Natural Specifications Yield Decidability for Distributed Synthesis of Asynchronous Systems*

Thomas Chatain, Paul Gastin, Nathalie Sznajder

LSV, ENS de Cachan, CNRS
61, Av. du Prés. Wilson, F-94230 Cachan, France
{Thomas.Chatain,Paul.Gastin,Nathalie.Sznajder}@lsv.ens-cachan.fr

Abstract. We study the synthesis problem in an asynchronous distributed setting: a finite set of processes interact locally with an uncontrollable environment and communicate with each other by sending signals – actions that are immediately received by the target process. The synthesis problem is to come up with a local strategy for each process such that the resulting behaviours of the system meet a given specification. We consider *external* specifications over *partial orders*. External means that specifications only relate input and output actions from and to the environment and not signals exchanged by processes. We also ask for some closure properties of the specification. We present this new setting for studying the distributed synthesis problem, and give decidability results: the non-distributed case, and the subclass of networks where communication happens through a strongly connected graph. We believe that this framework for distributed synthesis yields decidability results for many more architectures.
Keywords. Distributed synthesis, Asynchronous systems.

1 Introduction

The synthesis problem consists in, given a high-level description of a system, automatically producing a program that behaves according to this specification. This can be parametrized by the specification language and the target model.

In this work, we address this problem for open, distributed, asynchronous systems, with specifications over partial orders. In open reactive systems, the process interacts with an uncontrollable environment and its behavior depends on this interaction. The goal is then to synthesize strategies that control the actions of the system and not those of the environment (see for instance [1,7,8,15]). The distributed case (that is, considering a set of processes that can cooperate against an environment, each process having only a local view of the system) is more involved, and the main hardness result is due to [17]. They proved that, when all processes and the environment evolve synchronously, the general problem is undecidable for LTL specifications, and that LTL synthesis for pipelines is

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

decidable (though non elementary). Some other classes of architectures have been proved decidable: 2-ways pipelines for CTL^* specifications [9], doubly-flanked pipelines for local specifications [11], uniformly well-connected architecture for CTL^* specifications [5]. Synthesis in an asynchronous communication framework has first been studied in [16]. They considered single-process implementations and linear-time specifications. Later, [12] considered the problem in a distributed setting and exhibited a specific class of controllers for which distributed synthesis is decidable for trace-closed specifications. They strenghtened this result in [13], where restrictions on communication patterns of controllers have been reduced. Considering controllers with causal memories yields decidability results for another subclass of systems in [4]. To reason about distributed synthesis in a more abstract framework, both with synchronous and asynchronous semantics, [14] proposed the framework of distributed games – a specialized version of multiplayer games. Recently, the synthesis of asynchronous distributed systems in the general case of μ-calculus specifications was studied in [3].

Here we study a new model, different from the one of [12] in two ways: when there, processes evolve asynchronously only with respect to each other, in our model they also evolve locally asynchronously with respect to the environment. A second difference is in the communication mechanism: whereas in [12] the synchronization of processes is done by rendez-vous (handshaking), we use signals and define for each action an owner that can trigger it, the signal being immediately received by the other process regardless of whether it is willing to receive it or not. This communication mechanism is more convenient than shared variables communication, and more realistic than rendez-vous. As in [5], we do not allow our specifications to constrain the internal behaviour of the system: communications between processes are only restricted by the communication architecture, not by the specification. This assumption is more natural from a practical point of view – when describing the way a system is expected to work, one is only concerned with its external behaviour, the way it interacts with the environment and not by internal communications processes may set up in order to achieve the specified global behaviour. In the framework of asynchronous distributed systems, executions are partial orders of actions. Our specifications will then be formulae whose models are partial orders of external actions. In addition, in order to rule out unnatural constraints between actions, specifications considered in this paper will have some closure properties, ensuring that we do not prevent causalities between events (this would restrain communication abilities of processes) or impose causalities between others when it would make no sense. With this model, we prove decidability for the synthesis problem for the class of architectures whose communication graph is strongly connected. We believe that the synthesis problem will be decidable for many more architectures with these hypotheses.

2 The Model

An architecture defines how a set of processes may communicate with each other and with an (uncontrollable) external environment. An important parameter of the problem is the type of communications allowed between processes. We are interested in asynchronous distributed systems, hence it would be natural to use unbounded fifo channels. However, this leads to infinite state systems, making decidability results more uncertain to obtain.

A finite model can be obtained by using shared variables: processes can write on variables that can be read by other processes. But in an asynchronous system, communication is difficult to achieve with shared variables. Assume that process i wants to transmit to process j a sequence $m_1, m_2, \ldots$ of messages. First, i writes m_1 to some shared variable x. But since processes evolve asynchronously, i does not know when m_1 will be read by j. Hence, some acknowledgement is required from j to i before i may write m_2 to x. Depending on the architecture, this may not be possible. In any cases, it makes synthesis of distributed programs satisfying a given specification harder.

Hence, we will use point to point communication by signals in the vein of [10]. Sending a signal is an action but receiving a signal is not. Instead, all signals sent to some process j are automatically added to its local history, *without requiring actions from* j. The system is still asynchronous, meaning that processes evolve at different speeds. We are interested in synthesizing *local* programs, also called strategies. By *local* we mean that to decide which action it should execute next, a process j only knows its current local history, which automatically includes all signals sent to j in addition to the signals sent by j.

Formally, an architecture is a tuple $\mathcal{A} = (\mathrm{Proc}, E, (\mathrm{In}_i)_{i\in\mathrm{Proc}}, (\mathrm{Out}_i)_{i\in\mathrm{Proc}})$ where (Proc, E) is the directed communication graph whose nodes are processes and there is an edge $(i,j) \in E$ if process i may send signals to process j. For each process $i \in \mathrm{Proc}$, the sets In_i and Out_i define input and output signals that i may receive from or send to the environment. We assume that all these sets are pairwise disjoint. We let $\mathrm{In} = \bigcup_{i\in\mathrm{Proc}} \mathrm{In}_i$ and $\mathrm{Out} = \bigcup_{i\in\mathrm{Proc}} \mathrm{Out}_i$ be the sets of input and output signals of the whole system. Let also $\Gamma = \mathrm{In} \cup \mathrm{Out}$.

In order to realize a specification, the processes may choose for each communication link $(i,j) \in E$ a set $\Sigma_{i,j}$ of signals that i could send to j. Again, we assume that these sets are pairwise disjoint and disjoint from Γ. The complete alphabet (of signals) is then $\Sigma = \Gamma \cup \bigcup_{(i,j)\in E} \Sigma_{i,j}$. The actions in Γ are called *external* signals whereas the actions in $\Sigma \setminus \Gamma$ are called *internal* signals. For each $a \in \Sigma$ we let $\mathrm{process}(a)$ be the set of processes taking part in the execution of a: $\mathrm{process}(a) = \{i\}$ if $a \in \mathrm{In}_i \cup \mathrm{Out}_i$ and $\mathrm{process}(a) = \{i,j\}$ if $a \in \Sigma_{i,j}$.

It should be no surprise now that the *concrete* executions of our asynchronous distributed systems will be Mazurkiewicz traces. We consider the trace alphabet (Σ, D) where the dependence relation $D \subseteq \Sigma \times \Sigma$ is defined by: $(a,b) \in D$ if $\mathrm{process}(a) \cap \mathrm{process}(b) \neq \emptyset$. We recall that a Mazurkiewicz trace t over (Σ, D) is (an equivalence class of) a finite or infinite Σ-labelled poset $t = (V, \leq, \lambda)$ such that for all $x, y \in V$, ${\downarrow}x = \{y \in V \mid y \leq x\}$ is finite, $(\lambda(x), \lambda(y)) \in D$ implies

$x \leq y$ or $y \leq x$, and $x \lessdot y$ implies $(\lambda(x), \lambda(y)) \in D$ where $x \lessdot y$ means that $x < y$ and there is no $z \in V$ such that $x < z < y$.

We denote by $\mathbb{R}(\Sigma, D)$ the set of traces over (Σ, D) and by $\mathbb{M}(\Sigma, D)$ the set of finite traces. For $i \in \text{Proc}$, we denote by $\Sigma_i = \{a \in \Sigma \mid i \in \text{process}(a)\}$ the set of actions visible to process i and by $\Sigma_i^c = \text{Out}_i \cup \bigcup_{j \mid (i,j) \in E} \Sigma_{i,j}$ the set of actions *controlled* by process i. A local strategy for process i is a mapping $f_i : \Sigma_i^* \to \Sigma_i^c$. After a sequence of actions $w \in \Sigma_i^*$ visible to i (but not necessarily all initiated by i) $f_i(w)$ says which action *in* Σ_i^c the process i is willing to play. Observe that another action in $\Sigma_i \setminus \Sigma_i^c$ can be executed by another process before process i had time to play according to its strategy. This would modify its local history, and thus may modify its strategy: processes are then reactive to signals sent to them by other processes and by the environment. A distributed strategy (or program) is a tuple $F = (f_i)_{i \in Proc}$ of local strategies.

Let $t = (V, \leq, \lambda)$ be a run of the system and let $v \in V$. By definition of the dependence relation, the sets of events $\downarrow_i v = \{x \in \lambda^{-1}(\Sigma_i) \mid x \leq v\}$ and $\Downarrow_i v = \downarrow_i v \setminus \{v\}$ are totally ordered. We denote by $\lambda(\Downarrow_i v)$ the word $\lambda(x_1) \cdots \lambda(x_n)$ where $\{x_1, \cdots, x_n\} = \Downarrow_i v$ with $x_1 < \cdots < x_n$. Let us fix a distributed strategy F. We say that a run $t = (V, \leq, \lambda)$ is an *F-run* (or is compatible with strategy F) if all controllable events are played according to F, i.e., for all $v \in V$ such that $\lambda(v) \in \Sigma_i^c$, we have $\lambda(v) = f_i(\lambda(\Downarrow_i v))$. Observe that, for a fixed distributed strategy F, even if inputs from the environment follow the same pattern, there are multiple F-runs depending on the scheduling of internal signals. A run $t = (V, \leq, \lambda)$ is F-maximal if for any process i, either $V_i = \lambda^{-1}(\Sigma_i)$ is infinite, or f_i is undefined on $\lambda(V_i)$.

3 The Specification

The specifications we consider only constrain *external* actions from Γ, i.e., actions that reflect communications with the environment. We want the processes to collaborate freely in order to achieve the specification, hence we do not constrain internal signals. Moreover, our specifications will be on *partial orders*, and not linearizations of executions. Indeed, specifying over interleavings allows to differentiate between equivalent linearizations, which is not desirable for distributed systems.

For a *concrete* run $t = (V, \leq, \lambda)$ we define the *abstract* (observable) run as the projection $\pi_\Gamma(t) = (\lambda^{-1}(\Gamma), \leq \cap (\lambda^{-1}(\Gamma))^2, \lambda)$. Specifications will then be formulae in some logical formalism whose models are Γ-labelled partial orders. We say that a *concrete* run t satisfies a specification φ if its projection $\pi_\Gamma(t)$ satisfies φ.

Distributed synthesis: Given an architecture $(\text{Proc}, E, (\text{In}_i)_{i \in \text{Proc}}, (\text{Out}_i)_{i \in \text{Proc}})$ and a specification φ over Γ-labelled posest in an appropriate logic, decide whether there exist internal signal sets $(\Sigma_{i,j})_{(i,j) \in E}$ and a distributed strategy F such that *every* F-maximal concrete F-run satisfies the specification φ.

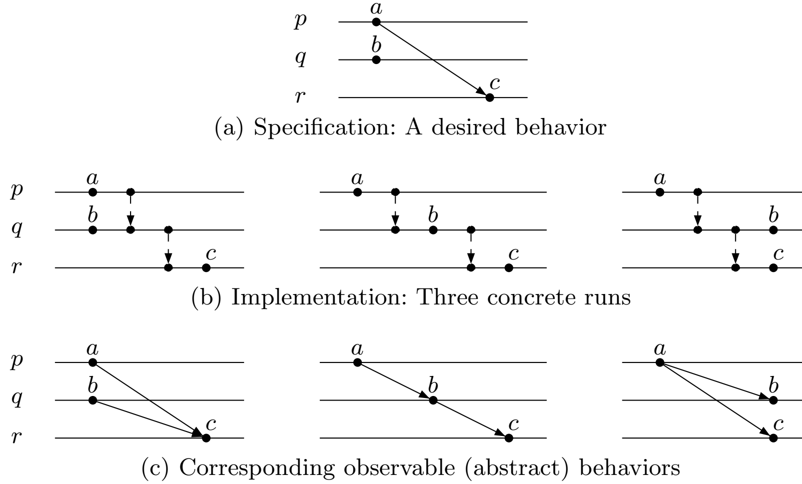


Fig. 1. Specifications must be closed under extensions

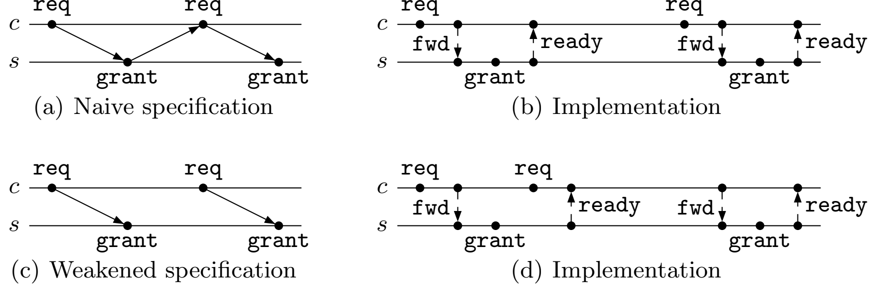
Acceptable Specifications. We explain now with some examples that not all specifications are *acceptable* in our framework. We start with an example showing that specifications must be closed under extensions of partial orders.

Consider a distributed system with $\mathrm{Proc} = \{p, q, r\}$ and $E = \{(p, q), (q, r)\}$. Note that p cannot directly send signals to r. A natural specification could be that q must output b and that if p receives input a from the environment then r must *later* output c. This corresponds to the partial order represented in Figure 1(a). In order to implement this specification, when process p receives a it must send a signal to q and q should forward this signal to r so that r knows it should output c. But these *internal* signals will induce some additional ordering between a and b or between b and c as can be seen in Figure 1(b). None of the corresponding abstract runs in Figure 1(c) correspond to the partial order of the specification, though they are all *extensions* of it. Hence, we need to *extend* this specification so that it can be implemented.

Formally, an (order) *extension* of a labelled partial order $t = (V, \leq_t, \lambda)$ is any partial order $s = (V, \leq_s, \lambda)$ with $\leq_t \subseteq \leq_s$. We will require our specifications to be closed under *extensions*.

Next, we show that the specification should also be closed under some weakenings of the partial order. This is due to the fact that inputs from the environment are uncontrollable events. Hence, it seems unrealistic to try to impose a *direct* causality between any action on some process and an input event from the environment on another process. For instance, consider an architecture with two processes, one receiving service requests from a client and the other granting the service: $\mathrm{In}_c = \{\texttt{request}\}$ and $\mathrm{Out}_s = \{\texttt{grant}\}$.

A naive specification could be an alternation of `request` and `grant` as presented in Figure 2(a). A possible implementation is presented in Figure 2(b)



(a) Naive specification (b) Implementation

(c) Weakened specification (d) Implementation

Fig. 2. Client–Server

using two internal signals `forward` $\in \Sigma_{c,s}$ and `ready` $\in \Sigma_{s,c}$. Since inputs from the environment are uncontrollable, we cannot enforce that the second `request` comes after the internal signal `ready`. Hence, the specification should also include the behavior of Figure 2(c) corresponding to the concrete run of Figure 2(d).

Formally, if a partial order $t = (V, \leq_t, \lambda)$ satisfies a specification and if $z \lessdot_t z'$ where z' is an input event from the environment and z, z' are not on the same process, then the *weakening* $s = (V, \leq_s, \lambda)$ with $\leq_s \, = \, \leq_t \setminus \{(z, z')\}$ should also satisfy the specification ($\leq_s$ is still an order relation since z' is a successor of z).

We will define the weakest partial order induced by t. Recall that actions in Γ are either inputs from the environment or outputs to the environment: $\Gamma = \mathrm{In} \cup \mathrm{Out}$ with $\mathrm{In} = \bigcup_{i \in \mathrm{Proc}} \mathrm{In}_i$ and $\mathrm{Out} = \bigcup_{i \in \mathrm{Proc}} \mathrm{Out}_i$. Consider now a Γ-labelled partial order $t = (V, \leq, \lambda)$ and define

$$\begin{aligned} W_t = \{(z, z') \in V^2 \mid \exists i \in \mathrm{Proc},\ & \lambda(z) \notin \Sigma_i \wedge \lambda(z') \in \mathrm{In}_i \wedge z < z' \\ & \wedge (\neg \exists y, \lambda(y) \in \mathrm{Out}_i \wedge z < y < z')\} \ . \end{aligned}$$

The set W_t consists of all those pairs (z, z') for which the ordering in t is fortuitous. This happens when z, z' are on different processes and z' is an uncontrollable input from the environment, except if we find an output event y between z and z' which is on the same process as z'. Indeed, output y may have been triggered by z so we do not remove orderings to output events.

We are now ready to define *acceptable* specifications.

Definition 1. *A specification is* acceptable *if it is closed under extension and weakening. Formally, a specification φ is* acceptable *if for all $t = (V, \leq_t, \lambda)$ such that $\lambda^{-1}(\Sigma_i)$ is totally ordered for all $i \in \mathrm{Proc}$, if $t \models \varphi$, then*

- *$r \models \varphi$ for all $r = (V, \leq_r, \lambda)$ with $\leq_t \, \subseteq \, \leq_r$ (extension),*
- *$s \models \varphi$ where $s = (V, \leq_s, \lambda)$ with $\leq_s \, = \, \leq_t \setminus W_t$ (weakening).*

Observe that this definition of weakening removes all fortuitous orderings at once, but, since the specification is also closed under extension, all intermediary partial orders can also be obtained.

AlocTL. Among different logics available to express specifications over partial orders, we will focus on local temporal logics (locTL), for they allow easy and intuitive specifications for distributed system, and they have a reasonable complexity. However, not all local temporal logic formulae are *acceptable*: the formula $\mathsf{EM}(a \wedge \neg \mathsf{F}\, b)$ meaning that there is a minimal a-event with no b-events in its future is not closed under extension (see e.g. [2] for a formal semantics of locTL). Also, the formula $\mathsf{EM}(a \wedge \mathsf{EX}\, c)$ meaning that there is a minimal a-event immediately followed by a c-event is not closed under extension. In fact, in order to stay in the class of specifications closed under extensions, we have to rule out any modality that *requires some concurrency* between two events. For the closure by weakening, we restrict the use of the order relation between events on different processes so that the greater event is not an input.

We introduce a *syntactic* restriction of a process based local temporal logic for which all formulae will be *acceptable*. The syntax of AlocTL(Γ, Proc) (or simply AlocTL if Γ and Proc are clear from the context) is given by:

$$\begin{aligned}\varphi ::= a \mid \neg a \mid \neg \mathsf{X}_i \top \mid \neg \mathsf{Y}_i \top \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ \mid \mathsf{X}_i\, \varphi \mid \varphi\, \mathsf{U}_i\, \varphi \mid \mathsf{G}_i\, \varphi \mid \mathsf{F}_{i,j}(\mathrm{Out} \wedge \varphi) \mid \mathsf{Y}_i\, \varphi \mid \varphi\, \mathsf{S}_i\, \varphi \mid \mathrm{Out} \wedge \mathsf{H}_{i,j}\, \varphi\end{aligned}$$

with $a \in \Gamma$ and $i, j \in \mathrm{Proc}$. The modalities X_i, Y_i, U_i and S_i are the usual *next*, *yesterday*, *until* and *since* restricted to the totally ordered events of process i. We can also express in our logic *release* (dual of *until*): $\varphi \mathsf{R}_i \psi = (\mathsf{G}_i\, \psi) \vee (\psi \mathsf{U}_i (\varphi \wedge \psi))$. When restricted to the events of some process i, our logic has the full expressive power of LTL or FO. We only restrict how one can switch from one process to another so that closure under extensions and weakenings will be obtained.

To switch from process i to process j, we use $\mathsf{F}_{i,j}$ or $\mathsf{H}_{i,j}$. The first one allows to specify a response property triggered on process i for which the output is delivered on process j, e.g., $\mathsf{G}(\mathtt{request} \longrightarrow \mathsf{F}_{i,j}(\mathrm{Out} \wedge \mathtt{grant}))$. The second modality may be used to specify that outputs should have a cause, e.g., $\mathsf{G}(\mathtt{grant} \longrightarrow (\mathrm{Out} \wedge \mathsf{H}_{j,i}\, \mathtt{request}))$. We do not include negations or modalities of the form $\mathsf{X}_{i,j}$ since they lead out of acceptable specifications.

We did not investigate the expressive power of our logic, but we believe it can express lots of interesting properties since it has the expressive power of FO when restricted to local events of each process, and allows *response* and *cause* properties between processes.

The semantics defines when $t, x \models \varphi$ where $t = (V, \leq, \lambda)$ is a Γ-labelled partial order with $V_i = \lambda^{-1}(\Sigma_i)$ totally ordered for each $i \in \mathrm{Proc}$, and $x \in V$:

- $t, x \models a \in \Gamma$ if $\lambda(x) = a$
- $t, x \models \mathsf{X}_i\, \varphi$ if $x \in V_i$ and $t, y \models \varphi$ for some $y \in V_i$ such that $x < y$ and for all $z \in V_i$, $z \leq x$ or $y \leq z$.
- $t, x \models \mathsf{G}_i\, \varphi$ if $x \in V_i$ and $t, y \models \varphi$ for all $y \in V_i$ such that $x \leq y$.
- $t, x \models \varphi\, \mathsf{U}_i\, \psi$ if $x \in V_i$ and $t, y \models \psi$ for some $y \in V_i$ such that $x \leq y$ and for all $z \in V_i$, $x \leq z < y$ implies $t, z \models \varphi$.
- $t, x \models \mathsf{F}_{i,j}(\varphi \wedge \mathrm{Out})$ if $x \in V_i$ and $t, y \models \varphi$ for some $y \in V_j$ such that $x \leq y$ and $\lambda(y) \in \mathrm{Out}$.

The other modalities are defined similarly.

As in [2], we have chosen to introduce initial formulae to address the problem of starting the evaluation of a formula. Those are defined by the syntax

$$\alpha ::= \bot \mid \top \mid \neg\, \mathsf{EM}_i\, \top \mid \mathsf{EM}_i\, \varphi \mid \alpha \vee \alpha \mid \alpha \wedge \alpha$$

where φ is a formula of AlocTL. The semantics is given by $t \models \mathsf{EM}_i\, \varphi$ if $t, x \models \varphi$ where x is the minimal vertex of V_i.

Proposition 1. *The logic* AlocTL *is closed under extension and weakening.*

Observe that AlocTL is a natural fragment of FO(<) which is closed under extensions and weakenings. In our setting, it provides a convenient way to specify desired properties. Our decidability results will be stated for AlocTL but they would still hold for more general logics defining regular properties which are closed under extensions and weakenings.

4 Decidability Results

In this section we solve the synthesis problem for the subclass of architectures having a strongly connected communication graph: every process can send signals to everyone (though maybe not directly). In the following, we will simply call them strongly connected architectures.

Singleton Architectures. A first step in solving the general problem is to handle the sequential case. This problem is slightly different from the asynchronous synthesis of [16] (where the communication was through shared variables) and [12] (where a single process does not evolve asynchronously with respect to its environment).

In the sequential case, there is no internal action and then $\Sigma = \Gamma = \mathrm{In} \cup \mathrm{Out}$, and all runs are total orders. The only specificity is that the system communicates asynchronously with the environment, i.e., there may be several signals from the environment before the process has a chance to play, and reciprocally. Since there is no possible weakening or extension, we are concerned with classical logics for specifications. We can deal both with linear time specifications (LTL, FO, MSO) or with branching time specifications (CTL*, μ-calculus) since all we need is regular specifications. With slight modifications of the proof technique used in [6], we obtain the following result.

Theorem 1. *The synthesis problem over the singleton architecture is decidable for regular specifications.*

Strongly Connected Architectures. Now, we show that the distributed synthesis problem is decidable for the whole subclass of strongly connected architectures. This is done by reduction to the synthesis problem over the singleton. On one hand, it is easy to simulate a distributed strategy with a sequential one. Conversely, when given a program for the singleton that produces only runs satisfying

the specification, we can distribute it over the strongly connected architecture. We use a master-slave algorithm: we centralize the information by making all processes forward their local histories to a master process that takes all decisions about which action to output. This master process consequently sends back orders to the other processes, based on information it has and the given sequential strategy. Formally, we will prove the following main result.

Theorem 2. *The distributed synthesis problem over strongly connected architectures is decidable for* AlocTL *specifications.*

The rest of the section is devoted to the proof of this theorem. Let $\mathcal{A} = (\mathrm{Proc}, E, (\mathrm{In}_i)_{i\in\mathrm{Proc}}, (\mathrm{Out}_i)_{i\in\mathrm{Proc}})$ be an architecture with (Proc, E) strongly connected. Let $\mathcal{S}$ be an architecture with a single process p and external signals $\mathrm{In}_p = \mathrm{In} = \bigcup_{i\in\mathrm{Proc}} \mathrm{In}_i$ and $\mathrm{Out}_p = \mathrm{Out} = \bigcup_{i\in\mathrm{Proc}} \mathrm{Out}_i$. We show that the distributed synthesis problem for $\varphi \in \mathrm{AlocTL}(\Gamma, \mathrm{Proc})$ over $\mathcal{A}$ can be reduced to the synthesis problem for an associated specification $\overline{\varphi} \in \mathrm{AlocTL}(\Gamma, \{p\})$ over $\mathcal{S}$. Then, we obtain Theorem 2 from Theorem 1. We have to change the specification since there is a single process in $\mathcal{S}$ and several processes in $\mathcal{A}$. We do so in such a way that for all Γ-labelled *total* order t and all $x \in t$, we have $t, x \models \varphi$ if and only if $t, x \models \overline{\varphi}$. For instance, $\overline{\mathsf{X}_i\,\varphi} = \Sigma_i \wedge \mathsf{X}((\neg\Sigma_i)\,\mathsf{U}\,(\Sigma_i \wedge \overline{\varphi}))$ and $\overline{\mathsf{F}_{i,j}(\mathrm{Out} \wedge \varphi)} = \Sigma_i \wedge \mathsf{F}(\mathrm{Out}_j \wedge \overline{\varphi})$ (where $A = \bigvee_{a\in A} a$ for $A \subseteq \Sigma$). The following two propositions state that the synthesis problem for φ over $\mathcal{A}$ is reduced to the synthesis problem for $\overline{\varphi}$ over $\mathcal{S}$.

Proposition 2. *If there are internal signals sets and a distributed winning strategy for φ over $\mathcal{A}$, then there is a winning strategy for $\overline{\varphi}$ over $\mathcal{S}$.*

The proof of this proposition is omitted due to lack of space.

Proposition 3. *If there is a winning strategy for $\overline{\varphi}$ over $\mathcal{S}$ then one can define internal signals sets and a distributed winning strategy for φ over $\mathcal{A}$.*

Proof (Sketch). We want to simulate a sequential run of $\mathcal{S}$ in the distributed system $\mathcal{A}$. Due to uncontrollable inputs from the environment, we cannot avoid some concurrency but we will restrict it as much as possible so that runs of $\mathcal{A}$ will be *weakenings* of sequential runs of $\mathcal{S}$. To do so, we select a cycle in the communication graph and force the processes to communicate in a sequential way through this virtual ring – note that there may be no simple cycle, and some technical details arise when a process appears several times in the ring. For simplicity, we present here the proof assuming there is a simple cycle in (Proc, E) and we rename the processes $\mathrm{Proc} = \{1, \ldots, n\}$ according to this cycle. Process 1 will be our master.

Let f be a winning strategy for $\overline{\varphi}$ over $\mathcal{S}$. To simulate f over $\mathcal{S}$, the master Process 1 transforms its local history σ into a compatible sequential history $\psi(\sigma)$ of $\mathcal{S}$ (the definition of ψ will be given later).

If f is undefined on $\psi(\sigma)$ then we let $f_1(\sigma) = (\mathrm{Msg}_1, \varepsilon)$ meaning that Process 1 wants to initiate a round collecting inputs received by other Processes. When receiving this signal, Process 2 sends a pair (Msg_2, τ) where $\tau \in \mathrm{In}_1^*$ is the

sequence of inputs received by Process 1 since the last time it has sent a signal to Process 2. The round continues similarly for the other Processes. Formally, for $1 < i \le n$, $\sigma \in \Sigma_i^* \cdot \Sigma_i^c \cup \{\varepsilon\}$, $\tau \in \mathrm{In}^*$ and $\tau_1, \tau_2 \in \mathrm{In}_i^*$, we let

$$f_i(\sigma \cdot \tau_1 \cdot (\mathrm{Msg}_{i-1}, \tau) \cdot \tau_2) = (\mathrm{Msg}_i, \tau \cdot \tau_1 \cdot \tau_2).$$

Here τ is the sequence of inputs collected by previous processes, τ_1 consists of inputs received before the signal Msg_{i-1} and τ_2 of the messages received after Msg_{i-1} and before Msg_i could be sent. This explains the reordering $\tau \cdot \tau_1 \cdot \tau_2$.

Assume now that f is defined on $\psi(\sigma)$ and let i with $a = f(\psi(\sigma)) \in \mathrm{Out}_i$. If $i = 1$ then we simply let $f_1(\sigma) = f(\psi(\sigma))$. Now, if $i > 1$ then we let $f_1(\sigma) = (\mathrm{Ord}_1, a)$ to transmit to Process i the order to output a. The order is forwarded by each intermediary Process $1 < j < i$ *only if j received no inputs since the last time it has sent a signal to $j + 1$*. Then output a is performed by i and an acknowledgement is sent to Process 1. This acknowledgement will also collect inputs received by remaining processes. Formally, for $1 < i \le n$, $\sigma \in \Sigma_i^* \cdot \Sigma_i^c \cup \{\varepsilon\}$, $\tau \in \mathrm{In}^*$ and $\tau_1, \tau_2 \in \mathrm{In}_i^*$, we let

$$\begin{aligned}
f_i(\sigma \cdot (\mathrm{Ord}_{i-1}, a)) &= (\mathrm{Ord}_i, a) && \text{if } a \notin \mathrm{Out}_i \\
f_i(\sigma \cdot (\mathrm{Ord}_{i-1}, a)) &= a && \text{if } a \in \mathrm{Out}_i \\
f_i(\sigma \cdot (\mathrm{Ord}_{i-1}, a) \cdot a \cdot \tau_2) &= (\mathrm{Ack}_i, \tau_2) && \text{if } a \in \mathrm{Out}_i \\
f_i(\sigma \cdot \tau_1 \cdot (\mathrm{Ack}_{i-1}, \tau) \cdot \tau_2) &= (\mathrm{Ack}_i, \tau \cdot \tau_1 \cdot \tau_2)
\end{aligned}$$

Now, if an intermediary process received some inputs from the environment before it could forward the order to Process i, then the basis on which Process 1 took is decision is no longer valid. Hence, we have to abort the order and signal this fact with a Nack. We also need to abort if i received the order but has also received inputs from the environment before it could execute the order. As above, Nack also collects inputs received by the remaining processes. Formally, for $1 < i \le n$, $\sigma \in \Sigma_i^* \cdot \Sigma_i^c \cup \{\varepsilon\}$, $\tau \in \mathrm{In}^*$ and $\tau_1, \tau_2 \in \mathrm{In}_i^*$, we let

$$\begin{aligned}
f_i(\sigma \cdot \tau_1 \cdot (\mathrm{Ord}_{i-1}, a) \cdot \tau_2) &= (\mathrm{Nack}_i, \tau_1 \cdot \tau_2) && \text{if } \tau_1 \cdot \tau_2 \neq \varepsilon \\
f_i(\sigma \cdot \tau_1 \cdot (\mathrm{Nack}_{i-1}, \tau) \cdot \tau_2) &= (\mathrm{Nack}_i, \tau \cdot \tau_1 \cdot \tau_2)
\end{aligned}$$

The sets of internal signals are implicitely defined by the strategies above: $\Sigma_{1,2} = (\{\mathrm{Ord}_1\} \times \mathrm{Out}) \cup \{(\mathrm{Msg}_1, \varepsilon)\}$ and for $1 < i \le n$ and $j = 1 + (i \mod n)$,

$$\Sigma_{i,j} = (\{\mathrm{Msg}_i, \mathrm{Ack}_i, \mathrm{Nack}_i\} \times \mathrm{In}^*) \cup (\{\mathrm{Ord}_i\} \times \mathrm{Out}).$$

Due to In^*, the sets $\Sigma_{i,j}$ are infinite. We explain in Remark 1 how to reduce to finite sets of signals, and strategies with finite-memories.

To conclude the construction, we define the map $\psi : \Sigma_1^* \to \Gamma^*$ by induction. First, $\psi(\varepsilon) = \varepsilon$. Next, $\psi(\sigma \cdot b) = \psi(\sigma) \cdot b$ for $\sigma \in \Sigma_1^*$ and $b \in \Sigma_1 \cap \Gamma$. Finally, for $\sigma \in \Sigma_1^*$, $a \in \mathrm{Out} \setminus \mathrm{Out}_1$, $\tau \in \mathrm{In}_1^*$ and $\tau' \in \mathrm{In}^*$, let:

$$\begin{aligned}
\psi(\sigma \cdot (\mathrm{Msg}_1, \varepsilon) \cdot \tau \cdot (\mathrm{Msg}_n, \tau')) &= \psi(\sigma) \cdot \tau' \cdot \tau \\
\psi(\sigma \cdot (\mathrm{Ord}_1, a) \cdot \tau \cdot (\mathrm{Ack}_n, \tau')) &= \psi(\sigma) \cdot a \cdot \tau' \cdot \tau \\
\psi(\sigma \cdot (\mathrm{Ord}_1, a) \cdot \tau \cdot (\mathrm{Nack}_n, \tau')) &= \psi(\sigma) \cdot \tau' \cdot \tau
\end{aligned}$$

Note that, after sending Msg_1 or Ord_1, ψ is undefined until the corresponding Msg_n, Ack_n or Nack_n has been received by Process 1. When ψ is undefined then f_1 is also undefined so that Process 1 waits for the end of the round. Note also that inputs in τ' may have been received before those in τ.

Let $t = (V, \leq_t, \lambda)$ be an F-maximal F-run. We can easily check that all output events in $\lambda^{-1}(\text{Out})$ are totally ordered. We can also show that the history t' computed by ψ is an f-maximal f-run which is a linear extension of $\pi_\Gamma(t)$. To conclude the proof, it remains to show that $\pi_\Gamma(t)$ is an extension of the weakening of t': $\leq_{t'} \setminus W_{t'} \subseteq \leq_t \subseteq \leq_t$. For this, we will use the following claim whose proof is omitted for lack of space.

Claim. For all $x, y \in \lambda^{-1}(\Gamma)$ such that $x \parallel_t y$, if $x <_{t'} y$ then $\lambda(y) \in \text{In}$.

So let $z, z' \in \lambda^{-1}(\Gamma)$ with $z <_{t'} z'$ and $z \parallel_t z'$. We have to show that $(z, z') \in W_{t'}$. By the above claim, we get $\lambda(z') \in \text{In}$. Let $i \in \text{Proc}$ be such that $\lambda(z') \in \text{In}_i$. Since $z \parallel_t z'$ we deduce $\lambda(z) \notin \Sigma_i$. Now, let $y \in \lambda^{-1}(\Gamma)$ be such that $z <_{t'} y <_{t'} z'$. Clearly, $z \parallel_t z'$ implies $z \parallel_t y$ and we deduce $\lambda(y) \in \text{In}$ by the claim stated above. Therefore, $(z, z') \in W_{t'}$. Finally, $t' \models \varphi$ since the strategy f is winning. We deduce that $\pi_\Gamma(t) \models \varphi$ since our specification logic is closed under weakenings and extensions. ☐

Remark 1. As they are defined, the sets $(\Sigma_{i,j})_{(i,j)\in E}$ are infinite, and the strategies for the distributed architecture need unbounded memory. However, it is possible to modify them to use only finite signal sets and strategies with finite memories. Recall that the strategy of our master process is essentially based on the strategy f of the singleton. As usual in the sequential case, when there is a winning strategy, then there is also a winning strategy using a finite memory which can be described by a deterministic finite automaton $\mathcal{M} = (Q, \Gamma, \delta, q_0, f)$ with $f : Q \to \text{Out}$. Consequently, each slave process may compute the transition function $\delta_\tau \in Q^Q$ associated with a sequence τ of inputs it has received and transmit δ_τ instead of τ. Therefore, we get finite sets of internal signals

$$\Sigma_{i,1+(i \bmod n)} = (\{\text{Msg}_i, \text{Ack}_i, \text{Nack}_i\} \times Q^Q) \cup (\{\text{Ord}_i\} \times \text{Out}).$$

and the memory needed by each process $1 < i \leq n$ is Q^Q. It is then easy to adapt the proof of Proposition 3.

5 Conclusion

In this paper, we have defined a new setting for the synthesis problem for distributed asynchronous systems, and proved that it is decidable for an interesting subclass of architectures. We believe that using signals in asynchronous systems, and restricting to acceptable specifications will help to overcome a lot of the common difficulties that usually lead to undecidability results.

Future work includes the generalization of our decidability result to larger classes of architectures. Other interesting problems would be to study the expressivity of AlocTL or to define other logics for acceptable specifications.

References

1. A. Church. Logic, arithmetics, and automata. In *Proc. of Int. Symp. of Mathematicians*, pages 23–35, 1962.
2. V. Diekert and P. Gastin. Local temporal logic is expressively complete for cograph dependence alphabets. In *Proc. of LPAR'01*, volume 2250 of *LNCS*, pages 55–69. Springer, 2001.
3. B. Finkbeiner and S. Schewe. Synthesis of asynchronous systems. In *Proc. of LOPSTR'06*, volume 4407 of *LNCS*, pages 127–142. Springer, 2006.
4. P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *Proc. of FSTTCS'04*, volume 3328 of *LNCS*, pages 275–286. Springer, 2004.
5. P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. In *Proc. of FSTTCS'06*, volume 4337 of *LNCS*, pages 321–332. Springer, 2006.
6. O. Kupferman, P. Madhusudan, P. S. Thiagarajan, and M. Y. Vardi. Open systems in reactive environments: Control and synthesis. In *Proc. of CONCUR'00*, volume 1877 of *LNCS*, pages 92–107. Springer, 2000.
7. O. Kupferman and M. Y. Vardi. Church's problem revisited. *Bull. Symbolic Logic*, 5(2):245–263, 1999.
8. O. Kupferman and M. Y. Vardi. μ-calculus synthesis. In *Proc. of MFCS'00*, volume 1893 of *LNCS*, pages 497–507. Springer, 2000.
9. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proc. of LICS'01*. Computer Society Press, 2001.
10. N. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, 1989.
11. P. Madhusudan and P. S. Thiagarajan. Distributed control and synthesis for local specifications. In *Proc. of ICALP'01*, volume 2076 of *LNCS*, pages 396–407. Springer, 2001.
12. P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In *Proc. of CONCUR'02*, volume 2421 of *LNCS*, pages 145–160. Springer, 2002.
13. P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 201–212. Springer, 2005.
14. S. Mohalik and I. Walukiewicz. Distributed games. In *Proc. of FSTTCS'03*, volume 2914 of *LNCS*, pages 338–351. Springer, 2003.
15. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of POPL'89*, pages 179–190, 1989.
16. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. of ICALP'89*, volume 372 of *LNCS*, pages 652–671. Springer, 1989.
17. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of FOCS'90*, volume II, pages 746–757. IEEE Press., 1990.