

Distributed synthesis for well-connected architectures^{*}

Paul Gastin¹, Nathalie Sznajder¹, and Marc Zeitoun²

¹ LSV, ENS de Cachan & CNRS

61, Av. du Président Wilson, F-94235 Cachan Cedex, France
{Paul.Gastin,Nathalie.Sznajder}@lsv.ens-cachan.fr

² LaBRI, Universités de Bordeaux & CNRS

351, Cours de la Libération, F-33405 Talence Cedex, France
mz@labri.fr

Abstract. We study the synthesis problem for external linear or branching specifications and distributed, synchronous architectures with arbitrary delays on processes. *External* means that the specification only relates input and output variables. We introduce the subclass of uniformly well-connected (UWC) architectures for which there exists a routing allowing each output process to get the values of all inputs it is connected to, as soon as possible. We prove that the distributed synthesis problem is decidable on UWC architectures if and only if the output variables are totally ordered by their knowledge of input variables. We also show that if we extend this class by letting the routing depend on the output process, then the previous decidability result fails. Finally, we provide a natural restriction on specifications under which the whole class of UWC architectures is decidable.

1 Introduction

Synthesis is an essential problem in computer science considered by Church in [5]. It consists in translating a system property, relating input and output events, into a low-level model which has to compute the output from the input, so that the property is met. The property may be given in a high level specification language (such as monadic second order logic) while the low-level model can be a finite state machine. More generally, the problem can be parametrized by the specification language and the target model. It extends naturally to the controller synthesis problem, for which a system is also part of the input. The goal is to synthesize a controller such that the system, synchronized with the controller, meets a given specification.

Early works consider centralized open systems, i.e., systems interacting with an unpredictable environment, ([21,1]). The analogy between synthesis of reactive programs and two-player infinite games is classical: in [25], two-player games in an automata-theoretic setting are studied in the context of the synthesis problem. In these papers, it was assumed that the controller has complete

^{*} Partly supported by the French National Research Agency ANR-06-SetIn-DOTS

information. Algorithms for solving synthesis problem with *incomplete information* are presented in [10,11,3]. For *synchronous distributed* systems, where the different processes have only a partial view of the system, the situation is more involved: the problem is undecidable in general, and only a few cases have been identified as decidable (see [22,12] where the problem is studied for temporal logic specifications). For local specifications, constraining only variables local to processes, the general problem is still undecidable [14] (though doubly flanked pipelines become decidable). The game-theoretic framework is also useful for solving the distributed case (see e.g. [4]). Realizability of specifications in the *logic of knowledge and linear time* has been studied in [26] for systems with a single agent, and in a distributed setting in [27].

For *asynchronous* communication, synthesis has first been studied in [20] for single-process implementations and linear-time specifications. In [15], the existence of a specific class of controllers in the distributed setting is proved decidable for trace-closed specifications. This result has been strengthened in [16], where the restrictions on the communication patterns of the controllers have been reduced. Another subclass of decidable systems, incomparable with the latter, has been identified in [8], in which causal memories for the controllers were considered. The synthesis of asynchronous distributed systems in the general case of μ -calculus specifications was studied in [7]. Also, the theory of asynchronous automata has been applied in [24] to solve the synthesis problem of *closed* distributed systems. Finally, unifying several formalisms, [17] proposed the framework of distributed games, which is a specialized version of multi-player games, to reason about distributed synthesis, both in the synchronous and asynchronous semantics.

In this paper, we address the synthesis problem for distributed open synchronous systems and temporal logic specifications. This specific question has been first studied in [22], where synthesis has been proved undecidable for LTL specifications and arbitrary architectures, whereas if we restrict to pipeline architectures, synthesis becomes non elementarily decidable for LTL specifications. The lower bound follows from a former result on multiplayer games [19].

The pipeline architecture has been shown decidable for CTL* *full* specifications [12], that is, specifications allowed to constrain all variables of the system. In this case, where decidability of the distributed synthesis is obtained, full specifications strengthen the result.

A decision criterion, established in [6] for full specifications, implies that the architecture of Figure 1 is undecidable. The reason is that full specifications make it possible to enforce a constant value on variable t , breaking the link between processes p_0 and p_1 . For the undecidability part of the criterion, allowing specifications on all variables weakens the result by yielding easy reductions to the basic undecidable architecture of Pnueli and Rosner [22], for instance by breaking communication links at will.

In the seminal paper [22], specifications were assumed to be *external*, or *input-output*: only variables communicating with the environment could be constrained. The way processes of the system communicate was only restricted by

the communication architecture, not by the specification. This is very natural from a practical point of view: when writing a specification, we are only concerned by the input/output behavior of the system and we should leave to the implementation all freedom on its internal behavior. For that reason, solving the problem for external specifications is more relevant and useful - albeit more difficult - than a decidability criterion for arbitrary specifications. We will show that the architecture of Figure 1 is decidable for *external* specifications, that is, if we do not constrain the internal variable t .

Contributions. We consider the synthesis problem for synchronous semantics, where each process is assigned a nonnegative delay. The delays can be used to model latency in communications, or slow processes. This model has the same expressive power as the one where delays sit on communication channels, and it subsumes both the 0-delay and the 1-delay classical semantics [22,12].

To rule out unnatural properties yielding undecidability, the specifications we consider are external, coming back to the original framework of [22,5]. We first determine a sufficient condition for undecidability with external specifications, that generalizes the undecidability result of [22]. We next introduce *uniformly well-connected* (UWC) architectures. Informally, an architecture is UWC if there exists a routing allowing each output process to get, as soon as possible, the values of all inputs it is connected to. Using tree automata, we prove that for such architectures and external specifications, the sufficient condition for undecidability becomes a criterion. As already pointed out, some architectures undecidable for general specifications become decidable for external specifications. We also propose a natural restriction on specifications, called *robust*, for which all UWC architectures become decidable.

Finally, we introduce the larger class of *well-connected architectures*, in which the routing of input variables to an output process may depend on that process. We show that our criterion is not necessary anymore for this larger class. The undecidability proof highlights the surprising fact that in Figure 1, blanking out a *single* information bit in the transmission of x_0 to p_1 through t suffices to yield undecidability. This is a step forward in understanding decidability limits for distributed synthesis. Whether the *robust* external specifications are always decidable for this class remains open.

An extended abstract of this work appeared in [9].

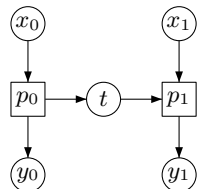


Fig. 1. Architecture decidable for external/undecidable for full specifications.

2 Preliminaries

Trees and tree automata. Given two finite sets X and Y , a Y -labeled (full) X -tree is a (total) function $t : X^* \rightarrow Y$ where elements of X are called directions, and elements of Y are called labels. A word $\sigma \in X^*$ defines a node of t and $t(\sigma)$ is its label. The empty word ε is the root of the tree. A word $\sigma \in X^\omega$ is a branch. In the following, a tree $t : X^* \rightarrow Y$ will be called an (X, Y) -tree.

A non-deterministic tree automaton (NDTA) $\mathfrak{A} = (X, Y, Q, q_0, \delta, \alpha)$ runs on (X, Y) -trees. It consists of a finite set of states Q , an initial state q_0 , a transition function $\delta : Q \times Y \rightarrow \mathcal{P}(Q^X)$ and an acceptance condition $\alpha \subseteq Q^\omega$. A *run* ρ of such an automaton over an (X, Y) -tree t is an (X, Q) -tree ρ such that for all $\sigma \in X^*$, $(\rho(\sigma \cdot x))_{x \in X} \in \delta(\rho(\sigma), t(\sigma))$. A run tree is accepting if all its branches $s_1 s_2 \cdots \in X^\omega$ are such that $\rho(\varepsilon)\rho(s_1)\rho(s_1 s_2) \cdots \in \alpha$. The specific acceptance condition chosen among the classical ones is not important in this paper.

Architectures. An *architecture* $\mathcal{A} = (V \uplus P, E, (S^v)_{v \in V}, s_0, (d_p)_{p \in P})$ is a finite directed acyclic bipartite graph, where $V \uplus P$ is the set of vertices, and $E \subseteq (V \times P) \cup (P \times V)$ is the set of edges, such that $|E^{-1}(v)| \leq 1$ for all $v \in V$. Elements of P will be called *processes* and elements of V *variables*. Intuitively, an edge $(v, p) \in V \times P$ means that process p can read variable v , and an edge $(p, v) \in P \times V$ means that p can write on v . Thus, $|E^{-1}(v)| \leq 1$ means that a variable v is written by at most one process. Input and output variables are defined, respectively, by

$$\begin{aligned} V_I &= \{v \in V \mid E^{-1}(v) = \emptyset\}, \\ V_O &= \{v \in V \mid E(v) = \emptyset\}. \end{aligned}$$

Variables in $V \setminus (V_I \cup V_O)$ will be called *internal*. We assume that no process is minimal or maximal in the graph: for $p \in P$, we have $E(p) \neq \emptyset$ and $E^{-1}(p) \neq \emptyset$.

Each variable v ranges over a finite domain S^v , given with the architecture. For $U \subseteq V$, S^U will denote $\prod_{v \in U} S^v$. A *configuration* of the architecture is given by a tuple $s = (s^v)_{v \in V} \in S^V$ describing the value of all variables. For $U \subseteq V$, we denote by $s^U = (s^v)_{v \in U}$ the projection of the configuration s to the subset of variables U . The initial configuration is $s_0 = (s_0^v)_{v \in V} \in S^V$.

We will assume that $|S^v| \geq 2$ for all $v \in V$, because a variable v for which $|S^v| = 1$ always has the same value and may be ignored. It will be convenient in some proofs to assume that $\{0, 1\} \subseteq S^v$ and that $s_0^v = 0$ for all $v \in V$.

Each process $p \in P$ is associated with a delay $d_p \in \mathbb{N}$ that corresponds to the time interval between the moment the process reads the variables $v \in E^{-1}(p)$ and the moment it will be able to write on its own output variables. Note that delay 0 is allowed. In the following, for $v \in V$, we will often write d_v for d_p where $E^{-1}(v) = \{p\}$.

An example of an architecture is given in Figure 2, where processes are represented by boxes and variables by circles.

Runs. A *run* of an architecture is an infinite sequence of configurations, *i.e.*, an infinite word over the alphabet S^V , starting with the initial configuration

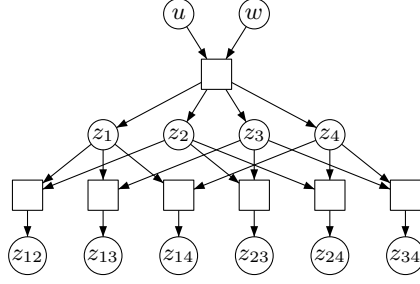


Fig. 2. An architecture

$s_0 \in S^V$ given by the architecture. If $\sigma = s_0 s_1 s_2 \dots \in (S^V)^\omega$ is a run, then its projection on $U \subseteq V$ is $\sigma^U = s_0^U s_1^U s_2^U \dots$. Also, we denote by $\sigma[i, j]$ the factor $s_i \dots s_j$ and by $\sigma[i]$ the prefix of length i of σ (by convention, $\sigma[i] = \varepsilon$ if $i \leq 0$). A *run tree* is a *full tree* $t : (S^{V_1})^* \rightarrow S^V$, where $t(\varepsilon) = s_0$ and for $\rho \in (S^{V_1})^*$, $r \in S^{V_1}$, we have $t(\rho \cdot r)^{V_1} = r$. The projection of t on $U \subseteq V$ is the tree $t^U : (S^{V_1})^* \rightarrow S^U$ defined by $t^U(\rho) = t(\rho)^U$.

Specifications. Specifications over a set $U \subseteq V$ of variables can be given, for instance, by a μ -calculus, CTL*, CTL, or LTL formula, using atomic propositions of the form $(v = a)$ with $v \in U$ and $a \in S^v$. We then say that the formula is in $\mathcal{L}(U)$ where \mathcal{L} is the logic used. A specification is *external* if $U \subseteq V_1 \cup V_0$. The validity of an external formula on a run tree t (or simply a run) only depends on its projection $t^{V_1 \cup V_0}$ onto $V_1 \cup V_0$.

Programs, strategies. We consider a discrete time, synchronous semantics. Informally, at step $i = 1, 2, \dots$, the environment provides new values for input variables. Then, each process p reading values written by its predecessors or by the environment at step $i - d_p$, computes values for the variables in $E(p)$, and writes them. Let $v \in V \setminus V_1$ and let $R(v) = E^{-2}(v)$ be the set of variables read by the process writing to v . Intuitively, from a word $\sigma^{R(v)}$ in $(S^{R(v)})^+$ representing the projection on $R(v)$ of some run prefix, a program (or a strategy) advices a value to write on variable v . But, since the process may have a certain delay d_v , the output of the strategy must not depend on the last d_v values of $\sigma^{R(v)}$.

Formally, a *program* (or *local strategy*) for variable v is a mapping $f^v : (S^{R(v)})^+ \rightarrow S^v$ compatible with the delay d_v , i.e., such that for all $\sigma, \sigma' \in (S^{R(v)})^i$, if $\sigma[i - d_v] = \sigma'[i - d_v]$, then $f^v(\sigma) = f^v(\sigma')$. This condition – called *delay-compatibility* or simply *d-compatibility* – ensures that the delay d_v is respected when computing the next value of variable v . A *distributed program* (or *distributed strategy*) is a tuple $F = (f^v)_{v \in V \setminus V_1}$ of local strategies. A run $\sigma \in (S^V)^\omega$ is an *F-run* (or *F-compatible*) if for all $v \in V \setminus V_1$, $s_i^v = f^v(\sigma^{R(v)}[i])$. Given an input sequence $\rho \in (S^{V_1})^\omega$, there is a unique run $\sigma \in (S^V)^\omega$ which is *F-compatible* and such that $\sigma^{V_1} = \rho$.

The *F-run tree* is the run tree $t : (S^{V_1})^* \rightarrow S^V$ such that each branch is labeled by a word $s_0 s_1 s_2 \dots \in (S^V)^\omega$ which is an *F-run*. Note that, in an *F-run*

$\sigma \in (S^V)^\omega$, the prefix $\sigma[i]$ only depends on the prefix $\sigma^{V_1}[i]$. This shows that the F -run tree is unique.

Distributed synthesis problem. Let \mathcal{L} be a specification language. The distributed synthesis problem for an architecture \mathcal{A} is the following: given a formula $\varphi \in \mathcal{L}$, decide whether there exists a distributed program F such that every F -run (or the F -run tree) satisfies φ . We will then say that F is a distributed implementation for the specification φ . If for some architecture the synthesis problem is undecidable, we say that the architecture itself is *undecidable* (for the specification language \mathcal{L}).

Memoryless strategies. The strategy f^v is *memoryless* if it does not depend on the past, that is, if there exists $g : S^{R(v)} \rightarrow S^v$ such that $f^v(s_1 \cdots s_i \cdots s_{i+d_v}) = g(s_i)$ for $s_1 \cdots s_{i+d_v} \in (S^{R(v)})^+$. In case $d_v = 0$, this corresponds to the usual definition of a memoryless strategy.

Summaries. For a variable $v \in V$, we let $\text{View}(v) = (E^{-2})^*(v) \cap V_1$ be the set of *input* variables v might depend on. Observe that if σ is an F -run, then for all $v \in V \setminus V_1$, for all $i \geq 0$, s_i^v only depends on $\sigma^{\text{View}(v)}[i]$. This allows us to define the summary $\hat{f}^v : (S^{\text{View}(v)})^+ \rightarrow S^v$ such that $\hat{f}^v(\sigma^{\text{View}(v)}[i]) = s_i^v$, corresponding to the composition of all local strategies used to compute v .

Remark 1. The compatibility of the strategies $F = (f^v)_{v \in V \setminus V_1}$ with the delays extends to the summaries $\hat{F} = (\hat{f}^v)_{v \in V \setminus V_1}$. Formally, a map $h : (S^{\text{View}(v)})^+ \rightarrow S^v$ is *d-compatible* (or compatible with the delays $(d_v)_{v \in V \setminus V_1}$) if for all $\rho \in (S^{\text{View}(v)})^i$, $h(\rho)$ only depends on the prefixes $(\rho^u[i - d(u, v)])_{u \in \text{View}(v)}$ where $d(u, v)$ is the smallest cumulative delay of transmission from u to v and is defined by $d(u, u) = 0$ and

$$d(u, v) = d_v + \min\{d(u, w) \mid w \in R(v) \text{ and there is a path from } u \text{ to } w\},$$

with $\min \emptyset = +\infty$ so that $d(u, v) = +\infty$ if there is no path from u to v .

3 Architectures with incomparable information

In this section, we state a sufficient condition for undecidability, which is an easy generalization of the undecidable architecture presented in [22].

Definition 2. *An architecture has incomparable information if there exist variables $x, y \in V_O$ such that $\text{View}(x) \setminus \text{View}(y) \neq \emptyset$ and $\text{View}(y) \setminus \text{View}(x) \neq \emptyset$. Otherwise the architecture has linearly preordered information.*

For instance, the architectures of Figures 1, 2, 4 and 5 have linearly preordered information, while the architecture \mathcal{A}' of Figure 3 has incomparable information. The following proposition extends the undecidability result of [22,6].

Proposition 3. *Architectures with incomparable information are undecidable for LTL or CTL external specifications.*

In [22], the architecture \mathcal{A}' shown in Figure 3 is proved undecidable, both for LTL and CTL specifications. We will reduce the synthesis problem of \mathcal{A}' to the synthesis problem of an architecture with incomparable information. This reduction is rather natural but not completely straightforward, for instance the specification needs to be changed in the reduction. For the sake of completeness, we give a precise proof of the reduction in the rest of this section.

Let $\mathcal{A} = (P \uplus V, E, (S^v)_{v \in V}, s_0, (d_p)_{p \in P})$ be an architecture with incomparable information. Recall that we can assume without loss of generality that $s_0^v = 0$ for all $v \in V$. By definition, there are two (disjoint) sets of variables $\{x_0, \dots, x_n\}$ and $\{y_0, \dots, y_m\}$ with $x_0, y_0 \in V_I$, $x_n, y_m \in V_O$, $x_0 \notin \text{View}(y_m)$, $y_0 \notin \text{View}(x_n)$, $x_0 E^2 x_1 E^2 \dots E^2 x_n$, and $y_0 E^2 y_1 E^2 \dots E^2 y_m$.

Let $\mathcal{A}' = (P' \cup V', E', (S'^v)_{v \in V'}, s'_0, (d'_p)_{p \in P'})$ be the architecture of Figure 3 with $V'_I = \{x_0, y_0\}$, $V'_O = \{x_n, y_m\}$, $S'^{x_0} = S'^{y_0} = \{0, 1\}$, $S'^{x_n} = S^{x_n}$, $S'^{y_m} = S^{y_m}$, $s'_0^v = 0$ for all $v \in V'$, $d'_{x_n} = d'_p = d_{x_1} + \dots + d_{x_n}$ and $d'_{y_m} = d'_q = d_{y_1} + \dots + d_{y_m}$.

The architecture \mathcal{A}' is undecidable for LTL or CTL specifications (it suffices to adapt the proofs of [22,6] taking into account different delays on the processes). We reduce the distributed synthesis problem for \mathcal{A}' to the same problem for \mathcal{A} . We first consider CTL specifications.

Note that we do need to modify the specification when reducing the distributed synthesis problem from \mathcal{A}' to \mathcal{A} . Indeed, observe that the specification

$$\psi = \text{EG}((x_0 = 0) \wedge (x_n = 0)) \wedge \text{EG}((x_0 = 0) \wedge (x_n = 1))$$

is not implementable over \mathcal{A}' while it is implementable over \mathcal{A} – provided $\text{View}(x_n) \setminus \{x_0\} \neq \emptyset$ and assuming no delays.

To define an implementation F' over \mathcal{A}' given an implementation F over \mathcal{A} , we simulate the behavior of F when all the variables in $V_I \setminus V'_I$ are constantly set to 0. This will be enforced when defining the reduction of the specification from \mathcal{A}' to \mathcal{A} , using the formula $\chi = (x_0 \in \{0, 1\}) \wedge (y_0 \in \{0, 1\}) \wedge \bigwedge_{v \in V_I \setminus V'_I} (v = 0)$. We define a reduction that maps a formula ψ of $\text{CTL}(V')$ into a formula $\bar{\psi}$ in

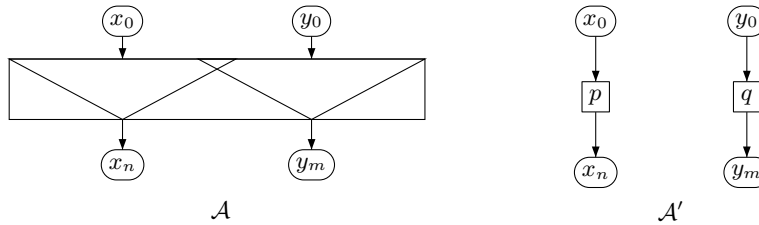


Fig. 3. Architectures \mathcal{A} and \mathcal{A}'

CTL($V_0 \cup V_1$) by

$$\begin{aligned}
\overline{(x = s)} &= (x = s) \\
\overline{\neg\psi} &= \neg\overline{\psi} \\
\overline{\varphi \vee \psi} &= \overline{\varphi} \vee \overline{\psi} \\
\overline{\text{EX}\psi} &= \text{EX}(\chi \wedge \overline{\psi}) \\
\overline{\text{E}\varphi \text{U}\psi} &= \text{E}(\chi \wedge \overline{\varphi}) \text{U} (\chi \wedge \overline{\psi}) \\
\text{and } \overline{\text{EG}\psi} &= \text{EG}(\chi \wedge \overline{\psi}).
\end{aligned}$$

For $r \in S^{V_1'}$, we define $\bar{r} \in S^{V_1}$ by $\bar{r}^{V_1'} = r$ and $\bar{r}^v = 0$ for all $v \in V_1 \setminus V_1'$, and we extend this definition to words. For a run tree $t : (S^{V_1})^* \rightarrow S^V$, we define $\bar{t} : (S^{V_1'})^* \rightarrow S^{V'}$ by $\bar{t}(\rho) = t(\bar{\rho})^{V'}$ for $\rho \in (S^{V_1'})^*$. We need the following lemma:

Lemma 4. *For every formula $\psi \in \text{CTL}(V')$, every tree $t : (S^{V_1})^* \rightarrow S^V$, and every $\rho \in (S^{V_1'})^*$ we have $t, \bar{\rho} \models \overline{\psi}$ if and only if $\bar{t}, \rho \models \psi$.*

Proof. By induction on ψ . Let $t : (S^{V_1})^* \rightarrow S^V$ and $\rho \in (S^{V_1'})^*$.

- Let $x \in V'$ and $s \in S^x$. Then, $\bar{t}, \rho \models (x = s)$ if and only if $\bar{t}(\rho)^x = s$ if and only if, by definition, $t(\bar{\rho})^x = s$, if and only if $t, \bar{\rho} \models \overline{(x = s)}$.
- The cases $\neg\varphi$ and $\varphi \vee \psi$ are trivial.
- $\bar{t}, \rho \models \text{EX}\varphi$ if and only if $\bar{t}, \rho \cdot r \models \varphi$ for some $r \in S^{V_1'}$, if and only if, by induction hypothesis, $t, \bar{\rho} \cdot \bar{r} \models \overline{\varphi}$ for some $r \in S^{V_1'}$, if and only if $t, \bar{\rho} \cdot s \models \chi \wedge \overline{\varphi}$ for some $s \in S^{V_1}$, if and only if $t, \bar{\rho} \models \text{EX}(\chi \wedge \overline{\varphi}) = \overline{\text{EX}\varphi}$.
- $\bar{t}, \rho \models \text{E}\varphi \text{U}\psi$ if and only if there exists $r_1 \cdots r_n \in (S^{V_1'})^*$ such that $\bar{t}, \rho \cdot r_1 \cdots r_n \models \psi$ and $\bar{t}, \rho \cdot r_1 \cdots r_i \models \varphi$ for all $0 \leq i < n$, if and only if, by induction hypothesis, there exists $r_1 \cdots r_n \in (S^{V_1'})^*$ such that $t, \bar{\rho} \cdot \bar{r}_1 \cdots \bar{r}_n \models \overline{\psi}$ and $t, \bar{\rho} \cdot \bar{r}_1 \cdots \bar{r}_i \models \overline{\varphi}$ for all $0 \leq i < n$, if and only if there exists $s_1 \cdots s_n \in (S^{V_1})^*$ such that $t, \bar{\rho} \cdot s_1 \cdots s_n \models \chi \wedge \overline{\psi}$ and $t, \bar{\rho} \cdot s_1 \cdots s_i \models \chi \wedge \overline{\varphi}$ for all $0 \leq i < n$, if and only if $t, \bar{\rho} \models \overline{\text{E}\varphi \text{U}\psi}$.
- $\bar{t}, \rho \models \text{EG}\varphi$ if and only if there exists $r_1 r_2 \cdots \in (S^{V_1'})^\omega$ such that $\bar{t}, \rho \cdot r_1 \cdots r_n \models \varphi$ for all $n \geq 0$, if and only if by induction hypothesis, there exists $r_1 r_2 \cdots \in (S^{V_1'})^\omega$ such that $t, \bar{\rho} \cdot \bar{r}_1 \cdots \bar{r}_n \models \overline{\varphi}$ for all $n \geq 0$, if and only if there exists $s_1 s_2 \cdots \in (S^{V_1})^\omega$, such that $t, \bar{\rho} \cdot s_1 \cdots s_n \models \chi \wedge \overline{\varphi}$ for all $n \geq 0$, if and only if $t, \bar{\rho} \models \overline{\text{EG}\varphi}$. \square

Lemma 5. *If there is a distributed program F' over \mathcal{A}' that satisfies ψ , then there is a distributed program F over \mathcal{A} that satisfies $\overline{\psi}$.*

Proof. Let $F' = (f^{!x_n}, f^{!y_m})$ be a distributed implementation for ψ over \mathcal{A}' . The variables x_1, \dots, x_{n-1} will be used to transmit the value of x_0 so that the strategy f^{x_n} can simulate $f^{!x_n}$. Formally, for $0 < k < n$, $\sigma \in (S^{R(x_k)})^+$, let

$$f^{x_k}(\sigma) = \begin{cases} s^{x_{k-1}} & \text{if } \sigma = \sigma_1 s \sigma_2 \text{ with } |\sigma_2| = d_{x_k} \text{ and } s^{x_{k-1}} \in \{0, 1\}, \\ 0 & \text{otherwise.} \end{cases}$$

The last variable x_n simulates f^{x_n} : for $\sigma \in (S^{R(x_n)})^+$, we let

$$f^{x_n}(\sigma) = \begin{cases} f^{x_n}(0^{|\sigma|}) & \text{if } |\sigma| \leq d'_{x_n}, \\ f^{x_n}(\sigma_2^{x_n-1} 0^{d'_{x_n}}) & \text{if } \sigma = \sigma_1 \sigma_2 \sigma_3 \text{ with } |\sigma_1| = d'_{x_n} - d_{x_n}, |\sigma_3| = d_{x_n}, \\ & \text{and } \sigma_2^{x_n-1} \in \{0, 1\}^+, \\ 0 & \text{otherwise.} \end{cases}$$

We first check that f^{x_n} is d -compatible. Let $\sigma, \sigma' \in (S^{R(x_n)})^i$ such that $\sigma[i - d_{x_n}] = \sigma'[i - d_{x_n}]$. If $i \leq d'_{x_n}$, then $f^{x_n}(\sigma) = f^{x_n}(\sigma') = f^{x_n}(0^i)$ by the first case of the definition of f^{x_n} . Otherwise, one can write $\sigma = \sigma_1 \sigma_2 \sigma_3$ and $\sigma' = \sigma_1 \sigma_2 \sigma'_3$, with $|\sigma_1| = d'_{x_n} - d_{x_n}$ and $|\sigma_3| = |\sigma'_3| = d_{x_n}$. If $\sigma_2^{x_n-1} \in \{0, 1\}^+$, then $f^{x_n}(\sigma) = f^{x_n}(\sigma') = f^{x_n}(\sigma_2^{x_n-1} 0^{d'_{x_n}})$, and otherwise, $f^{x_n}(\sigma) = f^{x_n}(\sigma') = 0$.

We define similarly f^{y_k} , $0 < k \leq m$. For every other variable v , we set $f^v = 0$. Let F be the resulting distributed strategy. We show that F is a distributed implementation for $\bar{\psi}$ over \mathcal{A} .

Let $t' : (S^{V_1'})^* \rightarrow S^{V'}$ be the F' -run tree over \mathcal{A}' and $t : (S^{V_1})^* \rightarrow S^V$ the F -run tree over \mathcal{A} . We show that $\bar{t} = t'$. Let $\rho = r_1 \cdots r_i \in (S^{V_1'})^+$ and let $\sigma = (t(\bar{r}_1)t(\bar{r}_1\bar{r}_2) \cdots t(\bar{\rho}))^{R(x_n)}$. We have $\bar{t}(\rho)^{V_1'} = t(\bar{\rho})^{V_1'} = r_i^{V_1'} = t'(\rho)^{V_1'}$.

If $|\rho| \leq d'_{x_n}$ then $\bar{t}(\rho)^{x_n} = t(\bar{\rho})^{x_n} = f^{x_n}(\sigma) = f^{x_n}(0^{|\sigma|}) = f^{x_n}(\rho^{x_0}) = t'(\rho)^{x_n}$, where the fourth equality follows from the d' -compatibility of f^{x_n} . Similarly, $\bar{t}(\rho)^{y_m} = t'(\rho)^{y_m}$ and we have shown $\bar{t}(\rho)^{V_{O'}} = t'(\rho)^{V_{O'}}$ when $|\rho| \leq d'_{x_n}$.

Finally, if $|\rho| > d'_{x_n}$, then write $\rho = \rho_2 \rho_3 \rho_4$ with $|\rho_3| = d_{x_n}$ and $|\rho_4| = d'_{x_n} - d_{x_n}$. Let $\delta_k = d_{x_1} + \cdots + d_{x_k}$. By induction on $0 < k < n$, and using the definition of f^{x_k} , we obtain $t(\bar{\rho}[i])^{x_k} = r_{i-\delta_k}^{x_0}$ with the convention that $r_j^{x_0} = 0$ if $j \leq 0$. Therefore, $\sigma^{x_n-1} = 0^{d'_{x_n} - d_{x_n}} \rho_2^{x_0} \rho_3^{x_0}$ and $\rho_2^{x_0} \in \{0, 1\}^+$. We obtain $\bar{t}(\rho)^{x_n} = t(\bar{\rho})^{x_n} = f^{x_n}(\sigma) = f^{x_n}(\rho_2^{x_0} 0^{d'_{x_n}}) = f^{x_n}(\rho^{x_0}) = t'(\rho)^{x_n}$, where the fourth equality follows from the d' -compatibility of f^{x_n} . Similarly, $\bar{t}(\rho)^{y_m} = t'(\rho)^{y_m}$ and we have shown $\bar{t}(\rho)^{V_{O'}} = t'(\rho)^{V_{O'}}$ when $|\rho| > d'_{x_n}$.

We have shown $\bar{t} = t'$ and since F' is a distributed implementation of ψ , we have $t', \varepsilon \models \psi$, i.e., $\bar{t}, \varepsilon \models \psi$. Now, Lemma 4 implies that $t, \varepsilon \models \bar{\psi}$. Hence F is a distributed implementation of $\bar{\psi}$. \square

Lemma 6. *If there is a distributed program F over \mathcal{A} that satisfies $\bar{\psi}$, then there is a distributed program F' over \mathcal{A}' that satisfies ψ .*

Proof. Suppose $F = (f^v)_{v \in V \setminus V_1}$ is a distributed implementation of $\bar{\psi}$ over \mathcal{A} . We need to define the strategies $f^{x_n} : (S^{x_0})^+ \rightarrow S^{x_n}$ and $f^{y_m} : (S^{y_0})^+ \rightarrow S^{y_m}$ of the variables in \mathcal{A}' . Intuitively, the processes of \mathcal{A}' will behave like the processes of \mathcal{A} writing respectively on x_n and y_m when the values of input variables other than x_0 and y_0 are always 0.

For $\rho \in (S^{V_1'})^+$ we set $f^{x_n}(\rho^{x_0}) = \hat{f}^{x_n}(\bar{\rho}^{\text{View}(x_n)})$. Observe that, due to incomparable information, \hat{f}^{x_n} does not depend on $\bar{\rho}^{y_0}$. Hence f^{x_n} only depends on ρ^{x_0} and is a correct strategy for variable x_n in the architecture \mathcal{A}' . Moreover, \hat{f}^{x_n} is d -compatible and so f^{x_n} is d -compatible. We define f^{y_m} similarly. Let t be the F -run tree. Since $t, \varepsilon \models \bar{\psi}$, Lemma 4 implies that $\bar{t}, \varepsilon \models \psi$. We have

$t'(\rho)^{x_n} = f^{t'x_n}(\rho^{x_0}) = \hat{f}^{x_n}(\bar{\rho}^{\text{View}(x_n)}) = t(\bar{\rho})^{x_n} = \bar{t}(\rho)^{x_n}$ and similarly, $t'(\rho)^{y_m} = \bar{t}(\rho)^{y_m}$. Hence $t' = \bar{t}$ and F' is a distributed implementation of ψ on \mathcal{A}' . \square

For LTL specifications, we use

$$\bar{\psi} = (\mathbf{G} \xi) \rightarrow \psi$$

where the formula ξ is defined by $\xi = (x_0 \in \{0, 1\}) \wedge (y_0 \in \{0, 1\})$.

The same constructions as the ones described in the proofs of Lemma 5 and Lemma 6 yield the reduction. Indeed, suppose F' given and F defined as in the proof of Lemma 5. Let $\rho \in (S^{V_I})^\omega$ and $\sigma \in (S^V)^\omega$ the induced F -run. If $\rho^{x_0} \notin \{0, 1\}^\omega$ or $\rho^{y_0} \notin \{0, 1\}^\omega$ then $\sigma, \varepsilon \models \bar{\psi}$. Otherwise, let $\sigma' = F'(\rho^{\{x_0, y_0\}})$. As in the proof of Lemma 5 we show $\sigma' = \sigma^{V'}$. We have $\sigma', \varepsilon \models \psi$ and since $\psi \in \text{LTL}(V')$ we deduce $\sigma, \varepsilon \models \psi$. Then $\sigma, \varepsilon \models \bar{\psi}$ and F is a distributed implementation of $\bar{\psi}$ over \mathcal{A} .

Conversely, given F define F' as in the proof of Lemma 6. Let $\rho \in (S^{V_I'})^\omega$, $\sigma' = F'(\rho)$ and $\sigma = F(\bar{\rho})$. Then, $\sigma, \varepsilon \models \mathbf{G} \xi$ and since F is a distributed implementation of $\bar{\psi}$ we get $\sigma, \varepsilon \models \psi$. We can show as in the proof of Lemma 6 that $\sigma' = \sigma^{V'}$ and since $\psi \in \text{LTL}(V')$ we deduce $\sigma', \varepsilon \models \psi$. Hence, F' is a distributed implementation of ψ over \mathcal{A}' .

We have defined a reduction of the distributed synthesis problem over architecture \mathcal{A}' to the distributed synthesis problem over an architecture with incomparable information for an LTL or CTL specification. The problem being undecidable for both logics over \mathcal{A}' we obtain undecidability for architectures with incomparable information.

4 Uniformly well-connected architectures

This section introduces the new class of uniformly well-connected (UWC) architectures and provides a decidability criterion for the synthesis problem on this class. It also introduces the notion of *robust* specifications and shows that UWC architectures are always decidable for external and robust specifications.

4.1 Definition

A *routing* for an architecture $\mathcal{A} = (V \uplus P, E, (S^v)_{v \in V}, s_0, (d_p)_{p \in P})$ is a family $\Phi = (f^v)_{v \in V \setminus (V_I \cup V_O)}$ of *memoryless* local strategies. Observe that a routing does not include local strategies for output variables. Informally, we say that an architecture is uniformly well connected if there exists a routing Φ that makes it possible to transmit with a minimal delay to every process p writing to an output variable v , the values of the variables in $\text{View}(v)$.

Definition 7. *An architecture \mathcal{A} is uniformly well-connected (UWC) if there exist a routing Φ and, for every $v \in V_O$ and $u \in \text{View}(v)$, a decoding function $g^{u,v} : (S^{R(v)})^+ \rightarrow S^u$ that can reconstruct the value of u , i.e., such that for any Φ -compatible sequence $\sigma = s_0 s_1 s_2 \dots \in (S^{V \setminus V_O})^+$, we have for $i \geq 0$*

$$s_i^u = g^{u,v}(\sigma^{R(v)}[i + d(u, v) - d_v]) \quad (1)$$

In case there is no delay, the uniform well-connectedness refines the notion of adequate connectivity introduced by Pnueli and Rosner in [22], as we no longer require each output variable to be communicated the value of *all* input variables, but only those in its view. In fact, this gives us strategies for internal variables, that are simply to route the input to the processes writing on output variables.

Observe that, whereas the routing functions are memoryless, memory is required for the decoding functions. Indeed, consider the architecture of Figure 4. The delays are written next to the processes, and all variables range over the domain $\{0, 1\}$. Observe first that this architecture is UWC: process p writes to t the xor of u_1 and u_2 with delay 1. This could be written $t = \Upsilon u_1 \oplus \Upsilon u_2$ where Υx denotes the previous value of variable x . In order to recover (decode) Υu_2 , process q_1 memorizes the previous value of u_1 and makes the xor with t : $\Upsilon u_2 = t \oplus \Upsilon u_1$. But if we restrict to memoryless decoding functions, then we only know u_1 and t and we cannot recover Υu_2 .

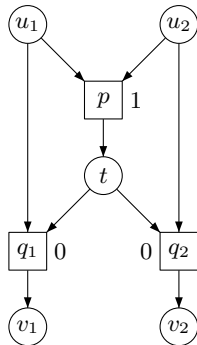


Fig. 4. A uniformly well-connected architecture

4.2 Decision criterion for UWC architectures

We first show that distributed programs are somewhat easier to find in a UWC architecture. As a matter of fact, in such architectures, to define a distributed strategy it suffices to define a collection of input-output strategies that respect the delays given by the architecture.

Lemma 8. *Let $\mathcal{A} = (V \uplus P, E, (S^v)_{v \in V}, s_0, (d_p)_{p \in P})$ be a UWC architecture. For each $v \in V_O$, let $h^v : (S^{\text{View}(v)})^+ \rightarrow S^v$ be an input-output mapping which is d -compatible. Then there exists a distributed program $F = (f^v)_{v \in V \setminus V_I}$ over \mathcal{A} such that $h^v = \hat{f}^v$ for all $v \in V_O$.*

Proof. Let $\Phi = (f^v)_{v \in V \setminus (V_I \cup V_O)}$ and $(g^{u,v})_{v \in V_O, u \in \text{View}(v)}$ be respectively the routing and the decoding functions giving the uniform well-connectedness of the architecture \mathcal{A} . We use the routing functions f^v as memoryless strategies for the

internal variables $v \in V \setminus (V_I \cup V_O)$. It remains to define f^v for $v \in V_O$. Let $\rho \in (S^{V_I})^i$ for $i > 0$ and let $\sigma \in (S^{V \setminus V_O})^i$ be the corresponding Φ -compatible sequence. For $v \in V_O$, we want to define f^v such that $f^v(\sigma^{R(v)}) = h^v(\rho^{\text{View}(v)})$. We need to verify that this is well-defined.

Let $i > 0$ and $\rho, \rho' \in (S^{V_I})^i$. Let $\sigma, \sigma' \in (S^{V \setminus V_O})^i$ be the corresponding Φ -compatible sequences, and assume $\sigma^{R(v)}[i - d_v] = \sigma'^{R(v)}[i - d_v]$. Then, for all $u \in \text{View}(v)$, $\rho^u[i - d(u, v)] = \rho'^u[i - d(u, v)]$. Indeed, for all $0 \leq j \leq i - d(u, v)$, we have $s_j^u = g^{u, v}(\sigma^{R(v)}[j + d(u, v) - d_v])$ and $s_j^u = g^{u, v}(\sigma'^{R(v)}[j + d(u, v) - d_v])$ by (1). Using $\sigma^{R(v)}[i - d_v] = \sigma'^{R(v)}[i - d_v]$ and $j + d(u, v) \leq i$ we get $s_j^u = s_j^u$ as desired. Since h^v is d -compatible, we deduce that $h^v(\rho^{\text{View}(v)}) = h^v(\rho'^{\text{View}(v)})$.

Hence for $\tau \in (S^{R(v)})^i$ with $i > 0$, we can define

$$f^v(\tau) = \begin{cases} h^v(\sigma^{\text{View}(v)}) & \text{if } \tau[i - d_v] = \sigma^{R(v)}[i - d_v] \\ & \text{for some } \Phi\text{-compatible sequence } \sigma \\ 0 & \text{otherwise} \end{cases}$$

By the above, f^v is well-defined and obviously it depends only on $\tau[i - d_v]$. Thus, it is indeed d -compatible.

Now, let $\rho \in (S^{V_I})^i$, and let σ be the F -run induced by ρ . We get, by definition, $\hat{f}^v(\rho^{\text{View}(v)}) = f^v(\sigma^{R(v)})$. Since $\sigma^{V \setminus V_O}$ is also a Φ -compatible sequence for ρ , we have $\hat{f}^v(\rho^{\text{View}(v)}) = f^v(\sigma^{R(v)}) = h^v(\rho^{\text{View}(v)})$. \square

We now give a decision criterion for this specific subclass of architectures.

Theorem 9. *A UWC architecture is decidable for external (linear or branching) specifications if and only if it has linearly preordered information.*

We have already seen in Section 3 that incomparable information yields undecidability of the synthesis problem for LTL or CTL external specifications. We prove now that, when restricted to the subclass of UWC architectures, this also becomes a necessary condition.

We assume that the architecture \mathcal{A} is UWC and has linearly preordered information, and therefore we can order the output variables $V_O = \{v_1, \dots, v_n\}$ so that $\text{View}(v_n) \subseteq \dots \subseteq \text{View}(v_1) \subseteq V_I$.

In the following, in order to use tree-automata, we extend a local strategy $f : (S^X)^+ \rightarrow S^Y$ by letting $f(\varepsilon) = s_0^Y$, so that it becomes an (S^X, S^Y) -tree. We proceed in two steps. First, we build an automaton accepting all the *global input-output 0-delay* strategies implementing the specification. A global input-output 0-delay strategy for \mathcal{A} is an $(S^{\text{View}(v_1)}, S^{V_O})$ -tree h satisfying $h(\varepsilon) = s_0^{V_O}$. This first step is simply the program synthesis for a single process with incomplete information (since we may have $\text{View}(v_1) \subsetneq V_I$). This problem was solved in [13] for CTL* specifications.

Proposition 10 ([13, Th. 4.4]). *Given an external specification $\varphi \in \text{CTL}^*(V_I \cup V_O)$, one can build a NDTA \mathfrak{A}_1 over $(S^{\text{View}(v_1)}, S^{V_O})$ -trees such that $h \in \mathcal{L}(\mathfrak{A}_1)$ if and only if the run tree induced by h satisfies φ .*

If $\mathcal{L}(\mathfrak{A}_1)$ is empty then we already know that there are no distributed implementations for the specification φ over \mathcal{A} . Otherwise, thanks to Lemma 8, we have to check whether for each $v \in V_O$ there exists an $(S^{\text{View}(v)}, S^v)$ -tree h^v which is d -compatible and such that the global strategy $\bigoplus_{v \in V_O} h^v$ induced by the collection $(h^v)_{v \in V_O}$ is accepted by \mathfrak{A}_1 . Formally, the *sum* of strategies is defined as follows. Let $X = X_1 \cup X_2 \subseteq V_I$ and $Y = Y_1 \uplus Y_2 \subseteq V_O$, and for $i = 1, 2$ let h_i be an (S^{X_i}, S^{Y_i}) -tree. We define the (S^X, S^Y) -tree $h = h_1 \oplus h_2$ by $h(\sigma) = (h_1(\sigma^{X_1}), h_2(\sigma^{X_2}))$ for $\sigma \in (S^X)^*$.

To check the existence of such trees $(h^v)_{v \in V_O}$, we will inductively eliminate the output variables following the order v_1, \dots, v_n . It is important that we start with the variable that *views* the largest set of input variables, even though, due to the delays, it might get the information much later than the remaining variables. Let $V_k = \{v_k, \dots, v_n\}$ for $k \geq 1$. The induction step relies on the following statement.

Proposition 11. *Let $1 \leq k < n$. Given a NDTA \mathfrak{A}_k accepting $(S^{\text{View}(v_k)}, S^{V_k})$ -trees, we can build a NDTA \mathfrak{A}_{k+1} accepting $(S^{\text{View}(v_{k+1})}, S^{V_{k+1}})$ -trees, such that a tree t is accepted by \mathfrak{A}_{k+1} if and only if there exists an $(S^{\text{View}(v_k)}, S^{V_k})$ -tree h^{v_k} which is d -compatible and such that $h^{v_k} \oplus t$ is accepted by \mathfrak{A}_k .*

The proof of Proposition 11 divides in two steps. Since $V_k = \{v_k\} \uplus V_{k+1}$, for each $(S^{\text{View}(v_k)}, S^{V_k})$ -tree t we have $t = t^{v_k} \oplus t^{V_{k+1}}$ (recall that t^U is the projection of t on U). So one can first transform the automaton \mathfrak{A}_k into \mathfrak{A}'_k that accepts the trees $t \in \mathcal{L}(\mathfrak{A}_k)$ such that t^{v_k} is d -compatible (Lemma 12). Then, one can build an automaton that restricts the domain of the directions and the labeling of the accepted trees to $S^{\text{View}(v_{k+1})}$ and $S^{V_{k+1}}$ respectively.

Lemma 12. *Let $v \in U \subseteq V_O$. Given a NDTA \mathfrak{A} over $(S^{\text{View}(v)}, S^U)$ -trees one can build a NDTA $\mathfrak{A}' = \text{compat}_v(\mathfrak{A})$ also over $(S^{\text{View}(v)}, S^U)$ -trees such that $\mathcal{L}(\mathfrak{A}') = \{t \in \mathcal{L}(\mathfrak{A}) \mid t^v \text{ is } d\text{-compatible}\}$.*

Proof. Intuitively, to make sure that the function t^v is d -compatible, the automaton \mathfrak{A}' will guess in advance the values of t^v and then check that its guess is correct. The guess has to be made $K = \max\{d(u, v), u \in \text{View}(v)\}$ steps in advance and consists in a d -compatible function $g : (S^{\text{View}(v)})^K \rightarrow S^v$ that predicts what will be K steps later the values of variable v . During a transition, the guess is sent in each direction $r \in S^{\text{View}(v)}$ as a function $r^{-1}g$ defined by $(r^{-1}g)(\sigma) = g(r\sigma)$ which is stored in the state of the automaton. Previous guesses are refined similarly and are also stored in the state of the automaton so that the new set of states is $Q' = Q \times \mathcal{F}$ where \mathcal{F} is the set of d -compatible functions $f : (S^{\text{View}(v)})^{<K} \rightarrow S^v$, where $Z^{<K} = \bigcup_{i < K} Z^i$. The value $f(\varepsilon)$ is the guess that was made K steps earlier and has to be checked against the current value of v in the tree.

Transitions of \mathfrak{A}' will be defined using the function $\Delta : \mathcal{F} \times S^{\text{View}(v)} \rightarrow \mathcal{P}(\mathcal{F})$ given by $\Delta(f, r) = \{f' \mid f'(\sigma) = f(r\sigma) \text{ for } |\sigma| < K - 1\}$. Note that the values $f'(\sigma)$ for $|\sigma| = K - 1$ do not depend on f and correspond to the new guess g

refined by r as intuitively described above. Now, the transition function of \mathfrak{A}' is defined for $(q, f) \in Q'$ and $s \in S^U$ with $s^v = f(\varepsilon)$ by

$$\delta'((q, f), s) = \left\{ (q_r, f_r)_{r \in S^{\text{View}(v)}} \mid \begin{array}{l} (q_r)_{r \in S^{\text{View}(v)}} \in \delta(q, s) \text{ and} \\ f_r \in \Delta(f, r) \text{ for all } r \in S^{\text{View}(v)} \end{array} \right\}.$$

Finally, the set of initial states of \mathfrak{A}' is $I' = \{q_0\} \times \mathcal{F}$ and $\alpha' = \pi^{-1}(\alpha)$ where $\pi : (Q \times \mathcal{F})^\omega \rightarrow Q^\omega$ is the projection on Q , i.e., a run of \mathfrak{A}' is accepted if and only if its projection on Q is an accepted run of \mathfrak{A} .

Let t be an $(S^{\text{View}(v)}, S^U)$ -tree accepted by \mathfrak{A} and suppose that t^v is d -compatible. Let $\rho : (S^{\text{View}(v)})^* \rightarrow Q$ be an accepting run of \mathfrak{A} over t . For $\tau \in (S^{\text{View}(v)})^*$, we define $(\tau^{-1}t^v)^{<K} : (S^{\text{View}(v)})^{<K} \rightarrow S^v$ by $(\tau^{-1}t^v)^{<K}(\sigma) = t^v(\tau\sigma)$. It is easy to see that $(\tau^{-1}t^v)^{<K}$ is d -compatible, hence it belongs to \mathcal{F} . Then we set $\rho' : (S^{\text{View}(v)})^* \rightarrow Q \times \mathcal{F}$ the full tree such that for all $\tau \in (S^{\text{View}(v)})^*$, $\rho'(\tau) = (\rho(\tau), (\tau^{-1}t^v)^{<K})$.

Since ρ is a run of \mathfrak{A} over t , we get

$$(\rho(\tau r))_{r \in S^{\text{View}(v)}} \in \delta(\rho(\tau), t(\tau)).$$

By construction, $\rho'(\tau r) = (\rho(\tau r), ((\tau r)^{-1}t^v)^{<K})$. Also, for $\sigma \in (S^{\text{View}(v)})^{<K-1}$, we have $((\tau r)^{-1}t^v)^{<K}(\sigma) = t^v(\tau r\sigma) = (\tau^{-1}t^v)^{<K}(r\sigma)$. Thus, $((\tau r)^{-1}t^v)^{<K} \in \Delta((\tau^{-1}t^v)^{<K}, r)$ for all $r \in S^{\text{View}(v)}$. Then $(\rho'(\tau r))_{r \in S^{\text{View}(v)}} \in \delta'(\rho'(\tau), t(\tau))$ and ρ' is a run of \mathfrak{A}' over t . The run ρ' is accepting since its projection on Q is ρ which is accepting.

Conversely, suppose there is an accepting run ρ' of \mathfrak{A}' over t . We need to show that t^v is d -compatible and that $t \in \mathcal{L}(\mathfrak{A})$. Let $\rho' : (S^{\text{View}(v)})^* \rightarrow Q \times \mathcal{F}$ be an accepting run of \mathfrak{A}' over t . We have $\rho' = (\rho, H)$ with $\rho : (S^{\text{View}(v)})^* \rightarrow Q$ and $H : (S^{\text{View}(v)})^* \rightarrow \mathcal{F}$. By definition of δ' , we immediately get that ρ is a run of \mathfrak{A} , which is accepting since ρ' is accepting.

It remains to prove that t^v is d -compatible. Since the transition function δ' is only defined on $((q, f), s)$ when $s^v = f(\varepsilon)$, we obtain $t^v(\tau) = H(\tau)(\varepsilon)$ for all $\tau \in (S^{\text{View}(v)})^*$. Hence, we need to show that the map $\tau \mapsto H(\tau)(\varepsilon)$ is d -compatible.

Let $\tau, \tau' \in (S^{\text{View}(v)})^i$ be such that $\tau^u[i - d(u, v)] = \tau'^u[i - d(u, v)]$ for all $u \in \text{View}(v)$. If $|\tau| = |\tau'| > K$ then we write $\tau = \tau_1\tau_2$, and $\tau' = \tau'_1\tau'_2$ with $|\tau_2| = |\tau'_2| = K$. Then, since $K \geq d(u, v)$ for $u \in \text{View}(v)$, we deduce from $\tau^u[i - d(u, v)] = \tau'^u[i - d(u, v)]$ that $\tau_1 = \tau'_1$ and $\tau_2^u[K - d(u, v)] = \tau'_2^u[K - d(u, v)]$ for all $u \in \text{View}(v)$. By successive applications of the transition function δ' and by definition of Δ , we obtain that $H(\tau)(\varepsilon) = H(\tau_1)(\tau_2)$ and $H(\tau')(\varepsilon) = H(\tau_1)(\tau'_2)$. By construction, $H(\tau_1)$ is d -compatible. Then we get $H(\tau_1)(\tau_2) = H(\tau_1)(\tau'_2)$ and therefore, $H(\tau)(\varepsilon) = H(\tau')(\varepsilon)$.

If $|\tau| < K$, then in an analogous manner we obtain $H(\tau)(\varepsilon) = H(\varepsilon)(\tau) = H(\varepsilon)(\tau') = H(\tau')(\varepsilon)$ since $H(\varepsilon)$ is d -compatible. \square

Proof (of Proposition 11). We consider the NDTA $\text{compat}_{v_k}(\mathfrak{A}_k)$. It remains to project away the S^{v_k} component of the label and to make sure that the $S^{V_{k+1}}$ component of the label only depends on the $S^{\text{View}(v_{k+1})}$ component of

the input. The first part is the classical projection on $S^{V_{k+1}}$ of the automaton and the second part is the *narrowing* construction introduced in [13]. The automaton \mathfrak{A}_{k+1} fulfilling the requirements of Proposition 11 is therefore given by $\text{narrow}_{\text{View}(v_{k+1})}(\text{proj}_{V_{k+1}}(\text{compat}_{v_k}(\mathfrak{A}_k)))$. Note that, even when applied to a NDTA, the narrowing construction of [13] yields an *alternating* tree automaton. Here we assume that the narrowing operation returns a NDTA using a classical transformation of alternating tree automata into NDTA [18]. The drawback is that this involves an exponential blow up. Unfortunately, this is needed since Lemma 12 requires a NDTA as input. \square

We can now conclude the proof of Theorem 9. Using Proposition 11 inductively starting from the NDTA \mathfrak{A}_1 of Proposition 10, we obtain a NDTA \mathfrak{A}_n accepting an $(S^{\text{View}(v_n)}, S^{v_n})$ -tree h^{v_n} if and only if for each $1 \leq i < n$, there exists an $(S^{\text{View}(v_i)}, S^{v_i})$ -tree h^{v_i} which is d -compatible and such that $h^{v_1} \oplus \dots \oplus h^{v_n}$ is accepted by \mathfrak{A}_1 . Therefore, using Remark 1 and Lemma 8, there is a distributed implementation for the specification over \mathcal{A} if and only if $\mathcal{L}(\text{compat}_{v_n}(\mathfrak{A}_n))$ is nonempty. The overall procedure is non-elementary due to the exponential blow-up of the inductive step in Proposition 11. \square

4.3 Decidability for UWC architectures and robust specifications

We now show that we can obtain decidability of the synthesis problem for the whole subclass of UWC architectures by restricting ourselves to specifications that only relate output variables to their own view.

Definition 13. A specification $\varphi \in \mathcal{L}$ with $\mathcal{L} \in \{LTL, CTL, CTL^*\}$ is robust if it is a (finite) disjunction of formulas of the form $\bigwedge_{v \in V_O} \varphi_v$ where $\varphi_v \in \mathcal{L}(\text{View}(v) \cup \{v\})$. Note that a robust formula is always external.

Proposition 14. The synthesis problem for robust CTL^* specifications is decidable over UWC architectures.

Proof. Let $\mathcal{A} = (V \uplus P, E, (S^u)_{u \in V}, s_0, (d_p)_{p \in P})$ be a UWC architecture and φ be a robust CTL^* specification. Without loss of generality, we may assume that $\varphi = \bigwedge_{v \in V_O} \varphi_v$ where $\varphi_v \in CTL^*(\text{View}(v) \cup \{v\})$. Using Proposition 10, for each $v \in V_O$ we find a NDTA \mathfrak{A}_v accepting a strategy $h : (S^{\text{View}(v)})^* \rightarrow S^v$ if and only if the induced run tree $t : (S^{\text{View}(v)})^* \rightarrow S^{\text{View}(v) \cup \{v\}}$ satisfies φ_v . The proposition then follows from the

Claim. There exists a distributed implementation of φ over \mathcal{A} if and only if for each $v \in V_O$, the automaton $\text{compat}_v(\mathfrak{A}_v)$ is nonempty.

First, let F be a distributed implementation of φ over \mathcal{A} and let $t : (S^{V_i})^* \rightarrow S^V$ be the induced run-tree. Fix some $v \in V_O$. The map $\hat{f}^v : (S^{\text{View}(v)})^* \rightarrow S^v$ is d -compatible by Remark 1. Let $t' : (S^{\text{View}(v)})^* \rightarrow S^{\text{View}(v) \cup \{v\}}$ be the run-tree induced by \hat{f}^v . For each $\sigma \in (S^{V_i})^*$ we have $t(\sigma)^{\text{View}(v) \cup \{v\}} = t'(\sigma^{\text{View}(v)})$. Since F implements φ , we have $t \models \varphi$ and then $t \models \varphi_v$. We can prove by

structural induction on the formula that for any $\psi \in \text{CTL}^*(\text{View}(v) \cup \{v\})$, any branch $\sigma \in (S^{V_i})^\omega$ and any position i we have $t, \sigma, i \models \psi$ if and only if $t', \sigma^{\text{View}(v)}, i \models \psi$. Since $\varphi_v \in \text{CTL}^*(\text{View}(v) \cup \{v\})$, we deduce that $t' \models \varphi_v$. Therefore, \hat{f}^v is accepted by \mathfrak{A}_v and also by $\text{compat}_v(\mathfrak{A}_v)$.

Conversely, for each $v \in V_O$, let $h^v : (S^{\text{View}(v)})^* \rightarrow S^v$ be a strategy accepted by $\text{compat}_v(\mathfrak{A}_v)$. By Lemma 12, h^v is d -compatible. Let $t_v : (S^{\text{View}(v)})^* \rightarrow S^{\text{View}(v) \cup \{v\}}$ be the run-tree induced by h^v . We have $t_v \models \varphi_v$ by definition of \mathfrak{A}_v and Proposition 10. Now, using Lemma 8 we find a distributed program $F = (f^v)_{v \in V \setminus V_i}$ such that $\hat{f}^v = h^v$ for each $v \in V_O$. Let $t : (S^{V_i})^* \rightarrow V^{V_i \cup V_O}$ be the run-tree induced by F . For each $\sigma \in (S^{V_i})^*$ we have $t(\sigma)^{\text{View}(v) \cup \{v\}} = t_v(\sigma^{\text{View}(v)})$ and we obtain as above that $t \models \varphi_v$. Therefore, $t \models \varphi$ and F implements φ on \mathcal{A} . \square

5 Well-connected architectures

It is natural to ask whether the decision criterion for UWC architectures can be extended to a larger class. In this section, we relax the property of uniform well-connectedness and show that, in that case, linearly preordered information is not anymore a sufficient condition for decidability.

Definition 15. *An architecture is said to be well-connected, if for each output variable $v \in V_O$, the sub-architecture consisting of $(E^{-1})^*(v)$ is uniformly well-connected.*

Intuitively this means that for each output variable v there is a routing making it possible to transmit the values of the input variables in $\text{View}(v)$ to the process that writes on v , but such a routing may vary from one output variable to another, contrary to the UWC where a single routing is used for all output variables. For instance, the architecture of Figure 2 is well-connected. Indeed, to transmit the values of u and v to z_{ij} , it is enough to write u on z_i and v on z_j . Note that this does not give a uniform routing. Actually, the architecture of Figure 2 is not UWC assuming that variables values range over $\{0, 1\}$ (see Proposition 17). Hence, the subclass of UWC architectures is strictly contained in the subclass of well-connected architectures.

In the proof of Proposition 17, we use the following lemma, established in [23] for solving the network information flow problem introduced in [2].

We say that two functions f and g from S^2 to S are independent if $(f, g) : S^2 \rightarrow S^2$ is invertible.

Lemma 16 ([23, Lemma 3.1]). *If f^1, \dots, f^n are pairwise independent functions from S^2 to S then $n \leq |S| + 1$.*

This lemma asserts that over a small alphabet, one cannot build a large set of pairwise independent functions. In our setting, it implies the following result:

Proposition 17. *Assuming that all variables are Boolean, the architecture of Figure 2 is well-connected but not uniformly well-connected.*

Proof. It is easy to see that the architecture \mathcal{A} of Figure 2 is well-connected. However, it is not uniformly well-connected. Indeed, suppose it is. Then there would exist a routing $\Phi = (f^{z_1}, f^{z_2}, f^{z_3}, f^{z_4})$ consisting of four memoryless strategies, and for all $v \in V_O$, a decoding function $g^v : \{0, 1\}^2 \rightarrow \{0, 1\}^2$. Therefore, uniform well-connectedness of \mathcal{A} implies that every pair (f^{z_i}, f^{z_j}) would be invertible, the inverse being $g^{z_{ij}}$. Lemma 16 implies that for Boolean variables, there is at most three pairwise independent functions. Hence the architecture cannot be uniformly well-connected. \square

Interestingly enough, the size of the alphabet has an influence on the possibility to have a *uniform* routing and Lemma 16 helps to understand why. In our setting, this means that by enlarging the domains of internal variables, we may obtain uniform well-connectedness from a well-connected architecture.

The following theorem asserts that, unfortunately, the decision criterion cannot be extended to well-connected architectures.

Theorem 18. *The synthesis problem for LTL specifications and well-connected, linearly preordered architectures is undecidable.*

Let \mathcal{A} be the architecture of Figure 5, in which all the delays are set to 0, and which is clearly well-connected and linearly preordered. To show its undecidability, fix a deterministic Turing machine M with tape alphabet Γ and state set Q . We reduce the non halting problem of M starting from the empty tape to the distributed implementability of an LTL specification over \mathcal{A} . Let $S^z = \{0, 1\}$ for $z \in V \setminus \{x, y\}$ and $S^x = S^y = \Gamma \uplus Q \uplus \{\#\}$ where $\#$ is a new symbol. As usual, the configuration of M defined by state q and tape content $\gamma_1\gamma_2$, where the head scans the first symbol of γ_2 , is encoded by the word $\gamma_1q\gamma_2 \in \Gamma^*Q\Gamma^+$ (we require that $\gamma_2 \neq \varepsilon$ for technical reasons, including in it some blank symbols if necessary). An input word $u \in 0^*1^p0\{0, 1\}^\omega$ encodes the integer $n(u) = p$ and similarly for v . We construct an LTL specification φ_M forcing any distributed implementation to output on variable x the $n(u)$ -th configuration of M starting from the empty tape. Processes p_0 and p_6 play the role of the two processes of the undecidable architecture of Pnueli and Rosner (\mathcal{A}' in Figure 3). The difficulty is to ensure that process p_6 cannot receive relevant information about u .

The specification $\varphi_M = \alpha \wedge \beta \wedge \gamma_M \wedge \delta \wedge \psi_M$ is a conjunction of five properties described below that can all be expressed in LTL($V_I \cup V_O$).

1. The processes p_i for $1 \leq i \leq 5$ have to output the current values of (u, w) on (u_i, w_i) until (including) the first 1 occurs on w . Afterwards, they are unconstrained. Process p_6 must always output the value of w on w_6 . Moreover, after the first 1 on w , it also has to output the current value of u on u_6 . Formally, this is defined by the LTL formula α :

$$\alpha \stackrel{\text{def}}{=} \mathbf{G}(w_6 = w) \wedge \left[((w = 0) \wedge \alpha') \mathbf{W} ((w = 1) \wedge \alpha' \wedge \mathbf{X} \mathbf{G}(u_6 = u)) \right], \text{ where}$$

$$\alpha' \stackrel{\text{def}}{=} \bigwedge_{1 \leq k \leq 5} (u_k = u) \wedge (w_k = w)$$

2. If the input word on u (resp. v) is in $0^q 1^p 0 \{0, 1\}^\omega$, then the corresponding output word x (resp. y) is in $\#^{q+p} \Gamma^* Q \Gamma^+ \#^\omega$. This is expressed by $\beta = \beta_{u,x} \wedge \beta_{v,y}$, where

$$\beta_{z,t} \stackrel{\text{def}}{=} ((z = 0) \wedge (t = \#)) \text{W} \left((z = 1) \wedge \left(((z = 1) \wedge (t = \#)) \text{W} ((z = 0) \wedge (t \in \Gamma^* Q \Gamma^+ \#^\omega)) \right) \right)$$

where

$$(t \in \Gamma^* Q \Gamma^+ \#^\omega) \stackrel{\text{def}}{=} (t \in \Gamma) \text{U} ((t \in Q) \wedge \text{X}(t \in \Gamma) \text{U} ((t \in \Gamma) \wedge \text{XG}(t = \#)))$$

3. We next express with a formula γ_M that if $n(u) = 1$ then x has to output the first configuration \mathcal{C}_1 of M starting from the empty tape. That is, if the input is in $0^q 1^0 \{0, 1\}^\omega$, then the corresponding output is $\#^{q+1} \mathcal{C}_1 \#^\omega$. The LTL formula is

$$\gamma_M \stackrel{\text{def}}{=} (u = 0) \text{W} ((u = 1) \wedge \text{X}((u = 0) \rightarrow (x \in \mathcal{C}_1 \#^\omega)))$$

where $(x \in \mathcal{C}_1 \#^\omega)$ can be expressed easily.

4. We say that the input words are *synchronized* if $u, v \in 0^q 1^p 0 \{0, 1\}^\omega$ or if $u \in 0^q 1^{p+1} 0 \{0, 1\}^\omega$ and $v \in 0^{q+1} 1^p 0 \{0, 1\}^\omega$. We use a formula δ to express the fact that if u and v are synchronized and $n(u) = n(v)$, then the outputs on x and y are equal. We first define the LTL formula

$$(n(u) = n(v)) \stackrel{\text{def}}{=} (u = v = 0) \text{U} ((u = v = 1) \wedge (u = v = 1) \text{U} (u = v = 0))$$

to express the fact that the input words u and v are synchronized and $n(u) = n(v)$. Then the formula δ is defined by:

$$\delta \stackrel{\text{def}}{=} (n(u) = n(v)) \rightarrow \text{G}(x = y)$$

5. Finally, one can express with an LTL formula ψ_M that if the input words are synchronized and $n(u) = n(v) + 1$ then the configuration encoded on x is

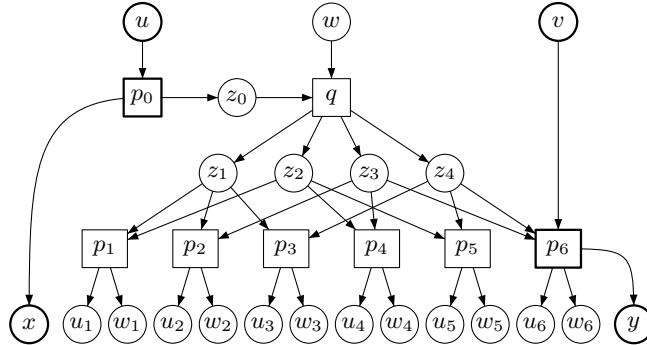


Fig. 5. Undecidable, well-connected and linearly preordered architecture

obtained by a computation step of M from the configuration encoded on y . We use the LTL formula $(n(u) = n(v) + 1)$ defined by

$$(u = v = 0) \text{ U } \left((u = 1) \wedge (v = 0) \wedge \text{X}((u = v = 1) \wedge (u = v = 1) \text{ U } (u = v = 0)) \right)$$

to express the fact that u and v are synchronized and $n(u) = n(v) + 1$. The formula ψ_M is defined by

$$\psi_M = (n(u) = n(v) + 1) \rightarrow \left((x = y) \text{ U } (\text{Trans}(y, x) \wedge \text{X}^3 \text{G}(x = y)) \right)$$

where $\text{Trans}(y, x)$ expresses the fact that the factor of length 3 of x is obtained from the one of y by a transition of the Turing machine M . We have

$$\begin{aligned} \text{Trans}(y, x) = & \bigvee_{(p,a,q,b,\leftarrow) \in T, c \in \Gamma} (y = cpa) \wedge (x = qcb) \\ & \vee \bigvee_{(p,a,q,b,\rightarrow) \in T, c \in \Gamma} (y = pac) \wedge (x = bqc) \\ & \vee \bigvee_{(p,a,q,b,\rightarrow) \in T} (y = pa\#) \wedge (x = bq\Box) \end{aligned}$$

Here $(x = abc)$ is an abbreviation for $(x = a) \wedge \text{X}(x = b) \wedge \text{X}^2(x = c)$. Furthermore, \Box is the blank symbol of the tape and T is the set of transitions of M (the transition (p, a, q, b, dir) , taken when M is in state p and scans symbol a , switches the state to q , writes symbol b and moves the head according to the direction $dir \in \{\leftarrow, \rightarrow\}$).

We first show that there exists a distributed implementation of φ_M over \mathcal{A} . Let \oplus be the addition modulo 2 (XOR). Process p_0 forwards u to z_0 . Process q forwards u to z_1 , $u \oplus w$ to z_2 and w to z_3 . The strategy for z_4 is not memoryless. Process q forwards w to z_4 until (including) the first 1 on w and then it forwards $u \oplus w$ to z_4 . Formally, $f^{z_4}(u, 0^q b) = b$ and $f^{z_4}(ua, 0^q 1wb) = a \oplus b$. We also use memoryless strategies for the processes p_i so that α is satisfied. For instance, the strategy for p_1 is $f^1(b_1, b_2) = (b_1, b_1 \oplus b_2)$ and the strategy for p_6 (y excluded) is $f^6(b_3, b_4) = (b_3 \oplus b_4, b_3)$. It is easy to see that with these strategies, the first property α of the specification is satisfied. Note that, until the first 1 on w , p_6 outputs 0 on u_6 , and after this first 1, p_5 cannot decode u and w anymore.

The strategy f^x (respectively f^y) is to output the p -th configuration of M starting from the empty tape when u (respectively v) encodes p . Then, the rest of the specification, $\beta \wedge \gamma_M \wedge \delta \wedge \psi_M$, is satisfied.

Remark 19. Actually, one can define another distributed implementation by changing only the strategy f^{z_4} : at each step, process q transmits to p_6 the value of u at the preceding step as the mod 2 difference between z_3 and z_4 , until the first 1 occurs on w . Formally, $f^{z_4}(a, b) = b$, $f^{z_4}(u \cdot a_1 \cdot a_2, 0^q b) = a_1 \oplus b$ and $f^{z_4}(ua, 0^q 1wb) = a \oplus b$. We also adapt the strategies of p_1, \dots, p_6 so that α

is satisfied. Note that these strategies are no longer memoryless, they have to remember the last bit of u . By XORing its two arguments, process p_6 can then recover the whole history of u , except the bit occurring simultaneously with the first 1 of w . Hence, we are almost in the situation of the decidable architecture of Figure 1, but surprisingly, *missing only one bit of information* suffices to yield undecidability.

Let now $F = (f^v)_{v \in V \setminus V_1}$ be a distributed implementation of φ_M on the architecture \mathcal{A} of Figure 5. We prove that f^x must simulate the computation of M starting from the empty tape.

Step 1: relating the strategies for z_3 and z_4 .

Lemma 20. *Let $g_1, g_2, g_3 : \{0, 1\}^2 \rightarrow \{0, 1\}$ be pairwise independent functions. Then, there exists $\varepsilon \in \{0, 1\}$ such that for all $a, b \in \{0, 1\}$:*

$$g_3(a, b) = \varepsilon \oplus g_1(a, b) \oplus g_2(a, b)$$

Proof. We first note that each function g_k is two to one, i.e., $|g_k^{-1}(c)| = 2$ for $c \in \{0, 1\}$. Indeed, if this is not the case then we have for instance $|g_k^{-1}(0)| \geq 3$ and the map (g_k, g_ℓ) for $\ell \neq k$ cannot be injective.

For the same reason, if $g_k(a, b) = g_k(a', b')$, then $g_\ell(a, b) \neq g_\ell(a', b')$. Therefore, permuting indices if necessary, we may assume that $g_1(0, 0) = g_1(0, 1)$, $g_2(0, 0) = g_2(1, 0)$ and $g_3(0, 0) = g_3(1, 1)$, so that each g_k is completely determined by its value on $(0, 0)$. A simple computation then shows that $g_1 \oplus g_2 \oplus g_3$ is constant. For instance, we have $(g_1 \oplus g_2 \oplus g_3)(1, 0) = (\neg g_1 \oplus g_2 \oplus \neg g_3)(0, 0) = (g_1 \oplus g_2 \oplus g_3)(0, 0)$. \square

Applying Lemma 20 both to $(\hat{f}^{z_1}, \hat{f}^{z_2}, \hat{f}^{z_3})$ and $(\hat{f}^{z_1}, \hat{f}^{z_2}, \hat{f}^{z_4})$ after an input $(0^q, 0^q)$ on (u, w) , we get:

Corollary 21. *For all $q \geq 0$, there exists $\varepsilon \in \{0, 1\}$ such that*

$$\forall a, b \in \{0, 1\}, \quad \hat{f}^{z_3}(0^q a, 0^q b) = \varepsilon \oplus \hat{f}^{z_4}(0^q a, 0^q b).$$

Proof. Fix $q \geq 0$. Let $g_i : \{0, 1\}^2 \rightarrow \{0, 1\}$ be defined by $g_i(a, b) = \hat{f}^{z_i}(0^q a, 0^q b)$. The conjunct α of the specification φ_M imposes to p_1, p_2 and p_4 to output the current value of (u, w) , hence they must distinguish the four possible values of (u, w) . Therefore, g_1, g_2 and g_3 are pairwise independent. Applying Lemma 20, we obtain $\varepsilon_3 \in \{0, 1\}$ such that $g_3(a, b) = \varepsilon_3 \oplus g_1(a, b) \oplus g_2(a, b)$ for all $(a, b) \in \{0, 1\}^2$. Similarly, considering outputs of processes p_1, p_3, p_5 , we deduce that g_1, g_2 and g_4 are also pairwise independent and that $g_4(a, b) = \varepsilon_4 \oplus g_1(a, b) \oplus g_2(a, b)$.

Therefore, for all $(a, b) \in \{0, 1\}^2$, we have $g_3(a, b) \oplus g_4(a, b) = \varepsilon_3 \oplus \varepsilon_4 = \varepsilon$ and we obtain $\hat{f}^{z_3}(0^q a, 0^q b) = \varepsilon \oplus \hat{f}^{z_4}(0^q a, 0^q b)$ as desired. \square

Step 2: masking one bit of u to p_6 .

Let $q \geq 0$. For $u = 0^q 1 u'$, we define $u^0 = 0^q 0 u'$. Observe that if $u \in 0^q 1^{p+1} 0 \{0, 1\}^\omega$ encodes $p + 1 > 1$ then $u^0 \in 0^{q+1} 1^p 0 \{0, 1\}^\omega$ encodes p . The next

lemma states that strategies f^{z_3} (resp. f^{z_4}) must output the same sequence for u and u^0 if the input word w is suitable, so that p_6 cannot distinguish between encodings of p and $p + 1$ on input variable u .

Lemma 22. *Let $u, w \in 0^q 1 \{0, 1\}^\omega$. For $k \in \{3, 4\}$, we have for all $n > 0$:*

$$\hat{f}^{z_k}(u^0[n], w[n]) = \hat{f}^{z_k}(u[n], w[n]). \quad (2)$$

Proof. We prove the lemma by induction on n . If $n \leq q$, then $u^0[n] = u[n]$ so (2) trivially holds.

Next, assume $n = q + 1$, so $u^0[n] = 0^q 0$ and $u[n] = 0^q 1 = w[n]$. Assume $\hat{f}^{z_3}(0^q 0, 0^q 0) = \hat{f}^{z_3}(0^q 0, 0^q 1)$ then we have $\hat{f}^{z_4}(0^q 0, 0^q 0) = \hat{f}^{z_4}(0^q 0, 0^q 1)$ by Corollary 21. Fixing some $v \in \{0, 1\}^n$, we deduce that process p_6 has observed exactly the same history on the input triples $(0^q 0, 0^q 0, v)$ and $(0^q 0, 0^q 1, v)$, therefore it would write at step n the same value on w_6 , a contradiction with requirement α . Therefore, $\hat{f}^{z_3}(0^q 0, 0^q 0) \neq \hat{f}^{z_3}(0^q 0, 0^q 1)$. Similarly, $\hat{f}^{z_3}(0^q 0, 0^q 0) \neq \hat{f}^{z_3}(0^q 1, 0^q 1)$. Since the map \hat{f}^{z_3} may only take two values, we get $\hat{f}^{z_3}(0^q 0, 0^q 1) = \hat{f}^{z_3}(0^q 1, 0^q 1)$. Applying again Corollary 21, we deduce that $\hat{f}^{z_4}(0^q 0, 0^q 1) = \hat{f}^{z_4}(0^q 1, 0^q 1)$ and (2) is proved for $n = q + 1$.

Finally, assume that $n > q + 1$. By induction hypothesis, for $k \in \{3, 4\}$, we have $\hat{f}^{z_k}(u^0[i], w[i]) = \hat{f}^{z_k}(u[i], w[i])$ for all $i < n$. Therefore, the history $z_3[n - 1]$ and $z_4[n - 1]$ is the same for the input (u, w) and (u^0, w) . Fixing some $v \in \{0, 1\}^n$, we deduce that process p_6 has observed exactly the same history on the input triples $(u^0[n - 1], w[n - 1], v[n - 1])$ and $(u[n - 1], w[n - 1], v[n - 1])$.

Consider now the 3 mappings from $\{0, 1\}^2$ to $\{0, 1\}^2$ defined by

$$\begin{aligned} h(c, d) &= (f^{u_6}, f^{w_6})(z_3[n - 1]c, z_4[n - 1]d, v) \\ h_1(a, b) &= (\hat{f}^{z_3}, \hat{f}^{z_4})(u[n - 1]a, w[n - 1]b) \\ h_0(a, b) &= (\hat{f}^{z_3}, \hat{f}^{z_4})(u^0[n - 1]a, w[n - 1]b) \end{aligned}$$

We deduce from the requirement α that h is an inverse of h_1 and also an inverse of h_0 . Therefore, $h_0 = h_1$ and we obtain $\hat{f}^{z_k}(u^0[n], w[n]) = \hat{f}^{z_k}(u[n], w[n])$ for $k \in \{3, 4\}$ as required. \square

Step 3: enforcing output of the $n(u)$ -th configuration of M on x .

Lemma 23. *If x is computed by f^x from the input word u then for all $p > 0$ we have*

$$\forall q \geq 0, \quad u \in 0^q 1^p 0 \{0, 1\}^\omega \implies x = \#^{p+q} \mathcal{C}_p \#^\omega \quad (3)$$

where \mathcal{C}_p is the p -th configuration reached by M starting from the empty tape.

Proof. The proof is by induction on p . The case $p = 1$ follows from the specification γ_M . Let now $p > 1$ and assume that $u \in 0^q 1^{p+1} 0 \{0, 1\}^\omega$. Let $v = 0^{q+1} 1^p 0^\omega$ and $w = 0^q 1^\omega$. By induction, for $u^0 \in 0^{q+1} 1^p 0 \{0, 1\}^\omega$ the output is $x = \#^{q+1+p} \mathcal{C}_p \#^\omega$. Using δ , we deduce that on the input triple (u^0, w, v) the output is $y = x = \#^{q+1+p} \mathcal{C}_p \#^\omega$. Now, by Lemma 22, on the input pairs (u^0, w)

and (u, w) , the outputs on z_3 and z_4 are the same. Hence, on the input triples (u^0, w, v) and (u, w, v) the outputs on y must be $y = \#^{q+1+p}C_p\#\omega$ by the above. Using ψ_M , we deduce that on the input triple (u, w, v) the output on x must be $x = \#^{q+1+p}C_{p+1}\#\omega$. This concludes the proof since x only depends on u . \square

By masking one bit of u to p_6 , we cause uncertainty with respect to the value of $n(u)$, preventing this process to “cheat”. In turn, process p_0 , which has no information about the other input values, only knows that p_6 is not always able to cheat, and has then to always output the correct Turing machine configuration.

Proof of Theorem 18. Starting from a Turing machine M , we have built a specification φ_M such that its distributed implementation forces x to output the $n(u)$ -th configuration of M . Therefore, there is a distributed implementation on this architecture for the formula $\varphi_M \wedge G(x \neq \text{halt})$ if and only if M does not halt starting from the empty tape. We have thus reduced the non halting problem of a Turing machine on the empty tape to the LTL distributed synthesis problem over a well-connected linearly preordered architecture, proving that this latter problem is undecidable (more precisely not co-RE). \square

6 Conclusion

In this paper, we have shown that every decidable architecture is linearly preordered, and that this condition is sufficient for deciding external specifications on UWC architectures. On the other hand, we have exhibited a linearly preordered, yet undecidable well-connected architecture for external LTL specifications, by simulating the loss of a single information bit on the UWC architecture of Figure 1.

Finally, we have shown that all UWC architectures are decidable for *robust* specifications, i.e., specifications constraining external variables which are causally related by a communication path. A challenging problem is to find whether this still holds for well-connected architectures.

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In G. A. et al., editor, *Proc. 17th ICALP*, Lect. Notes Comp. Sci. Springer-Verlag, 1989.
2. R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Inform. Theory*, 46(4):1204–1216, 2000.
3. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoret. Comput. Sci.*, 303:7–34, 2003.
4. J. Bernet and D. Janin. On distributed program specification and synthesis in architectures with cycles. In E. Najm, J. Pradat-Peyre, and V. Vigié Donzeau-Gouge, editors, *26th IFIP WG 6.1 International Conference on Formal Methods for Networked and Distributed Systems (FORTE 2006)*, volume 4229 of *Lect. Notes Comp. Sci.*, pages 175–190. Springer, 2006.

5. A. Church. Logic, arithmetic, and automata. In *Int. Symp. of Mathematicians*, pages 23–35, 1962.
6. B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proc. 20th IEEE Symp. on Logic in Computer Science (LICS 2005)*. IEEE Computer Society, 2005.
7. B. Finkbeiner and S. Schewe. Synthesis of asynchronous systems. In *LOPSTR'06*, Lect. Notes Comp. Sci. Springer-Verlag, 2006.
8. P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS'04*, Lect. Notes Comp. Sci., pages 275–286. Springer, 2004.
9. P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. In N. Garg and S. Arun-Kumar, editors, *Proceedings of FSTTCS'06*, volume 4337 of *Lect. Notes Comp. Sci.*, pages 321–332. Springer, 2006.
10. O. Kupferman and M. Vardi. Synthesis with incomplete information. In *Proc. ICTL*, pages 91–106, July 1997.
11. O. Kupferman and M. Y. Vardi. μ -calculus synthesis. In *Proceedings of MFCS'00*, 2000.
12. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proceedings of LICS'01*, pages 389–398. Computer Society Press, 2001.
13. O. Kupferman and M. Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245–263, June 1999.
14. P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In *Proceedings of ICALP'01*, volume 2076 of *Lect. Notes Comp. Sci.*, pages 396–407. Springer, 2001.
15. P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In *Proceedings of CONCUR'02*, volume 2421 of *Lect. Notes Comp. Sci.* Springer, 2002.
16. P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *FSTTCS'05*, pages 201–212, 2005.
17. S. Mohalik and I. Walukiewicz. Distributed games. In *Proceedings of FSTTCS'03*, volume 2914 of *Lect. Notes Comp. Sci.* Springer, 2003.
18. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoret. Comput. Sci.*, 2(1):90–121, 1995.
19. G. Peterson and J. Reif. Multiple-person alternation. In *20th Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico, 1979)*, pages 348–363. IEEE, New York, 1979.
20. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Automata, languages and programming (Stresa, 1989)*, volume 372 of *Lect. Notes Comp. Sci.*, pages 652–671. Springer, 1989.
21. A. Pnueli and R. Rosner. On the synthesis of reactive module. In *Proceedings of the 16th ACM symposium on principles of programming languages*, Austin, January 1989.
22. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of 31th IEEE Symp. FOCS*, pages 746–757, 1990.
23. A. Rasala Lehman and E. Lehman. Complexity classification of network information flow problems. In *Proceedings of SODA'04*, pages 142–150. SIAM, 2004.
24. A. Stănescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. In *CONCUR 2003 - Concurrency Theory*, volume 2761 of *Lect. Notes Comp. Sci.*, pages 27–41. Springer, 2003.
25. W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of STACS'95*, volume 900 of *Lect. Notes Comp. Sci.*, pages 1–13. Springer, 1995.

26. R. van der Meyden and M. Vardi. Synthesis from knowledge-based specifications. In *9th International Conf. on Concurrency Theory (CONCUR'98)*, Lect. Notes Comp. Sci., pages 34–49. Springer, Sept. 1998.
27. R. van der Meyden and T. Wilke. Synthesis of distributed systems from knowledge-based specifications. In *CONCUR 05*, Lect. Notes Comp. Sci., pages 562–562. Springer-Verlag, 2005.