A Decidable Class of Asynchronous Distributed Controllers

P. Madhusudan[1]* and P.S. Thiagarajan[2]

[1] Dept. of Computer and Information Science, Univ. of Pennsylvania
madhusudan@saul.cis.upenn.edu
[2] Department of Computer Science, National University of Singapore
thiagu@comp.nus.edu.sg

Abstract. We study the problem of synthesizing controllers in a natural distributed asynchronous setting: a finite set of plants interact with their local environments and communicate with each other by synchronizing on common actions. The controller-synthesis problem is to come up with a *local* strategy for each plant such that the controlled behaviour of the network meets a specification. We consider linear time specifications and provide, in some sense, a minimal set of restrictions under which this problem is effectively solvable: we show that the controller-synthesis problem under these restrictions is decidable while the problem becomes undecidable if any one or more of these three restrictions are dropped.

1 Introduction

An *open* reactive system interacts with its environment in a sustained fashion and its behaviour crucially depends on this interaction. The key feature of such a system is that one is required to distinguish between the capabilities of the system and its environment. In particular, while the actions performed by the system can be controlled, actions performed by the environment— typically modelling inputs— are uncontrollable. The *controller synthesis* problem arises naturally in such a setting: One starts with an open system— often called a *plant* in this context— and a specification of the desired behaviour of the plant. In this paper, we will be concerned only with *linear time* specifications which typically describe a language of action sequences. The goal now is come up with a strategy to control the actions of the plant so that the overall behaviour meets the specification. Thus the strategy will serve as a *controller* which restricts the behaviour of the plant— and not that of the environment— so that the controlled sequences satisfy the specification.

There is a wealth of literature on this problem and the related problem of *realizability* (which we shall not be concerned with here) for both linear time specifications and branching time specifications [Chu63,BL69,ALW89,PR89a,KV99,KV00]. However, all these studies are carried out in a *sequential* setting. Controller synthesis problems, however, arise

* Supported in part by NSF awards CCR99-70925 and ITR/SY 0121431

naturally in a distributed framework and here the main results in the literature are the ones due to Pnueli and Rosner [PR90] and some recent extensions [KV01,MT01]. Phrased in the context of the present paper— and in line with the results reported in [MT01]— Pnueli and Rosner exploit the notion of an *architecture* which basically consists of a set of sites and some communication channels between them. In the control-synthesis problem, each site has a plant associated with it which can interact with its local environment. The plants evolve together in lock-step synchronous fashion. One is also given a specification of the action sequences that this network of sites is permitted to generate. The problem is to come with a local strategy at each site so that the distributed controlled behaviour meets the specification. By a local strategy we mean one which knows only about the action sequence that the site has participated in (and any information that it may have thus gained via the communications). The surprising fact is that the controller synthesis problem is undecidable for almost all architectures [PR90,MT01,Mad01]. In [KV01], the authors extend the small class of acyclic decidable architectures to related cyclic architectures.

In this paper, we study the distributed controller synthesis problem in a more general setting, where the processes at the sites can evolve asynchronously with respect to each other. (The framework of [PR89b] is different; there, the interaction between a *global* plant and its environment is asynchronous.) The sites interact with their environments locally and communicate with each other by synchronizing on common actions. We are also given a linear time specification in terms of a regular set of action sequences that this network of sites is allowed to generate. The problem is to come up with local strategies for the sites so that the controlled distributed behaviour meets the specification.

Our main result in this setting is that we can provide a minimal set of restrictions under which the controller synthesis problem becomes decidable. We impose three restrictions R1, R2 and R3 ; while R1 is a restriction on the specification, R2 and R3 have to do with the nature of the local strategies that are permitted.

In the distributed setting we consider, an execution of the network can be naturally viewed as a labelled partial order, rather than as a linear order (i.e. a word). Roughly speaking if there is an execution of the network after which two actions a and b occur, and the processes that are involved with a and b are disjoint, then the actions are not causally dependent on each other and can be viewed as occuring in no particular order. The labelled partial order reflects the events and their causal relation.

R1 demands that the specification must be *robust* in that if one linearization of an execution is in the specification then all its linearizations must also be. Such robust specifications are well studied and there are a number of temporal logics developed in the literature that are, by design, robust (the survey article [HT98] provides a detailed summary). Here, for convenience, we shall be using ω-automata instead of temporal logics to capture specifications, with or without the restriction R1.

Turning now to the other two restrictions, R2 demands that each local strategy can only keep track of the local time and not the local history. More precisely, the strategy can only remember the *length* of the local history but not the history itself. Such a strategy will then be a function of the length of the local history *and* of course the current local state of the plant.

Finally R3 requires that the local strategies follow a *rigid* communication pattern. In other words, at a given stage in the computation, a local strategy cannot recommend two synchronizations that involve two different sets of sites; the synchronization actions recommended must all involve the same set of sites.

Our main positive result is that, if the specification satisfies R1, then determining if there exists a controller that satisfies R2 and R3 is decidable for *all* architectures. Further, whenever a controller exists, we show that we can synthesize a finite-state one. Our main negative result is that if *any* one (or more) of these restrictions is dropped then the corresponding controller synthesis problem is undecidable. For instance, suppose one does not impose any condition on the specifications except that it is regular (i.e. we drop restriction R1). Then determining if there exists a distributed controller is undecidable, whether or not we impose conditions R2 and/or R3. Similarly, we show that we cannot drop the conditions R2 or R3 and retain decidability. It is in this sense R1, R2 and R3 form a minimal set of restrictions in whose presence the controller synthesis problem can be effectively solved in a natural distributed asynchronous framework.

The most severe restriction of the above is R2 which requires that strategies are "clocked" — i.e. that they can only remember the length of the history and not the history itself. We can drop the restriction R2 and instead parameterize the control problem with a number b and require that strategies, in addition to the lengths of the local histories, use no more than b bits to remember information about the local past histories. We believe that this "graded" relaxation of R2 will frequently and quickly yield through an iterative process, a valid controller that meets the specification. However, since the introduction of this bounded memory complicates the technical presentation, we shall not deal with it here except for some additional remarks in the concluding section.

We believe our positive results are relevant for synthesizing distributed controllers for discrete event systems (DES). The objective here is to synthesize a controller for each agent (interacting with its environment) in the plant such that the controlled plant meets the specification. The requirement that the controllers are local and communicate with each other but evolve at different speeds is often essential when modelling spatially separate agents (for example, robots with embedded local control software).

Our results are also relevant to the setting of [PR90], where the model is a synchronous one and all the sites operate in a lock-step fashion. Since the communication pattern can be fixed in such a setting, condition R3 has no effect. Also, one can add a process that acts as a global observer by synchronizing with all the other processes. The specification can then be stated as a local specification (which is always robust) of this observer, and hence we can drop restriction

R1 as well. As a result, in the [PR90] framework, the controller synthesis problem is undecidable for almost all architectures if one imposes no restrictions and it becomes decidable for *all* architectures if we demand R2.

Turning briefly to the proof of decidability, the key insight is that in the presence of R2 and R3, our strategies can be encoded as restricted labelled partial orders called *Mazurkiewicz traces* which have a rich theory [DR95]. We handle these partial orders here via their linearizations. We show then that we can design an automaton to check whether a strategy (encoded as linearizations of a trace) is winning or not. The major technical hurdle is that we need to ensure that the choice of the linearization used to represent strategies does not interfere with the correctness of the decision procedure, and this crucially requires the fact that the specification satisfies R1.

The key technique for all the undecidability results is a reduction borrowed from [PR90] (see also [PR79]) with some insights provided by the undecidability arguments developed in [MT01,Mad01].

2 The Problem Setting

We fix a finite set of processes $\mathcal{P}$ for the rest of the paper and let p, q range over $\mathcal{P}$. A distributed alphabet over $\mathcal{P}$ is a pair (Σ, θ) where Σ is a finite alphabet of actions and $\theta : \Sigma \to 2^{\mathcal{P}} \setminus \{\emptyset\}$ identifies for each action, a nonempty set of processes that take part in each execution of the action. Let $\Sigma_p = \{a \in \Sigma \mid p \in \theta(a)\}$ denote the set of actions the process p partakes in. Fix such a distributed alphabet (Σ, θ) for the rest of this section.

Definition 1. *A* product system $\mathcal{S}$ *over* (Σ, θ) *is a family of transition systems,* $\{(S_p^E, S_p^C, \Sigma_p^E, \Sigma_p^C, \to_p, s_{in}^p)\}_{p \in \mathcal{P}}$*, where for each process* $p \in \mathcal{P}$*:*

- $S_p = S_p^E \uplus S_p^C$ *is the finite set of local states of* p*, with* S_p^E *denoting* environment states *and* S_p^C *denoting* controller states. S_p^E *and* S_p^C *are disjoint;*
- Σ_p^E *and* Σ_p^C *form a partition of* Σ_p *and denote the* environment actions *and* controllable actions*, respectively. We also require that environment actions are local— i.e. for every* $e \in \Sigma_p^E$, $\theta(e) = \{p\}$*;*
- $\to_p \subseteq (S_p^E \times \Sigma_p^E \times S_p^C) \cup (S_p^C \times \Sigma_p^C \times S_p^E)$ *is the deterministic localized transition relation (i.e. if* $(s, a, s'), (s, a, s'') \in \to_p$*, then* $s' = s''$*). Note that the system alternates between system states and environment states;*
- $s_{in}^p \in S_p^C$ *is the local initial state and is a controller state.* □

A product system is hence a set of local systems, one for each process, where each local system alternately performs controllable actions and environment actions. The coordination among the processes is achieved by synchronizing on common actions as dictated by the distribution function θ. We write $s \xrightarrow{a}_p s'$ to denote $(s, a, s') \in \to_p$. A state in S_p is called a p-state. A state $s \in \bigcup_{p \in \mathcal{P}} S_p$ is called a *local state*. For a p-state $s \in S_p$, let $en(s)$ denote the set of actions locally enabled at s— i.e. $en(s) = \{a \in \Sigma_p \mid \exists s' \in S_p, s \xrightarrow{a}_p s'\}$.

A global state of a product system $\mathcal{S}$ is a function $\mathbf{s} : \mathcal{P} \rightarrow \bigcup_{p \in \mathcal{P}} S_p$ such that $\mathbf{s}(p) \in S_p$, for each $p \in \mathcal{P}$. Let $S_{\mathcal{P}}$ be the set of all global states. The *global system* associated with a product system $\mathcal{S}$ is the transition system $\mathcal{G}_{\mathcal{S}} = (S_{\mathcal{P}}, \Sigma, \Rightarrow, \mathbf{s}_{in})$ where $\mathbf{s}_{in} \in S_{\mathcal{P}}$ is the global initial state with $\mathbf{s}_{in}(p) = s^p_{in}$, for each $p \in \mathcal{P}$. As usual, the transition relation $\Rightarrow \subseteq S_{\mathcal{P}} \times \Sigma \times S_{\mathcal{P}}$ is given by: $(\mathbf{s}, a, \mathbf{s}') \in \Rightarrow$ iff $\mathbf{s}(p) \xrightarrow{a}_p \mathbf{s}'(p)$, for every $p \in \theta(a)$ and $\mathbf{s}(p') = \mathbf{s}'(p')$, for every $p' \notin \theta(a)$.

Example 1. Consider the *dining-philosophers problem*. There are n philosophers (processes) P_i, $0 \leq i < n$ in a ring and there are n forks f_i, $0 \leq i < n$ — one fork between each neighbouring pair of processes. The philosophers can be thinking and when they get hungry, they need both forks (look upon these as resources) to eat. When they finish eating, they go back to the thinking state.

This problem can be modelled by having $2n$ processes, one for each philosopher and one for each fork — say $\{P_i, f_i \mid 0 \leq i < n\}$. The philosopher P_i, when thinking, can go into a state of hunger at any time, and send requests for the neighbouring forks f_{i-1} and f_i, in any order (addition and subtraction are modulo n). When (s)he gets both, (s)he goes into a state where (s)he eats and after finishing, goes back to the thinking state. We model this by having:

$\Sigma_{P_i} = \{\mathit{hungry}_i, \mathit{reqf}_{(i,i)}, \mathit{reqf}_{(i,i-1)}, \mathit{grf}_{(i,i)}, \mathit{grf}_{(i,i-1)}, \mathit{eat}_i, \mathit{finish}_i, \tau_i, \beta_i\}$ and
$\Sigma_{f_i} = \{\mathit{reqf}_{(i,i)}, \mathit{reqf}_{(i+1,i)}, \mathit{grf}_{(i,i)}, \mathit{grf}_{(i+1,i)}, \tau'_i\}$

The action $\mathit{reqf}_{(i,j)}$ is interpreted as the i'th philosopher requesting the j'th fork and $\mathit{grf}_{(i,j)}$ means the i'th philosopher is granted the j'th fork.

We can now have a transition system, for each philosopher, that models the philosopher getting hungry, requesting for his left and right forks (he can request in any order), getting them, eating and going back to the thinking state. Forks are also modelled as processes that get requests and grant them to philosophers requesting them. The environment actions are the local actions of the kind hungry_i, finish_i, τ_i and τ'_i. The τ_i and τ'_i actions are "dummy" actions used to interleave system actions with environment actions. Similarly, β_i actions are dummy system actions. We shall return to this example after introducing strategies.

Let Σ^* denote the set of finite words over Σ, Σ^ω denote the set of infinite words over Σ and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. For a word $\alpha \in \Sigma^\omega$ and $i, j \in \mathbb{N}$, let $\alpha[i]$ denote the i^{th} letter of α and $\alpha[i,j]$ denote the subsequence $\alpha[i]\alpha[i+1]\ldots\alpha[j]$. This notation is extended to finite words as well, for appropriate values of i and j. Also, for a language of finite words $L \subseteq \Sigma^*$, let L^* and L^ω denote the languages obtained by finite and infinite iterations of words in L. These notations are transported to other alphabets as well in the obvious way.

The distributed alphabet (Σ, θ) induces a natural equivalence relation over words over Σ. For a word $\alpha \in \Sigma^\infty$ and $\Sigma' \subseteq \Sigma$, let $\alpha \restriction \Sigma'$ denote the projection of α to the letters in Σ'. By abuse of notation, let $\alpha \restriction p$ denote $\alpha \restriction \Sigma_p$. The *Mazurkiewicz trace relation* (or just *trace relation*) associated with (Σ, θ) is the relation $\approx \subseteq \Sigma^\infty \times \Sigma^\infty$ given by $\alpha \approx \beta$ iff for every $p \in \mathcal{P}$, $\alpha \restriction p = \beta \restriction p$. We say a language $L \subseteq \Sigma^\infty$ is *trace-closed* if for every $\alpha \in L$ and $\beta \in \Sigma^\infty$ such that $\alpha \approx \beta$, we have $\beta \in L$. This is just a way of specifying a Mazurkiewicz trace language where a trace is a $\approx$-equivalence class or equivalently a suitably axiomatized Σ-labelled poset (see [DR95]).

A transition system over Σ is a tuple $TS = (S, \Sigma, \delta, s_{in})$ where S is a finite set of states, $s_{in} \in S$ is the initial state and $\delta \subseteq S \times \Sigma \times S$ is the transition relation. TS is said to be total if for every $s \in S$, $a \in \Sigma$, there is at least one transition of the form $(s, a, s') \in \delta$. For a word $\alpha \in \Sigma^\omega$, a run of a transition system $TS = (S, \Sigma, \delta, s_{in})$ on α is a sequence of states $\rho = s_1, s_2, \ldots$ such that $s_1 = s_{in}$ and $\forall i \in \mathbb{N}$, $s_i \xrightarrow{\alpha[i]} s_{i+1}$. Runs of TS on finite words are defined similarly. The set of infinite words over which TS has a run is called the language of TS (denoted $\mathcal{L}(TS)$).

For a product system $\mathcal{S}$ over (Σ, θ), the language of infinite words of $\mathcal{S}$ is the language of the global system associated with $\mathcal{S}$. In other words, $\mathcal{L}(\mathcal{S}) = \mathcal{L}(\mathcal{G}_S)$. It is easy to see that $\mathcal{L}(\mathcal{S})$ is trace-closed with respect to the equivalence relation induced by (Σ, θ).

A *plant* $\mathcal{S}$ is a product system over (Σ, θ), $\mathcal{S} = \{(S_p^E, S_p^C, \Sigma_p^E, \Sigma_p^C, \rightarrow_p, s_{in}^p)\}_{p \in \mathcal{P}}$. Let us fix such a plant. For $p \in \mathcal{P}$, an *advice function* for p is a map $\xi_p : S_p \rightarrow 2^{\Sigma_p}$ such that $\xi_p(s) \subseteq en(s)$, for each $s \in S_p$, and further, if $s \in S_p^E$, then $\xi_p(s) = en(s)$. The advice function thus recommends a subset of the enabled actions at each state s in S_p, while recommending all the enabled actions at s, if s is an environment state. Let Δ_p denote the set of advice functions for p.

For $p \in \mathcal{P}$, a *local strategy for p* is a function $f_p : \Sigma_p^* \rightarrow \Delta_p$. If p executes a sequence of actions $w \in \Sigma_p^*$ and reaches a state $s \in S_p$, the strategy recommends that the actions in $(f_p(w))(s)$ be offered as possible moves by the plant. A *controller* (or *strategy*) is a set of local strategies, $Str = \{f_p\}_{p \in \mathcal{P}}$, where each f_p is a local strategy for p.

Let us fix a strategy $Str = \{f_p\}_{p \in \mathcal{P}}$ for $\mathcal{S}$. A word $\alpha \in \Sigma^\infty$ is said to be a *play according to* the strategy Str if there is a run $\rho = \mathbf{s}_1, \mathbf{s}_2, \ldots$ of $\mathcal{G}_\mathcal{S}$ on α such that for every $0 \leq i < |\alpha|$:

$$\alpha[i+1] \in f_p(\alpha[1,i] \restriction p)(\mathbf{s}_{i+1}(p)), \text{ for every } p \in \theta(\alpha[i+1])$$

Thus a play under a strategy is just a word that can be executed by the plant when it respects the moves suggested by the strategy. At any stage the strategies of the components that take part in an action are consulted and the move can be made provided all of them are willing to make the move. Note that the strategy for a process p "knows" only the history of local events $(\alpha[1,i] \restriction p)$ that have taken place. We denote the set of finite plays and infinite plays according to Str as $plays_\mathcal{S}^f(Str)$ and $plays_\mathcal{S}(Str)$, respectively. It is easy to see that these sets are in fact trace-closed, with respect to (Σ, θ).

A strategy Str for $\mathcal{S}$ is said to be *non-blocking* if every finite play can be extended— i.e. $\forall x \in plays_\mathcal{S}^f(Str), \exists a \in \Sigma : xa \in plays_\mathcal{S}^f(Str)$.

A local strategy $f_p : \Sigma_p^* \rightarrow \Delta_p$ is said to be *finite-state* if there is a deterministic total transition system $TS_{f_p} = (R, \Sigma_p, \delta, r_{in})$ and a function $h : R \rightarrow \Delta_p$ such that for any $w \in \Sigma_p^*$, if the state reached by TS_{f_p} on w is r, then $h(r) = f_p(w)$. A strategy Str is said to be finite-state if each of its local strategies is finite-state.

Specifications that the controlled plant has to satisfy are given as ω-regular languages over Σ. A Büchi automaton over Σ is a structure $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, F)$ where $(Q, \Sigma, \delta, q_{in})$ is a total transition system and $F \subseteq Q$ is the set of *good* states (see [Tho90]). A run of $\mathcal{A}$ over a word $\alpha \in \Sigma^\omega$ is a run over the underlying transition system. For a run $\rho \in Q^\omega$, let $\mathit{inf}(\rho)$ denote the set of states that occur infinitely often in ρ. A run ρ is said to accepting if $\mathit{inf}(\rho) \cap F \neq \emptyset$. A word α is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on α. The set of words accepted by $\mathcal{A}$ is called the language of $\mathcal{A}$ and denoted by $\mathcal{L}(\mathcal{A})$. A language $\mathcal{L} \subseteq \Sigma^\omega$ is said to be an ω-regular language if there is a Büchi automaton $\mathcal{A}$ over Σ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}$.

The general control-synthesis problem in our setting is: Given a finite plant $\mathcal{S}$ over (Σ, θ) and an ω-regular language $\mathcal{L}_{spec} \subseteq \Sigma^\omega$, is there a nonblocking controller *Str* for $\mathcal{S}$ such that $\mathit{plays}_{\mathcal{S}}(\mathit{Str}) \subseteq \mathcal{L}_{spec}$?

Consider the dining-philosophers' problem described earlier. The specification can now include all sequences α where a philosopher, when (s)he gets hungry, always gets to eat (along with some fairness assumptions that one can enforce on α). A valid controller now will be a local program at each site (typically modelling in which way each philosopher requests for forks — left first or right first *and* in which way the fork-processes grant these requests) and will ensure that no philosopher starves.

As we show later, the general control-synthesis problem is undecidable. We impose three restrictions on the problem, which we now describe formally.

A local strategy f_p for p is said to be *clocked* if the advice the strategy offers depends only on the *length* of the history and not on the history itself. Formally, $f_p : \Sigma_p^* \to \Delta_p$ is clocked if for every $x, y \in \Sigma_p^*$ such that $|x| = |y|$, $f_p(x) = f_p(y)$. A strategy *Str* for $\mathcal{S}$ is said to be clocked if all the local strategies of *Str* are clocked. Note that since clocked strategies map into advice functions, the recommendation of the strategy also depends on the current local plant state.

A strategy $\mathit{Str} = \{f_p\}_{p \in \mathcal{P}}$ for $\mathcal{S}$ is called a *com-rigid* ("communication-rigid") strategy if, for every $p \in \mathcal{P}$, $w \in \Sigma_p^*$, there exists a set $X \subseteq \mathcal{P}$ such that for every $s \in S_p^C$ and $a \in f_p(w)(s)$, $\theta(a) = X$. In other words, if at any stage a local strategy offers a set of actions, the processes involved in these actions must be identical.

Through the rest of the paper we will refer to clocked, com-rigid controllers as *strict* controllers. The restricted control problem is then:

Definition 2 (Restricted control-synthesis problem). *Given a finite plant $\mathcal{S}$ over (Σ, θ) and a* trace-closed *ω-regular language $\mathcal{L}_{spec} \subseteq \Sigma^\omega$, is there a nonblocking strict controller Str for $\mathcal{S}$ such that $\mathit{plays}_{\mathcal{S}}(\mathit{Str}) \subseteq \mathcal{L}_{spec}$?* $\Box$

The control-synthesis problem is hence restricted in three ways: (R1) the specification $\mathcal{L}_{spec}$ is trace-closed, (R2) the controller is a clocked strategy and (R3) the controller is a com-rigid strategy. (The nonblocking property is essential for otherwise the problem will become trivial). The main result of the paper is:

Theorem 1. *The restricted control-synthesis problem is decidable. Also, if there is a nonblocking strict controller, one can effectively synthesize a finite-state one.*

Moreover, if one or more *of the restrictions R1, R2 or R3 are removed, the corresponding control-synthesis problem is undecidable.* □

For the dining philosophers problem as formulated above, there appears to be no natural local strategies that satisfy R2 and R3 for the robust specification demanding that no philosopher starves. However, as we point out in the concluding section, we can model this problem in our framework with an additional bounded memory. It then turns out that some standard solutions to the problem do fall into the restricted class of winning strategies we consider.

3 Decidability of restricted control synthesis

Since the complement of a trace-closed language is also trace-closed and since Büchi automata can be effectively complemented, we assume, without loss of generality, that we are presented a Büchi automaton $\bar{\mathcal{A}}_{spec}$ whose language is trace-closed and is the complement of $\mathcal{L}_{spec}$. Our goal is to show that given a plant $\mathcal{S}$ and such a specification automaton $\bar{\mathcal{A}}_{spec}$, whether there is a nonblocking strict controller for the plant such that $plays_{\mathcal{S}}(Str) \cap \mathcal{L}(\bar{\mathcal{A}}_{spec}) = \emptyset$, is decidable.

Let us fix an instance of the problem by fixing a distributed alphabet $\widetilde{\Sigma} = (\Sigma, \theta)$, a plant $\mathcal{S} = \{(S_p^E, S_p^C, \Sigma_p^E, \Sigma_p^C, \rightarrow_p, s_{in}^p)\}_{p \in \mathcal{P}}$ over (Σ, θ) and also a specification automaton $\bar{\mathcal{A}}_{spec}$ whose language is trace-closed with respect to (Σ, θ).

The key technical idea is that strict strategies can be represented as traces (i.e. equivalence classes of strings) over an appropriate distributed alphabet. A number of preliminaries are required before we can arrive at this representation. First, consider a strict strategy $Str = \{f_p\}_{p \in \mathcal{P}}$ for $\mathcal{S}$, where $f_p : \Sigma_p^* \rightarrow \Delta_p$. Since the strategy is clocked, we can view this instead as $\{g_p\}_{p \in \mathcal{P}}$, where $g_p : \mathbb{N}_0 \rightarrow \Delta_p$, for each $p \in \mathcal{P}$, by setting for each $i \in \mathbb{N}_0$, $g_p(i) = f_p(w)$ for some $w \in \Sigma_p^*$ such that $|w| = i$ ($\mathbb{N}_0$ denotes the set of nonnegative integers).

Next consider a strict strategy $Str = \{g_p\}_{p \in \mathcal{P}}$ for $\mathcal{S}$, where $g_p : \mathbb{N}_0 \rightarrow \Delta_p$. We say Str is proper if for every $p \in \mathcal{P}$, $n \in \mathbb{N}_0$ and $s \in S_p^C$, if $a \in g_p(n)(s)$, then there is a finite play $xa \in plays_{\mathcal{S}}^f(Str)$ such that $xa \upharpoonright p = w'a$, $|w'| = n$ and the p-state after executing x is s. That is, if a local strategy recommends a letter a at a controller state s after a history length n, then there must be some play according to the strategy that uses this recommendation. Note that this demand is only w.r.t. the controller states. It is not difficult to see that we can restrict our attention to proper clocked strategies without loss of generality.

Let $\xi_p^{\perp}$ represent the advice function that maps each $s \in S_p^C$ to the empty-set and maps each $s \in S_p^E$ to $en(s)$. If Str is a proper strict strategy, then it is clear that for every $p \in \mathcal{P}$, $n \in \mathbb{N}_0$, if n is odd, then $g_p(n) = \xi_p^{\perp}$ (since the local plant is at an environment state). Moreover, if there is some $n \in \mathbb{N}_0$, n even and $g_p(n) = \xi_p^{\perp}$, then the local plant is at a controller state where the strategy recommends no action. Hence, every play that reaches this stage has no more actions that involve p, and hence for every $n' \in \mathbb{N}_0$ such that $n' \geq n$, $g_p(n') = \xi_p^{\perp}$ (since Str is proper).

An advice function $\xi_p : S_p \to 2^{\Sigma_p}$ is said to be a *fixed-communication advice function* if there is a set $X \subseteq \mathcal{P}$ such that for every $s \in S_p^C$ and $a \in \xi_p(s)$, we have $\theta(a) = X$. If $\xi_p \neq \xi_p^{\perp}$, then this set X is unique and we denote it as $loc(\xi_p)$. Also, by convention, let $loc(\xi_p^{\perp}) = \{p\}$. Let $\widehat{\Delta}_p$ denote the set of all fixed-communication advice functions for p and let $\widehat{\Delta} = \bigcup_{p\in\mathcal{P}} \widehat{\Delta}_p$. Clearly for a proper strict strategy, for every $p \in \mathcal{P}$, $n \in \mathbb{N}$, $g_p(n) \in \widehat{\Delta}_p$.

We now introduce the distributed alphabet $\widetilde{\Pi} = (\Pi, \theta_\Pi)$ over $\mathcal{P}$; strategies will be represented as a single trace equivalence class over this distributed alphabet and our decision procedure will work over the words in this class. Let $Loc = \{\theta(a) \mid a \in \Sigma\}$ and $\Pi = \widehat{\Delta} \cup Loc$. Let $\theta_\Pi : \Pi \to 2^{\mathcal{P}} \setminus \emptyset$ be given by: $\theta_\Pi(\xi_p) = \{p\}$, if $\xi_p \in \widehat{\Delta}_p$ and $\theta_\Pi(X) = X$, $\forall X \in Loc$. Let $\approx_\Pi$ denote the equivalence relation induced by (Π, θ_Π).

Let Loc_p denote the collection of sets $X \in Loc$ such that $p \in X$. A word $\sigma \in \Pi^\omega$ is said to be an *str-linearization* if (i) $\sigma \restriction \widehat{\Delta}_p \in ((\widehat{\Delta}_p \setminus \{\xi_p^{\perp}\}) \cdot \{\xi_p^{\perp}\})^\infty$, for each $p \in \mathcal{P}$, and (ii) for each $p \in \mathcal{P}$, $\sigma \restriction p \in (\widehat{\Delta}_p \cdot Loc_p)^\infty$ such that for every prefix $\tau \cdot \xi_p \cdot X$ of $\sigma \restriction p$ (where $\tau \in \Pi^*$, $\xi_p \in \widehat{\Delta}_p$ and $X \in Loc_p$), we have $X = loc(\xi_p)$. An str-linearization σ is said to *represent* the strict strategy $Str = \{g_p\}_{p\in\mathcal{P}}$ if for every $n \in \mathbb{N}$, $g_p(n-1) = (\sigma \restriction p)[n]$, if $n \leq |\sigma \restriction p|$ and $g_p(n) = \xi_p^{\perp}$ for other values of n.

An str-linearization σ is, in some sense, intended to be one member of the equivalence class of strings (i.e. a trace) whose p-projections are captured by the strategy Str that σ is representing. More crucially, σ describes, in an interleaved fashion, how a play according to Str may evolve. For example, if $\sigma = \xi_p^1 \xi_q^1 \{p,q\} \xi_p^2 \{p\} \ldots$, then, by definition, $loc(\xi_p^1) = loc(\xi_q^1) = \{p,q\}$ and $loc(\xi_p^2) = \{p\}$, and σ intuitively represents a high-level description of a set of plays where the processes p and q first synchronize, followed by a local p-action, etc.

Some easy facts to establish are: Not every member of Π^ω is an str-linearization. An str-linearization represents exactly one strict strategy. Every non-blocking strict strategy has an str-linearization that represents it. Let $Lin(Str)$ be the set of str-linearizations that represent Str. Then $Lin(Str)$ is exactly one $\approx_\Pi$-equivalence class of strings in case Str is a non-blocking strict strategy.

For a finite or infinite play $\alpha \in \Sigma^\infty$, let $com(\alpha)$ denote the string $\pi \in Loc^\infty$ where $|\pi| = |\alpha|$ and for every $1 \leq i \leq |\alpha|$, $\pi[i] = \theta(\alpha[i])$. A play $\alpha \in plays_{\mathcal{S}}(Str)$ *corresponds* to an str-linearization σ of Str if for every $p \in \mathcal{P}$, $com(\alpha \restriction p) = \sigma \restriction Loc_p$. That is, α corresponds to σ if both of them follow the same pattern of communication, when σ is regarded without its $\widehat{\Delta}$-letters. For an str-linearization σ, let $plays_{Lin}(\sigma)$ denote the plays of Str that σ corresponds to. One can show that every play according to Str corresponds to some str-linearization of Str (i.e. $\bigcup_{\sigma\in Lin(Str)} plays_{Lin}(\sigma) = plays_{\mathcal{S}}(Str)$).

Our decision procedure is designed to work over inputs that are str-linearizations of strategies. The "false" inputs will be treated in a "don't-care" fashion and will be filtered out by a separate automaton. We will run an automaton $\mathcal{T}$ over the distributed alphabet $\widetilde{\Pi}$ which, when supplied with an str-linearization $\sigma \in \Pi^*$ representing some strict strategy *Str*, will use the plant structure to non-deterministically emit (one of the) the plays according to *Str*, in the form of a string over the distributed alphabet (Σ, θ). We will then run in tandem, a Büchi automaton $\mathcal{B}$ which will check if the output produced by $\mathcal{T}$ is accepted by $\bar{\mathcal{A}}_{spec}$. A strict controller will exist iff the complement of $\mathcal{B}$ accepts some str-linearization.

Let us now introduce asynchronous transducers. Let $\widetilde{\Sigma}_I = (\Sigma_I, \theta_I)$ and $\widetilde{\Sigma}_O = (\Sigma_O, \theta_O)$ be two distributed alphabets over $\mathcal{P}$ and let $\approx_I$ and $\approx_O$ be the equivalence relations induced by them.

Definition 3. *An* asynchronous transducer *over* $(\widetilde{\Sigma}_I, \widetilde{\Sigma}_O)$ *is a transition system* $\mathcal{T} = (\{U_p, u^p_{in}\}_{p\in\mathcal{P}}, \{\delta_a\}_{a\in\Sigma_I})$ *where each* U_p *is a finite set of states and* $u^p_{in} \in U_p$ *is the local initial state of* p*. For* $a \in \Sigma_I$*, let* U_a *denote the set of functions* $\mathbf{v}_a : \theta_I(a) \to \bigcup_{p\in\theta_I(a)} U_p$ *such that* $\mathbf{v}_a(p) \in U_p$*, for each* $p \in \theta_I(a)$*. Then, for each* $a \in \Sigma_I$*,* $\delta_a \subseteq (U_a \times (\{b \in \Sigma_O \mid \theta_O(b) = \theta_I(a)\} \cup \{\varepsilon\}) \times U_a)$*.* $\square$

An asynchronous transducer hence has a state-space which resembles that of a product system over $\mathcal{P}$. The main difference is that transitions are not entirely local: a transition on an action a depends on the states of the processes that are involved in the action. An element $\mathbf{v}_a \in U_a$ represents some set of states which the processes involved in a are in, and are called a-states. A transition for a is a tuple $(\mathbf{v}_a, b, \mathbf{v}'_a)$, where $\mathbf{v}_a$ and $\mathbf{v}'_a$ are a-states, and b is the letter output when this transition is taken (there is no output if $b = \varepsilon$). Note that if $b \in \Sigma_O$, we require that the processes that are involved in b are precisely those that are involved in a. The global system associated with an asynchronous transducer $\mathcal{T}$ is defined in the natural way: $\mathcal{G}(\mathcal{T}) = (\widehat{U}, \mathbf{u}_{in}, \delta)$ where $\widehat{U}$ is the set of functions $\mathbf{u} : \mathcal{P} \to \bigcup_{p\in\mathcal{P}} U_p$ such that $\mathbf{u}(p) \in U_p$ (for each $p \in \mathcal{P}$), $\mathbf{u}_{in} \in \widehat{U}$ such that $\mathbf{u}_{in}(p) = u^p_{in}$ (for each $p \in \mathcal{P}$). $\delta \subseteq \widehat{U} \times \Sigma_I \times (\Sigma_O \cup \{\varepsilon\}) \times \widehat{U}$ is given by: $(\mathbf{u}, a, b, \mathbf{u}') \in \delta$ iff for each $p \notin \theta(a)$, $\mathbf{u}(p) = \mathbf{u}(p')$ and there exists a transition $(\mathbf{v}_a, b, \mathbf{v}'_a) \in \delta_a$ such that for each $p \in \theta(a)$, $\mathbf{u}(p) = \mathbf{v}_a(p)$ and $\mathbf{u}'(p) = \mathbf{v}'_a(p)$.

For a word $\alpha \in (\Sigma_I)^\omega$, a run of $\mathcal{G}(\mathcal{T})$ on α is sequence of transitions over δ, $(\mathbf{u}_1, a_1, b_1, \mathbf{u}'_1)(\mathbf{u}_2, a_2, b_2, \mathbf{u}'_2)\ldots$ such that $\alpha = a_1 a_2 \ldots$, $\mathbf{u}_1 = \mathbf{u}_{in}$ and, for each $i \in \mathbb{N}$, $\mathbf{u}'_i = \mathbf{u}_{i+1}$. The *output* of such a run is the word $\beta = b_1 b_2 \ldots \in (\Sigma_O)^\infty$ (where ε is treated as the empty word). Again, one can show that the set of words over which $\mathcal{G}(\mathcal{T})$ has a run is trace-closed with respect to (Σ_I, θ_I). Further, one can easily show using trace theory:

Lemma 1. *Let* $\mathcal{T}$ *be an asynchronous transducer over* $(\widetilde{\Sigma}_I, \widetilde{\Sigma}_O)$*. If there is a run of* $\mathcal{G}(\mathcal{T})$ *on* $\alpha \in (\Sigma_I)^\omega$ *that outputs* $\beta \in (\Sigma_O)^\infty$*, then for every* $\alpha' \in \Sigma_I^\omega$ *such that* $\alpha \approx_I \alpha'$*, there exists* β' *such that* $\beta' \approx_O \beta$ *and* $\mathcal{G}(\mathcal{T})$ *has a run on* α' *that outputs* β'*.* $\square$

Let us now return to deciding the restricted control-synthesis problem. We show now that we can design an asynchronous transducer which, when supplied an str-linearization of a strict strategy *Str*, will non-deterministically produce the set of plays corresponding to the str-linearization as output.

Lemma 2. *There is an asynchronous transducer* $\mathcal{T}$ *over* $(\widetilde{\Sigma}, \widetilde{\Pi})$ *that, on any input* σ *which is an str-linearization of a strategy Str, produces non-deterministically as output the set* $plays_{Lin}(Str)$.

Proof: The automaton reads an str-linearization of a strategy and keeps track of the local p-states of the plant in its local p-state. When it encounters a local advice function ξ_p, it consults it to see which moves are recommended from the current state, and records this set of enabled actions. When it encounters a letter $X \in Loc$, it nondeterministically guesses an action $a \in \Sigma$, such that $\theta(a) = X$, which all the participating agents are willing to perform— it then updates its a-state according to the plant's transitions and outputs the letter a. If there is no such action a, then the local plant states move to a special state— from this point on, these processes never allow any other action a that involves them to occur in a play. In this way, the automaton working on an input str-linearization σ, has many runs on σ, and produces the plays in $plays_{Lin}(\sigma)$. □

We can now show that the set of str-linearizations σ such that there is a play corresponding to σ which is not in $\mathcal{L}_{spec}$, is regular.

Lemma 3. *There is a Büchi automaton* $\mathcal{B}$ *over* Π *that accepts an str-linearization* σ *iff there is a play* $\alpha \in plays_{Lin}(\sigma)$ *such that* $\alpha \in \mathcal{L}(\bar{\mathcal{A}}_{spec})$, or *the strategy that* σ *represents is blocking. Further, the set of str-linearizations accepted by* $\mathcal{B}$ *is trace-closed.*

Proof: Consider the asynchronous transducer $\mathcal{T}$ constructed in Lemma 2 that, on every str-linearization σ, produces the plays according to the strategy represented by σ. The state-space of $\mathcal{B}$ will be the product of the spaces of $\mathcal{G}(\mathcal{T})$ and $\bar{\mathcal{A}}_{spec}$. $\mathcal{B}$ will read an input σ and run $\mathcal{T}$ on it. Simultaneously, whenever $\mathcal{T}$ produces an output $a \in \Sigma$, it will simulate $\bar{\mathcal{A}}_{spec}$ on this output. The acceptance condition for $\mathcal{B}$ will be inherited from $\bar{\mathcal{A}}_{spec}$.

It is easy to construct the automaton $\mathcal{B}$ and show that it has the required properties. The one catch is we will have to also check if the strategy (represented by one of its str-linearizations that the automaton is running over) is blocking. But this can be detected by looking for a finite play which gets stuck and then,by moving to a good state, accept this blocking strategy.

Finally, $\mathcal{L}(\mathcal{B})$ being trace-closed can be shown using Lemma 1 and the fact that $\mathcal{L}(\bar{\mathcal{A}}_{spec})$ is trace-closed. □

Theorem 2. *The restricted control-synthesis problem is decidable. Moreover, if there is a restricted controller, we can effectively synthesize a finite-state strategy.*

Proof: By Lemma 3, we construct the automaton $\mathcal{B}$, and complement it to get $\mathcal{B}'$. Now $\mathcal{B}'$ will accept an str-linearization σ iff $plays_{Lin}(\sigma) \subseteq \mathcal{L}_{spec}$ *and* the strategy represented by σ is nonblocking. (Note that the strategy represented by σ is strict). Now, if $\alpha \in plays_{\mathcal{S}}(Str)$ where *Str* is the strategy represented by σ, then there is an str-linearization σ' that corresponds to α. But since all such str-linearizations are trace-equivalent, $\sigma' \approx_{\Pi} \sigma$ and hence σ' is accepted by $\mathcal{B}'$; thus $\alpha \in \mathcal{L}_{spec}$. Hence *Str* is a winning nonblocking strict strategy.

$\mathcal{B}'$ may however accept strings that are not str-linearizations of strategies at all. However, one can easily construct an automaton $\mathcal{B}_r$ that accepts the set of all str-linearizations over Π. The existence of a controller can now be determined by testing the intersection of the automata $\mathcal{B}'$ and $\mathcal{B}_r$, for emptiness: the intersection is nonempty iff there is a controller for the plant. If the intersection is nonempty then the emptiness test will exhibit an ultimately periodic word $\sigma = \tau \cdot (\tau')^\omega$ in the intersection. The p-projection of this word can be realized as a finite state controller for each p. □

4 Undecidability of partially restricted control-synthesis problems

Now, we claim that the restrictions R1, R2 and R3 are, in a sense, necessary in that if one imposes just one or two restrictions of the three, then the problem is still undecidable. Note that removing restriction R2 (or R3) does not a priori mean that the problem is simpler to handle — hence we have to prove undecidability for all the possible combinations of dropping R1, R2 and R3.

The main argument is based on an undecidability proof presented in [PR90], where the the reduction is from the halting problem for Turing machines on a blank tape. Let us fix a Turing machine $M = (\Gamma, \#, Q, q_{in}, \delta, q_h)$ where Γ denotes the tape alphabet, $\# \in \Gamma$ is a special 'blank' symbol, Q is the finite set of states of M, $q_{in} \in Q$ is the initial state and $\delta : Q \times \Gamma \to Q \times \Gamma \times \{l, r\}$ is the transition relation. A move $(q', b, d) \in \delta(q, a)$ means that M, when in state q and reading letter a, replaces a by b, changes to state q', and moves the tape-head one position in direction d (where l denotes 'left' and r denotes 'right').

A configuration of M is a string $C \in \Gamma^* \cdot (Q \times \Gamma) \cdot \Gamma^*$— a configuration $x.(q, a).y$ means that M is in state q, the tape content is $x.a.y$ and the tape-head is at cell $(|x|+1)$. The initial configuration C_{in} is $(q_{in}, \#)$. We write $C \vdash C'$ to denote that C' is the successor configuration of C. A run of M is a sequence of configurations $C_1, C_2, \ldots C_k$, $k \in \mathbb{N}$, where $C_1 = C_{in}$ and where $\forall 1 \leq i < k$, $C_i \vdash C_{i+1}$. Configurations of the form $(q_h, \#).(\#)^*$ are called halting configurations. A run of M is halting if the last configuration is halting; M is said to halt if there is a run of M that is halting.

Consider two sequences of configurations, where configurations are separated by a special symbol '\$': $\tau = C_1 \cdot \$ \cdot C_2 \cdot \$ \cdot \ldots$ and $\tau' = C'_1 \cdot \$ \cdot C'_2 \cdot \$ \ldots$. Consider now a system that reads these sequences in an interleaved manner. In particular, we are interested in two such interleavings, called *useful* interleavings— the interleavings where one sequence starts ahead of another by exactly one configuration and from then on the sequences strictly alternate. That is, one useful interleaving is where C_1 occurs first, then C_2 and C'_1 occur interleaved letter-by-letter, followed by C_3 and C'_2 interleaved letter-by-letter, etc. The other interleaving is the analogous one where C'_1 occurs first.

The crucial point is that a finite-state device can, from these two interleavings, determine whether both of them represent the correct run of M or not. First, it can determine whether both C_1 and C'_1 are the initial configuration of M. When reading the interleaving where C_1 is read first, whenever it gets two configurations C_{i+1} and C'_i interleaved, it checks whether $C'_i \vdash C_{i+1}$. On the other interleaving, while reading C_i and C'_{i+1} interleaved, it checks whether $C_i \vdash C'_{i+1}$. Since a successor configuration is different in at most three places, a finite-state device can check these properties. We say a useful interleaving is *correct* if it satisfies these conditions and also both sequences include a halting configuration. It is easy to show then that the configuration sequences τ and τ' both represent configuration runs of M that halt iff their two useful interleavings are correct.

For grouping together our undecidability arguments succinctly, let $\mathcal{R}$ range over the *proper* subsets of $\{R1, R2, R3\}$. Then the $\mathcal{R}$-controller synthesis problem is the general control-synthesis problem along with the restrictions in $\mathcal{R}$.

Theorem 3. *Given a finite-state plant $\mathcal{S}$ and an ω-regular language specification $\mathcal{L}_{spec}$, the problem of checking if there is a $\mathcal{R}$-controller is undecidable for* every *$\mathcal{R}$.* □

Proof: First assume that $\mathcal{R} = \{R2, R3\}$. Let $\mathcal{P} = \{p, q\}$ and $\Sigma = (\Gamma \cup \{e, \$\}) \times \{p, q\}$ where $\theta((a, p)) = p$ and $\theta((a, q)) = q$ for every $a \in \Gamma \cup \{e, \$\}$. Thus p and q do not synchronize for any action, and evolve independent of each other. Also, p and q have exactly one environment action— (e, p) and (e, q), respectively, which are enabled at every alternate state. The plant at p is a two-state system that can generate, at the controller state, any letter in $(\Gamma \cup \{\$\}) \times \{p\}$; the plant at q is similar. The specification demands the following: (i) The plants locally produce sequences of configurations and the first configuration output is C_{in} and (ii) If the plants happen to evolve in such a way so as to produce a useful interleaving, then the interleaving is correct.

Note that the specification is not trace-closed. We can show that there is a controller that meets the specification iff M halts, and hence the control-synthesis problem for $\mathcal{R} = \{R2, R3\}$ is undecidable. Now note that any controller that meets the specification needs to recommend only one move for p and q at each stage. Also, since there is only one environment action, *any* such controller is a clocked strategy. Further, all actions are local and thus any controller is com-rigid as well. It is hence clear that the problem remains undecidable for $\mathcal{R} = \{R2\}$, $\mathcal{R} = \{R3\}$ and $\mathcal{R} = \emptyset$.

Next assume that $\mathcal{R} = \{R1, R3\}$. We have two processes p and q, as in the above proof, that generate configurations. However, there are two environment actions at p— $Start_p, Next_p$. The plant at p is constructed so that it starts outputting a configuration only if the last environment action is $Start_p$— otherwise, it can only do the action \$. The plant at q is constructed in a similar fashion. The environment hence stutters the actions of p and q that encode configurations.

We also have a new process r, which synchronizes alternately with p and q, observes their controllable actions, and determines whether, when the actions occur according to a useful interleaving, they are correct. The specification is a local specification on the actions of r, and is hence trace-closed. This shows that the control-synthesis problem in this setting with $\mathcal{R} = \{R1, R3\}$ is undecidable.

Also, since r alternately synchronizes with p and q, any strategy is a com-rigid strategy. Hence it follows that the problem remains undecidable for $\mathcal{R} = \{R1\}$.

Finally assume that $\mathcal{R} = \{R1, R2\}$. We have three processes as in the previous proof, with process r observing the actions of p and q. However, there is only one environment action at p and at q, while we have two environment actions e_p and e_q at r. The process r decides which process to communicate with first (p or q), depending on the first environment action that occurs. The process r proceeds then, as usual, to check whether the two useful interleavings are correct. Note that a nonblocking strategy for r cannot be com-rigid. One can now show that there is a controller in this setting iff M halts.

In all the above proofs, it is easy to see that if there is a controller then there is a finite-state controller as well; hence in all the above problems, even searching for a finite-state controller is undecidable. □

5 Conclusions

We have considered a class of effectively synthesizable controllers in a natural asynchronous distributed setting. We have shown that when the specification is restricted to be robust (R1) and the controllers are restricted to follow a clocked (R2) rigid-communication-pattern (R3) strategy, the controller synthesis problem is effectively decidable. And when controllers exist, a finite state controller also exists and can be effectively synthesized. Further, the controller synthesis problem becomes undecidable if one or more of these restrictions are dropped.

Regarding the complexity of our decision procedure, it is easy to see that our algorithm works in time doubly exponential in the size of the input. This is a sharp drop in complexity from the non-elementary results reported in the [PR90] framework for even restricted synchronous architectures.

A number of extensions to our work are worth studying. Firstly, we could generalize our results to handle locally nondeterministic systems — where each local plant is nondeterministic on action labels. A local controller then has to recommend not only an action (which will decide what is to be communicated with the other processes) but also a transition on this action. Our decidability results can be easily modified to handle this (the encoding of strategies now have to encode the recommended transitions).

With this extension, we can relax the restriction R2, as mentioned in the introduction, and endow each local controller with a bounded amount of memory, say b bits, to remember information of its local past. We do this by blowing up the state-space of each local plant by all possible valuations of these b bits and make the transitions nondeterministic on this update of the b bits of information. A clocked controller then, looking at the length of the history *and* the current state of the plant and the b-bits of information, recommends a move as well as an update of its b bits. For example, in the dining philosophers' problem, it turns out that the solutions in the literature (see [Lyn96]) require very few bits of memory for each process (2 bits for the Right-Left dining philosophers' algorithm). Indeed, a variety of resource allocation problems (for example, round-robin based and turn-based solutions to mutual exclusion problems) can be modeled in our framework and this will be explored in future work.

Secondly, in the current setting the plant's transition relations are purely local. A good deal of additional modeling power can be obtained by making the local moves depend also on the states of the other agents involved in a synchronization action. Our results still go through with some additional notational complications.

A different line of research to pursue would be to impose some architectural restrictions and then look for suitable conditions (other than R1, R2 and R3) to achieve decidability. Since R1 and R3 are fairly natural, we suspect that the main challenge will be to replace R2 with suitable conditions that exploit the architectural constraints.

References

[ALW89] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th Int. Coll. on Automata, Languages and Programming*, volume 372 of *LNCS*, pages 1–17. Springer-Verlag, 1989.

[BL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

[Chu63] A. Church. Logic, arithmetics, and automata. In *Proc. International Congress of Mathematicians, 1962*, pages 23–35. institut Mittag-Leffler, 1963.

[DR95] V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.

[HT98] J.G. Henriksen and P.S. Thiagarajan. Distributed versions of Linear time temporal logic. In *Lectures on Petri Nets: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 643–682. Springer-Verlag, (1998).

[KV99] O. Kupferman and M.Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245 – 263, June 1999.

[KV00] O. Kupferman and M. Vardi. μ-calculus synthesis. In *MFCS: Symposium on Mathematical Foundations of Computer Science*, vol. 1893 of *LNCS*, 2000.

[KV01] O. Kupferman and M. Vardi. Synthesizing distributed systems. In *16th Annual IEEE Symp on Logic in Computer Science*, pages 16–19, USA, 2001. IEEE Comp. Society.

[Lyn96] N. A. Lynch. *Distributed Algorithms* Morgan Kaufmann Publ. Inc., 1996.

[Mad01] P. Madhusudan. *Control and synthesis of open reactive systems.* PhD thesis, Institute of Mathematical Sciences, 2001.

[MT01] P. Madhusudan and P. S. Thiagarajan. Distributed control and synthesis for local specifications. In *Proc., ICALP '01*, volume 2076 of *LNCS*, July 2001.

[PR79] G.L. Peterson and J.H. Reif. Multiple-person alternation. In *Proc. 20th IEEE Symposium on Foundation of Computer Science*, pages 348–363, 1979.

[PR89a] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symposium on Principles of Programming Languages*, 1989.

[PR89b] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. 16th ICALP*, volume 372, pages 652–671. LNCS, 1989.

[PR90] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. 31st IEEE Symp. on Foundation of Comp. Sc.*, pages 746–757, 1990.

[Tho90] W. Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Comp. Sc.*, volume B, pages 135–191, Elsevier, 1995.