Synthesis in Presence of Dynamic Links

Béatrice Bérard[1], Benedikt Bollig[2], Patricia Bouyer[2],
Matthias Függer[2,3], and Nathalie Sznajder[1]

[1]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[2]CNRS & LSV, ENS Paris-Saclay, Université Paris-Saclay, France [3]Inria, France

The problem of distributed synthesis is to automatically generate a distributed algorithm, given a target communication network and a specification of the algorithm's correct behavior.

Previous work has focused on static networks with an a priori fixed message size. This approach has two shortcomings: Recent work in distributed computing is shifting towards dynamically changing communication networks rather than static ones, and an important class of distributed algorithms are so-called full-information protocols, where nodes piggy-pack previously received messages onto current messages.

In this work, we consider the synthesis problem for a system of two nodes communicating in rounds over a dynamic link whose message size is not bounded. Given a network model, i.e., a set of link directions, in each round of the execution, the adversary choses a link from the network model, restricted only by the specification, and delivers messages according to the current link's directions. Motivated by communication buses with direct acknowledge mechanisms, we further assume that nodes are aware of which messages have been delivered.

We show that the synthesis problem is decidable for a network model if and only if it does not contain the empty link that dismisses both nodes' messages.

1 Introduction

Starting from Church's work [12] on synthesizing circuits from arithmetic specifications in the 1960s, automatic synthesis of programs or circuits has been widely studied.

In the case of a reactive system, given a specification, the goal is to find an implementation for a system that repeatedly receives inputs from the environment and generates outputs such that the system's behavior adheres to the specification. Early work [34, 35, 37] was synthesizing algorithms that require knowledge of the complete system state, inherently yielding single-process solutions.

Single-process synthesis is related to finding a strategy for a player representing the process that has to win against the adversarial environment, and has been studied in the context of games [3,9,40] as well as with automata techniques [24, 35].

For systems with more than one process, different models for how communication and computation is organized have been studied. Their two extremes are message-triggered asynchronous computation [17, 28] and round-wise synchronous computation.

An example for the latter is the work by Pnueli and Rosner [36], who considered synchronous distributed systems with an a priori fixed communication network. In their model, the network is given by a directed communication graph, whose nodes are the processes and with a link from process p to q if p can send messages to q (or write to and read from a shared variable). Messages are from a fixed, finite alphabet per link. A solution to the synthesis problem is a distributed algorithm that operates in rounds, repeatedly reading inputs, exchanging messages, and setting outputs. Already the case of two processes with separate inputs and outputs, and without a communication link to each other, was shown

Submitted to:
GandALF 2020

to be undecidable for linear temporal logic (LTL) specifications [33] on the inputs and outputs. As a positive result, the paper presents a solution for unidirectional process chains.

Still in the case of static architectures and bounded messages, Kupferman and Vardi [**?**, 25] extended decidability results to branching time specifications and proved sufficient conditions on communication networks for decidability, while Finkbeiner and Schewe [16] presented a characterization of networks where synthesis is decidable. Since specifications are allowed to talk about messages, however, they are powerful enough to break existing communication links between processes, leading to undecidability like in the two-process system without communication [36]. Gastin *et al.* [18] proved a necessary and sufficient condition for decidability on a class of communication networks if specifications are only on inputs and outputs. Like [18], our work only allows "input-output" specifications, so that we obtain decidability in several cases where the framework of [16] does not allow it.

Like in the single-process scenario, synthesis in distributed systems can be modeled as a game, which, in this context, are partial information games played between a cooperating set of processes against the environment [8, 29, 31, 32]. With the exception of [8], all the above approaches assume static, reliable networks. In [8], Berwanger *et al.* study games in which information that players have about histories is hierarchically ordered, and this order may change dynamically during a play. The main difference to our work is that we consider a memory model where messages carry the complete *causal* history allowing for unbounded communication messages, while [8] is based on local observations so that, at every round, a bounded amount of information is transmitted between players. Further, while asynchronous solutions to the synthesis problem considered potentially unbounded messages [17, 28], previous synchronous solutions assume an a priori fixed message size. Also [28] assume that processes that communicate infinitely often encounter each other within a bounded number of steps.

The above assumptions have two shortcomings:

Modeling unreliability. Distributed computing has a long history of studying algorithms that provide services in presence of unstable or unreliable components [27]. Indeed, classical process and link failures can be treated as particular dynamic network behavior [11]. Early work by Akkoyunlu *et al.* [6] considered the problem of two groups of gangsters coordinating a coup despite an unreliable channel between both parties; later on generalized to the Byzantine generals problem [26]. Protocols like the Alternating Bit Protocol [7] aim at tolerating message loss between a sender and receiver node, and [5] studies optimal transmission rates over unreliable links. Afek *et al.* [4] discuss protocols that implement reliable links on top of unreliable links. Further, for algorithms that have to operate in dynamic networks, see, e.g., [10, 13, 22], network changes are the normal case rather than the exception.

Synthesis with unstable or faulty components has been studied by Velner and Rabinovich [42] for two player games in presence of information loss between the environment and the inputs of a process. The approach is restricted to a single process, however. Dimitrova and Finkbeiner [14] study synthesis of fault-tolerant distributed algorithms in synchronous, fully connected networks. Processes are partitioned into correct and faulty. It is assumed that at every round at least one process is correct and the output of a correct process must not depend on the local inputs of faulty processes. While unreliable links can be mapped to process failures, the above assumptions are a priori too restrictive to cover dynamic networks.

Modeling full-information protocols. An important class of distributed algorithms are full-information protocols, where nodes piggy-pack previously received messages onto current messages [15, 27]. By construction, such algorithms do not have bounded message size. This kind of causal memory has been considered in [17, 19, 20, 28] for synthesis and control of Zielonka automata over Mazurkiewicz traces with various objectives, ranging from local-state reachability to ω-branching behaviors. Zielonka automata usually model *asynchronous processes* (there is no global clock so that processes evolve at their own speed until they synchronize) and *symmetric communication* (whenever processes synchronize, they mutually exchange their complete history).

In this work we consider the synthesis problem for a system of two nodes communicating in synchronous rounds, where specifications are given as LTL formulas or, more generally, ω-regular languages. The nodes are connected via a dynamic link. As in [10, 13], a network is a set of communication graphs, called *network model*. A distributed algorithm operates in rounds as in [36], with the difference that the communication graph is chosen by an adversary per round. Motivated by communication buses, like the industry standard I^2C bus [1] and CAN bus [2], with direct acknowledge mechanisms after message transfers, we assume that nodes are aware if messages have been delivered successfully. In contrast to the Pnueli-Rosner setting, we suppose full-information protocols where processes have access to their causal history. That is, the dynamic links have unbounded message size. Unlike in Zielonka automata over traces, however, we consider *synchronous processes* and potentially *asymmetric communication*. In particular, the latter implies that a process may learn all about the other's history without revealing its own. Observe that, when restricting to Zielonka automata, synthesis of asynchronous distributed systems is *not* a generalization of the synchronous case.

We show that the synthesis problem is decidable for a network model if and only if it does not contain the empty link that dismisses both nodes' messages. As we assume that LTL specifications can not only reason about inputs and outputs, but also about the communication graph, our result covers synthesis for dynamic systems where links change in more restricted ways. In particular, this includes processes that do not send further messages after their message has been lost, bounded interval omission faults, etc.

Outline. We define the synthesis problem for the dynamic two-process model in Section 2. In Section 3, we discuss the asymmetric model where communication to process 1 never fails. Central to the analysis is to show that, despite the availability of unbounded communication links, finite-memory distributed algorithms actually suffice. We then prove that the synthesis problem is decidable (Theorem 2). In Section 5 we reduce the general case of dynamic communication to the asymmetric case, obtaining our main result of decidability in network models that do not contain the empty link (Theorem 1). We conclude in Section 6. Missing proofs can be found in the appendix.

2 The Synthesis Problem

We start with a few preliminaries. Let $\mathbb{N} = \{0,1,2,\ldots\}$. For a (possibly infinite) alphabet A, the set of finite words over A is denoted by A^*, the set of nonempty finite words by A^+, and the set of countably infinite words by A^ω. We let ε be the empty word and denote the concatenation of $w_1 \in A^*$ and $w_2 \in A^* \cup A^\omega$ by $w_1 \cdot w_2$ or simply $w_1 w_2$.

Fix the set of processes $P = \{1,2\}$. Every process $p \in P$ comes with fixed finite sets X_p and Y_p of possible *inputs* and *outputs*, respectively. We assume there are at least two possible inputs and outputs per process, i.e., $|X_p| \geq 2$ and $|Y_p| \geq 2$.

We consider systems where computation and communication proceed in rounds. In round $r = 0,1,2,\ldots$, process $p \in P$ receives an input $x_p^r \in X_p$ and it produces an output $y_p^r \in Y_p$. The decision on y_p^r depends on the knowledge that process p has about the execution up to round r. In addition to all local inputs $x_p^0, \ldots, x_p^r$, this knowledge can also include inputs of the other process, which may be communicated through communication links.

Following Charron-Bost *et al.* [10], we consider a dynamic communication topology in terms of a *network model*, i.e., a fixed nonempty set $\mathcal{N} \subseteq \{\times, \leftarrow, \rightarrow, \leftrightarrow\}$ of potentially occurring communica-

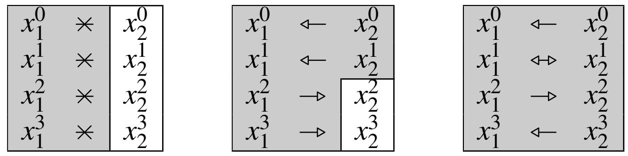

Figure 1: $[\![w]\!]_1$ for some histories w; the white part is unknown in the view, and replaced by $\perp$.

tion graphs. In round r, a graph $\rightleftharpoons^r \in \mathcal{N}$ is chosen non-deterministically with the following intuitive meaning:

- $\divideontimes$ No communication takes place. The knowledge of process p that determines y_p^r only includes the knowledge at round $r-1$ as well as the new input x_p^r.
- $\leftarrowtriangle$ Process 1 becomes aware of the whole input sequence $x_2^0 \dots x_2^r$ that process 2 has received so far. This includes x_2^r, which is transmitted without delay. The case $\rightarrowtriangle$ is analogous.
- $\leftrightarrowtriangle$ Both processes become aware of the whole input sequence of the other process.

As discussed in the introduction, the knowledge of process p at round r also includes the communication link $\rightleftharpoons^r$ at r, which is therefore common knowledge.

2.1 Histories and Views

Let us be more formal. Recall that we fixed the sets P, X_p, Y_p, and $\mathcal{N}$. We let $\Sigma = X_1 \times \mathcal{N} \times X_2$ be the set of *input signals*. For ease of notation, we write $\langle x_1 \rightleftharpoons x_2 \rangle$ instead of $(x_1, \rightleftharpoons, x_2) \in \Sigma$. Moreover, for $\rightleftharpoons \in \mathcal{N}$, we let $\Sigma_{\rightleftharpoons} = X_1 \times \{\rightleftharpoons\} \times X_2$. A word $w \in \Sigma^*$ represents a possible *history*, a sequence of signals to which the system has been exposed so far. For a process p, we inductively define the *view* $[\![w]\!]_p$ of p on w by replacing inputs that are invisible to p by the symbol $\perp$ (we suppose $\perp \notin X_1 \cup X_2$). First of all, let $[\![\varepsilon]\!]_1 = [\![\varepsilon]\!]_2 = \varepsilon$. Moreover, for $u \in \Sigma^*$:

$$
\begin{array}{rclcrcl}
[\![u\langle x_1 \leftrightarrowtriangle x_2\rangle]\!]_1 &=& u\langle x_1 \leftrightarrowtriangle x_2\rangle & \quad & [\![u\langle x_1 \leftrightarrowtriangle x_2\rangle]\!]_2 &=& u\langle x_1 \leftrightarrowtriangle x_2\rangle \\
[\![u\langle x_1 \leftarrowtriangle x_2\rangle]\!]_1 &=& u\langle x_1 \leftarrowtriangle x_2\rangle & & [\![u\langle x_1 \rightarrowtriangle x_2\rangle]\!]_2 &=& u\langle x_1 \rightarrowtriangle x_2\rangle \\
[\![u\langle x_1 \rightarrowtriangle x_2\rangle]\!]_1 &=& [\![u]\!]_1\langle x_1 \rightarrowtriangle \perp\rangle & & [\![u\langle x_1 \leftarrowtriangle x_2\rangle]\!]_2 &=& [\![u]\!]_2\langle \perp \leftarrowtriangle x_2\rangle \\
[\![u\langle x_1 \divideontimes x_2\rangle]\!]_1 &=& [\![u]\!]_1\langle x_1 \divideontimes \perp\rangle & & [\![u\langle x_1 \divideontimes x_2\rangle]\!]_2 &=& [\![u]\!]_2\langle \perp \divideontimes x_2\rangle
\end{array}
$$

With this, we let $\mathit{Views}_1 = \{[\![w]\!]_1 \mid w \in \Sigma^+\}$ and $\mathit{Views}_2 = \{[\![w]\!]_2 \mid w \in \Sigma^+\}$ be the sets of possible *views* of processes 1 and 2.

The view $[\![w]\!]_1$ is illustrated in Figure 1 for three different words w. For the history in the middle, we have $[\![\langle x_1^0 \leftarrowtriangle x_2^0\rangle\langle x_1^1 \leftarrowtriangle x_2^1\rangle\langle x_1^2 \rightarrowtriangle x_2^2\rangle\langle x_1^3 \rightarrowtriangle x_2^3\rangle]\!]_1 = \langle x_1^0 \leftarrowtriangle x_2^0\rangle\langle x_1^1 \leftarrowtriangle x_2^1\rangle\langle x_1^2 \rightarrowtriangle \perp\rangle\langle x_1^3 \rightarrowtriangle \perp\rangle$.

2.2 Linear-Time Temporal Logic

Let $\Omega = Y_1 \times Y_2$ be the set of *output signals*. An *execution* is a word from $(\Sigma \times \Omega)^\omega$, which records, apart from the input signals, the outputs at every round. A convenient specification language to define the *valid* system executions is *linear-time temporal logic* (LTL) interpreted over words from $(\Sigma \times \Omega)^\omega$. The logic can, therefore, talk about inputs, outputs, and communication links at a given position. Moreover, it has the usual temporal modalities. Formally, the set LTL($\mathcal{N}$) of LTL formulas is given by the grammar

$$\begin{aligned} \varphi ::= \;& (in_p = x) \mid (out_p = y) \mid (link = \rightleftharpoons) \mid && \text{atomic formulas} \\ & \mathsf{X}\varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi \mid \varphi \mathsf{U} \varphi \mid && \text{temporal modalities} \\ & \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \Longrightarrow \varphi \mid \varphi \Longleftrightarrow \varphi && \text{Boolean connectives} \end{aligned}$$

where $p \in P$, $x \in X_p$, $y \in Y_p$, and $\rightleftharpoons \in \mathcal{N}$. Let $e = \alpha_0\alpha_1\alpha_2\ldots$ be an execution with $\alpha_i \in \Sigma \times \Omega$ for all $i \in \mathbb{N}$ and $\alpha_0 = \big(\langle x_1^0 \rightleftharpoons^0 x_2^0\rangle, (y_1^0, y_2^0)\big)$. For $r \in \mathbb{N}$, let $e_{\geq r}$ denote its suffix $\alpha_r\alpha_{r+1}\alpha_{r+2}\ldots$, i.e., $e = e_{\geq 0}$. Boolean connectives are interpreted as usual. Moreover:

$$\begin{array}{llll} e \models (in_p = x) & \text{if } x_p^0 = x & e \models \mathsf{X}\varphi & \text{if } e_{\geq 1} \models \varphi \\ e \models (out_p = y) & \text{if } y_p^0 = y & e \models \mathsf{F}\varphi & \text{if } \exists r \geq 0 : e_{\geq r} \models \varphi \\ e \models (link = \rightleftharpoons) & \text{if } \rightleftharpoons^0 = \rightleftharpoons & e \models \mathsf{G}\varphi & \text{if } \forall r \geq 0 : e_{\geq r} \models \varphi \\ \multicolumn{4}{l}{e \models \varphi \mathsf{U} \psi \text{ if } \exists r \geq 0 : \big(e_{\geq r} \models \psi \wedge \forall 0 \leq r' < r : e_{\geq r'} \models \varphi\big)} \end{array}$$

Finally, we let $L(\varphi) = \{e \in (\Sigma \times \Omega)^\omega \mid e \models \varphi\}$ be the set of executions that satisfy φ.

Remark 1. In general, the sequence of communication graphs in an execution is arbitrary from $\mathcal{N}^\omega$, modeling a highly dynamic network without any restrictions on stability, eventual convergence, etc. Note that the specification is allowed to speak about the communication links along a history, however, with the possibility to restrict the behavior of the dynamic network and impose process behavior to depend on the network dynamics.

Example 1. Suppose $X_1 = X_2 = Y_1 = Y_2 = \{\mathbb{0}, \mathbb{1}\}$ and $\mathcal{N} = \{\leftarrow, \rightarrow\}$. Consider

$$\begin{aligned} \varphi_1 &= \mathsf{G}\big((out_1 = \mathbb{1}) \iff (out_2 = \mathbb{1})\big) \\ \varphi_2 &= \mathsf{GF}\big((in_1 = \mathbb{1}) \wedge (in_2 = \mathbb{1})\big) \iff \mathsf{GF}\big((out_1 = \mathbb{1}) \wedge (out_2 = \mathbb{1})\big) \\ \psi &= \big(\mathsf{GF}(link = \leftarrow) \wedge \mathsf{GF}(link = \rightarrow)\big) \implies \varphi_1 \wedge \varphi_2 \,. \end{aligned}$$

Formula φ_1 says that, in each round, both processes agree on their output. Formula φ_2 postulates that both processes simultaneously output $\mathbb{1}$ infinitely often if, and only if, both inputs are simultaneously $\mathbb{1}$ infinitely often. Finally, ψ requires φ_1 and φ_2 to hold if both communication links occur infinitely often. We will come back to these formulas later to illustrate the synthesis problem. $\lhd$

2.3 Synthesis Problem

A *distributed algorithm* is a pair $f = (f_1, f_2)$ of functions $f_1 : \mathit{Views}_1 \to Y_1$ and $f_2 : \mathit{Views}_2 \to Y_2$ that associate with each view an output. Given $w = \sigma_0\sigma_1\sigma_2\ldots \in \Sigma^\omega$, we define the execution $f(\!|w|\!) = \big(\sigma_0, (y_1^0, y_2^0)\big)\big(\sigma_1, (y_1^1, y_2^1)\big)\ldots \in (\Sigma \times \Omega)^\omega$ where $y_p^r = f_p([\![\sigma_0 \ldots \sigma_r]\!]_p)$. For a finite word $w \in \Sigma^*$, we define $f(\!|w|\!) \in (\Sigma \times \Omega)^*$ similarly (in particular, $f(\!|\varepsilon|\!) = \varepsilon$).

Let $L \subseteq (\Sigma \times \Omega)^\omega$ and $\varphi \in \text{LTL}(\mathcal{N})$. We say that f *fulfills* L (respectively φ) if, for all $w \in \Sigma^\omega$, we have $f(\!|w|\!) \in L$ (respectively $f(\!|w|\!) \in L(\varphi)$). Moreover, we say that L (respectively φ) is *realizable* if there is some distributed algorithm that fulfills L (respectively φ).

We are now ready to define our main decision problem:

Definition 1. For a fixed network model $\mathcal{N}$ (recall that we also fixed P, X_p, Y_p), the *synthesis problem* SYNTHESIS($\mathcal{N}$) is defined as follows:

Input: $\varphi \in \text{LTL}(\mathcal{N})$

Question: Is φ realizable?

Example 2. Consider the formulas $\varphi_1, \varphi_2, \psi$ from Example 1 over $\mathcal{N} = \{\leftarrow, \rightarrow\}$. We easily see that φ_1 is realizable by the distributed algorithm where both processes always output $\mathbb{1}$. However, $\varphi_1 \wedge \varphi_2$ is *not* realizable: if the communication link is always $\leftarrow$ (an analogous argument holds for $\rightarrow$), process 2 has no information about any of the inputs of process 1. Thus, it is impossible for the processes to agree on their outputs in every round while respecting φ_2.

Finally, formula ψ is realizable. We can now assume that both $\leftarrow$ and $\rightarrow$ occur infinitely often. A sequence of signals can be divided into maximal finite blocks with identical communication links as illustrated in Figure 2 for the prefix of an execution. The distributed algorithm proceeds as follows. By default, both processes ouput $\mathbb{0}$, with the following exception: at the first position of each block, a process outputs $\mathbb{1}$ if, and only if, the preceding block contains a round where both processes simultaneously received $\mathbb{1}$. Note that this preceding block is entirely contained in the view of both processes. The algorithm's outputs are illustrated in Figure 2. At rounds 4 and 6, they are $\mathbb{1}$ because the corresponding preceding blocks contain an input pair of $\mathbb{1}$'s. As every block has finite size, satisfaction of φ_2 is guaranteed. ◁

round		signal			
0	$\mathbb{0}$	$\mathbb{0}$	$\leftarrow$	$\mathbb{1}$	$\mathbb{0}$
1	$\mathbb{0}$	$\mathbb{1}$	$\rightarrow$	$\mathbb{0}$	$\mathbb{0}$
2	$\mathbb{0}$	$\mathbb{1}$	$\rightarrow$	$\mathbb{1}$	$\mathbb{0}$
3	$\mathbb{0}$	$\mathbb{0}$	$\rightarrow$	$\mathbb{0}$	$\mathbb{0}$
4	$\mathbb{1}$	$\mathbb{1}$	$\leftarrow$	$\mathbb{0}$	$\mathbb{1}$
5	$\mathbb{0}$	$\mathbb{1}$	$\leftarrow$	$\mathbb{1}$	$\mathbb{0}$
6	$\mathbb{1}$	$\mathbb{0}$	$\rightarrow$	$\mathbb{0}$	$\mathbb{1}$
7	$\mathbb{0}$	$\mathbb{0}$	$\leftarrow$	$\mathbb{1}$	$\mathbb{0}$
8	$\mathbb{0}$	$\mathbb{1}$	$\leftarrow$	$\mathbb{0}$	$\mathbb{0}$
9	$\mathbb{0}$	$\mathbb{1}$	$\rightarrow$	$\mathbb{1}$	$\mathbb{0}$

Figure 2: Fulfilling ψ

It is well known that the synthesis problem is undecidable if processes are not connected:

Fact 1 (Pnueli-Rosner)**.** *The problem* SYNTHESIS$(\{\times\})$ *is undecidable.*

One also observes that undecidability of the synthesis problem is upward-closed:

Lemma 1. *Let $\mathcal{N}_1 \subseteq \mathcal{N}_2$. If* SYNTHESIS$(\mathcal{N}_1)$ *is undecidable, then so is* SYNTHESIS$(\mathcal{N}_2)$.

Indeed, formula $\varphi_1 \in \mathrm{LTL}(\mathcal{N}_1)$ is realizable iff formula $\varphi_2 \in \mathrm{LTL}(\mathcal{N}_2)$ is realizable where we let $\varphi_2 = \big(\mathsf{G} \bigvee_{\rightleftharpoons \in \mathcal{N}_1} (link = \rightleftharpoons)\big) \Longrightarrow \varphi_1$.

Therefore, we will now focus on network models that do not contain $\times$. Our main result is the following:

Theorem 1. *For a network model $\mathcal{N}$,* SYNTHESIS$(\mathcal{N})$ *is decidable if and only if $\times \notin \mathcal{N}$.*

The "only if" direction follows from Fact 1 and Lemma 1. The rest of the paper is devoted to the proof of the "if" direction of Theorem 1. We will first consider $\mathcal{N} = \{\leftrightarrow, \leftarrow\}$ and then reduce the other cases to this particular network model. By Lemma 1, it is enough to do this reduction for $\{\leftrightarrow, \leftarrow, \rightarrow\}$.

3 Finite-Memory Distributed Algorithms for $\mathcal{N} = \{\leftrightarrow, \leftarrow\}$

In this section, we suppose $\mathcal{N} = \{\leftrightarrow, \leftarrow\}$. We show that, in this case, synthesis is decidable:

Theorem 2. *The problem* SYNTHESIS$(\{\leftrightarrow, \leftarrow\})$ *is decidable (in 4-fold exponential time).*

As our setting features a dynamic architecture and unbounded message size in terms of causal histories, the proof of the theorem requires some new techniques. In particular, we cannot apply the information-fork criterion from [16], since our specifications can only describe the link between the processes, and cannot constrain the contents of the messages.

The proof is spread over the remainder of this section as well as Section 4. It crucially relies on the fact that, for every realizable specification φ, there is a distributed algorithm with a sort of *finite memory* fulfilling it (as shown in this section). This allows us to reduce, in Section 4, the problem of finding a distributed algorithm to finding a winning strategy in a decidable game (that we will call a $(2,1)$-player game thereafter) involving two cooperating players, where one player has imperfect information, and an antagonistic environment.

Remark 2. For the sake of technical simplification, we assume in Sections 3 and 4, without loss of generality, that input sequences start with a symbol from $\Sigma_{\leftrightarrow} = X_1 \times \{\leftrightarrow\} \times X_2$. Instead of the original formula $\hat{\varphi}$, we then simply take $\varphi = \mathsf{X}\hat{\varphi}$. That is, we can henceforth consider that $Views_1 = \{[\![w]\!]_1 \mid w \in \Sigma_{\leftrightarrow}\Sigma^*\}$ and $Views_2 = \{[\![w]\!]_2 \mid w \in \Sigma_{\leftrightarrow}\Sigma^*\}$, and that a distributed algorithm f *fulfills* $\varphi \in \mathrm{LTL}(\mathscr{N})$ if, for all $w \in \Sigma_{\leftrightarrow}\Sigma^\omega$, we have $f(\!|w|\!) \in L(\varphi)$.

3.1 Finite-Memory Distributed Algorithms

Deterministic Rabin Word Automata. Our decidability proof and the definition of a finite-memory distributed algorithm rely on deterministic Rabin word automata (cf. [39]):

Definition 2. A *deterministic Rabin word automaton (DRWA)* over a finite alphabet A is a tuple $\mathscr{A} = (S, \iota, \delta, \mathscr{F})$, where S is a finite set of states, $\iota \in S$ is the *initial state*, $\delta : S \times A \to S$ is the transition function, and $\mathscr{F} \subseteq 2^S \times 2^S$ is the (Rabin) acceptance condition.

The DRWA $\mathscr{A}$ defines a language of infinite words $L(\mathscr{A}) \subseteq A^\omega$ as follows. We extend δ to a function $\delta : S \times A^* \to S$ letting $\delta(s,\varepsilon) = s$ and $\delta(s,aw) = \delta(\delta(s,a),w)$. Let $w = a_0a_1a_2\ldots \in A^\omega$. We define $Visit^\infty_{\mathscr{A}}(w) = \{s \in S \mid s = \delta(\iota, a_0 \ldots a_i)$ for infinitely many $i \in \mathbb{N}\}$. We say that w is *accepted* by $\mathscr{A}$ if there is $(F,F') \in \mathscr{F}$ such that $Visit^\infty_{\mathscr{A}}(w) \cap F \neq \emptyset$ and $Visit^\infty_{\mathscr{A}}(w) \cap F' = \emptyset$, i.e., some state of F is visited infinitely often, whereas all states from F' are visited only finitely often. We let $L(\mathscr{A}) = \{w \in A^\omega \mid w$ is accepted by $\mathscr{A}\}$.

Existence of Finite-Memory Distributed Algorithms. We are now ready to state that, if there is a distributed algorithm that fulfills a specification $\varphi \in \mathrm{LTL}(\mathscr{N})$, then there is also a distributed algorithm f with finite "synchronization memory" in the following sense: There is a DRWA $\mathscr{A}$ over $\Sigma \times \Omega$ such that the output of a process for a history wu with $u \in \Sigma_{\leftrightarrow}\Sigma^*_{\leftarrow}$ only depends on u and the state that $\mathscr{A}$ reaches after reading $f(\!|w|\!)$. Let $\Sigma_{\perp\leftarrow} = \{\perp\} \times \{\leftarrow\} \times X_2$.

Lemma 2. *Let $\varphi \in \mathrm{LTL}(\mathscr{N})$. There is a DRWA $\mathscr{A} = (S, \iota, \delta, \mathscr{F})$, with $\delta : S \times (\Sigma \times \Omega) \to S$, such that the following are equivalent:*

(1) There is a distributed algorithm $f = (f_1, f_2)$ that fulfills φ.

(2) There is a distributed algorithm $f = (f_1, f_2)$ that fulfills φ and such that, for all words $w, w' \in \{\varepsilon\} \cup \Sigma_{\leftrightarrow}\Sigma^*$ satisfying $\delta(\iota, f(\!|w|\!)) = \delta(\iota, f(\!|w'|\!))$, the following hold:

- *$f_1(wu) = f_1(w'u)$ for all $u \in \Sigma_{\leftrightarrow}\Sigma^*_{\leftarrow}$*
- *$f_2(wu) = f_2(w'u)$ for all $u \in \Sigma_{\leftrightarrow}\Sigma^*_{\perp\leftarrow}$*

Note that the acceptance condition and the language of $\mathscr{A}$ are not important in the lemma.

3.2 Distributed Algorithms as Strategy Trees

Section 3.2 is devoted to the proof of Lemma 2. The first step is to represent a distributed algorithm as a *strategy tree*, whose branching structure reflects the algorithm's choices depending on the various inputs. We then build a tree automaton that accepts a strategy tree iff it represents a distributed algorithm fulfilling the given formula φ. The challenge is to define the tree automaton in such a way that its strategies can be cast into hierarchical multiplayer games with *finite sets of observations*, and that winning strategies within these games are equivalent to distributed algorithms. We show in this section that this is possible by collapsing potentially unboundedly long input sequences into an unbounded branching structure. With this construction, we can show that, if the tree automaton recognizes *some* strategy tree, then it also accepts one that represents a finite-memory distributed algorithm.

Trees and Rabin Tree Automata. Let A be a nonempty (possibly infinite) alphabet and D be a nonempty (possibly infinite) set of *directions*. An *A-labeled D-tree* is a mapping $t : D^* \to A$. In particular, ε is the root with label $t(\varepsilon)$, and ud is the d-successor of node $u \in D^*$, with label $t(ud)$.

Definition 3. A (nondeterministic) *Rabin tree automaton (RTA)* over A-labeled D-trees is a tuple $\mathscr{T} = (S, \iota, \Delta, \mathscr{F})$ with finite set of states S, initial state $\iota \in S$, acceptance condition $\mathscr{F} \subseteq 2^S \times 2^S$, and (possibly infinite) set of transitions $\Delta \subseteq S \times A \times S^D$.

A *run* of $\mathscr{T}$ on an A-labeled D-tree t is an S-labeled D-tree $\rho : D^* \to S$ where $\rho(\varepsilon) = \iota$ (the root is assigned the initial state) and, for all $u \in D^*$, $\big(\rho(u), t(u), d \in D \mapsto \rho(ud)\big) \in \Delta$. The latter is the transition *applied* at u, and we denote it by $trans_\rho(u)$.

A path of run ρ is a word $\xi = d_0 d_1 d_2 \ldots \in D^\omega$, inducing the sequence $\varepsilon, d_0, d_0 d_1, d_0 d_1 d_2, \ldots$ of nodes visited along ξ. We let $\mathrm{Inf}(\xi)$ be the set of states that occur infinitely often as the labels of these nodes. Path ξ is *accepting* if there is $(F, F') \in \mathscr{F}$ such that $\mathrm{Inf}(\xi) \cap F \neq \emptyset$ and $\mathrm{Inf}(\xi) \cap F' = \emptyset$. Run ρ is *accepting* if all its paths are accepting. Finally, $\mathscr{T}$ defines the language of A-labeled D-trees $L(\mathscr{T}) = \{t : D^* \to A \mid \text{there is an accepting run of } \mathscr{T} \text{ on } t\}$.

Lemma 3. *Let A be a singleton alphabet, D a nonempty (possibly infinite) set of directions, and $\mathscr{T}$ an RTA over A-labeled D-trees (as A is a singleton, we say that $\mathscr{T}$ is* input-free*). Call a run ρ of $\mathscr{T}$ on the unique A-labeled D-tree* rational *if, for all $w, w' \in D^*$ with $\rho(w) = \rho(w')$, we have $trans_\rho(w) = trans_\rho(w')$. If $L(\mathscr{T}) \neq \emptyset$, then there is a rational accepting run of $\mathscr{T}$.*

The lemma essentially follows from the fact that Rabin games are positionally determined for the player that aims at satisfying the Rabin objective [21]. To account for our non-standard setting of tree automata with possibly infinite D, we give a direct proof in Appendix A.

Strategy Trees. Recall that our goal is to show Lemma 2 using strategy trees as a representation of distributed algorithms. Strategy trees are trees over the (infinite) set of directions $D = \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^*$, with the aim to isolate the positions where a resynchronization occurs, via a letter from $\Sigma_{\leftrightarrow}$. By Remark 2, we only have to consider $\Sigma_{\leftrightarrow}\Sigma^* = (\Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^*)^+ = D^+$. Hence, to avoid additional notation, we can identify nonempty words in D^* with words in $\Sigma_{\leftrightarrow}\Sigma^*$. It will always be clear from the context whether the underlying alphabet is D or Σ.

Intuitively, a node $u \in D^*$ represents a given history, and the label of u represents the outputs for possible continuations from $\Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^*$. More precisely, the set Λ of labels is the set of pairs $\lambda = (\lambda_1, \lambda_2)$ where $\lambda_1 : \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^* \to Y_1$ and $\lambda_2 : \Sigma_{\leftrightarrow}\Sigma_{\perp\leftarrow}^* \to Y_2$. For $w \in \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^*$, we define $\lambda(\!|w|\!) \in (\Sigma_{\leftrightarrow} \times \Omega)(\Sigma_{\leftarrow} \times \Omega)^*$ as expected (cf. the definition of $f(\!|w|\!)$ for a distributed algorithm f). Similarly, for $w \in \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^\omega$, we obtain a word $\lambda(\!|w|\!) \in (\Sigma_{\leftrightarrow} \times \Omega)(\Sigma_{\leftarrow} \times \Omega)^\omega$.

A *strategy tree* is a Λ-labeled D-tree $t : D^* \to \Lambda$. For $u \in D^*$, let $(\lambda_1^u, \lambda_2^u)$ refer to $t(u)$. The distributed algorithm associated with t is denoted by f_t and is defined as $f_t = (f_1, f_2)$ as follows (recall that $\Sigma_{\perp\leftarrow} = \{\perp\} \times \{\leftarrow\} \times X_2$):

- $f_1(uu') = \lambda_1^u(u')$ for all $u \in \{\varepsilon\} \cup \Sigma_{\leftrightarrow}\Sigma^*$ and $u' \in \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^*$
- $f_2(uu') = \lambda_2^u(u')$ for all $u \in \{\varepsilon\} \cup \Sigma_{\leftrightarrow}\Sigma^*$, and $u' \in \Sigma_{\leftrightarrow}\Sigma_{\perp\leftarrow}^*$

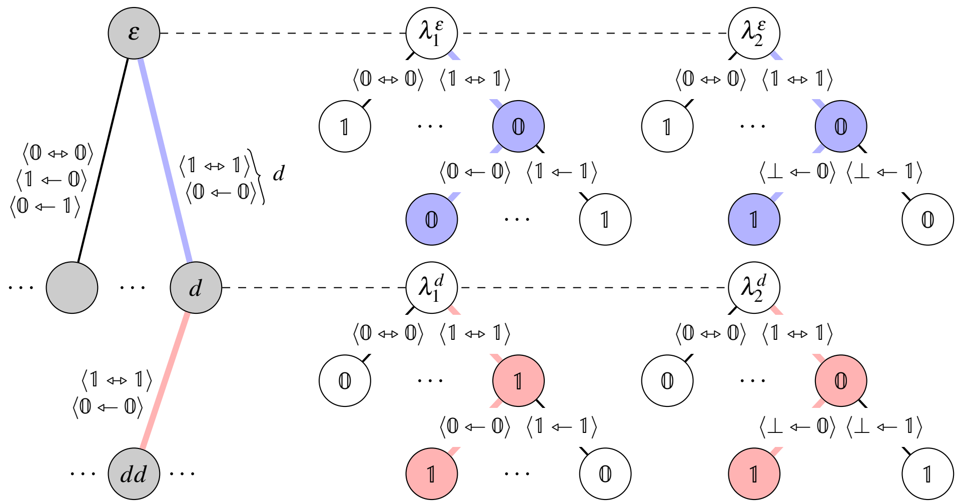

Figure 3: A strategy tree t.

In $\lambda_1^u(u')$ and $\lambda_2^u(u')$, we consider the unique decomposition of u over D so that f_1 and f_2 are well-defined.

Remark 3. The mapping $t \mapsto f_t$ is a bijection. In particular, for every distributed algorithm f, there is a strategy tree t such that $f_t = f$.

Example 3. Suppose $X_1 = X_2 = Y_1 = Y_2 = \{\mathbb{0}, \mathbb{1}\}$. Figure 3 depicts a part of a strategy tree t. Its nodes are gray-shaded. The labels of nodes of t are themselves represented as (infinite) trees. Consider the input sequence $w = \langle \mathbb{1} \leftrightarrow \mathbb{1}\rangle\langle \mathbb{0} \leftarrow \mathbb{0}\rangle\langle \mathbb{1} \leftrightarrow \mathbb{1}\rangle\langle \mathbb{0} \leftarrow \mathbb{0}\rangle \in \Sigma_{\leftrightarrow}\Sigma^*$. To know what f_t outputs for the first two signals, we look at the blue-colored nodes of the trees associated with the root of t. To determine the outputs for the two remaining signals, we look at the red-colored nodes of the trees associated with node d. We thus get $f_t(\!|w|\!) = (\langle \mathbb{1} \leftrightarrow \mathbb{1}\rangle, (\mathbb{0},\mathbb{0}))(\langle \mathbb{0} \leftarrow \mathbb{0}\rangle, (\mathbb{0},\mathbb{1}))(\langle \mathbb{1} \leftrightarrow \mathbb{1}\rangle, (\mathbb{1},\mathbb{0}))(\langle \mathbb{0} \leftarrow \mathbb{0}\rangle, (\mathbb{1},\mathbb{1}))$ for the whole word w. ◁

Now, Lemma 2 is a consequence of the following lemma:

Lemma 4. *Let $\varphi \in \mathrm{LTL}(\mathcal{N})$. There is a DRWA $\mathcal{A} = (S, \iota, \delta, \mathcal{F})$, with $\delta : S \times (\Sigma \times \Omega) \to S$, such that the following are equivalent:*

(1) There is a strategy tree t such that f_t fulfills φ.

(2) There is a strategy tree t such that (a) f_t fulfills φ, and (b) for all $w, w' \in D^*$ with $\delta(\iota, f_t(\!|w|\!)) = \delta(\iota, f_t(\!|w'|\!))$, we have $t(w) = t(w')$.

Proof. Let $\varphi \in \mathrm{LTL}(\mathcal{N})$ be the given formula. We first define $\mathcal{A}$ and then prove its correctness in terms of the statement of Lemma 4 using an RTA $\mathcal{T}_\varphi$ over strategy trees.

The DRWA $\mathcal{A}$. It is well known that there is a DRWA $\mathcal{A}_\varphi = (S_\varphi, \iota_\varphi, \delta_\varphi, \mathcal{F}_\varphi)$ over $\Sigma \times \Omega$, with doubly exponentially many states and exponentially many acceptance pairs, such that $L(\mathcal{A}_\varphi) = L(\varphi)$ (cf. [38, 41]). We refer to states of $\mathcal{A}_\varphi$ by $s \in S_\varphi$.

Starting from $\mathcal{A}_\varphi$, we now define the DRWA $\mathcal{A} = (S, \iota, \delta, \mathcal{F})$ such that, for words that contain infinitely many $\leftrightarrow$, it is enough to look at the sequence of states reached by $\mathcal{A}$ right before the $\leftrightarrow$-positions to determine whether the word is in $L(\mathcal{A}_\varphi)$ or not. The idea is to keep track of the set of states that are taken between two $\leftrightarrow$-positions. Accordingly, the set of states is $S = S_\varphi \times 2^{S_\varphi}$, with initial state $\iota = (\iota_\varphi, \emptyset)$. Concerning the transitions, for $(\mathfrak{s}, R) \in S$ and $\alpha = (\langle x_1 \rightleftharpoons x_2 \rangle, (y_1, y_2)) \in \Sigma \times \Omega$, we let

$$\delta((\mathfrak{s},R),\alpha) = \begin{cases} (\delta_\varphi(\mathfrak{s},\alpha), \{\delta_\varphi(\mathfrak{s},\alpha)\} \cup R) & \text{if } \rightleftharpoons\, = \,\leftarrow \\ (\delta_\varphi(\mathfrak{s},\alpha), \{\delta_\varphi(\mathfrak{s},\alpha)\}) & \text{if } \rightleftharpoons\, = \,\leftrightarrow . \end{cases}$$

Finally, the acceptance condition is given by $\mathscr{F} = \{(G_F, G_{F'}) \mid (F, F') \in \mathscr{F}_\varphi\}$ where $G_F = \{(\mathfrak{s}, R) \in S \mid F \cap R \neq \emptyset\}$ and $G_{F'} = \{(\mathfrak{s}, R) \in S \mid F' \cap R \neq \emptyset\}$.

The following claim states that $\mathscr{A}$ is correct wrt. executions with infinitely many synchronization points, while the acceptance condition is looking only at states reached right before these synchronizing points (see Appendix B for the proof):

Claim 1. Let $w_0, w_1, w_2, \ldots \in (\Sigma_{\leftrightarrow} \times \Omega)(\Sigma_{\leftarrow} \times \Omega)^*$. Moreover, let $w = w_0 w_1 w_2 \ldots$ be the concatenation of all w_i. Set $s_0 = \iota$ and, for $i \in \mathbb{N}$, $s_{i+1} = (\mathfrak{s}_{i+1}, R_{i+1}) = \delta(\iota, w_0 \ldots w_i)$. Then, $w \in L(\mathscr{A}_\varphi) \iff$ the sequence $s_0, s_1, s_2, \ldots$ satisfies $\mathscr{F} \iff w \in L(\mathscr{A})$.

The RTA $\mathscr{T}_\varphi$. To get finite-memory algorithms, we will rely on Lemma 3, which is based on tree automata. In fact, a crucial ingredient of the proof is an RTA $\mathscr{T}_\varphi$ over Λ-labeled D-trees such that

$$L(\mathscr{T}_\varphi) = \{t \mid t \text{ is a strategy tree such that } f_t \text{ fulfills } \varphi\}.$$

It is defined by $\mathscr{T}_\varphi = (S, \iota, \Delta, \mathscr{F})$ where S, ι, and $\mathscr{F}$ are taken from $\mathscr{A}$, and Δ is given by

$$\Delta = \left\{ (s = (\mathfrak{s}, R), \lambda, (s_d)_{d \in D}) \;\middle|\; \begin{array}{ll} s_d = \delta(s, \lambda \llparenthesis d \rrparenthesis) \text{ for all } d \in \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^* & \text{(T1)} \\ \lambda \llparenthesis w \rrparenthesis \in L(\mathscr{A}_\varphi[\mathfrak{s}]) \text{ for all } w \in \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^\omega & \text{(T2)} \end{array} \right\}.$$

Here, $\mathscr{A}_\varphi[\mathfrak{s}] = (S_\varphi, \mathfrak{s}, \delta_\varphi, \mathscr{F}_\varphi)$ is the automaton $\mathscr{A}_\varphi$ but where ι_φ has been replaced by $\mathfrak{s}$ as the initial state. While condition (T1) "unfolds" $\mathscr{A}$ into the tree structure taking care of input sequences with infinitely many synchronization points, condition (T2) guarantees that the distributed algorithm behaves correctly should there be no more synchronization.

Correctness of $\mathscr{T}_\varphi$, which relies on Claim 1, is shown in Appendix C.

Putting It Together. We now obtain Lemma 4 as a corollary from Lemma 3 using $\mathscr{T}_\varphi$.

Direction (2) $\Longrightarrow$ (1) is trivial. Let us show (1) $\Longrightarrow$ (2) and suppose $L(\mathscr{T}_\varphi) \neq \emptyset$. Consider the input-free RTA $\mathscr{T}'_\varphi = (S, \iota, \Delta', \mathscr{F})$ obtained from $\mathscr{T}_\varphi$ by replacing the transition relation with $\Delta' = \{(s, (s_d)_{d \in D}) \mid (s, \lambda, (s_d)_{d \in D}) \in \Delta\}$. Note that $L(\mathscr{T}'_\varphi) \neq \emptyset$. By Lemma 3, there is an accepting run ρ of $\mathscr{T}'_\varphi$ such that, for all $w, w' \in D^*$ with $\rho(w) = \rho(w')$, we have $\mathit{trans}_\rho(w) = \mathit{trans}_\rho(w')$. For all transitions $\theta = (s, (s_d)_{d \in D}) \in \Delta'$, fix $\lambda^\theta \in \Lambda$ such that $(s, \lambda^\theta, (s_d)_{d \in D}) \in \Delta$. Let $t : D^* \to \Lambda$ be the strategy tree defined by $t(w) = \lambda^{\mathit{trans}_\rho(w)}$.

We have $t \in L(\mathscr{T}_\varphi)$. Therefore, f_t fulfills φ, i.e., (2a) holds. It remains to show (2b). Let $w, w' \in D^*$ with $\delta(\iota, f_t\llparenthesis w \rrparenthesis) = \delta(\iota, f_t\llparenthesis w' \rrparenthesis)$. By induction, we can show that $\rho(w) = \delta(\iota, f_t\llparenthesis w \rrparenthesis) = \delta(\iota, f_t\llparenthesis w' \rrparenthesis) = \rho(w')$, i.e., $t(w) = t(w')$, which proves (2b). Indeed, $\delta(\iota, f_t\llparenthesis \varepsilon \rrparenthesis) = \iota = \rho(\varepsilon)$ and, for $u \in D^*$ and $d \in D$, we have $\delta(\iota, f_t\llparenthesis ud \rrparenthesis) = \delta(\iota, f_t\llparenthesis u \rrparenthesis \cdot \lambda^u\llparenthesis d \rrparenthesis) = \delta(\delta(\iota, f_t\llparenthesis u \rrparenthesis), \lambda^u\llparenthesis d \rrparenthesis) = \delta(\rho(u), \lambda^u\llparenthesis d \rrparenthesis) = \rho(ud)$. The last equation is by (T1) in the definition of the transition relation Δ of $\mathscr{T}_\varphi$. $\square$

4 From Finite-Memory Distributed Algorithms to Games

4.1 Games with Imperfect Information

The existence of finite-memory distributed algorithms shown in Section 3 paves the way for a reduction of the synthesis problem to $(2,1)$-player games with imperfect information, where two players form a coalition against an environment in order to fulfill some objective. The main differences between games and the synthesis problem are twofold: Games are played in an arena, on a finite set of nodes (or states), while the input of the synthesis problem is a logical specification. More importantly, in a game, communication between players occurs implicitly, by observing the nodes that are visited. Hence, communication between players is bounded by the finite nature of the arena, whereas in the synthesis problem, processes can send an unbounded amount of information at each communication point. Recall that $P = \{1,2\}$ is the set of processes. In the context of games, however, its elements are referred to as *players*.

Definition 4. A $(2,1)$-*player game* is a tuple $\mathscr{G} = (V, v_0, W, \Gamma, (A_p, \mathscr{O}_p, obs_p)_{p\in P}, \tau)$. Here, V is the finite set of nodes containing the initial node $v_0 \in V$. We assume a Rabin winning condition $W \subseteq 2^V \times 2^V$. Moreover, Γ is the finite set of actions of the environment, A_p is the finite set of actions of player p, $\mathscr{O}_p$ is the finite set of observations of p, and $obs_p : V \times \Gamma \to \mathscr{O}_p$ determines what p actually observes for a given node and environment action. Finally, $\tau : V \times \Gamma \times (A_1 \times A_2) \to V$ is the transition function.

The game proceeds in rounds $r \in \mathbb{N}$, the first round starting in v_0. When a round starts in $v \in V$, the environment first chooses an action $\gamma \in \Gamma$. Players 1 and 2 do not see γ, but only $obs_1(v,\gamma)$ and $obs_2(v,\gamma)$, respectively. Once the players receive these observations, they simultaneously choose actions $a_1 \in A_1$ and $a_2 \in A_2$. The next state is $\tau(v,\gamma,(a_1,a_2))$, etc.

Accordingly, a *play* (starting from v_0) is a sequence $\pi = (v_0,\gamma_0)(v_1,\gamma_1)\ldots \in (V\times\Gamma)^\omega$ such that, for all $r \in \mathbb{N}$, there is $(a_1,a_2) \in A_1 \times A_2$ such that $v_{r+1} = \tau(v_r,\gamma_r,(a_1,a_2))$. The observation that a player p collects in play π until round r is defined as $[\![(v_0,\gamma_0)\ldots(v_r,\gamma_r)]\!]_p^{\mathsf{game}} = obs_p(v_0,\gamma_0)\ldots obs_p(v_r,\gamma_r) \in \mathscr{O}_p^*$. The play is *winning* (for the coalition of players 1 and 2) if $v_0v_1v_2\ldots$ satisfies the Rabin winning condition in the expected manner.

A *strategy* for player p is a mapping $g_p : \mathscr{O}_p^+ \to A_p$. A *strategy profile* is a pair $g = (g_1, g_2)$ of strategies. We say that play $\pi = (v_0,\gamma_0)(v_1,\gamma_1)\ldots$ is *compatible* with g if, for all $r \in \mathbb{N}$, we have $v_{r+1} = \tau(v_r,\gamma_r,(a_1^r,a_2^r))$ where $a_p^r = g_p([\![(v_0,\gamma_0)\ldots(v_r,\gamma_r)]\!]_p^{\mathsf{game}})$. Strategy profile g is *winning* if all plays that are compatible with g are winning.

The following fact has been shown by Peterson and Reif [32] for games and corresponds to the undecidability result of Pnueli and Rosner [36] for two processes without communication.

Fact 2 (Peterson-Reif)**.** *The following problem is undecidable: Given a* $(2,1)$*-player game* $\mathscr{G}$*, is there a winning strategy profile?*

Therefore, we have to impose a restriction. It turns out that, when we translate the synthesis problem for $\mathscr{N} = \{\leftrightarrow, \leftarrow\}$ to games in Section 4.2, player 1 (who corresponds to process 1) will have perfect information. We say that player p has *perfect information* in $\mathscr{G}$ if $\mathscr{O}_p = V \times \Gamma$ and obs_p is the identity function.

The following result is by van der Meyden and Wilke [29, Theorem 6] with a proof in [30, Theorem 1].

Fact 3 (van der Meyden-Wilke)**.** *The following problem is decidable: Given a* $(2,1)$*-player game* $\mathscr{G}$ *such that player 1 has perfect information, is there a winning strategy profile?*

Note that the transition function of our game is deterministic so that we actually obtain decidability in exponential time exploiting a standard technique: We use a small tree automaton to represent the *global* (full information) winning strategies and another small alternating tree automaton for the local ones of player 2 that conform with some global strategy. The alternating automaton can be checked for nonemptiness in exponential time.

4.2 Reduction to Games

The analogies between synthesis and games suggest a natural translation of the former into the latter. However, the crucial difference being the access to histories, we rely on the fact that certain histories in distributed algorithms enjoy a finite abstraction. In fact, it is enough to reveal a bounded amount of information to player 2 at every environment action from $\Sigma_{\leftrightarrow}$.

Lemma 5. *Let $\varphi \in \mathrm{LTL}(\mathscr{N})$ with $\mathscr{N} = \{\leftrightarrow, \leftarrow\}$. We can effectively construct a $(2,1)$-player game $\mathscr{G}_\varphi$ such that player 1 has perfect information and the following holds: There is a distributed algorithm that fulfills φ iff there is a winning strategy profile in $\mathscr{G}_\varphi$.*

Proof. By Remark 2, input sequences that do not start with a symbol from $\Sigma_{\leftrightarrow}$ are discarded. Hence, we assume that those sequences are all trivially "winning", i.e., $(\Sigma_{\leftarrow} \times \Omega)(\Sigma \times \Omega)^\omega \subseteq L(\varphi)$. Let $\mathscr{A} = (S, \iota, \delta, \mathscr{F})$ be the DRWA according to Lemma 2. Recall that $S = S_\varphi \times 2^{S_\varphi}$, where S_φ is taken from $\mathscr{A}_\varphi$, and that the transition function is of the form $\delta : S \times (\Sigma \times \Omega) \to S$.

We construct the game $\mathscr{G}_\varphi = (V, v_0, W, \Gamma, (A_p, \mathscr{O}_p, obs_p)_{p \in P}, \tau)$ as follows. Obviously, player 1 corresponds to process 1 and player 2 to process 2. We simply set $V = S$ and $v_0 = \iota = (\iota_\varphi, \emptyset)$, and W contains, for all $(F_\varphi, F'_\varphi) \in \mathscr{F}_\varphi$, the pair $(F_\varphi \times 2^{S_\varphi}, F'_\varphi \times 2^{S_\varphi})$.

Moreover, $\Gamma = \Sigma$, the idea being that the environment chooses the inputs and the network graph. Accordingly, processes 1 and 2 choose their outputs so that $A_1 = Y_1$ and $A_2 = Y_2$.

Player 1's observations are $\mathscr{O}_1 = V \times \Sigma$ and we set $obs_1(s, \langle x_1 \rightleftharpoons x_2 \rangle) = (s, \langle x_1 \rightleftharpoons x_2 \rangle)$. Thus, player 1 has full information. Player 2's observations are $\mathscr{O}_2 = (S \times \Sigma_{\leftrightarrow}) \cup \Sigma_{\perp\leftarrow}$ and we set

$$obs_2(s, \langle x_1 \rightleftharpoons x_2 \rangle) = \begin{cases} (s, \langle x_1 \leftrightarrow x_2 \rangle) & \text{if } \rightleftharpoons\, = \,\leftrightarrow \\ \langle \perp \leftarrow x_2 \rangle & \text{if } \rightleftharpoons\, = \,\leftarrow . \end{cases}$$

That is, when the environment chooses a synchronizing input signal, the current state of $\mathscr{A}$ is revealed to player 2, which corresponds to passing the (abstracted) history to process 2. Finally, the transitions are given by $\tau(s, \langle x_1 \rightleftharpoons x_2 \rangle, (y_1, y_2)) = \delta\big(s, (\langle x_1 \rightleftharpoons x_2 \rangle, (y_1, y_2))\big)$.

Correctness of the reduction is proved in Appendix D. □

We have shown Theorem 2 saying that the problem SYNTHESIS($\{\leftrightarrow, \leftarrow\}$) is decidable.

Complexity. The size of $\mathscr{A}_\varphi$ is doubly exponential in the length of the formula. It follows that the size of $\mathscr{A}$ is triply exponential, and so is the size of $\mathscr{G}_\varphi$. Deciding the winner of our $(2,1)$-player game where one player has perfect information can be done in exponential time so that the overall decision procedure runs in 4-fold exponential time.

Note that SYNTHESIS($\{\leftrightarrow\}$), which is equivalent to centralized synthesis in presence of one single process, is 2EXPTIME-complete [34], from which we inherit the best known lower bound for our problem. Moreover, hierarchical information further increases the complexity: for static pipelines with variable number of processes, the problem is no longer elementary [36]. However, it may be possible to improve our upper bound, which is left for future work.

As, in the proof, the given LTL formula is translated into a DRWA, synthesis is decidable even when the specification is given by any common finite automaton over ω-words (starting with a nondeterministic Büchi automaton, we actually save one exponential wrt. LTL):

Corollary 1. *Over $\mathcal{N} = \{\leftrightarrow, \leftarrow\}$, the following problem is decidable: Given an ω-regular language $L \subseteq (\Sigma \times \Omega)^\omega$, is L realizable?*

5 Reduction from $\{\leftrightarrow, \leftarrow, \rightarrow\}$ to $\{\leftrightarrow, \leftarrow\}$

In this section, we show decidability for the network model $\mathcal{N} = \{\leftrightarrow, \leftarrow, \rightarrow\}$, with input alphabet $\Sigma = X_1 \times \mathcal{N} \times X_2$ and output alphabet $\Omega = Y_1 \times Y_2$. Recall that this also implies decidability for the network model $\{\leftarrow, \rightarrow\}$.

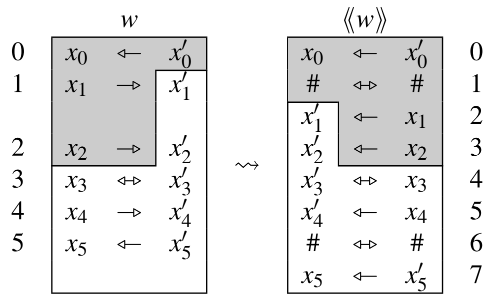

Figure 4: Illustration of $\langle\!\langle \cdot \rangle\!\rangle : \Sigma^* \to (\Sigma')^*$

The idea is to reduce the problem to the case of the network model $\mathcal{N}' = \{\leftrightarrow, \leftarrow\}$ that we considered in Sections 3 and 4, choosing as input alphabet $\Sigma' = X_1' \times \mathcal{N}' \times X_2'$ where $X_1' = X_2' = (X_1 \cup X_2) \uplus \{\#\}$, and as output alphabet $\Omega' = Y_1' \times Y_2'$ where $Y_1' = Y_2' = (Y_1 \cup Y_2) \uplus \{\#\}$. To do so, we will rewrite the given specification $\varphi \in \mathrm{LTL}(\mathcal{N})$ towards an (automata-based) specification over $\mathcal{N}'$ in such a way that process 1 can always simulate the "more informed" process and process 2 simulates the other process. Roughly speaking, what we are looking for is a translation $\langle\!\langle \cdot \rangle\!\rangle : \Sigma^* \to (\Sigma')^*$ of histories w over $\mathcal{N}$ to histories $\langle\!\langle w \rangle\!\rangle$ over $\mathcal{N}'$ such that the view of process 1 in $\langle\!\langle w \rangle\!\rangle$ is "congruent" to the view of the more informed process in w, and the view of process 2 in $\langle\!\langle w \rangle\!\rangle$ is "congruent" to the view of the less informed process in w. Note that [8] also uses a simulation technique to cope with dynamically changing hierarchies.

Example 4. Before defining $\langle\!\langle \cdot \rangle\!\rangle$ formally, we illustrate it in Figure 4 for a history w. Round 0 uses $\leftarrow$ so that there is nothing to change. Round 1 employs $\rightarrow$ so that process 1 henceforth simulates process 2 and vice versa. To make sure that the corresponding views in $\langle\!\langle w \rangle\!\rangle$ are still "congruent", we insert the dummy signal $\langle \# \leftrightarrow \# \rangle$. Actually, the gray-shaded view of process 1 in w after round 2 contains the same information as the gray-shaded view of process 2 in $\langle\!\langle w \rangle\!\rangle$ after round 3. Though w encounters $\leftrightarrow$ in round 3, we decide not to change roles again; we will only do so when facing another $\leftarrow$ (like in round 5). $\lhd$

Formally, $\langle\!\langle \cdot \rangle\!\rangle : \Sigma^* \to (\Sigma')^*$ is given by the sequential *transducer* shown in Figure 5. For the moment, we ignore the red part. A transition with label $\alpha \mid \beta$ reads α and transforms it into β. As the transducer is deterministic, it actually defines a function. When we include the red part, i.e., the symbols from Ω and Ω', we obtain an extension to $\langle\!\langle \cdot \rangle\!\rangle : (\Sigma \times \Omega)^* \to (\Sigma' \times \Omega')^*$. Finally, these mappings are extended to infinite words as expected.

Observe that the state of the transducer reached after reading $w \in \Sigma^*$ (or $w \in (\Sigma \times \Omega)^*$) reveals the process that process 1 is currently simulating. We denote this process by $\mathrm{sim}_1(w)$. Accordingly,

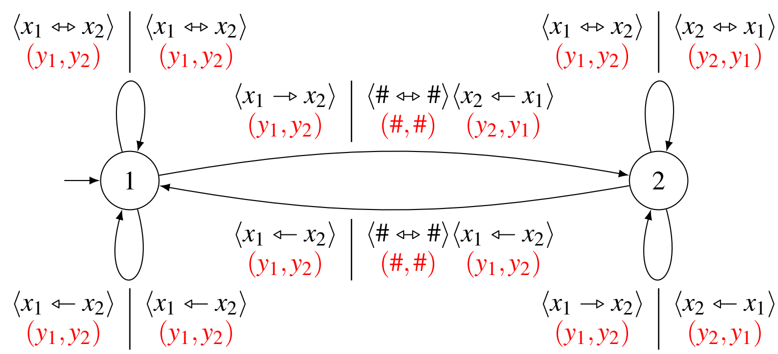

Figure 5: The mappings $\langle\!\langle \cdot \rangle\!\rangle : \Sigma^* \to (\Sigma')^*$ and $\langle\!\langle \cdot \rangle\!\rangle : (\Sigma \times \Omega)^* \to (\Sigma' \times \Omega')^*$

$\mathrm{sim}_2(w) = 3 - \mathrm{sim}_1(w)$ is the process that process 2 simulates after input sequence w. For the example word w in Figure 4, we get $\mathrm{sim}_1(w) = 1$ and $\mathrm{sim}_2(w) = 2$.

Note that, for all $w, w' \in \Sigma^*$ and $p \in \{1,2\}$, such that $[\![w]\!]_p = [\![w']\!]_p$, we have $\mathrm{sim}_p(w) = \mathrm{sim}_p(w')$. This is because the simulated process only depends on the sequence of links.

Note that the mappings $\langle\!\langle \cdot \rangle\!\rangle$ are all injective. Indeed, at the first position that distinguishes w and w', the transducer produces letters that distinguish $\langle\!\langle w \rangle\!\rangle$ and $\langle\!\langle w' \rangle\!\rangle$. There is an analogous statement for views (proved in Appendix E):

Lemma 6. *For all $w, w' \in \Sigma^*$ and $p \in \{1,2\}$, the following hold:*

(a) $[\![\langle\!\langle w \rangle\!\rangle]\!]_p = [\![\langle\!\langle w' \rangle\!\rangle]\!]_p \implies [\![w]\!]_{\mathrm{sim}_p(w)} = [\![w']\!]_{\mathrm{sim}_p(w')}$

(b) $[\![w]\!]_p = [\![w']\!]_p \implies [\![\langle\!\langle w \rangle\!\rangle]\!]_{\mathrm{sim}_p(w)} = [\![\langle\!\langle w' \rangle\!\rangle]\!]_{\mathrm{sim}_p(w')}$

Moreover, the transducer can be applied to ω-regular languages in the following sense:

Lemma 7. *Given a DRWA $\mathscr{A}$ over the alphabet $\Sigma \times \Omega$, there is a DRWA $\mathscr{A}'$ over $\Sigma' \times \Omega'$ of linear size such that $L(\mathscr{A}') = \langle\!\langle L(\mathscr{A}) \rangle\!\rangle := \{\langle\!\langle w \rangle\!\rangle \mid w \in L(\mathscr{A})\}$.*

Now, decidability for $\mathscr{N}$ is due to Lemma 7 and the following result, whose proof crucially relies on injectivity of $\langle\!\langle \cdot \rangle\!\rangle$ and Lemma 6 (cf. Appendix F):

Lemma 8. *Let $\varphi \in \mathrm{LTL}(\mathscr{N})$. The following statements are equivalent:*

(i) There is a distributed algorithm f (over $\mathscr{N}$) such that, for all $w \in \Sigma^\omega$, $f(\!|w|\!) \in L(\varphi)$.

(ii) There is a distributed algorithm f' (over $\mathscr{N}'$) such that, for all $w \in \Sigma^\omega$, $f'(\!|\langle\!\langle w \rangle\!\rangle|\!) \in \langle\!\langle L(\varphi) \rangle\!\rangle$.

In other words, an instance $\varphi \in \mathrm{LTL}(\mathscr{N})$ of the synthesis problem can be reduced to the existence of a distributed algorithm f' over $\mathscr{N}'$, Σ', and Ω' that fulfills $L = M \cup \langle\!\langle L(\varphi) \rangle\!\rangle$ where $M \subseteq (\Sigma' \times \Omega')^\omega$ is the set of words whose projection to Σ' is *not* contained in $\langle\!\langle \Sigma^\omega \rangle\!\rangle$. Using Lemma 7, we obtain a DRWA for L (of doubly exponential size) so that, by Corollary 1, the problem is decidable. Again, the overall procedure runs in 4-fold exponential time.

This concludes the proof of our main result, Theorem 1.

6 Conclusion

We showed that synthesis in a dynamic, synchronous two-node system is decidable for LTL specifications if and only if the network model does not contain the empty network. Our model covers full-information protocols where nodes communicate their complete unbounded causal history.

Future work is concerned with establishing the precise complexity of our problem and, possibly, improving the 4-fold exponential upper bound. Moreover, it would be interesting to identify the subsets of $\{\times, \leftarrow, \rightarrow, \leftrightarrow\}^\omega$ that give rise to a decidable synthesis problem. For example, one may allow boundedly many empty links in an input sequence. Finally, we plan to extend our model to distributed systems of arbitrary size. We conjecture that synthesis is solvable over a given network model if and only if, in each communication graph, any two nodes are connected via a directed path. This would yield an analogue of the information-fork criterion [16], which applies to static architectures. It remains to be seen whether the ideas presented in [16] can be lifted to dynamic architectures with causal memory.

Acknowledgment. We thank Dietmar Berwanger for valuable feedback.

References

[1] (2014): *I2C-Bus Specification and User Manual*. `https://www.nxp.com/docs/en/user-guide/UM10204.pdf`.

[2] (2014): *ISO 11898-1:2015. Road vehicles —- Controller area network (CAN) — Part 1: Data link layer and physical signalling*. `https://www.iso.org/standard/63648.html`.

[3] Martín Abadi, Leslie Lamport & Pierre Wolper (1989): *Realizable and unrealizable specifications of reactive systems*. In: International Colloquium on Automata, Languages, and Programming (ICALP'89), Springer, pp. 1–17.

[4] Yehuda Afek, Hagit Attiya, Alan Fekete, Michael Fischer, Nancy Lynch, Yishay Mansour, Dai-Wei Wang & Lenore Zuck (1994): *Reliable communication over unreliable channels*. Journal of the ACM (JACM) 41(6), pp. 1267–1297.

[5] Alfred V. Aho, Aaron D. Wyner, Mihalis Yannakakis & Jeffrey D. Ullman (1982): *Bounds on the size and transmission rate of communications protocols*. Computers & Mathematics with Applications 8(3), pp. 205–214.

[6] Eralp A. Akkoyunlu, Kattamuri Ekanadham & Richard V. Huber (1975): *Some constraints and tradeoffs in the design of network communications*. In: 5th ACM symposium on Operating Systems Principles, pp. 67–74.

[7] Keith A. Bartlett, Roger A. Scantlebury & Peter T. Wilkinson (1969): *A note on reliable full-duplex transmission over half-duplex links*. Communications of the ACM (CACM) 12(5), pp. 260–261.

[8] Dietmar Berwanger, Anup Basil Mathew & Marie van den Bogaard (2018): *Hierarchical information and the synthesis of distributed strategies*. Acta Informatica 55(8), pp. 669–701.

[9] J. Richard Büchi & Lawrence H. Landweber (1990): *Solving sequential conditions by finite-state strategies*. In: The Collected Works of J. Richard Büchi, Springer, pp. 525–541.

[10] Bernadette Charron-Bost, Matthias Függer & Thomas Nowak (2015): *Approximate Consensus in Highly Dynamic Networks: The Role of Averaging Algorithms*. In: 42nd International Colloquium on Automata, Languages, and Programming (ICALP'15), pp. 528–539.

[11] Bernadette Charron-Bost & André Schiper (2009): *The heard-of model: computing in distributed systems with benign faults*. Distributed Computing 22(1), pp. 49–71.

[12] Alonzo Church (1957): *Applications of recursive arithmetic to the problem of circuit synthesis – Summaries of talks*. Institute for Symbolic Logic, Cornell University.

[13] Étienne Coulouma & Emmanuel Godard (2013): *A Characterization of Dynamic Networks Where Consensus is Solvable*. In: International Colloquium on Structural Information and Communication Complexity (SIROCCO'13), Springer, pp. 24–35.

[14] Rayna Dimitrova & Bernd Finkbeiner (2009): *Synthesis of fault-tolerant distributed systems*. In: International Symposium on Automated Technology for Verification and Analysis (ATVA'09), Springer, pp. 321–336.

[15] Ronald Fagin, Yoram Moses, Joseph Y Halpern & Moshe Y. Vardi (2003): *Reasoning about knowledge*. MIT press.

[16] Bernd Finkbeiner & Sven Schewe (2005): *Uniform distributed synthesis*. In: *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, IEEE, pp. 321–330.

[17] Paul Gastin, Benjamin Lerman & Marc Zeitoun (2004): *Distributed games with causal memory are decidable for series-parallel systems*. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, Springer, pp. 275–286.

[18] Paul Gastin, Nathalie Sznajder & Marc Zeitoun (2009): *Distributed synthesis for well-connected architectures*. *Formal Methods in System Design* 34(3), pp. 215–237.

[19] Blaise Genest, Hugo Gimbert, Anca Muscholl & Igor Walukiewicz (2013): *Asynchronous Games over Tree Architectures*. In: *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, *Lecture Notes in Computer Science* 7966, Springer, pp. 275–286.

[20] Hugo Gimbert (2018): *On the Control of Asynchronous Automata*. In: *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'17)*, *LIPIcs* 93, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 30:1–30:15, doi:10.4230/LIPIcs.FSTTCS.2017.30. Available at `http://drops.dagstuhl.de/opus/volltexte/2018/8414`.

[21] Nils Klarlund (1994): *Progress Measures, Immediate Determinacy, and a Subset Construction for Tree Automata*. *Annals of Pure and Applied Logic* 69(2-3), pp. 243–268, doi:10.1016/0168-0072(94)90086-8. Available at `https://doi.org/10.1016/0168-0072(94)90086-8`.

[22] Fabian Kuhn, Nancy Lynch & Rotem Oshman (2010): *Distributed computation in dynamic networks*. In: *42nd ACM Symposium on Theory of Computing (STOC'10)*, pp. 513–522.

[23] Orna Kuperman & Moshe Y. Vardi (2001): *Synthesizing distributed systems*. In: *16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, IEEE, pp. 389–398.

[24] Orna Kupferman & Moshe Y. Vardi (1999): *Church's problem revisited*. *Bulletin of Symbolic Logic* 5(2), pp. 245–263.

[25] Orna Kupferman & Moshe Y. Vardi (2000): *Synthesis with incomplete information*. In: *Advances in Temporal Logic*, Springer, pp. 109–127.

[26] Leslie Lamport, Robert Shostak & Marshall Pease (1982): *The Byzantine Generals Problem*. *ACM Transactions on Programming Languages and Systems* 4(3), pp. 382–401.

[27] Nancy A. Lynch (1996): *Distributed algorithms*. Elsevier.

[28] P. Madhusudan, P.S. Thiagarajan & Shaofa Yang (2005): *The MSO theory of connectedly communicating processes*. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science (FoSSaCS'05)*, Springer, pp. 201–212.

[29] Ron Van der Meyden & Thomas Wilke (2005): *Synthesis of distributed systems from knowledge-based specifications*. In: *International Conference on Concurrency Theory (CONCUR'05)*, Springer, pp. 562–576.

[30] Ron Van der Meyden & Thomas Wilke (2005): *Synthesis of distributed systems from knowledge-based specifications. UNSW-CSE-TR-0504*. Technical Report.

[31] Swarup Mohalik & Igor Walukiewicz (2003): *Distributed games*. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, Springer, pp. 338–351.

[32] Gary L. Peterson & John H. Reif (1979): *Multiple-person alternation*. In: *20th Annual Symposium on Foundations of Computer Science (FOCS'79)*, IEEE, pp. 348–363.

[33] Amir Pnueli (1981): *The temporal semantics of concurrent programs*. *Theoretical Computer Science* 13(1), pp. 45–60.

[34] Amir Pnueli & Roni Rosner (1988): *A framework for the synthesis of reactive modules*. In: *International Conference on Concurrency (Concurrency 88)*, Springer, pp. 4–17.

[35] Amir Pnueli & Roni Rosner (1989): *On the synthesis of a reactive module*. In: *16th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL'89)*, pp. 179–190.

[36] Amir Pnueli & Roni Rosner (1990): *Distributed reactive systems are hard to synthesize*. In: *31st Annual Symposium on Foundations of Computer Science (FoCS'90)*, IEEE, pp. 746–757.

[37] Michael Oser Rabin (1972): *Automata on infinite objects and Church's problem*. 13, American Mathematical Soc.

[38] Shmuel Safra (1988): *On the Complexity of ω-Automata*. In: *29th Annual Symposium on Foundations of Computer Science (FoCS'88)*, IEEE Computer Society, pp. 319–327.

[39] Wolfgang Thomas (1990): *Automata on Infinite Objects*. In Jan van Leeuwen, editor: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, Elsevier and MIT Press, pp. 133–191.

[40] Wolfgang Thomas (1995): *On the synthesis of strategies in infinite games*. In: *Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, Springer, pp. 1–13.

[41] Moshe Y. Vardi & Pierre Wolper (1994): *Reasoning About Infinite Computations*. *Information and Computation* 115(1), pp. 1–37.

[42] Yaron Velner & Alexander Rabinovich (2011): *Church synthesis problem for noisy input*. In: *International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'11)*, Springer, pp. 275–289.

A Proof of Lemma 3

Lemma 3. *Let A be a singleton alphabet, D a nonempty (possibly infinite) set of directions, and $\mathscr{T}$ an RTA over A-labeled D-trees. Call a run ρ of $\mathscr{T}$ on the unique A-labeled D-tree* rational *if, for all $w, w' \in D^*$ with $\rho(w) = \rho(w')$, we have $trans_\rho(w) = trans_\rho(w')$. If $L(\mathscr{T}) \neq \emptyset$, then there is a rational accepting run of $\mathscr{T}$.*

The proof is inspired by [37, 39] where it is shown that every nonempty language recognized by a classical RTA contains a tree with only finitely many distinct subtress. Note that, here, we deal with trees that are not necessarily bounded branching. Moreover, we show a statement on runs rather than the recognized tree language.

Let $\mathscr{T} = (S, \iota, \Delta, \mathscr{F})$ be the RTA over A-labeled D-trees. Since $A = \{a\}$, we consider the transition relation Δ to be a subset of $S \times S^D$, fixing input a.

Call a state $s \in S$

- *absorbing* if the only transition in which s occurs is $(s, (s_d)_{d\in D}) \in \Delta$ with $s_d = s$ for all $d \in D$,
- *vanishing* if it is the initial state ι and has no incoming transition, i.e., there is no transition $(s, (s_d)_{d\in D}) \in \Delta$ with $s_d = s$ for some $d \in D$.
- *live* if it is neither absorbing nor vanishing.

We prove the statement of the theorem by induction on the number of live states in S.

Base case. Suppose there are no live states in S. Let ρ be an accepting run of $\mathscr{T}$. If $\rho(\varepsilon)$ is absorbing, then ρ is rational. If $\rho(\varepsilon)$ is vanishing, then $\rho(d)$ is absorbing for all $d \in D$ (i.e., for all children of the root). Again, it follows that ρ is rational. The base case follows.

Inductive step. Let ρ be an accepting run of $\mathscr{T}$. We distinguish between three cases.

Case 1. There exists a live state $s \in S$ that does not appear in ρ. If so, ρ is also an accepting run of the tree automaton $\mathscr{T}'$ that we obtain from $\mathscr{T}$ by removing state s and all transitions in which s occurs. By the induction hypothesis, there exists a rational run ρ' of $\mathscr{T}$. But ρ' is also a run of $\mathscr{T}$ (for all paths ξ in ρ, $\mathrm{Inf}(\xi)$ did not change). The step follows for Case 1.

Case 2. There exists a node $u \in D^*$ such that $s = \rho(u)$ is live and there exists a state $s' \in S$ that is live and that does not appear in u's subtree.

Define the RTA $\mathscr{T}_1$ obtained from $\mathscr{T}$ by removing all transitions from s and adding the "accepting" transition $(s, (s_d)_{d\in D})$ where $s_d = s$ for all $d \in D$. In fact, the transition is made accepting by adding s to all sets F of $(F, F') \in \mathscr{F}$. By construction, s is absorbing in $\mathscr{T}_1$. Thus, $\mathscr{T}_1$ has at least one live state (namely s) less. By the induction hypothesis, $\mathscr{T}_1$ has some rational accepting run ρ_1.

Define another RTA $\mathscr{T}_2$ obtained from $\mathscr{T}$ by setting the initial state to s and deleting s' from S (as well as all transitions that include s'). Thus, $\mathscr{T}_2$ has at least one live state (namely s') less. By the induction hypothesis, $\mathscr{T}_2$ has some rational accepting run ρ_2. Let S_2 be the set of states that occur in ρ_2. Moreover, for each $r \in S_2$, let ρ_2^r denote the (unique up to isomorphism) subtree of ρ_2 rooted at an r-node. Note that $\rho_2^s = \rho_2$.

Next, we define a tree ρ_1', which we obtain from ρ_1 as follows: Along each path of ρ_1, we are looking for the first occurrence of a node v such that $\rho_1(v) \in S_2$. For every such node v, we replace its subtree by $\rho_2^{\rho_1(v)}$. In particular, every subtree in ρ_1 whose root has state s is replaced by ρ_2.

Observe that ρ_1' is a rational run of $\mathscr{T}$. Further, ρ_1' is accepting: For a path ξ in ρ, we know that it is either a path that stays in ρ_1 (in which case there is $(F, F') \in \mathscr{F}$ with $F \cap \mathrm{Inf}(\xi) \neq \emptyset$ and $F' \cap \mathrm{Inf}(\xi) = \emptyset$), or it is a path that initially is in ρ_1 and then remains in ρ_2. In the latter case, $\mathrm{Inf}(\xi)$ is determined only by the suffix ξ_2 in ρ_2, for which we know that there is $(F,F') \in \mathscr{F}$ with $F \cap \mathrm{Inf}(\xi_2) \neq \emptyset$ and $F' \cap \mathrm{Inf}(\xi_2) = \emptyset$. The induction step follows for Case 2.

Case 3. Otherwise, all live states appear in all subtrees of nodes whose state is live.

Choose a path ξ_0 in ρ such that the set $\mathrm{Inf}(\xi_0)$ is the set of live states in S. Note that this is possible by assumption of Case 3. Since ρ is accepting, there is a pair $(F,F') \in \mathscr{F}$ such that $\mathrm{Inf}(\xi_0) \cap F \neq \emptyset$ and $\mathrm{Inf}(\xi_0) \cap F' = \emptyset$. Fix this pair (F,F') for the remainder of the proof. We observe:

(a) $\mathrm{Inf}(\xi_0)$ does not contain absorbing or vanishing states.

(b) If nonempty, F' only contains absorbing and vanishing states.

(c) $\mathrm{Inf}(\xi_0) \cap F$ is nonempty and contains a live state, say, s.

We build a rational run as follows:

Define $\mathscr{T}_1$ as in the second case above. State s is thus absorbing and not a live state in $\mathscr{T}_1$. Let ρ_1 be an accepting run of $\mathscr{T}_1$. By the induction hypothesis, we can suppose that ρ_1 is rational.

Define $\mathscr{T}_3$ as $\mathscr{T}$ with the following changes: S is replaced by $S \cup \{s_{\mathrm{new}}\}$ where s_{new} is a fresh "accepting" absorbing state (with corresponding absorbing transition added), each s that appears as s_d in a transition $(\hat{s}, (s_d)_{d \in D})$ in Δ is replaced by s_{new}, and finally we set s to be the initial state. State s is vanishing and not a live state. State s_{new} is absorbing. Let ρ_3 be a run of $\mathscr{T}_3$. By the induction hypothesis, we can assume that ρ_3 is rational.

Define run $\rho_{3,\mathrm{lim}}$ as the limit of the following process: Take ρ_3 and replace all subtrees of the nodes whose state is s_{new} with ρ_3. Let S_3 be the set of states that occur in $\rho_{3,\mathrm{lim}}$. Moreover, for each $r \in S_3$, let $\rho^r_{3,\mathrm{lim}}$ denote the (unique up to isomorphism) subtree of $\rho_{3,\mathrm{lim}}$ rooted at an r-node. In particular, we have $\rho^s_{3,\mathrm{lim}} = \rho_{3,\mathrm{lim}}$.

Similarly to Case 2, we obtain a tree ρ'_1 from ρ_1 as follows: Along each path of ρ_1, we are looking for the first occurrence of a node u such that $\rho_1(u) \in S_3$. For every such node u, we replace its subtree by $\rho^{\rho_1(u)}_{3,\mathrm{lim}}$. In particular, every subtree in ρ_1 whose root has state s is replaced by $\rho_{3,\mathrm{lim}}$.

By construction, ρ'_1 is a rational run of $\mathscr{T}$. One also verifies that ρ'_1 is accepting: Let ξ be a path in ρ'_1. If the path has a suffix that stays in ρ_1, acceptance by $\mathscr{T}$ follows from acceptance by $\mathscr{T}_1$. Otherwise, a suffix ξ' of ξ remains in $\rho_{3,\mathrm{lim}}$. If ξ' contains a finite number of s, then a suffix of it is in ρ_3. Acceptance by $\mathscr{T}$ follows from acceptance by $\mathscr{T}_1$.

It remains the case that ξ' is in $\rho_{3,\mathrm{lim}}$ and contains s infinitely often. From (c), we have that $s \in F$. Further, $\mathrm{Inf}(\xi')$ cannot contain vanishing states (they have no incoming transitions) and no absorbing states (otherwise, this contradicts the fact that s appears infinitely often). Thus, $\mathrm{Inf}(\xi') \subseteq \mathrm{Inf}(\xi_0)$. Together with (a) and (b), $\mathrm{Inf}(\xi') \cap F' = \emptyset$. It follows that ξ' is accepting by $\mathscr{T}$. The induction step follows for Case 3.

B Proof of Claim 1

Claim 1. Let $w_0, w_1, w_2, \ldots \in (\Sigma_{\leftrightarrow} \times \Omega)(\Sigma_{\leftarrow} \times \Omega)^*$. Moreover, let $w = w_0 w_1 w_2 \ldots$ be the concatenation of all w_i. Set $s_0 = \iota$ and, for $i \in \mathbb{N}$, $s_{i+1} = (\jmath_{i+1}, R_{i+1}) = \delta(\iota, w_0 \ldots w_i)$. Then, $w \in L(\mathscr{A}_\varphi) \iff$ the sequence $s_0, s_1, s_2, \ldots$ satisfies $\mathscr{F} \iff w \in L(\mathscr{A})$.

For $\jmath \in S_\varphi$ and $w = \alpha_0 \ldots \alpha_{n-1} \in (\Sigma \times \Omega)^*$, let

$$Visit_{\mathscr{A}_\varphi}(\jmath, w) = \{\delta(\jmath, \alpha_0), \ldots, \delta(\jmath, \alpha_1 \ldots \alpha_{n-1})\}$$

be the set of states that are traversed by $\mathscr{A}$ when reading w. Note that $\mathit{Visit}_{\mathscr{A}_\varphi}(\jmath,w)$ does not necessarily contain $\jmath$. For all $i \in \mathbb{N}$, we have $R_{i+1} = \mathit{Visit}_{\mathscr{A}_\varphi}(\jmath_i,w_i)$. With this, we get:

$$\begin{aligned}
& w \in L(\mathscr{A}_\varphi) \\
\iff\ & \exists (F,F') \in \mathscr{F}_\varphi\colon \mathit{Visit}^\infty_{\mathscr{A}_\varphi}(w) \cap F \neq \emptyset \text{ and } \mathit{Visit}^\infty_{\mathscr{A}_\varphi}(w) \cap F' = \emptyset \\
\iff\ & \exists (F,F') \in \mathscr{F}_\varphi\colon \begin{pmatrix} \mathit{Visit}_{\mathscr{A}_\varphi}(\jmath_i,w_i) \cap F \neq \emptyset & \text{for infinitely many } i \geq 0 \\ \mathit{Visit}_{\mathscr{A}_\varphi}(\jmath_i,w_i) \cap F' \neq \emptyset & \text{for finitely many } i \geq 0 \end{pmatrix} \\
\iff\ & \exists (F,F') \in \mathscr{F}_\varphi\colon \begin{pmatrix} R_{i+1} \cap F \neq \emptyset & \text{for infinitely many } i \geq 0 \\ R_{i+1} \cap F' \neq \emptyset & \text{for finitely many } i \geq 0 \end{pmatrix} \\
\iff\ & \exists (F,F') \in \mathscr{F}_\varphi\colon \begin{pmatrix} s_{i+1} \in G_F & \text{for infinitely many } i \geq 0 \\ s_{i+1} \in G_{F'} & \text{for finitely many } i \geq 0 \end{pmatrix} \\
\iff\ & \text{the sequence } s_0, s_1, s_2, \ldots \text{ satisfies the acceptance condition } \mathscr{F} \\
\iff\ & w \in L(\mathscr{A})
\end{aligned}$$

The last equivalence is due to the fact that the R-component is monotonically increasing when $\mathscr{A}$ is reading a word $\alpha_0 \ldots \alpha_{n-1} \in (\Sigma_{\leftrightarrow} \times \Omega)(\Sigma_{\leftarrow} \times \Omega)^*$. In fact, for $(\jmath'_0, R'_0) \in S$ and $(\jmath'_{i+1}, R'_{i+1}) = \delta((\jmath'_i, R'_i), \alpha_i)$, we have $R'_1 \subseteq R'_2 \subseteq \ldots \subseteq R'_n$.

C Correctness of $\mathscr{T}_\varphi$

We will show

$$L(\mathscr{T}_\varphi) = \{t \mid t \text{ is a strategy tree such that } f_t \text{ fulfills } \varphi\}.$$

We have to consider two inclusions:

Inclusion $\subseteq$: Suppose $t \in L(\mathscr{T}_\varphi)$. For $u \in D^*$, let $\lambda^u = (\lambda^u_1, \lambda^u_2)$ refer to $t(u)$. There is an accepting run $\rho : D^* \to S$ of $\mathscr{T}_\varphi$ on t. Let $w \in \Sigma_{\leftrightarrow}\Sigma^\omega$. We will show, using Claim 1, that $f_t(\!|w|\!) \in L(\mathscr{A}_\varphi)$.

- Suppose $w = d_0 d_1 \ldots d_{n-1} u$ where $d_0, \ldots, d_{n-1} \in \Sigma_{\leftrightarrow}\Sigma^*_{\leftarrow}$, with $n \in \mathbb{N}$, and $u \in \Sigma_{\leftrightarrow}\Sigma^\omega_{\leftarrow}$. In particular, seen as a word over Σ, w contains only finitely many letters from $\Sigma_{\leftrightarrow}$. We have

$$f_t(\!|w|\!) = \lambda^\varepsilon(\!|d_0|\!) \cdot \lambda^{d_0}(\!|d_1|\!) \cdot \lambda^{d_0 d_1}(\!|d_2|\!) \cdot \ldots \cdot \lambda^{d_0 d_1 \ldots d_{n-2}}(\!|d_{n-1}|\!) \cdot \lambda^{d_0 d_1 \ldots d_{n-1}}(\!|u|\!)\,.$$

By the definition of Δ, we have

$$\begin{aligned}
(\jmath_1, R_1) &:= \rho(d_0) = \delta(\iota, \lambda^\varepsilon(\!|d_0|\!)) \\
(\jmath_2, R_2) &:= \rho(d_0 d_1) = \delta(\rho(d_0), \lambda^{d_0}(\!|d_1|\!)) \\
(\jmath_3, R_3) &:= \rho(d_0 d_1 d_2) = \delta(\rho(d_0 d_1), \lambda^{d_0 d_1}(\!|d_2|\!)) \\
&\ \ \vdots \\
(\jmath_n, R_n) &:= \rho(d_0 d_1 d_2 \ldots d_{n-1}) = \delta(\rho(d_0 d_1 \ldots d_{n-2}), \lambda^{d_0 d_1 \ldots d_{n-2}}(\!|d_{n-1}|\!))
\end{aligned}$$

and $\lambda^{d_0 d_1 \ldots d_{n-1}}(\!|u|\!) \in L(\mathscr{A}_\varphi[\mathfrak{s}_n])$. This implies

$$\begin{aligned}
\mathfrak{s}_1 &= \delta_\varphi(\iota_\varphi, \lambda^{\varepsilon}(\!|d_0|\!))\\
\mathfrak{s}_2 &= \delta_\varphi(\mathfrak{s}_1, \lambda^{d_0}(\!|d_1|\!))\\
\mathfrak{s}_3 &= \delta_\varphi(\mathfrak{s}_2, \lambda^{d_0 d_1}(\!|d_2|\!))\\
&\vdots\\
\mathfrak{s}_n &= \delta_\varphi(\mathfrak{s}_{n-1}, \lambda^{d_1 \ldots d_{n-2}}(\!|d_{n-1}|\!))\,.
\end{aligned}$$

Therefore, together with $\lambda^{d_0 d_1 \ldots d_{n-1}}(\!|u|\!) \in L(\mathscr{A}_\varphi[\mathfrak{s}_n])$, we obtain $f_t(\!|w|\!) \in L(\mathscr{A}_\varphi)$.

- Suppose $w = d_0 d_1 d_2 \ldots$ where $d_0, d_1, d_2, \ldots \in \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^*$ for all $n \in \mathbb{N}$. Thus, w contains infinitely many letters from $\Sigma_{\leftrightarrow}$. We have

$$f_t(\!|w|\!) = \lambda^{\varepsilon}(\!|d_0|\!) \cdot \lambda^{d_0}(\!|d_1|\!) \cdot \lambda^{d_0 d_1}(\!|d_2|\!) \cdot \lambda^{d_0 d_1 d_2}(\!|d_3|\!) \cdot \ldots$$

Moreover, we have

$$\begin{aligned}
s_1 := \rho(d_0) &= \delta(\iota, \lambda^{\varepsilon}(\!|d_0|\!))\\
s_2 := \rho(d_0 d_1) &= \delta(\rho(d_0), \lambda^{d_0}(\!|d_1|\!))\\
s_3 := \rho(d_0 d_1 d_2) &= \delta(\rho(d_0 d_1), \lambda^{d_0 d_1}(\!|d_2|\!))\\
&\vdots
\end{aligned}$$

As ρ is an accepting run on t, the sequence $\iota, s_1, s_2, \ldots$ satisfies $\mathscr{F}$. By Claim 1, we obtain $f_t(\!|w|\!) \in L(\mathscr{A}_\varphi)$.

Inclusion $\supseteq$: Suppose t is a strategy tree such that f_t fulfills φ. Again, for $u \in D^*$, let $\lambda^u = (\lambda_1^u, \lambda_2^u)$ refer to $t(u)$. We will construct an accepting run $\rho : D^* \to S$ of $\mathscr{T}_\varphi$ on t. First of all, we let $\rho(\varepsilon) = \iota$.

Suppose that we defined $\rho(u)$ for $u = d_0 d_1 \ldots d_{n-1} \in D^*$ where $d_0, \ldots, d_{n-1} \in D = \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^*$, with $n \in \mathbb{N}$. For $d \in D$, we let

$$\rho(ud) = \delta(\rho(u), \lambda^u(\!|d|\!))\,.$$

Claim 3. For all $u \in D^*$ and $u' \in \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^* \cup \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^{\omega}$, the following hold:

$$f_t(\!|uu'|\!) = f_t(\!|u|\!) \cdot \lambda^u(\!|u'|\!) \tag{1}$$

$$\rho(u) = \delta(\iota, f_t(\!|u|\!)) \tag{2}$$

Proof of Claim 3. The first statement is due to the definition of f_t. The second statement follows from an easy induction on u (see also end of Section 3):

$$\begin{aligned}
\rho(\varepsilon) &= \delta(\iota, \varepsilon)\\
\rho(ud) &= \delta(\rho(u), \lambda^u(\!|d|\!))\\
&= \delta(\delta(\iota, f_t(\!|u|\!)), \lambda^u(\!|d|\!))\\
&= \delta(\iota, f_t(\!|u|\!) \cdot \lambda^u(\!|d|\!))\\
&= \delta(\iota, f_t(\!|ud|\!))
\end{aligned}$$

Note that the last equality is due to (1). ■

Let $u \in D^*$ and $(\jmath, R) = \rho(u)$. Let us establish that $(\rho(u), \lambda^u, (\rho(ud))_{d\in D})$ is a transition of $\mathscr{T}_\varphi$:

(T1) We have $\rho(ud) = \delta(\rho(u), \lambda^u(\!|d|\!))$ by the definition of ρ.

(T2) Let $u' \in \Sigma_{\leftrightarrow}\Sigma_{\leftarrow}^{\omega}$. As f_t fulfills φ, we have $f_t(\!|uu'|\!) \in L(\mathscr{A}_\varphi)$. By Claim 3(1), $\lambda^u(\!|u'|\!) \in L(\mathscr{A}_\varphi[\delta_\varphi(\iota_\varphi, f_t(\!|u|\!))])$. By means of Claim 3(2) and the definition of $\mathscr{A}$ wrt. to $\mathscr{A}_\varphi$, we can deduce $\lambda^u(\!|u'|\!) \in L(\mathscr{A}_\varphi[\jmath])$.

Finally, we show that ρ is accepting. Let $d_0, d_1, d_2, \ldots \in D$ and consider the path $\xi = d_0d_1d_2\ldots$ along with the induced infinite sequence

$$\rho(\varepsilon), \rho(d_0), \rho(d_0d_1), \rho(d_0d_1d_2), \rho(d_0d_1d_2d_3), \ldots$$

Recall that we have

$$f_t(\!|w|\!) = \lambda^{\varepsilon}(\!|d_0|\!) \cdot \lambda^{d_0}(\!|d_1|\!) \cdot \lambda^{d_0d_1}(\!|d_2|\!) \cdot \lambda^{d_0d_1d_2}(\!|d_3|\!) \cdot \ldots$$

as well as

$$\begin{aligned} \rho(d_0) &= \delta(\iota, \lambda^{\varepsilon}(\!|d_0|\!)) \\ \rho(d_0d_1) &= \delta(\rho(d_0), \lambda^{d_0}(\!|d_1|\!)) \\ \rho(d_0d_1d_2) &= \delta(\rho(d_0d_1), \lambda^{d_0d_1}(\!|d_2|\!)) \\ &\vdots \end{aligned}$$

As $f_t(\!|w|\!) \in L(\mathscr{A}_\varphi)$, by Claim 1, we have that ξ is accepting.

D Details for Proof of Lemma 5

There are now two directions to show.

Claim 4. If there is a winning strategy profile in $\mathscr{G}_\varphi$, then there is a distributed algorithm that fulfills φ.

Proof of Claim 4. Let $g = (g_1, g_2)$ be a winning strategy profile in $\mathscr{G}_\varphi$, with $g_p : \mathscr{O}_p^+ \to Y_p$.

We define $\nu : \Sigma^* \to V$ and $\eta : \Sigma^* \to (V \times \Sigma)^*$ inductively by

$$\begin{aligned} \nu(\varepsilon) &= \iota \\ \eta(\varepsilon) &= \varepsilon \\ \nu(w\langle x_1 \rightleftharpoons x_2\rangle) &= \tau(\nu(w), \langle x_1 \rightleftharpoons x_2\rangle, (y_1, y_2)) \\ \eta(w\langle x_1 \rightleftharpoons x_2\rangle) &= \eta(w) \cdot (\nu(w), \langle x_1 \rightleftharpoons x_2\rangle) \end{aligned}$$

where $y_p = g_p([\![\eta(w\langle x_1 \rightleftharpoons x_2\rangle)]\!]_p^{\mathsf{game}})$. That is, $\nu(w)$ is the node which is visited after input word w under strategy profile g, and $\eta(w)$ is the path corresponding to w in the game, starting at ι and applying g.

For every $p \in \{1, 2\}$ and $w \in \Sigma_{\leftrightarrow}\Sigma^*$, we define

$$f_p([\![w]\!]_p) = g_p([\![\eta(w)]\!]_p^{\mathsf{game}}).$$

This is well-defined by construction of the game, using the fact that g is known by both players. Indeed, $[\![w]\!]_1 = w$, then it is possible to compute $[\![\eta(w)]\!]_1^{\mathsf{game}}$ from $[\![w]\!]_1$. For player 2, one can show inductively that for all $w, w' \in \Sigma_{\leftrightarrow}\Sigma^*$ such that $[\![w]\!]_2 = [\![w']\!]_2$, $[\![\eta(w)]\!]_2^{\mathsf{game}} = [\![\eta(w')]\!]_2^{\mathsf{game}}$.

Let $w = \sigma_0\sigma_1\sigma_2\ldots \in \Sigma_{\leftrightarrow}\Sigma^\omega$. We have to show that $f(\!|w|\!) \in L(\varphi) = L(\mathscr{A}_\varphi)$. Let us determine the sequence $s_0, s_1, s_2, \ldots$ of states of $\mathscr{A}$ visited while reading $f(\!|w|\!)$. Set $s_0 = \iota$ and, for every $r \in \mathbb{N}$,

$$s_{r+1} = \delta\big(s_r, \big[\sigma_r, \big(f_1(\sigma_0\ldots\sigma_r), f_2([\![\sigma_0\ldots\sigma_r]\!]_2)\big)\big]\big)\,.$$

For all $r \in \mathbb{N}$, we have

$$\begin{aligned} s_{r+1} &= \delta\big(s_r, \big[\sigma_r, \big(f_1(\sigma_0\ldots\sigma_r), f_2([\![\sigma_0\ldots\sigma_r]\!]_2)\big)\big]\big) \\ &= \delta\big(s_r, \big[\sigma_r, \big(g_1(\eta(\sigma_0\ldots\sigma_r)), g_2([\![\eta(\sigma_0\ldots\sigma_r)]\!]_2^{\mathsf{game}})\big)\big]\big) \\ &= \tau\big(s_r, \sigma_r, \big(g_1(\eta(\sigma_0\ldots\sigma_r)), g_2([\![\eta(\sigma_0\ldots\sigma_r)]\!]_2^{\mathsf{game}})\big)\big)\,. \end{aligned}$$

Since g is winning and by the winning condition W of the game, we obtain $f(\!|w|\!) \in L(\mathscr{A}_\varphi)$, which concludes the proof of Claim 4. ■

Claim 5. If there is a distributed algorithm that fulfills φ, then there is a winning strategy profile in $\mathscr{G}_\varphi$.

Proof of Claim 5. Let $f = (f_1, f_2)$ be a distributed algorithm that fulfills φ. Due to Lemma 2, we can assume that for all words $w, w' \in \{\varepsilon\} \cup \Sigma_{\leftrightarrow}\Sigma^*$ satisfying $\delta(\iota, f(\!|w|\!)) = \delta(\iota, f(\!|w'|\!))$, we have $f_2(wu) = f_2(w'u)$ for all $u \in \Sigma_{\leftrightarrow}\Sigma^*_{\perp\leftarrow}$.

We have to define a strategy profile $g = (g_1, g_2)$ for the game. Recall that $g_p : \mathscr{O}_p^+ \to A_p$. For every $s \in S$, we will define an "access string" $w_s \in \Sigma^*$ as follows: Set $w_\iota = \varepsilon$. Moreover, for $s \in S \setminus \{\iota\}$, fix any word $w_s \in \Sigma_{\leftrightarrow}\Sigma^*$ such that $\delta(\iota, f(\!|w_s|\!)) = s$. If no such word exists, we let $w_s = \varepsilon$.

Note that, if the first environment action is not from $\Sigma_{\leftrightarrow}$, then we can output anything. So fix an arbitrary pair $(\mathsf{y}_1, \mathsf{y}_2) \in Y_1 \times Y_2$. Now, g is given as follows:

$$g_1 : \begin{cases} (V \times \Sigma)^+ \to Y_1 \\ (v_0, \sigma_0)\ldots(v_n, \sigma_n) \;\mapsto\; \begin{cases} f_1(\sigma_0\ldots\sigma_n) & \text{if } \sigma_0 \in \Sigma_{\leftrightarrow} \\ \mathsf{y}_1 & \text{otherwise} \end{cases} \end{cases}$$

$$g_2 : \begin{cases} \mathscr{O}_2^+ \to Y_2 \\ o \;\mapsto\; \mathsf{y}_2 & \text{for } o \in (\Sigma_{\perp\leftarrow})\mathscr{O}_2^* \\ o \cdot (s, \langle x_1 \leftrightarrow x_2\rangle) \cdot u \;\mapsto\; f_2(w_s \cdot \langle x_1 \leftrightarrow x_2\rangle \cdot u) & \text{for } o \in \{\varepsilon\} \cup (S \times \Sigma_{\leftrightarrow})\mathscr{O}_2^* \text{ and } u \in \Sigma^*_{\perp\leftarrow} \end{cases}$$

It remains to show that g is winning. So let $\pi = (s_0, \sigma_0)(s_1, \sigma_1)(s_2, \sigma_2)\ldots$ be a play that is compatible with g, with $s_r = (\mathfrak{s}_r, R_r)$. By our assumption that $(\Sigma_{\leftarrow} \times \Omega)(\Sigma \times \Omega)^\omega \subseteq L(\varphi)$, we only need to consider the case $\sigma_0 \in \Sigma_{\leftrightarrow}$. For all $r \in \mathbb{N}$, we have

$$(\mathfrak{s}_{r+1}, R_{r+1}) = \tau((\mathfrak{s}_r, R_r), \sigma_r, (a_1^r, a_2^r)) = \delta\big((\mathfrak{s}_r, R_r), (\sigma_r, (a_1^r, a_2^r))\big)$$

where $a_p^r = g_p([\![\pi_{\le r}]\!]_p^{\mathsf{game}})$ with $\pi_{\le r} = (s_0, \sigma_0)\ldots(s_r, \sigma_r)$.

It is enough to show that, for all $r \in \mathbb{N}$, we have

$$(\mathfrak{s}_{r+1}, R_{r+1}) = \delta\big((\mathfrak{s}_r, R_r), \big[\sigma_r, \big(f_1(\sigma_0\ldots\sigma_r), f_2([\![\sigma_0\ldots\sigma_r]\!]_2)\big)\big]\big)\,.$$

We proceed by induction on the number k of letters from $\Sigma_{\leftrightarrow}$ in $\pi_{\le r}$. So suppose

$$\pi_{\le r} = \underbrace{(s_0, \sigma_0)\ldots(s_{m-1}, \sigma_{m-1})}_{=:\,w}(s_m, \langle x_1^m \leftrightarrow x_2^m\rangle)(s_{m+1}, \langle x_1^{m+1} \leftarrow x_2^{m+1}\rangle)\ldots(s_r, \langle x_1^r \leftarrow x_2^r\rangle)\,.$$

Set $[\![\varepsilon]\!]_2^{\mathsf{game}} = \varepsilon$ and let $u = \langle \bot \leftarrow x_2^{m+1}\rangle \ldots \langle \bot \leftarrow x_2^r\rangle$. Then,

$$\begin{aligned}
s_{r+1} &= \delta\big(s_r, \big[\sigma_r, \big(g_1([\![\pi_{\le r}]\!]_1^{\mathsf{game}}), g_2([\![\pi_{\le r}]\!]_2^{\mathsf{game}})\big)\big]\big)\\
&= \delta\big(s_r, \big[\sigma_r, \big(g_1(\pi_{\le r}), g_2([\![w]\!]_2^{\mathsf{game}} \cdot (s_m, \langle x_1^m \leftrightarrow x_2^m\rangle) \cdot u)\big)\big]\big)\\
&= \delta\big(s_r, \big[\sigma_r, \big(f_1(\sigma_0 \ldots \sigma_r), f_2(w_{s_m} \cdot \langle x_1^m \leftrightarrow x_2^m\rangle \cdot u)\big)\big]\big)\\
&\overset{(*)}{=} \delta\big(s_r, \big[\sigma_r, \big(f_1(\sigma_0 \ldots \sigma_r), f_2(\sigma_0 \ldots \sigma_{m-1} \cdot \langle x_1^m \leftrightarrow x_2^m\rangle \cdot u)\big)\big]\big)\\
&= \delta\big(s_r, \big[\sigma_r, \big(f_1(\sigma_0 \ldots \sigma_r), f_2([\![\sigma_0 \ldots \sigma_r]\!]_2)\big)\big]\big)
\end{aligned}$$

Equation $(*)$ is trivial for $m = 0$ (i.e., $k = 1$), as then $w_{s_m} = \varepsilon$ (by definition). Otherwise, it follows from the induction hypothesis: The word $\sigma_0 \ldots \sigma_{m-1}$ contains $k-1$ signals from $\Sigma_{\leftrightarrow}$. That is, if $m \geq 1$, then $s_m = \delta(\iota, f(\!|\sigma_0 \ldots \sigma_{m-1}|\!))$. ■

E Proof of Lemma 6

Lemma 6. *For all $w, w' \in \Sigma^*$ and $p \in \{1,2\}$, the following hold:*

(a) $[\![\langle\!\langle w\rangle\!\rangle]\!]_p = [\![\langle\!\langle w'\rangle\!\rangle]\!]_p \implies [\![w]\!]_{\mathrm{sim}_p(w)} = [\![w']\!]_{\mathrm{sim}_p(w')}$

(b) $[\![w]\!]_p = [\![w']\!]_p \implies [\![\langle\!\langle w\rangle\!\rangle]\!]_{\mathrm{sim}_p(w)} = [\![\langle\!\langle w'\rangle\!\rangle]\!]_{\mathrm{sim}_p(w')}$

Part (a)

First, assume $p = 1$. Observe that for $w \in \Sigma^*$, $\langle\!\langle w\rangle\!\rangle \in (\Sigma')^*$. Further, within the domain $(\Sigma')^*$, $[\![\cdot]\!]_1$ is the identity. Together with the injectivity of $\langle\!\langle\cdot\rangle\!\rangle$, $[\![\langle\!\langle w\rangle\!\rangle]\!]_p = [\![\langle\!\langle w'\rangle\!\rangle]\!]_p$ implies $w = w'$; the lemma's statement follows for $p = 1$.

Second, assume $p = 2$ and that $[\![\langle\!\langle w\rangle\!\rangle]\!]_2 = [\![\langle\!\langle w'\rangle\!\rangle]\!]_2$. For $p' \in P$, let $\langle\!\langle\cdot\rangle\!\rangle_{p'}$ denote the transduction defined by the same transducer but with initial state p'. In particular, we have $\langle\!\langle\cdot\rangle\!\rangle = \langle\!\langle\cdot\rangle\!\rangle_1$. We start by observing that the function $[\![\cdot]\!]_2$ is length preserving and the projection onto a sequence of communication graphs is the same in $\langle\!\langle w\rangle\!\rangle$ and $\langle\!\langle w'\rangle\!\rangle$. Moreover, the latter are of the form

$$\begin{aligned}
\langle\!\langle w\rangle\!\rangle &= \hat{u}\,\sigma\,\overbrace{\langle z_1 \leftarrow x_1\rangle \ldots \langle z_n \leftarrow x_n\rangle}^{\hat{v}}\\
\langle\!\langle w'\rangle\!\rangle &= \hat{u}\,\sigma\,\underbrace{\langle z_1' \leftarrow x_1\rangle \ldots \langle z_n' \leftarrow x_n\rangle}_{\hat{v}'}
\end{aligned}$$

for some $\hat{u} \in (\Sigma')^*$ and $\sigma \in \{\varepsilon\} \cup \Sigma'_{\leftrightarrow}$ such that $\sigma \neq \varepsilon$ or $\hat{u} = \sigma = \varepsilon$.

- Suppose $\hat{u} = \sigma = \varepsilon$. Then, by the definition of $\langle\!\langle\cdot\rangle\!\rangle$, we have $w = \hat{v}$ and $w' = \hat{v}'$. We deduce $[\![w]\!]_{\mathrm{sim}_2(w)} = [\![w']\!]_{\mathrm{sim}_2(w')}$ with $\mathrm{sim}_2(w) = 2$.
- Suppose that $\varepsilon \neq \sigma = \langle \chi_1 \leftrightarrow \chi_2\rangle \neq \langle \# \leftrightarrow \#\rangle$. Then, $w = uv$ and $w' = u'v'$ for some u, v, u', v' such that $\langle\!\langle u\rangle\!\rangle = \hat{u}$ and $\langle\!\langle u'\rangle\!\rangle = \hat{u}$ and $\langle\!\langle v\rangle\!\rangle_{\mathrm{sim}_1(u)} = \sigma\hat{v}$ and $\langle\!\langle v'\rangle\!\rangle_{\mathrm{sim}_1(u)} = \sigma\hat{v}'$. By injectivity of $\langle\!\langle\cdot\rangle\!\rangle$, we have $u = u'$.
  - Suppose $\mathrm{sim}_1(u) = 1$, i.e., $\mathrm{sim}_2(u) = 2$. By the definition of $\langle\!\langle\cdot\rangle\!\rangle$, we obtain $v = \sigma\hat{v}$ and $v' = \sigma\hat{v}'$. Therefore, $[\![w]\!]_2 = [\![w']\!]_2 = u\,\sigma\langle \bot \leftarrow x_1\rangle \ldots \langle \bot \leftarrow x_n\rangle$.

- Suppose $\mathrm{sim}_1(u) = 2$, i.e., $\mathrm{sim}_2(u) = 1$. By the definition of $\langle\!\langle \cdot \rangle\!\rangle$, we can deduce that $v = \langle \chi_2 \leftrightarrow \chi_1 \rangle \langle x_1 \rightarrow z_1 \rangle \dots \langle x_n \rightarrow z_n \rangle$ and $v' = \langle \chi_2 \leftrightarrow \chi_1 \rangle \langle x_1 \rightarrow z'_1 \rangle \dots \langle x_n \rightarrow z'_n \rangle$. We conclude $[\![w]\!]_1 = [\![w']\!]_1 = u \langle \chi_2 \leftrightarrow \chi_1 \rangle \langle x_1 \rightarrow \perp \rangle \dots \langle x_n \rightarrow \perp \rangle$.

- Suppose that $\sigma = \langle \# \leftrightarrow \# \rangle$. Then, $n \geq 1$. Moreover, $w = uv$ and $w' = u'v'$ for some u, v, u', v' such that $\langle\!\langle u \rangle\!\rangle = \hat{u}$ and $\langle\!\langle u' \rangle\!\rangle = \hat{u}$ and $\langle\!\langle v \rangle\!\rangle_{\mathrm{sim}_1(u)} = \sigma \hat{v}$ and $\langle\!\langle v' \rangle\!\rangle_{\mathrm{sim}_1(u)} = \sigma \hat{v}'$. By injectivity of $\langle\!\langle \cdot \rangle\!\rangle$, we have $u = u'$.
  - Suppose $\mathrm{sim}_1(u) = 1$. By the definition of $\langle\!\langle \cdot \rangle\!\rangle$, we obtain $v = \langle x_1 \rightarrow z_1 \rangle \dots \langle x_n \rightarrow z_n \rangle$ and $v' = \langle x_1 \rightarrow z'_1 \rangle \dots \langle x_n \rightarrow z'_n \rangle$. Therefore, $\mathrm{sim}_2(w) = \mathrm{sim}_2(w') = 1$. We have that $[\![w]\!]_1 = [\![uv]\!]_1 = [\![u]\!]_1 \langle x_1 \rightarrow \perp \rangle \dots \langle x_1 \rightarrow \perp \rangle = [\![uv']\!]_1 = [\![w']\!]_1$.
  - Suppose $\mathrm{sim}_1(u) = 2$. Now, by the definition of $\langle\!\langle \cdot \rangle\!\rangle$, we obtain $v = \langle x_1 \leftarrow z_1 \rangle \dots \langle x_n \leftarrow z_n \rangle$ and $v' = \langle x_1 \leftarrow z'_1 \rangle \dots \langle x_n \leftarrow z'_n \rangle$. Therefore, $\mathrm{sim}_2(w) = \mathrm{sim}_2(w') = 2$. We have that $[\![w]\!]_2 = [\![uv]\!]_2 = [\![u]\!]_2 \langle \perp \leftarrow x_1 \rangle \dots \langle \perp \leftarrow x_n \rangle = [\![uv']\!]_2 = [\![w']\!]_2$.

Part (b)

Suppose $[\![w]\!]_p = [\![w']\!]_p$. Note that this implies $\mathrm{sim}_p(w) = \mathrm{sim}_p(w') =: p_w$.

First, assume that one of the following holds:

- $w = u\langle x_1 \leftrightarrow x_2 \rangle$, or
- $w = u\langle x_1 \leftarrow x_2 \rangle$ and $p = 1$, or
- $w = u\langle x_1 \rightarrow x_2 \rangle$ and $p = 2$.

Then, $w = w'$ and we are done.

For the remaining cases, we proceed by induction. The statement is obvious for $w = \varepsilon$.

Now, assume $w = u\langle x_1 \leftarrow x_2 \rangle$ and $p = 2$. Then, $p_w = 2$ and $[\![w]\!]_2 = [\![u]\!]_2 \langle \perp \leftarrow x_2 \rangle = [\![w']\!]_2$. Thus, we have $w' = u'\langle x'_1 \leftarrow x_2 \rangle$ for some u' and x'_1 such that $[\![u]\!]_2 = [\![u']\!]_2$. The latter implies $\mathrm{sim}_2(u) = \mathrm{sim}_2(u') =: p_u$. By induction hypothesis, we get $[\![\langle\!\langle u \rangle\!\rangle]\!]_{p_u} = [\![\langle\!\langle u' \rangle\!\rangle]\!]_{p_u}$.

Suppose $p_u = 1$. Then,

$$\begin{aligned} [\![\langle\!\langle w \rangle\!\rangle]\!]_{p_w} &= [\![\langle\!\langle u\langle x_1 \leftarrow x_2 \rangle \rangle\!\rangle]\!]_2 \\ &= [\![\langle\!\langle u \rangle\!\rangle \langle \# \leftrightarrow \# \rangle \langle x_1 \leftarrow x_2 \rangle]\!]_2 \\ &= \langle\!\langle u \rangle\!\rangle \langle \# \leftrightarrow \# \rangle \langle \perp \leftarrow x_2 \rangle \end{aligned}$$

and from $p_u = 1$ and $[\![u]\!]_2 = [\![u']\!]_2$

$$\begin{aligned} &= \langle\!\langle u' \rangle\!\rangle \langle \# \leftrightarrow \# \rangle \langle \perp \leftarrow x_2 \rangle \\ &= [\![\langle\!\langle u' \rangle\!\rangle \langle \# \leftrightarrow \# \rangle \langle x'_1 \leftarrow x_2 \rangle]\!]_2 \\ &= [\![\langle\!\langle u'\langle x'_1 \leftarrow x_2 \rangle \rangle\!\rangle]\!]_2 = [\![\langle\!\langle w' \rangle\!\rangle]\!]_{p_w} \end{aligned}$$

Suppose $p_u = 2$. Then,

$$\begin{aligned} [\![\langle\!\langle w \rangle\!\rangle]\!]_{p_w} &= [\![\langle\!\langle u\langle x_1 \leftarrow x_2 \rangle \rangle\!\rangle]\!]_2 \\ &= [\![\langle\!\langle u \rangle\!\rangle \langle x_1 \leftarrow x_2 \rangle]\!]_2 \\ &= [\![\langle\!\langle u \rangle\!\rangle]\!]_2 \langle \perp \leftarrow x_2 \rangle \end{aligned}$$

and by induction hypothesis

$$\begin{aligned}
&= \llbracket \langle\!\langle u' \rangle\!\rangle \rrbracket_2 \langle \bot \leftarrowtriangle x_2 \rangle \\
&= \llbracket \langle\!\langle u' \rangle\!\rangle \langle x_1' \leftarrowtriangle x_2 \rangle \rrbracket_2 \\
&= \llbracket \langle\!\langle u' \langle x_1' \leftarrowtriangle x_2 \rangle \rangle\!\rangle \rrbracket_2 = \llbracket \langle\!\langle w' \rangle\!\rangle \rrbracket_{p_w}
\end{aligned}$$

Now, assume $w = u\langle x_1 \rightarrowtriangle x_2 \rangle$ and $p = 1$. Then, $p_w = 2$ and $\llbracket w \rrbracket_1 = \llbracket u \rrbracket_1 \langle x_1 \rightarrowtriangle \bot \rangle = \llbracket w' \rrbracket_1$. Thus, we have $w' = u'\langle x_1 \rightarrowtriangle x_2' \rangle$ for some u' and x_2' such that $\llbracket u \rrbracket_1 = \llbracket u' \rrbracket_1$. The latter implies $\mathrm{sim}_1(u) = \mathrm{sim}_1(u') =: p_u$. By induction hypothesis, we get $\llbracket \langle\!\langle u \rangle\!\rangle \rrbracket_{p_u} = \llbracket \langle\!\langle u' \rangle\!\rangle \rrbracket_{p_u}$.

Suppose $p_u = 1$. Then,

$$\begin{aligned}
\llbracket \langle\!\langle w \rangle\!\rangle \rrbracket_{p_w} &= \llbracket \langle\!\langle u \langle x_1 \rightarrowtriangle x_2 \rangle \rangle\!\rangle \rrbracket_2 \\
&= \llbracket \langle\!\langle u \rangle\!\rangle \langle \# \leftrightarrowtriangle \# \rangle \langle x_2 \leftarrowtriangle x_1 \rangle \rrbracket_2 \\
&= \langle\!\langle u \rangle\!\rangle \langle \# \leftrightarrowtriangle \# \rangle \langle \bot \leftarrowtriangle x_1 \rangle
\end{aligned}$$

and from $p_u = 1$ and $\llbracket u \rrbracket_1 = \llbracket u' \rrbracket_1$

$$\begin{aligned}
&= \langle\!\langle u' \rangle\!\rangle \langle \# \leftrightarrowtriangle \# \rangle \langle \bot \leftarrowtriangle x_1 \rangle \\
&= \llbracket \langle\!\langle u' \rangle\!\rangle \langle \# \leftrightarrowtriangle \# \rangle \langle x_2' \leftarrowtriangle x_1 \rangle \rrbracket_2 \\
&= \llbracket \langle\!\langle u' \langle x_1 \rightarrowtriangle x_2' \rangle \rangle\!\rangle \rrbracket_2 = \llbracket \langle\!\langle w' \rangle\!\rangle \rrbracket_{p_w}
\end{aligned}$$

Suppose $p_u = 2$. Then,

$$\begin{aligned}
\llbracket \langle\!\langle w \rangle\!\rangle \rrbracket_{p_w} &= \llbracket \langle\!\langle u \langle x_1 \rightarrowtriangle x_2 \rangle \rangle\!\rangle \rrbracket_2 \\
&= \llbracket \langle\!\langle u \rangle\!\rangle \langle x_2 \leftarrowtriangle x_1 \rangle \rrbracket_2 \\
&= \llbracket \langle\!\langle u \rangle\!\rangle \rrbracket_2 \langle \bot \leftarrowtriangle x_1 \rangle
\end{aligned}$$

and by induction hypothesis

$$\begin{aligned}
&= \llbracket \langle\!\langle u' \rangle\!\rangle \rrbracket_2 \langle \bot \leftarrowtriangle x_1 \rangle \\
&= \llbracket \langle\!\langle u' \rangle\!\rangle \langle x_2' \leftarrowtriangle x_1 \rangle \rrbracket_2 \\
&= \llbracket \langle\!\langle u' \langle x_1 \rightarrowtriangle x_2' \rangle \rangle\!\rangle \rrbracket_2 = \llbracket \langle\!\langle w' \rangle\!\rangle \rrbracket_{p_w}
\end{aligned}$$

F Proof of Lemma 8

Lemma 8. *Let $\varphi \in \mathrm{LTL}(\mathcal{N})$. The following statements are equivalent:*

(i) There is a distributed algorithm f (over $\mathcal{N}$) such that, for all $w \in \Sigma^\omega$, $f\llparenthesis w \rrparenthesis \in L(\varphi)$.

(ii) There is a distributed algorithm f' (over $\mathcal{N}'$) such that, for all $w \in \Sigma^\omega$, $f'\llparenthesis \langle\!\langle w \rangle\!\rangle \rrparenthesis \in \langle\!\langle L(\varphi) \rangle\!\rangle$.

We start by showing *(i)* $\rightarrow$ *(ii)*. Let $f = (f_1, f_2)$ be a distributed algorithm over $\mathcal{N}$ that fulfills $L(\varphi)$. Let $f' = (f_1', f_2')$ be a distributed algorithm over $\mathcal{N}'$ such that, for all $w \in \Sigma^+$, $w' \in (\Sigma')^+$, and $p \in \{1,2\}$,

$$\begin{aligned}
f_p'(\llbracket \langle\!\langle w \rangle\!\rangle \rrbracket_p) &:= f_{\mathrm{sim}_p(w)}(\llbracket w \rrbracket_{\mathrm{sim}_p(w)}) \\
f_p'(w'\langle \# \leftrightarrowtriangle \# \rangle) &:= \# .
\end{aligned} \tag{3}$$

Note that this is well-defined due to Lemma 6(a). We have to show that, for all $w \in \Sigma^\omega$, we get $f'(\!|\llangle w \rrangle|\!) \in \llangle L(\varphi) \rrangle$. This follows from the fact that, for all $w \in \Sigma^*$, we have

$$f'(\!|\llangle w \rrangle|\!) = \llangle f(\!|w|\!) \rrangle$$

which we show by induction (in the following, let $f'(u)$ stand for $(f'_1(\llbracket u \rrbracket_1), f'_2(\llbracket u \rrbracket_2))$):

- From the definitions, we obtain $f'(\!|\llangle \varepsilon \rrangle|\!) = \llangle f(\!|\varepsilon|\!) \rrangle$.
- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 1$ and $\hat{w} = w\langle x_1 \leftarrow x_2 \rangle$ (therefore, $\mathrm{sim}_1(\hat{w}) = 1$), we have

$$\begin{aligned} f'(\!|\llangle w\langle x_1 \leftarrow x_2 \rangle \rrangle|\!) &= f'(\!|\llangle w \rrangle \langle x_1 \leftarrow x_2 \rangle|\!) \\ &= f'(\!|\llangle w \rrangle|\!) \cdot \big(\langle x_1 \leftarrow x_2 \rangle, f'(\llangle w \rrangle \langle x_1 \leftarrow x_2 \rangle)\big) \\ &= f'(\!|\llangle w \rrangle|\!) \cdot \big(\langle x_1 \leftarrow x_2 \rangle, f'(\llangle \hat{w} \rrangle)\big) \\ &\overset{(3)}{=} f'(\!|\llangle w \rrangle|\!) \cdot \big(\langle x_1 \leftarrow x_2 \rangle, (f_1(\llbracket \hat{w} \rrbracket_1), f_2(\llbracket \hat{w} \rrbracket_2))\big) \\ &\overset{\mathrm{IH}}{=} \llangle f(\!|w|\!) \rrangle \cdot \big(\langle x_1 \leftarrow x_2 \rangle, (f_1(\llbracket \hat{w} \rrbracket_1), f_2(\llbracket \hat{w} \rrbracket_2))\big) \end{aligned}$$

  and from $\mathrm{sim}_1(f(\!|w|\!)) = \mathrm{sim}_1(w) = 1$ (because the projection of $f(\!|w|\!)$ to Σ equals w)

$$\begin{aligned} &= \llangle f(\!|w|\!) \cdot (\langle x_1 \leftarrow x_2 \rangle, (f_1(\llbracket \hat{w} \rrbracket_1), f_2(\llbracket \hat{w} \rrbracket_2))) \rrangle \\ &= \llangle f(\!|w\langle x_1 \leftarrow x_2 \rangle|\!) \rrangle = \llangle f(\!|\hat{w}|\!) \rrangle \end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 1$ and $\hat{w} = w\langle x_1 \leftrightarrow x_2 \rangle$ (therefore, $\mathrm{sim}_1(\hat{w}) = 1$), we have

$$\begin{aligned} f'(\!|\llangle w\langle x_1 \leftrightarrow x_2 \rangle \rrangle|\!) &= f'(\!|\llangle w \rrangle \langle x_1 \leftrightarrow x_2 \rangle|\!) \\ &= f'(\!|\llangle w \rrangle|\!) \cdot \big(\langle x_1 \leftrightarrow x_2 \rangle, f'(\llangle w \rrangle \langle x_1 \leftrightarrow x_2 \rangle)\big) \\ &= f'(\!|\llangle w \rrangle|\!) \cdot \big(\langle x_1 \leftrightarrow x_2 \rangle, f'(\llangle \hat{w} \rrangle)\big) \\ &\overset{(3)}{=} f'(\!|\llangle w \rrangle|\!) \cdot \big(\langle x_1 \leftrightarrow x_2 \rangle, (f_1(\llbracket \hat{w} \rrbracket_1), f_2(\llbracket \hat{w} \rrbracket_2))\big) \\ &\overset{\mathrm{IH}}{=} \llangle f(\!|w|\!) \rrangle \cdot \big(\langle x_1 \leftrightarrow x_2 \rangle, (f_1(\llbracket \hat{w} \rrbracket_1), f_2(\llbracket \hat{w} \rrbracket_2))\big) \end{aligned}$$

  and from $\mathrm{sim}_1(f(\!|w|\!)) = \mathrm{sim}_1(w) = 1$

$$\begin{aligned} &= \llangle f(\!|w|\!) \cdot \big(\langle x_1 \leftrightarrow x_2 \rangle, (f_1(\llbracket \hat{w} \rrbracket_1), f_2(\llbracket \hat{w} \rrbracket_2))\big) \rrangle \\ &= \llangle f(\!|w\langle x_1 \leftrightarrow x_2 \rangle|\!) \rrangle = \llangle f(\!|\hat{w}|\!) \rrangle \end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 1$ and $\hat{w} = w\langle x_1 \rightarrow x_2 \rangle$ (therefore, $\mathrm{sim}_1(\hat{w}) = 2$), we have

$$\begin{aligned} f'(\!|\llangle w\langle x_1 \rightarrow x_2 \rangle \rrangle|\!) &= f'(\!|\llangle w \rrangle \langle \# \leftrightarrow \# \rangle \langle x_2 \leftarrow x_1 \rangle|\!) \\ &= f'(\!|\llangle w \rrangle|\!)(\langle \# \leftrightarrow \# \rangle, (\#,\#))(\langle x_2 \leftarrow x_1 \rangle, f'(\llangle w \rrangle \langle \# \leftrightarrow \# \rangle \langle x_2 \leftarrow x_1 \rangle)) \\ &= f'(\!|\llangle w \rrangle|\!)(\langle \# \leftrightarrow \# \rangle, (\#,\#))(\langle x_2 \leftarrow x_1 \rangle, f'(\llangle \hat{w} \rrangle)) \\ &\overset{(3)}{=} f'(\!|\llangle w \rrangle|\!)(\langle \# \leftrightarrow \# \rangle, (\#,\#))(\langle x_2 \leftarrow x_1 \rangle, (f_2(\llbracket \hat{w} \rrbracket_2), f_1(\llbracket \hat{w} \rrbracket_1))) \\ &\overset{\mathrm{IH}}{=} \llangle f(\!|w|\!) \rrangle(\langle \# \leftrightarrow \# \rangle, (\#,\#))(\langle x_2 \leftarrow x_1 \rangle, (f_2(\llbracket \hat{w} \rrbracket_2), f_1(\llbracket \hat{w} \rrbracket_1))) \end{aligned}$$

  and from $\mathrm{sim}_1(f(\!|w|\!)) = \mathrm{sim}_1(w) = 1$

$$\begin{aligned} &= \llangle f(\!|w|\!)(\langle x_1 \rightarrow x_2 \rangle, (f_1(\llbracket \hat{w} \rrbracket_1), f_2(\llbracket \hat{w} \rrbracket_2))) \rrangle \\ &= \llangle f(\!|w\langle x_1 \rightarrow x_2 \rangle|\!) \rrangle = \llangle f(\!|\hat{w}|\!) \rrangle \end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 2$ and $\hat{w} = w\langle x_1 \rightarrowtriangle x_2\rangle$ (therefore, $\mathrm{sim}_1(\hat{w}) = 2$), we have

$$\begin{aligned}
f'\llparenthesis\langle\!\langle w\langle x_1 \rightarrowtriangle x_2\rangle\rangle\!\rangle\rrparenthesis &= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\langle x_2 \leftarrowtriangle x_1\rangle\rrparenthesis \\
&= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle x_2 \leftarrowtriangle x_1\rangle, f'(\langle\!\langle w\rangle\!\rangle\langle x_2 \leftarrowtriangle x_1\rangle)) \\
&= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle x_2 \leftarrowtriangle x_1\rangle, f'(\langle\!\langle \hat{w}\rangle\!\rangle)) \\
&\overset{(3)}{=} f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle x_2 \leftarrowtriangle x_1\rangle, (f_2(\llbracket\hat{w}\rrbracket_2), f_1(\llbracket\hat{w}\rrbracket_1))) \\
&\overset{\mathrm{IH}}{=} \langle\!\langle f\llparenthesis w\rrparenthesis\rangle\!\rangle(\langle x_2 \leftarrowtriangle x_1\rangle, (f_2(\llbracket\hat{w}\rrbracket_2), f_1(\llbracket\hat{w}\rrbracket_1)))
\end{aligned}$$

and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 2$

$$\begin{aligned}
&= \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \rightarrowtriangle x_2\rangle, (f_1(\llbracket\hat{w}\rrbracket_1), f_2(\llbracket\hat{w}\rrbracket_2)))\rangle\!\rangle \\
&= \langle\!\langle f\llparenthesis w\langle x_1 \rightarrowtriangle x_2\rangle\rrparenthesis\rangle\!\rangle = \langle\!\langle f\llparenthesis\hat{w}\rrparenthesis\rangle\!\rangle
\end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 2$ and $\hat{w} = w\langle x_1 \leftrightarrowtriangle x_2\rangle$ (therefore, $\mathrm{sim}_1(\hat{w}) = 2$), we have

$$\begin{aligned}
f'\llparenthesis\langle\!\langle w\langle x_1 \leftrightarrowtriangle x_2\rangle\rangle\!\rangle\rrparenthesis &= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\langle x_2 \leftrightarrowtriangle x_1\rangle\rrparenthesis \\
&= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle x_2 \leftrightarrowtriangle x_1\rangle, f'(\langle\!\langle w\rangle\!\rangle\langle x_2 \leftrightarrowtriangle x_1\rangle)) \\
&= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle x_2 \leftrightarrowtriangle x_1\rangle, f'(\langle\!\langle \hat{w}\rangle\!\rangle)) \\
&\overset{(3)}{=} f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle x_2 \leftrightarrowtriangle x_1\rangle, (f_2(\llbracket\hat{w}\rrbracket_2), f_1(\llbracket\hat{w}\rrbracket_1))) \\
&\overset{\mathrm{IH}}{=} \langle\!\langle f\llparenthesis w\rrparenthesis\rangle\!\rangle(\langle x_2 \leftrightarrowtriangle x_1\rangle, (f_2(\llbracket\hat{w}\rrbracket_2), f_1(\llbracket\hat{w}\rrbracket_1)))
\end{aligned}$$

and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 2$

$$\begin{aligned}
&= \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \leftrightarrowtriangle x_2\rangle, (f_1(\llbracket\hat{w}\rrbracket_1), f_2(\llbracket\hat{w}\rrbracket_2)))\rangle\!\rangle \\
&= \langle\!\langle f\llparenthesis w\langle x_1 \leftrightarrowtriangle x_2\rangle\rrparenthesis\rangle\!\rangle = \langle\!\langle f\llparenthesis\hat{w}\rrparenthesis\rangle\!\rangle
\end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 2$ and $\hat{w} = w\langle x_1 \leftarrowtriangle x_2\rangle$ (therefore, $\mathrm{sim}_1(\hat{w}) = 1$), we have

$$\begin{aligned}
f'\llparenthesis\langle\!\langle w\langle x_1 \leftarrowtriangle x_2\rangle\rangle\!\rangle\rrparenthesis &= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\langle \# \leftrightarrowtriangle \#\rangle\langle x_1 \leftarrowtriangle x_2\rangle\rrparenthesis \\
&= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle \# \leftrightarrowtriangle \#\rangle, (\#,\#))(\langle x_1 \leftarrowtriangle x_2\rangle, f'(\langle\!\langle w\rangle\!\rangle\langle \# \leftrightarrowtriangle \#\rangle\langle x_1 \leftarrowtriangle x_2\rangle)) \\
&= f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle \# \leftrightarrowtriangle \#\rangle, (\#,\#))(\langle x_1 \leftarrowtriangle x_2\rangle, f'(\langle\!\langle \hat{w}\rangle\!\rangle)) \\
&\overset{(3)}{=} f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle \# \leftrightarrowtriangle \#\rangle, (\#,\#))(\langle x_1 \leftarrowtriangle x_2\rangle, (f_1(\llbracket\hat{w}\rrbracket_1), f_2(\llbracket\hat{w}\rrbracket_2))) \\
&\overset{\mathrm{IH}}{=} \langle\!\langle f\llparenthesis w\rrparenthesis\rangle\!\rangle(\langle \# \leftrightarrowtriangle \#\rangle, (\#,\#))(\langle x_1 \leftarrowtriangle x_2\rangle, (f_1(\llbracket\hat{w}\rrbracket_1), f_2(\llbracket\hat{w}\rrbracket_2)))
\end{aligned}$$

and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 2$

$$\begin{aligned}
&= \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \leftarrowtriangle x_2\rangle, (f_1(\llbracket\hat{w}\rrbracket_1), f_2(\llbracket\hat{w}\rrbracket_2)))\rangle\!\rangle \\
&= \langle\!\langle f\llparenthesis w\langle x_1 \leftarrowtriangle x_2\rangle\rrparenthesis\rangle\!\rangle = \langle\!\langle f\llparenthesis\hat{w}\rrparenthesis\rangle\!\rangle
\end{aligned}$$

We next show *(ii)* $\to$ *(i)*. Let $f' = (f'_1, f'_2)$ be a distributed algorithm over the network model $\mathcal{N}'$ such that, for all $w \in \Sigma^\omega$, $f'\llparenthesis\langle\!\langle w\rangle\!\rangle\rrparenthesis \in \langle\!\langle L(\varphi)\rangle\!\rangle$. We can assume that, for all $p \in P$ and $u \in (\Sigma')^*$, we have $f'_p(u\langle \# \leftrightarrowtriangle \#\rangle) = \#$. Let $f = (f_1, f_2)$ be the distributed algorithm over $\mathcal{N}$ defined, for all $w \in \Sigma^+$ and $p \in \{1,2\}$, by

$$f_p(\llbracket w\rrbracket_p) := f'_{\mathrm{sim}_p(w)}(\llbracket\langle\!\langle w\rangle\!\rangle\rrbracket_{\mathrm{sim}_p(w)})\,. \tag{4}$$

This is well-defined due to Lemma 6(b). We have to show that, for all $w \in \Sigma^\omega$, $f\llparenthesis w\rrparenthesis \in L(\varphi)$. By injectivity of $\langle\!\langle\cdot\rangle\!\rangle$, this follows from the fact that, for all $w \in \Sigma^*$, we have

$$\langle\!\langle f\llparenthesis w\rrparenthesis \rangle\!\rangle = f'\llparenthesis \langle\!\langle w\rangle\!\rangle \rrparenthesis\,.$$

To show the latter, we again proceed by induction:

- From the definitions, we obtain $\langle\!\langle f\llparenthesis \varepsilon\rrparenthesis \rangle\!\rangle = f'\llparenthesis \langle\!\langle \varepsilon\rangle\!\rangle \rrparenthesis$.
- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 1$ and $\hat{w} = w\langle x_1 \leftarrowtriangle x_2\rangle$, we have

$$\begin{aligned}\langle\!\langle f\llparenthesis w\langle x_1 \leftarrowtriangle x_2\rangle\rrparenthesis \rangle\!\rangle &= \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \leftarrowtriangle x_2\rangle,(f_1(\llbracket \hat{w}\rrbracket_1), f_2(\llbracket \hat{w}\rrbracket_2)))\rangle\!\rangle\\ &\overset{(4)}{=} \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \leftarrowtriangle x_2\rangle,(f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2)))\rangle\!\rangle\end{aligned}$$

  and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 1$

$$\begin{aligned}&= \langle\!\langle f\llparenthesis w\rrparenthesis\rangle\!\rangle(\langle x_1 \leftarrowtriangle x_2\rangle,(f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2)))\\ &\overset{\text{IH}}{=} f'\llparenthesis \langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle x_1 \leftarrowtriangle x_2\rangle,(f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2)))\\ &= f'\llparenthesis \langle\!\langle w\langle x_1 \leftarrowtriangle x_2\rangle\rangle\!\rangle\rrparenthesis = f'\llparenthesis \langle\!\langle \hat{w}\rangle\!\rangle\rrparenthesis\end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 1$ and $\hat{w} = w\langle x_1 \leftrightarrowtriangle x_2\rangle$, we have

$$\begin{aligned}\langle\!\langle f\llparenthesis w\langle x_1 \leftrightarrowtriangle x_2\rangle\rrparenthesis \rangle\!\rangle &= \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \leftrightarrowtriangle x_2\rangle,(f_1(\llbracket \hat{w}\rrbracket_1), f_2(\llbracket \hat{w}\rrbracket_2)))\rangle\!\rangle\\ &\overset{(4)}{=} \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \leftrightarrowtriangle x_2\rangle,(f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2)))\rangle\!\rangle\end{aligned}$$

  and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 1$

$$\begin{aligned}&= \langle\!\langle f\llparenthesis w\rrparenthesis\rangle\!\rangle(\langle x_1 \leftrightarrowtriangle x_2\rangle,(f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2)))\\ &\overset{\text{IH}}{=} f'\llparenthesis \langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle x_1 \leftrightarrowtriangle x_2\rangle,(f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2)))\\ &= f'\llparenthesis \langle\!\langle w\langle x_1 \leftrightarrowtriangle x_2\rangle\rangle\!\rangle\rrparenthesis = f'\llparenthesis \langle\!\langle \hat{w}\rangle\!\rangle\rrparenthesis\end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 1$ and $\hat{w} = w\langle x_1 \rightarrowtriangle x_2\rangle$, we have

$$\begin{aligned}\langle\!\langle f\llparenthesis w\langle x_1 \rightarrowtriangle x_2\rangle\rrparenthesis \rangle\!\rangle &= \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \rightarrowtriangle x_2\rangle,(f_1(\llbracket \hat{w}\rrbracket_1), f_2(\llbracket \hat{w}\rrbracket_2)))\rangle\!\rangle\\ &\overset{(4)}{=} \langle\!\langle f\llparenthesis w\rrparenthesis(\langle x_1 \rightarrowtriangle x_2\rangle,(f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2), f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1)))\rangle\!\rangle\end{aligned}$$

  and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 1$

$$\begin{aligned}&= \langle\!\langle f\llparenthesis w\rrparenthesis\rangle\!\rangle(\langle \# \leftrightarrowtriangle \#\rangle,(\#,\#))(\langle x_2 \leftarrowtriangle x_1\rangle,(f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2)))\\ &\overset{\text{IH}}{=} f'\llparenthesis \langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle \# \leftrightarrowtriangle \#\rangle,(\#,\#))(\langle x_2 \leftarrowtriangle x_1\rangle,(f'_1(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle \hat{w}\rangle\!\rangle\rrbracket_2)))\\ &= f'\llparenthesis \langle\!\langle w\rangle\!\rangle\rrparenthesis(\langle \# \leftrightarrowtriangle \#\rangle,(\#,\#))(\langle x_2 \leftarrowtriangle x_1\rangle,\\ &\qquad (f'_1(\llbracket \langle\!\langle w\rangle\!\rangle\langle \# \leftrightarrowtriangle \#\rangle\langle x_2 \leftarrowtriangle x_1\rangle\rrbracket_1), f'_2(\llbracket \langle\!\langle w\rangle\!\rangle\langle \# \leftrightarrowtriangle \#\rangle\langle x_2 \leftarrowtriangle x_1\rangle\rrbracket_2)))\\ &= f'\llparenthesis \langle\!\langle w\rangle\!\rangle\langle \# \leftrightarrowtriangle \#\rangle\langle x_2 \leftarrowtriangle x_1\rangle\rrparenthesis\\ &= f'\llparenthesis \langle\!\langle w\langle x_1 \rightarrowtriangle x_2\rangle\rangle\!\rangle\rrparenthesis = f'\llparenthesis \langle\!\langle \hat{w}\rangle\!\rangle\rrparenthesis\end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 2$ and $\hat{w} = w\langle x_1 \rightarrowtriangle x_2\rangle$, we have

$$\begin{aligned}\llangle f\llparenthesis w\langle x_1 \rightarrowtriangle x_2\rangle\rrparenthesis\rrangle &= \llangle f\llparenthesis w\rrparenthesis(\langle x_1 \rightarrowtriangle x_2\rangle, (f_1(\llbracket\hat{w}\rrbracket_1), f_2(\llbracket\hat{w}\rrbracket_2)))\rrangle \\ &\overset{(4)}{=} \llangle f\llparenthesis w\rrparenthesis(\langle x_1 \rightarrowtriangle x_2\rangle, (f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2), f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1)))\rrangle\end{aligned}$$

and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 2$

$$\begin{aligned}&= \llangle f\llparenthesis w\rrparenthesis\rrangle(\langle x_2 \leftarrowtriangle x_1\rangle, (f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1), f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2))) \\ &\overset{\mathrm{IH}}{=} f'\llparenthesis\llangle w\rrangle\rrparenthesis(\langle x_2 \leftarrowtriangle x_1\rangle, (f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1), f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2))) \\ &= f'\llparenthesis\llangle w\rrangle\rrparenthesis(\langle x_2 \leftarrowtriangle x_1\rangle, (f_1'(\llbracket\llangle w\rrangle\langle x_2 \leftarrowtriangle x_1\rangle\rrbracket_1), f_2'(\llbracket\llangle w\rrangle\langle x_2 \leftarrowtriangle x_1\rangle\rrbracket_2))) \\ &= f'\llparenthesis\llangle w\rrangle\langle x_2 \leftarrowtriangle x_1\rangle\rrparenthesis \\ &= f'\llparenthesis\llangle w\langle x_1 \rightarrowtriangle x_2\rangle\rrangle\rrparenthesis = f'\llparenthesis\llangle\hat{w}\rrangle\rrparenthesis\end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 2$ and $\hat{w} = w\langle x_1 \leftrightarrowtriangle x_2\rangle$, we have

$$\begin{aligned}\llangle f\llparenthesis w\langle x_1 \leftrightarrowtriangle x_2\rangle\rrparenthesis\rrangle &= \llangle f\llparenthesis w\rrparenthesis(\langle x_1 \leftrightarrowtriangle x_2\rangle, (f_1(\llbracket\hat{w}\rrbracket_1), f_2(\llbracket\hat{w}\rrbracket_2)))\rrangle \\ &\overset{(4)}{=} \llangle f\llparenthesis w\rrparenthesis(\langle x_1 \leftrightarrowtriangle x_2\rangle, (f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2), f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1)))\rrangle\end{aligned}$$

and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 2$

$$\begin{aligned}&= \llangle f\llparenthesis w\rrparenthesis\rrangle(\langle x_2 \leftrightarrowtriangle x_1\rangle, (f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1), f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2))) \\ &\overset{\mathrm{IH}}{=} f'\llparenthesis\llangle w\rrangle\rrparenthesis(\langle x_2 \leftrightarrowtriangle x_1\rangle, (f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1), f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2))) \\ &= f'\llparenthesis\llangle w\rrangle\rrparenthesis(\langle x_2 \leftrightarrowtriangle x_1\rangle, (f_1'(\llbracket\llangle w\rrangle\langle x_2 \leftrightarrowtriangle x_1\rangle\rrbracket_1), f_2'(\llbracket\llangle w\rrangle\langle x_2 \leftrightarrowtriangle x_1\rangle\rrbracket_2))) \\ &= f'\llparenthesis\llangle w\rrangle\langle x_2 \leftrightarrowtriangle x_1\rangle\rrparenthesis \\ &= f'\llparenthesis\llangle w\langle x_1 \leftrightarrowtriangle x_2\rangle\rrangle\rrparenthesis = f'\llparenthesis\llangle\hat{w}\rrangle\rrparenthesis\end{aligned}$$

- For $w \in \Sigma^*$ with $\mathrm{sim}_1(w) = 2$ and $\hat{w} = w\langle x_1 \leftarrowtriangle x_2\rangle$, we have

$$\begin{aligned}\llangle f\llparenthesis w\langle x_1 \leftarrowtriangle x_2\rangle\rrparenthesis\rrangle &= \llangle f\llparenthesis w\rrparenthesis(\langle x_1 \leftarrowtriangle x_2\rangle, (f_1(\llbracket\hat{w}\rrbracket_1), f_2(\llbracket\hat{w}\rrbracket_2)))\rrangle \\ &\overset{(4)}{=} \llangle f\llparenthesis w\rrparenthesis(\langle x_1 \leftarrowtriangle x_2\rangle, (f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1), f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2)))\rrangle\end{aligned}$$

and from $\mathrm{sim}_1(f\llparenthesis w\rrparenthesis) = \mathrm{sim}_1(w) = 2$

$$\begin{aligned}&= \llangle f\llparenthesis w\rrparenthesis\rrangle(\langle \# \leftrightarrowtriangle \#\rangle, (\#,\#))(\langle x_1 \leftarrowtriangle x_2\rangle, (f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1), f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2))) \\ &\overset{\mathrm{IH}}{=} f'\llparenthesis\llangle w\rrangle\rrparenthesis(\langle \# \leftrightarrowtriangle \#\rangle, (\#,\#))(\langle x_1 \leftarrowtriangle x_2\rangle, (f_1'(\llbracket\llangle\hat{w}\rrangle\rrbracket_1), f_2'(\llbracket\llangle\hat{w}\rrangle\rrbracket_2))) \\ &= f'\llparenthesis\llangle w\rrangle\rrparenthesis(\langle \# \leftrightarrowtriangle \#\rangle, (\#,\#))(\langle x_1 \leftarrowtriangle x_2\rangle, \\ &\qquad (f_1'(\llbracket\llangle w\rrangle\langle \# \leftrightarrowtriangle \#\rangle\langle x_1 \leftarrowtriangle x_2\rangle\rrbracket_1), f_2'(\llbracket\llangle w\rrangle\langle \# \leftrightarrowtriangle \#\rangle\langle x_1 \leftarrowtriangle x_2\rangle\rrbracket_2))) \\ &= f'\llparenthesis\llangle w\rrangle\langle \# \leftrightarrowtriangle \#\rangle\langle x_1 \leftarrowtriangle x_2\rangle\rrparenthesis \\ &= f'\llparenthesis\llangle w\langle x_1 \leftarrowtriangle x_2\rangle\rrangle\rrparenthesis = f'\llparenthesis\llangle\hat{w}\rrangle\rrparenthesis\end{aligned}$$