Distributed Controller Synthesis for Local Specifications

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

[1] The Institute of Mathematical Sciences madhu@imsc.ernet.in
[2] CREST, School of ECE, Georgia Institute of Technology, Atlanta, USA (On leave from Chennai Mathematical Institute, Chennai, India) thiagu@ece.gatech.edu

Abstract. We consider the problem of synthesizing distributed controllers for reactive systems against local specifications. We show that a larger class of architectures become decidable in comparison to the analogous problem for global specifications. We identify the exact class of architectures for which the problem is decidable. Our results also show the decidability of a related realizability problem for local specifications.

1 Introduction

An *open* reactive system is one which interacts with its environment on a systematic basis and whose behaviour crucially depends on this interaction. The key feature of open systems is that one is required to distinguish between the capabilities of the system and its environment. Typically, one views such an open system as getting inputs from the environment and reacting to it by outputting values. *Realizability* and *controller synthesis* problems arise naturally in the study of open systems.

The realizability problem is to determine, given a specification of an open system, say as a temporal logic formula, whether there exists a program for the system such that no matter how the environment behaves, the overall behaviour satisfies the specification. The program will fix the value the system outputs on receiving a particular input and this choice can depend on the past history of the interaction with the environment. The environment is allowed to input any value at any point.

The controller synthesis problem on the other hand, starts with an open system — often called a *plant* in this context — and a specification, say, a temporal logic formula. The plant is viewed as an existing program which specifies the ways in which the system can react to its inputs. The goal now is to come with a strategy to interact with the environment in a way allowed by the plant, such that all behaviours satisfy the specification. Thus the strategy, in this setting, acts as a *controller* for the plant which restricts the behaviours of the plant so that the specification is met.

There is a wealth of literature on realizability and controller synthesis problems for open systems as evidenced in [BL69,Rab72,Tho95,RW89,PR89,ALW89] for linear-time specifications and [KV96,MT98,KV99,KMTV00] for branching-time specifications. All of these studies are confined to programs and plants that consist of a *single* sequential module. However, these problems often arise in a distributed context and here, the main results presently available are due to Pnueli and Rosner [PR90]. We extend here their results and the point of departure is to consider *local* specifications. Before going into this in more detail, we wish to mention the work reported in [dAHM00,dAH00,AHK97] which are also concerned with control-related issues in a distributed setting but not directly connected to the concerns of the present work.

Pnueli and Rosner consider distributed programs by using the notion of an *architecture* which basically consists of a set of sites and some communication channels between them. The sites also have external input and output channels through which they interact with the environment. The specification consists of linear time temporal logic formulas whose atomic propositions can state properties of the values on the external input and output channels. The specification is hence a *global* one which can talk about simultaneous channel-values at different sites. The surprising main result is that even in the case of an architecture consisting of *two* sites and *no* internal communication channels, the realizability problem is undecidable. They also consider pipeline architectures which consists of a linear array of sites $s_0 \to s_1 \to \ldots \to s_n$ with an external input channel allowed only for s_0. They show that the realizability problem is decidable for this class of architectures.

It turns out that many results of [PR90] extend also to the control-synthesis problem that we wish to study. Here, we are given in addition to the specification, a set of programs at each site and the problem is to come up with *local* strategies for the sites to restrict the programs; by a local strategy at a site, we mean a strategy which only knows the values which the input channels to this site have carried so far. In [PR90], realizability is actually shown to be decidable for a larger class of architectures (called *hierarchical architectures*) but only the decidability results for pipelines carry over for the control-synthesis problem.

In this paper, we drop global specifications since they turn out to be unreasonably expressive, and instead consider local specifications and, for convenience, study only the controller synthesis problem. We identify a special class of architectures called *clean* pipelines (see Figure 1) which are just like the pipelines mentioned above except external inputs are also allowed for the right-end site s_n. Our main result is that the controller synthesis problem for local specifications is decidable for an architecture $\mathcal{A}$ iff each connected component of $\mathcal{A}$ is a clean pipeline or is a sub-architecture of a clean pipeline.

Our undecidability results go through for weaker acceptance conditions down to reachability. Thus our negative results show that even in the presence of local specifications, the controller synthesis problem is intractable for almost all architectures. However, on the positive side, our results show that for local specifications, one can handle the nontrivial distributed reactive system which consists of two sites, both of them interacting with the environment, and with an internal channel from one to the other. Indeed, our results show that the realizability problem can also be effectively solved in this important setting, where the specifications at the sites *can state properties of the internal channels as well.*

The undecidability result of [PR90] follows from the study of multi-player games of incomplete information by Peterson and Reif [PR79]. In this context, our results show that there are certain games where two players, playing against an adversary, have incomplete information about each other, and yet determining whether they have a winning strategy is decidable.

In the next section we introduce the formal setting for our work, Section 3 establishes our decidability results and in Section 4 we prove the undecidability results. Due to lack of space, we provide only the main proof ideas — more details can be found in [MT01].

2 Problem Setting

An *architecture* is a tuple $\mathcal{A} = (S, X, T, r, w)$ where $S = \{s_1, \ldots, s_k\}$ is a finite nonempty set of *sites*, $X = \{x_1, \ldots, x_l\}$ is a set of external (or environment) input channels and $T = \{t_1, \ldots, t_n\}$ is a set of internal channels. r is a function $r : X \cup T \rightarrow S$ which identifies for each channel a process which *reads* the channel; $w : T \rightarrow S$ identifies for each internal channel, a process which writes into it. We assume, without loss of generality, that each process has at most one external input channel and that there is at most one channel from one site to another.

We represent architectures graphically as directed graphs whose nodes are the sites and every channel $z \in X \cup T$ is represented by an edge — if $z \in T$, then it is an edge from $w(z)$ to $r(z)$ and if $z \in X$, then it is a sourceless edge to $r(z)$. We only consider *acyclic* architectures — i.e. those architectures whose graph representation does not have a directed cycle. We assume further that every site has at least one input (external or internal) channel.

In our framework, each site runs a program which reads its external and internal channel inputs and reacts by sending outputs along the internal channels to other processes and changing its state. The moves are synchronous — i.e. the programs read a set of external inputs and make one collective move while respecting the partial order imposed by the architecture.

For example, in the architecture $\mathcal{A}_2$ in Figure 2, in a synchronous step, s_1 will read the environment's input on x_1, change its state and write onto t_1; s_2 will read this input on t_1 and the input on x_2, change its state and write onto t_2; s_3 will read this value and change its internal state.

For a site $s \in S$, let $in(s) = r^{-1}(s)$, the set of channels which s reads from and $out(s) = w^{-1}(s)$, the set of channels S writes into. Given an architecture $\mathcal{A}$, a *domain definition* for $\mathcal{A}$ is a function D which associates with each $z \in X \cup T$ a finite set of values which can be sent along the channel z. We denote $D(z)$ sometimes as D_z. For a set of channels Z, a *valuation function for* Z is a function h whose domain is Z and which maps each $z \in Z$ to an element of D_z. Let $\mathcal{H}_Z$ denote the set of all valuation functions for Z.

Definition 1. *A plant is a tuple $(\mathcal{A}, D, \widehat{P})$ where $\mathcal{A}$ is an architecture (say having k sites $\{s_1, \ldots, s_k\}$), D is a domain definition for $\mathcal{A}$, $\widehat{P}$ is a set of* programs, *one at each site — i.e. $\widehat{P}$ is a tuple $\langle P_1, \ldots P_k \rangle$. Each P_i is a transition system $(Q_i, q_i^{in}, \delta_i)$ where Q_i is a set of states, $q_i^{in} \in Q_i$ is the initial state, δ_i is a non-deterministic transition function $\delta_i : Q_i \times \mathcal{H}_{in(s_i)} \rightarrow \mathcal{P}(Q_i \times \mathcal{H}_{out(s_i)})$.*[1] $\square$

The transition function of each program defines the different ways in which a site can react to a set of inputs on its input channels, by giving the possible sets of values which can be written on the output channels together with a corresponding state change. We say a plant is *finite* if Q_i is finite for each P_i.

Let $(\mathcal{A}, D, \widehat{P})$ be a plant. For a program $P = (Q, q^{in}, \delta)$ at a site s in $\mathcal{A}$, a *local strategy for* P is a function $f : Q \times \mathcal{H}^{+}_{in(s)} \rightarrow Q \times \mathcal{H}_{out(s)}$ such that $\forall q \in Q$, $\pi \in \mathcal{H}^{+}_{in(s)}$, if $\pi = \pi' \cdot h$ then $f(q, \pi) \in \delta(q, h)$. Thus the local strategy f is an advice function for P which looks at the history of values (π') on the local input channels and the current values on them (h), and prescribes a move which the local program P should take.

A *distributed control-strategy* is a set of local strategies, one for each site: i.e. a tuple $f = \langle f_1, \ldots f_k \rangle$ where f_i is a local strategy for P_i. We call a plant along-with a strategy $((\mathcal{A}, D, P), f)$ a *controlled system*. Let us fix for now a controlled system $((\mathcal{A}, D, P), f)$.

We need some notations for talking about sequences. For a sequence α, let $\alpha[i]$ denote the i^{th} atom in α and $\alpha[i, j]$ denote the finite subsequence of α from the i^{th} to the j^{th} element, both inclusive, for $0 \leq i \leq j$, $i, j \in \mathbb{N}$. Also, we denote by $inf(\alpha)$ the set of elements which occur infinitely often in α. If α is a sequence of functions on a domain Z, let $\alpha \downarrow Z'$, where $Z' \subseteq Z$, denote the sequence of functions obtained by restricting each function in α to Z'.

Consider an environment input sequence (on the channels in X) $\alpha \in \mathcal{H}_X$. Since P, when controlled by the strategy f, is deterministic, there is a unique way in which the plant and controller respond to the external input — i.e. there is a unique sequence of states each program takes and a unique set of channel values sent along each channel. We can define these sequences as follows. Let $\beta \in (\mathcal{H}_{X \cup T})^{\omega}$ and $\gamma \in (Q_1 \times \ldots Q_k)^{\omega}$ be such that:

1. $\gamma[0] = \langle q_1^{in}, \ldots, q_k^{in} \rangle$
2. $\beta \downarrow X = \alpha$
3. $\forall t \in T$, $j \in \mathbb{N}$, if $w(t) = s_i$ and $\gamma[j] = \langle q_1, \ldots q_k \rangle$ with $f_i(q_i, \beta[0, j] \downarrow in(s_i)) = (q_i', h)$, then $\beta[j](t) = h(t)$
4. $\forall j \in \mathbb{N}$, if $\gamma[j] = \langle q_1, \ldots q_k \rangle$ and $\gamma[j+1] = \langle q_1', \ldots q_k' \rangle$, then it must be the case that $\forall i \in \{1, \ldots, k\}$: $f_i(q_i, \beta[0, j] \downarrow in(s_i)) = (q_i', h)$ for some $h \in \mathcal{H}_{out(s_i)}$

The definitions above formalize how the programs behave when they get an external input. (1) says that the global state-sequence starts with the initial states. The next condition requires that the values which the external channels take are defined by the external input α. (3) and (4) demand that the values the internal channels take and the evolution of states are according to the moves

[1] $\mathcal{P}(R)$ denotes the power-set of R

defined by the local strategy. It is easy to see that, since the architecture is acyclic, there are unique sequences β and γ which satisfy the above conditions. We call γ the *state-behaviour* of the system for the input α.

The specification is now defined on the *local* state-behaviours of the system. Since we wish to capture local linear-time properties, we have local Rabin winning conditions and the specification then demands that the local runs of the controlled system meet these conditions. We could instead work with temporal logic specifications, one at each site, which describes the behaviour of the local channels (external *and internal*) and the local states. However, since we can cast this problem in terms of Rabin conditions (by building a deterministic Rabin automaton accepting the desired behaviours [Saf88] and taking its intersection with the local plant), we can reduce this problem to our setting. coded into the states of the plant.

A *local Rabin winning condition* $\mathcal{R}_i$ for a site s_i is a set $\{(R_1, G_1), \dots (R_m, G_m)\}$ where R_j, G_j are subsets of Q_i. A *Rabin winning condition* $\mathcal{W}$ is a tuple $\langle \mathcal{R}_1, \dots \mathcal{R}_k \rangle$ where each $\mathcal{R}_i$ is a local Rabin winning condition for s_i. Let $\gamma \in (Q_1 \times \dots Q_k)^\omega$ be a sequence of global states of the system. Let $\gamma \downarrow i$ denote the sequence in Q_i^ω obtained by projecting γ to the component involving Q_i. γ is said to *satisfy* a Rabin condition $\mathcal{W}$ if for each site s_i, there is a pair (R, G) in $\mathcal{R}_i$ such that $\mathit{inf}(\gamma \downarrow i) \cap R = \emptyset$ and $\mathit{inf}(\gamma \downarrow i) \cap G \neq \emptyset$.

Finally, a controlled system is said to satisfy a Rabin winning condition $\mathcal{W}$ if for every sequence of external inputs $\alpha \in (\mathcal{H}_X)^\omega$, the state-behaviour γ defined by α satisfies $\mathcal{W}$. Given a winning condition, a strategy f for a plant $(\mathcal{A}, D, P)$ is said to be *winning* if the controlled system $((\mathcal{A}, D, P), f)$ satisfies the winning condition. Now, the control-synthesis problem for an architecture $\mathcal{A}$ is: Given a finite plant $(\mathcal{A}, D, P)$, and a Rabin winning condition, does there exist a winning strategy for the plant?

An important class of architectures are the class of *clean* pipelines which are pipelines that have external inputs only at the two endpoints (see Fig. 1): An architecture $\mathcal{A}$ is said to be a *clean pipeline* if it has k sites $s_1, \dots s_k$ (for some $k \in \mathbb{N}$), two external input channels x_1 and x_2, $k-1$ internal channels $t_1, \dots t_{k-1}$, with $r(x_1) = s_1$, $r(x_2) = s_k$, $w(t_i) = s_i$ and $r(t_i) = s_{i+1}$, for $i \in \{1, \dots, k-1\}$.

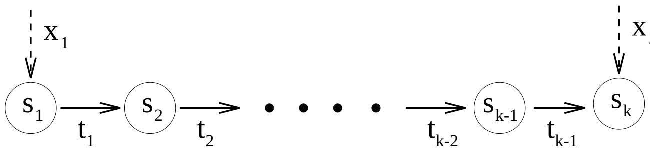

Fig. 1. A generic clean pipeline

We also need the notion of a sub-architecture — an architecture $\mathcal{A}'$ is a sub-architecture of an architecture $\mathcal{A}$ if the graph of $\mathcal{A}'$ is isomorphic to a subgraph of the graph of $\mathcal{A}$. Note that an architecture is a sub-architecture of itself. We can now state the main result of the paper.

Theorem 1. *Let $\mathcal{A}$ be an architecture. The control-synthesis problem for $\mathcal{A}$ is decidable iff each connected component of $\mathcal{A}$ is a sub-architecture of a clean pipeline.* $\square$

3 Decidable architectures

In this section, we show that architectures where each connected component is a sub-architecture of a clean pipeline is decidable. Since we have local winning conditions, it is easy to observe that the control-synthesis problem for an architecture $\mathcal{A}$ is decidable iff the control-synthesis problem is decidable for each of its connected components. Hence it suffices to prove that the problem is decidable for architectures which are sub-architectures of clean pipelines.

We use alternating and non-deterministic tree automata to prove our decidability results. Due to lack of space, we assume a standard presentation of these automata [Tho90,MS95].

A *tree* is a directed acyclic graph $T = (V, E)$ which has a designated *root* r which does not have a parent. Every other node is reachable from r and has a unique parent. We say v' is a child of v if $(v, v') \in E$. For a set Σ, a Σ-labelled tree is a pair (T, τ) where $T = (V, E)$ is a tree and $\tau : V \to \Sigma$. Let Υ be a finite set. Then Υ^* can be viewed as a tree $T_\Upsilon = (\Upsilon^*, E)$ where $(x, x \cdot d) \in E$, for every x in Υ^* and d in Υ. We refer to this as the Υ-tree.

Consider a plant $(\mathcal{A}, D, P)$ and a distributed control-strategy f for it. Let s be a site with an output channel t in $\mathcal{A}$. Let $\mathcal{L} \subseteq D_t^\omega$ be the language of infinite strings output on t (by considering all possible inputs on the external input channels of the plant). We call such a language of infinite words, *a communication language for the channel t*. Let $\mathit{Pref}(\mathcal{L}) = \{x \mid \exists \beta \in \mathcal{L}, x \textit{ is a prefix of } \beta\}$. Then it is not difficult to see that $\mathcal{L} = \mathit{lim}(\mathit{Pref}(\mathcal{L}))$ where $\mathit{lim}(L) = \{\alpha \in \Sigma^\omega \mid \textit{for every prefix } x \textit{ of } \alpha, x \in L\}$. Also, $\mathcal{L} \neq \emptyset$.

So $L = \mathit{Pref}(\mathcal{L}) \subseteq D_t^*$, the set of finite sequences sent along t represents the set of infinite sequences sent along the channel as well. So $\mathcal{L}$ can be represented (uniquely) by a $\{\top, \bot\}$-labelled D_t-tree $T = (D_t^*, \tau)$, where $\tau(x) = \top$ if $x \in L$ and $\tau(x) = \bot$, otherwise. In such a tree if a node has label $\top$ then it will have at least one child with the label $\top$ and if a node has label $\bot$ then all its children will have the label $\bot$. Also, the root, ϵ is labelled $\top$. Clearly each such $\{\top, \bot\}$-labelled D_t-tree (which we call t-type trees) uniquely represents a communication language of the channel t and our automata run over such trees. If T is a t-type tree then we let $\mathit{Lang}(T)$ denote the language of infinite strings it represents.

Let us fix a clean pipeline which has k sites, as shown in Figure 1. We refer to s_1 as the *left-site*, s_k as the *right-site* and each of the s_i's, $1 < i < k$ as *middle sites.*

Let s be the left-site of a clean pipeline with input channel x and output channel t. Suppose P is the program at s, and there is is a local *winning* strategy f at s and $\mathcal{L}$ is the language of infinite words sent along t as a result of the pair (P, f) reacting to all possible inputs on x. Then $\mathcal{L}$ is said to be an *s-successful* language. For a right-site s, with input channels t and x, we say that a language

$\mathcal{L}$ of infinite strings over D_t is s-successful, if there is a strategy f at s which can work on these inputs, and arbitrary inputs on x, and win locally. For a middle-site s with input channel t and output channel t', we say that $\mathcal{L}' \subseteq D(t')^\omega$ is successfully generable by s on $\mathcal{L} \subseteq D(t)^\omega$ if there is a strategy at s which wins locally on reading inputs from $\mathcal{L}$ on t and produces $\mathcal{L}'$ on t'.

Lemma 1. *Let s be the left-site of a clean pipeline with input channel x, output channel t and program P. Then there is an alternating tree automaton (on t-type trees) which accepts a t-type tree T iff the language represented by T has an s-successful sublanguage.*

Proof: The automaton we construct, while running over a t-type tree T, guesses a local strategy f for the program at s, makes sure that f produces no string which is not in $Lang(T)$ and also checks that f is locally winning.

The automaton has, in its state-space, a component encoding which state of the program it is currently simulating. Then reading a node y of T, it does the following:

- Guess a set of moves from the current state for each possible input d in $D(x)$.
- The automaton propagates, for each $d \in D(x)$, a copy along the direction $d' \in D(t)$ where d' is the output of the plant on d according to the guessed move. The corresponding successor state of the program is also propagated and the automaton will check in these copies whether the labels of the nodes it reads are $\top$. This will ensure that the outputs are allowed by T.

The acceptance condition ensures that paths on a run encode state-sequences which satisfy the local winning condition of P. Since each node in a run corresponds to a unique input history, the independent guessing of the strategy at these nodes is justified. □

Note that the automaton accepts a tree provided the language represented by the tree merely *contains* an s-successful language. It seems hard to strengthen this containment to equality. However, the present version will suffice.

Lemma 2. *Let s be a right-site of a pipeline with $in(s) = \{x, t\}$ and let the program at s be P. Then there is an alternating tree automaton on t-type trees which accepts a tree T iff the language that T represents is s-successful.*

Proof: The automaton guesses a local strategy for P at s on input sequences $\alpha \in Lang(T)$ along t and arbitrary input sequences $\beta \in D(x)^\omega$ on x and makes sure that f is winning for all local runs on these sequences.

The automaton keeps track in its state-space the current state of P it is simulating. Reading a node y of the input tree, it does the following:

- Guess $Y \subseteq D(t)$ corresponding to the set of successors of y labelled $\top$. The automaton will (in its next move) check if Y is indeed the set of $\top$-successors.
- The strategy has to handle all inputs in Y on the channel t along with an arbitrary input in $D(x)$ on channel x. The automaton guesses such a strategy at this point by guessing moves from the current state of P on each $h \in \mathcal{H}_{\{x,t\}}$ with $h(t) \in Y$. It then propagates along each direction d in Y,

a copy of the automaton for each $d' \in D(x)$ corresponding to the chosen move when channel t carries d and channel x carries d'. It propagates the corresponding state of P as well.

The acceptance condition is that all paths on a run encode a state-sequence in P which satisfies the local winning condition of P. Again, since each node in a run corresponds to a unique input history, the guessing of the strategy at these points independently is justified. $\square$

Theorem 2. *The two-site clean pipeline is decidable.*

Proof: Let the sites and channels of the pipeline be labelled as in Figure 1. Using Lemma 1, construct an automaton A_1 which accepts a t_1-type tree T iff s_1 can successfully generate a sublanguage of $Lang(T)$. Using Lemma 2, construct A_2 which accepts t_1-type trees which represent languages which s_2 can win on. The claim now is that a distributed winning strategy exists iff $L(A_1) \cap L(A_2)$ is nonempty.

Assume $T \in L(A_1) \cap L(A_2)$ and let $\mathcal{L}$ be the language it represents. Then there is a strategy f_2 at s_2 which wins on $\mathcal{L}$. Also, there is a local winning strategy f_1 at S_1 which generates a sublanguage $\mathcal{L}'$ of $\mathcal{L}$. However, since the local winning conditions are linear-time specifications, f_2 wins on $\mathcal{L}'$ as well. Hence $\langle f_1, f_2\rangle$ is a distributed winning strategy. Furthermore, one can construct, from the runs of A_1 and A_2 on a regular tree in $L(A_1) \cap L(A_2)$, a strategy which can be realized as a finite-state transition system. Also, it is easy to see that if $\langle f_1, f_2\rangle$ is any winning distributed strategy, then the tree corresponding to the language f_1 generates is accepted by A_1 as well as A_2. $\square$

Lemma 3. *Let s be a middle-site of a clean pipeline with $in(s) = \{t\}$ and $out(s) = \{t'\}$, and let the program at s be P. Let A be a nondeterministic automaton accepting t-type trees. Then there is an automaton on $\{\top, \bot\}$-labelled t'-type trees that accepts a tree T' iff there is a t-type tree T accepted by A and a language $\mathcal{L}_0 \subseteq Lang(T')$ such that $\mathcal{L}_0$ is successfully generable by s on $Lang(T)$.*

Proof: Let T' be an input to the automaton and $\mathcal{L}'$ be the language it represents. The automaton, while reading T', guesses a t-type tree T, guesses a run of A on T, guesses a strategy f on the strings represented in T and makes sure that the run on T is accepting, makes sure that the strategy outputs strings which are included in $\mathcal{L}'$ and makes sure that the strategy locally wins.

A node in the run on T' corresponds to a node y' in T' as well as a node x of the tree T being guessed — here x is the sequence in $D(t)^*$ on which the guessed strategy has output y'. Note that each sequence in $D(t)^*$ can lead to at most one sequence in $D(t')^*$ being output and hence guessing of the tree T at nodes of the run is justified.[2]

The state-space of the automaton has both the current state of P as well as a state of the automaton A which represents the state-label of the corresponding node in T in the guessed run on T. The automaton at a node in the run corresponding to the node y' in T' and x in T does the following:

[2] If the site also has an external input, this will not be the case.

- Guess the set $Y' \subseteq D(t')$ which corresponds to the children of y' labelled $\top$.
- Guess the labels of the children of x in T. This is the point where T is being guessed. Let $X \subseteq D(t)$ be the children of x labelled $\top$.
- The automaton now guesses a move of P from the current state on each $d \in X$ and makes sure that the output on t is in Y'. It then propagates along each direction $d' \in Y'$ in T, many copies of itself — each corresponding to a $d \in D(t)$ on which the guessed move outputs d'. The appropriate successor state of P is propagated. The automaton also guesses a transition of A from the node x and propagates these automaton states as well.

The acceptance condition makes sure that along any path in the run, the state-sequence of P meets the local winning condition of P and the state-sequence of the automaton meets the winning condition of A. $\square$

Theorem 3. *The control-synthesis problem for clean pipelines is decidable.*

Proof: We start with the left-site of the pipeline, use Lemma 1 and walk down the pipeline by successively using Lemma 3. After each site, we have to convert the alternating automata we get to a nondeterministic one [MS95] so as to apply Lemma 3. In the end, we intersect the automata we have got with that obtained using Lemma 2. Then by an argument similar to the one in Theorem 2, we can show that there is a nonempty intersection iff there is a distributed controller and if a controller exists, we can synthesize one which is finite-state. $\square$

The results imply the decidability of a related realizability problem: given a clean pipeline and localised temporal logic specifications at each site, where each specification can express properties of the local channels at that site, is there a program at each site such that the combined behaviour meets the specification? This problem can be reduced to the control-synthesis problem by choosing a trivial plant at each site which permits all possible ways of writing onto the local output channels.

4 Undecidable architectures

We show now that any architecture which is not a sub-architecture of a clean pipeline is undecidable. We show first the undecidability of the three basic architectures in Figure 2. The reductions are from the halting problem for deterministic Turing machines starting with blank tapes. Our proofs are extensions of the the undecidability proof developed in [PR90].

A configuration of a deterministic Turing machine M is a sequence $C \in \Gamma^* \cdot Q \cdot \Gamma^+$ where Γ is the set of tape symbols and Q is the set of states. If $C = xqy$, with $q \in Q$, then the machine has $x \cdot y$ written on the tape with the head position on the cell after x. The initial configuration, $C_{in} = q_{in} \cdot \flat$ where q_{in} is the initial state and $\flat$ is the special tape symbol called blank. The transition relation $\vdash$ on configurations is defined in the obvious way. We say that the machine halts on the blank-tape if $C_{in} \vdash^* C_h$ where the state in C_h is q_h, a designated halt state.

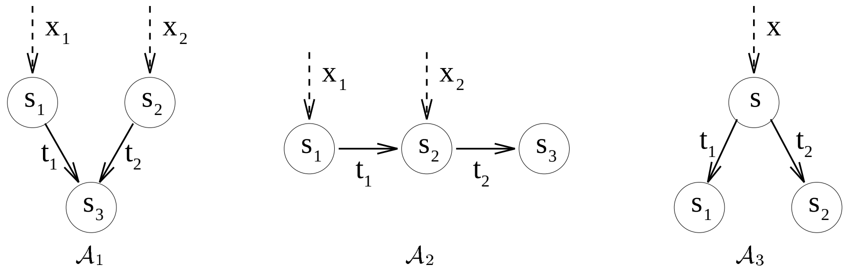

Fig. 2. Basic undecidable architectures

The sites host programs which have associated strategies to output configurations. The (finite state) program outputs words in $\Gamma^* \cdot Q \cdot \Gamma^+$ with the strategy deciding which configurations are output. The input channels carry two symbols S (Start outputting a new configuration) and N(output the next symbol of the current configuration). On receiving S, a program will output \$, followed by a configuration while reading N and finish the configuration by outputting \$. It waits while reading N, and outputs a special symbol $*$ during this time, till it get an S on which it starts outputting another configuration. The first configuration output by the program is always C_{in}.

Lemma 4. *The control-synthesis problem for the architecture $\mathcal{A}_1$ is undecidable.*

Proof: The main idea of the proof is to make s_1 and s_2 always send their current states to s_3. Site s_3 now has the global view of s_1 and s_2 and a global specification of s_1 and s_2 (exploited in [PR90] to get the undecidability argument) can be stated as a local specification for s_3. □

Lemma 5. *The control-synthesis problem for the architecture $\mathcal{A}_2$ is undecidable.*

Proof: Site s_1 will output configurations when prompted by the environment through channel x_1. Site s_3 will, when prompted by s_2 on t_2, "accept" configurations instead of outputting them; when it starts a configuration, it will generate it one unit time in advance and keep the generated symbol of $\Gamma \cup Q$ in its state-space. It proceeds from this state only if the input it receives on t_2 is the same as the symbol it has committed to. It then proceeds to commit the next symbol. This can be looked upon as s_3 generating configurations which s_2 has to predict correctly.

Site s_2 can go into two modes, A and B, the decision being taken according to the first environment input on x_2. In Mode A, the program at s_2 simply passes the configurations which it receives on t_1 to t_2. In Mode B, the program first outputs the initial configuration to s_3 and after that, each time it receives a configuration C on t, it propagates online C' to t_2 where $C \vdash C'$.

If s_3 receives a symbol it has not committed to, it goes to a reject state. Mode A ensures that the two sites output/accept the same configuration sequences while Mode B ensures that if the i^{th} configuration output by s_1 is C and the

$(i+1)^{th}$ configuration accepted by s_2 is C', then $C \vdash C'$. So the only way the plant can hope to win is by s_1 and s_3 accepting the configuration sequence of M. By introducing a winning condition on s_2 which makes sure that s_2 locally wins only if it outputs the halting configuration, one can show that the plant has a distributed winning strategy iff M halts on the blank tape. $\square$

Lemma 6. *The control-synthesis problem for the architecture $\mathcal{A}_3$ is undecidable.*

Proof: As done by s_3 of $\mathcal{A}_2$ in the previous lemma, s_1 and s_2 will now accept configuration of M. Site s can be in two modes, A and B, the mode chosen by the first input on x. In Mode A, the program at s_1 passes the initial configuration C_{in} to s_1 and makes s_2 wait. Then, while getting as input an arbitrary configuration C from the environment on x, it passes C to s_2 and simultaneously passes C' to s_1 where $C \vdash C'$. Mode B is analogous with the roles of s_1 and s_2 interchanged.

To force s_1 and s_2 to accept the correct configuration sequence of M, we would like the environment to win iff it can get the site scheduled first to be unstuck and get the other stuck. The trick is to have another mode C for s where the plant is forced to emulate the combined (product) behaviour of s_1 and s_2. The winning condition can now be stated on the state-space of s. Then one can make make sure that one of the sites, say s_1, wins when it accepts the halting configuration. One can show now that a distributed controller exists iff M halts on the blank tape. $\square$

Using Lemma 4 we can show that any architecture which has a site s with two internal input channels is undecidable. The idea is to pick the minimal sites above the two internal channels for s and make the rest of the sites "dummy" by making them just pass their input to their output and always win locally. We can then reduce the control-synthesis problem of $\mathcal{A}_1$ to that over this architecture. Similarly, using Lemma 6 we can show that any architecture which has a site with two internal output channels is undecidable.

What we are left with are pipelines. Since we require each process to have an input channel, the left-site of the pipeline must have an external input channel. Consider a pipeline (which is not a clean pipeline) with sites $\{s'_1, \ldots, s'_k\}$, $k > 2$, with s'_i having an external input channel where $1 < i < k$. We can reduce the control-synthesis problem for $\mathcal{A}_2$ to the control-synthesis problem for this pipeline, by coding the program at s_1 into s'_1, the program at s_2 into s'_i and the program at s_3 into s'_k. The remaining sites of the pipeline will be "dummy". Hence we have:

Theorem 4. *If $\mathcal{A}$ is an architecture which has a connected component which is not a sub-architecture of a clean pipeline, then the control-synthesis problem for $\mathcal{A}$ is undecidable.* $\square$

The results above can be suitably changed to show that even for weaker winning conditions such as Büchi, co-Büchi, or even safety conditions, the architectures remain undecidable.

Acknowledgement: We would like to thank Wolfgang Thomas and Christof Löding for several fruitful discussions on early drafts of this paper.

References

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. 38th IEEE FOCS*, pages 100–109, October 1997.

[ALW89] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th ICALP*, vol. 372, pages 1–17. LNCS, Springer-Verlag, July 1989.

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

[dAH00] Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *Proc., LICS '00, 15th Annual Conf.*, pages 141–154, 2000.

[dAHM00] Luca de Alfaro, Thomas A. Henzinger, and F.Y.C. Mang. The control of synchronous systems. In *Proc., CONCUR '00, 11th Int. Conf.*, vol. 1877 of *LNCS*, pages 458–473, Penn. State Univ, USA, Sept. 2000.

[KMTV00] O. Kupferman, P. Madhusudan, P.S. Thiagarajan, and M. Vardi. Open systems in reactive environments: Control and synthesis. In *Proc., CONCUR '00, 11th Int. Conf.*, vol. 1877 of *LNCS*, USA, Sept. 2000.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *CAV, Proc. 8th Intl. Conf.*, vol. 1102 of *LNCS*, pages 75–86. Springer-Verlag, 1996.

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

[MS95] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[MT98] P. Madhusudan and P. S. Thiagarajan. Controllers for discrete event systems via morphisms. In *In Proc., CONCUR'98, 9th Int. Conf.*, vol. 1466 of *LNCS*, pages. 18–33, France, September 1998.

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

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Prog. Languages*, Austin, January 1989.

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

[Rab72] M.O. Rabin. Automata on infinite objects and Church's problem. *Amer. Mathematical Society*, 1972.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. FOCS*, pages 319–327, White Plains, October 1988.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. In E.W. Mayr and C. Puech, editors, *Proc. 12th Symp. on Theoretical Aspects of Comp. Sc.*, vol. 900 of *LNCS*, pages 1–13. Springer-Verlag, 1995.

[MT01] P. Madhusudan and P. S. Thiagarajan. Distributed Controller Synthesis for Local Specifications. Technical Report TCS-01-1, *Chennai Mathematical Institute*, India. Available at `http://www.smi.ernet.in`