Controller Synthesis for Hierarchical Agent Interactions

Sunandita Patra[1] PATRAS@UMD.EDU
Paolo Traverso[2] TRAVERSO@FBK.EU
Malik Ghallab[3] MALIK@LAAS.FR
Dana Nau[1] NAU@CS.UMD.EDU

[1]Department of Computer Science, University of Maryland, College Park, Maryland 20742, USA
[2] Center for Information and Communication Technology, FBK, 38123 Povo - Trento, Italy
[3] Laboratory for Analysis and Architecture of Systems-CNRS, Toulouse, 31077, France

Abstract

We give a formalism and an algorithm for synthesizing controllers to coordinate interactions among hierarchically organized agents. Typical applications are, for example, in harbor or warehouse automation. The formalism models agents as hierarchical input/output automata, and models a system of interacting agents as the parallel composition of the automata. It extends the usual parallel composition operation of I/O automata with a *hierarchical composition* operation for refining abstract tasks into lower-level subtasks. We provide an algorithm to synthesize hierarchically organized controllers to coordinate the agents' interactions in order to drive the system toward desired states. Our contribution is mostly theoretical: we formally define the representation, and present theorems about its properties (i.e., the parallel and hierarchical composition are *distributive* operations), as well as the correctness and completeness of the synthesis algorithm.

1. Motivation

Consider a collection of collaborative agents, having different capabilities and programmed to do different things under different conditions. Given a complex task or goal to accomplish, and a description of how each agent is programmed to behave, how can we organize the agents and manage their interactions in order to jointly accomplish a desired objective?

In this paper we provide a knowledge representation framework and algorithms for the above problem. In our formalism, the agents are represented as hierarchical input/output automata. Our algorithms synthesize a hierarchically organized collection of finite-state controllersfor managing the interactions among the agents in order to achieve the goal.

As a motivating example, consider a warehouse automation infrastructure such as the Kiva system (D'Andrea, 2012) that controls thousands of robots moving inventory shelves to human pickers preparing customers orders. According to (Wurman, 2014), "*planning and scheduling are at the heart of Kiva's software architecture*". Right now, this appears to be done with extensive engineering of the environment, e.g., fixed robot tracks and highly structured inventory organization. A more flexible approach for dealing with contingencies, local failures, modular design and easier novel deployments, would be to model each agent (robot, shelf, refill, order preparation, etc.) through its possible interactions with the rest of the system, and automatically synthesize control programs to coordinate these interactions.

The idea of composing finite-state automata into a larger system has been used for a long time in the area of system specification and verification, e.g., (Harel, 1987). Although less popular, it has also been used in the field of automated planning for applications that naturally call for composition, e.g., planning in web services (Pistore et al., 2005; Bertoli et al., 2010), or for the automation of a harbor or a large infrastructure (Bucchiarone et al., 2012).

In our approach, each agent is modeled as an *input/output automaton* σ whose state transitions are governed by messages that are sent to and received from the other agents. If $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ is a set of such agents, *planning* for them does not mean generating a plan or policy as is typically done in AI planning. Instead, it means synthesizing a *control automaton* σ_c to manage the interactions among the agents in Σ. The agents don't send messages to each other directly, but instead send them to σ_c, which receives their messages and decides which messages to send to the agents to drive them toward a desired goal. Nondeterministic planning techniques can be used for synthesizing σ_c.

Known automata techniques provide the basis to our work, but are subject to several restrictions that limit their scope for our purpose. A large system such as a harbor (Bucchiarone et al., 2012) or a logistics network (Boese & Piotrowski, 2009) is generally both *distributed* and *hierarchical*:

- These aren't tightly-integrated monolithic systems. They are composed of agents that may even be geographically distributed. It is more convenient and scalable to rely on distributed controllers to coordinate their actions.
- Agents are composed hierarchically of components for various subtasks. One chooses which components (from among various alternatives) to incorporate into an agent.

The problem of generating a distributed hierarchy of controllers for such agents is novel in the field. It initially requires a theoretical basis, which is the purpose in this paper (no application nor experimental results are reported here). Our contributions are:

- We formally define the notion of refinement for hierarchical communicating input/output automata, call them IOAs, and propose a formalization of planning and acting problems for interacting agents in this original framework.
- We provide theorems about the main properties of this class of automata. In particular, the operations of parallel composition and refinement are *distributive*. The proof of this critical feature for the synthesis algorithm requires careful developments.
- Distributivity allows us to show that the synthesis of a hierarchical control structure for a set of IOAs can be addressed as a nondeterministic planning problem.
- We propose a new algorithm for solving this problem, and discuss its theoretical properties.

In the rest of the paper we present the representation and its properties, the algorithm for the synthesis of a hierarchical control structure comprised of multiple distributed controllers, we discuss the state of the art, and concluding remarks.

2. Representation

The proposed formalism relies on a class of automata endowed with composition and refinement operations. Furthermore, both agents and their components are modeled as hierarchical IOAs, hence in describing the formalism we sometimes will use "agent" and "component" interchangeably.

Automata. The building block of the representation is a particular input/output automata (IOA) $\sigma = \langle S, s_0, I, O, T, A, \gamma \rangle$, where S is a finite set of *states*, s_0 is the *initial state*, I, O, T and A are finite sets of labels called respectively *input, output, tasks* and *actions*, $\gamma : S \times (I \cup O \cup T \cup A) \to S$ is a deterministic *state transition function*. Our definition of IOA is similar to that (Lynch & Tuttle, 1988) apart from the fact that we also have transitions that are tasks that can be hierarchically refined. The IOA uses its *inputs* and *outputs* to interact with other IOAs and the environment. The semantics of an IOA views inputs as *uncontrollable* transitions, triggered by messages from the external world, while outputs, tasks, and actions are *controllable* transitions, freely chosen to drive the dynamics of the modeled system. An output is a message sent to another IOA; an action has some direct effects on the external world. No precondition/effect specifications are needed for actions, since a transition already spells out the applicability conditions and the effects. A task is refined into a collection of actions. We assume all transitions to be deterministic.

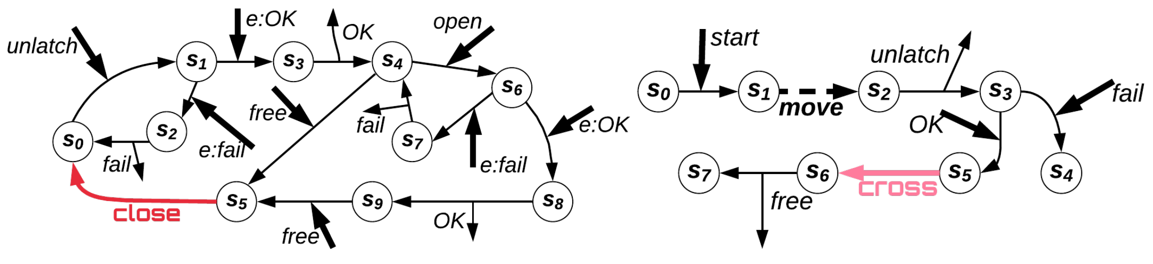
We define a state of an IOA as a tuple of internal state variables each of which keeps track of a particular information relevant for that IOA (a representation similar to the one described in Chapter 2 of (Ghallab et al., 2016)). States are a tuple of state variables' values, i.e., if $\{x_1, \ldots, x_k\}$ are the state variables of σ, and each has a finite range $x_i \in D_i$, then the set of states is $S \subseteq \prod_{i=1,k} D_i$, where D_i is a finite set of values that determine the range of the state variable x_i. We assume that for any state $s \in S$, all outgoing transitions have the same type, i.e., $\{u \mid \gamma(s, u)$ is defined$\}$ consists solely of either inputs, or outputs, or tasks, or actions. For simplicity we assume s can have only one outgoing transition if that transition is an output, action or a task. Alternative actions or outputs can be modeled by a state that precedes s and receives alternative inputs, one of them leading to s.

Note that despite the assumption that our transition function γ is deterministic, an IOA can model nondeterminism through its inputs. It may receive multiple different inputs at any particular state. These inputs can be messages from external world modeling nondeterministic outcomes of events or commands. For example, a sensing action a in state s is a command transition, $\langle s, a, s' \rangle$; several input transitions from s' model the possible outcomes of a; these inputs to σ are generated by the external world.

A *run* of an IOA is a sequence $\langle s_0, u_0, \ldots, s_i, u_i, s_{i+1}, \ldots \rangle$ such that $s_{i+1} = \gamma(s_i, u_i)\ \forall i$. It may or may not be finite.

Example 1. *The IOA in Figure 1(a) models a door with a spring-loaded hinge that closes automatically when the door is open and not held. To open the door requires unlatching it, which may not succeed if it is locked. Then it can be opened, unless it is blocked by some obstacle. Whenever the door is left free, the spring closes it (the "close" action shown in red).*

Parallel Composition. Consider a system $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, with each σ_i modeled as an IOA. These components interact by sending output and receiving input messages, while also triggering actions and tasks. The dynamics of Σ can be modeled by the *parallel composition* of the components, which is a straightforward generalization of the *parallel product* defined in (Bertoli et al.,



(a) (b)

Figure 1. (a): An IOA σ_d for a spring door. The bold incoming arrows represent inputs of σ_d coming from other IOAs or the environment. The outgoing arrows represent messages sent by σ_d to other IOAs. The red transition 'close' is a command. (b): An IOA for a robot going through a doorway.

2010) which is same as the asynchronous product of automata. The parallel composition of two IOAs σ_1 and σ_2 is $\sigma_1 \parallel \sigma_2 = \langle S_1 \times S_2, (s_{0_1}, s_{0_2}), I_1 \cup I_2, O_1 \cup O_2, T_1 \cup T_2, A_1 \cup A_2, \gamma\rangle$,

$$\text{where } \gamma((s_1, s_2), u) = \begin{cases} \gamma_1(s_1, u) \times \{s_2\} & \text{if } u \in I_1 \cup O_1 \cup A_1 \cup T_1, \\ \{s_1\} \times \gamma_2(s_2, u) & \text{if } u \in I_2 \cup O_2 \cup A_2 \cup T_2. \end{cases}$$

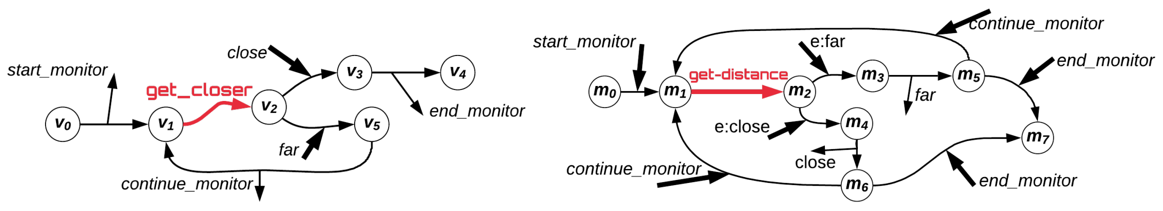
By extension, $\sigma_1 \parallel \sigma_2 \parallel \sigma_3 \parallel \ldots \parallel \sigma_n$ is the parallel composition of all of the IOAs in Σ. The order in which the composition operations is done is unimportant, because parallel composition is associative and commutative.[1]

We assume the state variables, as well as the input and output labels, are *local* to each IOA. This avoids potential confusion in the definition of the composed system. It also allows for a robust and flexible design, since components can be modeled independently and added incrementally to a system. However, the components are cooperative in the sense that all of them have a common goal.

If we restrict the n components of Σ to have *no tasks* but only inputs, outputs and actions, then driving Σ towards a set of *goal* [2] states can be addressed with a nondeterministic planning algorithm for the synthesis of a control automaton σ_c that interacts with the parallel composition $\sigma_1 \parallel \sigma_2 \parallel \sigma_3 \parallel \ldots \parallel \sigma_n$ of the automata in Σ. The control automaton's inputs are the outputs of Σ and its outputs are inputs of Σ. Several algorithms are available to synthesize such control automata, e.g., (Bertoli et al., 2010). But in this paper, we also allow the components to have hierarchy within themselves and we generate a hierarchical control structure.

Hierarchical Refinement. With each task we want to associate a set of *methods* for hierarchically refining the task into IOAs that can perform the task. This is in principle akin to HTN planning (Erol et al., 1994), but if the methods refine tasks into IOAs rather than subtasks, they produce a structure that incorporates control constructs such as branches and loops. This structure is like a hierarchical automaton (see, e.g., (Harel, 1987)). However, the latter relies on a *state hierarchy* (a state gets expanded recursively into other automata), whereas in our case the tasks to be refined are transitions. This motivates the following definition.

1. Proofs of all of the results stated in this paper are at https://www.cs.umd.edu/ patras/long_appendix.pdf.
2. *goal* is represented through a set of states of IOA



(a) (b)

Figure 2. (a): The IOA σ_{move} of a method for the move task. (b): The IOA $\sigma_{monitor}$ of a monitoring method.

A *refinement method* for a task t is a pair $\mu_t = \langle t, \sigma_\mu \rangle$, where σ_μ is an IOA that has both an initial state $s_{0\mu}$ and a *finishing* state $s_{f\mu}$. Unlike tasks in HTN planning (Nau et al., 1999), t is a single symbol rather than a term that takes arguments. Note that σ_μ may recursively contain other subtasks, which can themselves be refined. Consider an IOA $\sigma = \langle S, s_0, I, O, T, A, \gamma \rangle$ that has a transition $\langle s_1, t, s_2 \rangle$ in which t is a task. A method $\mu_t = \langle t, \sigma_\mu \rangle$ with $\sigma_\mu = \langle S_\mu, s_{0\mu}, s_{f\mu}, I_\mu, O_\mu, T_\mu, A_\mu, \gamma_\mu \rangle$ can be used to *refine* this transition by mapping s_1 to $s_{0\mu}$, s_2 to $s_{f\mu}$ and t to σ_t.[3] This produces an IOA, $\mathfrak{R}(\sigma, s_1, \mu_t) = \langle S_\mathfrak{R},\, s_{0\mathfrak{R}},\, I \cup I_\mu,\, O \cup O_\mu,\, T \cup T_\mu \setminus \{t\},\, A \cup A_\mu,\, \gamma_\mathfrak{R} \rangle$, where

$$S_\mathfrak{R} = (S \setminus \{s_1, s_2\}) \cup S_\mu; \qquad s_{0\mathfrak{R}} = \begin{cases} s_0 & \text{if } s_1 \neq s_0, \\ s_{0\mu} & \text{otherwise;} \end{cases}$$

$$\gamma_\mathfrak{R}(s,u) = \begin{cases} \gamma_\mu(s,u) & \text{if } s \in S_\mu \setminus \{s_{0\mu}, s_{f\mu}\}, \\ s_{0\mu} & \text{if } s \in S \text{ and } \gamma(s,u) = s_1, \\ s_{f\mu} & \text{if } s \in S \text{ and } \gamma(s,u) = s_2, \\ \gamma(s,u) & \text{if } s \in S \setminus \{s_1, s_2\} \text{ and } \gamma(s,u) \notin \{s_1, s_2\}, \\ \gamma(s_1,u) \cup \gamma_\mu(s,u) & \text{if } s = s_{0\mu}, \\ \gamma(s_2,u) \cup \gamma_\mu(s,u) & \text{if } s = s_{f\mu}. \end{cases}$$

Note that we do not require every run in σ_μ to actually end in $s_{f\mu}$. Some runs may be infinite, some other runs may end in a state different from $s_{f\mu}$. Such a requirement would be unrealistic, since the IOA of a method may receive different inputs from other IOA, which cannot be controlled by the method. Intuitively, $s_{f\mu}$ represents the "nominal" state in which a run should end, i.e., the nominal path of execution.[4]

Example 2. *Figure 1(b) shows an IOA for a robot going through a doorway. It has one task,* move *and one action,* cross*. It sends to σ_d (Figure 1(a)) the input* free *if it gets through the doorway successfully. The* move *task can be refined using the σ_{move} method in Figure 2(a).*

3. If σ contains multiple calls to t or σ_μ contains a recursive call to t, the states of σ_μ must first be renamed, in order to avoid ambiguity. This is like *standardizing* a formula in automated theorem proving.
4. Alternatively, we may assume we have only runs that terminate, and a set of finishing states $S_{f\mu}$. We simply add a transition from every element in $S_{f\mu}$ to the *nominal* finishing state $s_{f\mu}$.

Example 3. *Figure 2(a) shows a refinement method for the* move *task in Figure 1(b).* σ_{move} *starts with a* start_monitor *output to activate a monitor IOA that senses the distance to a target. It then triggers the task* get_closer *to approach the target. From state* v_2 *it receives two possible inputs:* close *or* far*. When close, it ends the monitor activity and terminates in* v_4*, otherwise it gets closer again.*

Figure 2(b) shows a method for the monitor *task. It waits in state* m_0 *for the input* start_monitor*, then triggers the sensing action* get-distance*. In response, the execution platform may return either* far *or* close*. In states* m_5 *and* m_6*, the input* continue_monitor *goes to* m_1 *to sense the distance again, otherwise the input* end_monitor *goes to the final state* m_7*.*

Planning Problem. We are now ready to define the planning problem for this representation. Consider a system modeled by $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ and a finite collection of methods $\mathcal{M}$, such that for every task t in Σ or in the methods of $\mathcal{M}$ there is at least one method $\mu_t \in \mathcal{M}$ for task t. An *instantiation* of $(\Sigma, \mathcal{M})$ is obtained by recursively refining every task in the composition $(\sigma_1 \parallel \sigma_2 \parallel ...\sigma_n)$ with a method in $\mathcal{M}$, down to primitive actions. Let $(\Sigma, \mathcal{M})^*$ be the set of all possible instantiations of that system, which is enumerable but not necessarily finite. Infinite instances are possible when the body of a method contains the same or another task which can further be refined leading to an infinite chain of refinements. A planning problem is defined as a tuple $P = \langle \Sigma, \mathcal{M}, S_g \rangle$, where S_g is a set of goal states. Each of the initial components in Σ has a set of goal states, and S_g is the Cartesian product of those sets. In other words, the job of the synthesized controller is to make the overall system reach a state such that each component in Σ is in one of its goal states. It is solved by finding refinements for tasks in Σ with methods in $\mathcal{M}$. In principle this is akin to HTN planning, but we have IOAs that receive inputs from the environment or from other IOAs, thus modelling nondeterminism. We need to control the set of IOAs Σ in order to reach (or to try to reach) a goal in S_g. For this reason a *solution* is defined by introducing a *hierarchical control structure* that drives an instantiation of $(\Sigma, \mathcal{M})$ to meet the goal S_g.

We will use the same terminology as in (Ghallab et al., 2016, Section 5.2.3). A *solution* just means that some of the runs will reach a goal state. Other runs may never end, or may reach a state that is not a goal state. A solution is *safe* if all of its finite runs terminate in goal states, and a solution is either *cyclic* or *acyclic* depending on whether it has any cycles.[5]

The hierarchical control structure is a pair $\langle \Sigma_c, \textit{rDict} \rangle$ where Σ_c is a set of control automata and *rDict* is a task refinement dictionary. A single control automaton drives an IOA σ by receiving inputs that are outputs of σ and generating outputs that act as inputs to σ. We represent the controlled system, i.e., σ controlled by σ_c, as $\sigma_c \triangleright \sigma$. The formal definition of controlled system is similar to the one in (Ghallab et al., 2016, Section 5.8). *rDict* is a dictionary which should have as its keys all of the tasks in Σ and its refinement. *rDict*$[t]$ is a method which should be used to refine task t in order to achieve S_g. So, *rDict* uniquely defines an instantiation of $(\Sigma, \mathcal{M})$. Finally, Σ is controlled by $\langle \Sigma_c, \textit{rDict} \rangle$, and the **hierarchical controlled system** $\boldsymbol{\phi_s} = \langle \Sigma_c, \textit{rDict} \rangle \triangleright (\Sigma, \mathcal{M})$ will have one

5. In the terminology of (Cimatti et al., 2003), a weak solution is what we call a solution, a strong cyclic solution is what we call a safe solution, and a strong solution is what we call an acyclic safe solution.

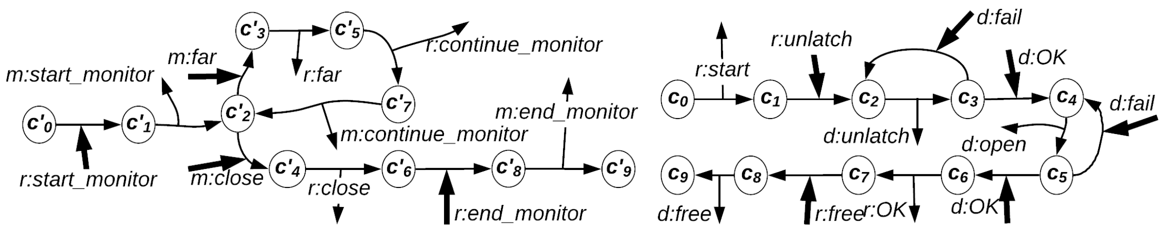


Figure 3. A hierarchical control structure for the 'door' (Figure 1(a)), refined the 'robot' for going through a doorway (Figures 1(b) and 2(a)), and the 'monitor' (Figure 2(b)). The inputs and outputs of the robot, door and monitor are preceded with *r:*, *d:* and *m:* respectively.

of the following forms:

$$\sigma_c \triangleright (\phi_1 \parallel \phi_2), \text{ where } \sigma_c \in \Sigma_c \text{ and } \phi_1, \phi_2 \text{ are IOAs;} \quad (1)$$

$$\sigma_c \triangleright \Re(\phi_3, s, \mu_t), \text{ where } \sigma_c \in \Sigma_c, \textit{rDict}[t] = \sigma_\mu, \phi_3 \text{ is an IOA and } t \text{ is a task in state } s; \quad (2)$$

$$\sigma_c \triangleright \sigma, \text{ where } \sigma_c \in \Sigma_c \text{ and } \sigma \in \Sigma. \quad (3)$$

Above, ϕ_1, ϕ_2 and ϕ_3 are hierarchical controlled systems. The form it will have depends on the ordering of parallel and hierarchical composition chosen by MakeCntrlStruct to synthesize the controller (see Section 3).

Example 4. *The IOA on the right in Figure 3 is a control automaton for the IOAs in Figures 1(a) and 1(b). This control automaton is for the system when the* move *task has not been refined. The IOA on the left controls the refined robot IOA in Figure 2(a) and the monitor IOA in Figure 2(b).*

3. Solving Planning Problems

This section describes our planning algorithm, MakeCntrlStruct (Figure 1(a)). It solves the planning problem $\langle \Sigma, \mathcal{M}, S_g \rangle$ where Σ is the set of IOAs, $\mathcal{M}$ is the collection of methods for refining different tasks and S_g is the set of goal states. The solution is a set of control automata, Σ_c and a task refinement dictionary, *rDict* such that Σ driven by Σ_c and refined following *rDict* reaches the desired goals states, S_g. Depending on how one of its subroutines is configured, MakeCntrlStruct can search either for acyclic safe solutions, or for safe solutions that may contain cycles.

Before getting into the details of how MakeCntrlStruct works, we need to discuss a property on which it depends. Given a planning problem, MakeCntrlStruct constructs a solution by doing a sequence of parallel composition and refinement operations. The following theorem shows that composition and refinement can be done in either order to produce the same result:

Theorem 1 (distributivity). *Let σ_1, σ_2 be IOAs, $\langle s_1, t, s_2 \rangle$ be a transition in σ_1, and $\mu_t = \langle t, \sigma_\mu \rangle$ be a refinement method for t. Then $\Re(\sigma_1, s_1, \mu_t) \parallel \sigma_2 = \Re(\sigma_1 \parallel \sigma_2, s_1^*, \mu_t)$, where $s_1^* = \{(s_1, s) | s \in S_{\sigma_2}\}$*

Thus the algorithm can choose the order in which to do those operations (line (*) in Table 1(a), which is useful because the order affects the size of the search space.

```
MakeCntrlStruct $(\Sigma_0, \mathcal{M}, S_g)$
    $\Sigma \leftarrow \Sigma_0$; $\Sigma_c \leftarrow \emptyset$; rDict $\leftarrow$ empty dictionary
    while (there are unrefined tasks in $\Sigma$ or $|\Sigma| > 1$):
        nondeterministically choose
            which-first $\in \{compose, refine\}$ (*)
        if (which-first = compose):
            select $\sigma_i, \sigma_j \in \Sigma$ and remove them
            $\sigma_{c_{ij}} \leftarrow$ MakeCntrlAutomaton $(\sigma_i \parallel \sigma_j, S_g)$
            if $\sigma_{c_{ij}}$ is a failure, then return failure
            else:
                $\Sigma_c \leftarrow \Sigma_c \cup \{\sigma_{c_{ij}}\}$
                $\Sigma \leftarrow \Sigma \cup \{\sigma_{c_{ij}} \triangleright (\sigma_i \parallel \sigma_j)\}$
        else if (which-first = refine):
            select $\sigma \in \Sigma$ which has task $t$
            (transition $\langle s_1, t, s_2 \rangle$) and remove it
            nondeterministically choose
                a method $\mu_t \in M$ to refine $t$
            $t_{new} \leftarrow$ unique new name for $t$
            rDict$[t_{new}] \leftarrow \sigma_\mu$
            $\Sigma \leftarrow \Sigma \cup \mathfrak{R}(\sigma, s_1, \mu_t)$
    return $\langle \Sigma_c$, rDict$\rangle$
```

(a)

```
ControlledActingWithIOAs $(\Sigma, M, S_g)$
    $\langle \Sigma_c$, rDict$\rangle \leftarrow$ MakeCntrlStruct$(\Sigma, M, S_g)$
    for $\sigma \in \Sigma_c \cup \Sigma$ :
        ExecuteAsync($\sigma$, rDict, $S_g$)

ExecuteAsync($\sigma$, rDict, $S_g$)
    $s \leftarrow$ initial state of $\sigma$
    while $s$ is not final and $s \notin S_g$ do
        $\langle s, a, s' \rangle \leftarrow$ transition coming out of $s$
        switch (type($a$)):
            case input: $a \leftarrow$ ReceiveInput( )
            case output: GenerateOutput($a$)
            case command: ExecuteCommand($a$)
            case task: $\sigma_\mu \leftarrow$ rDict[a]
                    ExecuteSync($\sigma_\mu$, rDict, $S_g$)
        $s \leftarrow \gamma(s, a)$
    if $s \in S_g$ then return Success
    else return Failure
```

(b)

Table 1. (a): Pseudocode for our controller synthesis algorithm. (b): Pseudocode for running IOAs with a synthesized hierarchical controlled structure.

Algorithm. Table 1(a) shows our algorithm for synthesizing hierarchical control structures using planning. It does a sequence of parallel and hierarchical compositions of the IOAs in Σ until there are no more unrefined tasks and all pairs of interacting components have been composed.

As discussed in the previous section, $(\Sigma, \mathcal{M})^*$ is the set of all possible instantiations of our system, which is enumerable but not necessarily finite. Among this set, some instantiations are desirable with respect to our goal. The while loop in MakeCntrlStruct implicitly constructs an instantiation of $(\Sigma, \mathcal{M})$ by doing a series of parallel and hierarchical compositions. In each iteration of the loop the algorithm makes the choice of whether to do a parallel composition or a refinement. The size of the search space depends on the order in which the choices are made. In an implementation, the choice would be made heuristically. We believe some of the heuristics will be analogous to constraint-satisfaction heuristics (Dechter, 2003; Russell & Norvig, 2009). The while loop exits when the implicit instantiation of $(\Sigma, \mathcal{M})$ is complete, i.e., there are no more tasks to refine, and all interactions between pairs of IOAs have been taken into account through parallel composition.

When MakeCntrlStruct chooses to *compose*, it uses the MakeCntrlAutomaton subroutine to create a control automaton $\sigma_{c_{ij}}$ for a pair of IOAs σ_i and σ_j which interact with each other. σ_i and σ_j are randomly selected from Σ. We do not include pseudocode for MakeCntrlAutomaton, because it may be any of several planning algorithms published elsewhere. For example, the algorithm in

(Bertoli et al., 2010) will generate an acyclic safe solution if one exists, and (Bertoli et al., 2010) discusses how to modify that algorithm so that it will find safe solutions that aren't restricted to be acyclic. Several of the algorithms in (Ghallab et al., 2016, Chapter 5) could also be used. If MakeCntrlAutomaton succeeds, we include $\sigma_{c_{ij}}$ in our set of solution control automata, Σ_c and add the controlled system, $\sigma_{c_{ij}} \triangleright (\sigma_i \parallel \sigma_j)$ to Σ. Otherwise, we fail and terminate this nondeterministic branch. Note that, we could allow new components to enter the system at this stage as follows. Instead of selecting σ_j randomly from Σ, we could lookup the components that interact with σ_i through a specific method and include these new components in Σ. Then, we could select σ_j from updated Σ. This simple extension allows new agents to join in at any stage of the synthesis without compromising on the correctness.

When MakeCntrlStruct chooses to *refine*, it chooses a refinement method μ_t selected from $\mathcal{M}$ to refine t. The task refinement dictionary *rDict* maps every instance of all tasks present in Σ to the body of the most optimal refinement method for them. So, we add σ_μ (the body of method μ_t) to the task refinement dictionary *rDict* with key t_{new}. Notice that we rename the task t to t_{new} to identify every instance of task t uniquely. Then, we add the resulting IOA, after doing the refinement, to Σ and continue the loop.

MakeCntrlStruct is sound and complete; footnote 1 has a link to the proof. Completeness guarantees that we find the hierarchical control structure when it exists, but does not guarantee that our algorithm will terminate or return "no" when there is no control structure for the problem.

4. Planning and Acting

In order to run a set of agents, represented through $\langle \Sigma, \mathcal{M} \rangle$ to achieve a common goal S_g, we need to choose among alternative methods $\mu_t \in \mathcal{M}$ for refining a task t, and alternative inputs in a state s that is followed by distinct actions or outputs. These decisions are determined by the controller synthesis algorithm in Table 1(a), through the synthesis of a pair $\langle \Sigma_c, \textit{rDict} \rangle$. Table 1(b) gives pseudocode for running the IOAs using a synthesized hieararchical control structure. We run all the IOAs in $\Sigma \cup \Sigma_c$ asynchronously, while *(i)* triggering the only action or output associated to a state whose outgoing transition is an action or an output, and *(ii)* following the received input for a state whose outgoing transitions are inputs. In some states, these received inputs are nondeterministic outputs from the external world. Hence, the hierarchical controlled system $\boldsymbol{\phi}_s$ formed by $\langle \Sigma_c, \textit{rDict} \rangle$ and Σ can be viewed as a classical reactive system, which interacts deterministically with a nondeterministic external world. Acting according to a deterministic automaton may seem straightforward in general, but in the proposed framework it raises several important issues that still lie ahead in our research agenda, e.g., *monitoring* and *interleaving* acting and planning.

Planning for unsafe solutions is generally easier than planning for a *safe* solution, which may not exist. If the solution is unsafe, monitoring needs to check whether the interaction with the external world is driving the system away from the intended goals and whether replanning is needed.

Interleaving planning and acting, which is particularly desirable given the hierarchical nature of our framework and the interaction with a nondeterministic external world. This corresponds to one of the motivation of our proposed framework. The idea here is to ignore some of the tasks in the planning stage and refine these tasks at the acting stage only. This will provide a basis for future work (see Section 6) on online synthesis of distributed controllers at acting time.

5. Related Work

To the best of our knowledge, there is no previous formalism for the synthesis of hierarchical distributed controllers for coordinating multiple agents. (Ghavamzadeh et al., 2006), (Osentoski & Mahadevan, 2010) and (Jong et al., 2008) use the notion of hierarchy for multi-agent reinforcement learning. These works allow for a hierarchical representation of the target plan, to be executed in a collaborative manner. In our framework, the hierarchical representation is in the agent itself; the synthesized controllers coordinate interactions among hierarchical agents.

(Atkin et al., 2001) proposes a Hierarchical Agent Control Architecture (HAC) with a hierarchical representation of actions, sensors, and goals. HAC includes a least-commitment partial hierarchical planner, relying on plan skeletons. Given a set of goals, plans are retrieved, simulated, and executed. HAC combines hierarchical planning with reasoning by procedural knowledge. Our approach is different since we allow for reasoning about alternative refinements of tasks through the automated synthesis of controllers. Hierarchical and procedure based frameworks have been used in robotic systems, e.g., PRS (Ingrand et al., 1996), RAP (Firby, 1987), TCA (Simmons, 1992; Simmons & Apfelbaum, 1998), XFRM (Beetz & McDermott, 1994), and the survey of (Ingrand & Ghallab, 2014). These approaches propose reactive systems, but none of them is based on a formal account with the synthesis techniques provided in this paper.

(Hu & Feijs, 2003) describes an agent-based architecture for networked devices, where each agent has a controller. However, the controller does not control inter-agent communication, and no synthesis of interactions is provided.

Hierarchical planning formalisms (including angelic hierarchical planning (Marthi et al., 2007) and its extension (Marthi et al., 2008), (Kuter et al., 2009)) do not represent agents that interact among each other and with the external environment. The hierarchical framework proposed in (Shivashankar et al., 2012) refines goals instead of tasks; no synthesis of controllers is provided.

Our approach shares some similarities with the hierarchical state machines of (Harel, 1987), which have been used for the specification and verification of reactive systems. We rely on the theory of input/output automata (Lynch & Tuttle, 1988), which has been used to specify distributed discrete event systems, and to formalize and analyse communication and concurrent algorithms. There is also a vast amount of literature on controllers for discrete-event systems, e.g., (Wong & Wonham, 1996; Mohajerani et al., 2011). All these works focus on the verification rather than on the synthesis of hierarchical agents through input/output automata. The work in (Kessler et al., 2004) is based on hierarchical state machines, however no automated synthesis is provided.

I/O automata have also been used to formalize non hierarchical interactions of web services and to plan for their composition (Pistore et al., 2005; Bertoli et al., 2010). Our work is also different from the work in (Bucchiarone et al., 2012, 2013), where abstract actions are represented with goals, and where (online) planning can be used to generate interacting processes that satisfy such goals.

Our contribution builds on the approach described in (Ghallab et al., 2016, Section 5.8). There, a system having multiple components is defined by the parallel composition of their automata

$\sigma_1 \| \ldots \| \sigma_n$, which describes the possible evolutions of the n components. A planner for that system synthesizes a control automaton that interacts with the σ_i's to drive the system to specified goals. The approach is shown to be solvable with nondeterministic planning algorithms. It is however limited to flat nonhierarchical automata.

6. Conclusions and Future Work

We have developed a formalism for synthesizing hierarchical control structure for systems that are composed of communicating components. This synthesis is done by combining parallel composition of I/O automata with hierarchical refinement of tasks into I/O automata. This approach can be used to synthesize plans that are not just sequences of actions, but include rich control constructs such as conditional and iterative plans. For synthesis of such plans, we describe a novel planning algorithm for synthesizing hierarchical control structure, that can deal with hierarchical refinements.

We believe this work will be important as a basis for algorithms for online synthesis of real-time systems, e.g.,, for web services, automation of large physical facilities such as warehouses or harbors, etc. In our future work, we intend to implement our algorithm and test it on representative problems from such problem domains. For that purpose, an important topic of future work will be to extend our algorithm for use in continual online planning. This should be straightforward, since our acting algorithm already synthesizes the control structure online (see last paragraph of Section 4). As another topic for future work, recall that Theorem 1 (Distributivity) shows that parallel and hierarchical composition operations can be done in either order and produce the same result. The size of the planner's search space depends on the order in which these operations are done, and we want to develop heuristics for choosing the best order.

Finally, there are several ways in which it may be useful to generalize our formalism. One is to allow tasks and methods to have parameters, so that a method can refine a variety of related tasks. Another is to extend the formalism to allow collaboration of two or more methods on a single task.

References

Atkin, M., King, G., Westbrook, D., Heeringa, B., & Cohen, P. (2001). Hierarchical agent control: A framework for defining agent behavior. *AAMAS*.

Beetz, M., & McDermott, D. (1994). Improving robot plans during their execution. *AIPS*.

Bertoli, Pistore, & Traverso (2010). Automated composition of Web services via planning in asynchronous domains. *Artif. Intel.*, *174*, 316–361.

Boese, F., & Piotrowski, J. (2009). Autonomously controlled storage management in vehicle logistics applications of RFID and mobile computing systems. *Intl. J. RF Tech: Res. and Appl.*.

Bucchiarone, A., Marconi, A., Pistore, M., & Raik, H. (2012). Dynamic adaptation of fragment-based and context-aware business processes. *ICWS*.

Bucchiarone, A., Marconi, A., Pistore, M., Traverso, P., Bertoli, P., & Kazhamiakin, R. (2013). Domain objects for continuous context-aware adaptation of service-based systems. *ICWS*.

Cimatti, Pistore, Roveri, & Traverso (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intel.*, *147*, 35–84.

D'Andrea, R. (2012). A revolution in the warehouse: A retrospective on Kiva Systems and the grand challenges ahead. *IEEE Trans. Automation Sci. and Engr.*, *9*, 638–639.

Dechter, R. (2003). *Constraint processing*.

Erol, K., Hendler, J., & Nau, D. (1994). UMCP: A sound and complete procedure for hierarchical task-network planning. *AIPS* (pp. 249–254).

Firby, R. (1987). An investigation into reactive planning in complex domains. *AAAI* (pp. 202–206).

Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated planning and acting*.

Ghavamzadeh, M., Mahadevan, S., & Makar, R. (2006). Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, *13*, 197–229.

Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Sci. of Comput. Prog.*, *8*.

Hu, J., & Feijs, L. (2003). An agent-based architecture for distributed interfaces and timed media in a storytelling application. *AAMAS* (pp. 1012–1013).

Ingrand, F., Chatilla, R., Alami, R., & Robert, F. (1996). PRS: A high level supervision and control language for autonomous mobile robots. *ICRA* (pp. 43–49).

Ingrand, F., & Ghallab, M. (2014). Deliberation for autonomous robots: A survey. *Artif. Intel.*.

Jong, N. K., Hester, T., & Stone, P. (2008). The utility of temporal abstraction in reinforcement learning. *AAMAS* (pp. 299–306).

Kessler, R., Griss, M., Remick, B., & Delucchi, R. (2004). A hierarchical state machine using jade behaviours with animation visualization. *AAMAS*.

Kuter, U., Nau, D., Pistore, M., & Traverso, P. (2009). Task decomposition on abstract states, for planning under nondeterminism. *Artif. Intel.*, *173*, 669–695.

Lynch, N., & Tuttle, M. (1988). An introduction to input output automata. *CWI Quarterly*.

Marthi, B., Russell, S., & Wolfe, J. (2007). Angelic semantics for high-level actions. *ICAPS*.

Marthi, B., Russell, S., & Wolfe, J. (2008). Angelic hierarchical planning: Optimal and online algorithms. *ICAPS* (pp. 222–231).

Mohajerani, S., Malik, R., Ware, S., & Fabian, M. (2011). Compositional synthesis of discrete event systems using synthesis abstraction. *Chinese Control and Decision Conf.* (pp. 1549–1554).

Nau, Cao, Lotem, & Muñoz-Avila (1999). SHOP: Simple hierarchical ordered planner. *IJCAI*.

Osentoski, S., & Mahadevan, S. (2010). Basis function construction for hierarchical reinforcement learning. *AAMAS* (pp. 747–754).

Pistore, M., Traverso, P., & Bertoli, P. (2005). Automated composition of web services by planning in asynchronous domains. *ICAPS* (pp. 2–11).

Russell, S. J., & Norvig, P. (2009). *Artificial intelligence: A modern approach*.

Shivashankar, V., Kuter, U., Nau, D., & Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. *AAMAS* (pp. 981–988).

Simmons, R. (1992). Concurrent planning and execution for autonomous robots. *IEEE Ctrl. Syst.*.

Simmons, R., & Apfelbaum, D. (1998). A task description language for robot control. *IROS*.

Wong, K., & Wonham, W. M. (1996). Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems*, *6*, 241–273.

Wurman, P. (2014). How to coordinate a thousand robots (invited talk). *ICAPS*.

Appendix

In this section, we prove some of the important theoretical results which ensure correctness and completeness of our algorithm to synthesize a hierarchical control structure.[6]

We will prove Theorem 1 by showing that every *run* of $\mathfrak{R}(\sigma_1, s_1, \mu_t) \parallel \sigma_2$ is also a run of $\mathfrak{R}(\sigma_1 \parallel \sigma_2, s_1^*, \mu_t)$, and vice-versa. Recall that a *run* of an IOA is a sequence $\langle s_0, u_0, \ldots, s_i, u_i, s_{i+1}, \ldots \rangle$ such that $s_{i+1} = \gamma(s_i, u_i)$ for every i. To do this, we will define something called a *path* which has a one-to-one correspondence with a run. We will divide a *path* into unique sub-sequences, calling them *projections*, which are responsible for transitions along each of the IOAs involved in a parallel or hierarchical composition. We will see that projections have certain properties in Theorem 2 and 3. We will manipulate these projections using their properties to form new sequences while maintaining a set of constraints that they satisfy. Then we show that satisfying this set of constraints is enough for the sequence to be a path of an IOA (Definition 1), thus proving Theorem 1. Let us start by defining a path.

A path of an IOA, $\sigma = \langle S, s^0, I, O, T, A, \gamma \rangle$ is a sequence of edge labels (see Figure 4) of the form $\langle a_1, a_2, ..., a_n \rangle$, with $a_i \in I \cup O \cup T \cup A$ such that there is a sequence of states $\langle s_0, s_1, ...s_n \rangle$ with $s_0 = s^0$, $s_i = \gamma(s_{i-1}, a_i)$. In general, such executions may be finite or infinite. A path is said to be closed if it is finite and if the last state, s_n is final, i.e., s_n has no edges coming out of it. *A path of an IOA corresponds to a unique run and a run of an IOA corresponds to a unique path.*

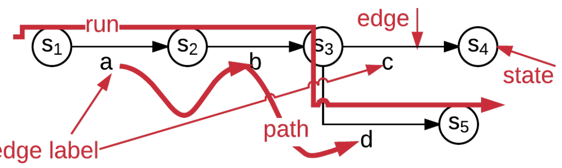


Figure 4. Examples of a state, edge, edge label, run, and path.

A parallel composition puts together two IOAs (say σ_1 and σ_2), that can evolve independently. Any state in $\sigma_1 \parallel \sigma_2$ is of the form, (s_1, s_2) where s_1 comes from σ_1 and s_2 comes from σ_2. As a result, we have edges in $\sigma_1 \parallel \sigma_2$ that correspond to edge labels of σ_1 which changes s_1 but s_2 remains unchanged, as well as edge labels that correspond to edges of σ_2 which changes s_2, but s_1

6. In this section we sketch the proofs. For detailed proofs, see https://www.cs.umd.edu/ patras/long_appendix.pdf.

remains unchanged. With unique names for edge labels of σ_1 and σ_2, we can determine whether an edge label is coming from σ_1 or σ_2. We can decompose any closed path, say α, of $\sigma_1 \parallel \sigma_2$ into two unique paths, called $proj_{\sigma_1}(\alpha)$ and $proj_{\sigma_2}(\alpha)$, each corresponding to closed paths of σ_1 and σ_2 respectively. We do this decomposition because we are interested in separating out the ordering of edge labels (a partial order) in a path that is relevant for a path to be a closed path of $\sigma_1 \parallel \sigma_2$.

Theorem 2. *The projections for parallel composition,* $\boldsymbol{proj}_{\sigma_1}(\alpha)$ *and* $\boldsymbol{proj}_{\sigma_2}(\alpha)$ *are unique sub-sequences of any closed path* α *of IOA* $\sigma_1 \parallel \sigma_2$ *such that:*

- $proj_{\sigma_1}(\alpha)$ *is a closed path of IOA* σ_1 *and* $proj_{\sigma_2}(\alpha)$ *is a closed path IOA* σ_2
- $|proj_{\sigma_1}(\alpha)| + |proj_{\sigma_2}(\alpha)| = |\alpha|$ *and* $\{a_1 | a_1 \in proj_{\sigma_1}(\alpha)\} \cap \{a_2 | a_2 \in proj_{\sigma_2}(\alpha)\} = \emptyset$

Similar to projections for parallel composition, we also define projections for hierarchical composition which decomposes any closed path, α of a refined IOA into sub-sequences, one of which is responsible for edges along the states of the body of the refinement method and another which is responsible for edges along the states of the IOA being refined. These sub-sequences, which we will call $proj_{\sigma_1}(\alpha)$ and $proj_{\mu_t}(\alpha)$, will always be unique as well.

Theorem 3. *The projections for hierarchical composition,* $\boldsymbol{proj}_{\sigma}(\alpha)$ *and* $\boldsymbol{proj}_{\mu_t}(\alpha)$ *for any closed path* α *of IOA* $\mathfrak{R}(\sigma, s_1, \mu_t)$ *with* $\mu_t = \langle t, \sigma_\mu \rangle$ *being a refinement method for task t, are unique sub-sequences of* α *satisfying the following properties.*

If refinement of t is a substring of α *(Figure 5), then* $proj_{\sigma}(\alpha)$ *is a path of* σ *that is either closed or ends with t, and* $proj_{\mu_t}(\alpha)$ *is a closed path of* $body(\mu_t)$*, such that*

$$|proj_{\sigma}(\alpha)| + |proj_{\mu_t}(\alpha)| = |\alpha| + 1 \quad \text{and} \quad \{a_1 | a_1 \in proj_{\sigma}(\alpha)\} \cap \{a_2 | a_2 \in proj_{\mu_t}(\alpha)\} = \emptyset.$$

If refinement of t is not a substring of α*, then* $\boldsymbol{proj}_{\sigma}(\alpha) = \alpha$ *is a closed path of* σ *and* $\boldsymbol{proj}_{\mu_t}(\alpha) = \langle\rangle$.

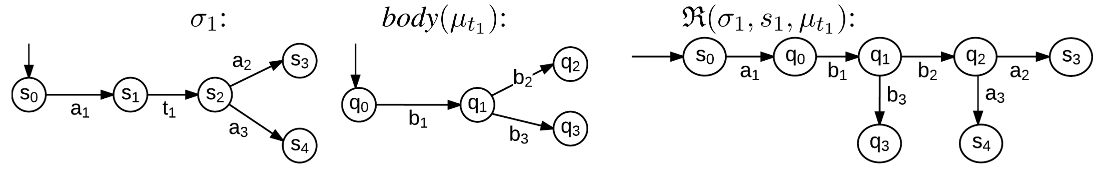


Figure 5. Three IOAs: σ_1, *body*(μ_{t_1}), and $\mathfrak{R}(\sigma_1, s_1, \mu_{t_1})$. Note that $a_1b_1b_2a_2$ is a closed path of $\mathfrak{R}(\sigma_1, s_1, \mu_{t_1})$, with $proj_{\sigma_1}(a_1b_1b_2a_2) = a_1t_1a_2$ and $proj_{\mu_t}(a_1b_1b_2a_2) = b_1b_2$.

In an IOA σ, there may be multiple edges with the same edge label. Thus σ may have multiple paths formed by rearranging the same edge labels (e.g., see Figure 6). In such cases, we want to find the relevant set of constraints that these paths should satisfy to be paths of σ.

Definition 1. *The set of constraints* ***PO(***α, σ***)*** *for a path* α *of an IOA* σ *is the set of relevant edge label orderings such that satisfying these constraints is a sufficient condition for* α *to be a closed path. In other words, any rearrangement of* α *that satisfies* $PO(\alpha, \sigma)$ *will be a closed path of* σ.

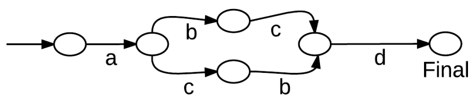


Figure 6. For the IOA σ shown here, the path constraints are $PO(\langle a,b,c,d\rangle,\sigma) = PO(\langle a,c,b,d\rangle,\sigma) = \{a \prec b, a \prec c, c \prec d, b \prec d\}$. The constraint $a \prec b$ means that a should appear before b in a path.

We are now ready to prove **Theorem 1**. In order to show that $\mathfrak{R}(\sigma_1, s_1, \mu_t) \parallel \sigma_2 = \mathfrak{R}(\sigma_1 \parallel \sigma_2, s_1^*, \mu_t)$, we need to show that every closed path of $\mathfrak{R}(\sigma_1, s_1, \mu_t) \parallel \sigma_2)$ is a closed path of $\mathfrak{R}(\sigma_1 \parallel \sigma_2, s_1^*, \mu_t)$ and vice-versa. We only prove the forward direction here (see footnote 6).

Let α be a closed path of $\mathfrak{R}(\sigma_1, s_1, \mu_t) \parallel \sigma_2$. Then, from Theorem 2, we have $\mathit{proj}_{\mathfrak{R}(\sigma_1,s_1,\mu_t)}(\alpha)$ is a closed path of $\mathfrak{R}(\sigma_1, s_1, \mu_t)$ and $\mathit{proj}_{\sigma_2}(\alpha)$ is a closed path of σ_2. Now, the proof is dependent upon whether $\mathit{proj}_{\mathfrak{R}(\sigma_1,s_1,\mu_t)}$ has a refinement of t as its substring.

Case 1: α does not have a refinement of t as its substring. Then from Theorem 3, $\mathit{proj}_{\sigma_1}(\mathit{proj}_{\mathfrak{R}(\sigma_1,s_1,\mu_t)}(\alpha)) = \mathit{proj}_{\mathfrak{R}(\sigma_1,s_1,\mu_t)}(\alpha)$ is a closed path of σ_1. Now, we will make use of the following lemma.

Lemma 1. ***If*** *α_1 is a closed path of IOA σ_1 and α_2 is a closed path of IOA σ_2,* ***then*** *$\alpha_1.\alpha_2$ is a closed path of $\sigma_1 \parallel \sigma_2$ and $PO(\alpha_1.\alpha_2, \sigma_1 \parallel \sigma_2) = PO(\alpha_1, \sigma_1) \cup PO(\alpha_2, \sigma_2)$.*

Let $\beta = \mathit{proj}_{\mathfrak{R}(\sigma_1,s_1,\mu_t)}(\alpha)$. Then β and $\mathit{proj}_{\sigma_2}(\alpha)$ are closed paths of σ_1 and σ_2, so from Lemma 1, $\beta.\mathit{proj}_{\sigma_2}(\alpha)$ is a closed path of $\sigma_1 \parallel \sigma_2$ and $PO(\beta.\mathit{proj}_{\sigma_2}(\alpha), \sigma_1 \parallel \sigma_2) = PO(\beta, \sigma_1) \cup PO(\mathit{proj}_{\sigma_2}(\alpha), \sigma_2)$. α satisfies this set of constraints because β is a sub-sequence of α. Thus, α is a closed path of $\sigma_1 \parallel \sigma_2$. Now, since α does not have a refinement of t as its substring, it is independent of whether t has been refined or not. As a result, α is a closed path of $\mathfrak{R}(\sigma_1 \parallel \sigma_2, s_1^*, \mu_t)$.

Case 2: α has a refinement of t as its substring. Then, from Theorem 3, $\mathit{proj}_{\sigma_1}(\mathit{proj}_{\mathfrak{R}(\sigma_1,s_1,\mu_t)}(\alpha))$ (call it ω) is a path of σ_1 ending with t or closed and $\mathit{proj}_{\mu_t}(\mathit{proj}_{\mathfrak{R}(\sigma_1,s_1,\mu_t)}(\alpha))$ (call it β) is a closed path of $\mathit{body}(\mu_t)$. We also conclude that α satisfies

$$\{u \prec v | u \in \beta \wedge v \in \omega \wedge t \prec v\} \cup \{v \prec u | u \in \beta \wedge v \in \omega \wedge v \prec t\} \tag{4}$$

Recall that we know $\mathit{proj}_{\sigma_2}(\alpha)$ is a closed path of σ_2. Thus from Lemma 1, $\mathit{proj}_{\sigma_2}(\alpha).\omega$ is a path of $\sigma_1 \parallel \sigma_2$ ending with t or closed and

$$PO(\mathit{proj}_{\sigma_2}(\alpha).\omega, \sigma_1 \parallel \sigma_2) = PO(\omega, \sigma_1) \cup PO(\mathit{proj}_{\sigma_2}(\alpha), \sigma_2). \tag{5}$$

Now, we will make use of the following lemma.

Lemma 2. ***If*** *β is a closed path of $\mathit{body}(\mu_t)$ for a refinement method μ_t for task t and either $\delta_1.t.\delta_2$ is a closed path of IOA σ_1 or $\delta_1.t.\delta_2$ is just a path with $\delta_2 = \langle\rangle$,* ***then*** *$\delta_1.\beta.\delta_2$ is a closed path of $\mathfrak{R}(\sigma_1, s_1, \mu_t)$ and $PO(\delta_1.\alpha.\delta_2) = PO(\beta, \mathit{body}(\mu_t)) \cup PO(\delta_1.t.\delta_2, \sigma_1) \cup \{(u \prec v) | u \in \beta, v \in \delta_2$ and $(t \prec v) \in PO(\delta_1.t.\delta_2, \sigma_1)\} \cup \{(v \prec u) | u \in \beta, v \in \delta_1$ and $(v \prec t) \in PO(\delta_1.t.\delta_2, \sigma_1)\}$.*

In our problem, we can write $\mathit{proj}_{\sigma_2}(\alpha).\omega$ as $\delta_1.t.\delta_2$ and $\mathit{proj}_{\mu_t}(\mathit{proj}_{\mathfrak{R}(\sigma_1,s_1,\mu_t)}(\alpha))$ as β satisfying the properties required by Lemma 2. Note that δ_2 may be an empty string. So, applying Lemma 2, $\delta_1.\beta.\delta_2$ is a closed path of $\mathfrak{R}(\sigma_1 \parallel \sigma_2, s_1^*, \mu_t)$, and

$$\begin{aligned}
&PO(\delta_1.\beta.\delta_2, \Re(\sigma_1 \parallel \sigma_2, s_1^*, \mu_t)) = PO(\beta, \mathit{body}(\mu_t)) \cup PO(\delta_1.t.\delta_2, \sigma_1 \parallel \sigma_2) \cup \\
&\quad \{u \prec v | u \in \beta, v \in \delta_2 \text{ and } (t \prec v) \in PO(\delta_1.t.\delta_2, \sigma_1 \parallel \sigma_2)\} \cup \\
&\quad \{v \prec u | u \in \beta, v \in \delta_1 \text{ and } (v \prec t) \in PO(\delta_1.t.\delta_2, \sigma_1 \parallel \sigma_2)\} \\
&= PO(\beta, \mathit{body}(\mu_t)) \cup PO(\omega, \sigma_1) \cup PO(\mathit{proj}_{\sigma_2}(\alpha), \sigma_2) \cup \\
&\quad \{u \prec v | u \in \beta, v \in \omega \text{ and } (t \prec v) \in PO(\omega, \sigma_1)\} \cup \\
&\quad \{v \prec u | u \in \beta, v \in \omega \text{ and } (v \prec t) \in PO(\omega, \sigma_1)\} \qquad \text{(from (5))}
\end{aligned}$$

But α is a permutation of $\delta_1.\beta.\delta_2$ that satisfies the above set of constraints because β and ω are projections of α and α satisfies (4). Therefore, α is a closed path of $\Re(\sigma_1 \parallel \sigma_2, s_1^*, \mu_t)$.

Theorem 4 (Correctness/Soundness)**.** *The hierarchical control structure $\langle \Sigma_c, \mathit{rDict} \rangle$ returned by* MakeCntrlStruct *($\Sigma_0, \mathcal{M}, S_g$) is such that the controlled system always reaches the goal states S_g.*

Proof. $\langle \Sigma_c, \mathit{rDict} \rangle \triangleright \langle \Sigma, \mathcal{M} \rangle$ is a hierarchical controlled system $\boldsymbol{\phi_s}$ of one of the forms of expressions 1, 2 or 3 (Page 8).

We do the proof by induction. In the base case, $\boldsymbol{\phi_s}$ is an IOA of the form $\sigma_c \triangleright \sigma$, where $\sigma \in \Sigma_0$ and $\sigma_c \in \Sigma_c$. We synthesize a control automaton for one IOA using procedure in (Bertoli et al., 2010) which is sound. Our induction hypothesis is that for all controlled systems $\boldsymbol{\phi_k}$ of size less than $\boldsymbol{\phi_s}$, $\boldsymbol{\phi_k} \models S_g$. In the inductive step, $\boldsymbol{\phi_s}$ can be of two forms.

Case 1: $\boldsymbol{\phi_s}$ is of the form $\sigma_c \triangleright (\boldsymbol{\phi_1} \parallel \boldsymbol{\phi_2})$. From the induction hypothesis, $\boldsymbol{\phi_1} \models S_g$ and $\boldsymbol{\phi_2} \models S_g$. For synthesizing σ_c, we use the procedure from (Bertoli et al., 2010) to coordinate the interaction between $\boldsymbol{\phi_1}$ and $\boldsymbol{\phi_2}$ which is sound. Therefore, $\boldsymbol{\phi_s} \models S_g$.

Case 2: $\boldsymbol{\phi_s}$ is of the form $\sigma_c \triangleright \Re(\boldsymbol{\phi_3}, s, \mu_t)$. From the induction hypothesis, $\boldsymbol{\phi_3} \models S_g$. For synthesizing σ_c, we use the procedure from (Bertoli et al., 2010) to coordinate the interaction between $\boldsymbol{\phi_3}$ and σ_μ which is sound. Therefore, $\boldsymbol{\phi_s} \models S_g$. □

Now, we will prove that MakeCntrlStruct is also complete. But before that, let us state another result about controlled systems which will be used in the proof of completeness.

Theorem 5. *For controlled systems $\boldsymbol{\phi}_1$, $\boldsymbol{\phi}_2$ and refinement method μ_t for task t, **if** there exists two control automata, σ_{c_1} and σ_{c_2}, such that the controlled system $\sigma_{c_1} \triangleright ((\sigma_{c_2} \triangleright \Re(\boldsymbol{\phi}_1, s_1, \mu_t)) \parallel \boldsymbol{\phi}_2)$ satisfies goal S_g **then** there also exists two control automata σ'_{c_1} and σ'_{c_2} such that the controlled system $\sigma'_{c_1} \triangleright (\Re(\sigma'_{c_2} \triangleright (\boldsymbol{\phi}_1 \parallel \boldsymbol{\phi}_2), s_1^*, \mu_t))$ satisfies S_g **and** vice-versa.*

Proof. We show that if there are control automata σ_{c_1} and σ_{c_2} such that $\sigma_{c_1} \triangleright ((\sigma_{c_2} \triangleright \Re(\boldsymbol{\phi}_1, s_1, \mu_t)) \parallel \boldsymbol{\phi}_2) \models S_g$ for some set of goal states S_g, then there are control automata σ'_{c_1} and σ'_{c_2} such that $\sigma'_{c_1} \triangleright \Re(\sigma'_{c_2} \triangleright (\boldsymbol{\phi}_1 \parallel \boldsymbol{\phi}_2), s^*, \mu_t) \models S_g$ (footnote 6 gives a link to the proof of the converse statement).

Note that σ_{c_2} is a control automaton for refined $\boldsymbol{\phi}_1$. So, it is independent of $\boldsymbol{\phi}_2$. In other words, $\boldsymbol{\phi}_2$ behaves independently whether or not it is controlled by σ_{c_2}. Thus, the system $(\sigma_{c_2} \triangleright \Re(\boldsymbol{\phi}_1, s, \mu_t)) \parallel \boldsymbol{\phi}_2$ functions same as the controlled system $\sigma_{c_2} \triangleright (\Re(\boldsymbol{\phi}_1, s, \mu_t) \parallel \boldsymbol{\phi}_2)$. Because controlled systems are also IOAs, using the Distributivity Theorem (Theorem 1), this is same as the controlled system, $\sigma_{c_2} \triangleright \Re(\boldsymbol{\phi}_1 \parallel \boldsymbol{\phi}_2, s^*, \mu_t)$. Considering σ_{c_1} as well, $\sigma_{c_1} \triangleright (\sigma_{c_2} \triangleright \Re(\boldsymbol{\phi}_1 \parallel \boldsymbol{\phi}_2, s^*, \mu_t))$ satisfies S_g. So, it is possible to construct a control structure for $\Re(\boldsymbol{\phi}_1 \parallel \boldsymbol{\phi}_2, s^*, \mu_t)$. This implies that there are two control automata σ'_{c_1} and σ'_{c_2} such that $\sigma'_{c_1} \triangleright \Re(\sigma'_{c_2} \triangleright (\boldsymbol{\phi}_1 \parallel \boldsymbol{\phi}_2), s^*, \mu_t) \models S_g$.

This can be easily shown by contradiction. Assume that σ'_{c_2} doesn't exist. This means that there is no control automaton such that $\phi_1 \parallel \phi_2 \models S_g$. So, there is no control automaton for any refinement of $\phi_1 \parallel \phi_2$. This is a contradiction. Similarly, assume σ'_{c_1} doesn't exist. σ'_{c_1} controls the interaction between *body*(μ_t) and $\phi_1 \parallel \phi_2$. This is independent of the controller for interaction between ϕ_1 and ϕ_2. If σ'_{c_1} does not exist, then refining task t in the IOA $\phi_1 \parallel \phi_2$ with method μ_t cannot be controlled to satisfy S_g. This is again a contradiction. □

Theorem 6 (Completeness)**.** *The procedure* MakeCntrlStruct *$(\Sigma, \mathcal{M}, S_g)$ returns a solution hierarchical control structure $\langle \Sigma_c, \textit{rDict} \rangle$ if it exists.*

Proof. Let the solution hierarchical controlled system be $\boldsymbol{\phi_s}$ as defined in the proof of Theorem 4 such that $\boldsymbol{\phi_s} \models S_g$. We are considering three cases here as others will be similar.

Case 1: $\boldsymbol{\phi_s} = \sigma_c \triangleright \sigma$. In this case, MakeCntrlStruct will find σ_c at the first step of control automaton synthesis.

Case 2: $\boldsymbol{\phi_s} = \sigma_c \triangleright (\boldsymbol{\phi_1} \parallel \boldsymbol{\phi_2})$ with $\boldsymbol{\phi_1}$ being equal to $\sigma_{c'} \triangleright \mathfrak{R}(\boldsymbol{\phi'}, s, \mu_t)$. Then, from Theorem 5, there exists a control automata $\sigma_{\bar{c}}$ and $\sigma_{\bar{c}'}$ such that $\sigma_{\bar{c}} \triangleright \mathfrak{R}(\sigma_{\bar{c}'} \triangleright (\boldsymbol{\phi'} \parallel \boldsymbol{\phi_2}), s', \mu_t) \models S_g$. Suppose our algorithm chose to do the parallel composition of $\boldsymbol{\phi'}$ and $\boldsymbol{\phi_2}$ before refining t in $\boldsymbol{\phi'}$. Then, it will generate the control automata $\sigma_{\bar{c}}$ and $\sigma_{\bar{c}'}$ and add them to Σ_c, thus, guaranteeing completeness.

Case 3: $\boldsymbol{\phi_s} = \sigma_c \triangleright \mathfrak{R}(\boldsymbol{\phi_1}, s, \mu_t)$ with $\boldsymbol{\phi_1}$ being equal to $\sigma_{c'} \triangleright (\boldsymbol{\phi'} \parallel \phi_2)$. Then, from Theorem 5, there exists control automata $\sigma_{\bar{c}}$ and $\sigma_{\bar{c}'}$ such that $\sigma_{\bar{c}} \triangleright (\sigma_{\bar{c}'} \triangleright \mathfrak{R}(\boldsymbol{\phi'}, s', \mu_t) \parallel \boldsymbol{\phi_2}) \models S_g$. Suppose our algorithm chose to do the refinement of $\boldsymbol{\phi'}$ first and then the parallel composition. Then, it will generate the control automata $\sigma_{\bar{c}}$ and $\sigma_{\bar{c}'}$ and add them to Σ_c, thus, guaranteeing completeness.

For building *rDict*, we nondeterministically explore all applicable methods μ_t for task t, hence ensuring completeness. □