Unbounded Orchestrations of Transducers for Manufacturing

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

Tomáš Brázdil
Mazaryk University
Brno, Czech Republic
xbrazdil@fi.muni.cz

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

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

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

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

Abstract

There has recently been increasing interest in using reactive synthesis techniques to automate the production of manufacturing process plans. Previous work has assumed that the set of manufacturing resources is known and fixed in advance. In this paper, we consider the more general problem of whether a controller can be synthesized *given sufficient resources*. In the unbounded setting, only the *types* of available manufacturing resources are given, and we want to know whether it is possible to manufacture a product using only resources of those type(s), and, if so, how many resources of each type are needed. We model manufacturing processes and facilities as transducers (automata with output), and show that the unbounded orchestration problem is decidable and the (Pareto) optimal set of resources necessary to manufacture a product is computable for uni-transducers. However, for multi-transducers, the problem is undecidable.

1 Introduction

There has recently been increasing interest in using reactive synthesis techniques to automate the synthesis of manufacturing process plans (de Silva et al. 2016; Felli, Logan, and Sardina 2016; de Silva et al. 2017; Felli et al. 2017; De Giacomo et al. 2018). A *process plan* matches the abstract manufacturing tasks in a *process recipe* specifying the steps needed to manufacture a product, 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. However, with the increasing servitization of manufacturing (sometimes called 'cloud manufacturing' (Lu, Xu, and Xu 2014)), where the products to be manufactured are not known in advance and are often produced in small batches to tight timescales, the manual production of process plans is becoming increasingly uneconomic.

Previous work on the automated synthesis of process plans has focussed on the 'on-the-fly' generation of process plan controllers. For example, in (de Silva et al. 2016; Felli et al. 2017), 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 via conveyor belts, AGVs etc., 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. In (De Giacomo et al. 2018) process recipes and resources are modelled as (multi)transducers (automata with output), and the problem of whether a given process recipe can be realized by a given set of manufacturing resources is then stated as an orchestration problem for transducers. They provide an algorithm for synthesizing a controller that realizes a recipe based on safety games.

In all of these approaches, the set of manufacturing resources in a manufacturing facility (or available in a manufacturing cloud) is assumed to be known and fixed in advance. In this paper, we consider the more general problem of whether a product can be manufactured *given sufficient resources*. In this setting, only the *types* of available manufacturing resources are given, and we want to know whether it is possible to manufacture a product using only resources of those type(s), and, if so, how many resources of each type are needed. This question is of interest when considering the provisioning of manufacturing facilities; for example, how many additional resources of each type would have to be purchased or leased in order to manufacture a new product? We model manufacturing processes and facilities as transducers (automata with output), and show that the unbounded orchestration problem is decidable and the (Pareto) optimal set of resources necessary to manufacture a product is computable for uni-transducers. However, for multi-transducers, the problem is undecidable.

2 Uni-Transducer Setting

As in (De Giacomo et al. 2018), we begin by modeling both manufacturing resources and process recipes as uni-transducers, that is, finite deterministic automata with outputs (Mealy machines) (Hopcroft and Ullman 1979). We consider a simple setting where all machines take a single input and produce a single output in each state.

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).*

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. 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 in a manufacturing context. The error state has a self-loop that on any input symbol, outputs an error symbol.

Example 1 (Uni-transducers). As an example, consider a simple process recipe T shown in Figure 1 (top) for joining two panels forming part of a product. Each panel is first positioned, p, the panels are then joined using fasteners, j, and finally the panels are released, r. Formally, the corresponding transducer is $T = (\Sigma, \Delta, S, s_0, f, g)$, where: $S = \{s_0, s_1, s_2, s_3, s_4, s_{err}\}$; $\Sigma = \{p, j, r\}$, where p corresponds to positioning a panel, j corresponds to joining two panels, and r corresponds to releasing a panel; $\Delta = \{p', j', r', err\}$, which correspond to signals that the manufacturing operations p, j, r were performed, and an error, respectively. The transitions are depicted in Figure 1; for example $f(s_0, p) = s_1$, and $g(s_0, p) = p'$. All combinations of states and inputs not depicted in Figure 1 result in a transition to s_{err} outputting err.

Such a recipe can be realized (manufactured) by an assembly cell containing two types of resource as shown in Figure 1 (bottom). The first type of resource, T_1, can position a panel, e.g., T_1 may be a robot arm that holds the panel in the appropriate position for joining. The second type of resource, T_2 can insert fasteners to join the panels. Clearly, to join two panels, we need two resources of type T_1 and one resource of type T_2.

2.1 Unbounded Orchestration

In this section, we formalize the unbounded orchestration problem. We assume we are given a set of *available transducer types* $\{T_1, \ldots, T_m\}$, 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

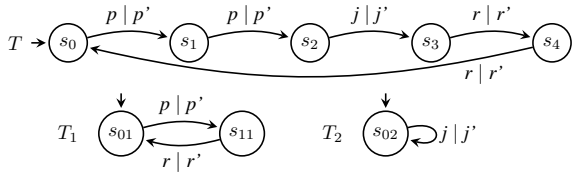

Figure 1: Process recipe T and manufacturing resources T_1 and T_2. The error states s_{err} for each transducer are not shown

a transducer with the same input and output alphabets as the T_js. The unbounded orchestration problem is to combine some number n_j of resources of each type T_j to be able to match the behavior of T.

We first give a formal definition of 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 $P = T_{1,1} \times T_{1,n_1} \times \cdots \times T_{m,1} \times T_{m,n_m}$ (i.e., the transducer corresponding to the whole production facility, with n_j copies of T_j for each transducer type T_j).

A controller for P is a function $C : \Sigma^+ \longrightarrow \{(1,1), \ldots, (m, n_m)\}$ 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

$$\tau^s_{w,C} = (s^0_{1,1}, \ldots, s^0_{m,n_j}), \ldots, (s^i_{1,1}, \ldots, s^i_{m,n_j}) \ldots$$

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

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

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 transducer types $T_1, \ldots, T_m$ and a production recipe transducer T, the orchestration problem is the question whether there are numbers of copies of each transducer type $n_1, \ldots, n_m$ such that there is a controller C for $P = T_{1,1} \times \cdots \times T_{m,n_m}$ which realizes T.*

Example 2 (Orchestration). Consider a sequence of inputs $p, p, j, r, r, \ldots$ The transducer T from Example 1 produces a sequence of outputs $p', p', j', r', r', \ldots$ on this input. A controller C for $T_{1,1} \times T_{1,2} \times T_{2,1}$ (where the number n_1 of copies of T_1 is 2, and $n_2 = 1$ is the number of copies of T_2)

imitates this behavior by the following mapping:

$$\begin{array}{ll} p \mapsto & (1,1) \\ p,p \mapsto & (1,2) \\ p,p,j \mapsto & (2,1) \\ p,p,j,r \mapsto & (1,1) \\ p,p,j,r,r \mapsto & (1,2) \end{array}$$

2.2 Orchestrator Synthesis

For the bounded case (when there is exactly one transducer of each type), the orchestration problem was solved in (De Giacomo et al. 2018) by using the idea that a controller can be synthesized as a strategy to solve *safety games* (Grädel, Thomas, and Wilke 2003; De Giacomo et al. 2010; Ehlers et al. 2017). They also showed that the bounded orchestration problem is EXPTIME-complete.

In this section, we will reduce the unbounded orchestration problem to a decidable problem on *multi-dimensional energy games*.

Let $\mathcal{C}$ be a finite set of n counters. A *multi-dimensional energy game* is a tuple $G = (Q, R)$ where Q is a finite set of states and R is a finite set of transitions of the form (q_1, op, q_2) where $q_1, q_2 \in Q$ and $op \in \{0, -1, 1\}^n$ is an update vector for the values of the counters (0 in position j stands for no change to the value of counter c_j, -1 stands for decrementing c_j, and 1 for incrementing c_j by 1). Q is partitioned into Q_0, which are states of Player 0, and Q_1, which are states of Player 1. A *counter evaluation* $\nu : C \to \mathbb{Z}$ is a function from the set of counters to the set of integers. A *configuration* γ is a pair (q, ν) where $q \in Q$ and ν is a counter evaluation. Transitions between configurations are defined as follows: given $\gamma_1 = (q_1, \nu_1)$, $\gamma_2 = (q_2, \nu_2)$ and $t = (q_1, op, q_2) \in R$, $\gamma_1 \longrightarrow^t \gamma_2$ iff $\nu_2 = \nu_1 + op$. A history is a finite sequence of configurations. A strategy of Player $i \in \{0, 1\}$ is a function σ_i that assigns to each history ending in a configuration γ with the state in Q_i a transition t that is possible from γ. Given strategies σ_0 and σ_1 of Player 0 and Player 1, respectively, together with an initial configuration γ_1 induce a *play*, an infinite sequence $\gamma_1, \gamma_2, \ldots$ of configurations such that $\gamma_k \longrightarrow^t \gamma_{k+1}$ where $\gamma_k = (q_k, \nu_k)$ for $q_k \in Q_i$ and $t = \sigma_i(\gamma_1, \ldots, \gamma_k)$. Player 0 wins such a play if $\nu_k(c) \geq 0$ for all $k \geq 1$ and $c \in \mathcal{C}$, i.e. all counters of all configurations of the play are non-negative. Player 0 wins the game in γ if it wins every play starting in γ. The set $W(G, 0)$ is the set of configurations γ in which player 0 wins. The Pareto frontier is $min(W(G, 0))$ where min is with respect to pointwise order of values. Given a counter evaluation ν, the *maximum norm* of ν is defined by $\max_{c \in C} \|\nu(c)\|$.

Theorem 1. *(Jurdzinski, Lazic, and Schmitz 2015) The Pareto frontier is computable in doubly exponential time and pseudo-polynomial assuming the number of counters is fixed. Moreover, the maximum norms of vectors of the Pareto frontier are bounded by an exponential value in the number of counters n and polynomial in the number of states and transitions.*

We will now reformulate the orchestration problem as a multi-dimensional energy game. This together with Theorem 1 implies that the orchestration problem is decidable in doubly exponential time.

We are given m resource transducer types $T_i = (\Sigma, \Delta, P_i, p_0^i, \alpha_i, \beta_i)$ and a target process recipe transducer $T = (\Sigma, \Delta, S, s_0, f, g)$. The realizability question for $\Pi T_i^{n_i} = T_{1,1} \times T_{1,n_1} \times \cdots \times T_{m,1} \times \cdots \times T_{m,n_m}$ can be formulated as a game between Controller (Player 0) and an Adversary (Player 1). The game starts with Adversary choosing an input symbol $a \in \Sigma$, yielding output $g(p, a)$, and changing the state of T to $f(s, a)$. Controller must choose a component $T_{i,j}$ where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n_i\}$ such that $\beta_i(p_j^i, a) = g(p, a)$, and change the state p_j^i of $T_{i,j}$ to $\alpha_i(p_j^i, a)$. Controller must be able to keep playing forever without causing a mismatch of outputs between T and $\Pi T_i^{n_i}$. If Controller has a winning strategy, then $\Pi T_i^{n_i}$ realizes T.

Let us denote a transition of T from state $s \in S$ to state $t = f(s, a)$, for $a \in \Sigma$ with output $b = g(s, a)$, by the tuple (s, a, b, t). Intuitively these are transitions of the Adversary (the way Player 1 can choose the next state).

Let the cardinality of the set of states of T_i, $|P_i| = k_i$. Then $\Pi T_i^{n_i}$ can be described by a Σk_i-ary nonnegative integer vector $\mathbf{v} = (v_0^1, \ldots, v_{k_1-1}^1, \ldots, v_0^m, \ldots, v_{k_m-1}^m)$, where $v_0^i + \ldots + v_{k_i-1}^i = n_i$. Intuitively, v_q^i denotes that there are v_q^i copies of T_i in state p_q^i. The initial state of $\Pi T_i^{n_i}$ is, therefore, the vector $(k_i, \mathbf{0})_i$ which says that for every T_i we have k_i copies of T_i in state p_0^i, and zero copies in every other state in P_i. Now applying an input symbol a to a particular configuration of $\Pi T_i^{n_i}$ means choosing a copy of T_i in some state p_q^i and relacing it by a copy in state $p_r^i = \alpha_i(p_q^i, a)$. The output corresponding to this transition is $b = \beta_i(p_q^i, a)$. We denote this transition by the tuple (p_q^i, a, b, p_r^i). Thus, the vector $\mathbf{v}$ representing a counter evaluation of $\Pi T_i^{n_i}$ is updated by adding to it the *update vector* $\mathbf{u} = (u_0^1, \ldots, u_{k_m-1}^m)$, where $u_q^i = -1$, $u_r^i = 1$, and all other components of $\mathbf{u}$ are 0. Note that adding an update vector to $\mathbf{v}$ does not change the sum of the components of $\mathbf{v}$. Let $U_{(a,b)}$ be the set of such update vectors for transitions with input $a \in \Sigma$ and output $b \in \Delta$.

As defined above, a controller $C : \Sigma^+ \longrightarrow \{(1,1) \ldots, (m, n_m)\}$ chooses a type T_i and a copy $T_{(i,j)}$ of it in $\Pi T_i^{n_i}$ and applies the current input symbol to that copy. But all copies that are in the same state behave in the same way, so we can say that the controller is a function $C : \Sigma^+ \longrightarrow \{(1,0), \ldots, (m, k_{m-1})\}$, which chooses i and a state in P_i. Consider now a transition (s, a, b, t) of T. The Controller has to simulate this by a transition (p_q^i, a, b, p_r^i) in $\Pi T_i^{n_i}$, where the vector $\mathbf{v}$ describing the current configuration of $\Pi T_i^{n_i}$ is updated by an update vector in $U_{(a,b)}$.

We define therefore the following multi-dimensional energy game $G_{T_1,\ldots,T_m,T}$:

- The state set of $G_{T_1,\ldots,T_m,T}$ is $(S \times \Sigma) \cup S$ where S is the state set of T and Σ the input alphabet of T. The partition is $Q_0 = S \times \Sigma$ and $Q_1 = S$.
- For each transition (s, a, b, t) of T, we have in $G_{T_1,\ldots,T_m,T}$ the transitions $((s, a), \mathbf{u}, t)$ for all $\mathbf{u} \in U_{(a,b)}$

(Player 0's moves) and a transition $(s, \mathbf{0}, (s, a))$ (Player 1's move). (Essentially we add an input symbol to the source state of the transition, instead of having it as a label of this transition.) Note that the only transitions of Player 0 possible from (s, a) are the ones which result in the same output symbol produced by one of the resource transducers as the output of T.

Intuitively, the initial configuration is $(s_0, \mathbf{v_0})$ where $\mathbf{v_0}$ corresponds to all copies of resource transducers being in their initial states. At each round of the game in configuration $(s, \mathbf{v})$, Adversary chooses a transition (s, a, b, t) for $t = f(s, a)$ and $b = g(s, a)$, with resulting configuration $((s, a), \mathbf{v})$. Controller responds by choosing an update vector $\mathbf{u} \in U_{(a,b)}$ resulting in $(t, \mathbf{v} + \mathbf{u})$. Recall that Controller wins the multi-dimensional energy game $G_{T_1,\ldots,T_m,T}$ if it is able to always keep all counters non-negative.

Theorem 2. *Let $T_1, \ldots, T_m$ be resource transducer types and T a process recipe transducer. $\Pi T_i^{n_i}$ realizes T iff Controller wins the game $G_{T_1,\ldots,T_m,T}$ from configuration $(s_0, (n_1, \mathbf{0}, \ldots, n_m, \mathbf{0}))$ where the counter corresponding to each initial state of T_i has value n_i and the rest of the counters are 0.*

Proof. Suppose there is a Controller for realizing T on $\Pi T_i^{n_i}$. Then a winning strategy for Player 0 on $G_{T_1,\ldots,T_m,T}$ is given as follows. Suppose the Controller's move on input Σ^+ is (i, j) (give current input to the jth copy of T_i) and the state of $\Pi T_i^{n_i}$ on input Σ^+ is $\mathbf{v}$. Then the update vector for the next move by Player 0 is to decrementing the counter for the current state of p_q^i of $T_{i,j}$ and incrementing the counter for the state of $T_{i,j}$ resulting from the input corresponding to the last symbol in Σ^+. By the assumption of realizability, the counter of p^i must be positive and hence the decrement does not make it negative. Hence Player 0 can always make a safe move and has a winning strategy in $G_{T_1,\ldots,T_m,T}$ from configuration $(s_0, (n_1, \mathbf{0}, \ldots, n_m, \mathbf{0}))$.

For the other direction, suppose Player 0 has a winning strategy in $G_{T_1,\ldots,T_m,T}$ from configuration $(s_0, (n_1, \mathbf{0}, \ldots, n_m, \mathbf{0}))$. For every sequence of game configurations (which include input symbols), there is a move by Player 0 which corresponds to incrementing some p_r^i counter and decrementing some p_q^i counter. So the Controller needs to pick some arbitrary copy j of transducer $T_{i,j}$ and give it the input. By assumption, there is at least one copy of T_i in state p_q^i, so this choice is always possible, and the transition $U_{a,b}$ will produce the output matching that of T on the same input by the construction of $U_{a,b}$. □

Given $T_1, \ldots, T_m$ and T, we can construct a game $G_{T_1,\ldots,T_m,T}$ and check, using Theorem 1, whether Player 0 has a winning strategy starting in a configuration with the counter evaluation corresponding to all T_i being in the initial state, that is, $(n_1, \mathbf{0}, \ldots, n_m, \mathbf{0})$ for some $n_1, \ldots, n_m$. If it does, then T is realizable using $T_1, \ldots, T_m$.

Corollary 1. *Let $T_1, \ldots, T_m$ be resource transducers and T a process recipe transducer. The unbounded orchestration problem for $T_1, \ldots, T_m$ and T is decidable in doubly exponential time.*

3 Multi-Transducer Setting

In this section, we consider the unbounded orchestration problem for multi-transducers. Multi-transducers were proposed in (De Giacomo et al. 2018) as a more natural formalism for modeling manufacturing resources that take multiple inputs and generate a single output (e.g., assemble parts together into a single compound part), or that take a single input and generate multiple outputs (e.g., cut a sheet of raw material into two or more parts).

In the multi-transducer setting, both the manufacturing resources and the process recipe are modeled 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 k input ports and l 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 that 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 that returns the outputs associated with a transition. 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. 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. In addition, binding constraints can be used to specify physical connections between resources on the shop floor, or the set of virtual connections determined by the possible communication routes between resources. Critically, the port bindings specifying connections between resources can be changed while the facility is manufacturing a product: parts (or signals) output by a resource can be directed to the input ports of different resources at different points during the realization of the recipe.

3.1 Orchestration

We now state the orchestration problem for mult-transducers. We are given a set of multi-transducer types $T_1, \ldots, T_m$, 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 $\tau^o(w, T) = T^o(\mathbf{a^0}), \ldots, T^o(\mathbf{a^0} \ldots \mathbf{a^i}) \ldots$

Consider P (i.e., the multi-transducer corresponding to the whole production facility) which is a composition of n_1 copies of $T_1, \ldots, n_m$ copies of T_m. In addition to picking a transducer $x \in \{(1, 1), \ldots, (m, n_m)\}$, the controller is also in charge of changing the port binding, as defined below.

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 $x \in \{0, (1,1), \ldots, (m, n_m)\}$ 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 $P = T_{1,1} \times \cdots \times T_{m,n_m}$ (i.e., the transducer corresponding to the whole production facility, with n_j copies of each type T_j). 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,

$$\begin{aligned} \tau_{w,C} &= (s^0_{1,1}, \ldots, s^0_{m,n_m}), \ldots, (s^i_{1,1}, \ldots, s^i_{m,n_m}) \ldots \\ \tau^o_{w,C} &= \mathbf{b^0}, \ldots, \mathbf{b^i}, \ldots \end{aligned}$$

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);
- $s^{i+1}_x = f_x(s^i_x, val^i(in_x))$ for $x \in \{(1,1) \ldots, (m, n_m)\}$;
- $val^i(out_x) = g_x(s^i_x, val_i(in_x))$ for $x = \{(1,1), \ldots, (m, n_m)\}$;
- $val^i(out_0) = \mathbf{a}^i$ (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: given a target T and resource types $T_1, \ldots, T_m$, are there numbers of copies $n_1, \ldots, n_m$ and a C for $P = T_{1,1} \times \cdots \times T_{m,n_m}$ such that C realizes T.

Note that orchestration in 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 $T_{1,1}, \ldots, T_{m,n_m}$ get input (possibly empty) and move at every step.

3.2 Orchestrator Synthesis

In the bounded case (when there is exactly one resource transducer of each type), the problem of synthesizing a controller that solves the orchestration problem is 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. This has been shown in (De Giacomo et al. 2018). However, as we show below, for the unbounded case the problem is undecidable. The main reason for the undecidability is the combination of multiple input and output ports and no bound on the number of transducers, since both the unbounded orchestration problem for uni-transducers, and the bounded problem for multi-transducers are decidable. A similar undecidable setting is data-flow composition (Lustig and Vardi 2009), that also assumes the ability to connect together unbounded number of components and control data flow from one component to another. Having constraints on port bindings is not essential for the undecidability.

Theorem 3. *The unbounded orchestration problem for multi-transducers is undecidable.*

Proof. We prove that the problem is undecidable by exibiting an example where such a controller exists if and only if a Turing machine M halts on empty input.

In our construction, the target transducer is T which on input a outputs a. Formally, $T = (\Sigma, S, s_0, f, g, 1, 1)$, where $\Sigma = \{a\}$, $S = \{s_0\}$, $f(s_0, a) = s_0$ and $g(s_0, a) = a$.

The resource transducer types intuitively correspond to cells of the tape of a Turing machine M. M is a single tape TM with instructions of the form $qc \longrightarrow c'dq'$, where $d \in \{-1, 1\}$, which means, if in state q reading symbol c, write symbol c', move to the left (if $d = -1$) or to the right ($d = 1$) and go into state q'. The leftmost cell contains a special symbol α, the tape starts blank, and the machine writes 0s and 1s in the cells it visits. The machine starts in initial state q_0 and halts if it goes into the accepting state q_n; otherwise it continues forever. Without loss of generality, we assume that M performs the following operations between executing any two 'real' instructions: it adds a cell to the right (writes a symbol ω in the cell to the right of the rightmost non-blank cell), then goes all the way to the left until it reaches the leftmost cell, and then comes back to where it was after the last 'real' instruction, and then executes the next 'real' instruction. The first unnecessary procedure (adding a cell to the right) ensures that if M does not halt, then it uses infinitely many cells. The second procedure is a technical device we need in the proof below to impose

a particular binding of input and output ports in any composition of resource transducers. Any TM can be converted to a TM which executes the two unnecessary procedures, by adding a fixed number of extra states and symbols. (Let us assume that in addition to 0, 1, α and ω, the machine can write symbols $h_1, \ldots, h_t$.) Finally, we also require that M only reaches the state q_n (if it does) if it is in the the rightmost cell which has been used, and on the last pass from left to right over the tape, it erases all the symbols on the tape apart from α.

We have three types of resource transducer: T_l, T_m and T_r, corresponding to the leftmost cell, middle cell(s), and rightmost cell of M, respectively. The alphabet of resource transducers is $\Sigma' = \{q_0, \ldots, q_n, a, err\}$, where $q_0, \ldots, q_n$ correspond to states of M.

The basic idea of encoding M's tape cells as transducers is that transducer's internal states correspond to which symbol is on the tape in the cell; inputs and outputs encode the head arriving into the cell and the state of M.

Left cell transducer T_l T_l has two input ports 1 and 2; intuitively, 1 is an input port for receiving input a from the environment, and 2 is an input port for receiving internal inputs from the cell on the right. T_l has one output port for communicating with the cell on the right. T_l starts in state s_{init} (intuitively, corresponding to containing α and being in state q_0) and on input a on input port 1 and blank on 2 (on input (a, ϵ)), it mimics the instruction of M: writes α again, and outputs symbol q_i which is the state M goes into upon empty input. T_l then goes into state s_α. If T_l receives input q_j from the right ((ϵ, q_j)), in s_α, it again does what the instruction of M requires, and outputs the new state q'. On all other inputs in all other states T_l outputs err and goes in state s_{err}. The states of S_l are $\{s_{init}, s_\alpha, s_{err}\}$.

Middle cell transducer T_m T_m has two input ports and two output ports. Intuitively, input 1 and output 1 connect to the cell on the left, and input 2 and output 2 connect to the cell on the right. Input (q, ϵ) means receiving q from the left, and input (ϵ, q) means receiving q from the right. Similarly for outputs: the output (q', ϵ) corresponds to moving to the cell to the left in state q', and (ϵ, q') means moving to the cell to the right in state q'. The states of T_m are s_ϵ (being blank, initial state), s_0 (containing 0), s_1 (containing 1), s_ω, $s_{h_1}, \ldots, s_{h_t}, s_{err}$. On input q when in state s_c the transducer will go into state $s_{c'}$ and output q' on the left output port (if $d = -1$) or on the right (if $d = 1$) output port. On all other inputs, for example input from the right when in state s_ϵ, it goes into s_{err} and outputs err.

Right cell transducer T_r T_r has one input port (from the cell to the left) and two output ports; one of them (the second one) can be used to output a. This only happens when M would have gone into state q_n (so instead of outputting q_n, T_r outputs a). Otherwise T_r mimics M's transitions: when it is in state s_c and receives q on the left, it goes into state $s_{c'}$ and outputs q' to the left. In all other cases, in particular when getting as input a state symbol which corresponds to moving one cell to the right to write ω in it, T_r goes into s_{err} and outputs err. The states of T_r are $S_r = \{s_\epsilon, s_\omega, s_0, s_1, s_{h_1}, \ldots, s_{h_t}, s_{err}\}$.

Let us call a composition of several T_l, T_m and T_r-type transducers *correctly wired* if it corresponds to a tape of M: there is a T_l on the left with its input 1 connected to the environment (or to a copy of T_r, see below); T_r on the right with its output 2 connected to the environment (or to a copy of T_l, see below); and between them are 0 or more T_ms, so that for each T_m x there is exactly one other transducer y such that x's input 1 and output 1 are connected to input 2 and output 2 of y (which can only be another T_m or T_l), and similarly there is exactly one z such that x's input 2 and output 2 are connected to input 1 and output 1 of z (which can only be another T_m or T_r). Note that we do not rule out a wiring corresponding to a sequential composition of several tapes of M: that is, when a T_r is wired to another T_l etc., so long as there is a T_l on the left and T_r on the right connected to the environment.

We are going to prove the following two statements:

A A controller for a correctly wired composition of length k realizing T exists if and only if M halts after using k cells.

B For any other composition (not correctly wired) there is no controller that realizes T.

From A and B, it follows that a controller for a composition of some number of copies of T_l, T_m and T_r exists if, and only if, M halts.

Let us prove A first. Suppose we have a correctly wired composition of length k. It has a single free input port (belonging to a T_l), so all a controller can do is to pass a to that port. On input a, T_l starts the imitation of a Turing machine M, passing symbols corresponding to the states of M back and forth along the composition, with cell transducers changing states to represent a new symbol on the tape of M in the corresponding cell. If M halts in the kth cell of the tape, eventually T_r gets input q_n, upon which it outputs a, and T is realised. If M does not halt, then eventually T_r will get an input for moving to the right to write ω in it, and T_r outputs err. Note that since before halting, M erases the tape, the transducers are returned to the initial state before the output of a, and are ready for the next input of a.

For B, first of all observe that the only type of transducer that can take input a is T_l, and the only one that can output a is T_r. So any composition which does not connect T_l's input 1 to the environment's output or T_r's output 2 to the input of the environment is not going to work. Next we need to show that the wiring in the middle is correct. First of all, the regular trip to the beginning of the tape that M performs makes sure that if the wiring is incorrect in connecting input 2 to output 1 ('left') of some transducer, then output err will be produced, because some cell would receive input 'from the right' (on its input port 2) while it is still blank (in state s_ϵ). Similarly, the lack of a consistent path 'to the right' (every cell's output 2 connected to the right neighbour's input 1) means that when adding an ω in the rightmost cell to the right, we either discover a blank cell before the next ω cell, or discover a blank cell while going to the left to the beginning of the tape. □

4 Related work

There is a substantial literature on flexible manufacturing from an engineering perspective. For example, *Flexible Manufacturing Systems* (Browne et al. 1984; Sethi and Sethi 1990; ElMaraghy 2005) increase the range of products that may be assembled, and *Reconfigurable Manufacturing Systems* (Bi et al. 2008; Koren et al. 1999; Mehrabi, Ulsoy, and Koren 2000; Smale and Ratchev 2009) reduce response time.

More recently, a range of AI approaches have been proposed to the automated synthesis of manufacturing process plans. For example, (Ciortea, Mayer, and Michahelles 2018) present an approach to flexible agent-based manufacturing systems, in which autonomous agents synthesize production plans using semantic descriptions of Web-based artifacts and coordinate with one another via multi-agent organizations. The motivation of their work, namely the repurposing manufacturing lines 'on-the-fly', has some similarities with that presented here. However, their approach is based on AI planning, and considers a fixed rather than unbounded set of resources.

As noted in the introduction, there has also been a strand of work on applying techniques based on AI behavior composition (Berardi et al. 2003; De Giacomo, Patrizi, and Sardiña 2013) to determine whether and how a particular product can be manufactured by a particular set of manufacturing resources. For example, (Felli, Logan, and Sardina 2016) have applied synthesis techniques in a traditional mass production setting. They introduce a novel solution concept, *target production processes*, that are able to manufacture multiple instances of a product simultaneously in a given manufacturing facility, and give a technique for synthesizing the largest target production process, together with an associated controller. The work on the automated synthesis of process plans discussed in the introduction (de Silva et al. 2016; Felli et al. 2017) is perhaps closer to our work, in focussing on the 'on-the-fly' generation of process plan controllers in a mass customization setting. 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 capable of synthesizing a single instance of the product at a time. Controller synthesis 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 approaches proposed in (de Silva et al. 2016; Felli et al. 2017) involve considerable bookkeeping and are 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; Pnueli and Rosner 1989; Lustig and Vardi 2009; 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 transfer operations.

A more general approach was proposed in (De Giacomo et al. 2018). That work generalizes the movement of parts and data in the system and considers both physical and logical connections between machines; it also abstracts away the execution of additional low-level actions by focusing only on the observable behavior of resources. Their approach is based on the standard model of input/output transducers and captures the essence of process recipes and manufacturing resources, thus relating the synthesis of process-plan controllers to classical reactive synthesis. The problem of whether a given manufacturing facility can realize a process recipe is shown to be decidable and EXPTIME-complete. We use the same framework as in (De Giacomo et al. 2018) but study the unbounded version of controller synthesis problem: when only the resource types but not the required number of each resource type are known.

The synthesis of transducers was also studied in (Exibard, Filiot, and Jecker 2018), but in the context of synthesizing a single deterministic transducer for a non-deterministic specification. In (Nourine, Hassen, and Toumani 2016) the authors establish decidability of service (transducer) composition when unbounded copies of services (transducers) are allowed. Their technique is quite complex and gives only an Ackermannian upper bound (more than non elementary), in the general case. The undecidability result for the multi-transducers is analogous to the undecidability for data-flow composition (Lustig and Vardi 2009).

The problem of synthesizing process-plan controllers 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).

5 Conclusions and Future Work

In this paper, we address the problem of unbounded orchestration, that is related to the problem of provisioning manufacturing facilities: given the process plans to be implemented, decide how many of each type of manufacturing resource is needed. We prove that if process plans and resources are represented as uni-transducers, the problem of whether the precess plan is realizable with given resource types is decidable in 2EXPTIME, and the Pareto optimal values for the numbers of each type of resource can be computed. In the case of multi-transducers, we show that the problem is undecidable.

In future work we plan to investigate decidable cases of orchestration for multitransducers. Other open questions include various notions of optimality for the resulting production plans (in addition to minimizing the number of resources of each type as we do here): e.g.,, spreading the load on various resources. In the longer term, we plan to explore implementations of our approach in a practical tool.

Acknowledgments. Work supported in part by the Sapienza project "Immersive Cognitive Environments", NSF grants CCF-1319459 and IIS-1527668, by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering", and by the Unibz DACoMan projec. T.B. was supported by the Czech Science Foundation, grant No. 18-11193S. We thank Sylvain Schmitz for discussion on complexity of energy games and Dominik Velan for carefully reviewing material relating to energy games.

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 *Proceedings of ICSOC*, 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.

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.

Ciortea, A.; Mayer, S.; and Michahelles, F. 2018. Repurposing manufacturing lines on the fly with multi-agent systems for the web of things. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*, 813–822.

De Giacomo, G.; Felli, P.; Patrizi, F.; and Sardiña, S. 2010. Two-player game structures for generalized planning and agent composition. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, 297–302.

De Giacomo, G.; Vardi, M.; Felli, P.; Alechina, N.; and Logan, B. 2018. Synthesis of orchestrations of transducers for manufacturing. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 6161–6168.

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 *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI-2016)*, 1449–1457.

de Silva, L.; Felli, P.; Chaplin, J. C.; Logan, B.; Sanderson, D.; and Ratchev, S. 2017. Synthesising industry-standard manufacturing process controllers (Demonstration). In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, 1811–1813.

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.

Exibard, L.; Filiot, E.; and Jecker, I. 2018. The complexity of transducer synthesis from multi-sequential specifications. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, volume 117 of *LIPIcs*, 46:1–46:16.

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

Felli, P.; Logan, B.; and Sardina, S. 2016. Parallel behavior composition for manufacturing. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 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 *LNCS*. Springer.

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

Jurdzinski, M.; Lazic, R.; and Schmitz, S. 2015. Fixed-dimensional energy games are in pseudo-polynomial time. In *Proceedings of ICALP 2015*, 260–272.

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.

Nourine, L.; Hassen, R. R.; and Toumani, F. 2016. Decidability and complexity of web service business protocol synthesis. *International Journal of Cooperative Information Systems* 25(3):1–43.

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, 179–190.

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 *Proceedings of IFAC Symposium on Information Control Problems in Manufacturing*, volume 13, 1923–1928.