepsilon}\}$, where $\boldsymbol{\varepsilon}$ maps each channel $c \in C$ to the empty word ε. The set of actions A of $\mathcal{S}$ is given by $A = \bigcup_{p\in P} A^p$. Observe that $\{A^p\}_{p\in P}$ is a partition of A. We define the transition relation $\rightarrow$ of $[\![\mathcal{S}]\!]$ to be the set of all triples (x_1, a, x_2), where $x_1 = (\boldsymbol{s}_1, \boldsymbol{w}_1)$ and $x_2 = (\boldsymbol{s}_2, \boldsymbol{w}_2)$ are configurations, such that, for some process $p \in P$, the following conditions are satisfied:

- $s_1^p \xrightarrow{a} s_2^p$ is a transition in $\mathcal{A}^p$, and $s_1^q = s_2^q$ for all other processes $q \in P \setminus \{p\}$.
- If a is an internal action, then $\boldsymbol{w}_1 = \boldsymbol{w}_2$.
- If $a = c!m$, then $w_2^c = w_1^c \cdot m$ and $w_2^d = w_1^d$ for all other channels $d \in C \setminus \{c\}$.
- If $a = c?m$, then $w_1^c = m \cdot w_2^c$ and $w_2^d = w_1^d$ for all other channels $d \in C \setminus \{c\}$.

So, in a transition $x_1 \xrightarrow{a} x_2$, exactly one process p moves (namely, the unique $p \in P$ such that $a \in A^p$), and the others stay put. The channels are updated according to the action a that is performed by the transition. Given a process $p \in P$, a *move* of p is any transition $x_1 \xrightarrow{a} x_2$ such that $a \in A^p$. Following [11], we define the *causal-equivalence* relation $\sim$ over runs as the least congruence, with respect to join, such that $(x_1, a, x_2, b, x_3) \sim (x_1, b, x_2', a, x_3)$ whenever a, b are actions of distinct processes. Informally, two runs are causal-equivalent if they can be transformed one into the other by iteratively commuting adjacent moves that (i) are *not* from the same process and (ii) do *not* form a "matching send/receive pair". It is readily seen that causal-equivalent runs necessarily start in the same configuration and end in the same configuration.

Statement of the Problem. Given a topology $\mathcal{T}$, the *reachability problem* for systems of communicating processes with topology $\mathcal{T}$, denoted by $\textsc{Reach}(\mathcal{T})$, is defined as follows:

Input: a system of communicating processes $\mathcal{S}$ with topology $\mathcal{T}$,
Output: whether there exists an accepting run in $[\![\mathcal{S}]\!]$.

Observe that we require all channels to be empty at the end of an accepting run. Also note that $\textsc{Reach}(\mathcal{T})$ is parametrised by a topology $\mathcal{T}$. The main result of this paper is a characterisation of the topologies $\mathcal{T}$ for which $\textsc{Reach}(\mathcal{T})$ is decidable. Our techniques are based on topological transformations that induce reductions between the associated reachability problems. We let $\preceq_m$ denote the *many-one reducibility* relation between decision problems. For example, $\textsc{Reach}(\mathcal{T}) \preceq_m \textsc{Reach}(\mathcal{U})$ when $\mathcal{T}$ is obtained from $\mathcal{U}$ by removing channels.

When only unary channels are present, the reachability problem is decidable by an immediate reduction to reachability in Petri nets.

Theorem 2.2 ([16,13,15]). *If $\mathcal{T}$ is a topology with only unary channels, then $\textsc{Reach}(\mathcal{T})$ is decidable.*

On the other hand, when the topology contains only non-unary channels the following characterisation for $\textsc{Reach}(\mathcal{T})$ is well-known.

Theorem 2.3 ([18,14]). *Given a topology $\mathcal{T}$ with no unary channel,* Reach($\mathcal{T}$) *is decidable if, and only if, $\mathcal{T}$ has no undirected cycle.*

In Sec. 5, we refine the latter condition to account for topologies with both unary and non-unary channels (see Theorem 5.3). We further generalise it in Sec. 6 to a more general setting comprising both FIFO and bag channels.

3 Synchronising and Splitting Channels

A useful technique in the analysis of communicating processes is to transform asynchronous communications into synchronous ones, without compromising the behaviour of the system. To this end, we use the notion of synchronous runs from [8,11]. Formally, a run $((\boldsymbol{s}_0, w_0), a_1, (\boldsymbol{s}_1, w_1), \ldots, a_n, (\boldsymbol{s}_n, w_n))$ is *synchronous* for a given channel c if $w_0^c = w_n^c = \varepsilon$ and $w_i^c = \varepsilon \vee w_{i+1}^c = \varepsilon$ for all $0 < i < n$. Intuitively, this means that each transmission on c is immediately followed by its matching reception, i.e., communication over c behaves like rendezvous synchronisation [17]. It is well-known that in a polyforest topology (i.e., with no undirected cycle), every run can be reordered to have *all* channels synchronous, and reachability can be solved by exploring the resulting finite transition system [18,14,8,11]. Since we are interested in analysing more complex topologies where not all channels can be simultaneously made synchronous in general, we need to consider channels individually rather than globally.

Synchronising Essential Channels. Whether a channel can always be made synchronous (by reordering moves in runs) is a semantic condition that depends on the complex behaviour of the whole system. In fact, this condition is an undecidable problem in general (by an easy reduction from the reachability problem). Therefore, we are interested in syntactic conditions that are sufficient for a channel to be made synchronous. One such condition is that of *essential channel* [8], which is a structural condition depending only on the topology.

Definition 3.1 ([8]). *A channel c is* essential *if all directed paths from* $\mathsf{src}(c)$ *to* $\mathsf{dst}(c)$ *contain c. In particular,* $\mathsf{src}(c) \neq \mathsf{dst}(c)$.

Lemma 3.2 ([8]). *If c is an essential channel, then every run that starts and ends with c empty is causal-equivalent to a run that is synchronous for c.*

Thus, we can replace asynchronous communications on an essential channel by synchronous ones. We loosely implement the latter through a topological transformation that replaces an essential channel c, with source p and destination q, by one unary channel c_m from p to q for each message $m \in \mathsf{msg}(c)$, as well as one unary channel c_{ack} back from q to p. These unary channels are enough to simulate synchronous communications over c. Each message m conveyed over c is placed in the corresponding unary channel c_m instead. After each transmission on c_m, the process p waits for an acknowledgement from c_{ack}. Conversely, q notifies p via c_{ack} after each reception from c_m. While it applies to arbitrary essential channels, this topological transformation is only useful for non-unary ones.

Definition 3.3. *Given a topology* $\mathcal{T} = \langle P, C, M, \mathsf{src}, \mathsf{dst}, \mathsf{msg}\rangle$ *and a channel* $c \in C$*, the* synchronisation *of* c *in* $\mathcal{T}$ *is the topology* $\mathcal{U} = \langle P, C', M, \mathsf{src}', \mathsf{dst}', \mathsf{msg}'\rangle$ *defined by* $C' = (C \setminus \{c\}) \cup \{c_m \mid m \in \mathsf{msg}(c)\} \cup \{c_{\text{ack}}\}$ *and*

$$(\mathsf{src}'(d), \mathsf{dst}'(d), \mathsf{msg}'(d)) \;=\; \begin{cases} (\mathsf{src}(d), \mathsf{dst}(d), \mathsf{msg}(d)) & \text{if } \; d \in C \setminus \{c\} \\ (\mathsf{src}(c), \mathsf{dst}(c), \{1\}) & \text{if } \; d = c_m \\ (\mathsf{dst}(c), \mathsf{src}(c), \{1\}) & \text{if } \; d = c_{\text{ack}} \end{cases}$$

where c_m*, for* $m \in \mathsf{msg}(c)$*, and* c_{ack} *are new channels that are not in* $P \cup C \cup M$*.*

Proposition 3.4 (Synchronisation). *If* c *is an essential channel in* $\mathcal{T}$*, then* $\textsc{Reach}(\mathcal{T}) \preceq_m \textsc{Reach}(\mathcal{U})$ *where* $\mathcal{U}$ *results from the synchronisation of* c *in* $\mathcal{T}$*.*

Remark 3.5. An essential channel could, alternatively, be removed by merging its endpoints (see [8]). Instead, our synchronisation construction replaces an essential channel by a collection of new unary channels. While either technique could be used for decidability (see Subsec. 5.2), synchronisation yields simpler proofs and avoids taking the product of LTSes at the process level, which cirvumvents an immediate exponential blow-up in our reduction to reachability in Petri nets (see Subsec. 5.3). From a practical viewpoint, the new unary channels are 1-bounded by construction; analyzers for Petri nets could take advantage of this fact.

Splitting Irreversible Channels. According to Proposition 3.4 above, a topology containing an essential non-unary channel c can always be simplified by synchronising c. The resulting topology is simpler in the sense that it contains one less non-unary channel. However, there are situations where a channel is not essential, and thus it cannot be synchronised in general, but it can be made essential after a small modification. We have shown one such simple case on Fig. 3a, where the channel c is not essential but it can be made so by *splitting*[3] the other channel $p \overset{b}{\Longrightarrow} q$ into two channels $p \overset{b_0}{\Longrightarrow} r$ and $q \overset{b_1}{\Longrightarrow} r$ for a new process r, see Fig. 3b. Here, r is a new process that simply matches messages received from b_0 and b_1; receptions on b become transmissions on b_1. Clearly, the new system with the split topology has at least the same runs as the original system. Moreover, in this case, the converse holds as well, since we can always schedule all actions of p before any action of q, thus causality between transmissions and receptions can be relaxed. Formally, the splitting operation is defined as follows.

Definition 3.6. *Given a topology* $\mathcal{T} = \langle P, C, M, \mathsf{src}, \mathsf{dst}, \mathsf{msg}\rangle$ *and a channel* $c \in C$*, the* split *of* c *in* $\mathcal{T}$ *is the topology* $\mathcal{U} = \langle P', C', M, \mathsf{src}', \mathsf{dst}', \mathsf{msg}'\rangle$ *defined by* $P' = P \cup \{r\}$*,* $C' = (C \setminus \{c\}) \cup \{c_0, c_1\}$*, and*

$$(\mathsf{src}'(d), \mathsf{dst}'(d), \mathsf{msg}'(d)) \;=\; \begin{cases} (\mathsf{src}(d), \mathsf{dst}(d), \mathsf{msg}(d)) & \text{if } \; d \in C \setminus \{c\} \\ (\mathsf{src}(c), r, \mathsf{msg}(c)) & \text{if } \; d = c_0 \\ (\mathsf{dst}(c), r, \mathsf{msg}(c)) & \text{if } \; d = c_1 \end{cases}$$

where r *is a new process and* c_0, c_1 *are new channels that are not in* $P \cup C \cup M$*.*

[3] Despite having similar names, this splitting notion and the splitting technique of [8] have little in common (see Sec. 6).

To justify splitting, we introduce the notion of causal run. Intuitively, in a run which is causal for a process p, only those processes that can "transitively" send messages to p may be scheduled before p.

Definition 3.7. *A run* $(x_0, a_1, x_1, \ldots, a_n, x_n)$ *is* causal *for a given process* p *if* $q \xrightarrow{*} p$ *for every process* q *such that* $a_i \in A^q$ *and* $a_j \in A^p$ *for some* $1 \leq i < j \leq n$.

Lemma 3.8. *Given a process* p*, every run is causal-equivalent to a run that is causal for* p.

Recall that the idea behind splitting is to relax the causality between transmissions and receptions. This does not introduce "spurious" runs provided that every run can be reordered to have all actions of the receiver after the last action of the sender. A sufficient condition is given by the notion of *irreversible channel*.

Definition 3.9. *A channel* c *is* reversible *if there is a directed path from its destination* $\mathsf{dst}(c)$ *to its source* $\mathsf{src}(c)$*. A channel is* irreversible *if it is not reversible.*

The following proposition states that the reachability problem for a given topology $\mathcal{T}$ can be reduced to the reachability problem for the topology obtained from $\mathcal{T}$ by splitting c. As we will see in Sec. 5, splitting will be the first of a series of reductions for decidable topologies.

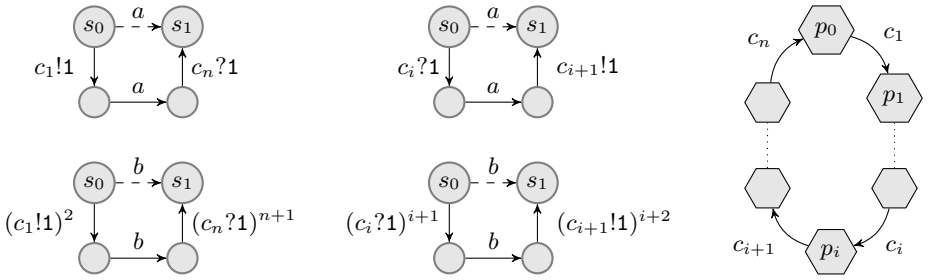
Proposition 3.10 (Split). *If* c *is an irreversible channel in* $\mathcal{T}$*, then it holds that* $\textsc{Reach}(\mathcal{T}) \preceq_m \textsc{Reach}(\mathcal{U})$ *where* $\mathcal{U}$ *results from the split of* c *in* $\mathcal{T}$.

4 The Power of Unary Channels

Let u be a process in a topology $\mathcal{T}$, and let $\mathcal{U}$ be a topology which is the same as $\mathcal{T}$ except that u is expanded into a strongly-connected sub-topology. In this section, we show that the behaviour of u in $\mathcal{T}$ can be *distributed* over its expansion in $\mathcal{U}$. We achieve this by demonstrating how processes in a strongly-connected sub-topology can simulate global rendezvous synchronisation over a finite alphabet of shared actions, which allows them to synchronise with each others and to step-wise simulate the behaviour of u. This technique works even when distributing behaviour over unary channels only, and it will be used to show the undecidability part of our characterisation (see Subsec. 5.1).

To formally define how a process is expanded into a sub-topology, it is more convenient to describe how to *fuse* a set of channels D. Intuitively, the fusion of D in a given topology is the topology obtained by merging together, in a single process, all endpoints of channels in D, by removing D, and by redirecting the other channels in the natural way.

Definition 4.1. *Given a topology* $\mathcal{T} = \langle P, C, M, \mathsf{src}, \mathsf{dst}, \mathsf{msg}\rangle$ *and a set of channels* $D \subseteq C$*, the* fusion *of* D *in* $\mathcal{T}$ *is the topology* $\mathcal{U} = \langle P', C', M, \mathsf{src}', \mathsf{dst}', \mathsf{msg}'\rangle$ *defined by* $P' = (P \setminus \{\mathsf{src}(c), \mathsf{dst}(c) \mid c \in D\}) \cup \{u\}$ *where* u *is a new process that is not in* $P \cup C \cup M$*,* $C' = C \setminus D$*, and, for every* $c \in C'$*,* $\mathsf{msg}'(c) = \mathsf{msg}(c)$*,* $\mathsf{src}'(c) = \mathsf{src}(c)$ *if* $\mathsf{src}(c) \in P'$ *and* $\mathsf{src}'(c) = u$ *otherwise, and similarly for* dst'.



(a) Process p_0 **(b)** Process $p_i \neq p_0$ **(c)** Topology

Fig. 4. Synchronisation protocol for a simple directed cycle

This section shows that fusing a strongly-connected sub-topology makes the reachability problem easier. We first deal with the simple case of directed cycles. The *support* of a directed walk/cycle is the set of channels that it visits.

Lemma 4.2. *If D is the support of a directed cycle in a topology $\mathcal{T}$ then* $\textsc{Reach}(\mathcal{U}) \preceq_m \textsc{Reach}(\mathcal{T})$ *where $\mathcal{U}$ results from the fusion of D in $\mathcal{T}$.*

Proof. Consider a directed cycle $p_0 \xRightarrow{c_1} p_1 \xRightarrow{c_2} \cdots \xRightarrow{c_n} p_n = p_0$ in $\mathcal{T}$ such that $D = \{c_1, \ldots, c_n\}$. This directed cycle is depicted in Fig. 4c. Denote by $\mathcal{U}$ the topology that results from the fusion of D in $\mathcal{T}$, and let u be the process in $\mathcal{U}$ that corresponds to the merging of all endpoints of D. Consider a system of communicating processes $\mathcal{S}$ with topology $\mathcal{U}$. We construct a new system $\mathcal{R}$ with topology $\mathcal{T}$ that simulates $\mathcal{S}$ by "distributing" the behaviour of the process u over $p_1, \ldots, p_n$. All other processes are left unchanged.

As a first step, let us assume that the processes $p_1, \ldots, p_n$ can perform multi-way rendezvous synchronisation over a finite alphabet of actions Σ. By *multi-way*, we mean that each time some process p_i performs an action a in Σ, then in fact *all* processes $p_1, \ldots, p_n$ perform the action a at the same time. For brevity, we will omit the "multi-way" qualifier from now on. It is easily shown that rendezvous synchronisation, even over a binary alphabet, allows $p_1, \ldots, p_n$ to coordinate in $\mathcal{T}$ and simulate the behaviour of u in $\mathcal{U}$. This way, we obtain from $\mathcal{S}$ a new system $\mathcal{S}'$, with topology $\mathcal{T}$, such that $[\![\mathcal{S}]\!]$ has an accepting run if, and only if, $[\![\mathcal{S}']\!]$ does. Moreover, $\mathcal{S}'$ does not use any channel in D.

As a second step, we explain how rendezvous synchronisation between $p_1, \ldots, p_n$ over a binary alphabet, say $\{a, b\}$, can be achieved through communications on the channels in D. In our simulation, rendezvous synchronisations are initiated by p_0, and then propagated along the directed cycle $p_0 \xRightarrow{c_1} p_1 \xRightarrow{c_2} \cdots \xRightarrow{c_n} p_n$ back to p_0. The latter then checks that the other processes correctly performed the desired rendezvous action. More precisely, whenever p_0 wants to handshake on some action (a or b), it sends some number of messages on c_1 to p_1 and waits for an acknowledgement on c_n before proceeding to its next move. In the meantime, the processes $p_1, \ldots, p_{n-1}$ do the same, but first receive and then transmit. The number of messages received by p_{i+1} from c_{i+1} is exactly the same as the number of messages sent

by p_i on c_{i+1}. The precise protocol is shown in Fig. 4, where each dashed transitions in $\mathcal{S}'$ is replaced in $\mathcal{R}$ by the alternative sequence below it (by introducing intermediate states). All other transitions are left unchanged. In $\mathcal{R}$, actions a and b are to be interpreted as internal, non-rendezvous actions. For instance, to simulate the rendezvous action a, p_0 first sends a message on c_1, internally performs a, and then receives a message from c_n.

To conclude the proof, we show that $[\![\mathcal{S}']\!]$ has an accepting run if, and only if, $[\![\mathcal{R}]\!]$ does. By construction, each rendezvous synchronisation in $[\![\mathcal{S}']\!]$ can be reproduced, through the above protocol, in $[\![\mathcal{R}]\!]$. We now argue that the protocol does not introduce any spurious behaviour. Recall that the channels $c_1, \ldots, c_n$ are empty at the beginning. So, for each $1 \leq i \leq n-1$, the process p_i may only simulate a rendezvous action after p_0 has initiated a synchronisation round. Let us look at the first rendezvous action that is simulated by p_0. If this action is a, then p_0 sends one message on c_1 and receives one message from c_n. This entails that all the other processes $p_1, \ldots, p_{n-1}$ simulate the rendezvous action a. At the end of this synchronisation round, all the channels $c_1, \ldots, c_n$ are again empty. If the first rendezvous action that p_0 simulates is b, then p_0 sends two messages on c_1 and receives $n+1$ messages from c_n. Again, this entails that $p_1, \ldots, p_{n-1}$ simulate, each, the rendezvous action b. Indeed, by contradiction, if p_i simulates a, then it must continue simulating a since there are not enough messages in c_i anymore to simulate b. Therefore, it simply relays messages from c_i to c_{i+1}, and cannot produce on c_{i+1} the extra message that p_{i+1} expects to simulate b. By applying the same arguments to the remaining processes $p_{i+1}, \ldots, p_{n-1}$, we obtain that p_{n-1} is not able to produce on c_n the $n+1$ messages that p_0 expects to complete its simulation of b, a contradiction. Again, at the end of this synchronisation round, all the channels $c_1, \ldots, c_n$ are empty. By repeating this analysis for each synchronisation round, we obtain that every accepting run of $[\![\mathcal{R}]\!]$ can be mapped back to an accepting run of $[\![\mathcal{S}']\!]$. □

We now show that the previous lemma still holds for closed directed walks, i.e., where processes/channels can be repeated. The proof is by induction on the cardinality of D. As expected, the induction step follows from Lemma 4.2.

Proposition 4.3 (Fusion). *If D is the support of a closed directed walk in a topology $\mathcal{T}$, then* $\textsc{Reach}(\mathcal{U}) \preceq_m \textsc{Reach}(\mathcal{T})$ *where $\mathcal{U}$ results from the fusion of D in $\mathcal{T}$.*

5 Characterisation of Decidable Topologies

We are now ready to state and prove our characterisation of decidable topologies. The characterisation is expressed in the same vein as Theorem 2.3, and generalises it. We first introduce some additional definitions and notations. Consider a topology $\mathcal{T} = \langle P, C, M, \mathsf{src}, \mathsf{dst}, \mathsf{msg} \rangle$, and let $D \subseteq C$ be a subset of channels. Two processes p and q are *synchronizable over D*, written $p \approx_D q$, if there exists a directed path from p to q and a directed path from q to p, both using only channels in D. Note that $\approx_D$ is an equivalence relation on P.

A *jumping circuit* is a sequence $(p_0, c_1, q_1, p_1, \ldots, c_n, q_n, p_n)$ of processes $p_i, q_i \in P$ and channels $c_i \in C$, with $n \geq 1$, such that $c_1, \ldots, c_n$ are pairwise distinct non-unary channels, $p_0 = p_n$, and $p_{i-1} \xrightarrow{c_i} q_i \approx_D p_i$ for all $1 \leq i \leq n$, where $D = C \setminus \{c_1, \ldots, c_n\}$. Recall that the binary relation $\xrightarrow{c}$ is the union of $\xRightarrow{c}$ and $\xLeftarrow{c}$. A *jumping cycle* is a jumping circuit $(p_0, c_1, q_1, p_1, \ldots, c_n, q_n, p_n)$ such that $q_i \not\approx_D q_j$ for all $1 \leq i < j \leq n$. To improve readability, jumping circuits and jumping cycles will often be written $p_0 \xrightarrow{c_1} q_1 \approx_D p_1 \cdots \xrightarrow{c_n} q_n \approx_D p_n$.

Remark 5.1. Every jumping circuit can be transformed into a jumping cycle.

Remark 5.2. For every jumping cycle $p_0 \xrightarrow{c_1} q_1 \approx_D p_1 \cdots \xrightarrow{c_n} q_n \approx_D p_n$, there exist n pairwise disjoint subsets $D_1, \ldots, D_n$ of the set $D = C \setminus \{c_1, \ldots, c_n\}$ such that $q_i \approx_{D_i} p_i$ for all $1 \leq i \leq n$.

Theorem 5.3. *Given a topology $\mathcal{T}$,* Reach($\mathcal{T}$) *is decidable if, and only if, $\mathcal{T}$ has no jumping cycle.*

The two directions of the theorem are proved in the two subsections below. To illustrate our characterisation, let us give some examples of decidable topologies. Certainly, polyforest topologies are decidable since they contain no undirected cycle, therefore no jumping cycle. Moreover, decidability is preserved if we add, for each channel of the polyforest, a unary channel in the opposite direction (see Lemma 5.4 below). Even further, we still get a decidable topology if each process is expanded into a sub-topology containing only unary channels. These operations introduce non-trivial cycles of unary and non-unary channels. Finally, adding unary channels looping on the same process always preserves decidability, as well as adding additional unary channels in parallel to already existing ones.

5.1 Undecidability

Consider a topology $\mathcal{T}$ containing a jumping cycle $\xi = (p_0, c_1, q_1, p_1, \ldots, c_n, q_n, p_n)$. By Remark 5.2, it holds that $p_0 \xrightarrow{c_1} q_1 \approx_{D_1} p_1 \cdots \xrightarrow{c_n} q_n \approx_{D_n} p_n$ for some pairwise disjoint subsets $D_1, \ldots, D_n$ of $C \setminus \{c_1, \ldots, c_n\}$. We may assume, w.l.o.g., that each D_i is the support of a closed directed walk in $\mathcal{T}$. To prove that Reach($\mathcal{T}$) is undecidable, we show that Reach($\mathcal{U}$) $\preceq_m$ Reach($\mathcal{T}$) for some topology $\mathcal{U}$ with an undirected cycle containing only non-unary channels, hence, for which reachability is undecidable by Theorem 2.3. To do so, we build a sequence of topologies $\mathcal{U}_0, \mathcal{U}_1, \ldots, \mathcal{U}_n$ by fusing together synchronizable processes, as follows: We define $\mathcal{U}_0 = \mathcal{T}$, and, for each $1 \leq i \leq n$, we let $\mathcal{U}_i$ result from the fusion of D_i in $\mathcal{U}_{i-1}$. Pairwise disjointness of $D_1, \ldots, D_n$ ensures that every channel in D_i is still a channel in $\mathcal{U}_{i-1}$. It is routinely checked that:

- D_i is still the support of a closed directed walk in $\mathcal{U}_{i-1}$, and
- ξ induces a closed undirected walk $u_0 \xrightarrow{c_1} u_1 \cdots \xrightarrow{c_n} u_n$ in $\mathcal{U}_n$.

The first item entails, by Proposition 4.3, that Reach($\mathcal{U}_i$) $\preceq_m$ Reach($\mathcal{U}_{i-1}$). By the transitivity of $\preceq_m$, we get that Reach($\mathcal{U}_n$) $\preceq_m$ Reach($\mathcal{U}_0$). Since $c_1, \ldots, c_n$

are pairwise distinct non-unary channels, we derive, from the second item, that $\mathcal{U}_n$ contains an undirected cycle with only non-unary channels. It follows from Theorem 2.3 that $\textsc{Reach}(\mathcal{U}_n)$ is undecidable. As $\mathcal{U}_0 = \mathcal{T}$, we conclude that $\textsc{Reach}(\mathcal{T})$ is undecidable.

5.2 Decidability

Starting from a topology $\mathcal{T}$ with no jumping cycle, we apply a sequence of topological transformations that produce a topology $\mathcal{U}$ with only unary channels, and such that $\textsc{Reach}(\mathcal{T}) \preceq_m \textsc{Reach}(\mathcal{U})$. Since the latter is decidable by Theorem 2.2, the former is decidable as well.

Given a topology $\mathcal{T}$ and a channel c in $\mathcal{T}$, an *acknowledgement channel* for c is a new unary channel, written $\overleftarrow{c}$, with source $\mathsf{dst}(c)$ and destination $\mathsf{src}(c)$. The following lemma says that adding an acknowledgement channel for an essential non-unary channel preserves the absence of jumping cycle. It immediately entails Corollary 5.5 below.

Lemma 5.4. *Consider a topology $\mathcal{T}$ and an essential non-unary channel c therein. Let $\mathcal{U}$ be the topology obtained from $\mathcal{T}$ by adding an acknowledgement channel for c. Then $\mathcal{T}$ contains a jumping cycle if $\mathcal{U}$ contains a jumping cycle.*

Proof (sketch). Assume that $\mathcal{T}$ has no jumping cycle, but adding $\overleftarrow{c}$ yields a topology $\mathcal{U}$ with a jumping cycle $p_0 \xrightarrow{c_1} q_1 \approx_{D_1} p_1 \cdots \xrightarrow{c_n} q_n \approx_{D_n} p_n = p_0$. Observe that $\overleftarrow{c}$ cannot be one of $c_1, \ldots, c_n$ since it is unary. If $\overleftarrow{c}$ does not appear in any D_i, then $\mathcal{T}$ has a jumping cycle. Hence, $\overleftarrow{c}$ appears on a closed directed walk π that synchronises two processes q_i and p_i. Since c is essential, it must appear on π too. We may assume, w.l.o.g., that $\overleftarrow{c}$ appears on the directed path from p_i to q_i on π, and that c appears on the directed path from q_i to p_i. We get that $q_i \xrightarrow{*} p \xrightarrow{c} q \xrightarrow{*} p_i \xrightarrow{*} q \xrightarrow{\overleftarrow{c}} p \xrightarrow{*} q_i$. Therefore, q_i can synchronise with p using only channel from $E_i = D_i \setminus \{c, \overleftarrow{c}\}$, and similarly p_i can synchronise with q via E_i. So we can build a jumping cycle in $\mathcal{T}$ from the jumping cycle in $\mathcal{U}$ by replacing $q_i \approx_{D_i} p_i$ by $q_i \approx_{E_i} p \xrightarrow{c} q \approx_{E_i} p_i$, a contradiction. □

Corollary 5.5. *Consider a topology $\mathcal{T}$ and an essential non-unary channel c therein. Let $\mathcal{U}$ be the topology resulting from the synchronisation of c in $\mathcal{T}$. Then $\mathcal{T}$ contains a jumping cycle if $\mathcal{U}$ contains a jumping cycle.*

Remark 5.6. The converse of Corollary 5.5 also holds, but it is not required for the proof of Theorem 5.3.

We say that a topology $\mathcal{T}$ is *divided* if the destination of every irreversible unary channel is a sink (i.e., is not the source of some channel) and is not the destination of some non-unary channel. The following two properties of divided topologies are crucial in the proof of Theorem 5.3.

Lemma 5.7. *Consider a topology $\mathcal{T}$ and a non-unary channel c therein. If $\mathcal{T}$ is divided, then so is the topology resulting from the synchronisation of c in $\mathcal{T}$.*

Lemma 5.8. *If $\mathcal{T}$ is a divided topology with no jumping cycle, then every non-unary channel in $\mathcal{T}$ is essential.*

We now prove the "if" direction of Theorem 5.3. Assume that $\mathcal{T}$ has no jumping cycle. Let $c_1, \ldots, c_n$ denote the non-unary channels of $\mathcal{T}$. We first build $\mathcal{U}_0$ from $\mathcal{T}$ by splitting all unary channels that are irreversible in $\mathcal{T}$. Note that $\mathcal{U}_0$ does not depend on the order in which the irreversible unary channels of $\mathcal{T}$ are split. It follows from Proposition 3.10 and the transitivity of $\preceq_m$ that $\textsc{Reach}(\mathcal{T}) \preceq_m \textsc{Reach}(\mathcal{U}_0)$. By construction, the topology $\mathcal{U}_0$ is divided, and it still has no jumping cycle. So, by Lemma 5.8, every non-unary channel in $\mathcal{U}_0$ is essential. Notice that $\mathcal{U}_0$ has the same non-unary channels as $\mathcal{T}$, namely $c_1, \ldots, c_n$. For each $1 \leq i \leq n$, let $\mathcal{U}_i$ be the topology resulting from the synchronisation of c_i in $\mathcal{U}_{i-1}$. By induction, it is immediate to prove that, for every $0 \leq i \leq n$, $\mathcal{U}_i$ has no jumping cycle, the induction step holding by Corollary 5.5, that $\mathcal{U}_i$ is divided, by Lemma 5.7, that $c_{i+1}, \ldots, c_n$ are still essential in $\mathcal{U}_i$, by Lemma 5.8, and that $\textsc{Reach}(\mathcal{U}_{i-1}) \preceq_m \textsc{Reach}(\mathcal{U}_i)$, by Proposition 3.4. By the transitivity of $\preceq_m$, we get that $\textsc{Reach}(\mathcal{T}) \preceq_m \textsc{Reach}(\mathcal{U}_n)$. Since $\mathcal{U}_n$ contains only unary channels, $\textsc{Reach}(\mathcal{U}_n)$ is decidable by Theorem 2.2. Thus $\textsc{Reach}(\mathcal{T})$ is decidable.

5.3 Complexity

We consider the reachability problem for systems of communicating processes whose topology has no jumping cycle. This problem, written ReachNJC, is the union of the problems $\textsc{Reach}(\mathcal{T})$ for topologies $\mathcal{T}$ with no jumping cycle. Note that deciding whether a topology has a jumping cycle is a simple graph-theoretic problem that can be solved in polynomial time. Hence, it can be checked efficiently whether a given system of communicating processes is an instance of ReachNJC or not. Here, we show that ReachNJC is equivalent to reachability in Petri nets.

The *size* of a labelled transition system $\mathcal{A} = \langle S, S_I, S_F, A, \to \rangle$ is defined as $|\mathcal{A}| = |S| + |\to|$. Similarly, the *size* of a topology $\mathcal{T} = \langle P, C, M, \mathsf{src}, \mathsf{dst}, \mathsf{msg} \rangle$ is $|\mathcal{T}| = |P| + |C| + \sum_{c \in C} |\mathsf{msg}(c)|$. Finally, the *size* of a system of communicating processes $\mathcal{S} = \langle \mathcal{T}, \{\mathcal{A}^p\}_{p \in P} \rangle$ is $|\mathcal{S}| = |\mathcal{T}| + \sum_{p \in P} |\mathcal{A}^p|$. The algorithm in Subsec. 5.2 transforms a system $\mathcal{S}$ over a topology with no jumping cycle, into a system $\mathcal{S}'$ with unary channels only. Since unary channels are essentially counters (over the natural numbers) that may only be incremented and decremented, $\mathcal{S}'$ can be naturally interpreted as a Petri net. Crucially, we show that $\mathcal{S}'$ (and thus the Petri net) has size polynomial in $|\mathcal{S}|$. This is possible since the synchronisation operation from Sec. 3 avoids taking the product of processes (at the cost of introducing 1-bounded unary channels/counters).

Theorem 5.9. ReachNJC *is equivalent to reachability in Petri nets under polynomial-time many-one reductions.*

6 Discussion

Unary vs. Bag Channels. A bag channel is a channel where messages can be freely reordered. Therefore, it suffices to *count* how many messages of each type

are in the channel. So, a bag channel over a message alphabet of cardinality n can be implemented with n unary channels in parallel. A topology of bag and FIFO channels is a topology (as defined in Sec. 2) where, in addition, each channel has a flag indicating whether it is ordered (FIFO) or not (bag). Our characterisation from Sec. 5 immediately generalises to bag channels by modifying the definition of jumping cycle and requiring that the c_i's be non-unary FIFO channels (instead of just non-unary).

Unary/Bag vs. Lossy Channels. Another over-approximation incomparable with bag channels is provided by *lossy channels*, where messages might be lost at any moment [2,7]. A complete characterisation of decidable topologies mixing perfect and lossy channels has been presented in [8]. In order to reduce to basic decidable topologies, they consider two reduction rules. The first one is the *fusion* of essential channels. This is similar in spirit to our *synchronisation* (see Proposition 3.4), with the only difference that fusing channels requires to take the product of the underlying processes, while synchronising channels just replaces one channel with several 1-bounded unary channels. This allows us to obtain precise complexity bounds in Subsec. 5.3. The second reduction rule is *splitting* a complex topology $\mathcal{T}$ into $\mathcal{T}_1$ and $\mathcal{T}_2$ when all channels between $\mathcal{T}_1$ and $\mathcal{T}_2$ are unidirectional and lossy. Despite similar names, this is different to our splitting technique (see Proposition 3.10), because we split (irreversible) channels, and not topologies. However, while lossy channels cannot be split, the lossy channels involved in the splitting of $\mathcal{T}$ in the sense of [8] are *irreversible* in our terminology, and thus could be split *if they were perfect channels*. Moreover, if we replace lossy channels with perfect bag channels, it additionally holds that, if $\mathcal{T}_1, \mathcal{T}_2$ above are decidable in our characterisation (i.e., no jumping cycles), then the same holds for $\mathcal{T}$. Since also fusion/synchronisation preserves decidable topologies in both settings, we have that any decidable topology of perfect and lossy channels is still a decidable topology by replacing lossy channels with perfect bag channels.

Moreover, some topologies which are undecidable with lossy channels become decidable with bag channels. For example, the topology with three parallel channels c, d, e between two processes with c perfect FIFO and d, e lossy FIFO is undecidable, while if d, e are bag channels it becomes decidable.

Finally, while the topology in Fig. 2 is undecidable when channels b_0 and b_1 are either both bag channels or both lossy channels, our construction with unary bag channels is correct *even if those are unary and lossy*. Indeed, as soon as any message gets lost due to lossiness, our synchronization protocol gets irremediably stuck. However, the construction from [8] does not generalise to unary channels in this case. Thus, we extend their undecidability result to this topology.

7 Conclusions and Future Work

We have presented a complete characterisation of the decidable topologies for networks of finite-state automata communicating over FIFO and bag channels. Remarkably, every decidable topology can be solved using two simple techniques

(synchronising essential channels and splitting irreversible channels), whereas every topology that cannot be solved with these two techniques is undecidable.

The same characterisation problem is solved in [8] but for networks mixing perfect and lossy FIFO channels. A direction for future research is to characterise decidable topologies of lossy/perfect FIFO/bag channels.

Relaxing FIFO channels to the bag type can be applied in other contexts as well. For example, the work [11] studies topologies of networks of *pushdown automata* communicating over FIFO channels, and it is natural to ask what happens when some channels and/or pushdown stores are bags instead of strings.

References

1. Abdulla, P.A., Atig, M.F., Cederberg, J.: Analysis of message passing programs using SMT-solvers. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 272–286. Springer, Heidelberg (2013)
2. Abdulla, P., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. 127(2), 91–101 (1996)
3. Boigelot, B., Godefroid, P.: Symbolic verification of communication protocols with infinite state spaces using QDDs. Form. Methods Sys. Des. 14, 237–255 (1999)
4. Bouajjani, A., Emmi, M.: Bounded phase analysis of message-passing programs. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 451–465. Springer, Heidelberg (2012)
5. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
6. Cécé, G., Finkel, A.: Verification of programs with half-duplex communication. Inf. Comput. 202(2), 166–190 (2005)
7. Cécé, G., Finkel, A., Purushothaman Iyer, S.: Unreliable channels are easier to verify than perfect channels. Inf. Comput. 124(1), 20–31 (1996)
8. Chambart, P., Schnoebelen, P.: Mixing lossy and perfect fifo channels. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 340–355. Springer, Heidelberg (2008)
9. Ganty, P., Majumdar, R.: Algorithmic verification of asynchronous programs. ACM Trans. Program. Lang. Syst. 34(1), 1–6 (2012)
10. Haase, C., Schmitz, S., Schnoebelen, P.: The power of priority channel systems. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013 – Concurrency Theory. LNCS, vol. 8052, pp. 319–333. Springer, Heidelberg (2013)
11. Heußner, A., Leroux, J., Muscholl, A., Sutre, G.: Reachability analysis of communicating pushdown systems. LMCS 8(3), 1–20 (2012)
12. Jhala, R., Majumdar, R.: Interprocedural analysis of asynchronous programs. SIGPLAN Not. 42(1), 339–350 (2007)
13. Kosaraju, S.R.: Decidability of reachability in vector addition systems. In: Proc. STOC 1982, pp. 267–281 (1982) (preliminary version)
14. La Torre, S., Madhusudan, P., Parlato, G.: Context-bounded analysis of concurrent queue systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
15. Leroux, J.: Vector addition system reachability problem: A short self-contained proof. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 41–64. Springer, Heidelberg (2011)

16. Mayr, E.: An algorithm for the general petri net reachability problem. In: Proc. STOC 1981, pp. 238–246 (1981)
17. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
18. Pachl, J.K.: Reachability problems for communicating finite state machines. Research Report CS-82-12, University of Waterloo (May 1982)
19. Sen, K., Viswanathan, M.: Model checking multithreaded programs with asynchronous atomic methods. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 300–314. Springer, Heidelberg (2006)
20. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice-Hall (2001)