Distributed synthesis for well-connected architectures*

Paul Gastin[1], Nathalie Sznajder[1], and Marc Zeitoun[2]

[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, Université Bordeaux 1 & 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.

Key words: Synthesis problem · Distributed systems · Synchronous architectures.

1 Introduction

Synthesis is an essential problem in computer science introduced by Church [6]. It consists in translating a system property which relates input and output events, into a low-level model which computes 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.

The controller synthesis problem, in which a system is also part of the input, extends the synthesis problem. The goal is to synthesize a controller such that the system, synchronized with the controller, meets the given specification. Thus, the synthesis problem corresponds to the particular case of the controller synthesis problem with a system having all possible behaviors. Both problems have a classical formulation in terms of games. See for instance [27, 28] for a presentation of relationships between two-player infinite games in an automata-theoretic setting, and the synthesis problem. Both problems also have several variants. Let us review some of them, in order to relate the contribution of the present paper to existing work.

* Partially supported by projects ARCUS Île-de-France/Inde, DOTS (ANR-06-SETIN-003), and P2R MODISTE-COVER/Timed-DISCOVERI.

1.1 Some variants of the synthesis problem

Closed vs. open systems. Early approaches consider closed systems, in which there is no interaction with an environment [7]. Synthesis has later been extended to open systems [22, 1], that is, to systems interacting with an unpredictable environment. The goal is to enforce the specification no matter how the environment acts. In this work, we consider open systems.

Centralized vs. distributed systems. A solution to Church's problem for centralized systems has been presented by Büchi and Landweber [5], for monadic second order specifications. A distributed system is made up of several communicating processes. The additional difficulty showing up with distributed systems is that the information acquired by each individual process about the global state of the system is only partial. Indeed, data exchanges between processes are constrained by a given communication architecture. For controller synthesis, the controller itself is required to be distributed over the same communication architecture, so that each of its components cannot have a complete knowledge of what happens. In this paper we also consider distributed systems.

Algorithms solving the synthesis problem with incomplete information are given in [12, 14, 3] for branching-time logic specifications. Synthesis has also been studied for specifications in the logic of knowledge and linear time, in [29] for systems with a single agent, and in [30] for distributed systems. The game-theoretic framework remains useful in the distributed case [4]. Unifying several formalisms, [19] proposed the framework of distributed games, a specialized variant of multiplayer games, to reason about distributed synthesis.

Synchronous vs. asynchronous systems. For distributed systems, two classical semantics have been previously considered. In synchronous systems, there is a global clock, and each process executes one computation step at each clock tick. In asynchronous systems, there is no such global clock: each process behaves at its own speed. This paper considers synchronous systems. Only a few cases for such systems have been identified as decidable. See [24, 15] where the problem is studied for temporal logic specifications.

Full vs. local vs. external specifications. In addition to the specification language itself, another natural parameter concerns the variables that a specification is allowed to refer to. Variables are of three kinds: input variables carry the values provided by the environment. Output variables are written by the system, and are not used for internal communication. Finally, for a distributed system, there is a fixed number of variables, called internal, corresponding to communication links between processes. We define three types of specifications:

- Full specifications are the most general ones: they may refer to any variable.
- External specifications only refer to input and output variables, but not to internal ones.
- Local specifications are Boolean combinations of p-local specifications, where p denotes a process. For a given process p, a specification is said *p-local* if it only refers to variables read or written by process p.

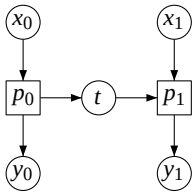

Fig. 1. Architecture decidable/undecidable for external/full specifications.

In this work, we use external specifications. Before discussing this choice and presenting our contributions, let us review the most salient existing results on the synthesis problem.

1.2 Synthesis for distributed systems: related work

For asynchronous systems, synthesis has first been studied in [23] for single-process implementations and linear-time specifications. In [17], the synthesis problem in the distributed setting is proved decidable for trace-closed specifications, yet for a quite specific class of controllers. This result has been strengthened in [18], where restrictions on the communication patterns of the controllers have been relaxed. Another subclass of decidable systems, incomparable with the preceding one, has been identified in [10], using an enhanced memory for controllers. The synthesis of asynchronous distributed systems in the general case of μ-calculus specifications was studied in [9]. Also, the theory of asynchronous automata has been applied in [26] to solve the synthesis problem of *closed* distributed systems.

For synchronous systems, undecidability is the point in common to most existing results. This question has been first studied in [24], where synthesis has been proved undecidable for LTL specifications and arbitrary architectures. For pipeline architectures (where processes are linearly ordered and each process communicates to its right neighbor), synthesis becomes non elementarily decidable for LTL specifications. The lower bound follows from a former result on multiplayer games [21]. Even for local specifications, constraining only variables local to processes, the problem is still undecidable for most communication architectures [16]. Synthesis has been shown decidable for the pipeline architecture and CTL* *full* specifications [15]. A decision criterion for full specifications has then been established in [8]. It implies that the problem is undecidable for the architecture of Figure 1. The reason is that full specifications make it possible to enforce a constant value on variable t, breaking the communication link between processes p_0 and p_1.

1.3 Contributions

We address the synthesis problem for open distributed synchronous systems and temporal logic specifications. In contrast to the situation in the asynchronous setting, most decidability results for synthesis of synchronous systems are negative. The goal of this paper is to investigate relevant restrictions to obtain decidability. Undecidability often

arises when dealing with full specifications. For the rare positive statements, as for the pipeline architecture, allowing full specifications strengthen the decidability result [15]. On the other hand, for the undecidability part of the criterion obtained in [8], allowing full specifications weakens the result by yielding easy reductions to the basic undecidable architecture of Pnueli and Rosner [24] (see Figure 1), for instance by breaking communication links at will.

In the seminal paper [24], 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 synthesis problem is decidable for the architecture of Figure 1 and external specifications, that is, if we do not constrain the internal variable t.

Results. 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 [24, 15].

To rule out unnatural properties yielding undecidability, the specifications we consider are external, coming back to the original framework of [24, 6]. In Section 3, we first determine a sufficient condition for undecidability with external specifications, that generalizes the undecidability result of [24]. We next introduce in Section 4 *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, synthesis may be undecidable for full specifications while decidable for external ones.) We also propose a natural restriction on specifications for which synthesis, on UWC architectures, becomes decidable. We call such specifications *robust specifications*. Finally, we introduce in Section 5 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 a necessary condition 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. It remains open whether the problem is decidable for robust external specifications and well-connected architectures.

An extended abstract of this work appeared in [11].

2 Preliminaries

Trees and tree automata. Given two finite sets X and Y, a Y-labeled X-tree (also called full tree) is a total function $t : X^* \to Y$ 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 $t : X^* \to Y$ 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 $\delta : Q \times Y \to 2^{Q^X}$ 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 $\rho(\varepsilon) = q_0$, and for all $\sigma \in X^*$, $(\rho(\sigma \cdot x))_{x\in X} \in \delta(\rho(\sigma),t(\sigma))$. The run ρ 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* $\mathscr{A} = (V \uplus P, E, (S^v)_{v\in V}, s_0, (d_p)_{p\in P})$ 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. An example of an architecture is given in Figure 2, where processes are represented by boxes and variables by circles. Input and output variables are defined, respectively, by

$$V_{\mathrm{I}} = \{v \in V \mid E^{-1}(v) = \emptyset\},$$
$$V_{\mathrm{O}} = \{v \in V \mid E(v) = \emptyset\}.$$

Variables in $V \setminus (V_{\mathrm{I}} \cup V_{\mathrm{O}})$ 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 $U \subseteq V$, we denote by S^U the set $\prod_{v\in U} S^v$. 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 $v \in V \setminus V_{\mathrm{I}}$, we will often write d_v for d_p where $E^{-1}(v) = \{p\}$.

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 $s_0 \in S^V$ given by the architecture. If $\sigma = s_0 s_1 s_2 \cdots \in (S^V)^\omega$ is a run, then its projection on $U \subseteq V$ is $\sigma^U = s_0^U s_1^U s_2^U \cdots$. Also, we denote by $\sigma[i,j]$ the factor $s_i \ldots s_j$ 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 $t : (S^{V_{\mathrm{I}}})^* \to S^V$, where

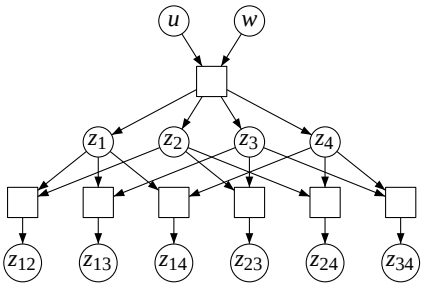

Fig. 2. An architecture with input variables u, w and output variables z_{ij}.

$t(\varepsilon) = s_0$ and for $\rho \in (S^{V_\mathrm{I}})^*$, $r \in S^{V_\mathrm{I}}$, we have $t(\rho \cdot r)^{V_\mathrm{I}} = r$. The projection of t on $U \subseteq V$ is the tree $t^U : (S^{V_\mathrm{I}})^* \to S^U$ 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 $\mathscr{L}(U)$ where $\mathscr{L}$ is the logic used. Specifications over U are *external* if $U \subseteq V_\mathrm{I} \cup V_\mathrm{O}$. The validity of an external formula on a run tree t (or simply a run) only depends on its projection $t^{V_\mathrm{I} \cup V_\mathrm{O}}$ onto $V_\mathrm{I} \cup V_\mathrm{O}$.

Programs, strategies. We consider a discrete time, synchronous semantics. Informally, at step $i = 1, 2, \ldots$, 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 $v \in V \setminus V_\mathrm{I}$ 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) advises 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 $f^v : \left(S^{R(v)}\right)^+ \to S^v$ 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 $F = (f^v)_{v \in V \setminus V_\mathrm{I}}$ of local strategies. A run $\sigma = s_0 s_1 s_2 \cdots \in (S^V)^\omega$ is an *F-run* (or is *F-compatible*) if for all $v \in V \setminus V_\mathrm{I}$ and all $i > 0$, $s_i^v = f^v(\sigma^{R(v)}[i])$. Given an input sequence $\rho \in (S^{V_\mathrm{I}})^\omega$, there is a unique run $\sigma \in (S^V)^\omega$ which is F-compatible and such that $\sigma^{V_\mathrm{I}} = \rho$.

The *F-run tree* is the run tree $t : (S^{V_\mathrm{I}})^* \to S^V$ such that each branch is labeled by a word $s_0 s_1 s_2 \cdots \in (S^V)^\omega$ 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 $\sigma^{V_\mathrm{I}}[i]$. This shows that the F-run tree is unique.

Distributed synthesis problem. Let $\mathscr{L}$ be a specification language. The distributed synthesis problem for an architecture $\mathscr{A}$ is the following: given a formula $\varphi \in \mathscr{L}$, decide whether there exists a distributed program F on $\mathscr{A}$ 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 $\mathscr{L}$).

Memoryless strategies. The strategy f^v is *memoryless* if it does not depend on the past, that is, if there exists $g : S^{R(v)} \to S^v$ such that $f^v(s_0 \cdots s_i \cdots s_{i+d_v}) = g(s_i)$ for $s_0 \cdots s_{i+d_v} \in (S^{R(v)})^+$. In case $d_v = 0$, this corresponds to the usual definition of a memoryless strategy.

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

Smallest cumulative delay. Throughout the paper, the notion of *smallest cumulative delay* of transmission from u to v will extensively be used. It is defined by

$$
\begin{aligned}
&d(u,u) = 0\\
&d(u,v) = +\infty \text{ if } v \notin (E^2)^+(u), \text{ i.e., if there is no path from } u \text{ to } v\\
&d(u,v) = d_v + \min\{d(u,w) \mid w \in R(v) \text{ and } w \in (E^2)^+(u)\}
\end{aligned}
$$

d-compatibility for summaries. The compatibility of the strategies $F = (f^v)_{v \in V \setminus V_\mathrm{I}}$ with the delays extends to the summaries $\hat{F} = (\hat{f}^v)_{v \in V \setminus V_\mathrm{I}}$. Formally, a map $h : (S^{\mathrm{View}(v)})^+ \to S^v$ is *d-compatible* (or compatible with the delays $(d_v)_{v \in V \setminus V_\mathrm{I}}$) if for all $\rho \in (S^{\mathrm{View}(v)})^i$, $h(\rho)$ only depends on the prefixes $(\rho^u[i - d(u,v)])_{u \in \mathrm{View}(v)}$.

3 Architectures with incomparable information

In this section, we state a sufficient condition for undecidability; this relies on an easy generalization of the undecidable architecture presented in [24].

Definition 1. *An architecture has* incomparable information *if there exist variables $x, y \in V_O$ such that $\mathit{View}(x) \setminus \mathit{View}(y) \neq \emptyset$ and $\mathit{View}(y) \setminus \mathit{View}(x) \neq \emptyset$. Otherwise the architecture has* linearly preordered information.

For instance, the architectures of Figures 1, 2, 5 and 6 have linearly preordered information, while the architecture $\mathscr{A}'$ of Figure 3 has incomparable information. The following proposition extends the undecidability result of [24, 8].

Proposition 2. *Architectures with incomparable information are undecidable for LTL or CTL external specifications.*

In [24], the architecture $\mathscr{A}'$ shown in Figure 3 is proved undecidable, both for LTL and CTL specifications. We will reduce the synthesis problem of $\mathscr{A}'$ to the synthesis problem of an architecture with incomparable information. This reduction is rather

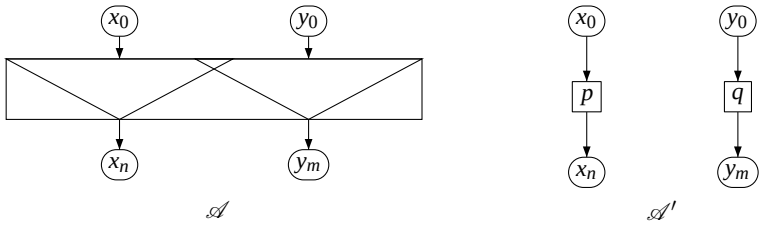

Fig. 3. Architectures $\mathscr{A}$ and $\mathscr{A}'$

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 $\mathscr{A} = (P \uplus V, E, (S^v)_{v\in V}, s_0, (d_p)_{p\in P})$ be an architecture with incomparable information. Without loss of generality, we assume that $s_0^v = 0$ for all $v \in V$. By definition, we find $x_0, y_0 \in V_\mathrm{I}$ and $x_n, y_m \in V_\mathrm{O}$ such that $x_0 \notin \mathrm{View}(y_m)$ and $y_0 \notin \mathrm{View}(x_n)$. Consider paths $x_0 E^2 x_1 E^2 \dots E^2 x_n$ from x_0 to x_n, and $y_0 E^2 y_1 E^2 \dots E^2 y_m$ from y_0 to y_m such that $d(x_0, x_n) = d_{x_1} + \dots + d_{x_n}$ and $d(y_0, y_m) = d_{y_1} + \dots + d_{y_m}$. Note that the sets of variables $\{x_0, \dots, x_n\}$ and $\{y_0, \dots, y_m\}$ are disjoint.

Let $\mathscr{A}' = (P' \uplus V', E', (S'^v)_{v\in V'}, s_0', (d_p')_{p\in P'})$ be the architecture of Figure 3 with $V_\mathrm{I}' = \{x_0, y_0\}$, $V_\mathrm{O}' = \{x_n, y_m\}$; with unchanged domains for output variables: $S'^{x_n} = S^{x_n}$ and $S'^{y_m} = S^{y_m}$; with $S'^{x_0} = S'^{y_0} = \{0,1\}$ as domain for input variables; and with $s_0' = s_0^{V'}$. The delays for x_n and y_m are the smallest cumulative delays of transmission from x_0 to x_n and y_0 to y_m as defined earlier: $d'_{x_n} = d'_p = d(x_0, x_n)$ and $d'_{y_m} = d'_q = d(y_0, y_m)$.

The architecture $\mathscr{A}'$ is undecidable for LTL or CTL specifications (it suffices to adapt the proofs of [24, 8] taking into account different delays on processes). We reduce the distributed synthesis problem for $\mathscr{A}'$ to the same problem for $\mathscr{A}$. We first consider CTL specifications.

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

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

is not implementable over $\mathscr{A}'$ whereas it is implementable over $\mathscr{A}$, provided $\mathrm{View}(x_n) \setminus \{x_0\} \neq \emptyset$ and assuming no delays.

To define an implementation F' over $\mathscr{A}'$ given an implementation F over $\mathscr{A}$, we simulate the behavior of F when all variables in $V_\mathrm{I} \setminus V_\mathrm{I}'$ are constantly set to 0. This will be enforced when defining the reduction of the specification from $\mathscr{A}'$ to $\mathscr{A}$, using the formula $\chi = (x_0 \in \{0,1\}) \wedge (y_0 \in \{0,1\}) \wedge \bigwedge_{v\in V_\mathrm{I}\setminus V_\mathrm{I}'} (v = 0)$. We define a reduction that maps a formula ψ of $\mathrm{CTL}(V')$ into a formula $\overline{\psi}$ in $\mathrm{CTL}(V_\mathrm{I} \cup V_\mathrm{O})$ that ensures ψ only on the subtree of executions respecting χ. This reduction is defined by

$$\overline{(x = s)} = (x = s) \qquad \overline{\neg\psi} = \neg\overline{\psi}$$
$$\overline{\varphi \vee \psi} = \overline{\varphi} \vee \overline{\psi} \qquad \overline{\mathsf{EX}\,\psi} = \mathsf{EX}(\chi \wedge \overline{\psi})$$
$$\overline{\mathsf{E}\,\varphi\,\mathsf{U}\,\psi} = \mathsf{E}(\chi \wedge \overline{\varphi})\,\mathsf{U}\,(\chi \wedge \overline{\psi}) \qquad \overline{\mathsf{EG}\,\psi} = \mathsf{EG}(\chi \wedge \overline{\psi}).$$

We use the following notation: for $r \in S'^{V_I'}$, we define $\overline{r} \in S^{V_I}$ by $\overline{r}^{V_I'} = r$ and $\overline{r}^v = 0$ for all $v \in V_I \setminus V_I'$, and we extend this definition to words (with $\overline{\varepsilon} = \varepsilon$). This allows us to fix the run tree $\tilde{t} : (S'^{V_I'})^* \to S'^{V'}$ over $\mathscr{A}'$ that corresponds to a run tree $t : (S^{V_I})^* \to S^V$ over $\mathscr{A}$: $\tilde{t}(\rho) = t(\overline{\rho})^{V'}$ for $\rho \in (S'^{V_I'})^*$. The reduction of the formula is correct in the following sense:

Lemma 3. *For every formula $\psi \in CTL(V')$, every tree $t : (S^{V_I})^* \to S^V$, and every $\rho \in (S'^{V_I'})^*$ we have $t, \overline{\rho} \models \overline{\psi}$ if and only if $\tilde{t}, \rho \models \psi$.*

Proof. By an easy induction on ψ. Let $t : (S^{V_I})^* \to S^V$ and $\rho \in (S'^{V_I'})^*$. When $\psi = (x = s)$ for some $x \in V'$ and $s \in S^x$, the result follows from $\tilde{t}(\rho)^x = t(\overline{\rho})^x$. The cases of boolean connectives are trivial. So let $\psi = \mathsf{E}\,\psi_1 \,\mathsf{U}\, \psi_2$ and assume that $t, \overline{\rho} \models \overline{\psi}$. Then we find $s_1 \cdots s_n \in (S^{V_I})^*$ such that $t, \overline{\rho} \cdot s_1 \cdots s_n \models \chi \wedge \overline{\psi_2}$ and $t, \overline{\rho} \cdot s_1 \cdots s_i \models \chi \wedge \overline{\psi_1}$ for all $0 \leq i < n$. Since $s_i \models \chi$, we deduce that $s_i = \overline{r_i}$ for $r_i = s_i^{V_I'} \in S'^{V_I'}$. By induction we obtain $\tilde{t}, \rho \cdot r_1 \cdots r_n \models \psi_2$ and $\tilde{t}, \rho \cdot r_1 \cdots r_i \models \psi_1$ for all $0 \leq i < n$. Therefore, $\tilde{t}, \rho \models \psi$. The converse implication can be shown similarly.

The cases EX and EG are left to the reader. □

Now we prove the reduction:

Lemma 4. *If there is a distributed program F' over $\mathscr{A}'$ that satisfies ψ, then there is a distributed program F over $\mathscr{A}$ that satisfies $\overline{\psi}$.*

Proof. Let $F' = (f'^{x_n}, f'^{y_m})$ be a distributed implementation for ψ over $\mathscr{A}'$. We will define a distributed strategy $F = (f^v)_{v \in V}$ for $\overline{\psi}$ over $\mathscr{A}$ so that the projection on V' of any F-run will be an F'-run. More precisely, if $\sigma \in (S^V)^+$ is a prefix of an F-run with $\sigma^{x_0} \in \{0,1\}^+$, then the following will hold:

$$f^{x_n}(\sigma^{R(x_n)}) = f'^{x_n}(\sigma^{x_0}) \tag{1}$$

and similarly for (y_0, y_m).

To do so, we use the variables $x_1, \ldots, x_{n-1}$ to transmit the value of x_0 through the architecture. Formally, at each step, f^{x_k} copies the last value of x_{k-1} it can read – the one that was written d_{x_k} steps before: for $0 < k < n$ and $\tau \in (S^{R(x_k)})^+$, we define

$$f^{x_k}(\tau) = \begin{cases} s^{x_{k-1}} & \text{if } \tau = \tau_1 s \tau_2 \text{ with } |\tau_2| = d_{x_k} \text{ and } s^{x_{k-1}} \in \{0,1\}, \\ 0 & \text{otherwise.} \end{cases}$$

By definition, f^{x_k} is clearly compatible with the delay d_{x_k}. It is easy to check that if we provide $\rho \in \{0,1\}^\omega$ as input on x_0 and follow the strategies (f^{x_k}) above then we get on x_{n-1} the outcome $0^{d(x_0, x_{n-1})}\rho$ corresponding to the shift by $d(x_0, x_{n-1}) = d'_{x_n} - d_{x_n}$.

In order to satisfy (1), the last strategy f^{x_n} simulates f'^{x_n} taking into account the shift by $d'_{x_n} - d_{x_n}$ of their respective inputs. To explain the definition of f^{x_n}, consider $\sigma \in (S^V)^+$ compatible with $(f^{x_k})_{0<k<n}$ and such that $\sigma^{x_0} \in \{0,1\}^+$. If $|\sigma| \leq d'_{x_n}$ then $f'^{x_n}(\sigma^{x_0})$ does not depend on σ so we define $f^{x_n}(\sigma^{R(x_n)}) = f'^{x_n}(0^{|\sigma|}) = f'^{x_n}(\sigma^{x_0})$ so that (1) holds. Assume now that $\sigma = \sigma_1\sigma_2\sigma_3$ with $|\sigma_1| = d'_{x_n} - d_{x_n}$, $|\sigma_3| = d_{x_n}$. Note that

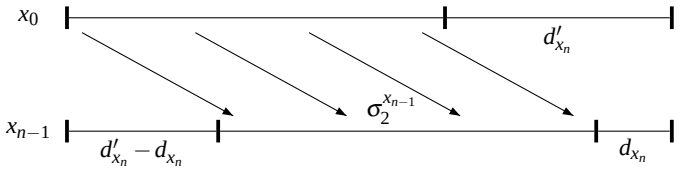

Fig. 4. Simulation of f'^{x_n} by f^{x_n}.

$f'^{x_n}(\sigma^{x_0})$ only depends on the prefix of σ^{x_0} of length $|\sigma_2|$ which is precisely $\sigma_2^{x_{n-1}}$ due to the shift induced by the strategies $(f^{x_k})_{0<k<n}$ (see Figure 4). Hence, in this case, we define $f^{x_n}(\sigma^{R(x_n)}) = f'^{x_n}(\sigma_2^{x_{n-1}} 0^{d'_{x_n}}) = f'^{x_n}(\sigma^{x_0})$ in order to get (1) again. The formal definition of f^{x_n} is given for arbitrary $\tau \in (S^{R(x_n)})^+$ by

$$f^{x_n}(\tau) = \begin{cases} f'^{x_n}(0^{|\tau|}) & \text{if } |\tau| \leq d'_{x_n}, \\ f'^{x_n}(\tau_2^{x_{n-1}} 0^{d'_{x_n}}) & \text{if } \tau = \tau_1\tau_2\tau_3 \text{ with } |\tau_1| = d'_{x_n} - d_{x_n}, |\tau_3| = d_{x_n} \\ & \qquad\qquad \text{and } \tau_2^{x_{n-1}} \in \{0,1\}^+, \\ 0 & \text{otherwise.} \end{cases}$$

By definition, f^{x_n} is clearly compatible with the delay d_{x_n}. Also, we have explained that (1) holds with this definition of $(f^{x_k})_{0<k\leq n}$. For $0 < k \leq m$, we define similarly f^{y_k} and for every other variable v, we set $f^v = 0$. The resulting distributed strategy $F = (f^v)_{v\in V}$ is indeed compatible with the delays. It remains to show that F is a distributed implementation for $\overline{\psi}$ over $\mathscr{A}$.

Let $t : (S^{V_1})^* \to S^V$ be the F-run tree over $\mathscr{A}$. We show below that $\tilde{t} : (S'^{V_1'})^* \to S^{V'}$ is in fact the F'-run tree over $\mathscr{A}'$. Then, since F' is a distributed implementation of ψ, we deduce $\tilde{t}, \varepsilon \models \psi$, and Lemma 3 implies $t, \varepsilon \models \overline{\psi}$. Hence F is a distributed implementation of $\overline{\psi}$.

First, it is easy to see that $\tilde{t}$ is a run-tree over $\mathscr{A}'$: $\tilde{t}(\varepsilon) = t(\bar{\varepsilon})^{V'} = s_0^{V'} = s'_0$, and for $\rho \in (S'^{V_1'})^*$ and $r \in S'^{V_1'}$ we have $\tilde{t}(\rho \cdot r)^{V_1'} = t(\bar{\rho} \cdot \bar{r})^{V_1'} = \bar{r}^{V_1'} = r^{V_1'}$. Next, to show that $\tilde{t}$ is the F'-run tree, we have to check that $\tilde{t}(\rho)^{x_n} = f'^{x_n}(\rho^{x_0})$ for each $\rho = r_1 \cdots r_i \in (S^{V_1'})^+$ and similarly for (y_0, y_m). Let $\sigma \in (S^V)^+$ be the F-run induced by $\bar{\rho}$: $\sigma = t(\varepsilon)t(\bar{r}_1)t(\bar{r}_1\bar{r}_2)\cdots t(\bar{\rho})$. Using (1) we obtain $\tilde{t}(\rho)^{x_n} = t(\bar{\rho})^{x_n} = f^{x_n}(\sigma^{R(x_n)}) = f'^{x_n}(\sigma^{x_0}) = f'^{x_n}(\rho^{x_0})$. Using the same arguments, we also obtain that $\tilde{t}(\rho)^{y_m} = f'^{y_m}(\rho^{y_0})$, and that $\tilde{t}$ is the F'-run tree. □

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

Proof. Suppose $F = (f^v)_{v\in V\setminus V_1}$ is a distributed implementation of $\overline{\psi}$ over $\mathscr{A}$. We need to define the strategies $f'^{x_n} : (S'^{x_0})^+ \to S^{x_n}$ and $f'^{y_m} : (S'^{y_0})^+ \to S^{y_m}$ of the variables in $\mathscr{A}'$. The difficulty here is that f'^{x_n} may have less input variables than f^{x_n} so it cannot simply simulate it. To overcome this, we use the fact that, due to the special form of $\overline{\psi}$, the F-run tree t satisfies $\overline{\psi}$ if and only if the sub-tree restricted to branches where all input variables other than x_0 and y_0 are always 0 also satisfies $\overline{\psi}$. So the processes of

$\mathscr{A}'$ will behave like the processes of $\mathscr{A}$ writing respectively on x_n and y_m in the special executions when the values of input variables other than x_0 and y_0 are always 0.

Formally, for $\rho \in (S'^{V_I'})^+$, we set $f'^{x_n}(\rho^{x_0}) = \hat{f}^{x_n}(\overline{\rho}^{\mathrm{View}(x_n)})$. Observe that, due to incomparable information, $\hat{f}^{x_n}$ does not depend on $\overline{\rho}^{y_0}$. Hence f'^{x_n} only depends on ρ^{x_0} and is a correct strategy for variable x_n in the architecture $\mathscr{A}'$. Moreover, $\hat{f}^{x_n}$ is d-compatible and so f'^{x_n} is d'-compatible. We define f'^{y_m} similarly. It is easy to check that $F' = (f'^{x_n}, f'^{y_m})$ is a distributed implementation of ψ over $\mathscr{A}'$: let t be the F-run tree and t' be the F'-run tree. We have $t'(\rho)^{x_n} = f'^{x_n}(\rho^{x_0}) = \hat{f}^{x_n}(\overline{\rho}^{\mathrm{View}(x_n)}) = t(\overline{\rho})^{x_n} = \tilde{t}(\rho)^{x_n}$ and similarly, $t'(\rho)^{y_m} = \tilde{t}(\rho)^{y_m}$. Hence $t' = \tilde{t}$ and since $t, \varepsilon \models \overline{\psi}$, Lemma 3 implies that $\tilde{t}, \varepsilon \models \psi$ and F' is a distributed implementation of ψ on $\mathscr{A}'$. □

We consider the reduction for LTL specifications. In this case, the specification over $\mathscr{A}$ only needs to ensure ψ when the input values on x_0 and y_0 are in the domain allowed by $\mathscr{A}'$. We use the reduction

$$\overline{\psi} = (\mathsf{G}\,\xi) \rightarrow \psi$$

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 4 and Lemma 5 yield the reduction. Indeed, let F' be a distributed implementation of ψ over $\mathscr{A}'$, and let F be defined as in the proof of Lemma 4. Let $\rho \in (S^{V_I})^\omega$ be an input sequence and $\sigma = s_0 s_1 s_2 \cdots \in (S^V)^\omega$ be the induced F-run. If $\rho^{x_0} \notin \{0,1\}^\omega$ or $\rho^{y_0} \notin \{0,1\}^\omega$ then $\sigma, \varepsilon \not\models \mathsf{G}\,\xi$. Otherwise, by equation (1), we get for all $i > 0$, $s_i^{x_n} = f^{x_n}(\sigma^{R(x_n)}[i]) = f'^{x_n}(\sigma^{x_0}[i])$ and $s_i^{y_m} = f^{y_m}(\sigma^{R(y_m)}[i]) = f'^{y_m}(\sigma^{y_0}[i])$. Then $\sigma^{V'}$ is an F'-run, and $\sigma^{V'}, \varepsilon \models \psi$. Since $\psi \in \mathrm{LTL}(V')$ we deduce $\sigma, \varepsilon \models \psi$. We obtain that any F-run σ is such that $\sigma, \varepsilon \models (\mathsf{G}\,\xi) \rightarrow \psi$, and F is a distributed implementation of $\overline{\psi}$ over $\mathscr{A}$.

Conversely, given F a distributed implementation of $\overline{\psi}$ over $\mathscr{A}$, define F' as in the proof of Lemma 5. Let $\rho \in (S'^{V_I'})^\omega$ be an input sequence and $\sigma = s_0 s_1 s_2 \cdots \in (S^V)^\omega$ be the F-run induced by $\overline{\rho}$. By definition of $\overline{\rho}$, we have $\sigma, \varepsilon \models \mathsf{G}\,\xi$ and since F is a distributed implementation of $\overline{\psi}$ we get $\sigma, \varepsilon \models \psi$. Again, $\psi \in \mathrm{LTL}(V')$ implies that $\sigma^{V'}, \varepsilon \models \psi$. Given that $\sigma^{V'}$ is in fact the F'-run induced by ρ (this is immediate from the definition of f'^{x_n} and f'^{y_m}), F' is a distributed implementation of ψ over $\mathscr{A}'$.

We have defined a reduction from the distributed synthesis problem over the architecture $\mathscr{A}'$ to the distributed synthesis problem over an architecture with incomparable information, for LTL or CTL specifications. Since the synthesis problem is undecidable both for LTL and CTL specifications over $\mathscr{A}'$, we obtain its 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* specification and shows that UWC architectures are always decidable for external and robust specifications.

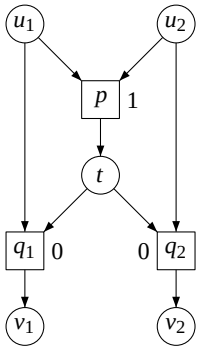

Fig. 5. A uniformly well-connected architecture

4.1 Definition

A *routing* for an architecture $\mathscr{A} = (V \uplus P, E, (S^v)_{v\in V}, s_0, (d_p)_{p\in P})$ is a family of *memoryless* local strategies $\Phi = (f^v)_{v\in V\setminus(V_I\cup V_O)}$. 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, all the values of the variables in View(v).

Definition 6. *An architecture $\mathscr{A}$ is* uniformly well-connected *(UWC) if there exist a routing Φ and, for every $v \in V_O$ and $u \in View(v)$, a decoding function $g^{u,v} : \left(S^{R(v)}\right)^+ \to S^u$ that can reconstruct the value of u, i.e., such that for any Φ-compatible sequence $\sigma = s_0 s_1 s_2 \cdots \in \left(S^{V\setminus V_O}\right)^+$, we have for $i \geq 0$*

$$s_i^u = g^{u,v}(\sigma^{R(v)}[i + d(u,v) - d_v]) \tag{2}$$

In case there is no delay, the uniform well-connectedness refines the notion of adequate connectivity introduced by Pnueli and Rosner in [24], as we no longer require each output variable to be communicated the value of *all* input variables, but only of those belonging to 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 5. 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 $t = \mathsf{Y}u_1 \oplus \mathsf{Y}u_2$ where $\mathsf{Y}x$ denotes the previous value of variable x. In order to recover (decode) $\mathsf{Y}u_2$, process q_1 memorizes the previous value of u_1 and makes the xor with t: $\mathsf{Y}u_2 = t \oplus \mathsf{Y}u_1$. But if we restrict to memoryless decoding functions, then we only know u_1 and t and we cannot recover $\mathsf{Y}u_2$.

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 7. *Let $\mathscr{A} = (V \uplus P, E, (S^v)_{v\in V}, s_0, (d_p)_{p\in P})$ be a UWC architecture. For each $v \in V_O$, let $h^v : (S^{View(v)})^+ \to S^v$ 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 $\mathscr{A}$ 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 $(g^{u,v})_{v\in V_O, u\in \mathrm{View}(v)}$ be respectively the routing and the decoding functions giving the uniform well-connectedness of the architecture $\mathscr{A}$. 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 $f^v(\sigma^{R(v)}) = h^v(\rho^{\mathrm{View}(v)})$. 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 $u \in \mathrm{View}(v)$, $\rho^u[i-d(u,v)] = \rho'^u[i-d(u,v)]$. Indeed, for all $0 \le j \le i-d(u,v)$, we have $s^u_j = g^{u,v}(\sigma^{R(v)}[j+d(u,v)-d_v])$ and $s'^u_j = g^{u,v}(\sigma'^{R(v)}[j+d(u,v)-d_v])$ by (2). Using $\sigma^{R(v)}[i-d_v] = \sigma'^{R(v)}[i-d_v]$ and $j+d(u,v) \le i$ we get $s^u_j = s'^u_j$ as desired. Since h^v is d-compatible, we deduce that $h^v(\rho^{\mathrm{View}(v)}) = h^v(\rho'^{\mathrm{View}(v)})$.

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

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

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

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

Theorem 8. *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 $\mathscr{A}$ is UWC and has linearly preordered information, and therefore we can order the output variables $V_O = \{v_1, \ldots, v_n\}$ so that $\mathrm{View}(v_n) \subseteq \cdots \subseteq \mathrm{View}(v_1) \subseteq V_I$.

In the following, in order to use tree-automata, we extend a local strategy $f : (S^X)^+ \to S^Y$ by letting $f(\varepsilon) = s^Y_0$, 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 $\mathscr{A}$ is an $(S^{\mathrm{View}(v_1)}, S^{V_O})$-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 $\mathrm{View}(v_1) \subsetneq V_I$). This problem was solved in [13] for CTL* specifications.

Proposition 9 ([13, Th. 4.4]). *Given an external specification $\varphi \in CTL^*(V_I \cup V_O)$, one can build a non-deterministic tree automaton (NDTA) $\mathfrak{A}_1$ over $(S^{View(v_1)}, S^{V_O})$-trees such that $h \in \mathscr{L}(\mathfrak{A}_1)$ if and only if the run tree induced by h satisfies φ.*

If $\mathscr{L}(\mathfrak{A}_1)$ is empty then we already know that there are no distributed implementations for the specification φ over $\mathscr{A}$. Otherwise, thanks to Lemma 7, we have to check whether for each $v \in V_O$ there exists an $(S^{\mathrm{View}(v)}, S^v)$-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 $v_1, \ldots, v_n$. 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 $V_k = \{v_k, \ldots, v_n\}$ for $k \geq 1$. The induction step relies on the following statement.

Proposition 10. *Let $1 \leq k < n$. Given a NDTA $\mathfrak{A}_k$ accepting $(S^{View(v_k)}, S^{V_k})$-trees, one can build a NDTA $\mathfrak{A}_{k+1}$ accepting $(S^{View(v_{k+1})}, S^{V_{k+1}})$-trees, such that a tree t is accepted by $\mathfrak{A}_{k+1}$ if and only if there exists an $(S^{View(v_k)}, S^{v_k})$-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 10 is split in two steps. Since $V_k = \{v_k\} \uplus V_{k+1}$, we have $t = t^{v_k} \oplus t^{V_{k+1}}$ for each $(S^{\mathrm{View}(v_k)}, S^{V_k})$-tree t (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 $t \in \mathscr{L}(\mathfrak{A}_k)$ such that t^{v_k} is d-compatible (Lemma 11). Then, one can build an automaton that restricts the domain of the directions and the labeling of the accepted trees to $S^{\mathrm{View}(v_{k+1})}$ and $S^{V_{k+1}}$ respectively.

Lemma 11. *Let $v \in U \subseteq V_O$. Given a NDTA $\mathfrak{A}$ over $(S^{View(v)}, S^U)$-trees one can build a NDTA $\mathfrak{A}' = \mathrm{compat}_v(\mathfrak{A})$ also over $(S^{View(v)}, S^U)$-trees such that $\mathscr{L}(\mathfrak{A}') = \{t \in \mathscr{L}(\mathfrak{A}) \mid t^v$ is d-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 $K = \max\{d(u,v), u \in \mathrm{View}(v)\}$ steps in advance and consists in a d-compatible function $g : (S^{\mathrm{View}(v)})^K \to S^v$ that predicts what will be K steps later the values of variable v. During a transition, the guess is sent in each direction $r \in S^{\mathrm{View}(v)}$ 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 $Q' = Q \times \mathscr{F}$ where $\mathscr{F}$ is the set of *d-compatible* functions $f : (S^{\mathrm{View}(v)})^{<K} \to S^v$, 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.

To formalize this, we define the (transition) function $\Delta : \mathscr{F} \times S^{\mathrm{View}(v)} \to 2^{\mathscr{F}}$ by

$$\Delta(f,r) = \{ f' \mid f'(\sigma) = f(r\sigma) \text{ for } |\sigma| < K-1 \} .$$

Intuitively, if we are in state $(q,f) \in Q \times \mathscr{F}$ at some node τ and move in direction $r \in S^{\mathrm{View}(v)}$ then $\Delta(f,r)$ computes the set of functions in $\mathscr{F}$ that could label the node $\tau \cdot r$. Observe that f' is determined by f and r for any σ such that $|\sigma| < K-1$ and corresponds to the specialization of f according to the new direction r. The functions $f' \in \Delta(f,r)$ differ only on values $f'(\sigma)$ for $|\sigma| = K-1$ which correspond to the new guesses.

Now, the transition function of $\mathfrak{A}'$ is defined for $(q,f) \in Q'$ and $s \in S^U$ only if $s^v = f(\varepsilon)$ (this ensures that the guess made K steps earlier was correct) and sends in each direction $r \in S^{View(v)}$ of the tree a copy of the automaton in the state (q_r, g_r) where q_r corresponds to the simulation of a run of $\mathscr{A}$ and $g_r \in \Delta(f,r)$. Formally, if $s^v = f(\varepsilon)$ then

$$\delta'\big((q,f),s\big) = \left\{ (q_r,g_r)_{r \in S^{\mathrm{View}(v)}} \,\middle|\, \begin{array}{c} (q_r)_{r \in S^{\mathrm{View}(v)}} \in \delta(q,s) \text{ and} \\ g_r \in \Delta(f,r) \text{ for all } r \in S^{\mathrm{View}(v)} \end{array} \right\}.$$

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

Let t be an $(S^{\mathrm{View}(v)}, S^U)$-tree accepted by $\mathfrak{A}$ and suppose that t^v is d-compatible. Let $\rho : (S^{\mathrm{View}(v)})^* \to Q$ be an accepting run of $\mathfrak{A}$ over t. There is a unique way to extend ρ to a run $\rho' : (S^{\mathrm{View}(v)})^* \to Q \times \mathscr{F}$ of $\mathfrak{A}'$ over t. The only possibility is to label a node $\tau \in (S^{\mathrm{View}(v)})^*$ by the map $f_\tau : (S^{\mathrm{View}(v)})^{<K} \to S^v$ defined by $f_\tau(\sigma) = t^v(\tau\sigma)$ for $\sigma \in (S^{View(v)})^{<K}$ so that all guesses are correct. Since t^v is d-compatible, we deduce that f_τ is also d-compatible, hence it belongs to $\mathscr{F}$. Then we can define the run ρ' by $\rho'(\tau) = \big(\rho(\tau), f_\tau\big)$ for $\tau \in (S^{\mathrm{View}(v)})^*$. We show that it is an accepting run of $\mathscr{A}'$ over t. First, we prove that at each node $\tau \in (S^{\mathrm{View}(v)})^*$ the transition function δ' is satisfied. Let $(q, f_\tau) = \rho'(\tau)$ and $(q_r, f_{\tau r}) = \rho'(\tau r)$ for all $r \in S^{\mathrm{View}(v)}$. By definition, $f_\tau(\varepsilon) = t^v(\tau)$ and $\delta'((q_r, f_{\tau r}), t(\tau))$ is defined. Now, since $\pi(\rho') = \rho$ which is a run of $\mathfrak{A}$ over t we have $(q_r)_{r \in S^{\mathrm{View}(v)}} \in \delta(q, t(\tau))$. It remains to show that $f_{\tau r} \in \Delta(f_\tau, r)$ for all $r \in S^{\mathrm{View}(v)}$. This is obvious from the definitions: $f_{\tau r}(\sigma) = t^v(\tau r \sigma) = f_\tau(r\sigma)$ for $\sigma \in (S^{View(v)})^{<K-1}$. Finally, the run ρ' is successful since its projection on Q is ρ which is successful.

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

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

Let $\tau, \tau' \in (S^{\mathrm{View}(v)})^i$ be such that $\tau^u[i - d(u,v)] = \tau'^u[i - d(u,v)]$ for all $u \in \mathrm{View}(v)$. We have to show $H(\tau)(\varepsilon) = H(\tau')(\varepsilon)$.

If $|\tau| = |\tau'| > K$ then we show that τ, τ' necessarily share a common prefix. More precisely, since $K \geq d(u,v)$ for all $u \in \text{View}(v)$, we deduce from the equalities $\tau^u[i - d(u,v)] = \tau'^u[i - d(u,v)]$ for all $u \in \text{View}(v)$ that $\tau = \tau_1\tau_2$, and $\tau' = \tau_1\tau_2'$ with $|\tau_2| = |\tau_2'| = K$ and $\tau_2^u[K - d(u,v)] = \tau_2'^u[K - d(u,v)]$ for all $u \in \text{View}(v)$. We can show, by successive applications of the transition function δ' and by definition of Δ, that the value of $H(\tau_1\tau_2)(\varepsilon)$ is indeed the guess made at node τ_1 for the direction defined by τ_2, i.e., $H(\tau_1\tau_2)(\varepsilon) = H(\tau_1)(\tau_2)$. Similarly, we obtain $H(\tau_1\tau_2')(\varepsilon) = H(\tau_1)(\tau_2')$. Since $H(\tau_1) \in \mathscr{F}$, it is d-compatible. Using $\tau_2^u[K - d(u,v)] = \tau_2'^u[K - d(u,v)]$ for all $u \in \text{View}(v)$, we deduce $H(\tau_1)(\tau_2) = H(\tau_1)(\tau_2')$. Therefore, $H(\tau)(\varepsilon) = H(\tau')(\varepsilon)$.

If $|\tau| < K$, then we obtain similarly that $H(\tau)(\varepsilon) = H(\varepsilon)(\tau) = H(\varepsilon)(\tau') = H(\tau')(\varepsilon)$ since $H(\varepsilon) \in \mathscr{F}$ is d-compatible. □

Proof (of Proposition 10). 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 10 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 [20]. The drawback is that this involves an exponential blow up. Unfortunately, this is needed since Lemma 11 requires a NDTA as input. □

We can now conclude the proof of Theorem 8. Using Proposition 10 inductively starting from the NDTA $\mathfrak{A}_1$ of Proposition 9, 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 $h^{v_1} \oplus \cdots \oplus h^{v_n}$ is accepted by $\mathfrak{A}_1$. Therefore, using Lemma 7, there is a distributed implementation for the specification over $\mathscr{A}$ if and only if $\mathscr{L}(\text{compat}_{v_n}(\mathfrak{A}_n))$ is nonempty. The overall procedure is non-elementary due to the exponential blow-up of the inductive step in Proposition 10. We do not know for now the lower bound of the complexity of this problem. □

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

Proposition 13. *The synthesis problem for robust CTL^* specifications is decidable over UWC architectures.*

Proof. Let $\mathscr{A} = (V \uplus P, E, (S^u)_{u\in V}, s_0, (d_p)_{p\in P})$ 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 $\varphi_v \in \mathrm{CTL}^*(\mathrm{View}(v) \cup \{v\})$. Using Proposition 9, for each $v \in V_O$ we find a NDTA $\mathfrak{A}_v$ accepting a strategy $h : (S^{\mathrm{View}(v)})^* \to S^v$ if and only if the induced run tree $t : (S^{\mathrm{View}(v)})^* \to S^{\mathrm{View}(v)\cup\{v\}}$ satisfies φ_v. The proposition then follows from the

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

First, let F be a distributed implementation of φ over $\mathscr{A}$ and let $t : (S^{V_I})^* \to S^V$ be the induced run-tree. Fix some $v \in V_O$. The map $\hat{f}^v : (S^{\mathrm{View}(v)})^* \to S^v$ is d-compatible. Let $t' : (S^{\mathrm{View}(v)})^* \to S^{\mathrm{View}(v)\cup\{v\}}$ be the run-tree induced by $\hat{f}^v$. For each $\sigma \in (S^{V_I})^*$ we have $t(\sigma)^{\mathrm{View}(v)\cup\{v\}} = t'(\sigma^{\mathrm{View}(v)})$. 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 $\psi \in \mathrm{CTL}^*(\mathrm{View}(v) \cup \{v\})$, any branch $\sigma \in (S^{V_I})^\omega$ and any position i we have $t, \sigma, i \models \psi$ if and only if $t', \sigma^{\mathrm{View}(v)}, i \models \psi$. Since $\varphi_v \in \mathrm{CTL}^*(\mathrm{View}(v) \cup \{v\})$, we deduce that $t' \models \varphi_v$. Therefore, $\hat{f}^v$ is accepted by $\mathfrak{A}_v$ and also by $\mathrm{compat}_v(\mathfrak{A}_v)$.

Conversely, for each $v \in V_O$, let $h^v : (S^{\mathrm{View}(v)})^* \to S^v$ be a strategy accepted by the automaton $\mathrm{compat}_v(\mathfrak{A}_v)$. By Lemma 11, h^v is d-compatible. Let $t_v : (S^{\mathrm{View}(v)})^* \to S^{\mathrm{View}(v)\cup\{v\}}$ be the run-tree induced by h^v. We have $t_v \models \varphi_v$ by definition of $\mathfrak{A}_v$ and Proposition 9. Now, using Lemma 7 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 $t : (S^{V_I})^* \to V^{V_I\cup V_O}$ be the run-tree induced by F. For each $\sigma \in (S^{V_I})^*$ we have $t(\sigma)^{\mathrm{View}(v)\cup\{v\}} = t_v(\sigma^{\mathrm{View}(v)})$ and we obtain as above that $t \models \varphi_v$. Therefore, $t \models \varphi$ and F implements φ on $\mathscr{A}$. □

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 14. *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 $\mathrm{View}(v)$ to the process that writes on v, but such a routing may vary from one output variable to another, in contrast with the case of UWC architectures, 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 $\{0,1\}$ (as shown by Proposition 16 below). Hence, the subclass of UWC architectures is strictly contained in the subclass of well-connected architectures.

In the proof of Proposition 16, we use the following lemma, established in [25] 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 $(f,g) : S^2 \to S^2$ is invertible.

Lemma 15 ([25, Lemma 3.1]). *If $f^1, \ldots, f^n$ 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 16. *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 $\mathscr{A}$ of Figure 2 is well-connected. However, it is not uniformly well-connected. Indeed, suppose it is. Then there 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 $g^v : \{0,1\}^2 \to \{0,1\}^2$. Therefore, uniform well-connectedness of $\mathscr{A}$ implies that every pair (f^{z_i}, f^{z_j}) is invertible, using $g^{z_{ij}}$ as inverse. This is in contradiction with Lemma 15, which implies that for Boolean variables, there are at most three pairwise independent functions. Hence the architecture is not uniformly well-connected. □

Interestingly enough, the size of the alphabet has an influence on the possibility to have a *uniform* routing and Lemma 15 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 17. *The synthesis problem for LTL specifications and well-connected architectures with linearly preordered information is undecidable.*

Let $\mathscr{A}$ be the architecture of Figure 6, 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 $\mathscr{A}$. 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 $\gamma_1 q \gamma_2 \in \Gamma^* Q \Gamma^+$ (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 ($\mathscr{A}'$ 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$(V_I \cup V_O)$.

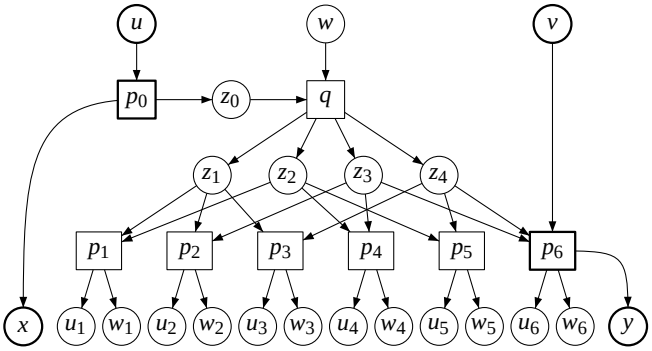

Fig. 6. Undecidable, well-connected and linearly preordered architecture

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:

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

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

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

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

where

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

3. We next express with a formula γ_M that if $n(u) = 1$ then x has to output the first configuration $\mathscr{C}_1$ of M starting from the empty tape. That is, if the input is in $0^q 10\{0,1\}^\omega$, then the corresponding output is $\#^{q+1}\mathscr{C}_1\#^\omega$. The LTL formula is

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

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

4. We say that the input words are *synchronized* either if $u,v \in 0^q 1^p 0\{0,1\}^\omega$ or else 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

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

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:

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

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 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

$$(u = v = 0)\,\mathsf{U}\,\Big((u = 1) \wedge (v = 0) \wedge \mathsf{X}((u = v = 1) \wedge (u = v = 1)\,\mathsf{U}\,(u = v = 0))\Big)$$

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

$$\psi_M = (n(u) = n(v) + 1) \rightarrow \Big((x = y)\,\mathsf{U}\,\big(\mathrm{Trans}(y,x) \wedge \mathsf{X}^3\,\mathsf{G}(x = y)\big)\Big)$$

where $\mathrm{Trans}(y,x)$ 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

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

Here $(x = abc)$ is an abbreviation for $(x = a) \wedge \mathsf{X}(x = b) \wedge \mathsf{X}^2(x = c)$. 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 $\mathscr{A}$. 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 $f^{z_4}(ua, 0^q 1 wb) = a \oplus b$. 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 18. 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 $f^{z_4}(ua, 0^q 1 wb) = a \oplus b$. We also adapt the strategies of $p_1, \ldots, p_6$ so that α is satisfied. Note that these strategies are no longer memoryless, they have to remember the last bit if 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 $F = (f^v)_{v \in V \setminus V_I}$ be a distributed implementation of φ_M on the architecture $\mathscr{A}$ of Figure 6. 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 19. *Let $g_1, g_2, g_3 : \{0,1\}^2 \to \{0,1\}$ 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 $(g_1 \oplus g_2 \oplus g_3)(1,0) = (\neg g_1 \oplus g_2 \oplus \neg g_3)(0,0) = (g_1 \oplus g_2 \oplus g_3)(0,0)$. □

Applying Lemma 19 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 20. *For all $q \geq 0$, there exists $\varepsilon \in \{0,1\}$ such that*

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

Proof. Fix $q \geq 0$. Let $g_i : \{0,1\}^2 \to \{0,1\}$ 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 19, 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 $g1$, 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. □

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

$$\hat{f}^{z_k}(u^0[n], w[n]) = \hat{f}^{z_k}(u[n], w[n]). \tag{3}$$

Proof. By induction on n. If $n \leq q$, then $u^0[n] = u[n]$ so (3) 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 20. 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 20, we deduce that $\hat{f}^{z_4}(0^q 0, 0^q 1) = \hat{f}^{z_4}(0^q 1, 0^q 1)$ and (3) is proved for $n = q+1$.

Finally, assume that $n > q+1$. By induction hypothesis, for $k \in \{3,4\}$ and all $i < n$, we have $\hat{f}^{z_k}(u^0[i], w[i]) = \hat{f}^{z_k}(u[i], w[i])$. Therefore, the history $z_3[n-1]$ and $z_4[n-1]$ is the same on the inputs (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. □

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

Lemma 22. *If x is computed by f^x from the input word u then for all $p > 0$ we have*

$$\forall q \geq 0, \qquad u \in 0^q 1^p 0\{0,1\}^\omega \Longrightarrow x = \#^{p+q} \mathscr{C}_p \#^\omega \tag{4}$$

where $\mathscr{C}_p$ 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 $x = \#^{q+1+p} \mathscr{C}_p \#^\omega$. Using δ, we deduce that on the input triple (u^0, w, v) the output is $y = x = \#^{q+1+p} \mathscr{C}_p \#^\omega$. Now,

by Lemma 21, 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 $y = \#^{q+1+p}\mathscr{C}_p\#^\omega$ by the above. Using ψ_M, we deduce that on the input triple (u, w, v) the output on x must be $x = \#^{q+1+p}\mathscr{C}_{p+1}\#^\omega$. This concludes the proof since x only depends on u. □

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 17). Starting from a Turing machine M, we have shown that any distributed implementation of the specification φ_M is forced to output on x the $n(u)$-th configuration of M. Therefore, there is a distributed implementation on this architecture for the formula $\varphi_M \wedge \mathsf{G}(x \neq \text{halt})$ 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 architecture with linearly preordered information, proving that this latter problem is undecidable (more precisely not co-RE). □

6 Conclusion

In this paper, we have shown that every decidable architecture must have linearly preordered information, and that this condition is sufficient for deciding external specifications on UWC architectures. On the other hand, we have exhibited a well-connected architecture with linearly preordered information, yet undecidable 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.

Acknowledgement. We thank the anonymous referees for their remarks which helped us to improve the presentation of the paper.

References

1. Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89)*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1989.
2. Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

3. André Arnold, Aymeric Vincent, and Igor Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 1(303):7–34, 2003.
4. Julien Bernet and David Janin. On distributed program specification and synthesis in architectures with cycles. In Elie Najm, Jean-François Pradat-Peyre, and Véronique Donzeau-Gouge, editors, *Proceedings of the 26th IFIP WG6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'06)*, volume 4229 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2006.
5. J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
6. Alonzo Church. Logic, arithmetics, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.
7. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the IBM Workshop on Logics of Programs*, 1981.
8. Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Proceedings of the 20th IEEE Annual Symposium on Logic in Computer Science (LICS'05)*, pages 321–330. IEEE Computer Society Press, 2005.
9. Bernd Finkbeiner and Sven Schewe. Synthesis of asynchronous systems. In Germán Puebla, editor, *Proceedings of the International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'06)*, volume 4407 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2006.
10. Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In Kamal Lodaya and Meena Mahajan, editors, *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2004.
11. Paul Gastin, Nathalie Sznajder, and Marc Zeitoun. Distributed synthesis for well-connected architectures. In Naveen Garg and S. Arun-Kumar, editors, *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 321–332. Springer, 2006.
12. Orna Kupferman and Moshe Y. Vardi. Synthesis with incomplete information. In *Proceedings of the 2nd International Conference on Temporal Logic (ICTL'97)*, pages 91–106, 1997.
13. Orna Kupferman and Moshe Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245–263, 1999.
14. Orna Kupferman and Moshe Y. Vardi. μ-calculus synthesis. In Mogens Nielsen and Branislav Rovan, editors, *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'00)*, volume 1893 of *Lecture Notes in Computer Science*, pages 497–507. Springer, 2000.
15. Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In Joseph Y. Halpern, editor, *Proceedings of the 16th IEEE Annual Symposium on Logic in Computer Science (LICS'01)*. IEEE Computer Society Press, 2001.
16. P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *Lecture Notes in Computer Science*, pages 396–407. Springer, 2001.
17. P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2002.

18. P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In Ramaswamy Ramanujam and Sandeep Sen, editors, *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2005.
19. Swarup Mohalik and Igor Walukiewicz. Distributed games. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, volume 2914 of *Lecture Notes in Computer Science*, pages 338–351. Springer, 2003.
20. David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1&2):69–107, 1995.
21. Gary L. Peterson and John H. Reif. Multiple-person alternation. In *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science (FOCS'79)*, pages 348–363. IEEE Computer Society Press, 1979.
22. Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL'89)*, pages 179–190. ACM, 1989.
23. Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89)*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989.
24. Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS'90)*, volume II, pages 746–757. IEEE Computer Society Press, 1990.
25. April Rasala Lehman and Eric Lehman. Complexity classification of network information flow problems. In J. Ian Munro, editor, *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 142–150. Society for Industrial and Applied Mathematics, 2004.
26. Alin Stefănescu, Javier Esparza, and Anca Muscholl. Synthesis of distributed algorithms using asynchronous automata. In Roberto Amadio and Denis Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2003.
27. Wolfgang Thomas. On the synthesis of strategies in infinite games. In Ernst W. Mayr and Claude Puech, editors, *Proceedings of the 12th International Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995.
28. Wolfgang Thomas. Church's problem and a tour through automata theory. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer, 2008.
29. Ron van der Meyden and Moshe Y. Vardi. Synthesis from knowledge-based specifications. In Davide Sangiorgi and Robert de Simone, editors, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 1998.
30. Ron van der Meyden and Thomas Wilke. Synthesis of distributed systems from knowledge-based specifications. In Martín Abadi and Luca de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 562–562. Springer, 2005.