Decidability of well-connectedness for distributed synthesis*

Paul Gastin and Nathalie Sznajder

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

Abstract. Although the synthesis problem is often undecidable for distributed, synchronous systems, it becomes decidable for the subclass of *uniformly well-connected (UWC)* architectures, provided that only robust specifications are considered. It is then an important issue to be able to decide whether a given architecture falls in this class. This is the problem addressed in this paper: we establish the decidability and precise complexity of checking this property. This problem is in 2-NEXPTIME in the general case, but becomes NP-complete if restricted to a natural subclass of architectures.

1 Introduction

The synthesis problem consists in, given a high-level description of a system, automatically deriving a program that behaves according to the specification. We consider here the synthesis problem for *reactive* and *open* systems, i.e., systems whose role are to maintain an ongoing interaction with an environment. The specification should describe the expected behavior of the system throughout its activity. This problem is closely related to Church's problem [4], that was solved in [2]. Later, [16] has observed that even though the specification is expressed on strings, its synthesis is actually a problem on tree automata.

Since then, it has received a lot of attention, and various refinements have been studied. The precise setting of the problem has several parameters: centralized or distributed systems (in a distributed setting, one is given, along with the specification, a set of processes and an architecture of communication between them - typically a communication through variables whose domains are fixed), synchronous (at each time step, the environment produces an input, and the system reacts with a corresponding output) or asynchronous behaviors (in which the environment and the system do not evolve at the same speed), type of specifications and type of memory assumed for the processes. In the centralized framework, Pnueli and Rosner have proven the synthesis problem decidable for both synchronous and asynchronous semantics and specifications expressed by

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

a formula in linear temporal logic – see [13] and [14], with a complexity doubly exponential in the size of the formula. Another interesting problem arises when considering distributed systems: the question is now to synthesize a program for each process, that only depends on its local knowledge, such that the overall behavior of the system meets the given formula. This has been studied by [15], for systems having a *synchronous* behavior, and with *external* specifications in linear temporal logic. By *external* we mean a specification that only relates input to output values and is not allowed to constrain internal variables of communication. Unfortunately, this problem is undecidable in general for LTL specifications [15] and later, [5] has adapted the undecidability proof to the case of CTL specifications. However, [15] proved that synthesis for pipeline architectures is non-elementarily decidable - the lower bound following from a former result on multiplayer games [12]. This decidability result has been strengthened in [8] where it is shown that synthesis for pipelines is decidable even with CTL* *full* specifications, i.e., specifications that are allowed to constrain internal links of communication between the processes. In [5] a decision criterion has been established, again for full specifications. A different approach has been taken in [9]: when considering *local* specifications, i.e., specifications that only relate input and output of a same process, the synthesis problem remains undecidable in general, but doubly-flanked pipelines are decidable.

For asynchronous systems, [10] have exhibited a specific subclass of controllers for which the problem is decidable, provided trace-closed specifications. This result has been strenghtened in [11] where a more generic class of controllers have been considered. In an incomparable framework, [6] have identified another subclass of decidable architectures, using an enhanced memory for the processes. More recently, [3] have considered a model where processes communicate through *signals* (a sort of handshaking but initiated by only one process). If the specifications respect some natural closure properties, the asynchronous synthesis problem becomes decidable for strongly connected architectures.

This very active line of research has clearly shown that the synthesis problem for distributed systems is largely influenced by the hypotheses on specifications and the architectures.

Contributions. In [7], we have considered the synthesis problem for external specifications, and synchronous semantics, where each process is assigned a non-negative 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 [15,8]. We have defined a subclass of architectures, the uniformly well-connected (UWC) architectures, for which we have found a decidability criterion. 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. However the decidability of the criterion itself had not been proved. Indeed, for the processes to decode the messages they receive and obtain the values of said inputs, we may need memory. If it is infinite, or without a bound on the size of this memory, a naive algorithm enumerating the different

possible routings and decoding functions may not terminate. In this paper we show that if an architecture is UWC, then there exist decoding functions with finite memories. We derive an algorithm to decide whether an architecture is UWC and evaluate its complexity: 2-NEXPTIME in the general case. We also make explicit the link between this notion and communication flow problems with network coding introduced in [1]. This relation allows us to establish a complexity lower bound. We also show that under natural restrictions checking UWC becomes NP-complete.

2 Preliminaries

We use the formalism and notations defined in [7].

Architectures. An *architecture* $\mathcal{A} = (V \uplus P, E, (S^v)_{v\in V}, s_1, (d_p)_{p\in P})$ 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

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

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. When $U \subseteq V$, S^U will denote $\prod_{v\in U} S^v$. A *configuration* of the architecture is given by a tuple $s = (s^v)_{v\in V} \in S^V$ 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 variables in U. The initial configuration is $s_1 = (s_1^v)_{v\in V} \in S^V$.

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 $s_1^v = 0$ 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 output variables in $E(p)$. 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 $E^{-1}(v) = \{p\}$.

An example of an architecture is given in Figure 1, where processes are represented by boxes and variables by circles.

Runs. A *run* of an architecture is an infinite sequence of configurations, *i.e.*, an infinite word over the alphabet S^V, starting with the initial configuration $s_1 \in S^V$ given with the architecture. If $\sigma = s_1 s_2 s_3 \cdots \in (S^V)^\omega$ is a run, then its

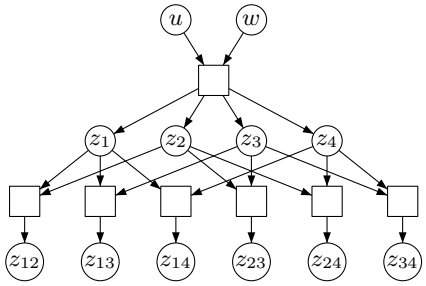


Fig. 1. An architecture

projection on $U \subseteq V$ is $\sigma^U = s_1^U s_2^U s_3^U \cdots$. Also, we denote by $\sigma[i]$ the prefix of length i of σ (by convention, $\sigma[i] = \varepsilon$ if $i \leq 0$).

Programs, strategies. We consider a discrete time, synchronous semantics. Informally, at each step 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) advices a value to write on variable v. But, since the process may have a certain delay d_v, the output of the strategy must not depend on the last d_v values of $\sigma^{R(v)}$.

Formally, a *program* (or *local strategy*) for variable v is a mapping $f^v : \left(S^{R(v)}\right)^+ \to S^v$ 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 $\sigma \in (S^V)^\omega$ is an *F-run* (or *F-compatible*) if for all $v \in V \setminus V_{\mathrm{I}}$, $s_i^v = f^v(\sigma^{R(v)}[i])$. 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 $\sigma^{V_{\mathrm{I}}} = \rho$.

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

Routing. A *routing* for an architecture $\mathcal{A} = (V \uplus P, E, (S^v)_{v \in V}, s_1, (d_p)_{p \in P})$ is a family $\Phi = (f^v)_{v \in V \setminus (V_{\mathrm{I}} \cup V_{\mathrm{O}})}$ of *memoryless* local strategies. Observe that a routing does not include local strategies for output variables.

View. 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.

Delays. The *smallest cumulative delay* of transmission from u to v is defined by $d(u, u) = 0$, $d(u, v) = +\infty$ if $v \notin (E^2)^+(u)$, i.e., if there is no path from u to v

in the architecture, and for $u \neq v \in (E^2)^+(u)$

$$d(u,v) = d_v + \min\{d(u,w) \mid w \in R(v) \text{ and } w \in (E^2)^+(u)\} \ .$$

We then say that an architecture is uniformly well connected if there exists a routing Φ that allows to transmit with a minimal delay to every process p writing to an output variable v, the values of the variables in View(v).

Definition 1. *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 [15], as we no longer require each output variable to be communicated the value of *all* input variables, but only those in its view.

3 Decidability and complexity of checking UWC

As already pointed out in [7], the definition of uniform well connectedness is not symetric: whereas the routing functions are memoryless, some memory is required for the decoding functions. We take the example given in [7] to prove this fact. Consider the architecture given in Figure 2. The delays are written next to the processes, and all variables range over the domain $\{0,1\}$. It is clear 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 make 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 $\mathsf{Y}\, u_2$.

Let us show formally that if we restrict to memoryless decoding functions, we cannot recover the values of the input variables. Fix a routing $\Phi = f^t$, where f^t is memoryless, and fix decoding functions $(g^{u,v})_{v \in V_O, u \in \text{View}(v)}$ satisfying (1). Note that, if $s_1 s_2 s_3 \in (S^V)^3$ is a Φ-sequence, then s_3^t only depends on $s_2^{\{u_1,u_2\}}$ since f^t is memoryless and compatible with the delays. Now, f^t cannot be injective, hence we find another Φ-sequence $s_1 s_2' s_3' \in (S^V)^3$ with $s_3^t = s_3'^t$ and $s_2^{\{u_1,u_2\}} \neq s_2'^{\{u_1,u_2\}}$ and $s_3^{\{u_1,u_2\}} = s_3'^{\{u_1,u_2\}}$. For instance $s_2^{u_2} \neq s_2'^{u_2}$ and we get

$$g^{u_2,v_1}\big((s_1 s_2 s_3)^{\{u_1,t\}}\big) = s_2^{u_2} \neq s_2'^{u_2} = g^{u_2,v_1}\big((s_1 s_2' s_3')^{\{u_1,t\}}\big).$$

It follows that g^{u_2,v_1} is not memoryless.

However, and interestingly, if an architecture is uniformly well-connected, the decoding functions have *finite* memory, which allows to prove the decidability of this criterion.

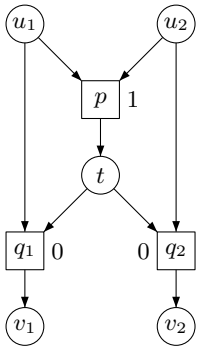


Fig. 2. A uniformly well-connected architecture

3.1 Decidability

To show decidability, we start by proving that if the architecture is UWC with a routing $\Phi = (f^v)_{v \in V \setminus (V_\mathrm{I} \cup V_\mathrm{O})}$ and decoding functions $(g^{u,v})_{u \in \mathrm{View}(v)}$ for all $v \in V_\mathrm{O}$, then the decoding functions are finite-state. In the following, we will denote by g^v the tuple $(g^{u,v})_{u \in \mathrm{View}(v)}$ of decoding functions associated with an output variable v. Observe that since the routing functions are memoryless and delay-compatible, the value of any variable in a Φ-compatible sequence is influenced only by a limited number of input values in the past. This is expressed by the following lemma.

First, we introduce some additional notations. Here we use not only the minimal transmission delay $d(u, v)$ from u to v but also the maximal transmission delay $D(u, v)$ defined by $D(u, u) = 0$, $D(u, v) = +\infty$ if $v \notin (E^2)^+(u)$, and if $u \neq v \in (E^2)^+(u)$ then

$$D(u, v) = d_v + \max\{D(u, w) \mid w \in R(v) \text{ and } w \in (E^2)^+(u)\} .$$

If $\sigma = s_1 s_2 \cdots$ is a sequence and i, j are integers then $\sigma[i \cdots j]$ denotes the subsequence $s_i \cdots s_j$, which is empty if $i > j$. We also use this notation when $i \leq 0$ or $j \leq 0$: if $j \leq 0$ then $\sigma[i \cdots j] = \varepsilon$ is the empty sequence, and if $i \leq 0 < j$ then $\sigma[i \cdots j] = \sigma[1 \cdots j]$.

Lemma 2. *For all $v \in V \setminus V_O$, for any routing Φ and any Φ-compatible sequence $\sigma = s_1 \dots s_i$, the value s_i^v only depends on $(\sigma^u[i - D(u, v) \cdots i - d(u, v)])_{u \in View(v)}$.*

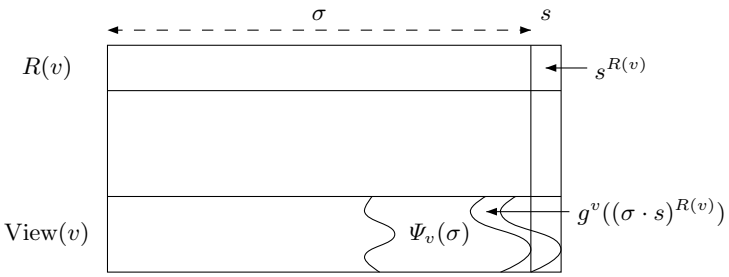
Proof. The proof is by induction on the variables of the acyclic architecture. Let v be an input variable. Then $\mathrm{View}(v) = \{v\}$, $D(v, v) = d(v, v) = 0$ and obviously s_i^v depends only on s_i^v. Now suppose $v \in V \setminus (V_\mathrm{O} \cup V_\mathrm{I})$. The sequence σ being a Φ-compatible sequence, we have $s_i^v = f^v(\sigma^{R(v)})$. The routing functions are memoryless, so $f^v(\sigma^{R(v)})$ depends only on $s_{i-d_v}^{R(v)}$. Let $w \in R(v)$. By induction hypothesis, $s_{i-d_v}^w$ depends only on $(\sigma^u[i - d_v - D(u, w) \cdots i - d_v - d(u, w)])_{u \in \mathrm{View}(w)}$. We have $\mathrm{View}(v) = \bigcup_{w \in R(v)} \mathrm{View}(w)$. Hence, using the definitions of the minimal and maximal transmission delays d and D we deduce that s_i^v depends only on $(\sigma^u[i - D(u, v) \cdots i - d(u, v)])_{u \in \mathrm{View}(v)}$. □

For all $v \in V_O$ and for all finite Φ-compatible sequence $\sigma = s_1 \dots s_{|\sigma|} \in (S^{V \setminus V_O})^*$, we define

$$\Psi_v(\sigma) = \big(\sigma^u[|\sigma| + 1 - D(u,v) + d_v \cdot\cdot |\sigma| - d(u,v) + d_v]\big)_{u \in \mathrm{View}(v)},$$
$$\overline{\Psi}_v(\sigma) = \big(\sigma^u[1 \cdot\cdot |\sigma| - d(u,v) + d_v]\big)_{u \in \mathrm{View}(v)}.$$

The next proposition shows that $\Psi_v(\sigma)$ is the only memory needed to decode the input values.

Proposition 3. *Let $\sigma \cdot s$ and $\sigma' \cdot s'$ be two Φ-compatible sequences with $\sigma, \sigma' \in (S^{V \setminus V_O})^*$ and $s, s' \in S^{V \setminus V_O}$ such that $\Psi_v(\sigma) = \Psi_v(\sigma')$ and $s^{R(v)} = s'^{R(v)}$. Then $g^v((\sigma \cdot s)^{R(v)}) = g^v((\sigma' \cdot s')^{R(v)})$ and $\Psi_v(\sigma \cdot s) = \Psi_v(\sigma' \cdot s')$.*



Proof. We write $\sigma \cdot s = s_1 \dots s_{|\sigma|} s_{|\sigma|+1}$ and similarly for $\sigma' \cdot s'$. Of course, in general, $(\sigma \cdot s)^{R(v)} \neq (\sigma' \cdot s')^{R(v)}$ We build another Φ-compatible sequence $\sigma'' \cdot s''$ such that $(\sigma \cdot s)^{R(v)} = (\sigma'' \cdot s'')^{R(v)}$ and $g^v((\sigma' \cdot s')^{R(v)} = g^v(\sigma'' \cdot s'')^{R(v)})$. Let $\sigma'' \cdot s''$ be such that for all $u \in \mathrm{View}(v)$ we have

$$\sigma''^u = \sigma^u[1 \cdot\cdot |\sigma| - d(u,v) + d_v] \cdot \sigma'^u[|\sigma'| + 1 - d(u,v) + d_v \cdot\cdot |\sigma'|]$$

and $s''^u = s'^u$. The remaining input variables can be set arbitrarily. We claim that $(\sigma'' \cdot s'')^{R(v)} = (\sigma \cdot s)^{R(v)}$. This claim is proved below but we first show how it allows to derive the lemma.

Since $\sigma' \cdot s'$ and $\sigma'' \cdot s''$ are both Φ-compatible sequences, for each $u \in \mathrm{View}(v)$ we have

$$g^{u,v}((\sigma' \cdot s')^{R(v)}) = s'^u_{|\sigma'|+1-d(u,v)+d_v}$$
$$g^{u,v}((\sigma'' \cdot s'')^{R(v)}) = s''^u_{|\sigma''|+1-d(u,v)+d_v}$$

and $s''^u_{|\sigma''|+1-d(u,v)+d_v} = s'^u_{|\sigma'|+1-d(u,v)+d_v}$ by definition of σ''. Using the claim we deduce $g^{u,v}((\sigma' \cdot s')^{R(v)}) = g^{u,v}((\sigma \cdot s)^{R(v)})$. Therefore, $g^v((\sigma \cdot s)^{R(v)}) = g^v((\sigma' \cdot s')^{R(v)})$. Together with $\Psi_v(\sigma) = \Psi_v(\sigma')$ this implies $\Psi_v(\sigma \cdot s) = \Psi_v(\sigma' \cdot s')$ which yields the lemma.

Now we prove the claim. Note that $|\sigma''| = |\sigma|$. By definition of σ'', we have $\overline{\Psi}_v(\sigma'') = \overline{\Psi}_v(\sigma)$. Let $w \in R(v)$ and $i \leq |\sigma| = |\sigma''|$. For each $u \in \mathrm{View}(w) \subseteq \mathrm{View}(v)$ we have $i - d(u,w) \leq |\sigma| - d(u,v) + d_v$. Using Lemma 2 and $\overline{\Psi}_v(\sigma'') = \overline{\Psi}_v(\sigma)$, we deduce that $s''^w_i = s^w_i$. Therefore, $\sigma''^{R(v)} = \sigma^{R(v)}$.

It remains to show that $s''^{R(v)} = s^{R(v)}$. Using $\Psi_v(\sigma) = \Psi_v(\sigma')$ and the definition of σ'' and s'', we get for each $u \in \mathrm{View}(v)$

$$\begin{aligned}(\sigma'' \cdot s'')^u[|\sigma'' \cdot s''| - D(u,v) + d_v \cdot\cdot |\sigma'' \cdot s''| - d(u,v) + d_v] \\ = (\sigma' \cdot s')^u[|\sigma' \cdot s'| - D(u,v) + d_v \cdot\cdot |\sigma' \cdot s'| - d(u,v) + d_v].\end{aligned}$$

Let $w \in R(v)$. Note that for $u \in \mathrm{View}(w)$ we have $D(u,w) \leq D(u,v) - d_v$ and $d(u,w) \geq d(u,v) - d_v$. We deduce using Lemma 2 that $s''^w = s^w$. Therefore, $s''^{R(v)} = s'^{R(v)}$. Using the hypothesis $s'^{R(v)} = s^{R(v)}$ we obtain $s''^{R(v)} = s^{R(v)}$ which concludes the proof of the claim. □

Now we can define a deterministic automaton $\mathcal{B}_v = (Q_v, q_0^v, S^{R(v)}, \delta_v, \alpha^v)$ with output that computes g^v with a finite memory:

- $Q_v = \{\Psi_v(\sigma) \mid \sigma \text{ is a finite } \Phi\text{-compatible sequence}\}$ is the finite set of states and $q_0^v = \Psi_v(\varepsilon)$ is the initial state.
- $S^{R(v)}$ is the input alphabet.
- $\delta_v : Q_v \times S^{R(v)} \to Q_v$ is the transition function defined by

$$\delta_v(\Psi_v(\sigma), s^{R(v)}) = \begin{cases} \Psi_v(\sigma \cdot s) & \text{if } \sigma \cdot s \text{ is a } \Phi\text{-compatible sequence} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

  Note that δ_v is well-defined by Proposition 3. We immediately deduce by induction that $\delta_v(q_0, \sigma^{R(v)}) = \Psi_v(\sigma)$ for all finite Φ-compatible sequence σ.
- $\alpha^v : Q_v \times S^{R(v)} \to S^{\mathrm{View}(v)}$ is the output function defined by

$$\alpha^v(\Psi_v(\sigma), s^{R(v)}) = \begin{cases} g^v((\sigma \cdot s)^{R(v)}) & \text{if } \sigma \cdot s \text{ is a } \Phi\text{-compatible sequence} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

  Note that α^v is also well-defined by Proposition 3. For any Φ-compatible sequence $\sigma \cdot s$, we have

$$g^v((\sigma \cdot s)^{R(v)}) = \alpha^v(\Psi_v(\sigma), s^{R(v)}) = \alpha^v(\delta_v(q_0^v, \sigma^{R(v)}), s^{R(v)}).$$

  Hence the finite state deterministic automaton $\mathcal{B}_v$ computes g^v.

Note that, if the architecture is 0-delay, i.e., $d_p = 0$ for all $p \in P$, then Q_v is a singleton and g^v is memoryless.

In the following, we let $D = \max\{D(u,v) - d_v \mid v \in V_{\mathrm{O}}, u \in \mathrm{View}(v)\}$, and we fix a routing $\Phi = (f^v)_{v \in V \setminus \{V_{\mathrm{I}} \cup V_{\mathrm{O}}\}}$.

Definition 4. *For any finite Φ-sequence $\sigma \in (S^{V \setminus V_O})^*$, we denote by* $\mathrm{Suff}_\Phi(\sigma)$ *the Φ-sequence of length D induced by $\rho = \sigma^{V_I}[|\sigma| + 1 - D \cdot\cdot |\sigma|] \in (S^{V_I})^D$.*

Remark 5. The automaton $\mathcal{B}_v$ satisfies $\delta_v(q_0^v, \sigma^{R(v)}) = \delta_v(q_0^v, (\mathrm{Suff}_\Phi(\sigma))^{R(v)})$ for all finite Φ-sequence σ. Indeed, since $\Psi_v(\sigma) = \Psi_v(\mathrm{Suff}_\Phi(\sigma))$ by definition, we have $\delta_v(q_0^v, \sigma^{R(v)}) = \Psi_v(\sigma) = \Psi_v(\mathrm{Suff}_\Phi(\sigma)) = \delta_v(q_0^v, (\mathrm{Suff}_\Phi(\sigma))^{R(v)})$.

We now show that verifying whether an automaton with output indeed computes the decoding functions is decidable. We will need for this the following intermediate results.

Lemma 6. *Let $\mathcal{C} = (Q, q_0, S^{R(v)}, \delta, \alpha)$ be a finite automaton and consider the property $P(\sigma)$ of finite Φ-sequences $\sigma \in (S^{V\setminus V_O})^*$ defined by:*

$$\delta(q_0, \sigma^{R(v)}) = \delta(q_0, (\mathrm{Suff}_\Phi(\sigma))^{R(v)}) \qquad P(\sigma)$$

Then, $P(\sigma)$ holds for any finite Φ-sequence if and only if $P(\sigma)$ holds for any Φ-sequence of length $D+1$.

Proof. Suppose that $P(\sigma)$ holds for any Φ-sequence of length $D+1$. Let σ be an arbitrary Φ-sequence. We prove that $P(\sigma)$ holds by induction on the length of σ. If $|\sigma| \leq D$ then $\mathrm{Suff}_\Phi(\sigma) = \sigma$ and $P(\sigma)$ holds. Assume now $|\sigma| = k+1$ with $k \geq D$. We write $\sigma = s_1 \dots s_{k+1}$ with $s_i \in S^{V\setminus V_O}$. We have

$$\begin{aligned}
\delta(q_0, \sigma^{R(v)}) &= \delta(\delta(q_0, \sigma[k]^{R(v)}), s_{k+1}^{R(v)}) \\
&= \delta(\delta(q_0, \mathrm{Suff}_\Phi(\sigma[k])^{R(v)}), s_{k+1}^{R(v)}) \text{ by induction hypothesis} \\
&= \delta(q_0, (\mathrm{Suff}_\Phi(\sigma[k]) \cdot s_{k+1})^{R(v)}).
\end{aligned}$$

Now, let $\sigma' = s'_1 \dots s'_{D+1} \in (S^{V\setminus V_O})^{D+1}$ be the Φ-sequence induced by the input sequence $\sigma^{V_I}[|\sigma| - D \cdot\cdot |\sigma|]$. From Definition 4 we deduce $\mathrm{Suff}_\Phi(\sigma[k]) = \sigma'[D]$. Also, for $w \in R(v)$ and $u \in \mathrm{View}(w)$, we have $D(u,w) \leq D(u,v) - d_v \leq D$, and Lemma 2 implies that $s'^{R(v)}_{D+1} = s^{R(v)}_{k+1}$. Therefore, $(\mathrm{Suff}_\Phi(\sigma[k]) \cdot s_{k+1})^{R(v)} = \sigma'^{R(v)}$. Since $|\sigma'| = D+1$, our hypothesis implies

$$\delta(q_0, \mathrm{Suff}_\Phi(\sigma[k]) \cdot s_{k+1})^{R(v)}) = \delta(q_0, \sigma'^{R(v)}) = \delta(q_0, \mathrm{Suff}_\Phi(\sigma')^{R(v)}).$$

To conclude, observe now that $\mathrm{Suff}_\Phi(\sigma') = \mathrm{Suff}_\Phi(\sigma)$. Hence, we get $P(\sigma)$. □

Lemma 7. *Let $\mathcal{C} = (Q, q_0, S^{R(v)}, \delta, \alpha)$ be a finite automaton with output such that $P(\sigma)$ holds for any finite Φ-sequence. Consider the property $P'(\sigma)$ of finite Φ-sequences $\sigma = s_1 \dots s_{k+1} \in (S^{V\setminus V_O})^*$ defined by:*

$$\alpha(\delta(q_0, \sigma^{R(v)}[k]), s_{k+1}^{R(v)}) = (s^u_{k+1-d(u,v)+d_v})_{u \in View(v)} \qquad P'(\sigma)$$

with the convention $s_i^u = 0$ for $i \leq 0$.

Then, $P'(\sigma)$ holds for any finite Φ-sequence if and only if $P'(\sigma)$ holds for any Φ-sequence of length at most $D+1$.

Proof. Suppose that $P'(\sigma)$ holds for any Φ-sequence of length at most $D+1$. Let $\sigma = s_1 \dots s_{k+1} \in (S^{V\setminus V_O})^*$ be a Φ-sequence with $k \geq D$. Using $P(\sigma[k])$, we get

$$\alpha(\delta(q_0, \sigma^{R(v)}[k]), s_{k+1}^{R(v)}) = \alpha(\delta(q_0, \mathrm{Suff}_\Phi(\sigma[k])^{R(v)}), s_{k+1}^{R(v)}).$$

As in the proof of Lemma 6, let $\sigma' = s'_1 \dots s'_{D+1} \in (S^{V \setminus V_O})^{D+1}$ be the Φ-sequence induced by $\sigma^{V_I}[|\sigma| - D \cdot\cdot |\sigma|]$. We have seen that $s^{R(v)}_{k+1} = s'^{R(v)}_{D+1}$ and $\mathrm{Suff}_\Phi(\sigma[k]) = \sigma'[D]$. Hence,

$$\alpha(\delta(q_0, \mathrm{Suff}_\Phi(\sigma[k])^{R(v)}), s^{R(v)}_{k+1}) = \alpha(\delta(q_0, \sigma'[D]), s'^{R(v)}_{D+1}).$$

Since $|\sigma'| = D + 1$ we may use $P'(\sigma')$ and we get

$$\alpha(\delta(q_0, \sigma'[D]), s'^{R(v)}_{D+1}) = (s'^{u}_{D+1-d(u,v)+d_v})_{u \in \mathrm{View}(v)}.$$

Now, by definition of σ', we have $s'^{u}_{D+1-d(u,v)+d_v} = s^u_{k+1-d(u,v)+d_v}$ for all $u \in \mathrm{View}(v)$. Therefore, $P'(\sigma)$ holds. □

From the above results, we will deduce that the problem of checking UWC for an architecture is decidable and we establish an upper bound on the complexity of this problem.

3.2 Complexity of checking UWC

In this subsection we state the main result of this paper: we formally show that checking uniform well-connectedness is decidable, and establish the precise complexity of checking it.

Proposition 8. *The problem of checking whether a given architecture is UWC is decidable. Moreover, this problem is*

1. *in NP if we restrict to architectures*
   - *which are 0-delay, i.e., such that $d_v = 0$ for all variables v,*
   - *the size of the variable domains is fixed, i.e., $|S^v| \leq c_s$ for all variables v, where c_s is a constant which does not depend on the input,*
   - *the read-degree is fixed, i.e., $|R(v)| \leq c_r$ for all variables v, where c_r is a constant which does not depend on the input.*
2. *in NEXPTIME if the delays are bounded by a constant, i.e., $d_v \leq c_d$ for all variables v, where c_d is a constant which does not depend on the input.*
3. *in 2-NEXPTIME if the architectures are arbitrary.*

Proof. Consider an architecture $\mathcal{A} = (V \uplus P, E, (S^v)_{v \in V}, s_1, (d_p)_{p \in P})$. The non-deterministic procedure to check whether $\mathcal{A}$ is UWC is the following:

- Guess a routing $\Phi = (f^v)_{v \in V \setminus (V_I \cup V_O)}$.
- For each output variable $v \in V_O$, guess a deterministic automaton with output $\mathcal{C} = (Q, q_0, S^{R(v)}, \delta, \alpha)$ with

$$|Q| \leq \prod_{u \in \mathrm{View}(v)} |S^u|^{D(u,v) - d(u,v)}$$

  and for any sequence $\rho \in (S^{V_I})^+$ of length $D + 1$, compute the induced Φ-sequence $\sigma \in (S^{V \setminus V_O})^+$ such that $\sigma^{V_I} = \rho$ and check that $P(\sigma)$ holds and that $P'(\sigma[k+1])$ holds for all $k \leq D$.

Indeed, assume first that the architecture is UWC with routing Φ and decoding functions $g^{u,v}$. For each output variable v, the automaton B_v defined after Proposition 3 satisfies the above requirements by Remark 5 and since it computes the decoding functions $g^v = (g^{u,v})_{u \in \mathrm{View}(v)}$.

Conversely, if we can find a routing Φ and for each output variable v an automaton $\mathcal{C}$ satisfying the requirements above, then using Lemma 7 we deduce that the output function computed by $\mathcal{C}$ satisfies Equation (1) of Definition 1. Therefore, the architecture is UWC.

We investigate now the complexity of this decision procedure. We first compute the size needed to store a routing Φ and an automaton $\mathcal{C}$. For each variable $v \in V \setminus (V_{\mathrm{I}} \cup V_{\mathrm{O}})$, the size needed to write the memoryless routing $f^v : S^{R(v)} \to S^v$ is $|S^{R(v)}| \cdot \log_2 |S^v|$. Hence,

$$|\Phi| \leq \sum_{v \in V \setminus (V_{\mathrm{I}} \cup V_{\mathrm{O}})} |S^{R(v)}| \cdot \log_2 |S^v|.$$

Note that, given Φ and $\rho \in (S^{V_{\mathrm{I}}})^+$, we can compute the induced Φ-sequence σ such that $\sigma^{V_{\mathrm{I}}} = \rho$ and also $\mathrm{Suff}_\Phi(\sigma)$ in polynomial time w.r.t. $|\Phi| + |\rho|$.

Now, for each output variable $v \in V_{\mathrm{O}}$, and each $u \in \mathrm{View}(v)$ we have $D(u,v) - d(u,v) \leq D(u,v) - d_v \leq D$. Hence, the size of the deterministic automaton $\mathcal{C}$ is $|\mathcal{C}| = |\delta| + |\alpha|$ where

$$\begin{aligned} |Q| &\leq \prod_{u \in \mathrm{View}(v)} |S^u|^D = |S^{\mathrm{View}(v)}|^D \\ |\delta| &\leq |Q| \cdot |S^{R(v)}| \cdot \log_2 |Q| \\ |\alpha| &\leq |Q| \cdot |S^{R(v)}| \cdot \log_2 |S^{\mathrm{View}(v)}| \end{aligned}$$

Given $\mathcal{C}$, a Φ-sequence σ and $\mathrm{Suff}_\Phi(\sigma)$, we can check whether $P(\sigma)$ and $P'(\sigma)$ hold in polynomial time w.r.t. $|\mathcal{C}| + |\sigma|$.

Considering that we write in binary the number of variables and of processes, the size of each domain S^v and the value of each delay d_p, the *size* of the architecture is

$$|\mathcal{A}| = \log_2 |V| + \log_2 |P| + |E| + \sum_{v \in V} \log_2 |S^v| + \sum_{p \in P} \log_2 (1 + d_p).$$

We consider now the three cases of Proposition 8.

1. Here, we have $|\Phi| \leq |V| \cdot c_s^{c_r} \cdot \log_2(c_s) = \mathcal{O}(|\mathcal{A}|)$ since $|V| \leq |E| \leq |\mathcal{A}|$. Also, $D = 0$ and $|Q| = 1$ hence we only have to guess the *memoryless* decoding function α with $|\alpha| = c_s^{c_r} \cdot c_r \cdot \log_2(c_s)$ which is a constant. Moreover, we only have to consider input sequences $\rho \in S^{V_{\mathrm{I}}}$ of length 1. We deduce that our non-deterministic algorithm works in polynomial time.
2. Here, we have $D = \mathcal{O}(|V|) = \mathcal{O}(|\mathcal{A}|)$. Now, for any subset $U \subseteq V$ we have $|S^U| \leq 2^{|U| \cdot |\mathcal{A}|}$ since $\log_2 |S^v| \leq |\mathcal{A}|$ for all $v \in V$. Using $|V| \leq |E| \leq |\mathcal{A}|$ we deduce $|S^U| \leq 2^{|\mathcal{A}|^2}$. We deduce that $|\Phi| = 2^{\mathcal{O}(|\mathcal{A}|^2)}$ and $|Q| = 2^{\mathcal{O}(|\mathcal{A}|^3)}$and

finally $|\mathcal{C}| = |\delta| + |\alpha| = 2^{\mathcal{O}(|\mathcal{A}|^3)}$. Now, the number of input sequences $\rho \in (S^{V_\mathrm{I}})^{D+1}$ that we have to consider in our algorithm is also in $2^{\mathcal{O}(|\mathcal{A}|^3)}$. We deduce that our non-deterministic algorithm works in exponential time.

3. In the last case, we can only bound D by $2^{|\mathcal{A}|}$ and we get double exponential time for our non-deterministic algorithm. □

To establish a lower bound for the complexity we first relate uniform well-connectedness to the network information flow problem introduced by Ahlswede, Cai, Li and Yeung in [1]. Instances of such problems are directed acyclic graphs in which two subsets of nodes have been distinguished: the *sources* and the *sinks*. Along with such a graph come a certain amount of *messages*, and each sink demands a subset of the messages. Formally, an instance of the network information flow problem is a tuple $(P, E, M, S, \mathrm{demand})$ where M is the set of messages (input variables), P is the set of processes, $E \subseteq (P \times P) \cup (M \times P)$ is the edge relation defining an acyclic graph and the set $V = E \cap (P \times P)$ corresponds to the (implicit) internal variables of the network. All variables in $M \cup V$ of the network have the same domain S. A process is a source if it is connected to some input message, i.e., the set of sources is $E(M)$. Finally, the map $\mathrm{demand} : P \to 2^M$ defines what messages should be routed to which processes. A sink is a process $p \in P$ with $\mathrm{demand}(p) \neq \emptyset$. We impose that $\mathrm{demand}(p) \subseteq (E^{-1})^*(p) \cap M = \mathrm{View}(p)$. For $(p, q) = v \in V$, we set $R(v) = R(p) = (M \cap E^{-1}(p)) \cup (E \cap (P \times \{p\}))$. The solution of such a problem is a *0-delay routing* $F = (f^v)_{v \in V}$ with $f^v : S^{R(v)} \to S$ and decoding functions $g^{m,p} : S^{R(p)} \to S^m$ for $p \in P$ and $m \in \mathrm{demand}(p)$ such that, for any F-compatible configuration s, we have $g^{m,p}(s^{R(p)}) = s^m$. We say then that F satisfies the demands.

One specific problem that has been more extensively studied in that area is the *multicast*: an instance of the aforementioned problem in which there is a unique source, and every sink demands all messages.

The networks considered in information flow problems mainly differ from our architectures on the following aspects. First, a variable is attached to an edge, hence there is exactly one process that can read each variable whereas in our case several processes might read the same variable. Second, there is a single (uniform) domain for all variables of the network instance of an information flow problem whereas we may have domains with different sizes for the variables of our architectures. And last, there is no delay in the transmission of information when we may add various delays to our processes. Hence, our architectures are more flexible and we get:

Lemma 9. *There is a polynomial reduction from the multicast problem to the uniform well-connectedness problem.*

Proof. Let $\mathcal{A} = (P, E, M, S, \mathrm{demand})$ be an instance of the multicast problem. We have $E(M) = \{p_0\}$ where p_0 is the unique source. We define an architecture $\mathcal{A}' = (V' \cup P', E', (S^v)_{v \in V'}, s_1, (d_p)_{p \in P'})$ by $P' = P$, $V' = {V_\mathrm{I}}' \uplus {V_\mathrm{O}}' \uplus V$ with

$V_{\mathrm{I}}{}' = M$ and $V_{\mathrm{O}}{}' = \{v_p \mid p \in P \text{ and } \mathrm{demand}(p) = M\}$,

$$E' = M \times \{p_0\} \cup \{(p, v_p) \mid \mathrm{demand}(p) = M\} \cup \bigcup_{v=(p,q)\in V} \{(p, v), (v, q)\},$$

$S^v = S$ for all $v \in V'$, $s_1 = (0)_{v\in V'}$ and $d_p = 0$ for all $p \in P'$. Observe that for all $v \in V_{\mathrm{O}}{}'$, we have $\mathrm{View}(v) = V_{\mathrm{I}}{}' = M$ and for all internal variables $v \in V$ we have $R'(v) = R(v)$. Hence, the notion of 0-delay routing on $\mathcal{A}$ coincides with the notion of routing on $\mathcal{A}'$.

Next, we have seen in Section 3.1 that if decoding functions exist for $\mathcal{A}'$ then these decoding functions have finite memory. We have also proved that memory $Q_v = \{\Psi_v(\sigma) \mid \sigma \text{ is a } \Phi\text{-compatible sequence}\}$ is sufficient for the decoding functions $g^v = (g^{u,v})_{u\in \mathrm{View}(v)}$. But when all delays are 0 then $|Q_v| = 1$, which means that the decoding functions $g^{u,v}$ are memoryless. Hence, Condition (1) of Definition 1 can be written $s_i^{\mathrm{View}(v)} = g^v(s_i^{R(v)})$. Therefore, the notion of decoding functions for $\mathcal{A}$ which are memoryless by definition coincides with the notion of decoding functions for $\mathcal{A}'$.

Therefore, the multicast problem for $\mathcal{A}$ coincides with the uniform well-connectedness problem for $\mathcal{A}'$. □

Rasala Lehman and Lehman [17] have shown that the multicast problem with alphabet size $q = p^k$, where p is prime, is NP-hard. Since this can be reduced to our problem by Lemma 9, we also obtain the NP-hardness.

Corollary 10. *The problem of checking whether a given architecture is UWC is NP-hard.*

Actually, it follows from [17] that the multicast problem restricted to the case where alphabet size is fixed and equal to 2, and indegree is fixed and such that $E^{-1}(v) \leq 2$ for any node v of the graph, is also NP-hard. Hence, using the same arguments than in the proof of Lemma 9, we get

Corollary 11. *The problem of checking whether a given architecture such that:*

- *the delay $d_v = 0$ for all variables v,*
- *the size of the variable domains is fixed, i.e., $|S^v| \leq c_s$ for all variables v, where c_s is a constant which does not depend on the input,*
- *the read-degree is fixed, i.e., $|R(v)| \leq c_r$ for all variables $v \in V \setminus (V_I \cup V_O)$, where $c_r \geq 2$ is a constant which does not depend on the input.*

is NP-complete.

4 Conclusion

We have proved that the problem of deciding wether an architecture is uniformly well-connected is decidable and established its complexity. In [7], we have studied another subclass of architectures: the well-connected architectures. An architecture is *well-connected*, if for each output variable $v \in V_{\mathrm{O}}$, the sub-architecture

consisting of $(E^{-1})^*(v)$ is uniformly well-connected. Informally this means that each output variable can be transmitted the values of input variables in its view, within a minimal delay, but we do not require anymore the routing to be uniform for all the output variables. Technically, to check if an architecture is well-connected, it suffices to verify that every sub-architecture consisting in $(E^{-1})^*(v)$ is uniformly well-connected. However, the architectures obtained in that way are less intricated than the general case, so the complexity of checking uniform well-connectedness of such architectures must be simpler. In case there is no delay, checking uniform well-connectedness when there is a single output variable is basically a flow problem, and can thus be solved in polynomial time. However the reduction does not work when we introduce delays in the architectures. It would be interesting to study the complexity of checking whether an architecture is well-connected in the general case.

References

1. R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.
2. J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
3. T. Chatain, P. Gastin, and N. Sznajder. Natural specifications yield decidability for distributed synthesis of asynchronous systems. In *Proceedings of the 35th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'09)*, Lecture Notes in Computer Science. Springer, 2009.
4. A. Church. Logic, arithmetics, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.
5. B. Finkbeiner and S. 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.
6. P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In K. Lodaya and M. 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.
7. P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. In N. 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.
8. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In J. Y. Halpern, editor, *Proceedings of the 16th IEEE Annual Symposium on Logic in Computer Science (LICS'01)*. IEEE Computer Society Press, 2001.
9. P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In F. Orejas, P. G. Spirakis, and J. 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.

10. P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In L. Brim, P. Jancar, M. Kretínský, and A. 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.
11. P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In R. Ramanujam and S. 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.
12. G. L. Peterson and J. 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.
13. A. Pnueli and R. 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.
14. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. 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.
15. A. Pnueli and R. 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.
16. M. O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Boston, MA, USA, 1972.
17. A. Rasala Lehman and E. Lehman. Complexity classification of network information flow problems. In J. I. Munro, editor, *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 142–150. Society for Industrial and Applied Mathematics, 2004.