# Process Plan Controllers for Non-Deterministic Manufacturing Systems

**Paolo Felli[1], Lavindra de Silva[1], Brian Logan[2], Svetan Ratchev[1]**
[1]Institute for Advanced Manufacturing, University of Nottingham, UK
[2]School of Computer Science, University of Nottingham, UK
{firstname.lastname}@nottingham.ac.uk

## Abstract

Determining the most appropriate means of producing a given product, i.e., which manufacturing and assembly tasks need to be performed in which order and how, is termed *process planning*. In process planning, abstract manufacturing tasks in a *process recipe* are matched to available manufacturing resources, e.g., CNC machines and robots, to give an executable *process plan*. A *process plan controller* then delegates each operation in the plan to specific manufacturing resources. In this paper we present an approach to the automated computation of process plans and process plan controllers. We extend previous work to support both non-deterministic (i.e., partially controllable) resources, and to allow operations to be performed in parallel on the same part. We show how implicit fairness assumptions can be captured in this setting, and how this impacts the definition of process plans.

## 1 Introduction

Product manufacturing is increasingly moving towards flexible, adaptive, intelligent, and networked manufacturing systems, in which manufacturing activities are distributed, and enterprises collaborate through the so-called manufacturing-as-a-service paradigm [Lu *et al.*, 2014]. In the manufacturing-as-a-service paradigm, each product may be different from the one before (batch size of one production) [TSB, 2012; Rhodes, 2015]. Traditional approaches to production control are unable to meet the demands of manufacturing-as-a-service or batch-size-of-one production. Manufacturing process planning is traditionally carried out by engineers who are experts in the particular processes used in a specific factory, and, with the exception of some limited support by Computer-Aided Process Planning (CAPP) tools, is largely a manual process. From the point of view of manufacturing-as-a-service where the product to be manufactured is not known in advance, the traditional approach has several drawbacks: it requires expensive human expertise to determine whether the customer's product can be manufactured by a given service provider; and even if the product is manufacturable, the small batch sizes (perhaps a single item) mean that manually producing a process plan is uneconomic. To realise the manufacturing-as-a-service vision, process planning must be fully automated, allowing service providers to 'bid' to manufacture products in real time.

To date, there has been relatively little work on the manufacture and assembly of highly-customised products in a highly-networked manufacturing environment. An exception is [de Silva *et al.*, 2016], where techniques are proposed to: (I) determine *whether* a particular product can be manufactured by a particular set of manufacturing resources (the *realisability problem*), and (II) *how* a particular customised product should be manufactured using available resources (the *control problem*). Their approach takes as input a *process recipe* specifying the tasks necessary to manufacture the product, and transforms the process recipe into an executable *process plan* specifying the low-level tasks to be executed by each manufacturing resource in the production line. The resulting process plan is used to orchestrate the activities of agents in the Evolvable Assembly Systems (EAS) architecture, an agent-based architecture for manufacturing control software designed to address rapidly changing product and process requirements including batch-size-of-one customised production [Chaplin *et al.*, 2015]. In EAS, each *resource agent* represents and controls a manufacturing resource, e.g., a machine tool or a robot.

The approach in [de Silva *et al.*, 2016] relies on several strong assumptions. In particular, it assumes that manufacturing resources are *deterministic*; that is, the execution of a manufacturing or assembly task from a given state of the system can result in only one possible new state. However many manufacturing resources are *non-deterministic* in the sense that performing a manufacturing task may result in one of a number of possible states. For example, a moulding operation may result in excess material that must be removed from the moulded part. A second major restriction is that the formalism in [de Silva *et al.*, 2016] prohibits performing manufacturing or assembly tasks in parallel on the same part or set of parts. As a result, they are unable to model, e.g., a flexible assembly cell in which one robot positions or holds a part while another robot performs an operation on the part.

In this paper we extend the approach of [de Silva *et al.*, 2016] to allow non-deterministic resources and tasks to be performed in parallel on the same part, and we define novel notions of process plans and process plan controllers for this setting. Crucially, as we consider non-deterministic re-

sources, their execution may result in cycles that are not under the control of the process plan; therefore, we investigate how implicit *fairness* assumptions can be captured in this setting, and how this impacts the definition of process plans.

## 2 Process Recipes

A process recipe specifies the sequence of tasks necessary to manufacture a product, including its constituent parts and associated parameters required to process and assemble these parts into the final product, any tests that must occur during the manufacturing process, and how to respond to test results.

As in [de Silva *et al.*, 2016], we formalise process recipes as labelled transition systems, where labels are complex "task expressions". Let $\mathcal{L} = \langle T, C \rangle$ be a library of tasks where $T$ is a finite set of *tasks* which represent operations and $C$ is a finite set of part constants. We assume that the set $T$ is partitioned into three mutually disjoint sets: the set of *observable* tasks $T_{ob}$ which correspond to manufacturing operations; the set of *internal* tasks $T_{in}$, which represent internal actions of the system; and the set of *synchronisation* tasks $T_{syn}$, which specify the transfer of parts between resources. Specifically, $T_{syn} = \{in^h | h \in \mathbb{N}\} \cup \{out^h | h \in \mathbb{N}\}$ is the set of *in* and *out* synchronisation tasks, by which a part is moved from a resource that performs task $out^h$ (releasing a part) to the resource that performs task $in^h$ (accepting a part). We also use $nop$ to denote idling.

The smallest task expressions are called *parameterised tasks* (or *p-tasks*). We extend the formalism of [de Silva *et al.*, 2016] to allow p-tasks of the form $t(\mathbf{x}, \mathbf{y}, \mathbf{z})$, where $t \in T_{ob} \cup T_{in}$, and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are sequences of part constants in $C$. The sequences $\mathbf{x}$ and $\mathbf{y}$ represent the "internal" and "external" inputs of $t$, respectively, and $\mathbf{z}$ represents the outputs. A resource executing task $t$ *consumes* the parts in $\mathbf{x}$ and *produces* the parts in $\mathbf{z}$, while the parts in $\mathbf{y}$ must be present in *another* resource which consumes them (as input) for its own task. For example, $drill(\epsilon, c, \epsilon)$ represents a drilling operation performed on a part $c$ that is present in another resource (e.g., a robot drilling a hole in a part held by another robot).

Formally, a *task expression* is a formula in the language $Lang(\mathcal{T})$ generated by the grammar:

$$\mathcal{T} \coloneqq t \mid \mathcal{T}; \mathcal{T} \mid \mathcal{T} \| \mathcal{T} \mid \mathcal{T} \text{ "|" } \mathcal{T}$$

where ";" denotes a sequence, "$\|$" denotes parallel composition, and "|" denotes interleaved composition. We call p-tasks and parallel compositions of p-tasks *atomic task expressions*. We denote by $Lang_{ob}(\mathcal{T})$ the subset of $Lang(\mathcal{T})$ where every p-task $t(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in T_{ob}$, i.e., only observable tasks are allowed. We impose the following additional constraints on $\mathcal{T}$. Any expression $\mathcal{T}_1 \| \ldots \| \mathcal{T}_m$ occurring in $\mathcal{T}$ is restricted such that each $\mathcal{T}_i$ is a p-task and:

- a part constant cannot appear in the inputs $\mathbf{x}$ (resp. outputs $\mathbf{z}$) of more than one task; i.e., before (resp. after) parallel tasks are performed on a part, it can only be present in *one* of their resources; and
- it does not hold that $\mathbf{x}_i, \mathbf{x}_j = \epsilon$ with $\mathbf{z}_i, \mathbf{z}_j \neq \epsilon$ for some $i, j \in [1, m]$, $i \neq j$, i.e., we disallow introducing more than one "fresh" part into the system (in parallel).

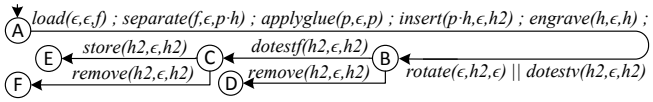


Figure 1: A process recipe.

We also restrict task expressions $\mathcal{T}_1 \mid \ldots \mid \mathcal{T}_m$ occurring in $\mathcal{T}$ such that each $\mathcal{T}_i$ does not mention the operator "|".

**Definition 1 (Process Recipe).** A *process recipe* is a tuple $\mathcal{R} = \langle s^0, S, L, \delta_{\mathcal{R}} \rangle$, where $S$ is a finite set of states, $s^0 \in S$ is the initial state, $L \subseteq Lang_{ob}(\mathcal{T})$ is a set of task expressions, and $\delta_{\mathcal{R}} \subseteq S \times L \times S$ is a non-empty transition relation. We denote a transition from state $s$ to $s'$, with task expression $\mathcal{T}$, either by $s \xrightarrow{\mathcal{T}} s'$ or $\langle s, \mathcal{T}, s' \rangle \in \delta_{\mathcal{R}}$. ■

As in [de Silva *et al.*, 2016], process recipes may contain (bounded) cycles, and we assume that cycles are removed by unfolding the recipe up to the bound in a pre-processing step. The unfolded recipe thus describes the (finite) process of manufacturing a given product; when the recipe reaches an end-state with no outgoing transitions the process is "completed". States in the recipe are essentially states in the manufacture of the product that are 'choice points', i.e., where a decision must be made at run-time what to do next based on, e.g., the specification of the current product instance (such as its colour) or testing the partially assembled product. Note that $\delta$ is a relation: in general, there may be more than one outgoing transition from each state, because a recipe may encode different alternatives to reach an end-state from the initial state $s^0$. We generalise [de Silva *et al.*, 2016] by allowing different outgoing transitions, labelled with the same task expression, from the same state of the process recipe.

Figure 1 shows an example of a process recipe (based on [de Silva *et al.*, 2016]) that specifies how to assemble a hinge. The first and second p-tasks load a new pallet fixture ($f$) and separate it into its constituents: the hinge pin ($p$) and hollow hinge ($h$). Then $p$ is glued onto the hinge $h$ to obtain a (non-hollow) hinge ($h2$), which is then engraved with a serial number. The next two (parallel) tasks involve a 360 degree visual test on $h2$, after which the recipe either requests a force test, or discards the hinge, depending on the runtime outcome of the visual test. Similarly, the hinge can be stored or discarded, depending on a further force-test (as these tests are performed at runtime, all alternatives must be accounted for).

## 3 Non-Deterministic Resources

We model manufacturing resources in the facility as labelled transition systems, and extend [de Silva *et al.*, 2016] by allowing non-deterministic transitions.

**Definition 2 (Resource).** A *resource* is a tuple $R = \langle \underline{s}^0, \underline{S}, T, \Longrightarrow \rangle$, where $\underline{S}$ is a set of states, $\underline{s}^0 \in \underline{S}$ is the initial state, $T$ is the set of tasks, and $\delta_R \subseteq \underline{S} \times T \times \underline{S}$ is a transition relation. ■

We write $\underline{s} \overset{t}{\Longrightarrow} \underline{s}'$ to denote a transition from $\underline{s}$ to $\underline{s}'$ by task $t$. For example, resource $R_1$ in Figure 2 can load new pallets. Loading is a non-deterministic transition as it may result in the pallet being misaligned; when this happens, an acoustic

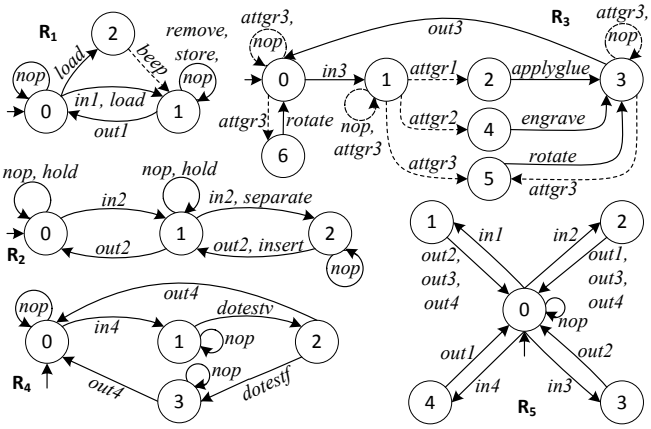


Figure 2: An assembly system with non-deterministic resources. The tasks in $T_{in}$ are $attgr1, attgr2, attgr3, beep$.

signal instructs the operator to realign the pallet. The pallet can then be either stored, removed, or moved to another resource (*out1*). Transfer of parts between resources is performed by resource $R_5$, which models a transportation system specifying the "legal routes" between production resources.

A *(production) topology* represents the synchronous execution of a set of resources, including where parts are processed and how parts are moved between resources.

**Definition 3 (Topology).** Let $\{R_1, \ldots, R_n\}$ be resources, with each $R_i = \langle \underline{s}_i^0, \underline{S}_i, T, \Longrightarrow_i \rangle$. A *topology* is a tuple $\mathbf{P} = \langle \underline{s}^0, \underline{S}, T^n, \Longrightarrow \rangle$, where $\underline{S} = \underline{S}_1 \times \cdots \times \underline{S}_n$ is the set of states; $\underline{s}^0 = \langle \underline{s}_1^0, \ldots, \underline{s}_n^0 \rangle$ is the initial state; $T^n$ is the set of concurrent tasks; and the transition relation $\Longrightarrow \subseteq \underline{S} \times T^n \times \underline{S}$ is such that $\underline{s} \xRightarrow{\mathbf{t}} \underline{s}'$ iff for all $i \in [1,n]$[1] either:

- $t_i \notin T_{syn}$ and $\underline{s}_i \xRightarrow{t_i}_i \underline{s}_i'$, i.e., resource $R_i$ can execute $t_i$;
- $t_i \in T_{syn}$ and $\underline{s}_i \xRightarrow{t_i}_i \underline{s}_i'$, and there exists exactly one $j \in [1,n]$ such that $\underline{s}_j \xRightarrow{t_j}_j \underline{s}_j'$ with $t_j = in^h$ and $t_i = out^h$ for some $h$, denoted $t_j \nleftarrow t_i$, or the opposite, denoted $t_j \nrightarrow t_i$. ■

The second condition checks that, within a transition, each *out* task is matched with an *in*. A transition $\underline{s} \xRightarrow{\mathbf{t}} \underline{s}'$ is said to be *observable* iff at least one task in the vector is observable.

Given a topology $\langle \underline{s}^0, \underline{S}, T^n, \Longrightarrow \rangle$, the assignment of parts to resources during production is represented by a *resource vector* $\mathbf{r} = \langle \mathbf{c}_1, \ldots, \mathbf{c}_n \rangle$, where each $\mathbf{c}_i \in C^*$ is a (possibly empty) sequence of parts that do not appear anywhere else in $\mathbf{r}$. We denote $\mathbf{c}_i$ by $\mathbf{r}(i)$ for $i \in [1,n]$; the set of all possible resource vectors as $V$; and the empty vector as $\mathbf{r^0} = \langle \epsilon, \ldots, \epsilon \rangle$. Hence, $\mathbf{r}(i)$ denotes the parts allocated to resource $R_i$ in the current state of the topology. We note that each element in $\mathbf{r}$ is a sequence and not a set, i.e., order matters when moving parts between resources, or when executing tasks on parts. We address a limitation in [de Silva *et al.*, 2016] by allowing the *simultaneous* execution of tasks on the *same* parts, in order to model "joint" tasks. A resource $R_i$ currently in state $\underline{s}_i$ can execute an (atomic) p-task $t(\mathbf{x}, \mathbf{y}, \mathbf{z})$ iff (I) the task $t$ is available from state $\underline{s}_i$ in $R_i$, and (II) the input parts are currently assigned to the resource, and, if external parts are required (i.e., $\mathbf{y} \neq \epsilon$), then there exist one or more other resources to which those parts are collectively assigned. After executing $t$, parts in $\mathbf{z}$ are allocated to the resource. For example, the task $insert\_into(c_1, c_2, \epsilon)$ inserts part $c_1$ into part $c_2$, where $c_2$ is currently assigned to another resource, and does not produce any parts. Given a p-task $t(\mathbf{x}, \mathbf{y}, \mathbf{z})$, we denote $\mathbf{x}$ by $in(t)$, $\mathbf{y}$ by $ext(t)$ and $\mathbf{z}$ by $out(t)$.

We now give a formal definition of simultaneous tasks. Let $\mathcal{T} = t_1 \| \ldots \| t_m$ be a task expression, $\mathbf{r}$ a resource vector, $\underline{s}$ a topology state, and $\underline{s} \xRightarrow{\mathbf{t}} \underline{s}'$ a transition with $\mathbf{t} = \langle t_1', \ldots, t_n' \rangle$. Let $I = \{i \in [1,n] \mid t_i' \in T_{ob}\}$ be the "observable indices" of $\mathbf{t}$. Then, we say that a resource vector $\mathbf{r}'$ is an *allocation* of $\mathcal{T}$ to $\mathbf{t}$ with respect to $\mathbf{r}$, denoted $\mathbf{r}' = \text{AL}(\mathbf{r}, \mathcal{T}, \mathbf{t})$, iff for all $i \in [1,n] \setminus I$ we have $\mathbf{r}'(i) = \mathbf{r}(i)$, and there exists a bijection $f : I \mapsto [1,m]$ s.t. for all $i \in I$, we have that $j = f(i)$ implies

- $t_j = t_i'$, namely the task label in position $i$ in the vector is equal to the task label of the $j$-th parallel p-task; and
- $\mathbf{r}(i) = in(t_j)$, $\mathbf{r}'(i) = out(t_j)$ and $ext(t_j) \neq \epsilon$ iff there exists a subset of indices $I' \subseteq [1,n] \setminus \{i\}$ such that $ext(t_j)$ is a concatenation of each $in(t_k)$, $k \in I'$.

Intuitively, an allocation of $\mathcal{T}$ to a topology transition labelled with vector $\mathbf{t}$ returns the new resource vector resulting from the simultaneous execution of each task in $\mathbf{t}$, provided that all the p-tasks in $\mathcal{T}$ are matched to an observable task in $\mathbf{t}$. For instance, a vector of tasks $\mathbf{t} = \langle hold, nop, engrave \rangle$ can execute a parallel task expression $\mathcal{T} = engrave(\epsilon, c, \epsilon) \| hold(c, \epsilon, c)$, so that $\langle c, \epsilon, \epsilon \rangle = \text{AL}(\langle c, \epsilon, \epsilon \rangle, \mathcal{T}, \mathbf{t})$. For sequences of the form $\mathcal{T}_1; \mathcal{T}_2$ we can compute allocations recursively, and for interleaved compositions of the form $\mathcal{T}_1 \mid \mathcal{T}_2$, we need to find a linearisation such that an allocation exists. Details are omitted for brevity, as they do not differ from [de Silva *et al.*, 2016] apart from the base case above.

For synchronisation tasks, a resource vector $\mathbf{r}'$ is a *transfer* of parts from $\mathbf{r}$ via $\mathbf{t}$, denoted $\mathbf{r}' = \text{MOV}(\mathbf{r}, \mathbf{t})$, if for each $i \in [1,n]$ we have (I) $t_i \nleftarrow t_j$ and $\mathbf{r}'(i) = \mathbf{r}(i) \cdot c$ with $\mathbf{r}(j) = c \cdot \mathbf{c}$; or (II) $t_i \nrightarrow t_j$ and $\mathbf{r}'(i) = \mathbf{c}$ with $\mathbf{r}(j) = c \cdot \mathbf{c}$; or (III) $\mathbf{r}'(i) = \mathbf{r}(i)$ otherwise. For example, if $\mathbf{r} = \langle c, \epsilon, \epsilon \rangle$ then $\text{MOV}(\mathbf{r}, \mathbf{t})$ with $\mathbf{t} = \langle out^1, nop, in^1 \rangle$ is $\langle \epsilon, \epsilon, c \rangle$.

## 4 Realisability of Recipes

Each transition in a process recipe is labelled with a task expression, but these tasks are not directly executable: actual realisations, in the form of "orchestrations" of the topology, must be found. We now introduce some technical definitions.

A *trace* of a topology $\mathbf{P}$ from a resource vector $\mathbf{r}_0$ is a sequence $\tau = \sigma_0 \xRightarrow{\mathbf{t}_1} \sigma_1 \xRightarrow{\mathbf{t}_2} \cdots$ such that for each $i \geq 0$ we have $\sigma_i = (\underline{s}_i, \mathbf{r}_i)$, where $\underline{s}_i$ is a state in the topology and $\mathbf{r}_i$ is a resource vector. A trace represents the evolution of the topology, *together with* the new resource vector computed at each step. We call a finite trace *history*, and given a history $\tau = \sigma_0 \xRightarrow{\mathbf{t}_1} \cdots \xRightarrow{\mathbf{t}_m} \sigma_m$ of length $m$, we denote $\sigma_m$ by $last(\tau)$. Unless otherwise specified, we assume $\underline{s}_0 = \underline{s}^0$ and $\mathbf{r}_0 = \mathbf{r}^0$, i.e., we take as initial the initial state of the topology and the empty resource vector.

[1] $\underline{s} = \langle \underline{s}_1, \ldots, \underline{s}_n \rangle, \underline{s}' = \langle \underline{s}_1', \ldots, \underline{s}_n' \rangle$ and $\mathbf{t} = \langle t_1, \ldots, t_n \rangle$.

We now introduce the notion of *plan*, which compactly represents the *set* of histories (there may be more than one, as the topology is non-deterministic) that realise a given task expression. Let $\mathcal{H}$ denote all possible histories of $\mathbf{P}$. Then, a *history-based plan* is a partial function

$$\pi : \mathcal{H} \mapsto T^n$$

which maps a history of $\mathbf{P}$ to a vector of tasks. A *trajectory* of a plan $\pi$ on $\mathbf{P}$ from $\sigma_0$ is a trace $\tau = \sigma_0 \xrightarrow{\mathbf{t}_1} \sigma_1 \xrightarrow{\mathbf{t}_2} \cdots$ of $\mathbf{P}$, of length $\ell \geq 0$, such that $\mathbf{t}_{i+1} = \pi(\tau|_i)$, where $\tau|_i$ denotes the fragment of $\tau$ of length $i \in [0, m-1]$. Given a history $\tau$, we call $\sigma_{|\tau|} = last(\tau)$ the *outcome* of $\tau$ from $\sigma_0$. A trajectory of $\pi$ is said to be *complete* with respect to $\pi$ iff it is finite and cannot be extended further, namely, iff $\pi(\tau)$ is undefined.

## 4.1 Strong Cyclic Plans

A history-based *terminating plan* is a plan such that all its trajectories are finite, i.e., it always terminates irrespective of the non-determinism of the topology. However, not all trajectories are finite. For example, in Figure 2, the resource $R_3$ encodes a robotic arm which can perform various operations by attaching different grippers. Since performing the *internal* task $attgr3$ in state 1 may lead to two possible successor states, there is no history-based terminating plan that is guaranteed to realise the recipe, as the observable task $rotate$ may never be reached. In reality, while performing a task may occasionally take a resource into an abnormal state, resources are engineered such that after a 'small' number of retries and/or some recovery steps, the intended state will be reached.[2] This 'fairness assumption' is however not captured explicitly in the resource or the topology.

Note that the appropriate notion of fairness for manufacturing resources does not correspond to *strong fairness* as in [De Giacomo *et al.*, 2010], where if the action is repeated infinitely often, then any outcome happens infinitely often. First, while internal tasks such as $attgr3$ can be non-deterministic, observable tasks such as $rotate$ are assumed to be deterministic: executing the task is assumed to lead to the intended state—otherwise the author of a process recipe would have to anticipate all possible failures of the (unknown) manufacturing resources used to manufacture the product. In this sense, fairness is only relevant to *internal* tasks, as only internal tasks forming part of the 'implementation' of an observable task may be retried: e.g., repeatedly executing $attgr3$ will eventually reach a state where $rotate$ can be executed. Second, repeating an observable task would violate the recipe, and in many cases this would be incorrect and/or unsafe (consider, for instance, operations such as casting or moulding). Third, parts/subassemblies often represent considerable investment of materials and process steps, so are usually only discarded as a last resort. As observable tasks may be non-deterministic in practice, a controller may have to implement a 'recovery' internal plan fragment to remedy undesired states (e.g., to remove excess material from a moulded part when required) prior to the next observable task in the recipe. Hence, we need to relax strong fairness, and consider a particular kind of *strong cyclic plan* [Cimatti *et al.*, 2003] whose associated trajectories can always terminate, and when they do, are guaranteed to achieve the goal.

[2] Note that there is no restriction in our formalism that the number of retries should be small.

## 4.2 Process Plans

We can now concretise our definition of process plan, which represents a *strong cyclic* history-based plan. First, we say that a history-based plan $\pi$ is *nonblocking* if any of its finite trajectories *can* be extended to a complete trajectory (namely, it is a prefix of a complete trajectory of $\pi$). Thus, it is always possible for these plans to terminate. However, they are not necessarily *terminating plans*, as they may have infinite trajectories. To be implementable in practice, we need to find a *finite* representation of such plans.

**Definition 4 (Process plan).** Given a task expression $\mathcal{T}$, a topology state $\underline{s}_0$ and a resource vector $\mathbf{r}_0$, we say that a history-based plan $\pi$ is a *process plan* for $\mathcal{T}$ from $(\underline{s}_0, \mathbf{r}_0)$ iff $\pi$ is nonblocking and any complete trajectory of $\pi$ from $(\underline{s}_0, \mathbf{r}_0)$ realises $\mathcal{T}$, as defined below.

Informally, a trajectory $\tau$ realises an atomic task expression $\mathcal{T}$ iff the task is allocated in (exactly) one step, which is necessarily an observable transition, while all other steps are arbitrary unobservable transitions (i.e., consisting only of internal tasks and synchronisation tasks). Formally, given a plan $\pi$, an initial state $\underline{s}_0$ and a resource vector $\mathbf{r}_0$, a trajectory $\tau = \sigma_0 \xrightarrow{\mathbf{t}_1} \cdots \xrightarrow{\mathbf{t}_m} \sigma_m$ of $\pi$ from $\sigma_0$, with $\sigma_i = (\underline{s}_i, \mathbf{r}_i)$ for each $i \in [1, m]$ *realises* an atomic task expression $\mathcal{T}$ iff:

- there exists an $\ell \in [1, m]$ such that $\mathbf{r}_\ell = \textsc{mov}(\mathbf{r}, \mathbf{t}_\ell)$ with $\mathbf{r} = \textsc{al}(\mathbf{r}_{\ell-1}, \mathcal{T}, \mathbf{t}_\ell)$;
- for any other $j \neq \ell$, $j \in [1, m]$, we have that $\mathbf{t}_j$ is unobservable and $\mathbf{r}_\ell = \textsc{mov}(\mathbf{r}_{\ell-1}, \mathbf{t}_\ell)$.

We extend this notion to sequences, and say that a trajectory $\tau$ realises a task expression of the form $\mathcal{T}_1; \mathcal{T}_2$ iff $\tau$ has the form $\tau = \tau_1 \cdot \tau_2$ and $\tau_1$ (resp. $\tau_2$) realises $\mathcal{T}_1$ (resp. $\mathcal{T}_2$), where $\tau_1 \cdot \tau_2$ denotes the concatenation of two finite trajectories sharing the last and first state, respectively (which is equal to $\tau_1$ when $\tau_2$ is the empty trajectory). The extension to interleaved compositions is analogous, by considering all possible linearisations. Without loss of generality, we assume that an observable transition is always the last one in the trajectory, as this strictly relates plans to the task expressions that they realise by disallowing subsequent arbitrary transitions (which, instead, should be executed to realise the *next* task expression in the recipe).

As process plans are not necessarily terminating, they may have infinite trajectories. We therefore introduce finite representations of such plans. First, given a history $\tau$, its *contractions*, denoted $acycl(\tau)$, is the set of histories obtained from $\tau$ by removing all cycles. The contractions correspond to the fixpoint of an operator $\lambda$ on $\{\tau\}$ defined as $\tau_1 \cdot \tau_2 \in \lambda(\{\tau, \cdots\})$ iff $\tau = \tau_1 \cdot \tau' \cdot \tau_2$, such that the first and last state of $\tau'$ are the same, i.e., $last(\tau_1) = last(\tau')$. It is trivial to see that a contraction always exists. A *basic plan* is a partial function

$$\pi_b : \mathcal{H}_a \mapsto T^n$$

that is defined only for acyclic histories $\mathcal{H}_a \subseteq \mathcal{H}$ of $\mathbf{P}$ (which are finite), and thus only generates finite and complete trajectories. We say that $\pi_b$ is a *basic plan for $\mathcal{T}$* iff every acyclic

trajectory of $\pi_b$ that is complete realises $\mathcal{T}$, and at least one such trajectory exists. These plans can be represented and implemented finitely; however, they are not process plans: they do not need to realise any task, and indeed in a non-deterministic topology they may not do so. Given a basic plan $\pi_b$, we can reconstruct a generic history-based plan $\pi_{\{\pi_b\}}$ as follows:

$$\pi_{\{\pi_b\}}(\tau) := \pi_b(\tau) \quad \text{iff it is defined;}$$
$$\pi_{\{\pi_b\}}(\tau) := \pi_{\{\pi_b\}}(\tau') \quad \text{otherwise,}$$

where $\tau'$ is a trajectory of $\pi_b$ and $\tau' \in acycl(\tau)$. Informally, $\pi_b$ is applied to $\tau$ by disregarding any cycle.

**Theorem 1.** If $\pi_b$ is a basic plan for $\mathcal{T}$ and $\pi_{\{\pi_b\}}$ is nonblocking, then $\pi_{\{\pi_b\}}$ is a process plan for $\mathcal{T}$.

*Proof Sketch.* It is easy to see that $\pi_b$ is defined for every contraction of one of its trajectories that still does not realise $\mathcal{T}$. This holds trivially for acyclic trajectories, since they must realise the task expression by definition. For cyclic trajectories, assume that for a contraction $\tau'$, $\pi_b(\tau')$ is undefined. Then $\tau'$ is not a trajectory of $\pi_b$ (which is instead required) otherwise, being complete, it would contradict the fact that every complete acyclic trajectory realises $\mathcal{T}$. Hence any cyclic trajectory of $\pi_{\{\pi_b\}}$ can always be extended to one that is finite and complete. Thus $\pi_{\{\pi_b\}}$ is nonblocking, which together with the hypothesis proves the claim. □

Basic plans are memory bounded, and as soon as they induce a cyclic trajectory, they immediately "reset" to a contraction. However, not every process plan can be reconstructed from a given basic $\pi_b$ (e.g., a process plan that prescribes a different action for the same cyclic behaviour of the system when the number of cycles is a prime number). Nevertheless, the result above is crucial for computing the "correct" set of basic plans $\pi_b$ for a task expression $\mathcal{T}$ (i.e., those where $\pi_{\{\pi_b\}}$ is nonblocking). It can be shown that from *the set* of *all* basic process plans $\bar{\Pi} = \{\pi_1, \ldots, \pi_q\}$ for a task expression $\mathcal{T}$ such that $\pi_{\{\pi_i\}}$ is nonblocking, we can reconstruct *any* process plan $\pi$ for $\mathcal{T}$. The proof of this claim is involved and we omit it due to lack of space, as it is not needed to prove the correctness of our approach. Rather it supports (in addition to Theorem 1) the intuition that basic plans represent the minimum information required to reconstruct process plans. Intuitively, a process plan for $\mathcal{T}$ can be reconstructed from $\bar{\Pi}$ by means of a function $f : \mathcal{H} \mapsto |\bar{\Pi}|$ that prescribes, at each step of the current trajectory $\tau$, which basic plan $\pi_{f(\tau)}$ to use.

### 4.3 Process-Plan Simulation Relation

In this section, we capture the ability to realise transitions in the recipe as a *property* relating states of the recipe with states of the topology and resource vectors.

**Definition 5.** Let $\mathbf{P} = \langle \underline{s}^0, \underline{S}, T, \Longrightarrow \rangle$ be a topology and $\mathcal{R} = \langle s^0, S, L, \delta_{\mathcal{R}} \rangle$ a recipe. A *process-simulation relation* is a relation $\text{PSIM} \subseteq \underline{S} \times S \times V$,[3] such that a tuple $\langle \underline{s}, s, \mathbf{r} \rangle \in \text{PSIM}$ implies that for any $\mathcal{T}$, if $s \xrightarrow{\mathcal{T}} s'$ for some $s'$, then there exists a process plan $\pi$ such that for each complete trajectory $\tau$ of $\pi$ from $(\underline{s}, \mathbf{r})$, we have that (I) $\tau$ realises $\mathcal{T}$ (as defined above); and (II) $\langle \underline{s}', s', \mathbf{r}' \rangle \in \text{PSIM}$, where $(\underline{s}', \mathbf{r}') = last(\tau)$.

[3] Recall that $V$ is the set of all resource vectors.

Thus, a state $s$ of a recipe $\mathcal{R}$ is said to be *process-simulated* by a state $\underline{s}$ of a topology $\mathbf{P}$ with respect to a resource vector $\mathbf{r}$ if there exists a process-simulation relation PSIM such that $\langle \underline{s}, s, \mathbf{r} \rangle \in \text{PSIM}$. Moreover, $\mathcal{R} = \langle s^0, S, L, \delta_{\mathcal{R}} \rangle$ is process-simulated by $\mathbf{P} = \langle \underline{s}^0, \underline{S}, T, \Longrightarrow \rangle$ if $s^0$ is process-simulated by $\underline{s}^0$ with respect to $\mathbf{r}^0$.

The definition states that no matter how the recipe evolves from $s$ (according to transition choices made by runtime tests), there exists a process plan whose complete trajectories realise the requested task expression. Crucially, the process recipe cannot control which particular trajectory of the process plan is followed, as the topology is non-deterministic.

**Definition 6 (Realisability of a recipe).** Given a state $\underline{s}_0$ and a resource vector $\mathbf{r}_0$, a recipe $\mathcal{R} = (s^0, S, L, \delta_{\mathcal{R}})$ is *realisable* from $(\underline{s}_0, \mathbf{r}_0)$ iff there exists a function $\omega : \underline{S} \times V \times \delta_{\mathcal{R}} \mapsto \Pi$ such that:

- for each transition $s^0 \xrightarrow{\mathcal{T}} s'$ in $\mathcal{R}$, the plan $\omega(\underline{s}_0, \mathbf{r}_0, s^0 \xrightarrow{\mathcal{T}} s') = \pi$ is defined and it is a process plan for $\mathcal{T}$ from $(\underline{s}_0, \mathbf{r}_0)$;
- if $\omega(\underline{s}, \mathbf{r}, s \xrightarrow{\mathcal{T}} s') = \pi$ is defined, then
  - $\pi$ is a process plan for $\mathcal{T}$ from $(\underline{s}, \mathbf{r})$; and
  - for each outcome $(\underline{s}', \mathbf{r}')$ of a complete trajectory $\tau$ of $\pi$ from $(\underline{s}, \mathbf{r})$, and for each $s' \xrightarrow{\mathcal{T}'} s''$ in $\mathcal{R}$, we have that $\omega(\underline{s}', \mathbf{r}', s' \xrightarrow{\mathcal{T}'} s'')$ is defined. ■

Recall that if $\pi$ is a process plan for $\mathcal{T}$ from $\sigma$, then any complete trajectory of $\pi$ from $\sigma$ realises $\mathcal{T}$. Finally, a recipe $\mathcal{R}$ is *realisable* in a *topology* $\mathbf{P} = \langle \underline{s}^0, \underline{S}, T, \Longrightarrow \rangle$ if $\mathcal{R}$ is realisable from $(\underline{s}^0, \mathbf{r}^0)$. The above definition only requires the *existence* of a function $\omega$ which returns *a possible* "correct" plan at each step. The notion of realisability in a topology is closely related to the notion of plan-based simulation in [De Giacomo *et al.*, 2016] and $\mathcal{T}$-realisation in [De Giacomo *et al.*, 2010]. However, unlike that setting, we do not consider a planning domain due to the high modularity of our setting, as well as the presence of various low-level details (such as resource vectors and simultaneous tasks).

**Theorem 2.** A process recipe $\mathcal{R}$ is realisable in a topology $\mathbf{P}$ iff $\mathbf{P}$ process-simulates $\mathcal{R}$.

*Proof Sketch.* Given $(\underline{s}, \mathbf{r})$, assume that a function $\omega$ as above is defined for each $s \xrightarrow{\mathcal{T}} s'$ in $\mathcal{R}$, namely, $\omega(\underline{s}, \mathbf{r}, s \xrightarrow{\mathcal{T}} s') = \pi$, but $\mathbf{P}$ does not process-simulate $\mathcal{R}$. By Def. 5 this implies that there is no process plan, including $\pi$, such that each of its complete trajectories $\tau$ realise $\mathcal{T}$, or the $\langle \underline{s}', s', \mathbf{r}' \rangle \notin \text{PSIM}$ with $(\underline{s}', \mathbf{r}') = last(\tau)$. These two cases violate the first and second item of Def. 6, respectively; hence, $\omega$ does not exist. If instead $\langle \underline{s}^0, s^0, \mathbf{r}^0 \rangle \in \text{PSIM}$, then, by definition, there exists a plan $\pi$ as in Def. 5 for each $s^0 \xrightarrow{\mathcal{T}} s'$ in $\mathcal{R}$; hence, we can build the function $\omega$ so that $\omega(\underline{s}^0, \mathbf{r}^0, s^0 \xrightarrow{\mathcal{T}} s') = \pi$. The same argument can be applied by induction on $\langle \underline{s}', s', \mathbf{r}' \rangle$, where $(\underline{s}', \mathbf{r}')$ is the outcome of any complete trajectory of $\pi$, as $\langle \underline{s}', s', \mathbf{r}' \rangle \in \text{PSIM}$ by hypothesis. □

### 4.4 Process Plan Controllers

Intuitively, a process plan controller encodes a *set of* functions $\omega$ as in Definition 6, i.e., a function that associates at

least one process plan to each transition (task expression) of a recipe, such that for each possible resulting trajectory, the function can still associate another process plan as the recipe is progressed forward. We use a finite-state representation based on the notion of process-plan simulation relation.

**Definition 7.** Given a topology $\mathbf{P}$ and process recipe $\mathcal{R}$, a *process plan controller* for $\mathcal{R} = \langle s^0, S, Lang(\mathcal{T}), \delta_{\mathcal{R}} \rangle$ in $\mathbf{P}$ is a tuple $\mathcal{C} = \langle \text{PSIM}, \delta_{\mathcal{R}}, \Pi, \delta \rangle$ with:

- $\text{PSIM} \neq \varnothing$ is a process plan-simulation relation, whose elements correspond to the set of states in the controller;
- $\delta_{\mathcal{R}}$ is the recipe transition relation;
- $\Pi$ is a set of process plans;
- $\delta : \text{PSIM} \times \delta_{\mathcal{R}} \times \Pi \times \text{PSIM}$ is a transition relation, defining transitions from one state to another, by executing a process plan $\pi$ that realises a task expression $\mathcal{T}$. A transition $\langle \underline{s}', s', \mathbf{r}' \rangle \in \delta(\langle \underline{s}, s, \mathbf{r} \rangle, tr, \pi)$ exists for $tr = s \xrightarrow{\mathcal{T}} s'$, also denoted $\langle \underline{s}, s, \mathbf{r} \rangle \xrightarrow{tr,\pi} \langle \underline{s}', s', \mathbf{r}' \rangle$, iff $s \xrightarrow{\mathcal{T}} s'$ is in $\delta_{\mathcal{R}}$ and:
  - $\pi$ is a process plan for $\mathcal{T}$, i.e., it is nonblocking and all complete trajectories of $\pi$ realise $\mathcal{T}$;
  - a complete trajectory $\tau$ of $\pi$ exists from $(\underline{s}, \mathbf{r})$ with $(\underline{s}', \mathbf{r}') = last(\tau)$, and for any complete trajectory $\tau'$ of $\pi$ from $(\underline{s}, \mathbf{r})$ with $(\underline{s}'', \mathbf{r}'') = last(\tau')$, we have $\langle \underline{s}, s, \mathbf{r} \rangle \xrightarrow{tr,\pi} \langle \underline{s}'', s', \mathbf{r}'' \rangle$. ■

As a direct consequence of Theorem 2 and the definition of process plan controller, we get the following result.

**Theorem 3.** A recipe $\mathcal{R}$ is realisable in a topology $\mathbf{P}$ iff there exists a process plan controller for $\mathcal{R}$ in $\mathbf{P}$.

*Proof Sketch.* It follows by construction of the process plan controller $\mathcal{C}$. It is possible to compute the function $\omega$ in Definition 6 as follows: $\omega(\underline{s}, \mathbf{r}, s \xrightarrow{\mathcal{T}} s') = \pi$ iff $\langle \underline{s}, s, \mathbf{r} \rangle \xrightarrow{tr,\pi} \langle \underline{s}', s', \mathbf{r}' \rangle$, considering $tr = s \xrightarrow{\mathcal{T}} s'$. Indeed by Theorem 2 if $\langle \underline{s}, s, \mathbf{r} \rangle \in \text{PSIM}$ then $\omega(\underline{s}, \mathbf{r}, s \xrightarrow{\mathcal{T}} s')$ is defined for every recipe transition $s \xrightarrow{\mathcal{T}} s'$. □

## 5 Computing Basic Plans

In this section, we provide an algorithm that computes a process-plan simulation relation given a process recipe $\mathcal{R}$ and a topology $\mathbf{P}$, and in doing so also computes a process plan controller for $\mathcal{R}$ and $\mathbf{P}$. We extend the algorithms in [de Silva *et al.*, 2016] to handle non-deterministic topologies and to compute a *set* of basic plans, having the property as in Theorem 1, for each transition of the process recipe. By the theorem, these can be used to reconstruct history-based plans.

Given a process recipe $\mathcal{R}$, a topology $\mathbf{P}$, a state $\underline{s}$ of $\mathbf{P}$, a resource vector $\mathbf{r}$ and a state $s$ of $\mathcal{R}$, Algorithm 1 determines, for each transition $tr = s \xrightarrow{\mathcal{T}} s'$, whether there exists a basic plan for $\mathcal{T}$ from $(\underline{s}, \mathbf{r})$, and in turn, whether the same holds for each transition from $s'$ (as required by Definition 6). To this end, $tr$ is passed as a parameter to Algorithm 2 to continue checking from $s'$. We use $\pi_b^0$ to denote the empty plan.

Algorithm 2 uses two auxiliary functions: given a task expression $\mathcal{T} = \mathcal{T}_1; \mathcal{T}_2; \ldots; \mathcal{T}_n$ with $n > 0$, the first element of $\mathcal{T}$ is defined as $\text{FST}(\mathcal{T}) = \mathcal{T}_1$ and the rest of its elements as $\text{RST}(\mathcal{T}) = \mathcal{T}_2; \ldots; \mathcal{T}_n$ if $n > 1$, and as $\text{RST}(\mathcal{T}) = \epsilon$ if $n = 1$. Intuitively, Algorithm 2 performs a depth-first search of the topology to check whether there exists a basic plan for the task expression $\mathcal{T}_{cur}$ (initially $\mathcal{T}$) from $\sigma_\downarrow$. In particular, the outer loop (lines 2 to 2) considers each unique label $\mathbf{t}$ in the outgoing topology transitions from $\underline{s}_\downarrow$; lines 2 to 2 apply if $\mathbf{t}$

**Algorithm 1** $\text{FINDSIM}(\mathcal{R}, \mathbf{P}, \underline{s}, \mathbf{r}, s)$

**Input:** a process recipe $\mathcal{R} = (s^0, S, L, \delta_{\mathcal{R}})$, a topology $\mathbf{P}$, the topology state $\underline{s}$, resource vector $\mathbf{r}$ and recipe state $s$.

1: $\delta, \delta', \text{PSIM}, \text{PSIM}' := \varnothing$
2: **for** each recipe transition $tr = (s, \mathcal{T}, s') \in \delta_{\mathcal{R}}$ **do**
3: $\quad(\text{PSIM}', \delta') := \text{EVAL}(\mathcal{R}, \mathbf{P}, \mathcal{T}, tr, (\underline{s}, \mathbf{r}), \pi_b^0, (\underline{s}, \mathbf{r}), \varnothing)$
4: $\quad$**if** $\text{PSIM}' = \varnothing$ **then return** $(\varnothing, \varnothing)$
5: $\quad\text{PSIM} := \text{PSIM} \cup \text{PSIM}'; \delta := \delta \cup \delta'$
6: **return** $(\text{PSIM} \cup \{(\underline{s}, s, \mathbf{r})\}, \delta)$

**Algorithm 2** $\text{EVAL}(\mathcal{R}, \mathbf{P}, \mathcal{T}_{cur}, tr, \tau, \pi_b, \sigma_0, \Sigma)$

**Input:** a process recipe $\mathcal{R}$, a topology $\mathbf{P} = (\underline{s}^0, \underline{S}, T^n, \Longrightarrow)$, the current task expression $\mathcal{T}_{cur}$ and recipe transition $tr = (s, \mathcal{T}, s_{nxt})$, the current trajectory $\tau$ and basic plan $\pi_b$, the initial couple $\sigma_0 = (\underline{s}_0, \mathbf{r}_0)$ and the visited couples $\Sigma$.

1: $\sigma_\downarrow := (\underline{s}_\downarrow, \mathbf{r}_\downarrow) := last(\tau)$
2: $(\text{PSIM}, \delta) := (\varnothing, \varnothing)$
3: **if** $\mathcal{T}_{cur} = \epsilon$ **then**
4: $\quad x := (\underline{s}_0, s, \mathbf{r}_0) \xrightarrow{tr,\pi_b} (\underline{s}_\downarrow, s_{nxt}, \mathbf{r}_\downarrow)$
5: $\quad(\text{PSIM}, \delta) := \text{FINDSIM}(\mathcal{R}, \mathbf{P}, \underline{s}_\downarrow, \mathbf{r}_\downarrow, s_{nxt})$
6: $\quad$**return** $(\text{PSIM}, \delta \cup \{x\})$
7: **if** $\sigma_\downarrow \in \Sigma$ **then return** $(\{\sigma_\downarrow\}, \varnothing)$
8: $\sigma_{fst} := \epsilon$
9: **for** each $\mathbf{t}$ such that $(\underline{s}_\downarrow, \mathbf{t}, \underline{s})$ exists in $\Longrightarrow$ **do**
10: $\quad(\text{PSIM}_\mathbf{t}, \delta_\mathbf{t}) := (\varnothing, \varnothing)$
11: $\quad$**if** $\mathbf{t}$ is observable and $\mathbf{r}' = \text{AL}(\mathbf{r}_\downarrow, \text{FST}(\mathcal{T}_{cur}), \mathbf{t})$ **then**
12: $\quad\quad$**for** each $(\underline{s}_\downarrow, \mathbf{t}, \underline{s}) \in \Longrightarrow$ **do**
13: $\quad\quad\quad\sigma := (\underline{s}, \text{MOV}(\mathbf{r}', \mathbf{t}))$
14: $\quad\quad\quad\pi_b(\tau) := \mathbf{t}$
15: $\quad\quad\quad(\text{PSIM}_{\underline{s}}, \delta_{\underline{s}}) :=$ $\text{EVAL}(\mathcal{R}, \mathbf{P}, \text{RST}(\mathcal{T}_{cur}), tr, \tau \xrightarrow{\mathbf{t}} \sigma, \pi_b, \sigma_0, \varnothing)$
16: $\quad\quad\quad$**if** $\text{PSIM}_{\underline{s}} = \varnothing$ **then** $(\text{PSIM}_\mathbf{t}, \delta_\mathbf{t}) := (\varnothing, \varnothing)$; **break**
17: $\quad\quad\quad$**else** $(\text{PSIM}_\mathbf{t}, \delta_\mathbf{t}) := (\text{PSIM}_\mathbf{t} \cup \text{PSIM}_{\underline{s}}, \delta_\mathbf{t} \cup \delta_{\underline{s}})$
18: $\quad\quad$**end for**
19: $\quad\quad(\text{PSIM}, \delta) := (\text{PSIM} \cup \text{PSIM}_\mathbf{t}, \delta \cup \delta_\mathbf{t})$
20: $\quad$**else if** $\mathbf{t}$ is unobservable **then**
21: $\quad\quad$**for** each $(\underline{s}_\downarrow, \mathbf{t}, \underline{s}) \in \Longrightarrow$ **do**
22: $\quad\quad\quad\sigma := (\underline{s}, \text{MOV}(\mathbf{r}_\downarrow, \mathbf{t}))$
23: $\quad\quad\quad\pi_b(\tau) := \mathbf{t}$
24: $\quad\quad\quad\Sigma' := \Sigma \cup \{\sigma_\downarrow\}$
25: $\quad\quad\quad(\text{PSIM}_{\underline{s}}, \delta_{\underline{s}}) := \text{EVAL}(\mathcal{R}, \mathbf{P}, \mathcal{T}_{cur}, tr, \tau \xrightarrow{\mathbf{t}} \sigma, \pi_b, \sigma_0, \Sigma')$
26: $\quad\quad\quad$**if** $\text{PSIM}_{\underline{s}} = \varnothing$ **then**
27: $\quad\quad\quad\quad(\text{PSIM}_\mathbf{t}, \delta_\mathbf{t}) := (\varnothing, \varnothing)$; $\sigma_{fst} := \epsilon$; **break**
28: $\quad\quad\quad$**else if** $\text{PSIM}_{\underline{s}} = \{\sigma'\}$ and $\delta_{\underline{s}} = \varnothing$ **then**
29: $\quad\quad\quad\quad\sigma_{fst} := \text{EARLIER}(\sigma_{fst}, \sigma', \tau)$
30: $\quad\quad\quad$**else** $(\text{PSIM}_\mathbf{t}, \delta_\mathbf{t}) := (\text{PSIM}_\mathbf{t} \cup \text{PSIM}_{\underline{s}}, \delta_\mathbf{t} \cup \delta_{\underline{s}})$
31: $\quad\quad$**end for**
32: $\quad\quad(\text{PSIM}, \delta) := (\text{PSIM} \cup \text{PSIM}_\mathbf{t}, \delta \cup \delta_\mathbf{t})$
33: **end for**
34: **if** $\text{PSIM} \neq \varnothing$ **then return** $(\text{PSIM}, \delta)$
35: **else if** $\sigma_{fst} \notin \{\epsilon, \underline{s}_\downarrow\}$ **then return** $(\{\sigma_{fst}\}, \varnothing)$
36: **else return** $(\varnothing, \varnothing)$

is observable and the first task in $\mathcal{T}_{cur}$ can be allocated to it, and lines 2 to 2 apply if $\mathbf{t}$ is unobservable.

For each observable transition associated with a particular label $\mathbf{t}$, lines 2 to 2 prescribe the following. First, the "successor" couple $\sigma = (\underline{s}, \mathbf{r})$ is created to represent the allocation of $\mathcal{T}_{cur}$ to $\mathbf{t}$, and the movement of parts via synchronisations. Second, line 2 updates the plan $\pi_b$ with the vector $\mathbf{t}$ labelling the topology transition $(\underline{s}_{\downarrow}, \mathbf{t}, \underline{s})$; this plan is then passed as a parameter in the recursive call to EVAL. Finally, if the rest of the tasks in $\mathcal{T}_{cur}$ cannot be realised from $\sigma$, all non-deterministic transitions labelled with $\mathbf{t}$ are disregarded. Otherwise, new tuples in $\text{PSIM}_{\underline{s}}$ and the corresponding controller transition $\delta_{\underline{s}}$ for $\mathbf{t}$ are stored (line 2), and the loop continues. If $\mathbf{t}$ is unobservable, the steps are similar to those described above, except for lines 2 and 2. Line 2 applies to the case in which $\tau \xRightarrow{\mathbf{t}} \sigma$ constitutes a cycle "back" in the current trajectory $\tau$ (i.e., line 2 was executed in the recursive call at line 2): if this is the case, then we compare $\sigma'$ with the state $\sigma_{fst}$ (initially the empty string) which is the "earliest" state that is possible to reach, through cycles, from the current topology state $\underline{s}_{\downarrow}$. The intuition is as follows: we need to make sure that there *exists* the possibility, from $\sigma_{\downarrow}$, to cycle back to a state $\sigma_{fst}$ from which it is still possible to find a complete trajectory whose execution allocates the task expression $\mathcal{T}_{cur}$. If this is the case, then the cycle in the current trajectory of plan $\pi_b$ is "admissible": it (still) guarantees the nonblocking condition of the plan $\pi_{\{\pi_b\}}$. Formally, we define the function $\text{EARLIER}(\sigma_1, \sigma_2, \tau) = \sigma_1$ if $\sigma_1$ occurs before $\sigma_2$ in $\tau$ (or $\sigma_2 = \epsilon$), and $\text{EARLIER}(\sigma_1, \sigma_2, \tau) = \sigma_2$ if $\sigma_2$ occurs before $\sigma_1$ in $\tau$ (or $\sigma_1 = \epsilon$). Then $(\text{PSIM}, \delta)$ is returned in line 2, consisting of the simulation relations and corresponding controller's transitions. Otherwise, we must rely on there being a state that appears in $\tau$ before $\underline{s}_{\downarrow}$, and possibly as early as $\sigma_{fst}$, from where a complete trajectory can be found and the task allocated. Thus, we "propagate back" $\sigma_{fst}$ until we can exit the current cycle, or until the recursive step is reached with $\sigma_{\downarrow} = \sigma_{fst}$, from where an exit must then exist if a non-empty simulation relation is to be returned. Line 2 checks whether the algorithm has reached a couple $\sigma_{\downarrow}$ visited earlier, and if so returns it, guaranteeing that the plan is bounded.

A variant of this algorithm for deterministic topologies has been implemented as part of an agent based control architecture for manufacturing, designed to address rapidly changing product and process requirements [de Silva *et al.*, 2017]. Within this architecture, a process plan controller is used to select basic plans for task expressions that appear in the recipe. Such plans are represented in the Behaviour to Markup Manufacturing Language (B2MML) ISA-95 standard, which is a format that is interpretable by real-world manufacturing execution systems. The implementation also takes into account additional details such as parameters and materials, which have been omitted in this paper.

## 6 Related Work

The problem we consider has similarities with AI planning [Nau *et al.*, 2004; Cimatti *et al.*, 2003]. We adopted a synthesis-based approach due to the difficulties of encoding process recipes and production topologies as planning goals and planning domains. In our setting, there are several features that make such an encoding difficult, including conditional goals (in the recipe), interleaved and parallel tasks.

The notion of process plan controller is closely related to other approaches. For example *agent planning programs* [De Giacomo *et al.*, 2010; 2016] are finite-state programs where transitions prescribe propositional achievement and maintenance goals that must be realised on a planning domain by implementing each transition via a conditional plan. The solution approach is based on restricted forms of LTL synthesis, and is able to cope with non-deterministic *propositional* planning domains with explicit fairness constraints, thus producing strong cyclic plans. However, as explained in Section 4.1, the manufacturing setting has a number of features which preclude a straightforward application of this approach, and standard notions of strong cyclic planning cannot be applied.

Our notion of a terminating plan is also similar to history-based terminating plans in [De Giacomo *et al.*, 2016], and related to the notion of (memoryless) strong acyclic plans [Cimatti *et al.*, 2003]. As we explain in Section 4.1, our approach differs from [De Giacomo *et al.*, 2016] in that the trajectories we consider are not finite. Similarly, the state-action table representation used for strong acyclic plans in [Cimatti *et al.*, 2003] is insufficiently flexible for process plans, as it is memoryless, whereas in our setting we must choose actions based on the current history (evolution of the topology) rather than simply the current state of the topology.

Over the last decade there has been a growing body of work on automation to achieve flexibility, resilience, and monitoring in manufacturing. For example, *Flexible Manufacturing Systems* [Browne *et al.*, 1984; Sethi and Sethi, 1990; ElMaraghy, 2005] increase the variety of parts and products that can be produced, while *Reconfigurable Manufacturing Systems* [Bi *et al.*, 2008; Koren *et al.*, 1999; Mehrabi *et al.*, 2000; Smale and Ratchev, 2009] allow more rapid response to market changes for a certain product family. In [Felli *et al.*, 2016] an approach is presented to the synthesis of controllers capable of producing multiple instances of the same product simultaneously. However, as none of this work has addressed the manufacture of products in the context of manufacturing as a service, nor considered fairness assumptions or differentiated observable and unobservable tasks, they are restricted to acyclic plans when resources are non-deterministic.

## 7 Conclusions and Future Work

We extended previous approaches to the realisability and control problems for process recipes in manufacturing systems consisting of non-deterministic resources, and where operations can be performed in parallel on the same part. We formally defined the notions of process plans and process plan controllers for these systems, where loops in plans must be allowed due to intrinsic fairness assumptions. In this paper we assume that a manufacturing facility is always initially in the initial state of the topology with an empty resource vector. In future work, we plan to relax this assumption and generalise our approach to consider an arbitrary initial state, where parts are already assigned to resources, and the topology is currently being orchestrated to realise another process recipe.

## References


[Bi *et al.*, 2008] Zhuming M. Bi, Sherman Y.T. Lang, Weiming Shen, and Lihui Wang. Reconfigurable manufacturing systems: the state of the art. *International Journal of Production Research*, 46(4):967–992, 2008.

[Browne *et al.*, 1984] Jim Browne, Didier Dubois, Keith Rathmill, Suresh P. Sethi, and Kathryn E. Stecke. Classification of flexible manufacturing systems. *The FMS magazine*, 2(2):114–117, 1984.

[Chaplin *et al.*, 2015] J. C. Chaplin, O. J. Bakker, L. de Silva, D. Sanderson, E. Kelly, B. Logan, and S. M. Ratchev. Evolvable assembly systems: A distributed architecture for intelligent manufacturing. In A. Dolgui, J. Sasiadek, and M. Zaremba, editors, *15th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2015)*, volume 48 of *IFAC-PapersOnLine*, pages 2065–2070, Ottawa, Canada, May 2015. Elsevier.

[Cimatti *et al.*, 2003] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35 – 84, 2003.

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardiña. Generalized planning with loops under strong fairness constraints. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 351–361, 2010.

[De Giacomo *et al.*, 2016] Giuseppe De Giacomo, Alfonso Emilio Gerevini, Fabio Patrizi, Alessandro Saetti, and Sebastian Sardiña. Agent planning programs. *Artificial Intelligence*, 231:64–106, 2016.

[de Silva *et al.*, 2016] Lavindra de Silva, Paolo Felli, Jack C. Chaplin, Brian Logan, David Sanderson, and Svetan Ratchev. Realisability of production recipes. In G. A. Kaminka, M. Fox, P. Bouquet, Hullermeijer E., Dignum F., Dignum V., and van Harmalen F., editors, *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, pages 1449–1457. IOS Press, 2016.

[de Silva *et al.*, 2017] Lavindra de Silva, Paolo Felli, Jack C. Chaplin, Brian Logan, David Sanderson, and Svetan Ratchev. Synthesising industry-standard manufacturing process controllers. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1811–1813, 2017. Accompanying video available at: https://youtu.be/SEveuikI3p8.

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

[Felli *et al.*, 2016] Paolo Felli, Brian Logan, and Sebastian Sardina. Parallel behavior composition for manufacturing. In Subbarao Kambhampati, editor, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 271–278, 2016.

[Koren *et al.*, 1999] Yoram Koren, Uwe Heisel, Francesco Jovane, Toshimichi Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel. Reconfigurable manufacturing systems. *CIRP Annals-Manufacturing Technology*, 48(2):527–540, 1999.

[Lu *et al.*, 2014] Yuqian Lu, Xun Xu, and Jenny Xu. Development of a hybrid manufacturing cloud. *Journal of Manufacturing Systems*, 33(4):551–566, 2014.

[Mehrabi *et al.*, 2000] Mostafa G. Mehrabi, A. Galip Ulsoy, and Yoram Koren. Reconfigurable manufacturing systems: key to future manufacturing. *Journal of Intelligent Manufacturing*, 11(4):403–419, 2000.

[Nau *et al.*, 2004] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

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

[Sethi and Sethi, 1990] Andrea Krasa Sethi and Suresh Pal Sethi. Flexibility in manufacturing: a survey. *International Journal of Flexible Manufacturing Systems*, 2(4):289–328, 1990.

[Smale and Ratchev, 2009] Daniel Smale and Svetan Ratchev. A capability model and taxonomy for multiple assembly system reconfigurations. In *Proceedings of IFAC Symposium on Information Control Problems in Manufacturing*, volume 13, pages 1923–1928, 2009.

[TSB, 2012] A Landscape for the Future of High Value Manufacturing in the UK. Technical report, Technology Strategy Board, 2012.