LTL$_f$ Goal-oriented Service Composition

Giuseppe De Giacomo[1,2], Marco Favorito[3] and Luciana Silo[2,4]

[1]University of Oxford, UK
[2]Sapienza University of Rome, Italy
[3]Banca d'Italia, Italy
[4]Camera dei Deputati, Italy

Abstract

Service compositions *à la* Roman model consist of realizing a virtual service by orchestrating suitably a set of already available services, where all services are described procedurally as (possibly nondeterministic) transition systems. In this paper, we study a goal-oriented variant of the service composition *à la* Roman Model, where the goal specifies the allowed traces declaratively via Linear Temporal Logic on finite traces (LTL$_f$). Specifically, we want to synthesize a controller to orchestrate the available services to produce together a trace satisfying a specification in LTL$_f$. To do so, we combine techniques from reactive synthesis, FOND Planning, and the Roman Model for service composition. This framework has several interesting applications, including Smart Manufacturing and Digital Twins.

Keywords

Service Composition, Linear Temporal Logic on finite traces, LTL$_f$ Synthesis, FOND Planning

1. Introduction

Service composition, a well-established topic in the field of Web services, refers to the ability to combine Web services into a business process. More properly, it involves managing and sequencing interactions between Web services, orchestrating them into a larger transaction. This approach enhances the flexibility and adaptability of business processes by enabling them to be constructed from reusable services, allowing organizations to quickly adjust their processes in response to changing business requirements or technological advancements. For example, a transaction involving the addition of a customer to a bank account service could concurrently initiate the creation of multiple accounts while updating the customer information within the customer service. All of these requests are managed in the context of a larger business process flow that either succeeds or fails as a whole [1]. The problem of service composition has been considered in the literature for over two decades. Particularly interesting in this context is the so-called Roman Model [2, 3, 4] where services are *conversational*, i.e., have an internal state and are procedurally described as finite transition systems (TS), where at each state the service offers a certain set of actions, and each action changes the state of the service in some way. The designer is interested in generating a new service, called *target*, which is described as the other service; however, it is virtual in the sense that no code is associated with its actions. Therefore, to execute the target, one has to delegate each of its actions to some of the available services by suitably orchestrating them, taking into consideration the state of the target and the available services are in. Service composition amounts to synthesizing a controller that can suitably orchestrate the executions of the available services so as to guarantee that the target actions are always delegated to some service that can actually execute them in its current state. The original paper on the Roman Model [2] has been the inspiration for a line of work in AI on behaviour composition where nondeterminism, in the sense of partial controllability as in Fully-Observable Non-Deterministic (FOND) strong planning [5, 6]) has played a prominent role [7]. Recently a renewed interest in service composition *à la* Roman

PMAI@ECAI24: International ECAI Workshop on Process Management in the AI era, October 19, 2024, Santiago De Compostela, Spain

degiacomo@diag.uniroma1.it (G. D. Giacomo); marco.favorito@bancaditalia.it (M. Favorito); silo@diag.uniroma1.it (L. Silo)
0000-0001-9680-7658 (G. D. Giacomo); 0000-0001-9566-3576 (M. Favorito); 0000-0001-7250-8979 (L. Silo)

Model is stemming out of applications in smart manufacturing, where, through digital twins technology, manufacturing devices can export their behaviour as transition systems and hence being orchestrated very much in the same way as service did back in the early 2000's, see e.g., [8, 9, 10].

Interestingly, these new applications are also promoting to move from a procedural specification of the target to a declarative one, as advocated by the declarative business processes literature, through the so-called DECLARE paradigm [11, 12, 13]. In other words, the target would ideally be specified in DECLARE, and so, in Linear Temporal Logic on finite and process traces (LTL_f) [14], with the assumption that specifications are about the possible sequence of actions (vs sequences of fluent values of the domain as in planning for LTL_f goals [15, 16]), and the simplification that only one action can be selected at each point in time [11, 17].

In fact, LTL_f as a specification of the target can be utilized to write two different kinds of target specification, namely *process-oriented target specification* or *goal-oriented target specification*. In the first case, very much like in the DECLARE paradigm, one uses the LTL_f specification to specify the process itself consisting of all the traces satisfying the LTL_f formula, which in turn corresponds to implicitly specifying the transition system consisting of the deterministic finite automaton (DFA) equivalent to the LTL_f formula [14]. In this case, after a preprocessing of the LTL_f specification to obtain the target transition system (i.e., the DFA corresponding to the formula) as in [14], the composition can be performed as with the techniques used for the standard Roman Model [7].

In this paper, we study the case where LTL_f is used as a goal-oriented specification. This is a novel variant of the composition problem, where we are given a LTL_f goal, and we want to synthesize an orchestrator that, on the one hand, reactively chooses actions to form a sequence that satisfies the goal and, on the other hand, delegates each action to an available service in such a way that at any point the delegated action can be executed by the delegated service and at the end of the sequence satisfying the LTL_f formula, all services are in their final states. Specifically, we consider the available services as nondeterministic, i.e., partially controllable (similarly to FOND strong planning) as in [3, 7].

Note that this problem is different from other goal-oriented service composition frameworks. In [18] Hierarchical Task Network (HTN) planning is used for service composition. HTN planning is based on the notion of composite tasks that can be refined to atomic tasks using predefined methods. While based on high-level specification of services, their approach does not support nondeterministic services. The work [19] allows the modeling of nondeterministic behaviors but not of stateful services nor high-level temporal goal specification. Authors in [20] describe services as atomic actions where only I/O behaviour is modelled, and the ontology is constituted by propositions and actions; hence services are not stateful as ours. De Giacomo et al. [21] study a stochastic version in a goal-oriented setting in which the optimization of the cost utilization is subordinated to the maximal probability of satisfaction of the goal by means of lexicographic optimization. Here instead we assume nondeterministic services.

To provide a solution technique in this case, from the LTL_f specification, we compute in linear time a *symbolic representation* of the corresponding *Nondeterministic Finite Automaton (NFA)*, we do so by essentially adopting the Alternating Automaton (AFA) associated to the formula as the symbolic representation of the NFA. Then we adapt the interleaving procedure introduced in [22] to encode the symbolic NFA corresponding to LTL_f temporally extended goals into special planning actions domains. However, while in [22] considers these symbolic NFAs in deterministic planning domains, we use the technique in a nondeterministic (adversarial) planning domain that encodes all the possible service actions at each point of the computation. Notably, this is possible in our case because the target specification is an LTL_f specification of the desired sequence of target actions and not of fluent evaluation as in standard planning. Note that, the possibility of exploiting symbolic NFAs contrasts with the need of adopting DFAs required for the process-oriented target specification, which are exponentially larger than corresponding NFAs in the worst case. We implemented our approach and evaluated different heuristics and Torres and Baier's encodings to show the feasibility of our solution technique.

Although this paper has a foundational nature, our paper gives the foundations and solution techniques of goal-oriented compositions, which are indeed envisioned in the current literature on smart manufacturing where the notion of goal-oriented target specification is increasingly championed [23, 8, 9]. The appendix with the proofs and experimental results can be found at this link: https://bit.ly/3x5lXre.

2. Preliminaries

Automata theory. A *deterministic finite automaton* (DFA) is a tuple $\mathcal{A} = \langle \mathcal{P}, Q, q_0, F, \delta \rangle$ where: (*i*) $\mathcal{P}$ is the alphabet, (*ii*) Q is a finite set of states, (*iii*) q_0 is the initial state, (*iv*) $F \subseteq Q$ is the set of accepting states and (*v*) $\delta : Q \times \mathcal{P} \to Q$ is a total transition function. A *nondeterministic finite automaton* (NFA) is defined similarly to DFA except that δ is defined as a relation, i.e. $\delta \subseteq Q \times \mathcal{P} \times Q$. An *alternating finite automaton* (AFA) [24, 25] is a generalization of DFA and NFA, where δ is defined as $\delta : Q \times \Sigma \to B^+(Q)$, where $B^+(Q)$ is a set of positive boolean formulas whose atoms are states of Q. By $\mathcal{L}(\mathcal{A})$, we mean the set of all traces over Σ accepted by an automaton $\mathcal{A}$. An AFA $\mathcal{A}_A = \langle \mathcal{P}, Q_A, q_0, F_A, \delta_A \rangle$ can be transformed into an equivalent NFA $\mathcal{A}_N = \langle \mathcal{P}, 2^{Q_A}, \{q_0\}, 2^{F_A}, \delta_N \rangle$, where $\delta_N = \{(\bar{q}, a, \bar{q}') \mid \bar{q}' \models \bigwedge_{q \in \bar{q}} \delta_A(q, a)\}$. Note that $\mathcal{A}_N$ can be constructed on-the-fly, see [22].

LTL$_f$ is a variant of Linear Temporal Logic (LTL) interpreted over finite traces [14]. Given a set $\mathcal{P}$ of atomic propositions, LTL$_f$ formulas φ are defined by $\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi$, where a denotes an atomic proposition in $\mathcal{P}$, $\bigcirc$ is the next operator, and $\mathcal{U}$ is the until operator. We use abbreviations for other Boolean connectives, as well as the following: eventually as $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$; always as $\Box\varphi \equiv \neg\Diamond\neg\varphi$; weak next as $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$ (note that, on finite traces, $\neg\bigcirc\varphi$ is not equivalent to $\bigcirc\neg\varphi$); and weak until as $\varphi_1 \,\mathcal{W}\, \varphi_2 \equiv (\varphi_1 \,\mathcal{U}\, \varphi_2 \vee \Box\varphi_1)$, i.e. φ_1 holds until φ_2 or forever. LTL$_f$ formulas are interpreted on finite (possibly empty) traces $a = a_0 \ldots a_{n-1}$ where a_i at instant i is a propositional interpretation over the alphabet $2^{\mathcal{P}}$, and n is the length of the trace. An LTL$_f$ formula can be transformed into an equivalent AFA in linear time in the size of the formula, in a NFA in at most EXPTIME and into an equivalent and in a DFA in at most 2EXPTIME [14]. LTL$_f$ is used in declarative process specification in BPM, through the so called DECLARE paradigm [11]. In this case it is assumed that only one proposition (corresponding to an action) is true at every time point: $\xi_{\mathcal{P}} = \Box(\bigvee_{a \in \mathcal{P}} a) \wedge \Box(\bigwedge_{a,b \in \mathcal{P}, a \neq b} a \to \neg b)$. We call this the DECLARE *assumption*, and we do adopt it in this paper.

FOND Planning. A *Fully-Observable Non-Deterministic* (FOND) domain model can be formalized as a tuple $\mathcal{D} = \langle \mathcal{F}, A, pre, \mathit{eff} \rangle$ where $\mathcal{F}$ is a set of positive literals, A is a set of action labels, pre and eff are two functions that define the preconditions and effects of each action $a \in A$. A planning state s is a subset of $\mathcal{F}$, and a positive literal f holds true in s if $f \in s$; otherwise, f is false in s. Both functions pre and eff take an action label $a \in A$ as an input and return a propositional formula over $\mathcal{F}$ and a set $\{\mathit{eff}_1, \ldots, \mathit{eff}_n\}$ of effects, respectively. Each effect $\mathit{eff}_i \in \mathit{eff}(a)$ is a set of conditional effects each of the form $c \triangleright e$, where c is a propositional formula over $\mathcal{F}$ and $e \subseteq \mathcal{F} \cup \{\neg f \mid f \in \mathcal{F}\}$ is a set of literals from $\mathcal{F}$. Sometimes we write e as a shorthand for the unconditional effect $\emptyset \triangleright e$. An action a can be applied in a state s if $pre(a)$ holds true in s (i.e., $s \models pre(a)$). A conditional effect $c \triangleright e$ is triggered in a state s if c is true in s. Applying a in s yields a successor state s' determined by an outcome *nondeterministically* drawn from $\mathit{eff}(a)$. Let $\mathit{eff}_i \in \mathit{eff}(a)$ be the chosen nondeterministic effect, the new state s' is such that $\forall f \in \mathcal{F}$, f holds true in s' if and only if either (*i*) f was true in s and no conditional effect $c \triangleright e \in \mathit{eff}_i$ triggered in s deletes it ($\neg f \in e$) or (*ii*) there is a conditional effect $c \triangleright e \in \mathit{eff}_i$ triggered in s that adds it ($f \in e$). In case of conflicting effects, similarly to other works [26], we assume delete-before-adding semantics. We use $\delta(s, a)$ to denote the set of possible successor states $\{s'_1, \ldots, s'_n\}$ obtained by executing a in s. Note that if $s \not\models pre(a)$ then $\delta(s, a) = \emptyset$. A FOND planning problem is a tuple $\Gamma = \langle \mathcal{D}, s_0, G \rangle$, where $\mathcal{D}$ is a domain model, $s_0 \subseteq \mathcal{F}$ is the initial state, and G is a formula over $\mathcal{F}$, also called the reachability goal. We now define what it means to solve a planning problem on $\mathcal{D}$. A *FOND planning problem* is a tuple $\Gamma = \langle \mathcal{D}, s_0, G \rangle$, where $\mathcal{D}$ is a domain model, $s_0 \subseteq \mathcal{F}$ is the *initial state*, and G is a formula over $\mathcal{F}$ specifying the goal states. A *trace* of Γ is a finite or infinite sequence $s_0, a_0, s_1, a_1, \ldots$ where s_0 is the initial state, and $s_i \models pre(a_i)$ and $s_{i+1} = \delta(s_i, a_i)$ for each s_i, a_i in the trace. A *strategy* (or *plan*) is a partial function $\pi : (2^{\mathcal{F}})^+ \to A$ such that for every $u \in (2^{\mathcal{F}})^+$, if $\pi(u)$ is defined then $last(u) \models pre(\pi(u))$, i.e., it selects applicable actions. If $\pi(u)$ is undefined, we write $\pi(u) = \bot$. A trace τ is *generated* by π, or simply an π*-trace*, if (i) if $s_0, a_0, \ldots, s_i, a_i$ is a prefix of τ then $\pi(s_0 s_1 \ldots s_i) = a_i$, and (ii) if τ is finite, say $\tau = s_0, a_0, \ldots, a_{n-1}, s_n$, then $\pi(s_0 s_1 \ldots s_n) = \bot$. A strategy π is a *(strong) solution* to Γ if every π-trace is a finite trace τ such that $s_n \models G$.

3. Goal-oriented LTL$_f$ Service Composition

Our composition framework follows the Roman model in the case the available services are nondeterministic. Unlike the Roman model, we have a high-level specification of a goal to accomplish expressed as an LTL$_f$ formula. We want to accomplish such a goal despite the available services having nondeterministic behaviour. We detail our framework below.

Following the Roman Model [2, 3, 7], each *(available) service* is defined as a tuple $\mathcal{S} = \langle \Sigma, A, \sigma_0, F, \delta \rangle$ where: (*i*) Σ is the finite set of service states, (*ii*) A is the finite set of services' actions, (*iii*) $\sigma_0 \in \Sigma$ is the initial state, (*iv*) $F \subseteq \Sigma$ is the set of final states (i.e., states in which the computation may stop but does not necessarily have to), and (*v*) $\delta \subseteq \Sigma \times A \times \Sigma$ is the service transition relation. For convenience, we define $\delta(\sigma, a) = \{\sigma' \mid (\sigma, a, \sigma') \in \delta\}$, and we assume that for each state $\sigma \in \Sigma$ and each action $a \in A$, there exist $\sigma' \in \Sigma$ such that $(\sigma, a, \sigma') \in \delta$ (possibly σ' is an error state σ_u that will never reach a final state). Actions in A denote interactions between service and clients. The behavior of each available service is described in terms of a finite transition system that uses only actions from A.

Our target specification consists of a goal specification φ expressed in LTL$_f$ over the set of propositions A. Given a community of n services $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, where each set of actions $A_i \subseteq A$, a *trace of* $\mathcal{C}$ is a finite or infinite alternating sequence of the form $t = (\sigma_{10} \ldots \sigma_{n0}), (a_0, o_0), (\sigma_{11} \ldots \sigma_{n1}), (a_1, o_1) \ldots$, where σ_{i0} is the initial state of every service $\mathcal{S}_i$ and, for every $0 \leq k$, we have (*i*) $\sigma_{ik} \in \Sigma_i$ for all $i \in \{1, \ldots, n\}$, (*ii*) $o_k \in \{1, \ldots, n\}$, (*iii*) $a_k \in A$, and (*iv*) for all i, $\sigma_{i,k+1} = \delta_i(\sigma_{ik}, a_k)$ if $o_k = i$, otherwise $\sigma_{i,k+1} = \sigma_{ik}$. Given a trace t, we call $\mathsf{states}(t)$ the *sequence of states* of t, i.e. $\mathsf{states}(t) = (\sigma_{10} \ldots \sigma_{n0}), (\sigma_{11} \ldots \sigma_{n1}), \cdots$. The *choices* of a trace t, denoted with $\mathsf{choices}(t)$, is the sequence of actions in t, i.e. $\mathsf{choices}(t) = (a_0, o_0), (a_1, o_1), \ldots$. Note that, due to nondeterminism, there might be many traces of $\mathcal{C}$ associated with the same sequence of choices. Moreover, we define the *action run* of a trace t, denoted with $\mathsf{actions}(t)$, the projection of $\mathsf{choices}(t)$ only to the components in A. Note that both $\mathsf{choices}(t)$ and $\mathsf{actions}(t)$ are empty if $t = (\sigma_{10} \ldots \sigma_{n0})$. A finite trace t is *successful*, denoted with $\mathsf{successful}(t)$, if (1) $\mathsf{actions}(t) \models \varphi$, and (2) all service states $\sigma_i \in \mathsf{last}(\mathsf{states}(t))$ are such that $\sigma_i \in F_i$.

An *orchestrator* is a partial function $\gamma : (\Sigma_1 \times \cdots \times \Sigma_n)^+ \to (A \times \{1 \ldots n\}) \cup \{\perp\}$ that, if defined given a sequence of states $\bar{\sigma} = (\sigma_{10} \ldots \sigma_{n0}) \ldots (\sigma_{1m} \ldots \sigma_{nm})$, returns the action to perform $a \in A$, and the service (actually the service index) that will perform it; otherwise we write $\gamma(\bar{\sigma}) = \perp$. Next, we define when an orchestrator is a composition that satisfies φ. A trace t is an *execution* of an orchestrator γ with $\mathcal{C}$ if for all $k \geq 0$, we have $(a_k, o_k) = \gamma((\sigma_{10} \ldots \sigma_{n0}) \ldots (\sigma_{1k} \ldots \sigma_{nk}))$ and, if t is finite, say of length m, then $\gamma((\sigma_{10} \ldots \sigma_{n0}) \ldots (\sigma_{1,m-1} \ldots \sigma_{n,m-1})) = \perp$. Note that due to the nondeterminism of the services, we can have many executions for the same orchestrator, despite the orchestrator being a deterministic function. We say that some finite execution t of γ is *successful*, if $\mathsf{successful}(t)$ and $\gamma(\mathsf{states}(t)) = \perp$. Finally, we say that an orchestrator γ *realizes the* LTL$_f$ *goal specification* φ *with* $\mathcal{C}$ if all its executions are finite traces t that are successful. Note that the orchestrator, at every step, chooses the action and the service to which the action is delegated. In doing so, it guarantees that the sequence of actions satisfies the LTL$_f$ goal specification and that at each step the action is delegated to a service that can actually carry out the action, despite the nondeterminism of the services. Moreover, when the orchestrator stops, all services are left in their final states. The composition problem is:

Problem 1 (Composition for LTL$_f$ Goal Specifications). *Given the pair $(\mathcal{C}, \varphi)$, where φ is an* LTL$_f$ *goal specification over the set of propositions A, and $\mathcal{C}$ is a community of n services $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, compute, if it exists, an orchestrator γ that realizes φ.*

Example 1. We present an example of using our framework, inspired by the "garden bots system" example [27]. The goal is to *clean* the garden by picking fallen leaves and removing dirt, *water* the plants, and *pluck* the ripe fruits and flowers. The action *clean* must be performed at least once, followed by *water* and *pluck* in any order. In DECLARE LTL$_f$, the goal can be expressed as $\varphi = clean \wedge \bigcirc(clean\, \mathcal{U}((water \wedge \bigcirc pluck) \vee (pluck \wedge \bigcirc water)))$. We assume there are three available garden bots, each with different capabilities and rewards. In Figure 1 the three services specifications and the automaton $\mathcal{A}_\varphi$ of the LTL$_f$ goal φ are shown. Such an automaton $\mathcal{A}_\varphi$ is a DFA in this simple case, instead of a (proper) NFA.

We are interested in a composition of the bots to satisfy the goal φ. Bot 1 will be used to perform $clean$. Although both bot 2 and 3 can be used for $pluck$, a strong solution cannot choose bot 2 because the action $pluck$ can lead to the failure state b_2; therefore, $pluck$ will be requested to bot 3. Bot 2 will be used for $water$. The order in which $water$ and $pluck$ are executed is irrelevant since both alternatives lead to the accepting state. Both bot 1 and bot 2 might need to be emptied to return to the initial accepting states a_0 and c_0, respectively, and the solution must handle this.

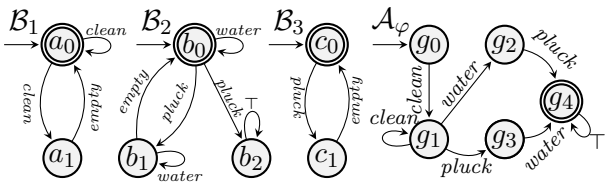


Figure 1: From left to right: the three available bots of the garden bot systems, and the automaton $\mathcal{A}_\varphi$ of the LTL$_f$ goal.

4. Solution Technique

To synthesize the orchestrator, we rely on a game-theoretic technique: i.e.: (*i*) we build a game arena where the *controller* (roughly speaking the orchestrator) and the *environment* (the service community) play as adversaries; (*ii*) we synthesize a strategy for the controller to win the game whatever the environment does; (*iii*) from this strategy we will build the actual orchestrator.

Specifically, we proceed as follows: (1) first, from the LTL$_f$ goal specification we compute the equivalent NFA; (2) in this NFA, we can give the control of the transition to the controller, constructing from the NFA a DFA $\mathcal{A}_{\mathsf{act}}$ over an extended alphabet; (3) compute a *product of such* DFA $\mathcal{A}_{\mathsf{act}}$ *with the services, obtaining a new* DFA $\mathcal{A}_{\varphi,\mathcal{C}}$; (4) the latter DFA can be seen as an arena over which we play is the so-called DFA *game* [28]; (5) if a solution of such DFA game is found, from that solution, we can derive an orchestrator that realizes φ. We now detail each step.

Step 1. The NFA $\mathcal{A}_\varphi = (A, Q, q_0, F, \delta)$ of an LTL$_f$ formula, which can be exponentially larger than the size of the formula, can be computed by exploiting a well-known correspondence between LTL$_f$ formulas and finite-word automata [14]. Note that we can build $\mathcal{A}_\varphi$ in such a way that its alphabet is A and not 2^A since, by the DECLARE assumption, only one action is executed at each time instant.

Step 2. From the NFA of the formula φ, $\mathcal{A}_\varphi$, which is on the alphabet A, we define a *controllable* DFA $\mathcal{A}_{\mathsf{act}} = (A \times Q, Q, q_0, F, \delta_{\mathsf{act}})$ on the alphabet $A \times Q$, with the same states, initial state and final state of $\mathcal{A}_\varphi$ but with δ_{act} defined as follows: $\delta_{\mathsf{act}}(q, (a, q')) = q'$ iff $(q, a, q') \in \delta$. Note that the "angelic" nondeterminism of $\mathcal{A}_\varphi$ is cancelled by moving the choice of the next NFA state and the next system service state in the alphabet $\mathcal{A} \times Q$ of the DFA $\mathcal{A}_{\mathsf{act}}$. Intuitively, with the DFA $\mathcal{A}_{\mathsf{act}}$, we are giving to the controller not only the choice of actions but also the choice of transitions of the original NFA $\mathcal{A}_\varphi$, so that those chosen transitions lead to the satisfaction of the formula. In other words, for every sequence of actions $a_0, \ldots, a_{m-1}$ accepted by the NFA $\mathcal{A}_\varphi$, i.e. satisfying the formula φ, there exists a corresponding alternating sequence $q_0, a_0, \ldots, q_m$ accepted by the DFA $\mathcal{A}_{\mathsf{act}}$, and viceversa.

Step 3. Then, given $\mathcal{A}_{\mathsf{act}}$ and $\mathcal{C}$, we build the *composition* DFA $\mathcal{A}_{\varphi,\mathcal{C}} = (A', Q', q'_0, F', \delta')$ as follows: $A' = \{(a, q, i, \sigma_j) \mid (a, q, i, \sigma_j) \in A \times Q \times \{1, \ldots, n\} \times \left(\bigcup_i \Sigma_i\right)$ and $\sigma_j \in \Sigma_i\}$; $Q' = Q \times \Sigma_1 \times \cdots \Sigma_n$; $q'_0 = (q_0, \sigma_{10} \ldots \sigma_{n0})$; $F' = F \times F_1 \times \cdots \times F_n$; $\delta'((q, \sigma_1 \ldots \sigma_i \ldots \sigma_n), (a, q', i, \sigma'_i)) = (q', \sigma_1 \ldots \sigma'_i \ldots \sigma_n)$ iff $\delta_i(\sigma_i, a) = \sigma'_i$, and $\delta_{\mathsf{act}}(q, (a, q')) = q'$. Intuitively, the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ is a synchronous cartesian product between the NFA $\mathcal{A}_\varphi$ and the service $\mathcal{S}_i$ chosen by the current symbol $(a, q, i, \sigma) \in A'$. It can be shown that there is a relationship between the accepting runs of the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ and the set of successful executions of some orchestrator γ with community $\mathcal{C}$ for the specification φ. Given a word $(a_0, q_1, o_0, \sigma_{o_0}) \ldots (a_{m-1}, q_m, o_{m-1}, \sigma_{o_{m-1}}) \in A'^*$, which induces the run $r = (q_0, \sigma_{10} \ldots \sigma_{n0}), \ldots, (q_m, \sigma_{1m} \ldots \sigma_{nm})$ over $\mathcal{A}_{\varphi,\mathcal{C}}$, we define the history $h = \tau_{\varphi,\mathcal{C}}(w) = (\sigma_{10} \ldots \sigma_{n0}), (a_0, o_0), \ldots, (\sigma_{1m} \ldots \sigma_{nm})$. We consider the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ as a DFA game.

Step 4. DFA *games* are games between two players, here called respectively the *environment* and the *controller*, that are specified by a DFA. We have a set $\mathcal{X}$ of *uncontrollable symbols*, which are under the control of the environment, and a set $\mathcal{Y}$ of *controllable symbols*, which are under the control of the controller. A *round* of the game consists of both the controller and the environment choosing

the symbols they control. A (complete) *play* is a word in $\mathcal{X} \times \mathcal{Y}$ describing how the controller and environment set their propositions at each round till the game stops. A play is *winning* for the controller if such a play leads from the initial to a final state. A *strategy* for the controller is a function $f : \mathcal{X}^* \to \mathcal{Y}$ that, given a history of choices from the environment, decides which symbols $\mathcal{Y}$ to pick next. In this context, a history has the form $w = (X_0, Y_0) \cdots (X_{m-1}, Y_{m-1})$. Let us denote by $w_{\mathcal{X}}|_k$ the sequence projected only on $\mathcal{X}$ and truncated at the k-th element (included), i.e., $w_{\mathcal{X}}|_k = X_0 \cdots X_k$. A *winning strategy* is a strategy $f : \mathcal{X}^* \to \mathcal{Y}$ such that for all sequences $w = (X_0, Y_0) \cdots (X_{m-1}, Y_{m-1})$ with $Y_i = f(w_{\mathcal{X}}\ |_k)$, we have that w leads to a final state of our DFA game. The *realizability* problem consists of checking whether there exists a *winning strategy*. The *synthesis* problem amounts to actually computing such a strategy. A DFA game can be solved by *backward least-fixpoint computation*, by computing the winning region $Win(\mathcal{G})$, that is, the states where the controller has a winning strategy to reach a final state. First, we start with $Win_0(\mathcal{G}) = F$, and $Win_i(\mathcal{G})$ is the set of states for which the controller can force the game to move in a state of $Win_{i-1}(\mathcal{G})$. Let $Win(\mathcal{G}) \subseteq Q$ be the smallest set that satisfies the winning condition. It can be shown that a DFA game $\mathcal{G}$ admits a winning strategy iff $s_0 \in Win(\mathcal{G})$ [28]. The resulting strategy is a transducer $T = (\mathcal{X} \times \mathcal{Y}, Q', q'_0, \delta_T, \theta_T)$, defined as follows: $\mathcal{X} \times \mathcal{Y}$ is the input alphabet, Q' is the set of states, q'_0 is the initial state, $\delta_T : Q' \times \mathcal{X} \to Q'$ is the transition function such that $\delta_T(q, X) = \delta'(q, (X, \theta_T(q))$, and $\theta_T : Q \to \mathcal{Y}$ is the output function defined as $\theta_T(q) = Y$ such that if $q \in Win_{i+1}(\mathcal{G}) \setminus Win_i(\mathcal{G})$ then $\forall X.\delta'(q, (X, Y)) \in Win_i(\mathcal{G})$.

Step 5. Given a strategy in the form of a transducer T, we can obtain an orchestrator that realizes the specification. Let the *extended transition function* δ^*_T of T is $\delta^*_T(q, \epsilon) = q$ and $\delta^*_T(q, wa) = \delta_T(\delta^*_T(q, w), a)$. Then, for every sequence w of length $m \geq 0$ $w = (X_0, Y_0) \dots (X_m, Y_m)$, where for each index k, Y_k and X_k are of the form (a_k, q_{k+1}, o_k) and $\sigma_{o_k,k}$ respectively, we define the orchestrator $\gamma_T((\sigma_{10} \dots \sigma_{n0}), (\sigma_{11} \dots \sigma_{o_1,1} \dots \sigma_{n1}), \dots (\sigma_{1m} \dots \sigma_{o_k,m} \dots \sigma_{nm})) = (a_m, o_m)$, where $(a_m, q_{m+1}, o_m) = \theta_T(\delta^*_T(q_0, w))$, and whenever the trace is successful, the next choice is $\perp$. Hence, we can reduce the problem of service composition for LTL_f goal specifications to solving the DFA game over $\mathcal{A}_{\varphi,\mathcal{C}}$ with uncontrollable symbols $\mathcal{X} = \bigcup_i \Sigma_i$ and controllable symbols $\mathcal{Y} = A \times Q \times \{1, \dots, n\}$.

Proposition 2. *$a_0 \dots a_{m-1} \in \mathcal{L}(\mathcal{A}_\varphi)$ iff $(a_0, q_1) \dots (a_{m-1}, q_m) \in \mathcal{L}(\mathcal{A}_{\mathsf{act}})$, for some $q_1 \dots q_m$.*

Proposition 3. *Let h be a history with $\mathcal{C}$ and φ be a specification. Then, h is successful iff there exist a word $w \in A'^*$ such that $h = \tau_{\varphi,\mathcal{C}}(w)$ and $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$.*

Proposition 3 shows that there is a tight relationship between accepting runs of $\mathcal{A}_{\varphi,\mathcal{C}}$ and successful histories with $\mathcal{C}$ for specification φ.

Theorem 4. *Realizability for $\langle \varphi, \mathcal{C} \rangle$ can be solved by checking whether $q'_0 \in Win(\mathcal{A}_{\varphi,\mathcal{C}})$.*

Proof sketch. Soundness can be proved by induction on the maximum number of steps i for which the controller wins the DFA game from q'_0, building γ in a backward fashion such that it chooses $(a_k, o_k) \in A'$ that allows forcing the win in the DFA game (which exists by assumption). Completeness can be shown by contradiction, assuming that there exists an orchestrator γ that realizes φ with community $\mathcal{C}$, but that $q'_0 \notin Win(\mathcal{A}_{\varphi,\mathcal{C}})$, implying that we can build an arbitrarily long unsuccessful history, by definition of winning region, contradicting that γ realizes φ. □

Theorem 5. *Let the composition problem be $\langle \varphi, \mathcal{C} \rangle$, and let the transducer T be a winning strategy over the game arena $\mathcal{A}_{\varphi,\mathcal{C}}$. Let γ_T be the orchestrator extracted from T, as defined above. Then, γ_T realizes φ with community $\mathcal{C}$.*

Considering the cost of each step above, we get the following upper bound for the worst-case computational cost (we conjecture this cost is the exact complexity characterization):

Theorem 6. *Problem 1 can be solved in at most exponential time in the size of the formula, in at most exponential time in the number of services, and in polynomial time in the size of the services.*

5. Linear Encoding in FOND Planning

In this section, we refine the solution from the previous section, taking advantage of the possibility of using FOND adversarial planning to handle the NFA corresponding to the goal symbolically in linear

time. Specifically, we show how we can compute a PDDL specification that is *linear* in the size of the formula φ, by exploiting a technique for planning for LTL$_f$ goals in [22]. Their construction is based on representing the NFA of the goal by a symbolic representation directly stemming from the corresponding AFA, and then exploring (possibly partially) the NFA *on-the-fly* while planning. In contrast to [22], we do this in a nondeterministic (adversarial) setting, which normally would require using DFAs instead of NFAs, see [15]. This is possible in our case because the LTL$_f$ goal specifies the desired sequence of *target actions*, whose actual choice is *under the full control of the controller* that we are synthesizing.

Encoding in Adversarial FOND Planning. The game-theoretic solution technique presented in the previous section gives us the theoretical foundations for reasoning about the problem and is useful for theoretical analysis regarding the correctness and worst-case computational cost. However, the size of the game arena can be exponential in the size of the formula and exponential in the number of services, meaning that computing the entire arena beforehand may be infeasible. However, this is not required since we can build the arena on-the-fly while searching for a solution. We can do this by leveraging on FOND adversarial planning (aka strong planning in FOND), where the agent controls the actions and the environment controls the fluents [6]. In particular, we focus on planning for temporally extended goals [29, 30], work on declarative and procedural constraints [31, 32, 33], temporal logics with finite-trace semantics, such as LTL$_f$ [32, 33, 14, 22], and in FOND planning [28, 15, 16]. More recently, [34] and [35] proposed, for classical and FOND planning, respectively, a polynomial-time compilation in PDDL for goals in Pure-Past Linear Temporal Logic (PPLTL) [36]. In our setting, the orchestrator controls the actions to satisfy the LTL$_f$ goal, while the evolution of the selected services is the uncontrollable part of the process. A forward search-based approach able to handle environment nondeterminism is, therefore, particularly interesting for our case since it could possibly avoid the exponential construction of the entire arena since the procedure stops as soon as a winning strategy is found. While the services are already given in input, the NFA $\mathcal{A}_\varphi$ must be computed from the input goal specification φ. To avoid constructing the entire NFA in advance, we will rely on an on-the-fly construction based on Torres and Baier's work [22]; more details later in this section. Another crucial feature of (FOND) planning is that the domain specification is assumed to be *factorized*, i.e., a state in the arena is a propositional assignment of a set of fluents $\mathcal{F}$. This feature fits well in our setting since a state of the game arena (i.e. of the composition DFA $\mathcal{A}_{\varphi,\mathcal{C}}$) is made of several state components: one state component for each service, each keeping track of the current state for each service, and one component from the NFA state $\mathcal{A}_\varphi$, which keeps track of the partial satisfaction of the goal formula φ. Regarding the latter, we consider the NFA states Q_N as assignments of the states of the AFA Q_A viewed as atomic propositions (i.e. $Q_N \subseteq 2^{Q_A}$) [14, 22]. As for the services $\mathcal{S}_i$, they also could be represented in *compact* (i.e. logarithmic) form with fluents $\mathcal{F}_i$, i.e. $\Sigma_i = 2^{\mathcal{F}_i}$. Indeed, one could always build a binary encoding of the state space Σ_i using $\log |\Sigma_i|$ bits.

Adapting Torres and Baier's Construction Most of the techniques in FOND planning for temporal goals specify a PDDL encoding that takes a domain and a problem in PDDL with a temporal goal as input and generates new PDDL domain and problem files with a simple reachability goal. Such files can then be given in input to a (non-temporal) FOND planning solver, and the correctness of the encoding guarantees that, from a solution for the new version of the problem, we can compute a solution for the original planning problem. In our case, we adopt the encoding proposed in [22] for solving temporally-extended planning in deterministic domains. The interesting feature of their technique is that it makes it possible to encode in PDDL the construction of the NFA on-the-fly, by exploiting the relationship between AFA A_A of the LTL$_f$ formula φ and its equivalent NFA A_N. Not only is this a worst-case optimal construction in the size of the formula, but it could possibly avoid the entire construction of the NFA. The other alternatives are either superseded, not easily applicable, or worst-case non-optimal: [33]'s translation is worst-case exponential but builds the entire NFA explicitly and only works for deterministic domains; and the encoding proposed in [16] requires the construction of the entire DFA, which is not optimal since its size is doubly exponential in the size of the formula. the encoding in [35] is designed for PPLTL goals, and despite PPLTL and LTL$_f$ are equally expressive, translating one into the other (and vice versa) is generally prohibitive since the best-known algorithms

require 3EXPTIME [36]. Although Torres and Baier's encoding was originally used for deterministic planning domains, we observe that their construction works even in the case of nondeterministic domains, with the restriction that the propositions $\mathcal{P}$ in the LTL$_f$ goal specification are such that *all are controllable by the agents* and, therefore, *not controllable by the environment*. In fact, this is the case in our framework: we can rely on the NFA-based construction in the definition of the game arena since the nondeterminism from the NFA and the nondeterminism from the services do not directly interfere with each other. Note that if some of the propositions in the goal formula were not under the control of the agent (as is typically the case in temporally extended goals, which talk about sequences of fluent evaluations in the domain), we would end up mixing the *angelic* nondeterminism of the NFA with the *devilish* nondeterminism of the environment, and to avoid it we need determinize the NFA getting a DFA where the angelic nondeterminism has been removed. As a consequence, the complexity of planning in this case becomes 2EXPTIME-complete [15].

There are two minor issues to cope with when using Torres and Baier's encoding in our setting: (*i*) first, the formulas can only be evaluated over state fluents and not over actions; (*ii*) second, the compilation does not handle nondeterminism. In the next part, we describe how to solve these issues. First, we define the *service community domain* $\mathcal{D}_\mathcal{C} = \langle \mathcal{F}', A', pre', eff' \rangle$ and *problem* $\Gamma_\mathcal{C} = \langle \mathcal{D}_C, s'_0, G' \rangle$, where: *(i)* $\mathcal{F}' = \{\mathbf{start}\} \cup A \cup \Sigma_1 \cup \cdots \Sigma_n$; *(ii)* $A' = A \times \{1 \dots n\}$; *(iii)* $pre'(\langle a, i \rangle) = true$ (since δ_i are total functions); *(iv)* $eff'(\langle a, i \rangle) = \{eff'_{a,i}(\sigma_{i1}, \sigma_{i2}) \mid \sigma_{i2} \in \delta_i(\sigma_{i1}, a)\}$ where $eff'_{a,i}(\sigma_{i1}, \sigma_{i2}) = \{\{\sigma_{i1}\} \triangleright \{\neg\sigma_{i1}, \sigma_{i2}, a\} \cup \bar{A}(a)\}$; *(v)* $s'_0 = \{\mathbf{start}\} \cup \{\sigma_{i0} \mid \sigma_{i0}$ is the initial state of $\mathcal{S}_i\}$; *(vi)* $G' = \{(\sigma_1, \dots, \sigma_n) \mid \sigma_i \in F_i$ for all $i = 1, \dots n\}$. Intuitively, $\mathcal{D}_\mathcal{C}$ simulates executions of $\mathcal{C}$. The state induced by the fluents $\mathcal{F}'$ is represented as tuples of the form $(\sigma_1, \dots, \sigma_n, a)$, where $\sigma_i \in \Sigma_i$ and $a \in A$. The action space A' is a set of pairs of the form (a, i), where a is the chosen action and i is the index of the chosen service that should execute it. The preconditions are not needed (though conditional effects are) since we assumed the services' transition functions are complete. The effect of action (a, i) are all the possible state pairs for which there is a transition via a, but only the effects from the same current state σ_{i1} can be triggered (see the condition), and all the successor states $\sigma_{i2} \in \delta_i(\sigma_{i1}, a)$ are considered (see the set comprehension for terms $eff'_{a,i}(\sigma_{i1}, \sigma_{i2})$). Crucially, the action (a, i) also has the effect of adding the fluent a in the next state and removing all other action fluents; such negative effects are denoted with $\bar{A}(a) = \{\neg a' \mid a' \in A, a' \neq a\}$, hence forcing the DECLARE assumption at the semantic level. The initial state contains the auxiliary fluent **start**, which will be used to shift by one step the evaluation of the formula; intuitively, this is because the state-based evaluation starts from the initial state, where no action is taken yet. The auxiliary fluent **start** and the presence of the last action taken in the state allow us to solve the first issue. Finally, the set of goal states G' corresponds to the set of services' configurations where all the states are final. The next step is to apply the translation rules specified in [22]. Given a problem instance $\langle \varphi, \mathcal{C} \rangle$, and given the domain $\mathcal{D}_\mathcal{C}$ and problem $\Gamma_\mathcal{C}$ as defined above, we get a new domain $\mathcal{D}_{\varphi,\mathcal{C}}$ and problem $\Gamma_{\varphi,\mathcal{C}}$ by applying the translation rules with the formula $\varphi' = \mathbf{start} \wedge \bigcirc\varphi$. Intuitively, the evaluation of φ' will read the initial state and, from there on, will evaluate the chosen actions, which, by construction, are added to the subsequent states. To support nondeterminism, and so to fix the second challenge, the only change we apply is that, in "World Mode" (cfr. [22]), we consider all possible nondeterministic effects coming from $\mathcal{D}_\mathcal{C}$, i.e. $eff(a') = \{E \cup \{\mathbf{copy}, \neg\mathbf{world}\} \mid E \in eff(a')\}$.

Theorem 7. *Let $\langle \varphi, \mathcal{C} \rangle$ be an instance of LTL$_f$-goal oriented service composition. φ is realizable with $\mathcal{C}$ iff there exists a strong solution for the FOND adversarial problem $\Gamma_{\varphi,\mathcal{C}}$.*

Proof sketch. By construction of $\Gamma_{\varphi,\mathcal{C}}$, and by correctness of Torres and Baier's construction, there is a one-to-one correspondence between traces of $\Gamma_{\varphi,\mathcal{C}}$ and the game arena $\mathcal{A}_{\varphi,\mathcal{C}}$ (modulo deletion of the first evaluation of **start**). In other words, the game arena induced by $\mathcal{D}_{\varphi,\mathcal{C}}$ and $\Gamma_{\varphi,\mathcal{C}}$ is essentially the same of the one induced by $\mathcal{A}_{\varphi,\mathcal{C}}$. Therefore, there exist a strong solution for $\Gamma_{\varphi,\mathcal{C}}$ iff there exist a strong solution for $\mathcal{A}_{\varphi,\mathcal{C}}$. The claim follows by Theorem 4. □

Theorem 8. *LTL$_f$ goal-oriented service composition can be solved in at most exponential time in the size of the formula, in at most exponential time in the number of services, and in polynomial time in the size of services (exponential if the service representation is in logarithmic form).*

We observe that in the construction of $\mathcal{D}_{\varphi,\mathcal{C}}$ we introduce action fluents, hence increasing the size of the state space. This could be avoided by modifying Torres and Baier's encoding with PDDL3 action-trajectory constraints [37], and hence by evaluating the temporal goal over the action-trajectory only. A detailed analysis of this option is left as future work.

6. Implementation and Applications

We implemented a software prototype to solve the composition problem, and we tested it on industrial case studies taken from the literature. The code can be found at https://bit.ly/3XjJbEF.

The tool. Our tool takes in input a list of services (in explicit representation) and a LTL$_f$ goal specification and computes a PDDL specification (i.e. domain and problem files) of the corresponding FOND planning task, using the technique formalized in the previous section.

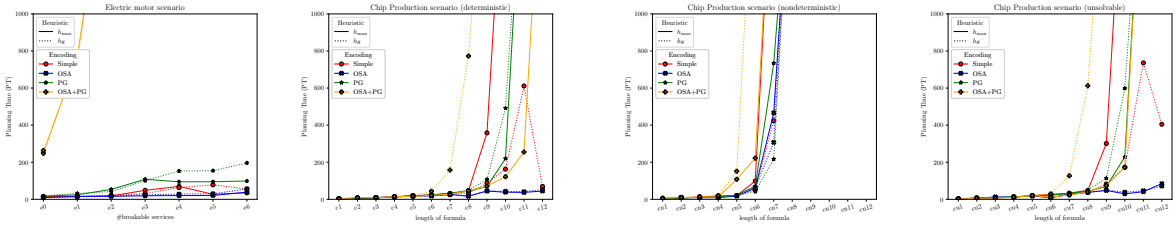
First, we construct $\mathcal{D}_{\mathcal{C}}$ and $\Gamma_{\mathcal{C}}$ in PDDL form. The PDDL domain file represents the nondeterministic behaviour of the services. One of the challenges we encountered was that some planning systems do not support the `when` expression with complex effect types, such as `oneof`; this prevented us from specifying the transitions as a list of `when` expressions, one for each possible starting state, each followed by a `oneof` expression that includes all the possible successors. To workaround this issue, given an action $\langle a, i\rangle$ of $\mathcal{D}_{\mathcal{C}}$, we defined a PDDL operator $\langle a, i, \sigma_{ij}\rangle$, one for each possible starting state $\sigma_{ij} \in \Sigma_i$ of service i; In this way, we can use the `oneof` effect without nesting it into a `when` expression.

Then, to include the on-the-fly evaluation of the LTL$_f$ goal specification φ, we rely on the [22]'s translator (in the following, denoted with TB), implemented in SWI-Prolog, to include the on-the-fly evaluation of the LTL$_f$ goal in the planning domain. The encoding of the goal formula in PDDL follows the syntax supported by the TB translator. The TB translator supports four modes: Simple, OSA, PG, and OSA+PG, where Simple is the "naive" translation (cfr. [22], Section 4) and OSA, OSA+PG are two optimizations called "Order for Synchronization Action" and "Positive Goals" (cfr. *ibid.*, Section 4.3). OSA+PG is the combination of OSA and PG.

The final `(:goal)` section includes, in conjunction: (*i*) the goal specified by the TB encoding, and (*ii*) the formula that specifies the accepting configuration for all services. Regardin (*ii*), the PDDL formula for the accepting service configuration has the form `(and (or (curstate_s1` σ_{11}`)(curstate_s1` σ_{12}`)...)...(or (curstate_sn` σ_{n1}`) (curstate_sn` σ_{n2}`)...))`, for all services $\mathcal{S}_i$ with $i = 1 \dots n$, and $\sigma_{ij} \in F_i$. Intuitively, the formula captures the condition that for each service, it holds that in the current planning state each service is in either one of its final states. Note that we could have encoded the final acceptance condition by means of the formula $\Diamond(\phi \wedge \bullet true)$, where ϕ is a propositional formula, which is the formula that is accepting whenever, in the current state of the trace, ϕ is true. However, this would have burdened the TB translator with a larger LTL$_f$ formula, ending up in enlarging the overhead of the encoding (i.e. more sync actions, more NFA states, etc.).

Case Studies. To test our tool, we considered case studies inspired by the literature on service composition applied to the Smart Manufacturing and Digital Twins industry.

Electric Motor (EM). We consider a simplified version of the electric motor assembly, proposed in the context of Digital Twins composition for Smart Manufacturing [38]. The main components of an electric motor are the stator, the rotor, and, in the case of alternate current motors with direct current power (e.g., in the case of electric cars), the inverter. These three components are built or retrieved in any order, but the final assembly step must have all the previous components available. Moreover, after the assembly step, it is required that at least one test between an electric test and a full static test must be performed. This goal is captured by the following LTL$_f$ constraints: $\Diamond(assembleMotor) \wedge (\neg assembleMotor\, \mathcal{U}\, buildStator) \wedge (\neg assembleMotor\, \mathcal{U}\, buildRotor) \wedge (\neg assembleMotor\, \mathcal{U}\, buildInverter) \wedge \Diamond(staticTest \vee electricTest) \wedge (\neg electricTest\, \mathcal{U}\, assembleMotor) \wedge (\neg staticTest\, \mathcal{U}\, assembleMotor)$. The $\mathcal{U}$-clauses prevent the assembly step before all the components are available, while the reachability goal is specified by $\Diamond(assembleMotor)$ and $\Diamond(staticTest \vee electricTest)$. We consider two types of services:



(a) PT metric for the EM scenario. (b) PT metric for the CP scenario (det). (c) PT metric for the CP scenario (nondet). (d) PT metric for the CP scenario (unsolvable).

infallible and *breakable* (Figure 2). The former has only one accepting state and supports one operation op; the latter, when executing the operation, can nondeterministically go into a "broken" state, from which a $repair$ action is required to make it available again. In our experiments, we will have exactly one service for each process action, and scale on the number of breakable services.

Chip Production (CP). Here we consider a smart factory scenario in which the goal is to produce chips [8]. In our simplified setting, the goal specification consists of a sequence of operations to be performed: cleaning the silicon wafers, thin film deposition, resist coating, etc. We consider three variants of this scenario, one for each service type. In particular, in the first variant, all services are of type infallible; in the second variant, they are of type breakable; and in the third variant, the services are all of type *irreparable*, i.e. they are like the breakable services except that they cannot be repaired. The LTL$_f$ goal specification is a sequential goal with the following actions: $cleaning$, $filmDeposition$, $resistCoating$, $exposure$, $development$, $etching$, $impuritiesImplantation$, $activation$, $resistStripping$, $assembly$, $testing$, and $packaging$. Hence, the formula has the following form: $\Diamond(cleaning \wedge \neg filmDeposition \wedge \cdots \wedge \neg filmDeposition \wedge \dots)$. Note that at each step we negate the presence of all the other. Moreover, for all variants, we will have exactly one service for each process action. We use the number of actions as a scaling parameter.

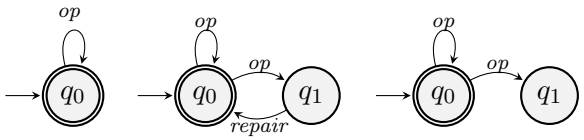


Figure 2: The infallible, breakable and irreparable services templates, respectively.

Evaluation. We evaluated the MyND Planner [39], combined with the h_{ff} and $h_{\max}$ heuristics, over the PDDL files produced by our tool and the 4 available encoding of the TB translator. The metrics we considered are: pre-processing time, i.e., translation and SAS computation (TT), planning time (PT), the number of nodes expanded during the search (EN), and the policy size (PS). As benchmarks, we considered: e_i, with $i = 0, \dots, 6$, are instances of the electric motor scenario with i builder services breakable and $6 - i$ infallible; c_i are the instances on the chip production scenario, with $i = 1, \dots, 12$ being the length of the sequence of operations, with all services infallible; cn_i as c_i but with all services breakable; and cu_i as c_i but with all services irreparable. We set a timeout of 1000 seconds ($\approx$ 15 minutes). Due to lack of space we discuss only the PT metric. Other results can be found in the appendix.

Platform. The experiments have been run on an Ubuntu 22.04 machine, endowed with 12th Gen Intel(R) Core(TM) i7-1260P, with 16 CPU threads (12 cores) and 64GB of RAM. The JVM version is 14 for compatibility with MyND. The maximum RAM for the JVM was 16GB.

Results. The results of our evaluation on all benchmarks, both using $h_{\max}$ and h_{ff}, are shown in the plots for the PT metric for each scenario: Figure 3a, 3b, 3c, and 3d. For the EM scenario (Figure 3a), we observe that the PT of the OSA encoding is generally lower than the others, and in particular $h_{\max}$ is slightly better in PT than h_{ff}. The Simple encoding has comparable but slightly higher PT. The PG encoding has the PT higher than the PT in the Simple and OSA case, for all instances, sometimes higher by a factor of 4-5. The OSA+PG encoding was considerably worse than the other since the evaluation on many instances reached the timeout. In the deterministic CP scenario (Figure 3b), we have that the OSA encoding, with respect to the PT metric, is better than the others, with no strict dominance between $h_{\max}$ and h_{ff}. The other encodings, from better to worse, were OSA+PG, PG and Simple

for the h_{ff} heuristic, and Simple, PG and OSA+PG for the $h_{\max}$ heuristic. In the nondeterministic CP scenario (Figure 3c), the performances were quite worse than the deterministic case for all encodings and heuristics; the executions from cn8 on timed out. We noticed a certain advantage of using h_{ff} with the PG encoding, and $h_{\max}$ with the OSA encoding. In the unsolvable CP scenario (Figure 3d), the OSA was considerably better than the other encodings, with comparable performances between $h_{\max}$ and h_{ff}.

7. Discussion and Conclusion

In this paper, we have studied an advanced form of task-oriented compositions of nondeterministic services. Our goal is to synthesize an orchestrator that, using the available services, produces a trace that satisfies an LTL$_f$ specification. To underline the importance of the ever-increasing use of service composition in smart manufacturing, we evaluate two valid case studies taken from the literature: one concerning the production of an electric motor and the other concerning the production of chips. The tool prototype we implemented shows the feasibility of our approach. It would be interesting to test other encodings for temporal goals, such as [16] and [34, 35]. This work is highly motivated by a renovated interest in service composition techniques with impactful applications in smart manufacturing [40, 41, 42]. In particular, the use of service composition has been advocated in an industrial scenario [10], where the composition of a target service (manufacturing goal) is managed by means of a community of Digital Twins (manufacturing actors) modelled as stochastic services.

References

[1] J. McGovern, S. Tyagi, M. Stevens, S. Mathew, Java web services architecture, 2003.

[2] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, Automatic composition of e-services that export their behavior, in: ICSOC, 2003.

[3] D. Berardi, D. Calvanese, G. De Giacomo, M. Mecella, Composition of services with nondeterministic observable behavior, in: ICSOC, 2005.

[4] G. De Giacomo, M. Mecella, F. Patrizi, Automated service composition based on behaviors: The Roman model, in: Web services foundations, Springer, 2014.

[5] A. Cimatti, M. Pistore, M. Roveri, P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, Artif. Intell. (2003).

[6] H. Geffner, B. Bonet, A Concise Introduction to Models and Methods for Automated Planning, M&C Publishers, 2013.

[7] G. De Giacomo, F. Patrizi, S. Sardiña, Automatic behavior composition synthesis, Artif. Intell. (2013).

[8] F. Monti, L. Silo, F. Leotta, M. Mecella, On the suitability of AI for service-based adaptive supply chains in smart manufacturing, in: ICWS, 2023.

[9] F. Monti, L. Silo, F. Leotta, M. Mecella, Services in smart manufacturing: Comparing automated reasoning techniques for composition and orchestration, in: SOC, 2023.

[10] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, L. Silo, Digital twin composition in smart manufacturing via markov decision processes, Comput. Ind. (2023).

[11] M. Pesic, H. Schonenberg, W. M. Van der Aalst, Declare: Full support for loosely-structured processes, in: EDOC, 2007.

[12] C. Di Ciccio, M. Montali, Declarative Process Specifications: Reasoning, Discovery, Monitoring, Springer, 2022.

[13] M. Dumas, F. Fournier, L. Limonad, A. Marrella, M. Montali, J. Rehse, R. Accorsi, D. Calvanese, G. De Giacomo, D. Fahland, A. Gal, M. L. Rosa, H. Völzer, I. Weber, Ai-augmented business process management systems: A research manifesto, ACM Trans. Manag. Inf. Syst. (2023).

[14] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, 2013.

[15] G. De Giacomo, S. Rubin, Automata-theoretic foundations of FOND planning for ltl_f and ldl_f goals, in: IJCAI, 2018.
[16] A. Camacho, M. Bienvenu, S. A. McIlraith, Towards a unified view of AI planning and reactive synthesis, in: ICAPS, 2019.
[17] V. Fionda, G. Greco, LTL on finite and process traces: Complexity results and a practical reasoner, J. Artif. Intell. Res. 63 (2018) 557–623.
[18] E. Sirin, B. Parsia, D. Wu, J. A. Hendler, D. S. Nau, HTN planning for web service composition using SHOP2, J. Web Semant. (2004).
[19] J. Alves, J. Marchi, R. Fileto, M. A. R. Dantas, Resilient composition of web services through nondeterministic planning, in: ISCC, 2016.
[20] M. Pistore, A. Marconi, P. Bertoli, P. Traverso, Automated composition of web services by planning at the knowledge level, in: IJCAI, 2005.
[21] G. De Giacomo, M. Favorito, L. Silo, Composition of stochastic services for ltl_f goal specifications, in: FoIKS, 2024.
[22] J. Torres, J. A. Baier, Polynomial-time reformulations of LTL temporally extended goals into final-state goals, in: IJCAI, 2015.
[23] A. Marrella, M. Mecella, S. Sardiña, Supporting adaptiveness of cyber-physical processes through action-based formalisms, AI Commun. (2018).
[24] A. K. Chandra, D. Kozen, L. J. Stockmeyer, Alternation, J. ACM (1981).
[25] M. Y. Vardi, An automata-theoretic approach to linear temporal logic, 1995.
[26] G. Röger, F. Pommerening, M. Helmert, Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting, in: ECAI, 2014.
[27] N. Yadav, S. Sardina, Decision theoretic behavior composition, in: AAMAS, 2011.
[28] G. De Giacomo, M. Y. Vardi, Synthesis for LTL and LDL on finite traces, in: IJCAI, 2015.
[29] F. Bacchus, F. Kabanza, Planning for temporally extended goals, Ann. Math. Artif. Int. (1998).
[30] F. Bacchus, F. Kabanza, Using temporal logics to express search control knowledge for planning, Artif. Intell. (2000).
[31] J. A. Baier, C. Fritz, M. Bienvenu, S. A. McIlraith, Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners, in: AAAI, 2008.
[32] J. A. Baier, S. A. McIlraith, Planning with first-order temporally extended goals using heuristic search, in: AAAI, 2006.
[33] J. A. Baier, S. A. McIlraith, Planning with temporally extended goals using heuristic search, in: ICAPS, 2006.
[34] L. Bonassi, G. De Giacomo, M. Favorito, F. Fuggitti, A. E. Gerevini, E. Scala, Planning for temporally extended goals in pure-past linear temporal logic, in: ICAPS, 2023.
[35] L. Bonassi, G. De Giacomo, M. Favorito, F. Fuggitti, A. E. Gerevini, E. Scala, FOND planning for pure-past linear temporal logic goals, in: ECAI, 2023.
[36] G. De Giacomo, A. Di Stasio, F. Fuggitti, S. Rubin, Pure-past linear temporal and dynamic logic on finite traces, in: IJCAI, 2020.
[37] L. Bonassi, A. E. Gerevini, E. Scala, Planning with qualitative action-trajectory constraints in PDDL, in: IJCAI, 2022.
[38] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, F. Monti, L. Silo, AIDA: A tool for resiliency in smart manufacturing, in: CAiSE Forum, 2023.
[39] R. Mattmüller, Informed progression search for fully observable nondeterministic planning, Ph.D. thesis, 2013.
[40] T. Catarci, D. Firmani, F. Leotta, F. Mandreoli, M. Mecella, F. Sapio, A conceptual architecture and model for smart manufacturing relying on service-based digital twins, in: ICWS, 2019.
[41] G. De Giacomo, P. Felli, B. Logan, F. Patrizi, S. Sardiña, Situation calculus for controller synthesis in manufacturing systems with first-order state representation, Artif. Intell. (2022).
[42] G. De Giacomo, M. Y. Vardi, P. Felli, N. Alechina, B. Logan, Synthesis of orchestrations of transducers for manufacturing, in: AAAI, 2018.