Synthesis of Orchestrations of Transducers for Manufacturing

Giuseppe De Giacomo
Sapienza Università di Roma
Roma, Italy
degiacomo@dis.uniroma1.it

Moshe Y. Vardi
Rice University
Houston, USA
vardi@cs.rice.edu

Paolo Felli
University of Bozen-Bolzano
Bolzano, Italy
pfelli@unibz.it

Natasha Alechina
University of Nottingham
Nottingham, UK
nza@cs.nott.ac.uk

Brian Logan
University of Nottingham
Nottingham, UK
bsl@cs.nott.ac.uk

Abstract

In this paper, we model manufacturing processes and facilities as transducers (automata with output). The problem of whether a given manufacturing process can be realized by a given set of manufacturing resources can then be stated as an orchestration problem for transducers. We first consider the conceptually simpler case of uni-transducers (transducers with a single input and a single output port), and show that synthesizing orchestrations for uni-transducers is EXPTIME-complete. Surprisingly, the complexity remains the same for the more expressive multi-transducer case, where transducers have multiple input and output ports and the orchestration is in charge of dynamically connecting ports during execution.

1 Introduction

Manufacturing is transitioning from a 'mass production' model, in which large volumes of a product are produced at a time, to a 'manufacturing as a service' model where the products to be manufactured are not known in advance and each product may differ from the products manufactured immediately before and immediately after it (TSB 2012; Rhodes 2015). In 'manufacturing as a service', manufacturing resources are advertised and shared between members of a 'manufacturing cloud', and products are manufactured by different enterprises connected via a dynamically configured supply chains (Lu, Xu, and Xu 2014). This trend toward flexible, networked manufacturing systems has been termed the *Fourth Industrial Revolution*, or Industry 4.0, and is viewed as essential to maintain the competitiveness of manufacturing in high-labor cost economies (Kagermann et al. 2013).

Determining the manufacturing and assembly tasks necessary to produce a product and their ordering is termed *process planning* (Groover 2007). In process planning, manufacturing tasks in a process recipe are matched against *manufacturing resources*, e.g., computer/numerical-controlled machines, robots etc., to give an executable process plan that realizes the process recipe. The *process plan* specifies the specific manufacturing resources to be used for each manufacturing and assembly operation, and how materials and parts move between the various manufacturing resources. Process planning is traditionally carried out by manufacturing engineers who are experts in the particular processes used in a specific factory, and is largely a manual process. Such a manual process is uneconomic for the small batch sizes typical of the manufacturing as a service model. Even when the batch size is large enough to justify a human-authored process plan, the time required to produce a plan does not allow manufacturers to bid to manufacture for products in real time. To fully realize the manufacturing as a service vision, process planning must be fully automated.

There is a substantial literature on automation to achieve flexibility in manufacturing, for example (Browne et al. 1984; Sethi and Sethi 1990; ElMaraghy 2005; Bi et al. 2008; Koren et al. 1999; Mehrabi, Ulsoy, and Koren 2000; Smale and Ratchev 2009; Felli, Logan, and Sardina 2016). There has been little work, however, on the automated synthesis of process plans. An exception is (de Silva et al. 2016; Felli et al. 2017), where techniques based on AI behavior composition (De Giacomo, Patrizi, and Sardiña 2013) are proposed to determine whether a particular product is realizable (can be manufactured by a particular set of manufacturing resources), and how the product should be manufactured using the resources. Their approach takes as inputs a *process recipe* and a *production topology* specifying the available manufacturing resources and their interconnection, and outputs a *process plan controller* that specifies the tasks to be executed by each manufacturing resource in the production line. Process recipes and manufacturing resources are represented using labelled transition systems, and they define a special *task simulation relation* that captures the notion of realizability. Controller synthesis is a byproduct of computing the simulation relation, and is polynomial in the size of the topology (which is exponential in the number of resources and polynomial in their size) and exponential in the size of the process recipe and number of resources.

The approach proposed in (de Silva et al. 2016; Felli et al. 2017) involves considerable bookkeeping and is somewhat ad-hoc, which makes it difficult characterize how the synthesis of controllers for manufacturing relates to the existing rich literature and tools on reactive synthesis, e.g., (Grädel, Thomas, and Wilke 2003; De Giacomo et al. 2010; Ehlers et al. 2017). In particular, materials and unfinished parts are represented explicitly and manufacturing operations transform sets of input parts into sets of output parts. Moreover, resources may perform additional low-level actions not explicitly prescribed by the process recipe, including the movement of parts between resources through synchronized transfer operations. In this work we generalize the movement of parts and data in the system and consider both physical and logical connections between machines; we also abstract away the execution of additional low-level actions by focusing only on the observable behavior of resources. We develop a model based on the standard model of input/output transducers that captures the essence of process recipes and manufacturing resources, thus relating the synthesis of process-plan controllers to classical reactive synthesis. We start by modeling process recipes and manufacturing resources as (uni-)transducers. Uni-transducers are automata with output that have a single input port and a single output port. The definition of a composition of a set of uni-transducers is simple, and allows us to introduce the main ideas of our approach. We show that the *orchestration problem* for uni-transducers (whether it is possible to synthesize a controller that specifies the tasks to be executed by each manufacturing resource in a way that exactly mimics the process recipe) is decidable in EXPTIME, and that it is also EXPTIME-hard by a reduction to standard behavior composition. We then consider multi-transducers, which are more expressive and provide a better model for manufacturing processes. They allow us to model, e.g., cutting parts from raw materials and sending different parts to different resources, and combing several input parts into a single assembly. We use the hardness result for uni-transducers to show that the orchestration problem for multi-transducers is also EXPTIME-hard. Somewhat surprisingly, the orchestration problem for multi-transducers is also in EXPTIME.

Technically our problem is a significant extension of behavior and service composition studied in other areas of CS and AI (Berardi et al. 2003; De Giacomo, Patrizi, and Sardiña 2013), though the techniques developed there cannot be directly applied in our setting. Instead, we resort to game theoretic techniques used in LTL reactive synthesis (Pnueli and Rosner 1989; Lustig and Vardi 2009; Ehlers et al. 2017). For our specific problem, however, we can avoid the technical difficulties typical of temporal reactive synthesis, and utilise much simpler safety games (Grädel, Thomas, and Wilke 2003; Cassez et al. 2005; De Giacomo et al. 2010; Ehlers et al. 2017), obtaining a substantially more effective setting that is readily implementable, algorithmically well-behaved, and amenable to advanced forms of optimization.

Our work is also related to supervisory control. However, in supervisory control, the focus is on controlling a plant so as to maintain a safety condition. In our case, synthesis generates an orchestrator to coordinate several available machines so as to realize a target plant. In the simpler case of service composition, the similarities and differences between supervisory control and orchestration have been studied in detail in (Barati and St.-Denis 2015; Felli, Yadav, and Sardiña 2017).

2 Uni-Transducer Setting

A transducer is a finite deterministic automaton with outputs, in particular, we consider here transducers which are Mealy machines (Hopcroft and Ullman 1979). In this section, we consider a simple setting where all machines take a single input and produce a single output in each state. We model both the manufacturing resources (machines) and the process recipe as (uni-)transducers.

Definition 1. *A (uni-)transducer $T = (\Sigma, \Delta, S, s_0, f, g)$ is a deterministic transition system with inputs and outputs, where Σ is the input alphabet, Δ is the output alphabet, S is the set of states, s_0 the initial state, $f : S \times \Sigma \longrightarrow S$ is the transition function (which takes a state and an input symbol and returns the successor state) and $g : S \times \Sigma \longrightarrow \Delta$ is the output function (which returns the output of the transition).*

Note that we assume that a successor state is defined for any pair of state and input symbol; this can be achieved by introducing an 'error state' to which a transition is made if the pair of state and symbol does not make sense, and which has a self-loop on any symbol outputting an error symbol.

A transducer takes an infinite string of symbols from Σ as an input, and outputs an infinite string of symbols from Δ as an output. In a manufacturing setting, one can think of the input and output sequences as manufacturing events. For example, an input event may consist of a manufacturing operation and a part to which the operation is to be applied.

Example 1 (Uni-transducer). To illustrate how transducers can be used to model manufacturing problems, consider the following simple example. As in (Felli, Logan, and Sardina 2016; de Silva et al. 2016), we assume that we are given a *process recipe* specifying the manufacturing process to be realized, and a set of manufacturing resources, e.g., milling machines, painting machines, robots etc., that comprise the manufacturing facilities of a particular factory. The recipe specifies that parts are first cleaned and then painted. Square parts are painted green, round parts are painted yellow. (More generally, the colour of a part could be determined by, e.g., an RFID tag attached to the part, but this keeps things simple.) We have three manufacturing resources: a cleaning machine, and two painting machines, one that paints parts green and the other paints them yellow. We model both the recipe and the manufacturing resources as transducers, depicted in Figure 1. The recipe is a transducer $T = (\Sigma, \Delta, S, s_0, f, g)$ where $\Sigma = \{cleansq, cleanrd, greensq, yellowrd\}$. Each element of the input alphabet encodes an operation (clean, paint green, paint yellow) and the part on which the operation is performed (square sq or round rd). The output alphabet $\Delta = \{sqclean, rdclean, sqgreen, rdyellow, err\}$ encodes the result of performing an operation on a part; for example, $sqclean$ indicates a square part that has been cleaned. err denotes an error condition, and is explained below. $S = \{s_0, s_1, s_2, s_e\}$ where s_0 is the initial state. The transition and output functions are defined as follows: $f(s_0, cleansq) = s_1$ and $g(s_0, cleansq) = sqclean$ (in s_0 on input of a square part, the system goes to state s_1 and outputs a cleaned square part), $f(s_0, cleanrd) = s_2$ and $g(s_0, cleanrd) = rdclean$, $f(s_1, greensq) = s_0$ and $g(s_1, greensq) = sqgreen$, $f(s_2, yellowrd) = s_0$ and $g(s_2, yellowrd) = rdyellow$. All other pairs of states and inputs transit to the error state s_e which has a loop to itself on any symbol, outputting err. Similarly, the manufacturing resources can be represented as transducers T_1, T_2 and T_3

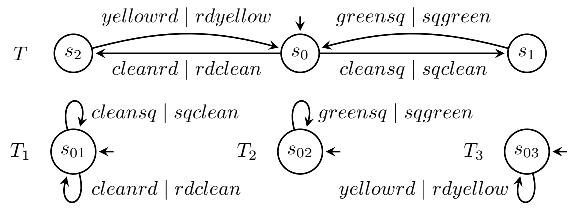

Figure 1: Process recipe T and manufacturing resources T_1, T_2 and T_3. The error states s_{ej} are not depicted

with the same Σ and Δ. For instance, the cleaning machine can be modeled as $T_1 = (\Sigma, \Delta, S_1, s_{01}, f_1, g_1)$, where $S_1 = \{s_{01}, s_{e1}\}$, $f_1(s_{01}, cleansq) = s_{01}$ and $g_1(s_{01}, cleansq) = sqclean$, $f_1(s_{01}, cleanrd) = s_{01}$ and $g_1(s_{01}, cleanrd) = rdclean$. T_2 and T_3 are similar, and are shown in Figure 1.

2.1 Orchestration

We now consider the problem of synthesizing a controller for a fixed set of resources to realize a process recipe in the uni-transducer setting. The task of the controller is to decide, at each cycle, to which resource the input should be assigned.

We are given a set of *available transducers* $\{T_1, \ldots, T_m\}$, where each $T_j = (\Sigma, \Delta, S_j, s_{0j}, f_j, g_j)$ (i.e., all T_j are over the same input and output alphabet) which represent manufacturing resources. We are also given a process recipe T, which is also a transducer with the same input and output alphabets as the T_js. The idea is to combine the resources to be able to match the behavior of T. The behavior of T on input $w = a^0 a^1 \ldots$ is described by the following sequence of states and outputs (where T^s is the state and T^o the output):

$$T^s(a^0) = f(s_0, a^0)$$
$$T^o(a^0) = g(s_0, a^0)$$
$$\ldots$$
$$T^s(a^0 \ldots a^i) = f(T^s(a^0 \ldots a^{i-1}), a^i)$$
$$T^o(a^0 \ldots a^i) = g(T^s(a^0 \ldots a^{i-1}), a^i)$$

The observable output sequence of T on input w is $\tau^o_{w,T} = T^o(a^0), \ldots, T^o(a^0 \ldots a^i) \ldots$

Consider $P = T_1 \times \cdots \times T_m$ (i.e., the transducer corresponding to the whole production facility). A controller for P is a function $C : \Sigma^+ \longrightarrow \{1, \ldots, m\}$ that, for each finite input string, picks a transducer in P to make a transition. The sequence of (global) states generated by the controller on input w is

$$\tau^s_{w,C} = (s^0_1, \ldots, s^0_m), \ldots, (s^i_1, \ldots, s^i_m) \ldots$$

where only one of the local states changes in each transition:

$$s^{i+1}_h = \begin{cases} s^i_h & \text{if } C(a^0 \ldots a^i) \neq h \\ f_h(s^i_h, a^i) & \text{if } C(a^0 \ldots a^i) = h \end{cases}$$

The output of C on P over the input w is $\tau^o_{w,C} = b^0, \ldots, b^i, \ldots$, where $b^i = g_h(s^i_h, a^i)$. C *realizes* T if $\tau^o_{w,T} = \tau^o_{w,C}$ for all w.

Definition 2. *Given a set of transducers $T_1, \ldots, T_m$ and a production recipe transducer T, the orchestration problem is the question whether there is a controller C for $P = T_1 \times \cdots \times T_m$ which realizes T.*

Example 2 (Orchestration). Consider a sequence of inputs $cleansq, greensq, cleanrd, yellowrd \ldots$ The transducer T from Example 1 produces a sequence of outputs $sqclean, sqgreen, rdclean, rdyellow, \ldots$ on this input. A controller C for $T_1 \times T_2 \times T_3$ imitates this behavior by the following mapping:

$$\begin{array}{ll} cleansq \mapsto & 1 \\ cleansq, greensq \mapsto & 2 \\ cleansq, greensq, cleanrd \mapsto & 1 \\ cleansq, greensq, cleanrd, rdyellow \mapsto & 3 \end{array}$$

2.2 Orchestrator Synthesis

The key idea behind our solution to the orchestration problem is that a controller C can be synthesized as a strategy to solve *safety games* (Grädel, Thomas, and Wilke 2003; De Giacomo et al. 2010; Ehlers et al. 2017).[1] Safety games are games between the agent and the environment; we formulate them here as *DFA games* (De Giacomo and Vardi 2015), where the *specification* of the game is given by a Deterministic Finite State Automaton (DFA). A safety game G is defined by a tuple $G = (\mathcal{X} \times \mathcal{Y}, Q, q_0, \delta, F)$, where:

- $\mathcal{X} \times \mathcal{Y}$ is the alphabet of the game;
- Q are the states of the game;
- q_0 is the initial state of the game;
- $\delta : Q \times \mathcal{X} \times \mathcal{Y} \to Q$ is the transition function, i.e., a partial function such that given the current state q and a choice of symbols X and Y for the environment and the agent, $\delta(q, (X, Y)) = q'$ is the resulting state of the game.
- $F = \emptyset$ are the final states of the game, i.e., the game should never terminate.

A *round* of the game consists of both the agent and the environment setting the values of the variables they control. A (complete) *play* is a (possibly infinite) word in $(\mathcal{X} \times \mathcal{Y})^* \cup (\mathcal{X} \times \mathcal{Y})^\omega$ describing how the agent and environment set their variables at each round until the game stops (possibly never). A play is *winning* for the agent if such a play is infinite, that is the agent can continue play forever. A *strategy* for the agent is a function $f : (\mathcal{X})^+ \to \mathcal{Y}$ that, given a history of choices from the environment, decides which variables $\mathcal{Y}$ to set to true/false next. A *winning strategy* is a strategy $f : (\mathcal{X})^+ \to \mathcal{Y}$ such that for all play π with $Y_i = f(\pi_{\mathcal{X}}|_i)$ (i.e., Y_i played according to f), we have that π continues forever.

In our orchestration problem, $\mathcal{X}$ are the inputs of the target transducer T, which are not under the control of the orchestrator, and $\mathcal{Y}$ are the indexes of the transducer to which the input is sent and which returns the corresponding output.

To actually compute the strategy, we start by defining the *controllable preimage* $PreC(\mathcal{E})$ of a set $\mathcal{E}$ of states of G as

[1] Also related to checking nonemptiness of looping tree automata (Vardi and Wolper 1986).

the set of states s such that there exists a choice of values for variables $\mathcal{Y}$ such that for all choices of values for variables $\mathcal{X}$, G progresses to states in $\mathcal{E}$. Formally:

$$\begin{aligned}PreC(\mathcal{E}) = \{q \in Q \mid & \text{ for all } X \in \mathcal{Y} \\ & \text{exists } Y \in \mathcal{Y} \text{ such that } \delta(q, (X, Y)) \in \mathcal{E}\}\end{aligned}$$

We then define the set $Win(G)$ of winning states of the safety game G, i.e., the set of states from which the agent can win G, as a *greatest-fixpoint*, by making use of approximates $Win_i(G)$ denoting all states where the controller wins in at most i steps:

- $Win_0(G) = S$ (all states of G);
- $Win_{i+1}(G) = Win_i(G) \cap PreC(Win_i(G))$.

Then, $Win(G) = \bigcap_i Win_i(G)$. Notice that computing $Win(G)$ requires *linear time* in the number of states in G. Indeed, after at most a linear number of steps $Win_{i+1}(G) = Win_i(G) = Win(G)$. A safety game G admits a winning strategy iff $q_0 \in Win(G)$.

Then, we define a *strategy generator* based on the winning sets $Win_i(G)$. This is a nondeterministic transducer, where nondeterminism is of the "don't-care" variety: all nondeterministic choices are equally good. The strategy generator $\mathcal{T}_G = (\mathcal{X} \times \mathcal{Y}, Q, q_0, \varrho, \gamma)$ is as follows:

- $\mathcal{X} \times \mathcal{Y}$ is the alphabet of the transducer;
- Q are the states of the transducer;
- q_0 is the initial state;
- $\varrho : Q \times \mathcal{X} \to 2^Q$ is the transition function such that
  $$\varrho(s, X) = \{q' \mid q' = \delta(q, (X, Y)) \text{ and } Y \in \gamma(s, X)\};$$
- $\gamma : Q \times \mathcal{X} \to \mathcal{Y}$ is the output function such that
  $$\begin{aligned}\gamma(q, X) = \{Y \mid & \text{ if } q \in Win(G) \\ & \text{then } \delta(q, (X, Y)) \in Win(G)\}.\end{aligned}$$

The transducer $\mathcal{T}_G$ generates strategies in the following sense: for every way of restricting $\gamma(q, X)$ to return only one of its values (chosen arbitrarily), we get a strategy.

It is notable that there are efficient backward and forward algorithms for solving safety games that are amenable to symbolic implementation (Grädel, Thomas, and Wilke 2003; Cassez et al. 2005).

For our orchestration problem, we define the corresponding safety game as follows. Given the target transducer T and the available transducers $\{T_1, \ldots, T_m\}$ we build the safety game $G = (\Sigma, \{1, \ldots m\}, Q, q_0, \delta)$ where

- Σ is the input alphabet;
- $Q = S \times S_1 \times \cdots \times S_m \cup \{q_{err}\}$ is the cartesian product of the states of the target and the available transducers, plus a special error state q_{err}; the state $(s, s_1, \ldots, s_m)$ stores the state of each of the transducers, including the target;
- $q_0 = (s_0, s_{01}, \ldots, s_{0m})$;
- $\delta : Q \times \Sigma \times \{1, \ldots, m\} \to Q$ is defined as follows:
  - $\delta((s, s_1, \ldots, s_m), a, h) = (s', s'_1, \ldots, s'_m)$ with $s' = f(s, a)$, $s'_h = f_h(s_h, a)$ and $s'_j = s_j$ for $j \neq h$, if $g(s, a) = g_h(s_h, a)$;
  - $\delta((s, s_1, \ldots, s_m), a, h) = q_{err}$, for all $a \in \Sigma$, if $g(s, a) \neq g_h(s_h, a)$.

Notice that the only state that does not have transitions is q_{err}, which has no transitions at all. All other states do have transitions for all $a \in \Sigma$. It is easy to see that every strategy for the safety game G corresponds to a controller C for P realizing T and vice versa. Hence we have:

Theorem 1. *Checking whether there exists a controller C for P that realizes T can be done by solving the safety game G defined above.*

By the discussion above, Theorem 1 gives us an implementable algorithm for synthesizing the controller.

Considering that the number of states of G is polynomial in the states of T and exponential in the number m, and that computing a winning strategy for G is linear, we get:

Theorem 2. *Checking whether C realizes T can be done in EXPTIME, and in particular is polynomial in T and exponential in the number m of available transducers.*

2.3 Lower Bound

We reduce service composition in the Roman model (Berardi et al. 2003) which is known to be EXPTIME-complete to the case of bounded orchestration (Muscholl and Walukiewicz 2008; De Giacomo, Patrizi, and Sardiña 2013). Let us consider services of the form $B_j = (\Sigma, S_j, s_{0j}, \delta_j)$, where Σ is the set of actions shared by all services, S_j is the finite set of states, $s_{0j} \in S_j$ is the initial state and $\delta_j : S_j \times \Sigma \mapsto \Sigma$ is the transition function. δ_j may be *partial*, i.e., in certain states certain actions are not allowed. For simplicity, as in (Muscholl and Walukiewicz 2008) where the EXPTIME-hardness is proven, we do not consider final states as in (Berardi et al. 2003; De Giacomo, Patrizi, and Sardiña 2013).

We extend δ_j to an execution function $\hat{\delta}_j$, which is a total function on states and words defined as follows:

$$\hat{\delta}_j : S_j \times \Sigma^* \to S_j \cup \{\bot\}$$

where $\hat{\delta}_j(s, \epsilon) = s$ and

$$\hat{\delta}_j(s, w \cdot a) = \begin{cases} \delta_j(\hat{\delta}_j(s, w), a) & \text{if } \hat{\delta}_j(s, w) = s' \neq \bot \\ & \text{and } \delta_j(s', a) \text{ is defined} \\ \bot & \text{otherwise} \end{cases}$$

Note that given a word w such that $\hat{\delta}_j(s, w) = \bot$ then $\hat{\delta}_j(s, w \cdot a) = \bot$, for all a.

Now consider a set $B_1, \ldots B_m$ of available services, and a target service B_0. An orchestrator O for B_0 on $B_1, \ldots B_m$ is a partial function of the form $O : \Sigma^+ \mapsto \{1, \ldots, m\}$ such that $O(w)$ is defined iff $\hat{\delta}_0(s_{00}, w) \neq \bot$. We also define the execution function $\hat{\delta}_O$ of $B_1, \ldots B_m$ orchestrated by O as:

$$\hat{\delta}_O : S_1 \times \cdots \times S_m \times \Sigma^* \to (S_1 \times \cdots \times S_m) \cup \{\bot\}$$

where $\hat{\delta}_O(s_1, \ldots, s_m, \epsilon) = s_1, \ldots, s_m$ and

$$\hat{\delta}_O(s_1, \ldots, s_m, w \cdot a) = \begin{cases} s'_1, \ldots, s'_h \ldots, s'_m \\ \quad \text{if } \hat{\delta}_O(s_1, \ldots, s_m, w) = \\ \quad\quad s'_1, \ldots, s''_h \ldots, s'_m \neq \bot \\ \quad \text{and } O(w \cdot a) = h \\ \quad \text{and } \delta_h(s''_h, a) = s'_h \\ \bot \text{ otherwise} \end{cases}$$

Using these notions we have that O *realizes* B_0 *over* $B_1, \ldots, B_m$ iff, for all words $w \in \Sigma^*$:

$$\hat{\delta}_0(s_{00}, w) \neq \bot \text{ implies } \hat{\delta}_O(s_{01}, \ldots, s_{0m}, w) \neq \bot.$$

That is, every sequence of actions executable by the target B_0 can also be executed by the available services $B_1, \ldots, B_m$ suitably orchestrated by O. This problem is known to be EXPTIME-complete.

We now reduce this problem to orchestration of transducers. For every service B_j we consider a transducer $T_j = (\Sigma, \Delta, S_j, s_{0j}, f_j, g_j)$ where $\Delta = \{\top, \bot\}$ and

- $f_j(s,a) = \begin{cases} \delta_j(s,a) & \text{if } \delta_j(s,a) \text{ defined} \\ s & \text{otherwise} \end{cases}$
- $g_j(s,a) = \begin{cases} \top & \text{if } \delta_j(s,a) \text{ defined} \\ \bot & \text{otherwise} \end{cases}$

To the transducers $T_0, T_1, \ldots, T_m$, we add an additional dummy transducer: $T_\bot = (\Sigma, \Delta, \{s_\bot\}, s_\bot, f_\bot, g_\bot)$ where $f_\bot(s_\bot, a) = s_\bot$ and $g_\bot(s_\bot, a) = \bot$, for all $a \in \Sigma$.

Let $P = T_0 \times T_1 \times \cdots \times T_m \times T_\bot$. We define the controller $C : \Sigma^+ \to \{1, \ldots, m, \bot\}$ for P as follows:

$$C(w) = \begin{cases} O(w) & \text{if } O(w) \text{ defined} \\ \bot & \text{otherwise} \end{cases}$$

The theorem below states the correctness of the reduction.

Theorem 3. *O realizes B_0 over $B_1, \ldots, B_m$ iff C realizes T_0.*

Proof. Recall that C realizes T_0 iff $\tau^o_{w,T_0} = \tau^o_{w,C}$ for all $w \in \Sigma^*$. Let us denote by $out(w,T)$ the last element of $\tau^o_{w,T}$, then equivalently we may say that C realizes T_0 iff $out(w,T_0) = out(w,C)$ for all $w \in \Sigma^*$.

Only-if: Toward contradiction, suppose there exists a word w such that if $\hat{\delta}_0(s_{00}, w) \neq \bot$ then also $\hat{\delta}_O(s_{01}, \ldots, s_{0m}, w) \neq \bot$, but $out(w,T_0) \neq out(w,C)$. Observe that if $out(w,T_0) = \bot$ then $\hat{\delta}_0(s_{00}, w) = \bot$ hence $C(w) = \bot$ and so $out(w,C) = \bot$. So it must be the case that $out(w,T_0) = \top$ and $out(w,C) = \bot$. But by construction $out(w,T_0) = \top$ implies $\hat{\delta}_0(s_{00}, w) \neq \bot$ and $out(w,C) = \bot$ implies that $\hat{\delta}_O(s_{01}, \ldots, s_{0m}, w) = \bot$.

If: Toward contradiction, let word $w \cdot a$ be the smallest word such that $out(w \cdot a, T_0) = out(w \cdot a, C)$, but $\hat{\delta}_0(s_{00}, w \cdot a) \neq \bot$ and $\hat{\delta}_O(s_{01}, \ldots, s_{0m}, w \cdot a) = \bot$. Observe that if $out(w \cdot a, T_0) = \bot$ then $\hat{\delta}_0(s_{00}, w \cdot a) = \bot$, hence it must be the case that $out(w \cdot a, T_0) = out(w \cdot a, C) = \top$. As $w \cdot a$ is the smallest word that makes $\hat{\delta}_C(s_{01}, \ldots, s_{0m}, w \cdot a) = \bot$, it follows that $\hat{\delta}_0(s_{00}, w) = (s_1, \ldots, s_m) \neq \bot$. Hence either *1.* $O(w \cdot a)$ is undefined, or *2.* $O(w \cdot a) = h$ and $\delta_h(s_h, a)$ is undefined. In case *1.* $C(w \cdot a) = \bot$ and hence $out(w \cdot a, C) = \bot$. In case *2.* $C(w \cdot a) = h$, and, because $\delta_h(s_h, a)$ is undefined, $out(w \cdot a, C) = \bot$. In both cases we get a contradiction. $\square$

As composition in the Roman model is EXPTIME-hard (Muscholl and Walukiewicz 2008), from Theorem 3 we immediately get the following.

Theorem 4. *Checking whether a controller C for P realizes T_0 is EXPTIME-hard.*

3 Multi-Transducer Setting

In this section, we introduce a formalism that can represent the ability of manufacturing resources to take things apart (e.g., cut raw material into parts), and to assemble parts together into a single thing. In order to do this, we need to model resources that can take multiple inputs and produce multiple outputs. We model both the manufacturing resources and the recipe as *multi-port* transducers, or multi-transducers for short. A multi-transducer $T = (\Sigma, S, s_0, f, g, k, l)$ is a deterministic transition system with inputs and outputs, where k is the number of T's input ports and l is the number of T's output ports. Σ is the alphabet (of both inputs and outputs), S is the set of states, s_0 the initial state, $f : S \times \Sigma^k \longrightarrow S$ is the transition relation (takes a state and k input symbols and returns the successor state), and $g : S \times \Sigma^k \longrightarrow \Sigma^l$ is the output function.

Ports can be physical or virtual, that is, accept/output physical objects such as parts, or signals such as messages specifying that a particular operation should be performed. We assume that a physical output port can be bound only to one (physical) input port, while a virtual output port can be bound to multiple (virtual) input ports. An input port, however, should not be bound to more than one output port. More generally, we can support arbitrary forms of binding constraints disallowing undesirable binding combinations (these can be specified propositionally, see Section 3.1).

Example 3 (Multi-transducer).

We modify Example 1 in two ways. First, we assume that a square and a round part are processed in parallel, and then glued together to form a simple toy. We also change the input and output alphabet to consist of symbols corresponding to physical parts and symbols corresponding to operations. The recipe is as follows: take a square and a round part, clean them both, paint the square part green and round part yellow, and glue them together. On completion of the recipe, a final transition simply requests the resulting product. Formally, the target transducer is $T = (\Sigma, S, s_0, f, g, 4, 3)$ where $\Sigma = \{sq, rd, sqrd, clean, green, yel, glue, glued, err, sqclean, rdclean, sqgreen, rdyellow\}$ and $S = \{s_0, s_1, s_2, s_3, s_e\}$. A quadruple of inputs intuitively corresponds to at most two physical inputs (parts sq and rd) and two commands ($clean$ for clean, $green$ for paint green, yel for paint yellow, and $glue$ for glue). When fewer than four inputs are required, the remaining places are unfilled (or contain the empty symbol ϵ). Two virtual output ports output symbols describing the evolution of the process: $sqclean$ and $rdclean$ indicate that the parts have been cleaned; $sqgreen$ and $rdyellow$ that they have been painted, and finally $glued$ that they have been glued together. A third physical output port outputs the symbol $sqrd$ for the final product at the end of the recipe. The transducer T encoding the recipe is shown in Figure 2. $f(s_0, (sq, rd, clean, clean)) = s_1$, $g(s_1, (sq, rd, clean, clean)) = (sqclean, rdclean, \epsilon)$; $f(s_1, (\epsilon, \epsilon, green, yel)) = s_2$, $g(s_2, (\epsilon, \epsilon, green, yel)) = (sqgreen, rdyellow, \epsilon)$; $f(s_2, (\epsilon, \epsilon, glue, \epsilon)) = s_3$, $g(s_3, (\epsilon, \epsilon, glue, \epsilon)) = (glued, \epsilon, \epsilon)$, and finally $f(s_3, (\epsilon, \epsilon, \epsilon, \epsilon)) = s_0$, $g(s_3, (\epsilon, \epsilon, \epsilon, \epsilon)) = (\epsilon, \epsilon, sqrd)$. Any other input results in a transition to s_e with output (err, err, err).

3.1 Orchestration

We are given a set of multi-transducers $T_1, \ldots, T_m$, where each $T_j = (\Sigma, S_j, s_{0j}, f_j, g_j, k_j, l_j)$ (i.e., all T_j are over the same alphabet) representing manufacturing resources. We are also given a recipe T that is a multi-transducer with the same alphabet as the T_js.

The behavior of T on input $w = \mathbf{a^0 a^1} \ldots$, where $\mathbf{a^i} \in \Sigma^k$, is described by the following sequence of states and outputs:

$$T^s(\mathbf{a^0}) = f(s_0, \mathbf{a^0})$$
$$T^o(\mathbf{a^0}) = g(s_0, \mathbf{a^0})$$
$$\ldots$$
$$T^s(\mathbf{a^0} \ldots \mathbf{a^i}) = f(T^s(\mathbf{a^0} \ldots \mathbf{a^{i-1}}), \mathbf{a^i})$$
$$T^o(\mathbf{a^0} \ldots \mathbf{a^i}) = g(T^s(\mathbf{a^0} \ldots \mathbf{a^{i-1}}), \mathbf{a^i})$$

The observable output sequence of T on input w is $\tau^o(w, T) = T^o(\mathbf{a^0}), \ldots, T^o(\mathbf{a^0} \ldots \mathbf{a^i}) \ldots$

We denote by $in_{x,y}$ the input port y of multi-transducer T_x, and by $out_{x,y}$ the output port y of T_x. For convenience we extend this notation by using index $x = 0$ to denote the inputs and outputs of the environment. Note that these have a reversed role: the output of the environment is the input of the target/set of transducers, and the input to the environment is the output of the target/set of transducers. We denote by $val(in_{x,y})/val(out_{x,y})$ the value at the input/output port y of transducer T_x. We also denote by $val(in_x)/val(out_x)$ the vector of values at the input/output ports of T_x.

A *port binding*, or simply binding, is a pair of the form $(out_{x',y'},\ in_{x,y})$ which represents a connection between the output port y' of multi-transducer x' and input port y of multi-transducer x. A set c of port bindings, henceforth called *binding set*, must be consistent with a set of *binding constraints* $\mathcal{B}$, specified as boolean combinations of atoms of the form $(out_{x',y'}, in_{x,y})$; a binding set c is said to be *legal* iff $c \models \mathcal{B}$. We use the set $\mathcal{B}$ to impose three kinds of requirements:

i for all x, y (with $x \in \{0, 1, \ldots, m\}$ and $y \in \{1, \ldots, k_x\}$) there exists at most one x', y' such that $(out_{x',y'}, in_{x,y}) \in c$ (if, for some $z \in \{1, \ldots, k_x\}$, $in_{x,z}$ does not appear in c, its value is assumed to be empty, i.e., $val(in_{x,z}) = \epsilon$);

ii all physical output ports $out_{x',y'}$ occur in at most one binding $(out_{x',y'}, in_{x,y}) \in c$; and

iii arbitrary requirements specifying, e.g., the possible physical connections between machines on the shop floor, or the set of virtual connections determined by the possible communication routes between resources.

We denote the set of all possible port binding sets by $Cntl$.

Consider $P = T_1 \times \cdots \times T_m$ (i.e., the transducer corresponding to the whole production facility). A controller C for P is a strategy $C : (\Sigma^k)^+ \longrightarrow Cntl$. The sequence of (global) states and outputs generated by the controller on $w = \mathbf{a^0} \ldots \mathbf{a^i} \ldots$ is, respectively,

$$\begin{aligned} \tau_{w,C} &= (s_1^0, \ldots, s_m^0), \ldots, (s_1^i, \ldots, s_m^i) \ldots \\ \tau^o_{w,C} &= \mathbf{b^0}, \ldots, \mathbf{b^i}, \ldots \end{aligned}$$

where

- $C(\mathbf{a^0} \ldots \mathbf{a^i}) = c^i$ and c^i is legal;
- $val^i(in_{x,y}) = val^i(out_{x',y'})$ for $(out_{x',y'}, in_{x,y}) \in c^i$ (recall that $val^i(in_{x,y}) = \epsilon$ whenever $in_{x,y}$ does not appear in c^i);
- $s_x^{i+1} = f_x(s_x^i, val^i(in_x))$ for $x \in \{1, \ldots, m\}$;
- $val^i(out_x) = g_x(s_x^i, val_i(in_x))$ for $x = \{1, \ldots, m\}$;
- $val^i(out_0) = \mathbf{a}^i$ (note the inversion of out/in for 0);
- $val^i(out_{x,y}) = \mathbf{b}^i_{y'}$ if $(out_{x,y}, in_{0,y'}) \in c^i$.

C realizes T if $\tau^o(w, T) = \tau^o(w, C)$ for all w. The orchestration problem for multi-transducers is the same as for uni-transducers: is there a C for P such that C realizes T.

Note that the orchestration for the multi-transducer case works differently from the uni-transducer case. In the uni-transducer case, the controller selects only one transducer to make a move. In the multi-transducer case, the controller binds ports, and all transducers $T_1, \ldots, T_m$ get input (possibly empty) and move at every step.

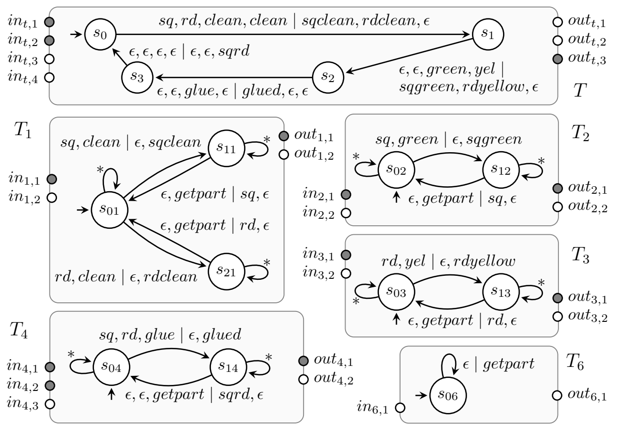
$in_{t,1}$
$in_{t,2}$
$in_{t,3}$
$in_{t,4}$
s_0
$sq, rd, clean, clean \mid sqclean, rdclean, \epsilon$
s_1
$out_{t,1}$
$out_{t,2}$
$out_{t,3}$
$\epsilon, \epsilon, \epsilon, \epsilon \mid \epsilon, \epsilon, sqrd$
s_3
s_2
$\epsilon, \epsilon, green, yel \mid$
$sqgreen, rdyellow, \epsilon$
$\epsilon, \epsilon, glue, \epsilon \mid glued, \epsilon, \epsilon$
T
T_1
$sq, clean \mid \epsilon, sqclean$
s_{11}
$out_{1,1}$
$out_{1,2}$
$in_{1,1}$
$in_{1,2}$
s_{01}
$\epsilon, getpart \mid sq, \epsilon$
$\epsilon, getpart \mid rd, \epsilon$
$rd, clean \mid \epsilon, rdclean$
s_{21}
T_2
$sq, green \mid \epsilon, sqgreen$
s_{02}
s_{12}
$\epsilon, getpart \mid sq, \epsilon$
$in_{2,1}$
$in_{2,2}$
$out_{2,1}$
$out_{2,2}$
T_3
$in_{3,1}$
$in_{3,2}$
$rd, yel \mid \epsilon, rdyellow$
s_{03}
s_{13}
$\epsilon, getpart \mid rd, \epsilon$
$out_{3,1}$
$out_{3,2}$
T_4
$sq, rd, glue \mid \epsilon, glued$
$in_{4,1}$
$in_{4,2}$
$in_{4,3}$
s_{04}
s_{14}
$\epsilon, \epsilon, getpart \mid sqrd, \epsilon$
$out_{4,1}$
$out_{4,2}$
T_6
$\epsilon \mid getpart$
s_{06}
$in_{6,1}$
$out_{6,1}$
*

Figure 2: Target transducer T and resource transducers T_1, $\ldots, T_6$ (T_5 is identical to T_1 and it is not shown). Physical ports are shown greyed, and error states s_{ej}, for each j, are not shown. $(*)$ on self-loops stands for $\epsilon, \cdots, \epsilon \mid \epsilon, \epsilon$.

Example 4 (Orchestration).

We show how to realize the recipe from Example 3 using the manufacturing resources in Figure 2. Transducers T_1 and T_5 are two identical cleaning machines, T_2 paints square parts green, T_3 paints round parts yellow, T_4 glues square parts and round parts together, and T_6 models the part-handling (transport) system. All transducers have the same alphabet Σ as in Example 3. Each transducer T_j has initial state s_{0j} and an error state s_{ej}.

In T_1 the physical port $in_{1,1}$ is used to receive symbols denoting the part to clean, while the virtual port $in_{1,2}$ is used to receive a symbol indicating the operation to execute (in this case, either *clean* to clean the part, or *getpart* to output the part on $out_{1,1}$). The virtual port $out_{1,2}$ outputs the symbol *sqclean* after cleaning a part. The functions f and g are as in Figure 2. In particular, note that $f_1(s_{01}, (\epsilon, \epsilon)) = s_{01}$ and $g_1(s_{01}, (\epsilon, \epsilon)) = (\epsilon, \epsilon)$, and the same for s_{11} and s_{12}, i.e., when there is no input binding, T_1 remains idle, looping on the current state. Any other combination of states and inputs other than those above results in a transition to s_{e1} and a loop outputting (err, err). T_2 to T_5 are analogous to to T_1. T_6 models a part-handling system that controls when parts can be transferred through physical ports. In this example, part transfer is always enabled: the transducer outputs a symbol $getpart$ at each step, which (by appropriate port binding) the controller can use to cause, e.g., T_1 to transfer a cleaned square part to T_2. In more complex examples, restrictions on the possible bindings can be imposed by the binding constraints $\mathcal{B}$, such as $\neg(out_{5,1}, in_{3,1})$.

Let us consider the orchestration for the input sequence $(sq, rd, clean, clean)$, $(\epsilon, \epsilon, green, yel)$, $(\epsilon, \epsilon, glue, \epsilon)$, $(\epsilon, \epsilon, \epsilon, \epsilon)$. On input $w = (sq, rd, clean, clean)$, the controller can select the bindings: $(out_{0,1}, in_{1,1})$ and $(out_{0,2}, in_{1,2})$ to send the part sq and the command $clean$ to the input ports of T_1; $(out_{0,3}, in_{5,1})$ and $(out_{0,4}, in_{5,2})$ to send rd and $clean$ to T_5; $(out_{1,2}, in_{0,1})$ and $(out_{5,2}, in_{0,2})$ to send the symbols $sqclean$ and $rdclean$, emitted by T_1 and T_5, to the environment's virtual input ports. T_6 is not required for this step as sq and rd are loaded into the production cell from outside. As a result, $\tau^o_{w,T} = \tau^o_{w,C}$. At the second step on input $w = (sq, rd, clean, clean)$, $(\epsilon, \epsilon, green, yel)$, C can select the bindings: $(out_{0,3}, in_{2,2})$ and $(out_{0,4}, in_{3,2})$ to send the commands $green$ and yel to the virtual input ports of T_2 and T_3; $(out_{6,1}, in_{1,2})$ and $(out_{6,1}, in_{5,2})$ to output parts from T_1 and T_5; and $(out_{1,1}, in_{2,1})$ and $(out_{5,1}, in_{3,1})$ to physically transfer the parts to the physical input ports of T_2 and T_3. The bindings $(out_{2,2}, in_{0,1})$ and $(out_{3,2}, in_{0,2})$ send $sqgreen$ and $rdyellow$ to the environment to match $\tau^o_{w,T}$. For $w = (sq, rd, clean, clean)$, $(\epsilon, \epsilon, green, yel)$, $(\epsilon, \epsilon, glue, \epsilon)$, C can select the bindings: $(out_{0,3}, in_{4,3})$ and $(out_{6,1}, in_{2,2})$; $(out_{6,1}, in_{3,2})$ and $(out_{2,1}, in_{4,1})$; and $(out_{3,1}, in_{4,2})$ and $(out_{4,2}, in_{0,1})$. This corresponds to sending the $glue$ command to the third input port of T_4, transfering the two parts from T_1 and T_5 to the physical input ports of T_4, and sending the symbol $glued$ to the environment. For the final transition, the bindings $(out_{6,1}, in_{4,3})$ and $(out_{4,1}, in_{0,3})$ are sufficient.

3.2 Orchestrator Synthesis

Surprisingly, synthesizing a controller for multi-transducers that solves the orchestration problem is essentially no more difficult than in the case of uni-transducers, in spite of the fact that we need to control multiple inputs and outputs in the available transducers and the port bindings, which change dynamically over time.

Again we adopt automata-theoretic techniques and resort to solving a safety game. Given the target transducer T and the available transducers $\{T_1, \ldots, T_m\}$, we build the safety game $G_{multi} = (\Sigma^k, Cntl, Q, q_0, \delta)$ where

- Σ^k is the input alphabet of the target T;
- $Cntl$ is the set of possible control actions (i.e., possible port binding sets);
- $Q = S \times S_1 \times \cdots \times S_m \cup \{q_{err}\}$ is the cartesian product of the states of the target and the available transducers, plus a special error state q_{err}; the state $(s, s_1, \ldots, s_m)$ stores the state of each of the transducers including the target;
- $q_0 = (s_0, s_{01}, \ldots, s_{0m})$;
- δ is defined as follows: $\delta((s, s_1, \ldots, s_m), \mathbf{a}, c) = (s', s'_1, \ldots, s'_m,)$ if the following conditions hold
  - $s' = f(s, \mathbf{a})$
  - c is legal;
  - $val(in_{x,y}) = val(out_{x',y'})$ for $(out_{x',y'}, in_{x,y}) \in c$;
  - $s'_x = f_x(s_x, val(in_x))$ for $x \in \{1, \ldots, m\}$;
  - $val(out_x) = g_x(s_x, val(in_x))$, for $x = \{1, \ldots, m\}$;
  - $val(out_0) = \mathbf{a}$ (again note the inversion of out/in for 0);
  - $val(out_{x,y}) = g(s, val(out_0))_{y'}$ if $(out_{x,y}, in_{0,y'}) \in c$, that is, the value $\mathbf{b}_{y'}$ of the output port y' of the target is the same of the value of the output port y of the available transducer x, when c binds y to y'.

  $\delta((s, s_1, \ldots, s_m), \mathbf{a}, c) = q_{err}$, otherwise.

Again, the only state that does not have transitions is q_{err}, which has no transitions at all. All other states have transitions for all $\mathbf{a} \in \Sigma^k$. It is easy to see that every strategy that solves the safety game G_{multi} corresponds to a controller C such that C realizes T and vice versa. Hence we have:

Theorem 5. *Checking whether there exists a C such that C realizes T can be done by solving the safety game G_{multi} defined above.*

As in the case of uni-transducers, Theorem 5 gives us an implementable algorithm for synthesizing the controller.

Next we turn to complexity. The EXPTIME-hardness of the simpler case of uni-transducers also applies in the more general case of multi-transducers. For membership, we observe that the number of states of G_{multi} is polynomial in the states of T, exponential in the number of available targets m and exponential in the number of the input ports of T. Hence, considering that solving a safety game is linear in the number of its states, we get:

Theorem 6. *Checking whether C realizes T is EXPTIME-complete, and in particular polynomial in the states of T, exponential in the number of available transducers m and exponential in the number of input ports of T and m.*

4 Discussion and Future Work

We have shown that the problem of whether a given manufacturing facility can realize a process recipe is decidable and EXPTIME-complete when recipes and manufacturing resources are represented as transducers. We provide an algorithm for synthesizing a controller that realizes a recipe. Our approach avoids the technical difficulties of LTL reactive synthesis (Pnueli and Rosner 1989; Lustig and Vardi 2009), such as determinization, and allows for effective implementation.

Our aim in this work is to automate manual process planning; as such we have not addressed the quantitative aspects of production. However, as our approach computes all correct orchestrations at once, the controller we synthesize can serve as a basis for quantitative reasoning (as is currently done in manual process planning, where optimization of the plan is separate step). We consider deterministic models, as is common in traditional process planning—manufacturing engineers typically assume that manufacturing resources reliably implement manufacturing operations. Nevertheless, our approach can be extended to cover stochastic settings along the lines of (Yadav and Sardiña 2011; Brafman et al. 2017). Another extension would be to introduce temporal constraints on port bindings, and perhaps a preference order on binding constraints (e.g., hard and soft constraints). We can also consider incremental repair of the controller when the set of available resources or the set of constraints change, along the lines of (De Giacomo, Patrizi, and Sardiña 2013).

So far, we addressed the *bounded* orchestration problem: i.e., the number of available manufacturing resources is fixed. A more ambitious extension is considering *unbounded* orchestration: is there *some number* of manufacturing resources of a certain type that can realize a given recipe. We conjecture that for uni-transducers, this problem would still be decidable, while for multi-transducers, it is undecidable.

Acknowledgments. Work was partially supported by the Sapienza project "Immersive Cognitive Environments", NSF grants CCF-1319459 and IIS-1527668, and by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering".

References

Barati, M., and St.-Denis, R. 2015. Behavior composition meets supervisory control. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 115–120.

Berardi, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Mecella, M. 2003. Automatic composition of e-services that export their behavior. In *1st International Conference on Service-Oriented Computing*, 43–58.

Bi, Z. M.; Lang, S. Y.; Shen, W.; and Wang, L. 2008. Reconfigurable manufacturing systems: the state of the art. *International Journal of Production Research* 46(4):967–992.

Brafman, R. I.; De Giacomo, G.; Mecella, M.; and Sardiña, S. 2017. Service composition under probabilistic requirements. In *Proceedings of ICAPS 2017 Workshop on Generalized Planning*.

Browne, J.; Dubois, D.; Rathmill, K.; Sethi, S. P.; and Stecke, K. E. 1984. Classification of flexible manufacturing systems. *The FMS magazine* 2(2):114–117.

Cassez, F.; David, A.; Fleury, E.; Larsen, K. G.; and Lime, D. 2005. Efficient on-the-fly algorithms for the analysis of timed games. In *16th International Conference on Concurrency Theory*, 66–80.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. In *24th International Joint Conference on Artificial Intelligence*, 1558–1564.

De Giacomo, G.; Felli, P.; Patrizi, F.; and Sardiña, S. 2010. Two-player game structures for generalized planning and agent composition. In *24th AAAI Conference on Artificial Intelligence*, 297–302.

De Giacomo, G.; Patrizi, F.; and Sardiña, S. 2013. Automatic behavior composition synthesis. *Artificial Intelligence* 196:106–142.

de Silva, L.; Felli, P.; Chaplin, J. C.; Logan, B.; Sanderson, D.; and Ratchev, S. 2016. Realisability of production recipes. In *22nd European Conference on Artificial Intelligence*, 1449–1457.

Ehlers, R.; Lafortune, S.; Tripakis, S.; and Vardi, M. Y. 2017. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems* 27(2):209–260.

ElMaraghy, H. A. 2005. Flexible and reconfigurable manufacturing systems paradigms. *International Journal of Flexible Manufacturing Systems* 17(4):261–276.

Felli, P.; de Silva, L.; Logan, B.; and Ratchev, S. 2017. Process plan controllers for non-deterministic manufacturing systems. In *26th International Joint Conference on Artificial Intelligence*, 1023–1030.

Felli, P.; Logan, B.; and Sardina, S. 2016. Parallel behavior composition for manufacturing. In *25th International Joint Conference on Artificial Intelligence*, 271–278.

Felli, P.; Yadav, N.; and Sardiña, S. 2017. Supervisory control for behavior composition. *IEEE Transactions on Automatic Control* 62(2):986–991.

Grädel, E.; Thomas, W.; and Wilke, T. 2003. *Automata, Logics, and Infinite Games: A Guide to Current Research*, Volume 2500 of Lecture Notes in Computer Science. Springer

Groover, M. P. 2007. *Automation, production systems, and computer-integrated manufacturing*. Prentice Hall

Hopcroft, J., and Ullman, J. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley

Kagermann, H.; Helbig, J.; Hellinger, A.; and Wahlster, W. 2013. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry; Final Report of the Industrie 4.0 Working Group*. Forschungsunion.

Koren, Y.; Heisel, U.; Jovane, F.; Moriwaki, T.; Pritschow, G.; Ulsoy, G.; and Van Brussel, H. 1999. Reconfigurable manufacturing systems. *CIRP Annals-Manufacturing Technology* 48(2):527–540.

Lu, Y.; Xu, X.; and Xu, J. 2014. Development of a hybrid manufacturing cloud. *Journal of Manufacturing Systems* 33(4):551–566.

Lustig, Y., and Vardi, M. Y. 2009. Synthesis from component libraries. In de Alfaro, L., ed., *Proceedings of FOSSACS*, 395–409.

Mehrabi, M. G.; Ulsoy, A. G.; and Koren, Y. 2000. Reconfigurable manufacturing systems: key to future manufacturing. *Journal of Intelligent Manufacturing* 11(4):403–419.

Muscholl, A., and Walukiewicz, I. 2008. A lower bound on web services composition. *Logical Methods in Computer Science* 4(2).

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *POPL*.

Rhodes, C. 2015. *Manufacturing: Statistics and Policy. Briefing Paper*. House of Commons Library.

Sethi, A. K., and Sethi, S. P. 1990. Flexibility in manufacturing: a survey. *International Journal of Flexible Manufacturing Systems* 2(4):289–328.

Smale, D., and Ratchev, S. 2009. A capability model and taxonomy for multiple assembly system reconfigurations. In *13th IFAC Symposium on Information Control Problems in Manufacturing*, volume 13, 1923–1928.

2012. A landscape for the future of high value manufacturing in the UK. Technical report, Technology Strategy Board.

Vardi, M., and Wolper, P. 1986. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and Systems Sciences* 32(2):182–221.

Yadav, N., and Sardiña, S. 2011. Decision theoretic behavior composition. In *10th International Conference on Autonomous Agents and Multiagent Systems*, 575–582.