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 set of all sets of input variables visible by output variables is totally ordered, under set inclusion. 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 [2]. It consists in translating a system property, given in a high level specification language (such as temporal logic) into a low-level equivalent model (such as a finite automaton). The problem can be parametrized by the specification language and the target model. For instance, synthesis for infinite sequential systems from monadic second order formulas is simply Büchi's theorem.

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 [11], where general synthesis has been proved undecidable for LTL specifications, and LTL synthesis for pipeline architectures has been shown non elementarily decidable, the lower bound following from a former result on multiplayer games [10]. For local specifications, constraining only variables local to processes [8], the general problem is undecidable (though doubly flanked pipelines become decidable.)

* Work partly supported by the European research project HPRN-CT-2002-00283 GAMES and by the ACI Sécurité Informatique 2003-22 (VERSYDIS).

The pipeline architecture has been shown decidable for CTL* *full* specifications [5], 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 [3] for full specifications, implies that the architecture of Figure 1 is undecidable. The reason is that specifications are allowed 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 allows easy reductions to the basic undecidable architecture of Pnueli and Rosner [11], for instance by breaking communication links at will.

In the seminal paper [11], 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 [11,5].

To rule out unnatural properties yielding undecidability, the specifications we consider are external, coming back to the original framework of [11,2]. We first determine a sufficient condition for undecidability with external specifications, that generalizes the undecidability result of [11]. We next introduce *uniformly well-connected* (UWC) architectures. Informally, an architecture is UWC if there exists a routing of input variables 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, the sufficient condition for undecidability becomes a criterion, for external specifications. We also propose a natural restriction on specifications for which all UWC architectures becomes decidable.

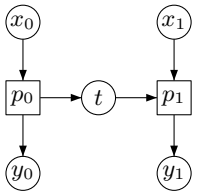

Fig. 1. Architecture decidable for external/undecidable for full specifications.

Finally, we introduce the larger class of *well-connected architectures*, in which the routing of input variables to an output process can depend on that process. We show that our criterion is not necessary anymore for this larger class. Whether the restricted external specifications are always decidable for this class, as it is the case for UWC architectures, remains open. 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.

Due to lack of space, proofs are omitted or only sketched in this extended abstract. A full version is available in [4].

2 Preliminaries

Trees and tree automata. Given two finite sets X and Y, a Y-labeled (full) X-tree is a (total) function $t : X^* \to Y$ 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 $\delta : Q \times Y \to \mathcal{P}(Q^X)$ and an acceptance condition $\alpha \subseteq Q^\omega$. A *run* ρ of such an automaton over a (X, Y)-tree t is a (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 $s_1 s_2 \cdots$ 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 $V_{\mathrm{I}} = \{v \in V \mid E^{-1}(v) = \emptyset\}$ and $V_{\mathrm{O}} = \{v \in V \mid E(v) = \emptyset\}$. 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. Each variable v ranges over a finite domain S^v, given with the architecture. The initial value of the variables is $s_0 = (s_0^v)_{v \in V} \in \prod_{v \in V} S^v$. We will consider that $|S^v| \geq 2$ for all $v \in V$. In fact, if not, such a variable would always have the same value, and could 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 $E^{-1}(v) = \{p\}$.

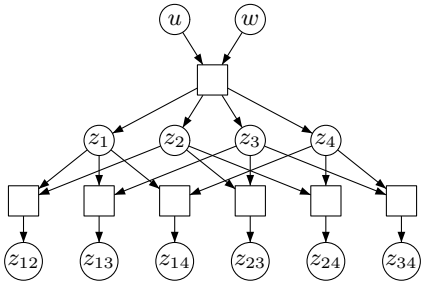

Fig. 2. An architecture

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

Runs. When $U \subseteq V$, S^U will denote $\prod_{v \in U} S^v$. A *configuration* of the architecture is given by a tuple $s \in S^V$ describing the value of the variables. For $s = (s^v)_{v \in V} \in S^V$, $U \subseteq V$, we denote by $s^U = (s^v)_{v \in U}$ the projection of the configuration s to the subset of variables U. 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 is $\sigma^U = s_0^U s_1^U s_2^U \cdots$. Also, we denote 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 $t(\varepsilon) = s_0$ 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, with atomic propositions of the form $(v = a)$ for $v \in U$ 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 $U \subseteq V_\mathrm{I} \cup V_\mathrm{O}$. 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 it writes to, 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 $(s_0\sigma)^{R(v)}$ 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 $(s_0\sigma)^{R(v)}$. Since all runs begin by s_0, this initial configuration is irrelevant for a strategy which only depends on $\sigma^{R(v)}$. 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 $\rho, \rho' \in (S^{R(v)})^i$, if $\rho[i - d_v] = \rho'[i - d_v]$, then $f^v(\rho) = f^v(\rho')$. This condition 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 $F = (f^v)_{v \in V \setminus V_I}$ of local strategies. A run $\sigma \in (S^V)^\omega$ is an *F-run* (or *F-compatible*) if for all $v \in V \setminus V_I$, $s_i^v = f^v(\sigma^{R(v)}[i])$. Given an input sequence $\rho \in (S^{V_I})^\omega$, there is a unique run σ which is F-compatible and such that $\sigma^{V_I} = \rho$. The *F-run tree* is the run tree $t : (S^{V_I})^* \to S^V$ 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, the prefix $\sigma[i]$ only depends on the prefix $\rho[i]$. This shows that the F-run tree is unique.

For a variable $v \in V$, we let $\mathrm{View}(v) = (E^{-2})^*(v) \cap V_I$ be the set of input variables v might depend on. Observe that if $s_0\sigma$ is an F-run then, for all $v \in V \setminus V_I$, for all $i > 0$, 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 obtain the value of v.

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

$$d(u,v) = min\{d_{v_1} + \cdots + d_{v_n} \mid u\ E^2\ v_1\ E^2\ \ldots\ E^2\ v_n = v \text{ is a path in } \mathcal{A}\}.$$

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 $f^v(s_1 \cdots s_i \cdots s_{i+d_v}) = g(s_i)$ 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.

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 $\mathcal{L}$).

3 Architectures with uncomparable information

In this section, we state a necessary condition for decidability.

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

For instance, the architectures of Figures 1 and 3 have linearly preordered information. The following proposition extends the undecidability result of [11,3].

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

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.

A *routing* for an architecture $\mathcal{A} = (V \cup P, E, (S^v)_{v\in V}, s_0, (d_p)_{p\in P})$ 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 allows to transmit to every output variable v, with a minimal delay, the value of the variables in View(v).

Definition 4. *An architecture $\mathcal{A}$ is* uniformly well-connected *(UWC) if there exist a routing Φ and, for every $v \in V_O$ and $u \in View(v)$, 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 $\sigma = s_1 s_2 \cdots \in \left(S^{V\setminus V_O}\right)^+$, we have for $i > 0$*

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

In case there is no delay, the uniform well-connectedness refines the notion of adequate connectivity introduced by Pnueli and Rosner in [11], 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, given an architecture, there is a finite number of routings and a finite number of decoding functions, so that the property of being UWC is NP. Actually, the problem is NP-complete: using a natural reduction, this follows from the NP-hardness of the multicast problem [7], which is a special instance of the network information flow problem [1].

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 5. *Let $\mathcal{A} = (V \cup P, E, (S^v)_{v\in V}, s_0, (d_p)_{p\in P})$ 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 compatible with the delays of $\mathcal{A}$. 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 $v \in V_O$.*

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

Theorem 6. *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 uncomparable 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 $V_{\mathrm{O}} = \{v_1, \ldots, v_n\}$ so that $\mathrm{View}(v_n) \subseteq \cdots \subseteq \mathrm{View}(v_1) \subseteq V_{\mathrm{I}}$.

In the following, in order to use tree-automata, we extend a strategy $f : (S^X)^+ \to S^Y$ by $f(\varepsilon) = s_0^Y$ so that it becomes a (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 a $(S^{\mathrm{View}(v_1)}, S^{V_{\mathrm{O}}})$-tree h satisfying $h(\varepsilon) = s_0^{V_{\mathrm{O}}}$. This first step is simply the program synthesis for a single process with incomplete information (since we may have $\mathrm{View}(v_1) \subsetneq V_{\mathrm{I}}$). This problem was solved in [6] for CTL* specifications.

Proposition 7 ([6, Th. 4.4]). *Given an external specification $\varphi \in CTL^*(V_I \cup V_O)$, one can build a NDTA $\mathfrak{A}_1$ over $(S^{View(v_1)}, S^{V_O})$-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 5, we have to check whether for each $v \in V_{\mathrm{O}}$ there exists an $(S^{\mathrm{View}(v)}, S^v)$-tree h^v which is compatible with the delays and such that the global strategy $\bigoplus_{v \in V_{\mathrm{O}}} h^v$ induced by the collection $(h^v)_{v \in V_{\mathrm{O}}}$ is accepted by $\mathfrak{A}_1$. Formally, the *sum* of strategies is defined as follows. Let $X = X_1 \cup X_2 \subseteq V_{\mathrm{I}}$ and $Y = Y_1 \uplus Y_2 \subseteq V_{\mathrm{O}}$, and for $i = 1, 2$ let h_i be a (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 $(h^v)_{v \in V_{\mathrm{O}}}$, 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 8. *Let $1 \leq k < n$. Given a NDTA $\mathfrak{A}_k$ accepting $(S^{View(v_k)}, S^{V_k})$-trees, we 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 a $(S^{View(v_k)}, S^{v_k})$-tree h^{v_k} which is compatible with the delays and such that $h^{v_k} \oplus t$ is accepted by $\mathfrak{A}_k$.*

The proof of Proposition 8 divides in two steps. Since $V_k = \{v_k\} \cup V_{k+1}$, for each $(S^{\mathrm{View}(v_k)}, S^{V_k})$-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 turn 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 compatible with the delays (Lemma 9). 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 9. *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 $\mathcal{L}(\mathfrak{A}') = \{t \in \mathcal{L}(\mathfrak{A}) \mid t^v$ is compatible with the delays$\}$.*

Proof. Intuitively, to make sure that the function t^v is compatible with the delays, 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 function $g : (S^{\mathrm{View}(v)})^K \to S^v$ that is already compatible with the delays and that predicts what will be the v-values K steps later. 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 $Q' = Q \times \mathcal{F}$ where $\mathcal{F}$ is the set of functions $f : (S^{\mathrm{View}(v)})^{<K} \to S^v$ which are compatible with the delays, 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 v-value of the tree.

Transitions of $\mathfrak{A}'$ will be defined using the function $\Delta : \mathcal{F} \times S^{\mathrm{View}(v)} \to \mathcal{P}(\mathcal{F})$ 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

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

Finally, the set of initial states of $\mathfrak{A}'$ is $I' = \{q_0\} \times \mathcal{F}$ and $\alpha' = \pi^{-1}(\alpha)$ where $\pi : (Q \times \mathcal{F})^\omega \to Q^\omega$ 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 $\mathfrak{A}$. One can check that the automaton $\mathfrak{A}'$ satisfies the requirements of Lemma 9. □

Proof (of Proposition 8). We consider the NDTA $\mathrm{compat}_{v_k}(\mathfrak{A}_k)$. 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 $S^{\mathrm{View}(v_{k+1})}$ 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 [6]. The automaton $\mathfrak{A}_{k+1}$ fulfilling the requirements of Proposition 8 is therefore given by $\mathrm{narrow}_{\mathrm{View}(v_{k+1})}(\mathrm{proj}_{V_{k+1}}(\mathrm{compat}_{v_k}(\mathfrak{A}_k)))$. Note that, even when applied to a NDTA, the narrowing construction of [6] 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 [9]. The drawback is that this involves an exponential blow up. Unfortunately, this is needed since Lemma 9 requires a NDTA as input. □

We can now conclude the proof of Theorem 6. Using Proposition 8 inductively starting from the NDTA $\mathfrak{A}_1$ of Proposition 7, we obtain a NDTA $\mathfrak{A}_n$ accepting a $(S^{\mathrm{View}(v_n)}, S^{v_n})$-tree h^{v_n} if and only if for each $1 \leq i < n$, there exists a $(S^{\mathrm{View}(v_i)}, S^{v_i})$-tree h^{v_i} which is compatible with the delays and such that $h^{v_1} \oplus \cdots \oplus h^{v_n}$ is accepted by $\mathfrak{A}_1$. Therefore, using Remark 1 and Lemma 5, there is a distributed implementation for the specification over $\mathcal{A}$ if and only if $\mathcal{L}(\mathrm{compat}_{v_n}(\mathfrak{A}_n))$ is nonempty. The overall procedure is non-elementary due to the exponential blow-up of the inductive step in Proposition 8. □

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 10. *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 $\varphi_v \in \mathcal{L}(\mathit{View}(v) \cup \{v\})$.*

Proposition 11. *The synthesis problem for UWC architectures and external robust CTL* specifications is decidable.*

Proof. Let $\mathcal{A} = (V \cup P, E, (S^u)_{u \in V}, s_0, (d_p)_{p \in P})$ be a UWC architecture and φ be an external and 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 7, 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 $t_v : (S^{\mathrm{View}(v)})^* \to S^{\mathrm{View}(v) \cup \{v\}}$ satisfies φ_v. Using Remark 1 and Lemma 5 one can show the following claim from which Proposition 11 follows.

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

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 12. *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.*

The architecture of Figure 2 is well-connected but not UWC when the variables are boolean. This follows from similar results on the multicast problem [7]. Hence, the subclass of UWC architectures is strictly contained in the subclass of well-connected architecture. Note that the size of the variable domains has a major influence: any well-connected architecture with sufficiently large domain sizes is UWC.

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

Theorem 13. *The synthesis problem for LTL specifications and well-connected, linearly preordered architectures is undecidable.*

Let $\mathcal{A}$ be the architecture of Figure 3, 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 $\gamma_1 q \gamma_2 \in \Gamma^* Q \Gamma^+$. 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 $n(u)^{\text{th}}$ 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. The difficulty is to ensure that process p_6 cannot receive relevant information about u.

The specification φ_M is a conjunction of five properties described below that can all be expressed in $\text{LTL}(V_\text{I} \cup V_\text{O})$.

1. The processes p_i for $i \neq 6$ have to output the current values of u and w 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. We can describe this property with a formula α.
2. If the input word on u (resp. v) is in $0^q1^p0\{0,1\}^\omega$, then the corresponding output word x (resp. y) is in $\#^{q+p}\Gamma^*Q\Gamma^+\#^\omega$. This can be expressed by a formula β.
3. We next express with a formula γ that if $n(u) = 1$, then the output on x is the first configuration $\mathcal{C}_1$ of M starting from the empty tape.
4. We say that the input words are *synchronized* if $u, v \in 0^q1^p0\{0,1\}^\omega$ or if $u \in 0^q1^{p+1}0\{0,1\}^\omega$ and $v \in 0^{q+1}1^p0\{0,1\}^\omega$. 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.
5. Finally, one can express with an LTL formula ψ that if the input words are synchronized and if $n(u) = n(v) + 1$, then the configuration encoded on x is obtained by a computation step of M from the configuration encoded on y.

We 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

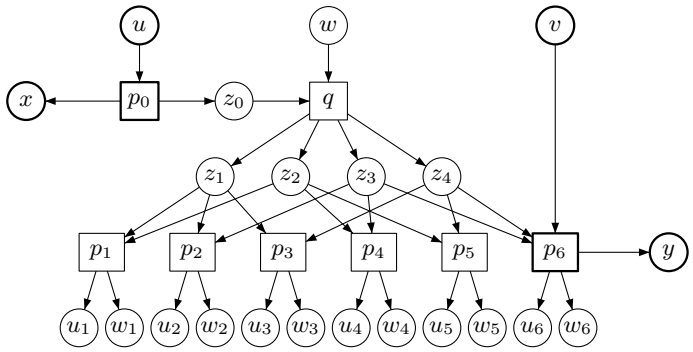

Fig. 3. Undecidable, well-connected, comparable-information architecture

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 $f^{z_4}(ub_1, 0^q 1 w b_2) = b_1 \oplus b_2$. 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.

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, $\beta \wedge \gamma \wedge \delta \wedge \psi$, is satisfied.

Remark 14. 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, $f^{z_4}(u \cdot a_1 \cdot a_2, 0^q b) = a_1 \oplus b$ and we adapt the strategies of $p_1, \ldots, p_6$ so that α is satisfied. 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 induce 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 3. We prove that f^x must simulate the computation of M starting from the empty tape.

Let $q \geq 0$. For $u = 0^q 1 u'$, we define $u^0 = 0^q 0 u'$. 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. This is the main technical lemma whose proof relies on the specification α.

Lemma 15. *Let $u, w \in 0^q 1 \{0,1\}^\omega$. For $k \in \{3, 4\}$, we have for all $n > 0$:*

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

Lemma 16. *If x is computed by f^x from the input word u then for all $p > 0$ we have*

$$\forall q \geq 0, \qquad u \in 0^q 1^p 0 \{0,1\}^\omega \Longrightarrow x = \#^{p+q} \mathcal{C}_p \#^\omega \tag{3}$$

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. Assume now that $u \in 0^q 1^{p+1} 0 \{0,1\}^\omega$ and 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, v, w) the output is $y = x = \#^{q+1+p} \mathcal{C}_p \#^\omega$. Now, by Lemma 15, 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, v, w) and (u, v, w) the outputs on y must be $y = \#^{q+1+p} \mathcal{C}_p \#^\omega$ by the above. Using ψ, we deduce that on the input triple (u, v, w) the output on x must be $x = \#^{q+1+p} \mathcal{C}_{p+1} \#^\omega$. This concludes the proof since x only depends on u. □

It is then easy to get the undecidability of the architecture $\mathcal{A}$ of Figure 3 by considering the specification $\varphi_M \wedge \mathsf{G}(x \neq \text{halt})$.

6 Conclusion

In this paper, we have argued that it is meaningful to rule out specifications for distributed architectures constraining internal variables. 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 *external* and *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. R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Inform. Theory*, 46(4):1204–1216, 2000.
2. A. Church. Logic, arithmetic, and automata. In *Int. Symp. of Mathematicians*, pages 23–35, 1962.
3. B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proc. 20th IEEE Symp. on Logic in Computer Science (LICS 2005)*. IEEE Computer Society, 2005.
4. P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. Technical report, LSV, 2006.
5. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proceedings of LICS'01*. Computer Society Press, 2001.
6. O. Kupferman and M. Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245–263, June 1999.
7. A. R. Lehman and E. Lehman. Complexity classification of network information flow problems. In *Proceedings of SODA'04*, pages 142–150. SIAM, 2004.
8. 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.
9. 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.
10. 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.
11. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of 31th IEEE Symp. FOCS*, pages 746–757, 1990.