Asynchronous Distributed Automata: A Characterization of the Modal Mu-Fragment*

Fabian Reiter

IRIF, Université Paris Diderot, Paris, France
fabian.reiter@gmail.com

Abstract

We establish the equivalence between a class of asynchronous distributed automata and a small fragment of least fixpoint logic, when restricted to finite directed graphs. More specifically, the logic we consider is (a variant of) the fragment of the modal μ-calculus that allows least fixpoints but forbids greatest fixpoints. The corresponding automaton model uses a network of identical finite-state machines that communicate in an asynchronous manner and whose state diagram must be acyclic except for self-loops. Exploiting the connection with logic, we also prove that the expressive power of those machines is independent of whether or not messages can be lost.

1998 ACM Subject Classification C.2.4 Distributed Systems, F.1.1 Models of Computation, F.4.1 Mathematical Logic

Keywords and phrases Finite automata, distributed computing, modal logic, μ-calculus

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.100

1 Introduction

One of the core disciplines of distributed computing is to design and analyze message passing algorithms that solve graph problems in computer networks. Usually, the problem instance considered in that context is precisely the graph defined by the network in which the computations are performed. All nodes of the network run the same algorithm concurrently, and often make no prior assumptions about the size and topology of the graph. Typical problems that can be solved by such distributed algorithms include graph coloring, leader election, and the construction of spanning trees and maximal independent sets. A comprehensive treatment of the subject can be found in [10] and [11].

The present paper follows up on relatively recent results by Hella et al. and Kuusisto, which establish novel connections between modal logic and some restricted classes of distributed algorithms. These weak types of algorithms, referred to in the following as *distributed automata*, can be represented as deterministic finite-state machines that read sets of states instead of the usual alphabetic symbols. Intuitively, to run a distributed automaton on some node-labeled directed graph G, a separate copy of the same machine is placed on every node and initialized to a state that may depend on the node's label. Each node v communicates with its peers by sending its current state q to every outgoing neighbor, while at the same time collecting the states received from its incoming neighbors into a set S. The successor state of q is then computed as a function of q and S. In particular, this means that v cannot distinguish between two incoming neighbors that share the same state. Acting as a semi-decider, the automaton accepts G at position v precisely if v visits an accepting state at some point in time. Either way, all machines of the network run and communicate forever.

* This work is supported by the DeLTA project (ANR-16-CE40-0007).

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 100; pp. 100:1–100:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In [5, 6], Hella et al. have compared several classes of distributed algorithms, of which the weakest uses the restricted communication model described above. Deviating only in nonessential details from their original definition, we can think of those weakest algorithms as *local synchronous* distributed automata. Here, "synchronous" means that all nodes of the network share a global clock, thereby allowing the computation to proceed in an infinite sequence of rounds. In each round, all the nodes compute their next state simultaneously, based on the information collected in the previous round. By the term "local" we mean that the nodes stop changing their state after a constant number of rounds, a usage in accordance with the established terminology of distributed computing (see, e.g., [12]). Equivalently, the state diagram of a local automaton is acyclic as long as we ignore sink states (i.e., states that cannot be left once reached). The work of Hella et al. reveals an intriguing link between distributed computing and modal logic. In particular, it follows immediately from [5, 6, Thm. 1] that the graph properties recognizable by local synchronous automata are precisely those definable in *backward modal logic*, the variant of (basic) modal logic where the usual modal operators are replaced by their backward-looking variants.

Motivated by the preceding result, the connection with modal logic was further investigated by Kuusisto in [7] and [8]. The former paper lifts the constraint of locality imposed in [5, 6], thereby allowing automata with arbitrary state diagrams. These (nonlocal) synchronous automata are then given a logical characterization in terms of a new recursive logic dubbed *modal substitution calculus*. Furthermore, [7, Prp. 7] shows that on finite graphs, synchronous automata can easily recognize all the properties definable in the least fixpoint fragment of the backward μ-calculus. This logic, which we shall refer to simply as the *backward μ-fragment*, extends backward modal logic with a least fixpoint operator that may not be negated. It thus allows to express statements using least fixpoints, but unlike in the full backward μ-calculus, greatest fixpoints are forbidden. On the other hand, the reverse conversion from synchronous automata to the backward μ-fragment is not possible in general. As explained in [7, Prp. 6], it is easy to come up with a synchronous automaton that makes crucial use of the fact that a node can determine whether it receives the same information from all of its incoming neighbors at exactly the same time. Such a behavior cannot be simulated in the backward μ-fragment. By the same token, even the much more expressive monadic second-order logic (MSO) is incomparable with synchronous automata.

Given that the preceding argument relies solely on synchrony, it seems natural to ask whether removing this feature can lead to a distributed automaton model that has the same expressive power as the backward μ-fragment. The present paper provides a positive answer to this question. We introduce several classes of *asynchronous automata* that transfer the standard notion of asynchronous algorithm to the setting of finite-state machines. Basically, this means that we eliminate the global clock from the network, thus making it possible for nodes to operate at different speeds and for messages to be delayed for arbitrary amounts of time, or even be lost. From the syntactic point of view, an asynchronous automaton is the same as a synchronous one, but it has to satisfy an additional semantic condition: its acceptance behavior must be independent of any timing-related issues. Taking a closer look at the automata obtained by translating formulas of the backward μ-fragment, we can easily see that they are in fact asynchronous. Furthermore, their state diagrams are almost acyclic, except that all the states are allowed to have self-loops (not only the sink states). We call this property *quasi-acyclic*. The paper's main contribution is to show that now we can also go in the other direction: every quasi-acyclic asynchronous automaton can be converted into an equivalent formula of the backward μ-fragment. Incidentally, this remains true even if we consider a seemingly more powerful variant of asynchronous automata, where all messages

are guaranteed to be delivered. To illustrate the basic concepts, an example of an automaton and an equivalent formula will be provided in Figure 1, at the end of the next section.

The remainder of this paper is organized as follows: After giving the necessary formal definitions in Section 2, we state and briefly discuss the main result in Section 3. The proof is then developed in the last two sections. Section 4 presents the rather straightforward translation from logic to automata. The reverse translation is given in Section 5, which is a bit more involved and therefore occupies the largest part of the paper.

2 Preliminaries

We denote the set of Boolean values by $\mathbb{2} = \{0, 1\}$, the set of non-negative integers by $\mathbb{N} = \{0, 1, 2, \dots\}$, and the set of positive integers by $\mathbb{N}_{>0} = \mathbb{N} \setminus \{0\}$. With respect to a given set S, we write 2^S for the power set, S^k for the set of k-tuples ($k \in \mathbb{N}$), and $|S|$ for the cardinality. As a special case of k-tuples, $\mathbb{2}^k$ denotes the set of all binary strings of length k. Furthermore, the length of a string x is written as $|x|$.

For $\ell \in \mathbb{N}$, a (finite) *ℓ-bit labeled directed graph*, abbreviated *digraph*, is a structure $G = (V, E, \lambda)$, where V is a finite nonempty set of nodes, $E \subseteq V \times V$ is a set of directed edges, and $\lambda \colon V \to \mathbb{2}^\ell$ is a labeling that assigns a binary string of length ℓ to each node. Isomorphic digraphs are considered to be equal. If v lies in V, we call the pair (G, v) a *pointed digraph*. Moreover, if uv is an edge in E, then u is called an *incoming neighbor* of v.

▶ **Definition 1** (Distributed Automaton). A (distributed) *automaton* with ℓ-bit input is a tuple $A = (Q, \delta_0, \delta, F)$, where Q is a finite set of states, $\delta_0 \colon \mathbb{2}^\ell \to Q$ is an initialization function, $\delta \colon Q \times \mathbb{2}^Q \to Q$ is a transition function, and $F \subseteq Q$ is a set of accepting states.

To run such an automaton A on a digraph G, we regard the edges of G as FIFO buffers. Each buffer vw will always contain a sequence of states previously traversed by node v. An adversary chooses when v evaluates δ to push a new state to the back of the buffer, and when the current first state gets popped from the front. The details are clarified in the following.

A *trace* of an automaton $A = (Q, \delta_0, \delta, F)$ is a finite nonempty sequence $\sigma = q_1 \dots q_n$ of states in Q such that $q_i \neq q_{i+1}$ and $\delta(q_i, S_i) = q_{i+1}$ for some $S_i \subseteq Q$. We say that A is *quasi-acyclic* if its set of traces $\mathfrak{Q}$ is finite. In other words, its state diagram must not contain any directed cycles, except for self-loops.

For any states $p, q \in Q$ and any (possibly empty) sequence σ of states in Q, we define the unary postfix operators first, last, $\mathsf{pushlast}$ and $\mathsf{popfirst}$ as follows: $p\sigma.\mathsf{first} = \sigma p.\mathsf{last} = p$,

$$\sigma p.\mathsf{pushlast}(q) = \begin{cases} \sigma pq & \text{if } p \neq q, \\ \sigma p & \text{if } p = q, \end{cases} \qquad \text{and} \qquad p\sigma.\mathsf{popfirst} = \begin{cases} \sigma & \text{if } \sigma \text{ is nonempty,} \\ p\sigma & \text{if } \sigma \text{ is empty.} \end{cases}$$

An (asynchronous) *timing* of a digraph $G = (V, E, \lambda)$ is an infinite sequence $\tau = (\tau_1, \tau_2, \tau_3, \dots)$ of maps $\tau_t \colon V \cup E \to \mathbb{2}$, indicating which nodes and edges are active at time t, where 1 is assigned infinitely often to every node and every edge. More formally, for all $t \in \mathbb{N}_{>0}$, $v \in V$ and $e \in E$, there exist $i, j > t$ such that $\tau_i(v) = 1$ and $\tau_j(e) = 1$. We refer to this as the *fairness property* of τ. As a restriction, we say that τ is *lossless-asynchronous* if $\tau_t(uv) = 1$ implies $\tau_t(v) = 1$ for all $t \in \mathbb{N}_{>0}$ and $uv \in E$. Furthermore, τ is called the (unique) *synchronous timing* of G if $\tau_t(v) = \tau_t(e) = 1$ for all $t \in \mathbb{N}_{>0}$, $v \in V$ and $e \in E$.

▶ **Definition 2** (Asynchronous Run). Let $A = (Q, \delta_0, \delta, F)$ be a distributed automaton with ℓ-bit input and $\mathfrak{Q}$ be its set of traces. Furthermore, let $G = (V, E, \lambda)$ be an ℓ-bit labeled

digraph and $\tau = (\tau_1, \tau_2, \tau_3, \dots)$ be a timing of G. The (asynchronous) *run* of A on G timed by τ is the infinite sequence $\rho = (\rho_0, \rho_1, \rho_2, \dots)$ of *configurations* $\rho_t \colon V \cup E \to \mathfrak{Q}$, with $\rho_t(V) \subseteq Q$, which are defined inductively as follows, for $t \in \mathbb{N}$, $v \in V$ and $vw \in E$:

$$\rho_0(v) = \rho_0(vw) = \delta_0(\lambda(v)),$$

$$\rho_{t+1}(v) = \begin{cases} \rho_t(v) & \text{if } \tau_{t+1}(v) = 0, \\ \delta\big(\rho_t(v), \{\rho_t(uv).\mathsf{first} \mid uv \in E\}\big) & \text{if } \tau_{t+1}(v) = 1, \end{cases}$$

$$\rho_{t+1}(vw) = \begin{cases} \rho_t(vw).\mathsf{pushlast}(\rho_{t+1}(v)) & \text{if } \tau_{t+1}(vw) = 0, \\ \rho_t(vw).\mathsf{pushlast}(\rho_{t+1}(v)).\mathsf{popfirst} & \text{if } \tau_{t+1}(vw) = 1. \end{cases}$$

If τ is the synchronous timing of G, we refer to ρ as the *synchronous run* of A on G.

Throughout this paper, we assume that our digraphs, automata and logical formulas agree on the number ℓ of labeling bits. An automaton A *accepts* a pointed digraph (G, v) under timing τ if v visits an accepting state at some point in the run ρ of A on G timed by τ, i.e., if there exists $t \in \mathbb{N}$ such that $\rho_t(v) \in F$. If we simply say that A accepts (G, v), without explicitly specifying a timing τ, then we stipulate that ρ is the synchronous run of A on G.

Given a digraph $G = (V, E, \lambda)$ and a class T of timings of G, the automaton A is called *consistent* for G and T if for all $v \in V$, either A accepts (G, v) under every timing in T, or A does not accept (G, v) under any timing in T. We say that A is *asynchronous* if it is consistent for every possible choice of G and T, and *lossless-asynchronous* if it is consistent for every choice where T contains only lossless-asynchronous timings. By contrast, we call an automaton *synchronous* if we wish to emphasize that no such consistency requirements are imposed. Intuitively, all automata can operate in the synchronous setting, but only some of them also work reliably in environments that provide fewer guarantees.

A *digraph property* is a set L of pointed digraphs. We call L the digraph property *recognized* by an automaton A if it consist precisely of those pointed digraphs that are accepted by A. We denote by AA , LA and SA the classes of digraph properties recognizable by asynchronous, lossless-asynchronous and synchronous automata, respectively. Similarly, QAA , QLA and QSA are the corresponding classes recognizable by quasi-acyclic automata.

Turning to logic, let Var be an infinite supply of propositional variables. We define the formulas of *backward modal logic* with ℓ propositional constants by means of the grammar

$$\varphi ::= \bot \mid \top \mid P_i \mid \neg P_i \mid X \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid \bar{\Diamond}\varphi \mid \bar{\Box}\varphi\,,$$

where $0 \leq i < \ell$ and $X \in \mathrm{Var}$. Note that this syntax ensures that variables cannot be negated. Given such a formula φ, an ℓ-bit labeled digraph $G = (V, E, \lambda)$ and a variable assignment $\alpha \colon \mathrm{Var} \to 2^V$, we write $[\![\varphi]\!]_{G,\alpha}$ to denote the subset of nodes of G at which φ holds with respect to α. For *atomic propositions* P_i and X, the corresponding semantics are defined by $[\![P_i]\!]_{G,\alpha} = \{v \in V \mid \lambda(v)(i) = 1\}$ and $[\![X]\!]_{G,\alpha} = \alpha(X)$, where $\lambda(v)(i)$ is the i-th bit of $\lambda(v)$. The Boolean constants and connectives are interpreted in the usual way, for instance, $[\![\top]\!]_{G,\alpha} = V$ and $[\![(\varphi \vee \psi)]\!]_{G,\alpha} = [\![\varphi]\!]_{G,\alpha} \cup [\![\psi]\!]_{G,\alpha}$. Finally, the *backward diamond* $\bar{\Diamond}$ and the *backward box* $\bar{\Box}$ represent backward-looking modal operators, with the semantics

$$[\![\bar{\Diamond}\varphi]\!]_{G,\alpha} = \big\{v \in V \;\big|\; u \in [\![\varphi]\!]_{G,\alpha} \text{ for some } u \in V \text{ such that } uv \in E\big\} \quad \text{and}$$
$$[\![\bar{\Box}\varphi]\!]_{G,\alpha} = \big\{v \in V \;\big|\; u \in [\![\varphi]\!]_{G,\alpha} \text{ for all } u \in V \text{ such that } uv \in E\big\}.$$

Traditionally, the modal μ-calculus is defined to comprise individual fixpoints which may be nested. However, it is well-known that we can add simultaneous fixpoints to the

μ-calculus without changing its expressive power, and that nested fixpoints of the same type (i.e., least or greatest) can be rewritten as non-nested simultaneous ones (see, e.g., [3, § 3.7] or [9, § 4.3]). The following definition directly takes advantage of this fact. We shall restrict ourselves to the *μ-fragment of the backward μ-calculus*, abbreviated *backward μ-fragment*, where only least fixpoints are allowed, and where the usual modal operators are replaced by their backward-looking variants. Without loss of generality, we stipulate that each formula of the backward μ-fragment with ℓ propositional constants is of the form

$$\varphi = \mu \begin{pmatrix} X_0 \\ \vdots \\ X_k \end{pmatrix} . \begin{pmatrix} \varphi_0(P_0, \dots, P_{\ell-1}, X_0, \dots, X_k) \\ \vdots \\ \varphi_k(P_0, \dots, P_{\ell-1}, X_0, \dots, X_k) \end{pmatrix},$$

where $X_0, \dots, X_k \in \mathrm{Var}$, and $\varphi_0, \dots, \varphi_k$ are formulas of backward modal logic with ℓ propositional constants that may contain no other variables than $X_0, \dots, X_k$.

For every digraph $G = (V, E, \lambda)$, the tuple $(\varphi_0, \dots, \varphi_k)$ gives rise to an operator $f\colon (2^V)^{k+1} \to (2^V)^{k+1}$ that takes some valuation of $\vec{X} = (X_0, \dots, X_k)$ and reassigns to each X_i the resulting valuation of φ_i. More formally, f maps $\vec{W} = (W_0, \dots, W_k)$ to $(W_0', \dots, W_k')$ such that $W_i' = [\![\varphi_i]\!]_{G,[\vec{X} \mapsto \vec{W}]}$. Here, $[\vec{X} \mapsto \vec{W}]$ can be any variable assignment that interprets each X_i as W_i. A (simultaneous) *fixpoint* of the operator f is a tuple $\vec{W} \in (2^V)^{k+1}$ such that $f(\vec{W}) = \vec{W}$. Since, by definition, variables occur only positively in formulas, the operator f is *monotonic*. This means that $\vec{W} \subseteq \vec{W}'$ implies $f(\vec{W}) \subseteq f(\vec{W}')$ for all $\vec{W}, \vec{W}' \in (2^V)^{k+1}$, where set inclusions are to be understood componentwise (i.e., $W_i \subseteq W_i'$ for each i). Therefore, by virtue of a theorem due to Knaster and Tarski, f has a *least fixpoint*, which is defined as the unique fixpoint $\vec{U} = (U_0, \dots, U_k)$ of f such that $\vec{U} \subseteq \vec{W}$ for every other fixpoint $\vec{W}$ of f. As a matter of fact, the Knaster-Tarski theorem even tells us that $\vec{U}$ is equal to $\bigcap \{\vec{W} \in (2^V)^{k+1} \mid f(\vec{W}) \subseteq \vec{W}\}$, where set operations must also be understood componentwise. Another, perhaps more intuitive, way of characterizing $\vec{U}$ is to consider the inductively constructed sequence of approximants $(\vec{U}^0, \vec{U}^1, \vec{U}^2, \dots)$, where $\vec{U}^0 = (\emptyset, \dots, \emptyset)$ and $\vec{U}^{j+1} = f(\vec{U}^j)$. Since this sequence is monotonically increasing and V is finite, there exists $n \in \mathbb{N}$ such that $\vec{U}^n = \vec{U}^{n+1}$. It is easy to check that $\vec{U}^n$ coincides with the least fixpoint $\vec{U}$. For more details and proofs, see, e.g., [4, § 3.3.1].

Having introduced the necessary background, we can finally establish the semantics of φ with respect to G: the set $[\![\varphi]\!]_G$ of nodes at which φ holds is precisely U_0, the first component of $\vec{U}$. A pointed digraph (G, v) satisfies φ, in symbols $(G, v) \models \varphi$, if $v \in [\![\varphi]\!]_G$. Accordingly, the digraph property *defined* by φ is $\{(G, v) \mid (G, v) \models \varphi\}$, and we denote by $\Sigma_1^{\overline{\mu}}$ the class of all digraph properties defined by some formula of the backward μ-fragment.

As usual, two devices (i.e., automata or formulas) are *equivalent* if they specify (i.e., recognize or define) the same property. Figure 1 provides an example of such an equivalence.

3 Main result

Based on the definitions given in Section 2, asynchronous automata are a special case of lossless-asynchronous automata, which in turn are a special case of synchronous automata.[1] Furthermore, quasi-acyclicity constitutes an additional (possibly orthogonal) restriction on these models. We thus immediately obtain the hierarchy of classes depicted in Figure 2a.

[1] This may seem counterintuitive at first sight, but it is actually consistent with the standard terminology of distributed computing: an asynchronous algorithm can always serve as a synchronous algorithm (i.e., it can be executed in a synchronous environment), but the converse is not true.

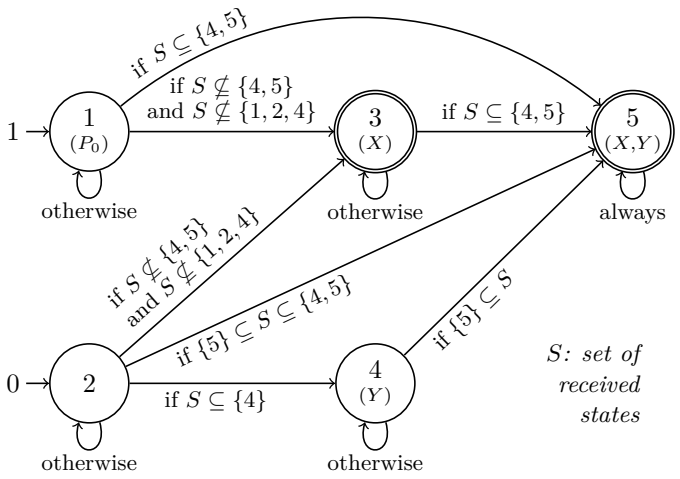
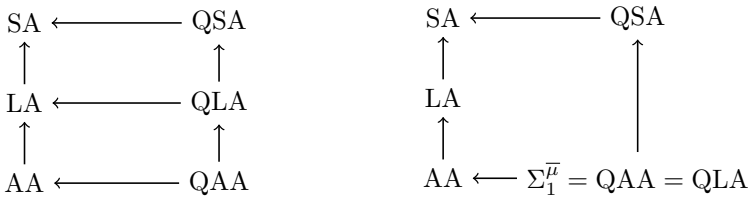


Figure 1 A quasi-acyclic asynchronous distributed automaton equivalent to the formula

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (P_0 \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

of the backward μ-fragment. A given pointed 1-bit labeled digraph (G, v) is accepted by this automaton if and only if, starting at v and following G's edges in the backward direction, it is possible to reach some node u labeled with 1 from which it is impossible to reach any directed cycle.



(a) immediate by the definitions **(b)** collapse shown in this paper

Figure 2 Hierarchy of the classes of digraph properties recognizable by distributed automata, depending on whether the automata are synchronous (S), lossless-asynchronous (L), asynchronous (A), or quasi-acyclic (Q). The arrows denote inclusion (e.g., LA ⊆ SA).

Our main result provides a simplification of this hierarchy: the classes QAA and QLA are actually equal to the class of digraph properties definable in the backward μ-fragment. This yields the revised diagram shown in Figure 2b.

▶ **Theorem 3** ($\Sigma_1^{\overline{\mu}}$ = QAA = QLA)**.** *When restricted to finite digraphs, the backward μ-fragment is effectively equivalent to the classes of quasi-acyclic asynchronous automata and quasi-acyclic lossless-asynchronous automata.*

Proof. The forward direction is given by Proposition 4 (in Section 4), which asserts that $\Sigma_1^{\overline{\mu}} \subseteq$ QAA, and the trivial observation that QAA ⊆ QLA. For the backward direction, we use Proposition 7 (in Section 5), which asserts that QLA ⊆ $\Sigma_1^{\overline{\mu}}$. ◀

As stated before, synchronous automata are more powerful than the backward μ-fragment (and incomparable with monadic second-order logic). This holds even if we consider only quasi-acyclic automata, i.e., the inclusion $\Sigma_1^{\overline{\mu}} \subset$ QSA is known to be strict (see [7, Prp. 6]). Moreover, an upcoming paper will show that the inclusion QSA ⊂ SA is also strict.

In contrast, it remains open whether quasi-acyclicity is in fact necessary for characterizing $\Sigma_1^{\overline{\mu}}$. On the one hand, this notion is crucial for our proof (see Proposition 7), but on the other hand, no digraph property separating AA or LA from $\Sigma_1^{\overline{\mu}}$ has been found so far.

4 Computing least fixpoints using asynchronous automata

In this section, we prove the easy direction of the main result. Given a formula φ of the backward μ-fragment, it is straightforward to construct a (synchronous) distributed automaton A that computes on any digraph the least fixpoint $\vec{U}$ of the operator associated with φ. As long as it operates in the synchronous setting, A simply follows the sequence of approximants $(\vec{U}^0, \vec{U}^1, \dots)$ described in Section 2. It is important to stress that the very same observation has previously been made in [7, Prp. 7] (formulated from a different point of view). In the following proposition, we refine this observation by giving a more precise characterization of the obtained automaton: it is always quasi-acyclic and capable of operating in a (possibly lossy) asynchronous environment.

▶ **Proposition 4** ($\Sigma_1^{\overline{\mu}} \subseteq$ QAA). *For every formula of the backward μ-fragment, we can effectively construct an equivalent quasi-acyclic asynchronous automaton.*

Proof. Let $\varphi = \mu(X_0, \dots, X_k).(\varphi_0, \dots, \varphi_k)$ be a formula of the backward μ-fragment with ℓ propositional constants. Without loss of generality, we may assume that the subformulas $\varphi_0, \dots, \varphi_k$ do not contain any nested modal operators. To see this, suppose that $\varphi_i = \overline{\Diamond}\psi$. Then φ is equivalent to $\varphi' = \mu(X_0, \dots, X_i, \dots, X_k, Y).(\varphi_0, \dots, \varphi_i', \dots, \varphi_k, \psi)$, where Y is a fresh propositional variable and $\varphi_i' = \overline{\Diamond}Y$. The operator $\overline{\Box}$ and Boolean combinations of $\overline{\Diamond}$ and $\overline{\Box}$ are handled analogously.

We now convert φ into an equivalent automaton $A = (Q, \delta_0, \delta, F)$ with state set $Q = 2^{\{P_0, \dots, P_{\ell-1}, X_0, \dots, X_k\}}$. The idea is that each node v of the input digraph has to remember which of the atomic propositions $P_0, \dots, P_{\ell-1}, X_0, \dots, X_k$ have, so far, been verified to hold at v. Therefore, we define the initialization function such that $\delta_0(x) = \{P_i \mid x(i) = 1\}$ for all $x \in 2^\ell$. Let us write $(q, S) \models \varphi_i$ to indicate that a pair $(q, S) \in Q \times 2^Q$ satisfies a subformula φ_i of φ. This is the case precisely when φ_i holds at any node v that satisfies exactly the atomic propositions in q and whose incoming neighbors satisfy exactly the propositions specified by S. Note that this satisfaction relation is well-defined in our context because the nesting depth of modal operators in φ_i is at most 1. With that, the transition function of A can be succinctly described by $\delta(q, S) = q \cup \{X_i \mid (q, S) \models \varphi_i\}$. Since $q \subseteq \delta(q, S)$, we are guaranteed that the automaton is quasi-acyclic. Finally, the accepting set is given by $F = \{q \mid X_0 \in q\}$.

It remains to prove that A is asynchronous and equivalent to φ. Let $G = (V, E, \lambda)$ be an ℓ-bit labeled digraph and $\vec{U} = (U_0, \dots, U_k) \in (2^V)^{k+1}$ be the least fixpoint of the operator f associated with $(\varphi_0, \dots, \varphi_k)$. Due to the asynchrony condition, we must consider an arbitrary timing $\tau = (\tau_1, \tau_2, \dots)$ of G. The corresponding run $\rho = (\rho_0, \rho_1, \dots)$ of A on G timed by τ engenders an infinite sequence $(\vec{W}^0, \vec{W}^1, \dots)$, where each tuple $\vec{W}^t = (W_0^t, \dots, W_k^t) \in (2^V)^{k+1}$ specifies the valuation of every variable X_i at time t, i.e., $W_i^t = \{v \in V \mid X_i \in \rho_t(v)\}$. Since A is quasi-acyclic and V is finite, this sequence must eventually stabilize at some value $\vec{W}^\infty$, and each node accepts if and only if it belongs to W_0^∞. Reformulated this way, our task is to demonstrate that $\vec{W}^\infty$ equals $\vec{U}$, regardless of the timing τ.

"$\vec{W}^\infty \subseteq \vec{U}$": We show by induction that $\vec{W}^t \subseteq \vec{U}$ for all $t \in \mathbb{N}$. This obviously holds for $t = 0$, since $\vec{W}^0 = (\emptyset, \dots, \emptyset)$. Now, consider any node $v \in V$ at an arbitrary time t. Let q be the current state of v and S be the set of current states of its incoming neighbors. Depending

on τ, it might be the case that v actually receives some outdated information S' instead of S. However, given that the neighbors' previous states cannot contain more variables than their current ones (by construction), and that variables can only occur positively in each φ_i, we know that $(q, S') \models \varphi_i$ implies $(q, S) \models \varphi_i$. Hence, if v performs a local transition at time t, then the only new variables that can be added to its state must lie in $\{X_i \mid (q, S) \models \varphi_i\}$. On a global scale, this means that $\vec{W}^{t+1} \setminus \vec{W}^t \subseteq f(\vec{W}^t)$. Furthermore, by the induction hypothesis, the monotonicity of f, and the fact that $\vec{U}$ is a fixpoint, we have $f(\vec{W}^t) \subseteq f(\vec{U}) = \vec{U}$. Putting both together, and again relying on the induction hypothesis, we obtain $\vec{W}^{t+1} \subseteq \vec{U}$.

"$\vec{W}^\infty \supseteq \vec{U}$": For the converse direction, we make use of the Knaster-Tarski theorem, which gives us the equality $\vec{U} = \bigcap\{\vec{W} \in (2^V)^{k+1} \mid f(\vec{W}) \subseteq \vec{W}\}$. With this, it suffices to show that $f(\vec{W}^\infty) \subseteq \vec{W}^\infty$. Consider some time $t \in \mathbb{N}$ such that $\vec{W}^{t'} = \vec{W}^\infty$ for all $t' \geq t$. Although we know that every node has reached its final state at time t, the FIFO buffers of some edges might still contain obsolete states from previous times. However, the fairness property of τ guarantees that our customized popfirst operation is executed infinitely often at every edge, while the pushlast operation has no effect because all the states remain unchanged. Therefore, there must be a time $t' \geq t$ from which on each buffer contains only the current state of its incoming node, i.e., $\rho_{t''}(uv) = \rho_{t''}(u)$ for all $t'' \geq t'$ and $uv \in E$. Moreover, the fairness property of τ also ensures that every node v reevaluates the local transition function δ infinitely often, based on its own current state q and the set S of states in the buffers associated with its incoming neighbors. As this has no influence on v's state, we can deduce that $\{X_i \mid (q, S) \models \varphi_i\} \subseteq q$. Consequently, we have $f(\vec{W}^{t'}) \subseteq \vec{W}^{t'}$, which is equivalent to $f(\vec{W}^\infty) \subseteq \vec{W}^\infty$. ◀

5 Capturing asynchronous runs using least fixpoints

This section is dedicated to proving the converse direction of the main result, which will allow us to translate any quasi-acyclic lossless-asynchronous automaton into an equivalent formula of the backward μ-fragment (see Proposition 7). Our proof builds on two concepts: the invariance of distributed automata under backward bisimulation (stated in Proposition 5) and an ad-hoc relation "$\triangleright$" that captures the possible behaviors of a fixed lossless-asynchronous automaton A (in a specific sense described in Lemma 6).

We start with the notion of backward bisimulation, which is defined like the standard notion of bisimulation (see, e.g., [1, Def. 2.16] or [2, Def. 5]), except that edges are followed in the backward direction. Formally, a *backward bisimulation* between two ℓ-bit labeled digraphs $G = (V, E, \lambda)$ and $G' = (V', E', \lambda')$ is a binary relation $R \subseteq V \times V'$ that fulfills the following conditions for all $vv' \in R$:

1. $\lambda(v) = \lambda'(v')$,
2. if $uv \in E$, then there exists $u' \in V'$ such that $u'v' \in E'$ and $uu' \in R$, and, conversely,
3. if $u'v' \in E'$, then there exists $u \in V$ such that $uv \in E$ and $uu' \in R$.

We say that the pointed digraphs (G, v) and (G', v') are *backward bisimilar* if there exists such a backward bisimulation R relating v and v'. It is easy to see that distributed automata cannot distinguish between backward bisimilar structures:

▶ **Proposition 5.** *Distributed automata are invariant under backward bisimulation. That is, for every automaton A, if two pointed digraphs (G, v) and (G', v') are backward bisimilar, then A accepts (G, v) if and only if it accepts (G', v').*

Proof. Let R be a backward bisimulation between G and G' such that $vv' \in R$. Since acceptance is defined with respect to the synchronous behavior of the automaton, we need

only consider the synchronous runs $\rho = (\rho_0, \rho_1, \dots)$ and $\rho' = (\rho'_0, \rho'_1, \dots)$ of A on G and G', respectively. Now, given that the FIFO buffers on the edges of the digraphs merely contain the current state of their incoming node, it is straightforward to prove by induction on t that every pair of nodes $uu' \in R$ satisfies $\rho_t(u) = \rho'_t(u')$ for all $t \in \mathbb{N}$. ◀

We now turn to the mentioned relation "$\triangleright$", which is defined with respect to a fixed automaton. For the remainder of this section, let A denote an automaton (Q, δ_0, δ, F), and let $\mathfrak{Q}$ denote its set of traces. The relation $\triangleright \subseteq (2^{\mathfrak{Q}} \times \mathfrak{Q})$ specifies whether, in a lossless-asynchronous environment, a given trace σ can be traversed by a node whose incoming neighbors traverse the traces of a given set $\mathfrak{S}$. Loosely speaking, the intended meaning of $\mathfrak{S} \triangleright \sigma$ ("$\mathfrak{S}$ enables σ") is the following: Take an appropriately chosen digraph under some lossless-asynchronous timing τ, and observe the corresponding run of A up to a specific time t; if node v was initially in state $\sigma.\mathsf{first}$ and at time t it has *seen* its incoming neighbors traversing precisely the traces in $\mathfrak{S}$, then it is possible for τ to be such that at time t, node v has traversed exactly the trace σ. This relation can be defined inductively: As the base case, we specify that for every $q \in Q$ and $S \subseteq Q$, we have $S \triangleright q.\mathsf{pushlast}(\delta(q, S))$. For the inductive clause, consider a trace $\sigma \in \mathfrak{Q}$ and two finite (possibly equal) sets of traces $\mathfrak{S}, \mathfrak{S}' \subseteq \mathfrak{Q}$ such that the traces in $\mathfrak{S}'$ can be obtained by appending at most one state to the traces in $\mathfrak{S}$. More precisely, if $\pi \in \mathfrak{S}$, then $\pi.\mathsf{pushlast}(p) \in \mathfrak{S}'$ for some $p \in Q$, and conversely, if $\pi' \in \mathfrak{S}'$, then $\pi' = \pi.\mathsf{pushlast}(\pi'.\mathsf{last})$ for some $\pi \in \mathfrak{S}$. We shall denote this auxiliary relation by $\mathfrak{S} \rightrightarrows \mathfrak{S}'$. If it holds, then $\mathfrak{S} \triangleright \sigma$ implies $\mathfrak{S}' \triangleright \sigma.\mathsf{pushlast}(q)$, where $q = \delta(\sigma.\mathsf{last}, \{\pi'.\mathsf{last} \mid \pi' \in \mathfrak{S}'\})$.

The next step is to show (in Lemma 6) that our definition of "$\triangleright$" does indeed capture the intuition given above. To formalize this, we first introduce two further pieces of terminology.

First, the notions of configuration and run can be enriched to facilitate discussions about the past. Let $\rho = (\rho_0, \rho_1, \dots)$ be a run of A on a digraph $G = (V, E, \lambda)$ (timed by some timing τ). The corresponding *enriched run* is the sequence $\hat{\rho} = (\hat{\rho}_0, \hat{\rho}_1, \dots)$ of *enriched configurations* that we obtain from ρ by requiring each node to remember the entire trace it has traversed so far. Formally, for $t \in \mathbb{N}$, $v \in V$ and $e \in E$,

$$\hat{\rho}_0(v) = \rho_0(v), \qquad \hat{\rho}_{t+1}(v) = \hat{\rho}_t(v).\mathsf{pushlast}(\rho_{t+1}(v)) \qquad \text{and} \qquad \hat{\rho}_t(e) = \rho_t(e).$$

Second, we will need to consider finite segments of timings and enriched runs. A *lossless-asynchronous timing segment* of a digraph G is a finite sequence $\tau = (\tau_1, \dots, \tau_r)$ that could be extended to a whole lossless-asynchronous timing $(\tau_1, \dots, \tau_r, \tau_{r+1}, \dots)$. Likewise, for an initial enriched configuration $\hat{\rho}_0$ of G, the corresponding *enriched run segment* timed by τ is the sequence $(\hat{\rho}_0, \dots, \hat{\rho}_r)$, where each $\hat{\rho}_{t+1}$ is computed from $\hat{\rho}_t$ and τ_{t+1} in the same way as for an entire enriched run.

Equipped with the necessary terminology, we can now state and prove a (slightly technical) lemma that will allow us to derive benefit from the relation "$\triangleright$". This lemma essentially states that if $\mathfrak{S} \triangleright \sigma$ holds and we are given enough nodes that traverse the traces in $\mathfrak{S}$, then we can take those nodes as the incoming neighbors of a new node v and delay the messages received by v in such a way that v traverses σ, without losing any messages.

▶ **Lemma 6.** *For every trace $\sigma \in \mathfrak{Q}$ and every finite (possibly empty) set of traces $\mathfrak{S} = \{\pi_1, \dots, \pi_\ell\} \subseteq \mathfrak{Q}$ that satisfy the relation $\mathfrak{S} \triangleright \sigma$, there exist lower bounds $m_1, \dots, m_\ell \in \mathbb{N}_{>0}$ such that the following statement holds true:*

For any $n_1, \dots, n_\ell \in \mathbb{N}_{>0}$ satisfying $n_i \geq m_i$, let G be a digraph consisting of the nodes $(u_i^j)_{i,j}$ and v, and the edges $(u_i^j v)_{i,j}$, with index ranges $1 \leq i \leq \ell$ and $1 \leq j \leq n_i$. If we start from the enriched configuration $\hat{\rho}_0$ of G, where

$$\hat{\rho}_0(u_i^j) = \pi_i, \qquad \hat{\rho}_0(u_i^j v) = \pi_i \qquad \text{and} \qquad \hat{\rho}_0(v) = \sigma.\mathsf{first},$$

then we can construct a (nonempty) lossless-asynchronous timing segment $\tau = (\tau_1, \dots, \tau_r)$ *of* G*, where* $\tau_t(u_i^j) = 0$ *and* $\tau_t(v) = 1$ *for* $1 \le t \le r$*, such that the corresponding enriched run segment* $\hat{\rho} = (\hat{\rho}_0, \dots, \hat{\rho}_r)$ *timed by* τ *satisfies*

$$\hat{\rho}_{r-1}(u_i^j v) = \pi_i.\mathsf{last} \qquad \textit{and} \qquad \hat{\rho}_r(v) = \sigma.$$

Proof. We proceed by induction on the definition of "$\triangleright$". In the base case, where $\mathfrak{S} = \{p_1, \dots, p_\ell\} \subseteq Q$ and $\sigma = q.\mathsf{pushlast}(\delta(q, \mathfrak{S}))$ for some $q \in Q$, the statement holds with $m_1 = \cdots = m_\ell = 1$. This is witnessed by a timing segment $\tau = (\tau_1)$, where $\tau_1(u_i^j) = 0$, $\tau_1(v) = 1$, and $\tau_1(u_i^j v)$ can be chosen as desired.

For the inductive step, we assume that the statement holds for σ and $\mathfrak{S} = \{\pi_1, \dots, \pi_\ell\}$ with some values $m_1, \dots, m_\ell$. Now consider any other set of traces $\mathfrak{S}' = \{\pi'_1, \dots, \pi'_{\ell'}\}$ such that $\mathfrak{S} \rightrightarrows \mathfrak{S}'$, and let $\sigma' = \sigma.\mathsf{pushlast}(q)$, where $q = \delta(\sigma.\mathsf{last}, \{\pi'_k.\mathsf{last} \mid \pi'_k \in \mathfrak{S}'\})$. Since $\mathfrak{S} \triangleright \sigma$, we have $\mathfrak{S}' \triangleright \sigma'$. The remainder of the proof consists in showing that the statement also holds for σ' and $\mathfrak{S}'$ with some large enough integers $m'_1, \dots, m'_{\ell'}$. Let us fix $m'_k = \sum \{m_i \mid \pi_i.\mathsf{pushlast}(\pi'_k.\mathsf{last}) = \pi'_k\}$. (As there is no need to find minimal values, we opt for easy expressibility.)

Given any numbers $n'_1, \dots, n'_{\ell'}$ with $n'_k \ge m'_k$, we choose suitable values $n_1, \dots, n_\ell$ with $n_i \ge m_i$, and consider the corresponding digraph G described in the lemma. Because we have $\mathfrak{S} \rightrightarrows \mathfrak{S}'$, we can assign to each node u_i^j a state p_i^j such that $\pi_i.\mathsf{pushlast}(p_i^j) \in \mathfrak{S}'$. Moreover, provided our choice of $n_1, \dots, n_\ell$ was adequate, we can also ensure that for each $\pi'_k \in \mathfrak{S}'$, there are exactly n'_k nodes u_i^j such that $\pi_i.\mathsf{pushlast}(p_i^j) = \pi'_k$. (Note that nodes with distinct traces $\pi_i, \pi_{i'} \in \mathfrak{S}$ might be mapped to the same trace $\pi'_k \in \mathfrak{S}'$, in case $\pi_{i'} = \pi_i p_i^j$.) It is straightforward to verify that such a choice of numbers and such an assignment of states are always possible, given the lower bounds $m'_1, \dots, m'_{\ell'}$ specified above.

Let us now consider the lossless-asynchronous timing segment $\tau = (\tau_1, \dots, \tau_r)$ and the corresponding enriched run segment $\hat{\rho} = (\hat{\rho}_0, \dots, \hat{\rho}_r)$ provided by the induction hypothesis. Since the $\mathsf{popfirst}$ operation has no effect on a trace of length 1, we may assume without loss of generality that $\tau_t(u_i^j v) = 0$ if $\hat{\rho}_{t-1}(u_i^j v)$ has length 1, for $t < r$. Consequently, if we start from the alternative enriched configuration $\hat{\rho}'_0$, where

$$\hat{\rho}'_0(u_i^j) = \pi_i.\mathsf{pushlast}(p_i^j), \qquad \hat{\rho}'_0(u_i^j v) = \pi_i.\mathsf{pushlast}(p_i^j) \qquad \text{and} \qquad \hat{\rho}'_0(v) = \sigma.\mathsf{first},$$

then the corresponding enriched run segment $(\hat{\rho}'_0, \dots, \hat{\rho}'_r)$ timed by τ can be derived from $\hat{\rho}$ by simply applying "$\mathsf{pushlast}(p_i^j)$" to $\hat{\rho}_t(u_i^j)$ and $\hat{\rho}_t(u_i^j v)$, for $t < r$. We thus get

$$\hat{\rho}'_{r-1}(u_i^j v) = \pi_i.\mathsf{last}.\mathsf{pushlast}(p_i^j) \qquad \text{and} \qquad \hat{\rho}'_r(v) = \sigma.$$

We may also assume without loss of generality that $\tau_r(u_i^j v) = 1$ if $\hat{\rho}'_{r-1}(u_i^j v)$ has length 2, since this does not affect $\hat{\rho}$ and lossless-asynchrony is ensured by $\tau_r(v) = 1$. Hence, it suffices to extend τ by an additional map τ_{r+1}, where $\tau_{r+1}(u_i^j) = 0$, $\tau_{r+1}(v) = 1$, and $\tau_{r+1}(u_i^j v)$ can be chosen as desired. The resulting enriched run segment $(\hat{\rho}'_0, \dots, \hat{\rho}'_{r+1})$ satisfies

$$\hat{\rho}'_r(u_i^j v) = p_i^j = \pi'_k.\mathsf{last} \ \text{ (for some } \pi'_k \in \mathfrak{S}') \qquad \text{and} \qquad \hat{\rho}'_{r+1}(v) = \sigma.\mathsf{pushlast}(q) = \sigma'. \quad \blacktriangleleft$$

Finally, we can put the pieces together and prove the converse direction of Theorem 3:

▶ **Proposition 7** ($\mathrm{QLA} \subseteq \Sigma_1^{\overline{\mu}}$)**.** *For every quasi-acyclic lossless-asynchronous automaton, we can effectively construct an equivalent formula of the backward* μ*-fragment.*

Proof. Assume that $A = (Q, \delta_0, \delta, F)$ is a quasi-acyclic lossless-asynchronous automaton with ℓ-bit input. Since it is quasi-acyclic, its set of traces $\mathfrak{Q}$ is finite, and thus we can afford

to introduce a separate propositional variable X_σ for each trace $\sigma \in \mathfrak{Q}$. Making use of the relation "$\triangleright$", we convert A into an equivalent formula $\varphi = \mu\big[X_0, (X_\sigma)_{\sigma\in\mathfrak{Q}}\big].\big[\varphi_0, (\varphi_\sigma)_{\sigma\in\mathfrak{Q}}\big]$ of the backward μ-fragment, where

$$\varphi_0 = \bigvee_{\substack{\sigma\in\mathfrak{Q}\\ \sigma.\mathsf{last}\in F}} X_\sigma, \tag{a}$$

$$\varphi_q = \bigvee_{\substack{x\in\mathfrak{D}^\ell\\ \delta_0(x)=q}} \Big(\bigwedge_{x(i)=1} P_i \wedge \bigwedge_{x(i)=0} \neg P_i \Big) \qquad \text{for each } q \in Q, \quad \text{and} \tag{b}$$

$$\varphi_\sigma = X_{\sigma.\mathsf{first}} \wedge \bigvee_{\substack{\mathfrak{S}\subseteq\mathfrak{Q}\\ \mathfrak{S}\triangleright\sigma}} \Big(\big(\bigwedge_{\pi\in\mathfrak{S}} \bar{\Diamond} X_\pi\big) \wedge \big(\bar{\Box} \bigvee_{\pi\in\mathfrak{S}} X_\pi\big) \Big) \qquad \text{for each } \sigma \in \mathfrak{Q} \text{ with } |\sigma| \geq 2. \tag{c}$$

Note that this formula can be constructed effectively because an inductive computation of "$\triangleright$" must terminate after at most $|\mathfrak{Q}| \cdot 2^{|\mathfrak{Q}|}$ iterations.

To prove that φ is indeed equivalent to A, let us consider an arbitrary ℓ-bit labeled digraph $G = (V, E, \lambda)$ and the corresponding least fixpoint $\vec{U} = (U_0, (U_\sigma)_{\sigma\in\mathfrak{Q}}) \in (\mathfrak{D}^V)^{|\mathfrak{Q}|+1}$ of the operator f associated with $(\varphi_0, (\varphi_\sigma)_{\sigma\in\mathfrak{Q}})$.

The easy direction is to show that for all nodes $v \in V$, if A accepts (G, v), then (G, v) satisfies φ. For that, it suffices to consider the synchronous enriched run $\hat{\rho} = (\hat{\rho}_0, \hat{\rho}_1, \dots)$ of A on G. (Any other run timed by a lossless-asynchronous timing would exhibit the same acceptance behavior.) As in the proof of Proposition 5, we can simply ignore the FIFO buffers on the edges of G because $\hat{\rho}_t(uv) = \hat{\rho}_t(u).\mathsf{last}$. Using this, a straightforward induction on t shows that every node $v \in V$ satisfies $\{\hat{\rho}_t(u) \mid uv \in E\} \triangleright \hat{\rho}_{t+1}(v)$ for all $t \in \mathbb{N}$. (For $t = 0$, the claim follows from the base case of the definition of "$\triangleright$""; for the step from t to $t+1$, we can immediately apply the inductive clause of the definition.) This in turn allows us to prove that each node v is contained in all the components of $\vec{U}$ that correspond to a trace traversed by v in $\hat{\rho}$, i.e., $v \in U_{\hat{\rho}_t(v)}$ for all $t \in \mathbb{N}$. Naturally, we proceed again by induction: For $t = 0$, we have $\hat{\rho}_0(v) = \delta_0(\lambda(v)) \in Q$, hence the subformula $\varphi_{\hat{\rho}_0(v)}$ defined in equation (b) holds at v, and thus $v \in U_{\hat{\rho}_0(v)}$. For the step from t to $t+1$, we need to distinguish two cases. If $\hat{\rho}_{t+1}(v)$ is of length 1, then it is equal to $\hat{\rho}_t(v)$, and there is nothing new to prove. Otherwise, we must consider the appropriate subformula $\varphi_{\hat{\rho}_{t+1}(v)}$ given by equation (c). We already know from the base case that the conjunct $X_{\hat{\rho}_{t+1}(v).\mathsf{first}} = X_{\hat{\rho}_0(v)}$ holds at v, with respect to any variable assignment that interprets each X_σ as U_σ. Furthermore, by the induction hypothesis, $X_{\hat{\rho}_t(u)}$ holds at every incoming neighbor u of v. Since $\{\hat{\rho}_t(u) \mid uv \in E\} \triangleright \hat{\rho}_{t+1}(v)$, we conclude that the second conjunct of $\varphi_{\hat{\rho}_{t+1}(v)}$ must also hold at v, and thus $v \in U_{\hat{\rho}_{t+1}(v)}$. Finally, assuming A accepts (G, v), we know by definition that $\hat{\rho}_t(v).\mathsf{last} \in F$ for some $t \in \mathbb{N}$. Since $v \in U_{\hat{\rho}_t(v)}$, this implies that the subformula φ_0 defined in equation (a) holds at v, and therefore that (G, v) satisfies φ.

For the converse direction of the equivalence, we have to overcome the difficulty that φ is more permissive than A, in the sense that a node v might lie in U_σ, and yet not be able to follow the trace σ under any timing of G. Intuitively, the reason why we still obtain an equivalence is that A cannot take advantage of all the information provided by any particular run, because it must ensure that for *all* digraphs, its acceptance behavior is independent of the timing. It turns out that even if v cannot traverse σ, some other node v' in an indistinguishable digraph will be able to do so. More precisely, we will show that

> if $v \in U_\sigma$, then there exists a pointed digraph (G', v'), backward bisimilar to (G, v), and a lossless-asynchronous timing τ' of G', such that $\hat{\rho}'_t(v') = \sigma$ for some $t \in \mathbb{N}$, (∗)

where $\hat{\rho}'$ is the enriched run of A on G' timed by τ'. Now suppose that (G, v) satisfies φ. By

equation (a), this means that $v \in U_\sigma$ for some trace σ such that $\sigma.\mathsf{last} \in F$. Consequently, A accepts the pointed digraph (G', v') postulated in $(*)$, based on the claim that v' traverses σ under timing τ' and the fact that A is lossless-asynchronous. Since (G, v) and (G', v') are backward bisimilar, it follows from Proposition 5 that A also accepts (G, v).

It remains to verify $(*)$. We achieve this by computing the least fixpoint $\vec{U}$ inductively and proving the statement by induction on the sequence of approximants $(\vec{U}^0, \vec{U}^1, \dots)$. Note that we do not need to consider the limit case, since $\vec{U} = \vec{U}^n$ for some $n \in \mathbb{N}$.

The base case is trivially true because all the components of $\vec{U}^0$ are empty. Furthermore, if σ consists of a single state q, then we do not even need to argue by induction, as it is evident from equation (b) that for all $j \geq 1$, node v lies in U_q^j precisely when $\delta_0(\lambda(v)) = q$. It thus suffices to set $(G', v') = (G, v)$ and choose the timing τ' arbitrarily. Clearly, we have $\hat{\rho}'_0(v') = \delta_0(\lambda(v)) = q$ if $v \in U_q^j$.

On the other hand, if σ is of length at least 2, we must assume that statement $(*)$ holds for the components of $\vec{U}^j$ in order to prove it for U_σ^{j+1}. To this end, consider an arbitrary node $v \in U_\sigma^{j+1}$. By the first conjunct in (c) and the preceding remarks regarding the trivial cases, we know that $\delta_0(\lambda(v)) = \sigma.\mathsf{first}$ (and incidentally that $j \geq 1$). Moreover, the second conjunct ensures the existence of a (possibly empty) set of traces $\mathfrak{S}$ that satisfies $\mathfrak{S} \triangleright \sigma$ and that represents a "projection" of v's incoming neighborhood at stage j. By the latter we mean that for all $\pi \in \mathfrak{S}$, there exists $u \in V$ such that $uv \in E$ and $u \in U_\pi^j$, and conversely, for all $u \in V$ with $uv \in E$, there exists $\pi \in \mathfrak{S}$ such that $u \in U_\pi^j$.

Now, for each trace $\pi \in \mathfrak{S}$ and each incoming neighbor u of v that is contained in U_π^j, the induction hypothesis provides us with a pointed digraph $(G'_{u:\pi}, u'_\pi)$ and a corresponding timing $\tau'_{u:\pi}$, as described in $(*)$. We make $n_{u:\pi} \in \mathbb{N}$ distinct copies of each such digraph $G'_{u:\pi}$. From this, we construct $G' = (V', E', \lambda')$ by taking the disjoint union of all the $\sum n_{u:\pi}$ digraphs, and adding a single new node v' with $\lambda'(v') = \lambda(v)$, together with all the edges of the form $u'_\pi v'$ (i.e., one such edge for each copy of every u'_π). Given that every $(G'_{u:\pi}, u'_\pi)$ is backward bisimilar to (G, u), we can guarantee that the same holds for (G', v') and (G, v) by choosing the numbers of digraph copies in G' such that each incoming neighbor u of v is represented by at least one incoming neighbor of v'. That is, for every u, we require that $n_{u:\pi} \geq 1$ for some π.

Finally, we construct a suitable lossless-asynchronous timing τ' of G', which proceeds in two phases to make v' traverse σ in the corresponding enriched run $\hat{\rho}'$. In the first phase, where $0 < t \leq t_1$, node v' remains inactive, which means that every τ_t assigns 0 to v' and its incoming edges. The state of v' at time t_1 is thus still $\sigma.\mathsf{first}$. Meanwhile, in every copy of each digraph $G'_{u:\pi}$, the nodes and edges behave according to timing $\tau'_{u:\pi}$ until the respective copy of u'_π has completely traversed π, whereupon the entire subgraph becomes inactive. By choosing t_1 large enough, we make sure that the FIFO buffer on each edge of the form $u'_\pi v'$ contains precisely π at time t_1. In the second phase, which lasts from $t_1 + 1$ to t_2, the only active parts of G' are v' and its incoming edges. Since the number $n_{u:\pi}$ of copies of each digraph $G'_{u:\pi}$ can be chosen as large as required, we stipulate that for every trace $\pi \in \mathfrak{S}$, the sum of $n_{u:\pi}$ over all u exceeds the lower bound m_π that is associated with π when invoking Lemma 6 for σ and $\mathfrak{S}$. Applying that lemma, we obtain a lossless-asynchronous timing segment of the subgraph induced by v' and its incoming neighbors. This segment determines our timing τ' between $t_1 + 1$ and t_2 (the other parts of G' being inactive), and gives us $\hat{\rho}'_{t_2}(v') = \sigma$, as desired. Naturally, the remainder of τ', starting at $t_2 + 1$, can be chosen arbitrarily, so long as it satisfies the properties of a lossless-asynchronous timing.

As a closing remark, note that the pointed digraph (G', v') constructed above is very similar to the standard unraveling of (G, v) into a (possibly infinite) tree. (The set of nodes

of that tree-unraveling is precisely the set of all directed paths in G that start at v; see, e.g., [1, Def. 4.51] or [2, § 3.2]). However, there are a few differences: First, we do the unraveling backwards, because we want to generate a backward bisimilar structure, where all the edges point toward the root. Second, we may duplicate the incoming neighbors (i.e., children) of each node in the tree, in order to satisfy the lower bounds imposed by Lemma 6. Third, we stop the unraveling process at a finite depth (not necessarily the same for each subtree), and place a copy of the original digraph G at every leaf. ◀

Acknowledgments. I would like to thank Olivier Carton, my PhD supervisor, for many pleasant discussions and constructive comments.

References

1. Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 2002. doi:10.1017/CBO9781107050884.
2. Patrick Blackburn and Johan van Benthem. Modal logic: a semantic perspective. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 1–84. Elsevier, 2007. doi:10.1016/S1570-2464(07)80004-8.
3. Julian Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 721–756. Elsevier, 2007. doi:10.1016/S1570-2464(07)80015-2.
4. Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. doi:10.1007/3-540-68804-8.
5. Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. In Darek Kowalski and Alessandro Panconesi, editors, *ACM Symposium on Principles of Distributed Computing, PODC'12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 185–194. ACM, 2012. doi:10.1145/2332432.2332466.
6. Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Computing*, 28(1):31–53, 2015. doi:10.1007/s00446-013-0202-3.
7. Antti Kuusisto. Modal logic and distributed message passing automata. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 452–468. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.452.
8. Antti Kuusisto. Infinite networks, halting and local algorithms. In Adriano Peron and Carla Piazza, editors, *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014*, volume 161 of *EPTCS*, pages 147–160, 2014. doi:10.4204/EPTCS.161.14.
9. Giacomo Lenzi. The modal μ-calculus: a survey. *TASK Quarterly – Scientific Bulletin of the Academic Computer Centre in Gdansk*, 9(3):293–316, 2005. URL: http://task.gda.pl/quart/05-3.html.
10. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

11 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*, volume 5 of *SIAM Monographs on Discrete Mathematics and Applications*. Society for Industrial and Applied Mathematics (SIAM), 2000. doi:10.1137/1.9780898719772.

12 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013. doi:10.1145/2431211.2431223.